

Monitoring ROS2: from Requirements to Autonomous Robots

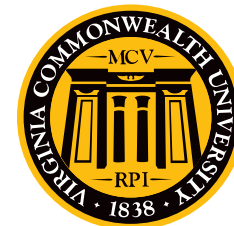
**Ivan
Perez**
KBR Inc

Anastasia
Mavridou
KBR Inc

Tom
Pressburger
NASA

Alexander
Will
VCU

Patrick J.
Martin
VCU



VCU

Context / goals

- Capture mission requirements in high-level language.
- Monitor requirements for robots.

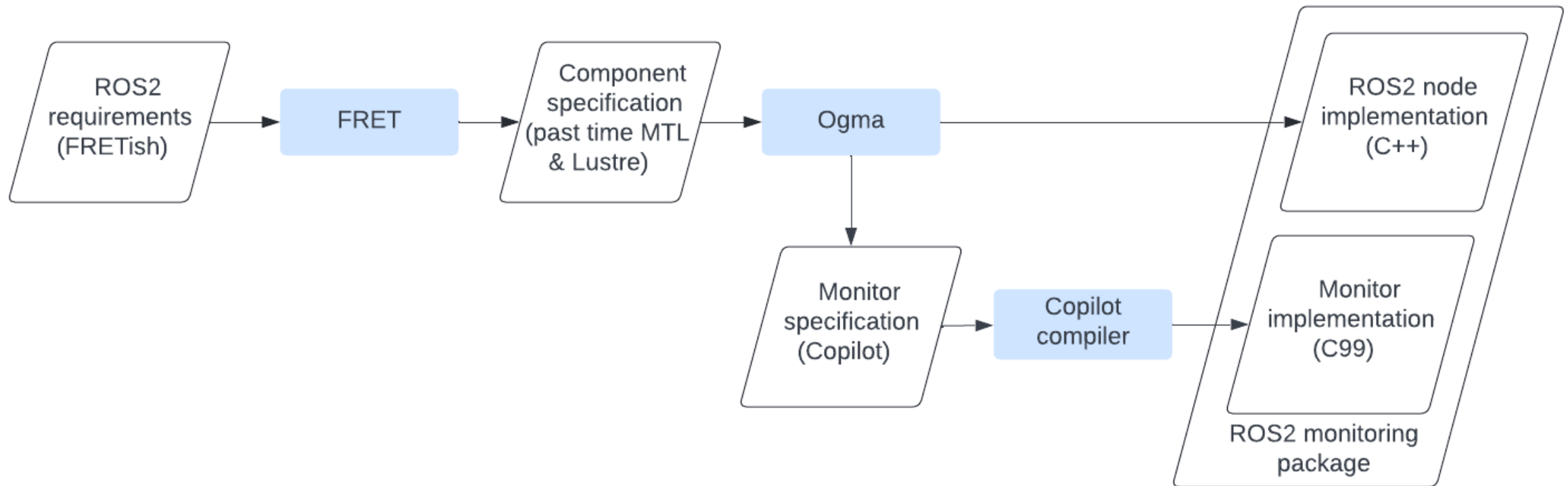
At a glance

- Transform requirements into runtime monitors for ROS2:
 - Requirements elicitation.*
 - Transform requirements into Temporal Logic formulas.*
 - Transform Temporal Logic formulas into runtime monitors.*
 - Generate hard real-time code for monitors.*
 - Generate ROS2 application directly from requirements.+

*Steps done in prior work.

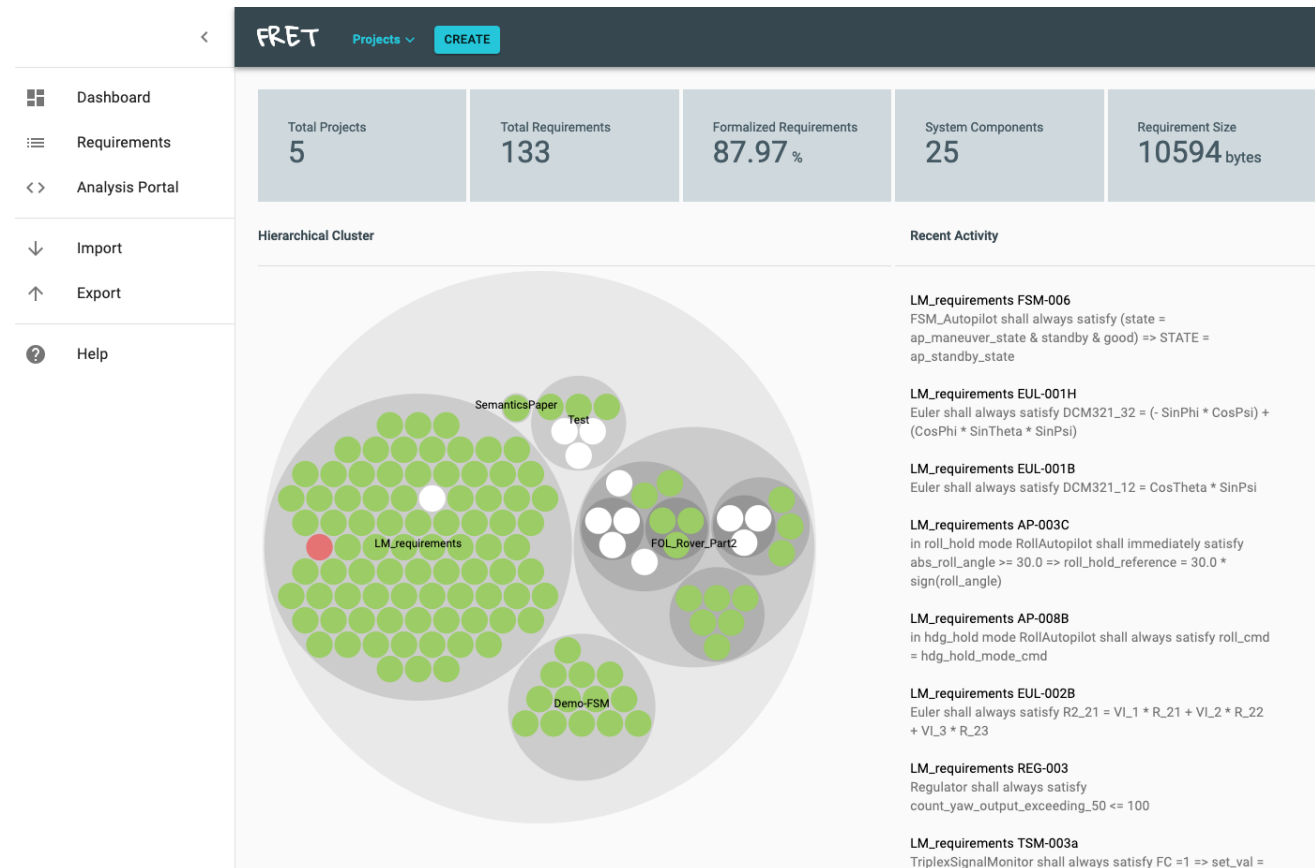
+Steps extended from prior work.

At a glance



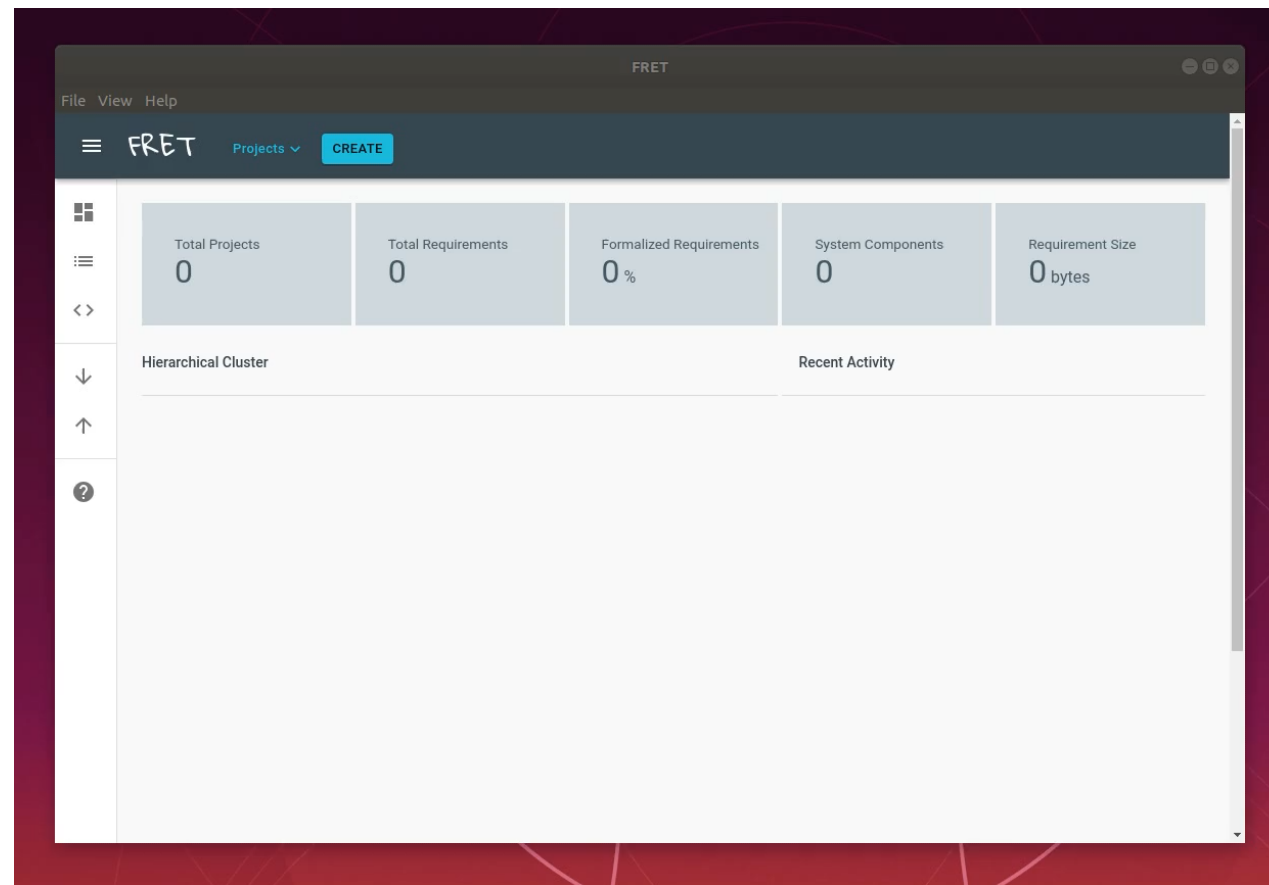
Requirements elicitation: FRET

- FRET is a tool for requirements elicitation developed at NASA Ames.



Requirements elicitation: FRET

- FRET is a tool for requirements elicitation developed at NASA Ames.



Requirements elicitation: FRET

- FRET requirements are expressed in *structured natural language*:

scope **condition** **component*** shall* **timing** **response***

- Example:

NL: “*While flying, if the airspeed is below 100 m/s, the autopilot shall increase the airspeed to at least 100 m/s within 10 seconds.*”

FRETish: **in flight mode** **if airspeed < 100** **the aircraft** shall **within 10 seconds** **satisfy (airspeed >= 100)**

FRET: Step 1: FRETish to temporal logic

NL: “While flying, if the airspeed is below 100 m/s, the autopilot shall increase the airspeed to at least 100 m/s within 10 seconds.”

FRETish: **in flight mode** **if** **airspeed < 100** **the aircraft** **shall** **within 10 seconds** **satisfy** **(airspeed ≥ 100)**

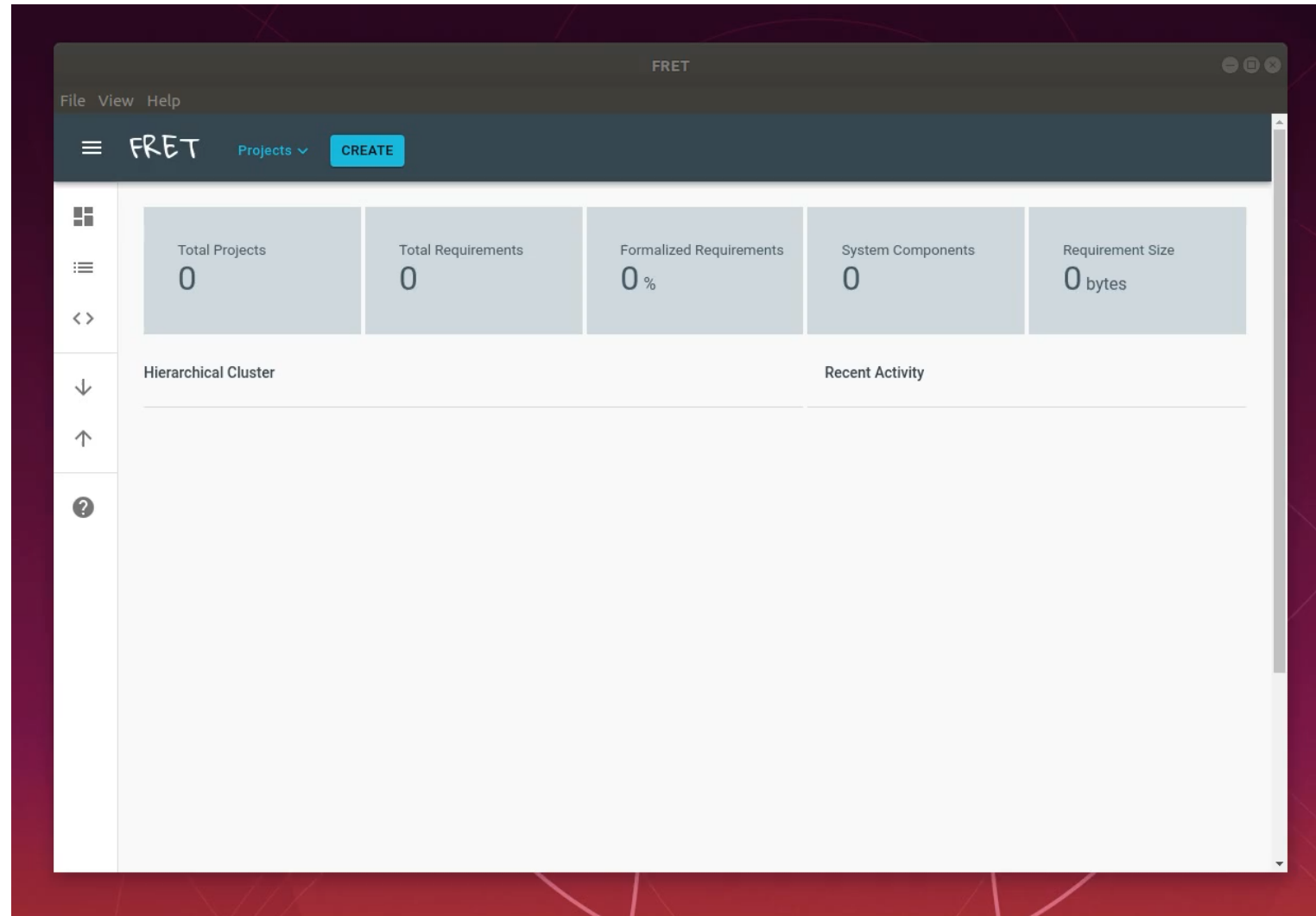
pmLTL: $H (Lin_flight \rightarrow (Y ((O_{[=10]} (((airspeed < 100) \& ((Y (!(airspeed < 100)))) \mid Fin_flight))) \& (!(airspeed \geq 100)))) \rightarrow (O_{[<10]} (Fin_flight \mid (airspeed \geq 100)))) S (((O_{[=10]} (((airspeed < 100) \& ((Y (!(airspeed < 100)))) \mid Fin_flight))) \& (!(airspeed \geq 100)))) \rightarrow (O_{[<10]} (Fin_flight \mid (airspeed \geq 100)))) \& Fin_flight)))) \& (!(Lin_flight) S (!(Lin_flight) \& Fin_flight)) \rightarrow (((O_{[=10]} (((airspeed < 100) \& ((Y (!(airspeed < 100)))) \mid Fin_flight))) \& (!(airspeed \geq 100)))) \rightarrow (O_{[<10]} (Fin_flight \mid (airspeed \geq 100)))) S (((O_{[=10]} (((airspeed < 100) \& ((Y (!(airspeed < 100)))) \mid Fin_flight))) \& (!(airspeed \geq 100)))) \rightarrow (O_{[<10]} (Fin_flight \mid (airspeed \geq 100)))) \& Fin_flight)),$
 where *Fin_flight* (First timepoint in flight mode) is *flight* & (FTP | Y !flight), *Lin_flight* (Last timepoint in flight mode) is !flight & Y flight, FTP (First Time Point) is ! Y true.

FRET: Steps 2 & 3: Variables data and Component Specification

Update Variable

FRET Project	FRET Component
TACASpaper	aircraft
FRET Variable	Variable Type*
airspeed	Internal
Data Type*	
double	
Variable Assignment in Copilot*	
altitude >	
Parse Errors: mismatched input '<EOF>' expecting {'(', '[', 'constant', 'drop', 'extern', OPOne, 'mux', IDENT}	
☐ Lustre ☒ CoPilot	
CANCELUPDATE	

FRET: Process



Generation of C code for monitors: Copilot

- Domain-specific language (DSL) to describe runtime monitors.
- Compiler generates hard real-time C code.
- Streams carry input data of different types.
- Monitors are defined by Boolean-carrying streams.
- Operators on streams allow transforming arbitrary streams into Boolean streams.

Copilot: Example

one :: Stream Double

one = 1.0

two :: Stream Double

two = one + one

	t=0	t=1	t=2	...
one	1.0	1.0	1.0	

Copilot: Example

`one :: Stream Double`

`one = 1.0`

`twoThenOne :: Stream Double`

`twoThenOne = [2.0] ++ one`

	t=0	t=1	t=2	...
one	1.0	1.0	1.0	
twoThenOne	2.0	1.0	1.0	

Copilot: Example

counter :: Stream Double

counter = [0.0] ++ (counter + 1.0)

sinusoidal :: Stream Double

sinusoidal = sin counter

	t=0	t=1	t=2	...
counter	0.0	1.0	2.0	
sinusoidal	0.0	0.8415	0.9093	

Copilot: Example

`airspeed :: Stream Float`

`airspeed = extern "airspeed" Nothing`

`airspeedFast :: Stream Bool`

`airspeedFast = airspeed > 10`

	t=0	t=1	t=2	...
airspeed	11	9	12	
airspeedFast	T	F	T	

Copilot: Features

- Temporal logics (past-time, metric, etc.).
- Voting.
- Regular expressions.
- Basic statistics.

Copilot: Compilation

- Generates hard real-time C99 code (no mallocs, no loops, no recursion).
- All monitors are checked by calling a top-level function (e.g., `step()`).
- Monitors that go off call handlers *written in C by the user*.

Transformation of requirements into monitors: OGMA

- Ogma is a tool to facilitate the integration of safe runtime monitors into other systems.
- Ogma extends **Copilot**, a high-level runtime verification framework that generates hard real-time C99 code.
- Main features:
 - Translating FRET requirements into Copilot monitors.
 - **Generating ROS2 runtime monitoring applications.**

OGMA: Interface

```
$ ogma fret-component-spec --fret-file-name aircraftReqSpec.json
```

```
import Copilot.Compile.C99
```

```
...
```

```
flight :: Stream Bool
```

```
flight = extern "flight" Nothing
```

```
propAvoidStall :: Stream Bool
```

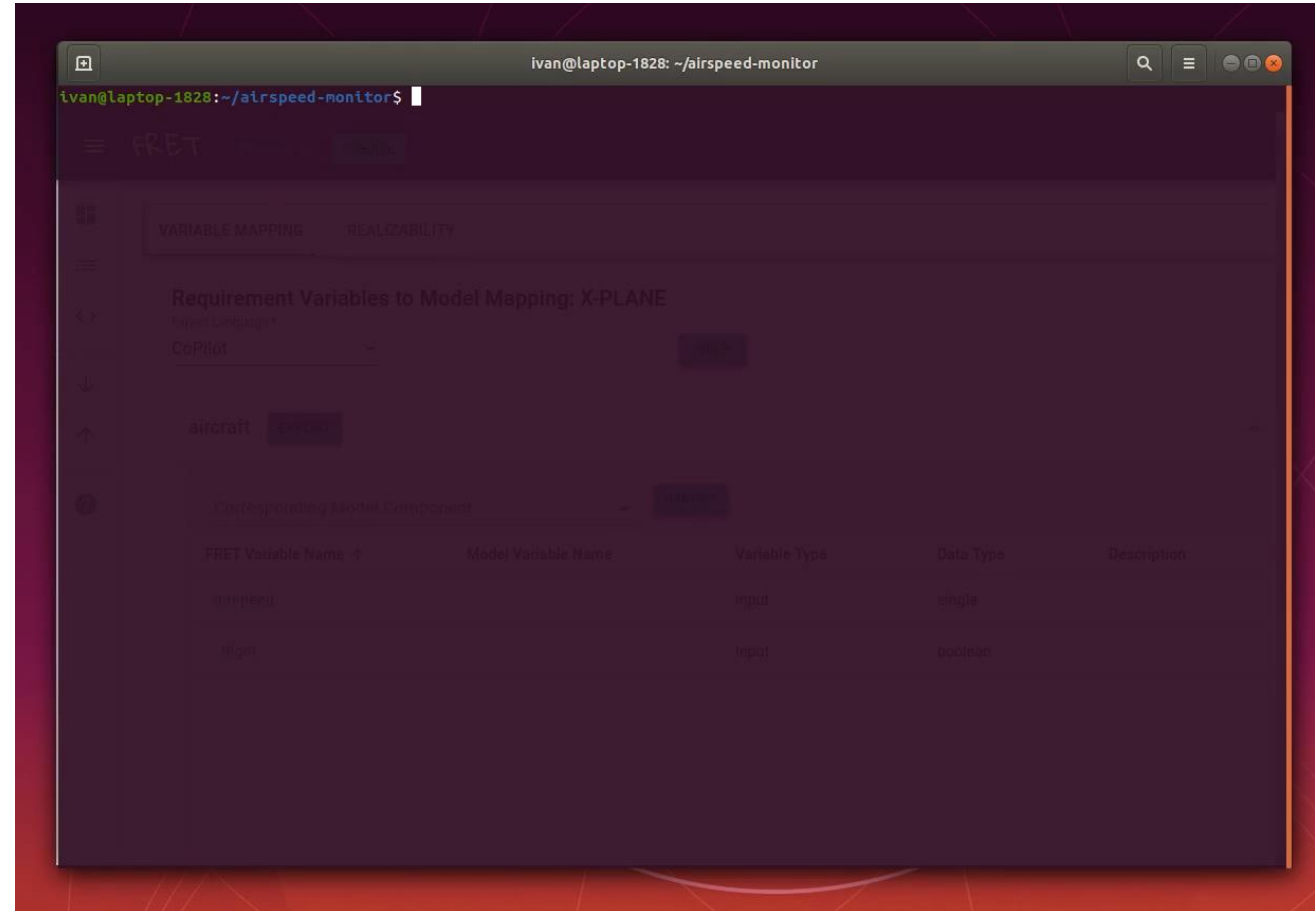
```
propAvoidStall = ((PTLTL.alwaysBeen (((not (flight)) && ... )))
```

```
...
```

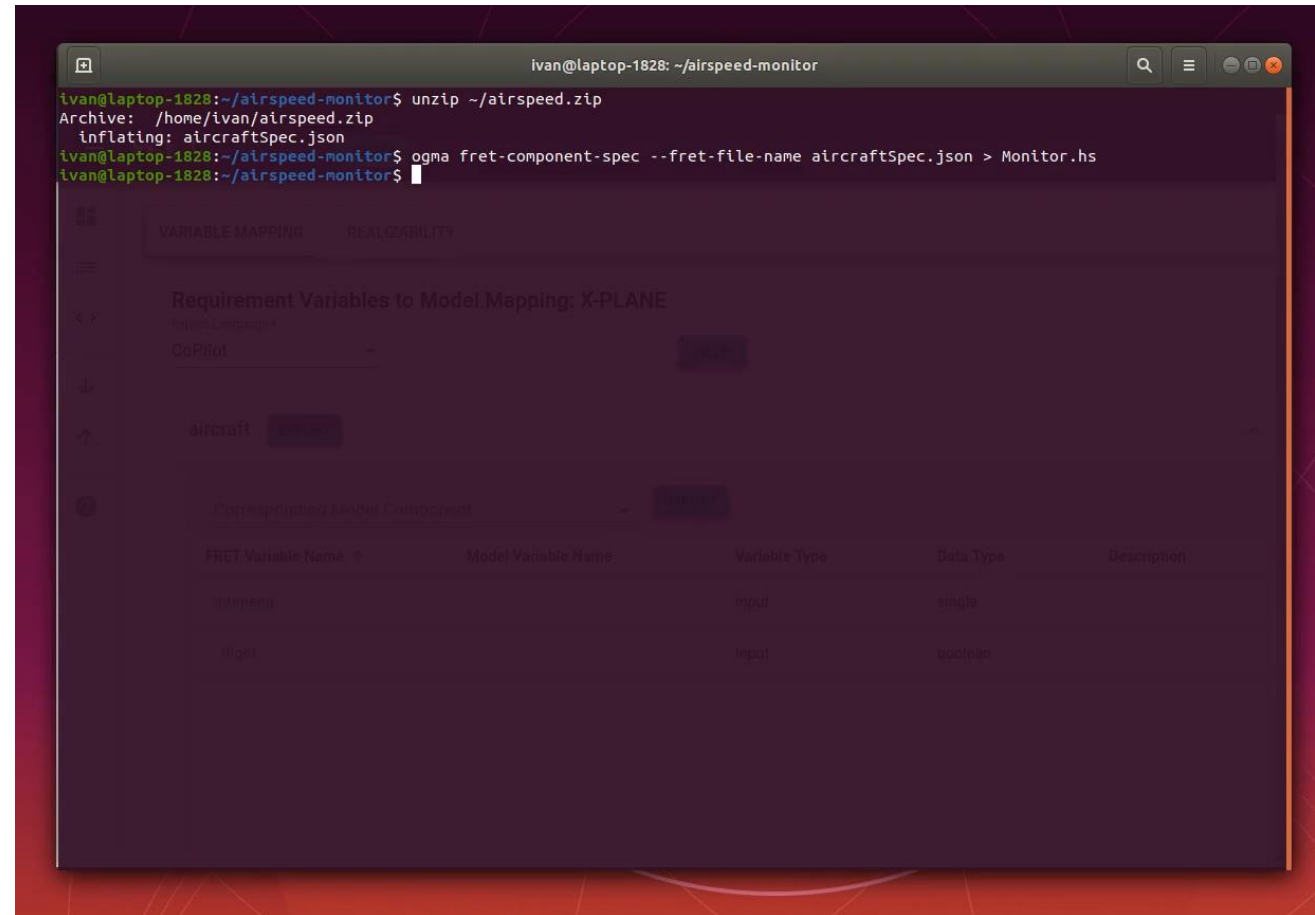
OGMA: Details

- FRET provides multiple encodings of requirements in different TLs. Ogma picks pmLTL
- Ogma generates definitions for temporal operators and expressions used by FRET but not available in Copilot (e.g., first-time point).
- Code is populated with original requirements in NL to facilitate traceability.

OGMA: Process



OGMA: Process

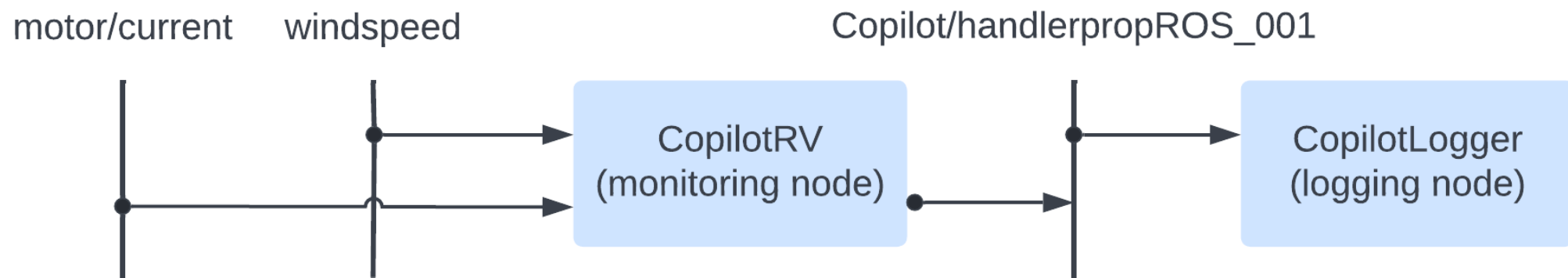


ROS2

- Robot Operating System is a middleware to build robotics applications.
- ROS2 *packages* are defined by *nodes* connected to other nodes via *topics*.
- Communication is based on publish-subscribe: nodes subscribe to topics and are notified when new information is published to them.

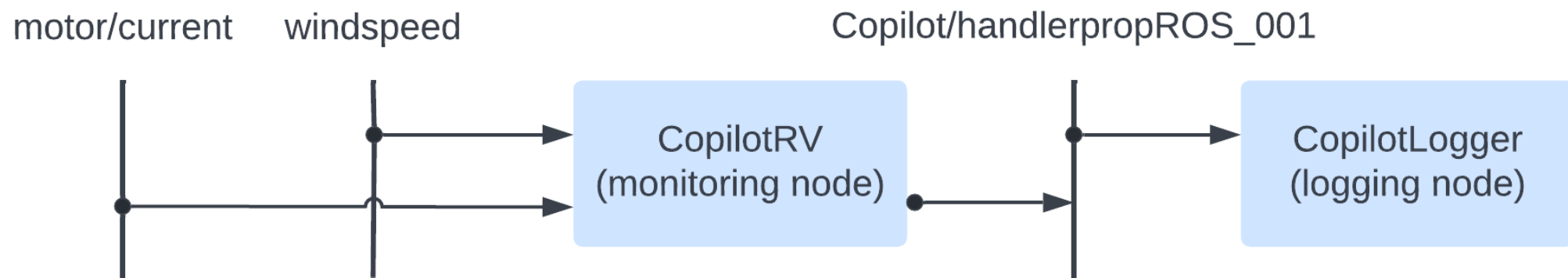
OGMA: ROS2 backend

- Each set of requirements is transformed into one node.
- Each *external variable* used by requirements leads to a subscription to the topic that publishes the variable's values.
- Each monitor result leads to a message published to a dedicated topic.

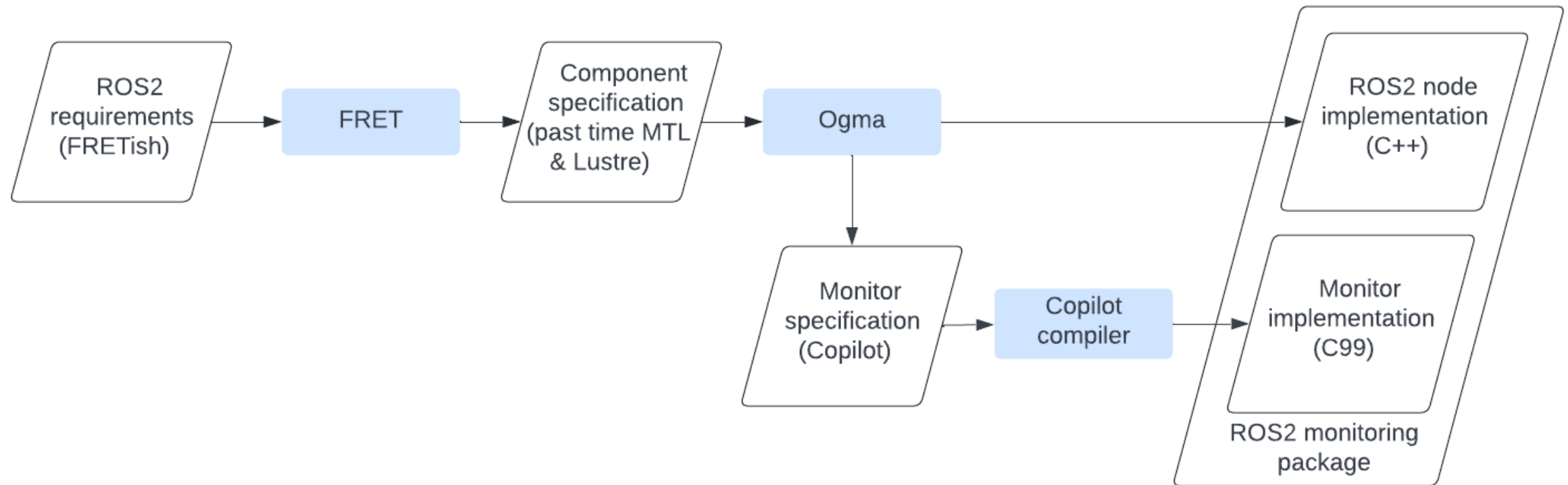


OGMA: ROS2 backend (2)

- Each time new inputs arrive, the CopilotRV node re-evaluates the monitors.
- The logging node is optional: it can serve to observe the behavior of the monitors or used as a template to customize responses.



OGMA: ROS2 backend (3)



Further details

- *"Automated Translation of Natural Language Requirements to Runtime Monitors"*, 2022, Perez et al. TACAS 22.
- *"Integrating FRET with Copilot: Automated Translation of Natural Language Requirements to Runtime Monitors"*, 2022, Perez et al. NASA/TM-20220000049 (extended paper with details on experiments)

<> Code Issues 1 Pull requests Actions Projects Wiki Security Insights

Further details

- <https://github.com/NASA/ogma>
- <https://github.com/NASA-SW-VnV/fret>
- <https://github.com/Copilot-Language/copilot>

develop
6 branches
3 tags
Go to file
Code
About

ivanperez-keera Merge branch 'release-1.0.2' into develop. Close #43.
d68680a 14 days ago
110 commits

docs Briefly introduce FRET, add image of FRET and X-Plane. Refs #16. 4 months ago
changelog. Refs #43. 14 days ago
ogma-core Document changes in changelog. Refs #43. 14 days ago
ogma-extra Document changes in changelog. Refs #43. 14 days ago
ogma-language-c Document changes in changelog. Refs #43. 14 days ago
ogma-language-cocospec Document changes in changelog. Refs #43. 14 days ago
ogma-language-copilot Document changes in changelog. Refs #43. 14 days ago
ogma-language-fret-cs Document changes in changelog. Refs #43. 14 days ago
ogma-language-fret-reqs Document changes in changelog. Refs #43. 14 days ago
ogma-language-smv Document changes in changelog. Refs #43. 14 days ago
.gitignore Introduce library for C header parsing/AST representation. Refs #1. 7 months ago
LICENSE.pdf Replace symbolic link for destination. Refs #5. 7 months ago
README.md Introduce license and README at top level. Refs #4. 7 months ago

No description, v provided.
Readme
180 stars
5 watching
8 forks
Releases
3 tags
Packages
No packages published
Languages
Haskell 93.0%
Makefile 1.3%

Summary

- We use existing FRET to elicit requirements in structured natural language.
- We use Ogma to transform Past-time Linear Temporal Logic encodings of requirements into Copilot monitors.
- We use Copilot to generate C code.
- **We extend Ogma to produce a ROS2 package with a monitoring node that connects the monitors generated from requirements to the rest of the ROS2 application.**
- All tools are open source.

THANK YOU!

Questions?

ivan.perezdominguez@nasa.gov

anastasia.mavridou@nasa.gov

"Monitoring ROS2: from Requirements to Autonomous Robots"

Ivan
Perez

KBR Inc

Anastasia Mavridou

KBR Inc

Tom Pressburger

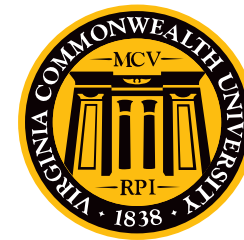
NASA

Alexander
Will

VCU

Patrick J.
Martin

VCU



VCU