Requirements with FRET

Anastasia Mavridou
KBR / NASA Ames Research Center
Lockheed Martin Cyber-Physical System Challenge, component FSM:

• Exceeding sensor limits shall latch an autopilot pullup when the pilot is not in control (not standby) and the system is supported without failures (not apfail).
• The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).

**every time these conditions hold or only when they become true?**

• The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
• The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
• The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.
• ...

are these requirements consistent? does my model/code satisfy them?
what formal analysis tools understand

Lockheed Martin Cyber-Physical System Challenge, component FSM:

```plaintext
var autopilot: bool = (not standby) and supported and (not apfail);
var pre_autopilot: bool = false -> pre autopilot;
var pre_limits: bool = false -> pre limits;
guarantee "FSM-001v2" S(((((autopilot and pre_autopilot and pre_limits) and (pre ( not (autopilot and pre_autopilot and pre_limits))))) or ((autopilot and pre_autopilot and pre_limits) and FTP)) => (pullup)) and FTP), (((autopilot and pre_autopilot and pre_limits) and (pre ( not (autopilot and pre_autopilot and pre_limits)))) or ((autopilot and pre_autopilot and pre_limits) and FTP)) => (pullup));
```
FRET bridges the gap

- **Captures** requirements with an extensible grammar that defines a restricted natural language with **unambiguous semantics**
- **Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation
- **Assists** in writing requirements through requirement **templates**
- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner
- **Checks** **consistency** of requirements and provides feedback
- **Connects** with **analysis tools** and **exports** verification code
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for runtime analysis of C programs with Copilot
Welcome to FRET

https://github.com/NASA-SW-VnV/fret

Team: Dimitra Giannakopoulou, Andreas Katis, Tom Pressburger, Johann Schumann, Khanh Trinh

Alumni: David Bushnell, Tanja DeJong, George Karamanolis, David Kooi, Julian Rhein, Nija Shi
Capturing and explaining requirements

Scope (optional)
specifies where the requirement must hold: either globally or in intervals defined with respect to a MODE, e.g.,

- before MODE, only before MODE
- in MODE, not in MODE, only in MODE
- after MODE, only after MODE

MODE is a string identifier starting with an upper- or lowercase letter, followed by letters, digits or underscores.
The only modes mean that when the system is not in the specified relationship to the mode (i.e., the system is not in/after/before the mode) the response must not occur.
Examples:

- global The system shall always satisfy count >= 0
- After boot mode the system shall immediately satisfy prompt_for_password
- Only after arming mode shall the system eventually satisfy fired
- In landing mode the system shall eventually satisfy decrease_speed
- When not in initialization mode the system shall always satisfy commands_accepted
- Only in landing mode shall the system eventually satisfy landing_gear_down
- Before energized mode the system shall always satisfy energized_indicator_off
- Only before energized mode shall the system eventually satisfy manually_touchable

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.

when in cruising mode the altitude_hold_autopilot shall
Requirement templates

Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.
Requirement templates

Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.
Requirement templates

Rationale and Comments

Rationale

Comments

The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).

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.

component shall always satisfy if (input_state & condition) then output_state

Examples:

FSM_Autopilot shall always satisfy if (state = ap_standby_state & (standby & !apfail)) then STATE = ap_transition_state
The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).

The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
• The autopilot shall change states from **TRANSITION** to **STANDBY** when the pilot is in control (standby).

• The autopilot shall change states from **TRANSITION** to **NOMINAL** when the system is supported and sensor data is good.

State: **TRANSITION**
Checking consistency

Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from **TRANSITION** to **STANDBY** when the pilot is in control (standby).
- The autopilot shall change states from **TRANSITION** to **NOMINAL** when the system is supported and sensor data is good.

State: **TRANSITION**
Condition 1: pilot is in control
Condition 2: system is supported
    sensor data is good
Checking consistency

Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.

State: TRANSITION
Condition 1: pilot is in control
Condition 2: system is supported
sensor data is good
Checking consistency

Lockheed Martin Cyber-Physical System Challenge, component FSM:

• The autopilot shall change states from **TRANSITION** to **STANDBY** when the pilot is in control (standby).

• The autopilot shall change states from **TRANSITION** to **NOMINAL** when the system is supported and sensor data is good.

State: **TRANSITION**
Condition 1: pilot is in control
Condition 2: system is supported
    sensor data is good

Next state 1: **STANDBY**
Next state 2: **NOMINAL**
Checking consistency
### Requirement Variables to Model Mapping: Demo-FSM

#### Export Language

- **Autopilot**

#### FSM

<table>
<thead>
<tr>
<th>FRET Variable Name</th>
<th>Model Variable Name</th>
<th>Variable Type</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_MANEUVER_STATE</td>
<td>apfail</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>AP_NOMINAL_STATE</td>
<td>good</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>AP_STANDBY_STATE</td>
<td>limits</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>AP_TRANSITION_STATE</td>
<td>pullup</td>
<td>Output</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>GOOD</td>
<td>request</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>LIMITS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PULLUP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQUEST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEN_FAULT_STATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Connection with analysis tools
FRET’s mission is to provide an intuitive platform for capturing precise requirements, to serve as a portal to a variety of analysis tools, and to support requirements repair based on analysis feedback.

FRET v2.0 coming soon
https://github.com/NASA-SW-VnV/fret


