A skill fault model for autonomous systems

Gabriela Catalán Medina ¹,²  Jérémie Guiochet ¹
Charles Lesire ²  Augustin Manecy ²

¹LAAS CNRS, University of Toulouse, France
²ONERA/DTIS, University of Toulouse, France

4th International Workshop on Robotics Software Engineering, ROSE@ICSE

May 9th, 2022
Introduction

Context
- Autonomous systems and architecture: 3 layers

Problematic
- Faults in autonomous architectures may occur at all layers, usually managed with local treatment
- How to specify consistent fault detection and recovery mechanisms in such architecture?
Our approach

- Use a systematic approach for fault analysis (model-based), and to specific recovery mechanisms
- Study how this detection and recovery mechanisms can be implemented at the executive layer (skill layer)

Our case study

- Autonomous drone (DJI M600), Infrastructure inspection Beyond Visual Line Of Sight (BVLOS)
Skills > Overview

- Skill layer converts plans sent by the decisional layer into primitive services (skills) realized by the functional under layer.
- Skills are primitive services described by a skill model and a skill implementation.
- Skill model and skill implementation are used to generate skill managers for the skill layer.
- All skills share a same finite state machine (FSM), see next slide.
Skills > Skill FSM

- NV
- CR
- NR
- RI
- RI
- dispatch
- Rg
- T_i
- I_g
- T_N

- valid
- start
- pre
- invariant
- interrupt
- terminate_{M_1}
- post_{M_1}
- effect_{M_1}
- post_{M_1}
- effect_{M_1}
- ...
Skills > Skill model example

```plaintext
skill takeoff {
  input {
    height: float64 // validate can fail if h>h_geo_fence
    speed: float64 // maximum ascending speed
  }
  effect {
    take_control: axes_authority -> USED
    release_control: axes_authority -> AVAILABLE
  }
  precondition {
    sdk_authority: resource=(SDK_authority==AVAILABLE)
    not_moving: resource=(axes_authority==AVAILABLE)
    on_ground: resource=(flight_status==ON_GROUND)
    home_valid: resource=(homepoint_status==VALID)
    success take_control
  }
  invariant {
    keep_sdk_authority: resource=(SDK_authority==AVAILABLE)
    -> violation=release_control
    in_control: resource=(axes_authority==USED)
  }
  result {
    AT_ALTITUDE: apply=release_control
    BLOCKED: apply=release_control
    ABORTED: apply=release_control
  }
}
```
Skill fault model pattern

DM / FM Legend

DM1 = pre (skill model)
FM1 = NR (skill FSM state)
DM2 = valid (Skill implementation)
FM2 = NV (skill FSM state)
DM3 = invariant (Skill model)
FM3 = RI (skill FSM state)
DM4 = callback function (Skill implementation)
FM4 = M_i or \( \bar{M_i} \) (skill model)
DM5 = post (skill model)
FM5 = \( \bar{M_i} \) (skill model)
DM6 = callback function (skill implementation)
FM6 = M_i or \( \bar{M_i} \) (skill model)
Skill Fault Analysis Process

1. **Error analysis**: list all the events (errors) that may impact correct skill execution.

2. **Fault tree analysis**: design **skill fault tree** based on the **skill fault model pattern**. Connection of each event listed in (1) with each **skill fault tree**.

3. **Detection and recovery**: determination of potential detection mechanisms (DMₖ) and skill failure modes (FMₖ) for each branch of the fault tree.

4. **Design/Verification**: Design / Verify the skill model or implementation to add or correct a missing or incomplete DMₖ or FMₖ.
Case study > Architecture overview
Case study > Fault tree for Take-off skill
Next step is then to identify inconsistencies or unmanaged events in the fault tree and to modify the skill model or skill implementation:

- Identify errors can propagate up to top of the fault tree, i.e. lead to a skill failure without being handled by a DM/FM mechanism (e.g. modification of the skill model, see paper).
- Errors of different nature could lead to the same failure modes of the skill, but with different detection mechanisms.
- Find redundant checks (DM) and optimize them
Conclusion

- Support for skill design and analysis to deal with detection and recovery mechanisms
- Generic and model-based (skill model + fault trees)
- But only validated so far on drone applications and no we need to evaluate how this really improve safety and availability at system level
- Current development: study how the fault tree can also be used to generate test cases (to validate the detection and recovery mechanisms)