Automotive WG Update - *Tell tales an evolution use case towards driver assistance?!*

September 7 - 8, 2022 | Virtual Event

Philipp Ahmann, Robert Bosch GmbH
supported by work from
Paul Albertella, Codethink and Christopher Temple, Arm

Automotive WG

Goal(s):

“Discuss the conditions and prerequisites the automotive sector needs to integrate Linux into a safety critical system. We focus on actual use cases from the Automotive domain to derive the technical requirements to the kernel and the development process.”

Activities:

- Documentation (item, system, requirements, architecture, design, …)
- Implementation (reference implementation illustrating the theory in practice)
- Iterating forth and back between activities
- Reach out to other projects and communities

Agenda

- What is a tell tale and why is it the use case of the Automotive WG?
- STPA (if you don’t know what it stands for)
- How the WG approaches the use case using STPA and what is the status.
- Next steps & Outlook
- Derivative work from the use case.

Why tell tales?

Dashboards (and warning signs) become digital

- Mechanical dashboards and LED based safety related warning signs (tell tales) turn into pure display solutions implemented in the OS.
- Use case needed as a backdrop to derive safety requirements
- Tell tales don’t involve complex sensor/actuator setups
- Relatively easy to understand
- Basic challenges representative for more complex use cases

Photo by Randy Tarampi on Unsplash

Use case benefits

- Fewer subsystems & components, compared to driver assistance use cases.
- Good to visualize and explain to others.
- Moderate safety integrity level *(typically ASIL A or B)*
- Relaxed timing constraints. *(Driver reaction time longer than display timing.)*
- Fairly simple high-level system diagram. *(See in a few slides)*
- Sample implementations (without safety in mind) exist e.g. from AGL.
- System level architecture in place, with possibility to scale to different hardware and software architectures and increase safety demands to the Kernel step by step.

Use case benefits

How to approach this use case?

STPA
(if you don't know what it stands for)

STPA Analysis

- STPA stands for **S**ystems **T**heoretic **P**rocess **A**nalysis
- With simple words STPA can be compared to a HARA
- Concrete use for Automotive WG telltale use case in the slides from Paul Albertella

- Reading more about the methodology in the handbook: https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

What is STPA

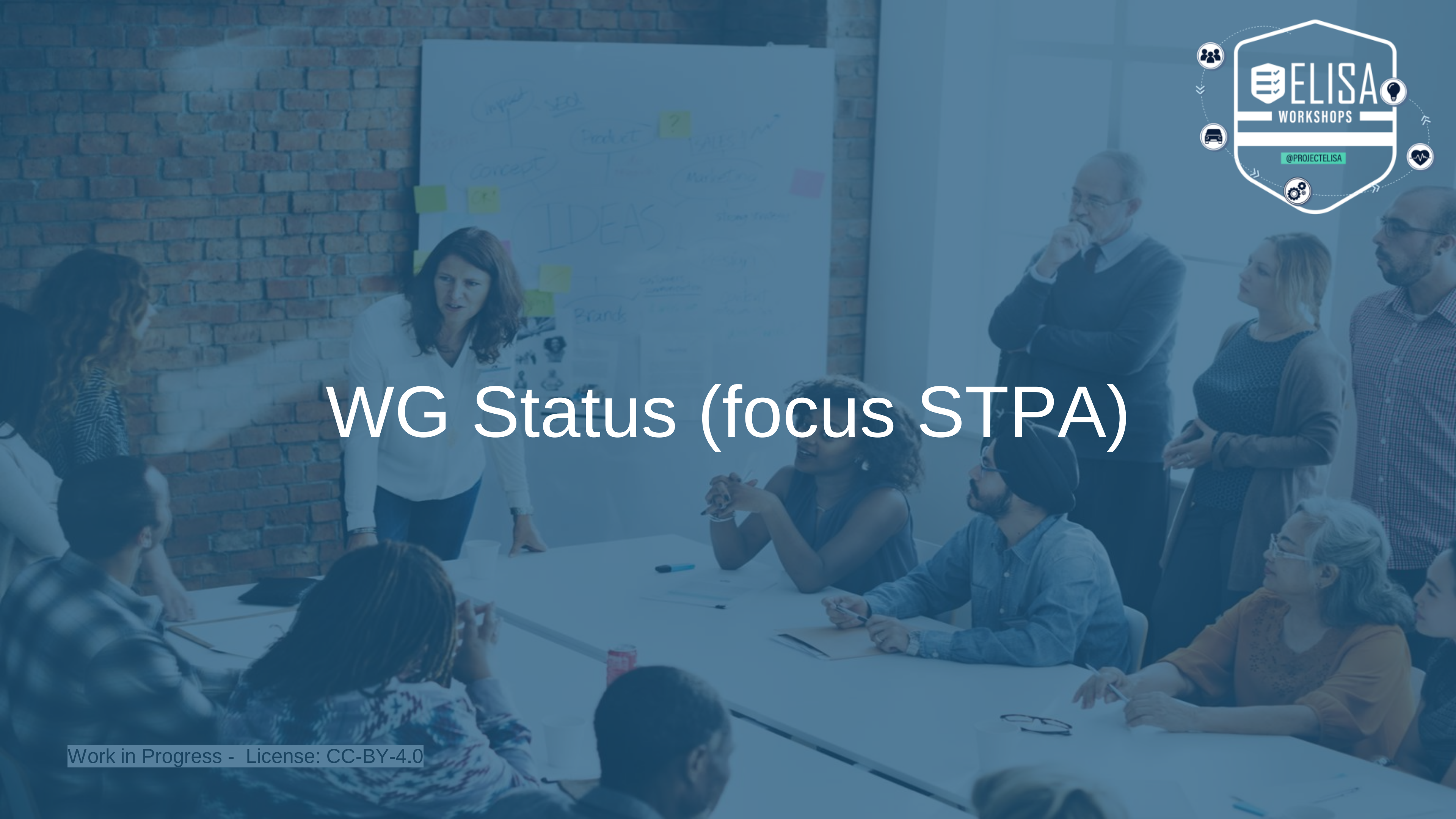
“STPA (System-Theoretic Process Analysis) is a relatively new hazard analysis technique based on an extended model of accident causation. In addition to component failures, STPA assumes that accidents can also be caused by unsafe interactions of system components, none of which may have failed.“

Nancy Leveson, 2018, STPA handbook, chapter 1

Advantages of STPA

- Very complex systems can be analyzed.
- can be started in early concept analysis to assist in identifying safety requirements and constraints.
- help make more and more detailed design decisions
- includes software and human operators in the analysis
- provides documentation of system functionality
- can be easily integrated into (model-based) system engineering process

https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf



WG Status (focus STPA)

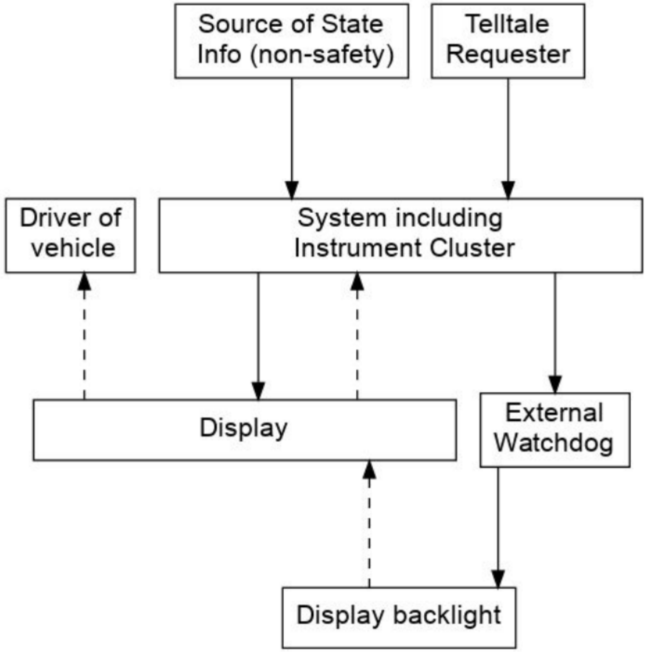
Use case analysis (using STPA)

- STPA was selected to describe requirements to safety in useful details.
- With the help of STPA important system elements and their safety responsibilities as well as the safety responsibility of Linux should be identified.
- The kernel level was left by intention and the use case was put on a higher system level to frame the problem there, before going back to Kernel level
  - Exclude parts of the system which are not relevant for the problem (Level 2 diagram), but have a way back to the wider system context (Level 1)
- The analysis enables the chance to put risks to different domains/units.
  - Where are strength and limitations of Linux and what can be achieved?

Tell tale system context

- Part of a system that includes the Instrument cluster
- Responsible for displaying safety-critical information (tell tales) as part of cluster display
- Includes an external watchdog

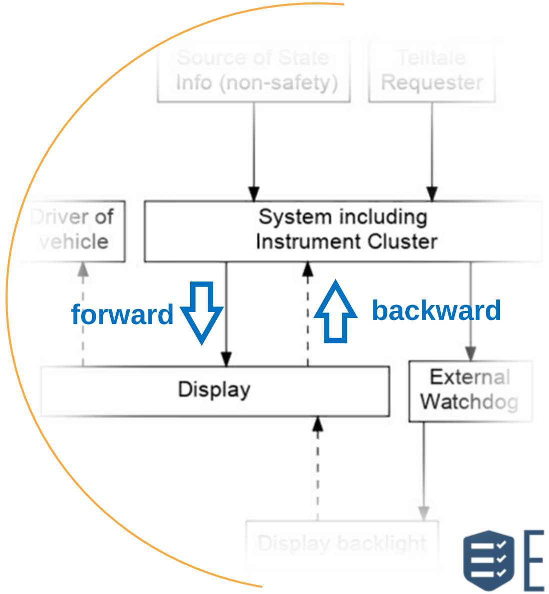
- Analyzed using STPA



The thing with forward & backward path

- ⇩ Forward: Rendering the warning sign on the display
- ⇧ Backward: Read and check the displayed tell tale for correctness.

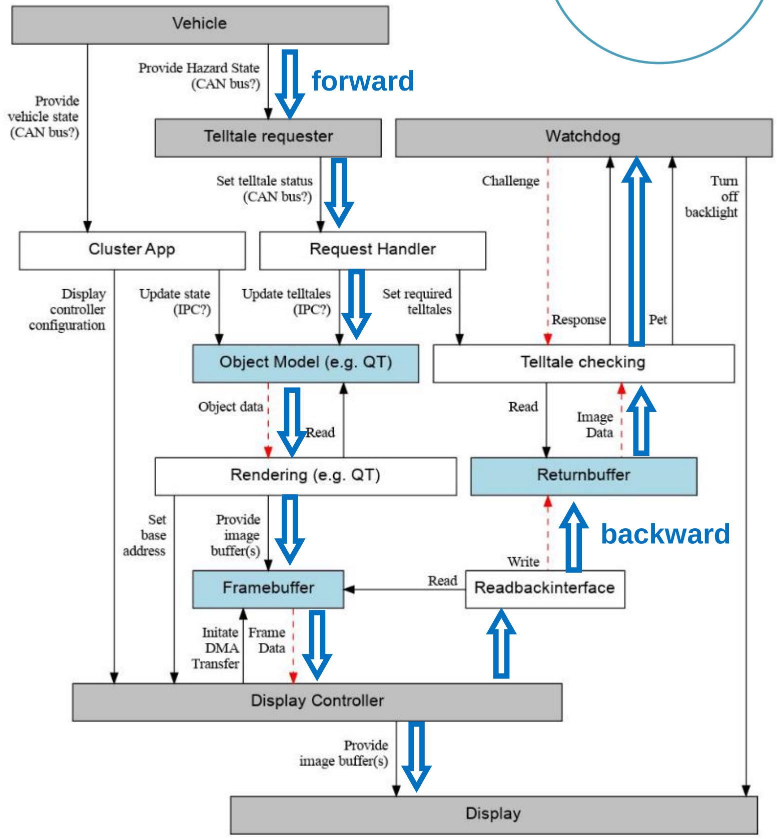
- Faults can occur that would disrupt the correct execution of the forward path.
- Although we rely on the external watchdog, there is responsibility for Linux.
- Linux has the responsibility to be reliable, robust, stable, dependable, … (These are not safety properties. 😉)
- The system is considered in a way, that the watchdog is never triggering safe state.



Control structure

- Safety critical path is the tell tale checking.
- The *Readbackinterface* will be implemented differently in real world
  (! break with example implementation)

- This use case level is detached from Linux as an OS and can be easily transferred to other operating system and use cases.
- Also important: Object Model, Framebuffer, Rendering, Returnbuffer may have different timings/frequencies.



Similarity with other use cases

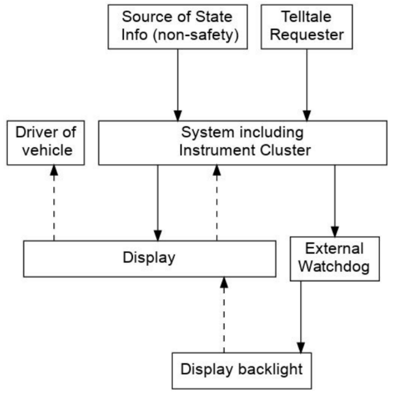
- The use case was selected to derive requirements to the Linux Kernel
- It follows the SEooC flow from ISO 26262 part 10
  - “SEooC” means “safety element in an assumed context”

Warning: slight oversimplification

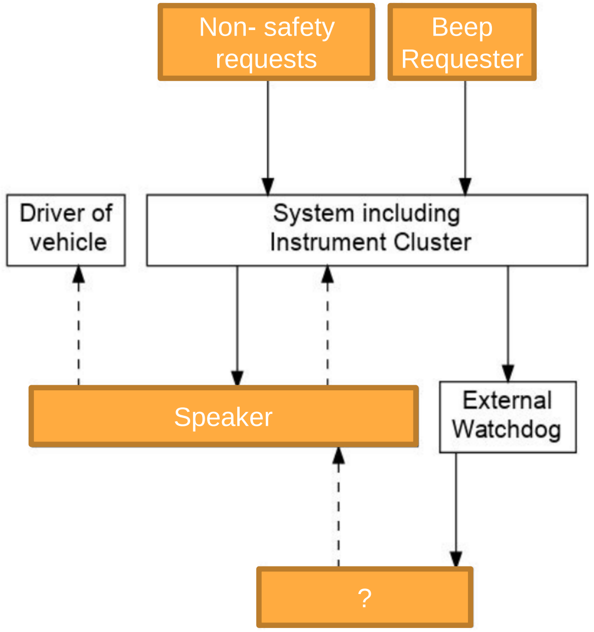
- The use case boils down to “a memory region need to be checked against another memory region within a specified time”

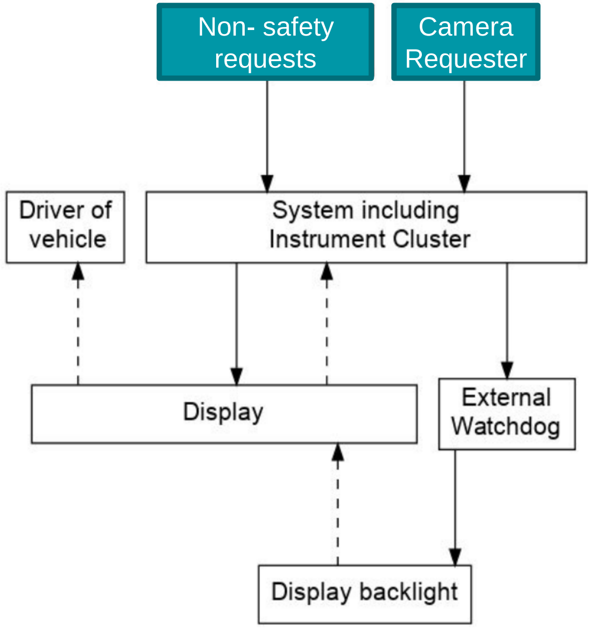
Similarity with other use cases

Basic challenges representative for more complex use cases



Use the tell tale use case work to derive your own use case.





Outlook & Next steps

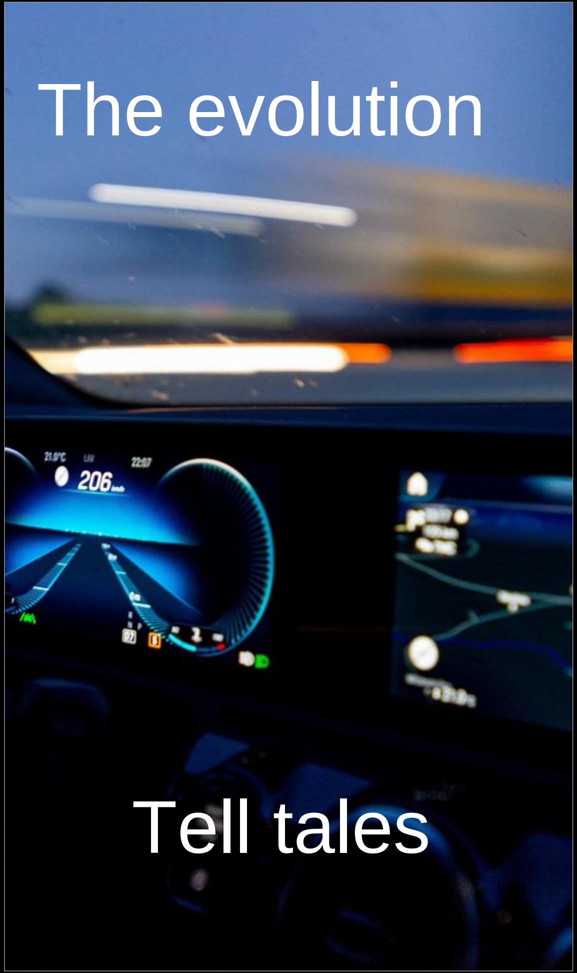
Getting back to code (via design)

- Write down design, which should also bring us closer Architecture WG
  - How do we implement this?
  - How do we satisfy the requirements?
- Focus on “*what matters for safety*”, rather than “*you have to do it in this way*”.
- Figure out which low level components are need to fulfill the safety requirements and how they generate a complete Unsafe Control Actions coverage.

Additional next steps

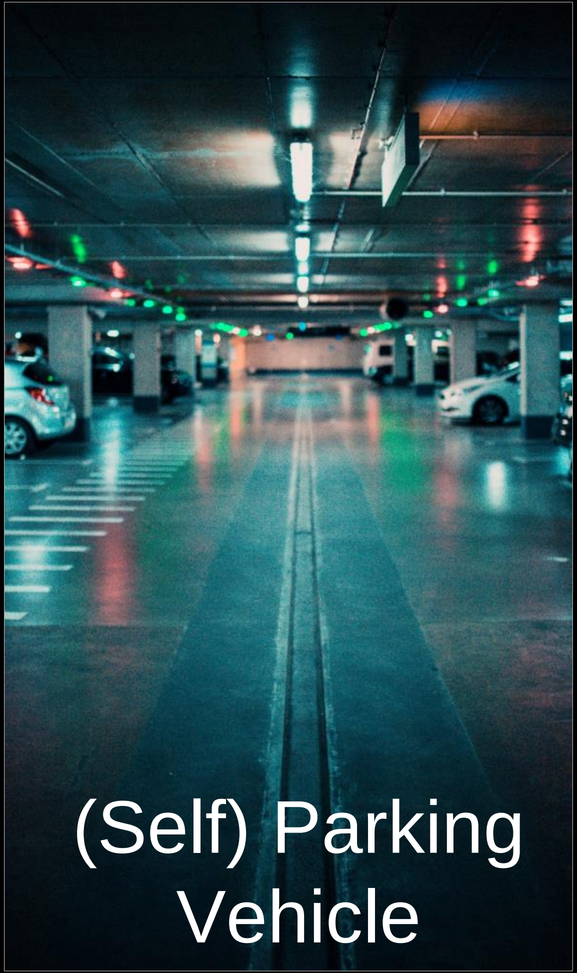
- Align with Architecture work group
    - Discuss in person during ELISA Workshop in Manchester
    - Join their weekly meetings
- Bring up the tell tale demo together with Systems WG
- Promote the use case as a way forward and starting point for additional use cases

The evolution



Tell tales

Photo by Randy Tarampi on Unsplash



(Self) Parking Vehicle

Photo by Patryk Sikora on Unsplash

Autonomous Driving

Photo by Roberto Nickson on Unsplash

Photo by Pete Pedroza on Unsplash

Licensing of Workshop Results

All work created during the workshop is licensed under *Creative Commons Attribution 4.0 International (CC-BY-4.0)* [https://creativecommons.org/licenses/by/4.0/] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.