# Safety Relevant Time Intervals for MobSTr

Jan Steffen Becker Björn Koopmann Ingo Stierand

**Abstract**

This report briefly explaines the safety concept in the MobStr use case and derives timing requirements and a contract specification of the safety concept based on the original 2019 WATERS verification challenge.

## 1 Introduction

The MobStr (Model-based Safety Assurance and Traceability) data set [10, 9] is a collection of example artifacts showcasing an ISO 26262-compliant development process. It is based on the software model and use case description of the 2019 WATERS verification challenge[1]. As part of the PANORAMA project, experts from academia and industry have extended the original challenge description and model with additional artifacts that are needed for an ISO 26262-compliant development process and safety case. So far, a component model, hazard and risk analysis (HARA), safety requirements and safety analyses (FTA/FMEA) as well as a traceability model have been added. The artifacts do not aim at completeness, but rather shall provide a rich set of different and linked development artifacts. The artifacts are available online at GitHub[2]. A stable snapshot of the dataset is also available under [11].

In the MobSTr data set, the system under design (SUD) is an autonomous passenger car that shall follow a pre-defined route, and thereby safely avoid collisions with stationary objects or other road users. The SUD is equipped with different sensors, including camera, LIDAR, and GPS. The logical software architecture is depicted in Figure 1. The grabber components collect and preprocess the raw sensor data, and provide them to the localization and detection components which locate the ego vehicle and its environment on a map. This information is used by the planner task to calculate a trajectory and provide high-level steering and speed commands to the controller component, which translates them into low-level actuator commands. The CAN bus interface is used for communication with the vehicle actuators and for acquiring additional odometry data. Technically, localization and detection is realized by a structure from motion (SFM) approach with an extended Kalman filter. As part of the MobSTr data set, the original architecture description has been accomplished with safety mechanisms, such as redundant object localization (once via camera and once via LIDAR), and an explicit *sensor fusion* component that merges the two sources of information and uses them for improved error detection.

[1] `https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/waters-industrial-challenge/index.html,lastvisitedon2021-08-09`

[2] `https://github.com/panorama-research/mobstr-dataset`

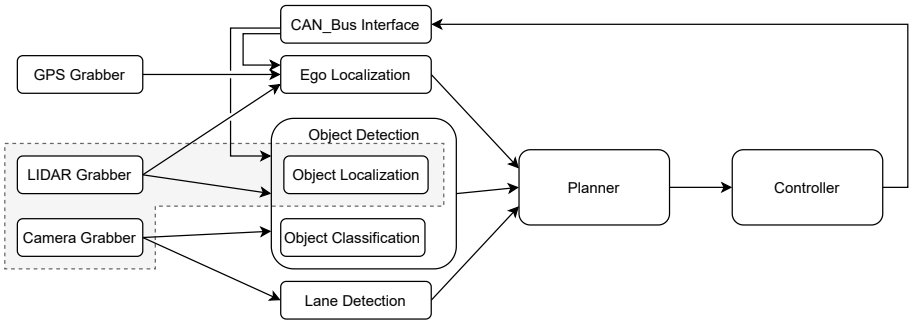


Figure 1: Logical software architecture. The highlighted part is specified with formal contracts in Section 5

This document describes how safety relevant time intervals for the MobSTr safety concept are derived. Furthermore, timing requirements for the camera grabber, LIDAR grabber, and object localization components are defined. In a classical V-process, the component-level timing requirements would be derived top down from the item-level safety requirements, and the software would be designed in order to meet them. Because for the MobSTr data set the software and hardware is fixed in advance by the WATERS challenge model, it is not possible to follow this process entirely. The WATERS challenge model has to be seen rather as a kind of legacy system. The component-level timing requirements are derived in a meet-in-the-middle approach from both the top-level safety requirements and an response-time analysis of the legacy system. As a source of response times, the solution of the WATERS challenge by Krawczyk et. al. [6] is used.

It is important to notice that it is not required for the response time analysis of the legacy system to be accurate and complete. In the safety process proposed in the PANORAMA project, integration and timing analyses are applied iteratively together with optimization of the design. The purpose of the approach described here is therefore to come up with an *initial* set of timing requirements that serves as a starting point for further refinement—it is not necessary that it is correct and complete by design. For the MobSTr data set this is even impossible, since the WATERS challenge model does not reflect the safety extensions to the SUD introduced in PANORAMA. In a safety process, virtual integration tests and timing analysis can be used to validate and verify the final safety requirements. In the outlook (Section 7) of this report, expected findings of those analyses with respect to the timing requirements derived here are described.

The reminder of the report is structured as follows. Section 2 gives an overview on the MobSTr safety concept. In Section 3, the item and system-level fault tolerant time intervals are calculated based on a simple kinematic model and simple assumptions. In Section 4, exemplary timing behavior for the controller components is inferred from their implementation as modeled in the WATERS challenge. Based on that, Section 5 gives a specification of the object detection and grabber components in a formal contract specification language. The report concludes with a short summary and outlook in Section 7.

## 2 Safety Concept Overview

The overall safety concept is as follows. As identified in the HARA (cf. [9] and the online dataset), the system need to ensure that avoidance and braking maneuvers are initiated in time. In this document, the case of an (emergency) brake is evaluated, because it is easier to handle than avoidance maneuvers or lane keeping. However, these cases are structured analogous.

When developing a safety concept in accordance to ISO 26262, at first the *safe state* of the system has to be defined. For SAE levels 3 and 4, a safe state cannot be reached by immediately shutting down the system, since this would lead to a loss of control [8]. Instead, possible solutions are stopping the car in a safe place, or transfer control to the driver before shutting down the system. Both approaches lead to a safe state. As calculated in the next section, it needs to be possible to initiate an emergency brake within about 2.5s if necessary, which is too short to reach one of the safe states. Note that initiating an emergency brake upon occurrence of a fault is considered as *unsafe* behavior, since it would unreasonably increase the risk of other traffic participants crashing into the ego vehicle[3]. Hence, the chosen safety concept is to enter a degraded mode whenever a fault is detected and let the driver take over. Until driver take-over is complete, the system still must execute required avoidance maneuvers.

As a consequence, the MobSTr safety concept defines different safety mechanisms that can handle at least the single point faults. In this document, single point faults occurring in the camera and LIDAR sensor are considered. The basic idea of the safety concept is to detect and filter corrupted camera and LIDAR data frames as part of the camera and LIDAR grabber components. Here, filtering means that the corrupted data frame is simply dropped and no update of the processed data is made. Instead, a warning message is displayed to the driver. The driver warning is not elaborated in this report.

## 3 System-level Fault-tolerant Time Interval

The overall approach for calculating the safety relevant time intervals follows the work of Frese et al. [4], that has been adopted to the use case. In contrast to the original work, the safety concept here assumes that a fault is detected *before* a controller output is set, instead of correcting wrong outputs. Figure 2 illustrates how the different time intervals relate to each other.

Table 1 lists the constants used in the following calculations. Note that these are example values based on a rough guess, and may differ from real world data. They have been chosen in order to fit the MobSTr use case.

First, the Fault Tolerant Time Interval (FTTI) from a global perspective has to be determined. In case of an obstacle, it must be possible to stop the vehicle within the sensor range. We can split this range into the *reaction distance* $d_{\mathsf{react}}$ and *braking distance* $d_{\mathsf{brake}}$ leading to

$$d_{\mathsf{sens}} = d_{\mathsf{react}} + d_{\mathsf{brake}}$$

with

$$d_{\mathsf{brake}} = \frac{v_0^2}{2a_{\mathsf{brake}}}$$

[3]Unintended braking has been identified as a hazard

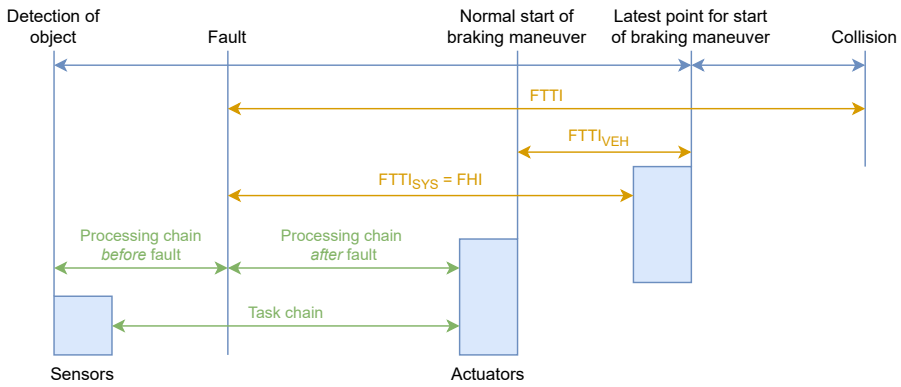


Figure 2: Safety-relevant time intervals

| | variable | value (approx.) |
|---|---|---|
| Sensor range | $d_{\mathsf{sens}}$ | $50m$ |
| Speed when fault occurs | $v_0$ | $50km/h \approx 14m/s$ |
| Emergency brake deceleration | $a_{\mathsf{brake}}$ | $7m/s^2$ |
| Sensor processing time | $t_{\mathsf{sense}}$ | $50ms$ |
| Actuator time | $t_{\mathsf{act}}$ | $100ms$ |
| Event chain WCRT (normal operation) | $t_{\mathsf{chain}}$ | $< 800ms$ |

Table 1: Constants

and

$$d_{\mathsf{react}} = d_{\mathsf{sens}} - \frac{v_0^2}{2 \times a_{\mathsf{brake}}} \quad .$$

Switching from the spatial to the timing domain, the *reaction* and *braking times* are

$$t_{\mathsf{react}} = v_0 \times d_{\mathsf{react}} = \frac{d_{\mathsf{sens}}}{v_0} - \frac{v_0}{2 \times a_{\mathsf{brake}}}$$

and

$$t_{\mathsf{brake}} = \frac{v_0}{a_{\mathsf{brake}}}$$

leading to

$$\mathsf{FTTI} < \mathsf{FTTI}_{\max} = t_{\mathsf{react}} + t_{\mathsf{brake}} = \frac{d_{\mathsf{sens}}}{v_0} + \frac{v_0}{2 \times a_{\mathsf{brake}}} \approx 4.57s \quad .$$

Note that the bound for the FTTI has been calculated for faults occurring at the first point in time where an object could be detected by the sensors (i.e the fault is "object not detected"). For faults that occur later in the processing chain, the FTTI is shortened by the time from object detection until occurrence of the fault. So we have as a lower bound

$$\mathsf{FTTI} > \mathsf{FTTI}_{\max} - t_{\mathsf{chain}} \quad .$$

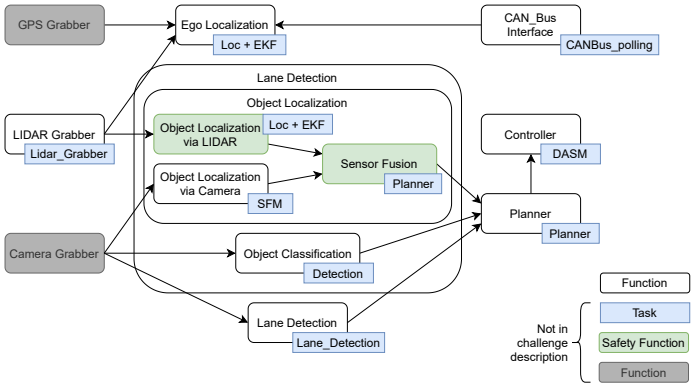


Figure 3: Mapping components to task chains

We specify the example time bounds in the MobSTr data set with respect to sensor faults. From an engineering perspective, the time interval $\mathsf{FTTI}_{\mathsf{SYS}}$ is the relevant one, as it describes the time that the system under design has for fault handling. In terms of ISO 26262 [5], this is the item-level *fault handling time interval (FHI)*. Considering the braking, actuator and sensing times, this is

$$\mathsf{FHI} = \mathsf{FTTI}_{\mathsf{SYS}} \leq t_{\mathsf{react}} - t_{\mathsf{act}} - t_{\mathsf{sense}} \approx 2.43s$$

assuming that the fault is first visible at the sensor output.

**A short remark on fault detection and reaction times** According to ISO 26262 and [4], the FHI can be split into *fault detection time interval (FDI)* and *fault reaction time interval (FRI)*. There is no unique approach on how to choose the split point between FDI and FRI, except that $\mathsf{FDI} + \mathsf{FRI} \leq \mathsf{FHI}$ must hold. ISO 26262 requires explicit specification of the FDI (and so the FRI) only in case of multi-point faults; in case of single-pont faults as considered in this report, specification of the FHI is sufficient. Furthermore, splitting the FHI into FDI and FRI does not make sense here for the following reasons: ISO 26262 as well as Frese et al. assume that a safety mechanism detects a wrong output within the FDI and mitigates it within the FRI. However, this does not apply to the software-based safety concept implemented in this work. As described in the following section, the safety mechanisms that handle sensor failures are implemented as part of the controller software. Hence, sensor data is checked for correctness before being processed, regardless of the timing. The system is safe as long as the processing chain mitigates these faults and the worst case processing time, whether or not faults have been detected, is below the FHI as calculated above. Because sensor faults are handled by *not* updating output data, no events corresponding to fault detection can be observed. Overall, only the FHI itself but not the split into FDI and FRI are relevant for the safety concept.

# 4 Component-level Timing

The timing contracts will formalize two characteristic attributes of each component: The input/output delay $\Delta(\mathsf{COMP}) = [\Delta^-(\mathsf{COMP}), \Delta^+(\mathsf{COMP})]$, which is the time between the creation times of an input event and the corresponding output event. Secondly, the period $A(\mathsf{COMP}) = [A^-(\mathsf{COMP}), A^+(\mathsf{COMP})]$ with that input events are assumed to be produced. In order to derive realistic timing requirements for the safety mechanisms that are likely to be fulfilled in the APP4MC model, we backwards calculate these values from the FMTV challenge solution presented by Krawczyk et al. in [6]. As a basis, each component $\mathsf{COMP}$ is mapped to a task chain $T(\mathsf{COMP})$. The components and tasks are depicted in Figure 3. Within the MobSTr data set, this mapping is represented as `implements component` tracelinks. Furthermore, we denote by $T_0(\mathsf{COMP})$ the first task in this chain and by $T_{-1}(\mathsf{COMP})$ the set of tasks that produce inputs that are consumed by the chain. For each task $\tau$ in a chain, we denote its period, best and worst care response time by $\pi_\tau$, $\mathsf{BCRT}(\tau)$ and $\mathsf{WCRT}(\tau)$. Having these values at hand, we calculate

$$
\begin{aligned}
A^+(\mathsf{COMP}) &\gtrapprox \max_{\tau \in T_{-1}(\mathsf{COMP})} (\pi_\tau + \mathsf{WCRT}(\tau) - \mathsf{BCRT}(\tau)) \\
A^-(\mathsf{COMP}) &\lessapprox \min_{\tau \in T_{-1}(\mathsf{COMP})} (\pi_\tau - \mathsf{WCRT}(\tau) + \mathsf{BCRT}(\tau)) \\
\Delta^+(\mathsf{COMP}) &\gtrapprox \sum_{\tau \in T(\mathsf{COMP})} (\pi_\tau + \mathsf{WCRT}(\tau)) \\
\Delta^-(\mathsf{COMP}) &\lessapprox \sum_{\tau \in T(\mathsf{COMP})} \mathsf{BCRT}(\tau)
\end{aligned}
$$

The actual values for $A(\mathsf{COMP})$ and $\Delta(\mathsf{COMP})$ are chosen by rounding the right-hand side equations to multiples of $5ms$. Figures 4 and 5 sketch the idea behind the above formulae. Note, that these are very conservative and rough approximations without guaranteed accuracy. Nevertheless, they are sufficient here, because only an initial set of requirements shall be derived that is to be refined in future iterations of the development process. Since they are not correct by construction, they must be subject to verification and validation later on (see Section 6). Table 3 lists assumed input periods ($A(\mathsf{COMP})$) and required response times ($\Delta(\mathsf{COMP})$) for individual components as calculated according to above formulae. Note that the delay may exceed the input period because the new input may occur already before the last one has been processed. Table 2 lists the task periods and response times as analyzed in [6]. In the paper [6], no BCRT is reported. In the table, BCRT is approximated. In case of CPU offloading, the offset $O$ of the post-processing runnable is used as an approximation. For other tasks, the BCRT is approximated as $C^- + \lambda A^-$, where $C^-$ is the best-case execution (CPU) time of the task, and $\lambda A^-$ the best-case communication overhead.

Because the WCRT of an event chain may be below the sum of worst case delays between pairs of consecutive events, the effect of skipping single task instances can hardly been foreseen exactly, but it is very unlikely that it will exceed the $\mathsf{FTTI}_{\mathsf{SYS}}$. Whether the task chain still meets the safety goal needs to be verified, of course.

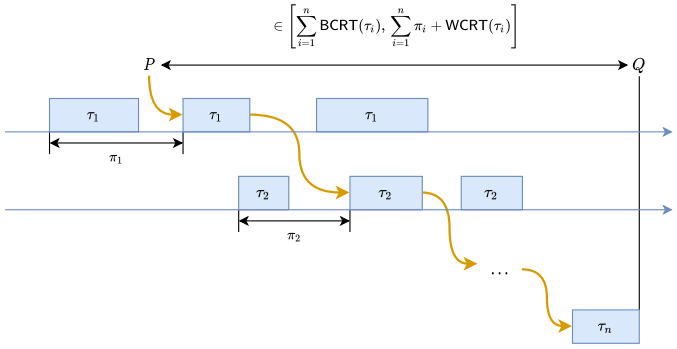


Figure 4: Calculation of the response time $\Delta(\mathsf{COMP})$

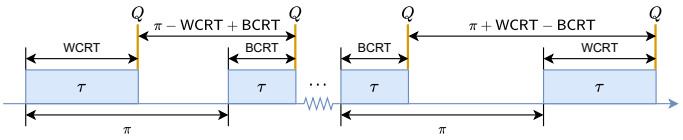


Figure 5: Calculation of the input period $A(\mathsf{COMP})$

| Task | $\pi$ | WCRT | BCRT |
|---|---|---|---|
| Lidar_Grabber | $33ms$ | $25.7ms$ | $10.2 + 1.1 = 11.3ms$ |
| Planner | $12ms$ | $12ms^*$ | $9.5 + 0.2 = 9.7ms$ |
| DASM | $5ms$ | $1.9ms$ | $1.3 + 0.0 = 1.3ms$ |
| Detection | $200ms$ | $151.3ms$ | $108.3ms$ |
| Lane_Detection | $66ms$ | $58.9ms$ | $47.8 + 1.3 = 49.1ms$ |
| Localization | $400ms$ | $372.0ms$ | $117.1ms$ |
| EKF | $15ms$ | $5.4ms$ | $4.0 + 0.0 = 4.0ms$ |
| SFM | $33ms$ | $30.2ms$ | $22.2 + 0.5 = 22.7ms$ |

Table 2: Task periods $\pi$ and response times WCRT as calculated by [6]. The BCRT is approximated either by $\mathsf{BCRT} = C^- + \lambda\mathcal{A}^-$ or by the offset $O$ of the post-processing task, in case of GPU-offloading.

| Component | Tasks | $T_{-1}$ | $A(\mathsf{COMP})$ | $\Delta(\mathsf{COMP})$ |
|---|---|---|---|---|
| LIDAR Grabber | Lidar_Grabber | – | – | $[10, 60]ms$ |
| Camera Grabber | – | – | – | $([10, 60]ms)$ |
| Sensor Fusion | Planner | EKF, SFM | $[10, 45]ms$ | $[5, 25]ms$ |
| Path Planner | Planner | Lane_Detection, Detection, EKF, SFM | $[10, 450]ms$ | $[5, 25]ms$ |
| Controller | DASM | Planner | $[5, 15]ms$ | $[0, 10]ms$ |
| Object Classification | Detection | – | $([15, 40]ms)$ | $[100, 320]ms$ |
| Lane Detection | Lane_Detection | – | $([15, 40]ms)$ | $[45, 125]ms$ |
| Obj. Localization LIDAR | Loc., EKF | Lidar_Grabber | $[15, 40]ms$ | $[120, 795]ms$ |
| Obj. Localization Camera | SFM | – | $([15, 40]ms)$ | $[20, 55]ms$ |
| Ego Localization | Loc., EKF | Lidar_Grabber | $[15, 40]ms$ | $[120, 795]ms$ |

Table 3: Timing requirements and assumptions. Values in brackets depend on camera and GPS grabbers which are not part of the WATERS challenge, and therefore have been chosen freely instead of being calculated.

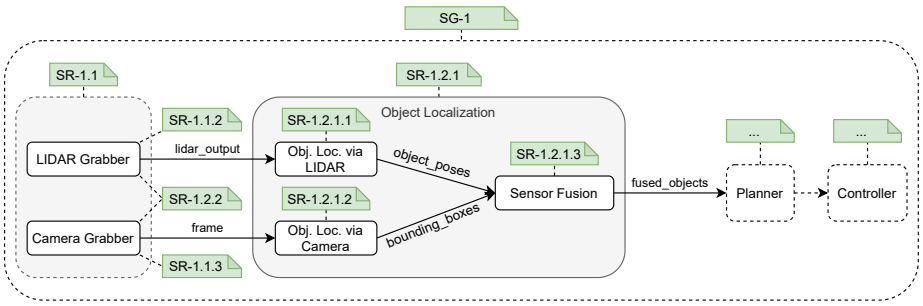


Figure 6: Components and mapping to safety requirements

# 5 Timing Contracts

In this section, the results from the previous two sections are used to calculate time bounds for the textual requirements given in Table 4 and mapped to the components as depicted in Figure 6. The time bound for SG-1 is the FHI as calculated in Section 3. The time bounds for the lowest level safety requirements SR-1.1.2, SR-1.1.3, SR-1.2.1, and SR-1.2.2 is the $\Delta^+$ bound as calculated in Section 4. For the mid-level safety requirements SR-1.1 and SR-1.2 the time bounds have been chosen in order to fit both the time bounds at lower and upper levels. Once again, recall that the time bounds are still subject to verification.

For specifying contracts, we adopt the extension of the MTSL pattern lan-

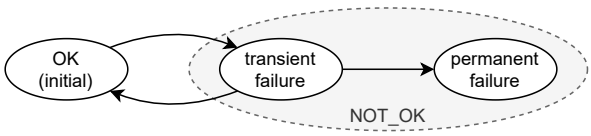


Figure 7: Failure modes

| ID | Description | Time Bound |
|---|---|---|
| SG-1 | The system shall prevent omitting required braking maneuvers. | 2.43s |
| SR-1.1 | The system shall identify sensor failures. | 60ms |
| SR-1.1.2 | The system shall identify when the lidar sensor has failed. | 60ms |
| SR-1.1.3 | The system shall identify when the camera has failed. | 60ms |
| SR-1.2 | The system shall mitigate sensor failures. | 800ms |
| SR-1.2.1 | The system shall detect objects by using a combination of different sensors. | 800ms |
| SR-1.2.1.1 | The system shall detect objects using lidar. | 795ms |
| SR-1.2.1.2 | The system shall detect objects using camera. | 55ms |
| SR-1.2.1.3 | The system shall fuse the objects detected by lidar and camera into a single data source. | 25ms |
| SR-1.2.2 | The system shall use only information from working sensors. | not applicable |
| SR-1.3 | ... | ... |
| ... | ... | ... |

Table 4: Textual safety requirements

guage with modes proposed in [7]. Basically, we use the repetition pattern to formalize assumptions as

$$E_{\mathtt{IN}}\ \texttt{occurs every}\ A(\mathsf{COMP})\texttt{.}$$

and guarantees as

$$\texttt{Reaction(}E_{\mathtt{IN}}\texttt{,}\ E_{\mathtt{OUT}}\texttt{)}\ \texttt{within}\ \Delta(\mathsf{COMP})\texttt{.}$$

or, if COMP does not have inputs,

$$E_{\mathtt{OUT}}\ \texttt{occurs every}\ \Delta(\mathsf{COMP})\texttt{.}$$

where $E_{\mathrm{IN}}$ the component inputs and $E_{\mathrm{OUT}}$ the outputs. For components that do fault detection, the above patterns are extended with mode specification. We assume that a sensor has three different modes, as depicted in Figure 7: `OK`, `transient_fault`, `permanent_fault`. The set of failure modes {`transient_fault`,`permanent_fault`} is abbreviated as `NOT_OK`. For specifying assumptions about failure modes we introduce two failure patterns that can be reduced to patterns from the MTSL with modes. The failure pattern

$$C\ \texttt{fails at most once every}\ P^{-}\ \texttt{with latency}\ I\texttt{.} \tag{1}$$

is equivalent to the conjunction of

- `set(`$C$`,transient_fault) occurs every [`$P^{-}$`,inf[ in mode {`$C$`.transient_fault, `$C$`.OK}.`
- `Reaction(change(`$C$`,transient_fault), set(`$C$`, OK)) within `$I$`.`

The failure pattern

$$\texttt{At most one of \{}C_1\texttt{,}\ C_2\texttt{\} fails permanently.} \tag{2}$$

is equivalent to the conjunction of the following patterns:

| | |
|---|---|
| **Object Detection** | |
| A1 | `Lidar fails at most once every 200ms with latency [10,30] ms.` |
| A2 | `Camera fails at most once every 400ms with latency [10,30] ms.` |
| A3 | `At most one of {Camera, Lidar} fails permanently.` |
| G | `fused_objects occurs every 800ms` |
| **Lidar Grabber** | |
| A | `Lidar fails at most once every 200ms with latency [10,30]ms.` |
| G1 | `lidar_output does not occur in mode lidar.NOT_OK.` |
| G2 | `lidar_output occurs every [10,60] ms in mode lidar.OK.` |
| **Camera Grabber** | |
| A | `Camera fails at most once every 400ms with latency [100,300] ms.` |
| G1 | `frame does not occur in mode camera.NOT_OK.` |
| G2 | `frame occurs every [10,60] ms in mode camera.OK.` |
| **Object Localization via LIDAR** | |
| A | `lidar_output occurs every [15,inf[ ms.` |
| G | `Reaction(lidar_output, object_poses) within [120,795] ms.` |
| **Oject Localization via Camera** | |
| A | `frame occurs every [15,inf[ ms.` |
| G | `Reaction(frame, bounding_boxes) within [20,55] ms.` |
| **Sensor Fusion** | |
| A | `One of {object_poses, bounding_boxes} occurs every 700ms.` |
| G1 | `Reaction(object_poses, fused_objects) within [5,25] ms.` |
| G2 | `Reaction(bounding_boxes, fused_objects) within [5,25] ms.` |

Table 5: Contracts

- `Change(`$C_1$`,OK) does not occur in mode` $C_1$`.permanent_fault.`
- `Change(`$C_2$`,OK) does not occur in mode` $C_2$`.permanent_fault.`
- `Change(`$C_1$`,permanent_fault) does not occur in mode` $C_2$`.permanent_fault.`
- `Change(`$C_2$`,permanent_fault) does not occur in mode` $C_1$`.permanent_fault.`

Using this patterns, the safety concept is formalized per component as in Table 5.

Some of the contracts (those for the grabbers) specify failure-mode dependent behavior. An implementation of these contracts typically relies on some base safety mechanism for the detection of named sensor failures. Because such safety mechanisms have typically a detection coverage of $0 < DC_0 < 1$, the contract implementations have a residual failure rate of at least $\lambda(1 - DC_0)$, with $\lambda$ being the rate of the detected failure. Additionally, the contracts make assumptions about the occurrence of failures, i.e., the minimum distance between transient faults (pattern (1)) and multi-point permanent faults (pattern (2)). Usually, fault models consider those faults to be independent. Hence, a violation of those assumptions cannot be excluded and shall be considered as a residual multi-point fault. In case of pattern (1), this increases the residual failure rate by $\lambda(1 - e^{-\lambda P^-}) \approx \lambda^2 P^-$. Here, $1 - e^{-\lambda P^-} \approx \lambda P^-$ is the portion of transient failures that are followed (or preceded, equivalently) by a second failure within $[0, P^-]$. The approximation is good for low failure rates, i.e., if

$\lambda P^-$ is still close to zero. For pattern (2), the residual failure rate increases accordingly by $\lambda_1 \lambda_2 T$, where $\lambda_1$ and $\lambda_2$ are failure rates of $C_1$ and $C_2$, respectively, and $T$ the time that the system must maintain a safe state after occurrence of the first failure. In the MobSTr use case, this is the time needed for a driver takeover.

# 6 Providing Evidence

The aim of verification and validation activities is to provide an evidence that the developed product finally meets it safety goals. In the following, it is briefly discussed how virtual integration tests and timing checks with the developed contracts can contribute to an evidence for safety goal SG-1 from Table 4.

The argument follows a meet-in-the-middle approach that traverses the system architecture top-down and bottom-up.

Starting top-down with the HARA, the considerations in Sections 2 and 3 need to be approved. Possible ways to do that could involve expert knowledge, field experiments, and simulations. This yields an argument as follows: If the system implements the safety concept sketched in Section 2 and mitigates faults within the FHI, then all necessary avoidance maneuvers will be performed in time. Note that the safety concept described here is still incomplete, as it only considers a single type of avoidance maneuver and only a very limited set of sensor faults. So the argument needs to be repeated for each identified fault and hazard related to the safety goal.

Now, an argument is needed that the hardware and software parts together correctly implement the safety concept. This can be done bottom-up: At the bottom, hardware and software have to be verified against both the functional and timing requirements at the leaves of the requirements tree. Expressing requirements in formal language allows to apply formal verification techniques here. The timing behavior of the software is modeled within the AMALTHEA model and specified partially by the contracts in Table 5. A timing analysis (for example with the RTana model checker [1]) of the AMALTHEA model can verify the timing behavior. Of course evidence is needed that the AMALTHEA model itself is correct. Note that the contracts derived in this work are still incomplete, because only a selected set of components from the chain is specified. This can be seen in Figure 6. In general, the requirement list provided with the MobSTr data set is intentionally incomplete. So, the analysis results based on this work only give part of the argument. However, the rest of the system can be specified and analyzed analogously. Although the contracts also specify a limited amount of functional aspects, they are not sufficient to argue functional correctness of the software, nor is the AMALTHEA model suitable for that.

Finally, from satisfaction of the bottom-level requirements an satisfaction of the safety goal is derived by proving that the requirements (and so the contracts) on each level of the design correctly and completely refine the requirements on the next higher level, up to the safety goal [3]. Provided that the requirements have been completely formalized, this can be done with the help of a virtual integration test. With respect to timing, satisfaction of the FHI can be checked based on the contracts developed here. However, recall that the contract specification in Table 5 is incomplete.

# 7 Conclusion

In this report, time bounds and formal contracts are derived for the MobSTr dataset based on the WATERS challenge solution provided by Krawczyk et al. in [6]. As a consequence, the approach *does not* follow a safety process – rather it is a way to come up with timing requirements for the MobSTr use case. Nevertheless it creates realistic and reasonable requirements. As it has been highlighted several times during this report, the requirements derived are neither complete nor consistent. Both is not a showstopper when using the MobSTr dataset. Missing requirements can be accomplished following the approach in Sections 4 and 5 or following some safety process when supplementing the whole use case. Inconsistencies in the contract set in Section 5 originate from the rough over-approximations made in Section 4 and can serve as a showcase for virtual integration testing (e.g [3, 2]). It is expected that a virtual integration test can find that the LIDAR and camera grabbers not yet satisfy the assumption of the object localization components. Model checkers (at least those aiming completeness) will be challenged by the different failure modes and the large hyper period of more than 800ms.

Future work is to further accomplish the requirement set and to provide a second version in which the known and yet unknown inconsistencies are resolved.

# References


[1] Jan Steffen Becker, Björn Koopmann, and Ingo Stierand. AMALTHEA timing analyses with RTana2sim, 2021. online: `https://panorama-research.org/pdf/aramis2-timing-analysis.pdf`.

[2] Werner Damm, Günter Ehmen, Kim Grüttner, Philipp Ittershagen, Björn Koopmann, Frank Poppen, and Ingo Stierand. Multi-layer time coherency in the development of adas/ad systems: Design approach and tooling. In *Proceedings of the Workshop on Design Automation for CPS and IoT*, pages 20–30, 2019.

[3] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[4] Thomas Frese, Torsten Leonhardt, Denis Hatebur, Isabelle Côté, Hans-Jörg Aryus, and Maritta Heisel. Fault tolerance time interval. In Heike Proff, editor, *Neue Dimensionen der Mobilität: Technische und betriebswirtschaftliche Aspekte*, pages 559–567. Springer Fachmedien Wiesbaden, Wiesbaden, 2020.

[5] ISO. ISO 26262:2018: Road vehicles - Functional safety, 2018.

[6] Lukas Krawczyk, Mahmoud Bazzal, Ram Prasath Govindarajan, and Carsten Wolff. Model-based timing analysis and deployment optimization for heterogeneous multi-core systems using eclipse app4mc. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering*

*Languages and Systems Companion (MODELS-C)*, pages 44–53. IEEE, 2019.

[7] Janis Kröger, Björn Koopmann, Ingo Stierand, Nadra Tabassam, and Martin Fränzle. Handling of operating modes in contract-based timing specifications. 2021. submitted for publication.

[8] Andreas Reschka and Markus Maurer. Conditions for a safe state of automated road vehicles. *it-Information Technology*, 57(4):215–222, 2015.

[9] Jan-Philipp Steghöfer, Björn Koopmann, Jan Steffen Becker, Mikaela Törnlund, Yulla Ibrahim, and Mazen Mohamad. Design decisions in the construction of traceability information models for safe automotive systems. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 185–196. IEEE, 2021.

[10] Jan-Philipp Steghöfer, Björn Koopmann, Jan Steffen Becker, Ingo Stierand, Marc Zeller, Maria Bonner, David Schmelter, and Salome Maro. The mobstr dataset – an exemplar for traceability and model-based safety assessment. In *Proceedings of the 29th IEEE International Requirements Engineering Conference, Notre Dame, South Bend, USA*. IEEE, Aug 2021.

[11] Jan-Philipp Steghöfer, Björn Koopmann, Jan Steffen Becker, Ingo Stierand, Marc Zeller, Maria Bonner, David Schmelter, and Salome Maro. The mobstr dataset: Model-based safety assurance and traceability, Jun 2021.