Dependable and Efficient Cloud-Based Safety-Critical Applications by Example of Automated Valet Parking

Christian Drabek[1](✉), Dhavalkumar Shekhada[1], Gereon Weiss[1], Mario Trapp[1], Tasuku Ishigooka[2], Satoshi Otsuka[2], and Mariko Mizuochi[3]

[1] Fraunhofer IKS, Munich, Germany
{christian.drabek,dhavalkumar.shekhada,gereon.weiss, mario.trapp}@iks.fraunhofer.de

[2] Research and Development Group, Hitachi Ltd., Ibaraki, Japan
{tasuku.ishigoka.kc,satoshi.otsuka.hk}@hitachi.com

[3] Hitachi Europe GmbH, Schwaig, Germany
mariko.mizuochi@hitachi-eu.com

Abstract. Future embedded systems and services will be seamlessly connected and will interact on all levels with the infrastructure and cloud. For safety-critical applications this means that it is not sufficient to ensure dependability in a single embedded system, but it is necessary to cover the complete service chain including all involved embedded systems as well as involved services running in the edge or the cloud. However, for the development of such Cyber-Physical Systems-of-Systems (CPSoS) engineers must consider all kinds of dependability requirements. For example, it is not an option to ensure safety by impeding reliability or availability requirements. In fact, it is the engineers' task to optimize the CPSoS' performance without violating any safety goals.

In this paper, we identify the main challenges of developing CPSoS based on several industrial use cases and present our novel approach for designing cloud-based safety-critical applications with optimized performance by the example of an automated valet parking system. The evaluation shows that our monitoring and recovery solution ensures a superior performance in comparison to current methods, while meeting the system's safety demands in case of connectivity-related faults.

Keywords: Cyber-Physical Systems of Systems · Automated recovery · Monitoring · Fail-operational · Graceful degradation · Self-awareness

1 Introduction

Nowadays, Cyber-Physical Systems (CPS) become more and more flexibly interconnected with other services as well as with local and remote infrastructure.

A. L. Martins et al. (Eds.): INTSYS 2020, LNICST 364, pp. 90–109, 2021.
https://doi.org/10.1007/978-3-030-71454-3_6

For instance, off-loading software services of embedded systems into cloud computing environments bears the potential to offer more features and to bypass resource restrictions of local CPS. This will further become essential, as upcoming autonomous machines with artificial intelligence capabilities have an insatiable need for computational power [25]. Moreover, cloud infrastructure facilitates better maintainability and scalability, e.g., easier service upgrades and data aggregation of multiple machines. Such interconnected CPS can be seen as a *Cyber-Physical System-of-Systems (CPSoS)* [6]. Examples range from intelligent networked vehicles over industrial control systems to collaborating robots. However, in many application scenarios these machines also perform safety-critical functions. In consequence, this means that it is not sufficient anymore to ensure the safety of single systems. Instead, it is necessary to ensure the safety of the complete CPSoS including all systems and services. As an additional challenge, a CPSoS is not static, but integrates and removes systems and services from different vendors dynamically at runtime. Moreover, it is unavoidable to use non-safe components such as the communication infrastructure or cloud servers. This requires the CPSoS to be resilient. In case a service temporarily fails or is not available at all, the CPSoS should still work safely. To this end, the CPS might continue in a degraded mode and resume with full capacity, when the cloud service becomes available again.

In this paper, we examine the challenges of developing safe CPSoS considering various different practical application scenarios. Focusing on optimizing efficiency without violating safety, we introduce our design approach that holistically covers safety and other dependability requirements. We illustrate the application of our approach using an industrial case study realizing an *Automated Valet Parking (AVP)* service. Eventually, we evaluate our approach using a simulation-based prototype of the AVP. In this evaluation, we focus on service availability due to communication problems and show the advantages of the resulting dynamic management using dedicated monitoring and recovery solutions over the current approaches only considering the local assurance or optimization of single systems. We compare the performance of the approaches by measuring the average speed of the vehicles in the parking area. Summarizing, the main contribution of the paper is the introduction and the practical evaluation of a holistic design approach for safety-critical CPSoS.

The remainder of the paper is outlined as follows. Section 2 introduces several industrial use cases of CPSoS and their high-level requirements, which we use to derive the general challenges posed by such systems. Section 3 discusses related work on optimizing and analyzing CPSoS. In Sect. 4, we present our approach for the design of safe and efficient CPSoS by example of an AVP system. The evaluation described in Sect. 5 compares the performance of the designed CPSoS with a pure local variant, before we conclude the paper in Sect. 6.

2 Challenges and Use Cases of CPSoS

CPSoS comprise several subsystems and components. These in turn may also include safety-critical applications. In comparison to traditional CPS which act on their own (e.g., control units in a car or industrial control systems), CPSoS combine them into highly interconnected large systems. Therewith, the independence of these autonomous units is abolished and their correct function is considered on system level. For instance, machine learning-based systems can leverage cloud infrastructure to compute complex scenarios or remote operation of a vehicle requires the whole control chain to function correctly. For identifying the main challenges of upcoming CPSoS, exemplary industrial use cases have been selected and are presented in the following.

2.1 Industrial Use Cases of CPSoS

The following use cases introduce applications of CPSoS and provide examples for industrial scenarios. For brevity, only selected challenges are listed for each use case.

Vehicle Remote Operation (VRO) (cf. Fig. 1, left) is a use case that considers remotely driving a vehicle to a safe place with the situation-wise adequate speed, e.g., in case of a failure or unavailability of a driver. This requires *real-time execution and sensor data upload* with low latency and jitter for control stability, and local fallback in the vehicle to guarantee minimum safety, e.g., if the link to the remote service is lost suddenly.

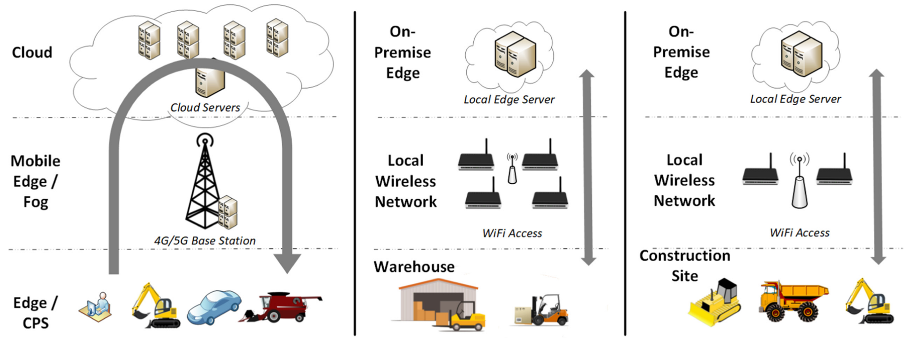

Fig. 1. Use cases: Vehicle Remote Operation (left), Warehouse Management (middle) and Construction Site Management (right).

The use case *Warehouse Management* (WM) (cf. Fig. 1, middle) describes moving shipments within an automated warehouse. Tasks are sent to automated forklifts which operate at low vehicle speeds in the warehouse. Scalability needs to address sharing of sensor information between vehicles and infrastructure. Moreover, efficiency of the system, e.g., the ability to quickly move goods when needed, is important and stand-still should be avoided.

With the use case *Construction Site Management* (CSM) (cf. Fig. 1, right) individual construction machine operation at a restricted construction site is considered. The operation of local machinery, e.g., excavators or bulldozers, is automated based on the fusion of each machine's sensor data. As such, challenges arise from data uploads for continuous training and logging, as well as maintenance to update local maps, tasks and software in general.

As an example of a CPSoS in which domain boundaries are vanishing, the use case of *Cross-Domain Unknown Services* (CDUS) (cf. Fig. 2, left) considers cross-domain service orchestration. The specific use case addresses the usage of services of other domains after successful authentication. We refer to them as unknown services, as they are developed independently without considering each other and information must be translated between domains. For example, an automated valet parking system cannot directly process information from construction sites in the parking area, but will benefit from knowing which part of the area is not available. As unknown services need to interact, establishing interoperability between the involved systems without compromising security or safety is a prominent challenge.

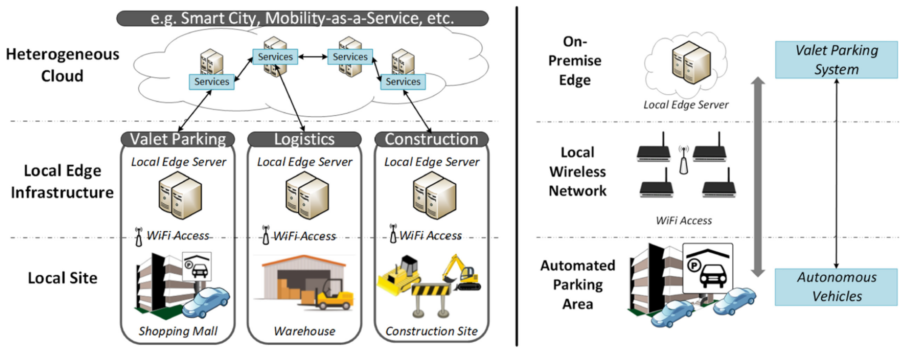

Fig. 2. Use cases: Cross-Domain Unknown Services (left) and Automated Valet Parking (right).

The *Automated Valet Parking (AVP)* use case (cf. Fig. 2, right) includes a management of automated vehicles in a restricted parking area. Mainly, the *Parking Area Management* assigns the best free parking space to arriving vehicles and allows a safe remote navigation by sending trajectories to the vehicles moving at low speeds. This is carried out by monitoring and controlling blocks of the parking area and granting driving permissions to vehicles. As for all CPSoS, the comprising subsystems are usually developed independently, but need to interface with each other. Unless the parking area is limited to a specific group of vehicle manufacturers, AVP needs to be an open system. Therefore, standards are required to ensure interoperability. Moreover, to cover cases where no or only limited communication is possible, a safe fallback is needed. This use case is studied and evaluated within this work and therefore presented in more detail in Sect. 4.

While only selected challenges have been introduced when presenting the above application use cases to provide examples, many challenges apply to several of the use cases. To summarize the challenges, Table 1 presents an overview of the identified challenges for each of the above application use cases.

Table 1. Challenges of industrial use cases of CPSoS

Challenge	VRO	WM	CSM	CDUS	AVP
Safety to ensure no collision with obstacles	✓	✓	✓	✓	✓
Efficiency to avoid stand-still		✓	✓		✓
Real-time execution with low latency	✓	✓	✓	✓	✓
Real-time sensor data upload	✓				
Scalability of sharing real-time sensor information		✓	✓		✓
Interoperability between different manufacturers	✓	✓	✓	✓	✓
Connectivity for control data	✓	✓	✓	✓	✓
Security against hijacking	✓	✓	✓	✓	✓
Data uploading for continuous training and logging		✓	✓		
Maintenance w.r.t. map, task and software updates		✓	✓		
Self-awareness to monitor and recover remote control	✓	✓	✓	✓	✓

2.2 Common CPSoS Challenges

Based on our analysis of the examined use cases, we derived the following common challenges of upcoming CPSoS in such safety-critical environments:

– **Safety and efficiency:** Faults and threats must be identified and managed dynamically, i.e., by providing suitable countermeasures.
– **Real-time:** Planning of flexible and reliable end-to-end architectures, including dynamic allocation of distributed resources.
– **Connectivity & interoperability:** Interfaces definitions & standardization for cross-domain inter-operation is needed.
– **Security:** Identification and mitigation of additional threats created by widening the system boundaries.
– **Data-uploads & maintenance:** Collect training data and distribute updates of the various subsystems without interfering with normal operation.

These challenges share the need to provide an in-time and complete overview of the present CPSoS state. For this, run-time monitoring mechanisms at different levels of a CPSoS are indispensable. By this, a so-called *self-awareness* at run-time can be achieved, which constitutes the basis for taking measures to keep the system in a safe and performant state. The goal of managing such a CPSoS is to increase availability and meet reliability, particularly considering defective resources and the integration of unknown services. In order to achieve resilient behavior of a CPSoS, in general diverse changes of different nature, prospect, and timing [13] must be considered. By embedding self-adaptation as fault-tolerance mechanisms into CPSoS it is capable to optimize the performance, while meeting dependability requirements. We have selected the AVP use case for detailed study and evaluation, because of the availability of detailed specifications and its requirements for safety and optimized performance.

3 Related Work

When designing CPSoS [26], ensuring safety and optimizing performance are two of the main challenges. Previous approaches related to our work are briefly introduced in the following.

For optimizing performance of CPSoS, diverse approaches and tools are already available [4]. Optimization tools like FogNetSim++ [17], iFogSim [7], EdgeCloudSim [24] focus on the simulation of edge-, fog-, cloud-systems and support identifying optimal parameters or deployments for specific scenarios. In addition, network simulations like ns-3 [19] and OMNET++ [27] can provide estimates on the performance of specific network infrastructures for a CPSoS. However, these tools do not consider the safety of the system.

Approaches targeting to provide reliability develop patterns to identify host or network failures, or in general, violations of safety properties [23] and apply specific recovery strategies to provide fault tolerance in distributed systems [14,15]. Others aim to establish resilience for stateful IoT applications [16] by enabling the recovery of their states. In case a safety-critical function cannot be easily stopped or replaced, graceful degradation [9], which reduces functionality in order to retain safety properties even under the presence of certain faults, has proven useful. If faults and desired reconfigurations are known beforehand, the availability in fail-operational automotive systems can be planned and verified already at design time [20]. While general safety requirements and distribution of functions for automated valet parking have been analyzed [22], we focus on the presence of an unreliable connection between cloud and vehicle. As already discussed by other authors [12], we pursue a state-based monitoring approach for this challenge in safety-critical CPSoS.

In comparison, safety-related approaches often focus to ensure the systems' safety properties, not its performance or availability. An alternative example considering performance optimization is a safety envelope used within autonomous systems [11], which allows a system to optimize its performance within this operating envelope by monitoring violations. Therefore, a tighter integration of methods for safety analysis into other software-engineering disciplines could benefit both sides [3,5]. Our approach leverages such an integration in an iterative process of identifying safety and performance faults and, thereby, allows a novel improvement of a design for both mitigation and optimization.

4 Design of Safe and Efficient CPSoS

As previously motivated, designing safety-critical CPSoS that are safe and yet efficient is a challenging task. The design of such systems is a multivariate optimization problem in order to always provide the best possible performance, while fulfilling safety requirements at any time. While a holistic approach is required to ensure no safety requirements are missed, CPSoS quickly become large and complex. Therefore, this complexity needs to be broken down during design. In the following, we describe our design approach for such systems (cf. Fig. 3).

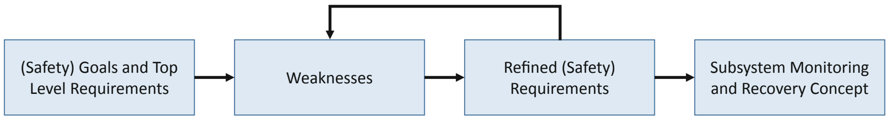
(Safety) Goals and Top Level Requirements
Weaknesses
Refined (Safety) Requirements
Subsystem Monitoring and Recovery Concept

Fig. 3. Overview of our design approach for efficient safety-critical CPSoS.

An initial requirements analysis in the first step defines the top-level requirements of the CPSoS and utilizes safety analysis methods to identify the main safety goals in terms of additional requirements. Moreover, it manifests an architecture draft that enables the analysis of critical interactions between subsystems. In the next step, weaknesses of the system and its subsystems are iteratively identified and mitigated by refining the requirements as well as documenting assumptions and verification methods. Finally, self-awareness of subsystems is established by deriving monitors from the requirements that identify important changes in the current context and thereby trigger the planned recovery methods.

We use the example of an AVP system with a centralized, cloud-based *Valet Parking System (VPS)* that guides *autonomous vehicles (AV)* to demonstrate our approach. In the automotive domain, safe design against hardware failures is performed according to the functional safety standard ISO 26262 [1] and safety design against performance limitations is performed according to ISO/PAS 21448 [2].

4.1 Top-Level Requirements

The top-level requirements are utilized to sketch the desired features of the CPSoS. For brevity, we only present the headline of each requirement, which provides a short summary of its description. Nevertheless, this level of detail should be sufficient to follow the general idea of the design approach. The definition of top-level requirements can be quite coarse. They will be refined, broken down and assigned to subsystems in the next phase. The main purpose of the top-level requirements is to describe the important functional and non-functional goals of the system.

For the AVP use case, we consider the following functional requirements:

FR-1	Management of parking space (occupied or empty).
FR-2	Connection between AV and VPS when AV enters the service area.
FR-3	Find suitable, empty parking space.
FR-4	Calculate trajectory (based on vehicle properties) and send to AV.

FR-5	Move AV to parking space by referring to trajectory of FR-4.
FR-6	Terminate VPS for AV when it arrives at the target space.

With a preliminary hazard and risk analysis (HARA), the following dependability requirements have been identified. The first two requirements are related to the system's safety, while the remaining improve reliability and availability.

DR-1	Local emergency stop.
DR-2	Remote emergency stop.
DR-3	Detect unreachable parking place and request another.
DR-4	Communication diversity in physical architecture.
DR-5	Collision prevention by free space detection and block control.
DR-6	Wrong waypoint and parking place detection.

Based on these top-level requirements, a possible architecture design of the AVP system has been developed, as shown in Fig. 4. VPS and AV do not know each other's internal architecture, as they will be developed by different vendors. However, the systems need to interface with each other. The identification and description of necessary interfaces are facilitated, if exemplary architectures are assumed. After receiving a parking request, the parking area management identifies a suitable destination and planning generates a trajectory. *Traffic Monitor (TM)* passes trajectories and permissions to AVs. Trajectories describe waypoints to the destination, e.g., a parking place, and permissions allow the AV to proceed to a certain waypoint. *Trajectory Following Control (TFC)* in the AV is then responsible for following this trajectory as permitted. The AV's sensors are utilized to trigger *Passed Notifications (PN)* sent to TM. In turn, TM updates its database of permissions and locations. PN will also trigger a new parking request if the destination is not reachable.

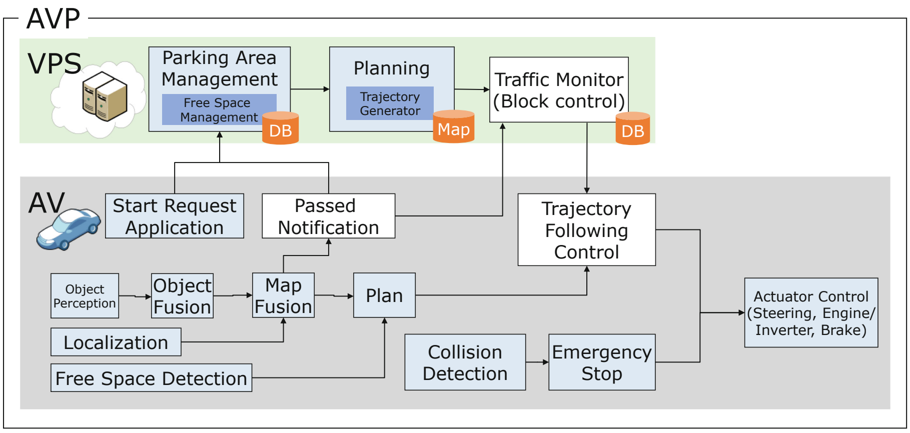

Fig. 4. Exemplary architecture design for an AVP based on requirements.

4.2 Iterative Weakness-Driven Design Refinement

In the next phase of our approach, we use an iterative weakness-driven design refinement to uncover the system's weaknesses and to integrate countermeasures along the refinement. In order to denote not only safety-related issues, i.e., faults, we define the term *weakness* as *any deviation from the system's intended function.* This could be with respect to a potential safety hazard or failure [1], a security threat or vulnerability [21], or a breach of performance thresholds. The general intent is to identify potential weaknesses of the CPSoS and determine what conditions are necessary to handle them on subsystem level.

The inputs to this phase, top-level requirements and a draft of the system architecture, have been presented in the previous subsection. Potential weaknesses are systematically identified using a HAZOP-oriented process [8], i.e., by applying a set of guide-words to the requirement descriptions. Examples for guide words are *not*, *more*, *less*, *early* and *late.* To scrutinize the consistency of the requirements, the following questions are used in addition to common HAZOP guide-words:

– Internal: How can the subject itself fail to fulfill the requirement?
– External: What external influences can cause the subject to fail (the intend of) the requirement?
– Integrity: Are there any terms, definitions or values used by the requirement that can impact its intend, if chosen incorrectly?

Each of the identified weaknesses needs to be resolved by verification, assumptions, or other requirements. A *verification* describes a method to verify why a weakness will not occur or is mitigated sufficiently. An *assumption* is a statement describing a property that is assumed to be valid. Therefore, assumptions are formulated to document parts of the system that are expected to work or resolved by the detailed design of an individual subsystem. For example, we assume a correct implementation of requirements but will not specify how this is achieved. Other requirements that resolve weaknesses either refine vulnerable requirements to detail how the failure is avoided, or impose additional requirements to mitigate cause or effect of certain faults. New requirements may introduce new weaknesses that are identified and mitigated in further iterations. This is continued until all weaknesses are resolved. The resulting requirements describe the roles of subsystems in the CPSoS and their interfaces. We highly recommend to record bidirectional relations between requirements, weaknesses and resolutions. Thereby, validation of an implementation is facilitated, when the reason for a requirement can be traced easily. In the following, we will detail our approach by example of the AVP system.

AVP is a CPSoS comprising independently developed, resilient systems: VPS and several AVs. They each may provide their own mitigation strategy in case the connection is lost. Yet, their cooperation needs to be safe and efficient at any time. Without the loss of generality, we select *loss of connection* in the following as exemplary fault to outline our approach and its related weaknesses, as they cannot be handled by a single system of the CPSoS individually. As this is a high-level analysis targeting cloud-based autonomous CPS, functional safety aspects of single systems are less of concern, which is documented using assumptions. The critical parts, where the two CPS types interface with each other, can be identified already in the architecture draft. An extract of the analysis with their main requirements and the identified weaknesses related to the loss of connection from the first iteration are given in Table 2. The weaknesses are also assigned to categories for indicating the kind of deviation, e.g., reliability for safety-related faults or performance in case they merely impact the efficiency.

Table 2. Extract of requirements and identified weaknesses of TM, TFC and PN before the first refinement.

System	ID	Cat.	Top-Level Req.	Short summary
TM	R4	Gen.	FR-5, DR-2, DR-5, DR-6	TM shall monitor and control access to blocks based on trajectories calculated by Planning
	V4a	Rel.	DR-2, DR-5, DR-6	TM can fail to monitor cars
	V4b	Rel.	DR-2, DR-5, DR-6	TM can fail to control manually driven cars
	V4c	Rel.	DR-2, DR-5, DR-6	TM can fail to control not connected cars
	V4d	Perf.	FR-5	TM can be slow to provide access to blocks
	V4k	Perf.	FR-5	TM can receive passed notifications late or never
TFC	R8	Gen.	FR-5, DR-2, DR-5, DR-6	TFC shall calculate actuator controls based on the trajectory and permission provided by VPS and the information from perception chain
	V8c	Rel	FR-5, DR-2, DR-5, DR-6	TFC can fail to receive or process permissions from VPS
PN	R18	Gen.	FR-5, FR-6, DR-6	PN shall signal right after a block was cleared by the vehicle
	V18c	Perf.	FR-5, FR-6	PN can send the notification too late

The first iteration addresses the fault of TFC not receiving permissions. Therefore, a new requirement requires TFC to stop the vehicle if no permission is provided in time. In addition, this prohibits rogue movement of AVs in the parking area. Care must be taken that an AV is always able to stop in time and will not move without permission, i.e., by slowing down at the end of the permitted trajectory. We assume that there are no manual operated vehicles in the parking area and thus, TM is in sole control of all movement. This addition makes the system safe, and we will refer to this as *Mitigation Variant 1 (MV1)* in the evaluation section. However, additional performance weaknesses remain.

AVs can get stuck, if they have neither permission nor connection. Hence, after a specified timeout, its own TFC shall permit a trajectory based on the AV's sensors. However, from the perspective of TM, this is a rogue AV. Moreover, insufficient monitoring of parking area by the TM is now a safety issue, as it may provide a conflicting permission to another AV, if it does not sufficiently locate the rogue AV. Further, if VPS never receives a pass notification, it is forced to reserve the permitted space infinitely. Therefore, a requirement for VPS to track vehicles using infrastructure sensors is added and the architecture is updated as in Fig. 5. We assume sensors to work as expected. While VPS can now notice rogue AVs, TM also needs to validate and possibly adjust already provided permissions. To mitigate the weakness of AVs relying on outdated information, permissions sent from TM have a limited lifetime. Therefore, an AV must be capable to stop in time before a permission becomes invalid. Depending on the selection of timeouts, it may directly switch to permissions based on its own sensors. Moreover, to facilitate predictability of an AV's movement, VPS should send a complete trajectory to AVs initially, which is followed even by a disconnected AV. We define this as *Mitigation Variant 2 (MV2)*. More weaknesses could be identified, e.g., that a trajectory may get blocked by an obstacle, but are beyond the scope of this example.

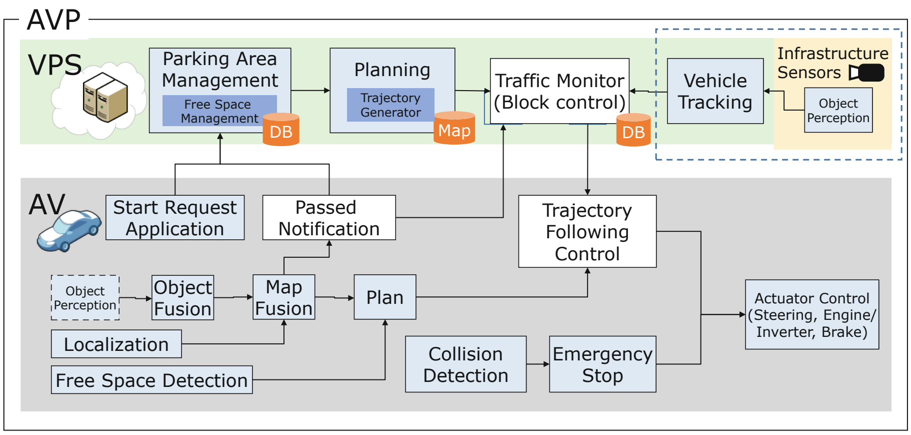

Fig. 5. Exemplary architecture design extended with infrastructure sensors.

4.3 Monitoring and Recovery Concept

With the next step of our approach, we map the identified requirements to a monitoring and recovery concept for each subsystem. At this stage, the requirements already capture the critical interaction patterns between the individual systems, so that the remainder of each system can be developed independently. However, to make it more explicit what a subsystem must monitor and what reactions are required for coordinated interactions, this is formulated as monitoring and recovery concepts. Transforming the requirements of the critical interactions into a monitoring and recovery concept has the additional benefit that this enables an abstract simulation of these interactions without full implementations of each subsystem. This helps to validate the concept early. In summary, the monitor identifies current context states and the recovery will adjust the operation to remain in a safe and efficient overall system state. The monitor aggregates all information that must be observed in order to establish a *self-awareness* of critical changes in the environment or within the system itself. The recovery will change operation parameters or start and stop services as needed to reach the state inferred by the weakness analysis. The behavior needs to be managed dynamically based on current context. However, this can still be planned at design time, if the involved capabilities are known. Thereby, existing concepts for graceful degradation in the automotive domain [9] can be used to facilitate the integration of recovery in alignment with ISO 26262 and ISO/PAS 21448. In the following, we explain how the monitoring and recovery concepts for the AVP example were derived.

In MV1, an AV stops and waits if there is no permission update, e.g., because of a lost connection. As the scope of this paper is limited to such faults, an AV will never act outside the expectations of VPS and there is no specific need for a monitor or recovery on VPS side from a safety perspective. In contrast, AVs must ensure they are always able to stop in time. While a monitor could be designed to directly observe the state of the connection, just a loss-of-connection is no direct safety hazard to AVP. A hazardous situation would arise only if the AV overshoots its permitted trajectory. In this case, it is much more effective for the monitor to observe AV's ability to stop before this occurs. Therefore, the monitor must keep track of the remaining distance to the end of its permitted trajectory (D_p), get an estimate of the current brake distance (D_b) and compare the two. If the difference is less than a given minimum distance (D_m), recovery needs to force the AV to decelerate. This is illustrated in Fig. 6. The minimum distance must be larger than the distance traveled by the AV between two checks and can be calculated based on the rate at which this check is performed and the (maximum) speed of the vehicle in the parking area. The monitor will thereby signal the AV if it approaches the boundary of its permission and the recovery will force the AV to decelerate. If the permission is extended again, the AV may proceed normally again. Actually, the resulting monitor is independent of a lost connection and also triggers if VPS updates were received but the AV needs to yield right of way at an intersection.

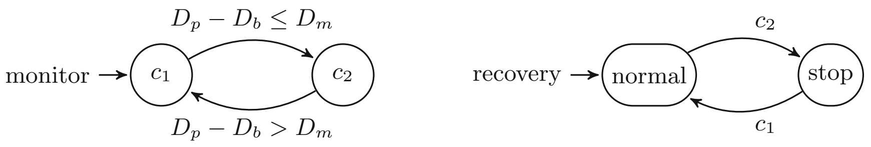

Fig. 6. Monitoring and recovery concept for AV in mitigation variant 1.

In MV2, an AV also mitigates the weakness that it may get stuck at some place with no connection. In this case, other AVs may be moving without permission from the cloud. An AV must be able to receive updates of permissions issued by VPS. Otherwise, they might be outdated and not reliable anymore. The critical property for this is the time since the last update (T_u), which is accompanied by two thresholds. The first specifies when the remote permissions should be considered outdated (T_o). The other indicates the time when local permissions may be issued (T_l). The chosen thresholds may allow for an overlap of remote and local permission ($T_o \geq T_l$), or an AV will have to stop before it can switch to local sensors ($T_o < T_l$). While the monitor for MV1 only checks the distance, for MV2 an AV's monitor also has to consider the time needed to slow down. Therefore, T_b specifies the time needed to slow down to stop or to the speed currently permitted by local sensors. The overall monitoring and recovery concept for MV2 is shown in Fig. 7. This consists of three separate monitors, where recovery will react to their combined state. T_m is the margin, i.e., the amount of time reserved for cycle times and processing.

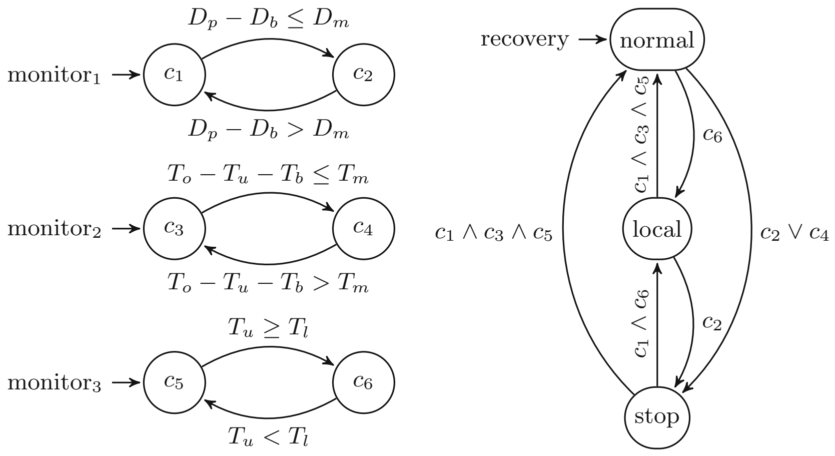

Fig. 7. Monitoring and recovery concept for AV in mitigation variant 2.

If AVs are moving on their own permissions, VPS is required to monitor for this and needs to recover conflicting permissions. The VPS' responsibility is to provide such updates to connected AVs. Therefore, VPS must track the AVs in the parking area. Locations provided by AVs can and will likely be affected by similar connection losses as provided permissions. Therefore, secondary means are required, e.g., using infrastructure sensors to monitor AVs. Moreover, the VPS must check if the rogue AV interferes with existing permissions and must update those accordingly. This works like a remote emergency brake, but allows for a more graceful deceleration if updates can be provided early.

As MV2 utilizes two sources for permissions, the concept must ensure that they cooperate well. This is achieved by putting strict requirements on when an AV may provide its own permission, so that the VPS will always have sufficient time to adapt permissions for other AVs or that the permission of other (temporarily) disconnected AVs will expire. A critical situation could arise, if an AV moves without permission of VPS into space that is reserved for another car. Thus, an AV may only move into space it has no permission for, if it ensures (using its own sensors) that no other car will reach the space in time $T_i + T_o + T_m$, where T_i is the additional time needed by infrastructure sensors to identify the rogue AV's intent and by VPS to update permissions.

A longer permission lifetime (T_o) improves performance, as this allows bridging longer gaps in permission updates. However, any not connected AV must be able to predict its clearance for at least this duration. A shorter permission lifetime will lessen this need, but also limits the maximum speed of AVs, as they must be able to safely switch to local permissions or stop by the end. Tuning of these parameters is beyond the scope of this paper and would require including details about capabilities of the AVs' sensors.

5 Evaluation

We evaluate the effectiveness of the proposed concept in simulations of the AVP use case. While we also verify the concept's safety by checking no collisions occur, the primary goal is evaluating the potential benefits in efficiency by using an optimized and safe cloud-based control despite unreliable connections between VPS and AVs. Therefore, we compare the presented cloud-based mitigation variants with scenarios where AVs use only their own sensors to calculate permissions. We neglected further optimizations of the control algorithms, e.g., we are using randomly assigned parking places and the shortest route instead of carefully selected waypoints that would allow cars to avoid each other. We believe scenarios utilizing the cloud are likely to benefit more from them, as decisions can be based on more information.

5.1 Evaluation Setup

An overview of the evaluation setup is shown in Fig. 8. The evaluation uses an abstract representation of the AVP system to simulate scenarios. The Robot Operating System (ROS) [18] is used in its second version, ROS2, as middleware to connect cloud node and car nodes. The parking area is divided into blocks that are controlled by the VPS cloud node. Every 300 ms, the VPS will broadcast updated permissions to all registered AVs, i.e., up to which block they are allowed to proceed. In turn, AVs send their updated location to VPS with a rate of 200 ms after moving based on the permission and used mitigation variant. ROS2's simulation time was used with 100 ms steps and messages could only be processed by the receiver in the next step.

There is a new car node created for each AV that enters the parking area. It starts at one of the designated entry points, registers with the VPS and requests to park. The VPS selects a parking place and calculates waypoints using A* algorithm and the AV receives this as a list of blocks. The movements, actions and permissions of AVs can be visualized using RViz [10] for further inspection (cf. Fig. 9).

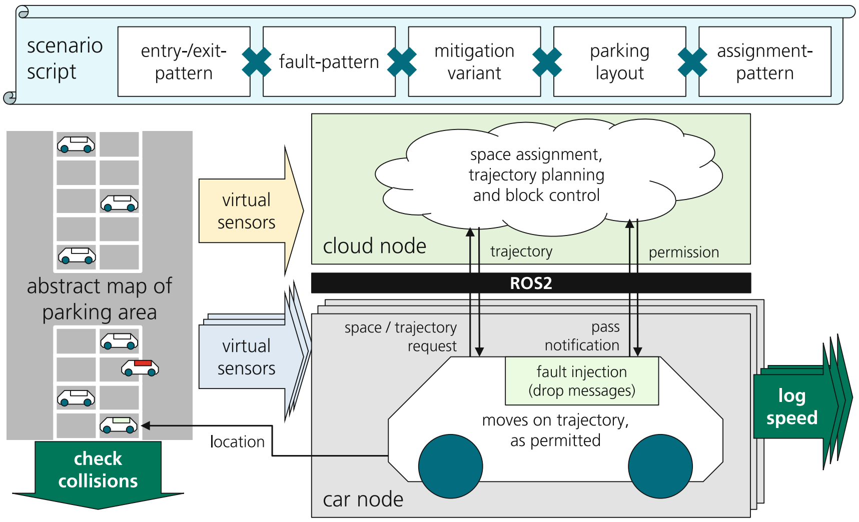

Fig. 8. Overview of the simulation setup used for evaluation.

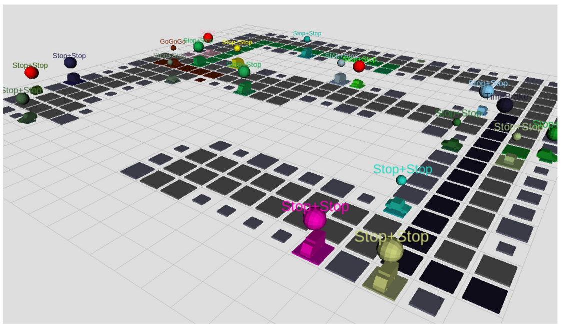

Fig. 9. Screenshot of simulation visualization. VPS permissions are represented by blocks tainted in an AV's color. The size of orbs above AVs correlates to the duration since the last update from VPS, red color indicates expired VPS permissions. The AV's current action is shown as text. (Color figure online)

Each scenario has a duration of 200 s and comprises modular scripts for which all sensible combinations were run. Pattern modules define random distributions that would allow for unlimited variants based on their parameters. To get comparable results and reduce noise, sets with entry-/exit-patterns and fault-patterns were generated beforehand and then used in respective combinations. AVs enter during the first 135 s with an average inter-arrival delay of 2, 4, or 20 s. They start to exit between 30 to 60 s later. This is used to model a busy parking area, while also imitating realistic parking behavior. Failures of connections have been injected based on either a permanently jammed location and radius, e.g., to simulate a failed access point, or time-based patterns per AV. These were mitigated by MV1 and MV2.

When MV2 is in its *local* operating state (cf. Fig. 7), it uses local sensors to generate its permissions. To focus the simulation on evaluating the effects of loss of connectivity and eliminate the choices of sensor types and perception algorithms, it was assumed that local sensors will safely grant permissions up to and including the next block the AV has not yet entered. In the simulation, local permission is granted if no other vehicle can enter the block within T_o, i.e., the time until permissions are outdated. In the evaluation, T_o and T_l were fixed to 3 s. A block that can be reached from more than one other block, i.e., intersections, may only be permitted if the AV has completely entered the adjacent block. Similarly, two offline design variants are included in the evaluation for comparison: *Local Sensors (LS)* is MV2 with the time since the last update (T_u) set to infinity and, therefore, will never use the *normal* operating state. As a result, AVs using LS still receive waypoints from the VPS but ignore received permissions. To see the possible effects of improved local sensors, an additional block may be permitted in *Local Sensors 2 (LS2)*.

Scenarios can take an amorphous layout that is based on a real parking area, or square grid parking layouts with a side-length of 11 or 21 blocks. They can be seen in Fig. 10. Finally, the random seed for the parking place assignment completes the scenario.

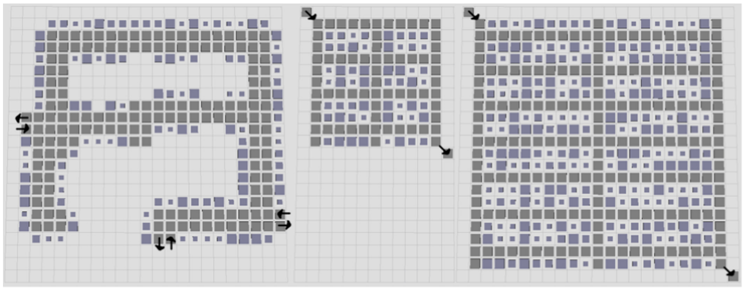

Fig. 10. The parking area layouts used by scenarios. Blue blocks are parking places and sized according to the suitable vehicle size. Arrows indicate entry and exit points. (Color figure online)

As metric for the efficiency of the system in each scenario, the *average speed* of AVs is calculated based on the duration from receiving waypoints to reaching the destination and length of this path. To verify the safety of the proposed concept, a check for (near) collisions of AVs has been implemented that reports if two AVs occupy the same block.

5.2 Evaluation Results

Figure 11 presents an overview of the average speed in meters per second, observed in the presence of various faults, more details can be derived by Table 3. The column headings name the mitigation variant and fault pattern. In **jam**, a third of the parking area around an intersection is permanently without connection. A repeated pattern of a functioning connection for around X seconds followed by a disruption for Y seconds is designated with **c**X**d**Y. No faults occurred in **none**. LS and LS2 do not consider permissions issued by VPS and are thus unaffected by connection faults. Subset cross-validation among 44 random seeds for space assignments showed no significant influence on the average speed with all other factors equal. Therefore, we present results averaged across all random seeds.

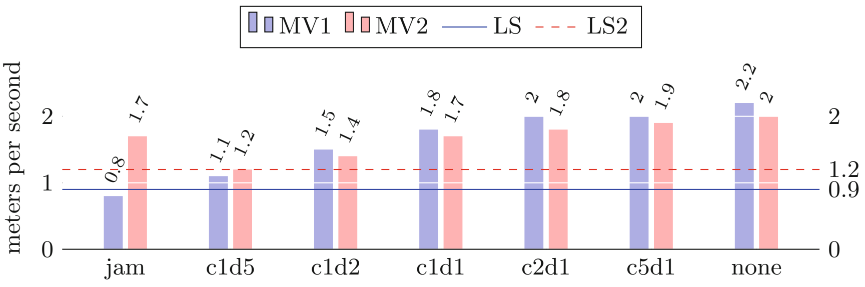

Fig. 11. Average speed of the variants for different fault patterns.

Table 3. Average speed of AVs in meters per second for selected scenarios.

		LS	LS2	MV1							MV2						
Layout	Entry	*	*	jam	c1d5	c1d2	c1d1	c2d1	c5d1	none	jam	c1d5	c1d2	c1d1	c2d1	c5d1	none
Grid11	2s	0.8	1.1	0.4	0.4	0.6	0.9	1.2	1.2	1.5	1	0.6	0.6	0.8	1	1	1.2
	4s	1	1.3	0.6	1.2	1.8	1.9	2	2.1	2.1	1.9	1.4	1.5	1.9	2	2.1	2.1
	20s	1.3	1.6	1.1	1.8	2.2	2.2	2.2	2.3	2.3	2.2	1.9	2.1	2.2	2.2	2.3	2.3
Grid21	2s	0.9	1.2	0.7	0.6	0.8	1.3	1.7	1.8	2	1.4	0.8	0.7	1.1	1.5	1.5	1.7
	4s	1.1	1.4	1.1	1.6	2.4	2.6	2.6	2.7	2.8	2.4	1.7	1.9	2.5	2.5	2.6	2.8
	20s	1.4	1.7	1.7	2.5	2.9	2.9	2.9	3	3	2.8	2.2	2.7	2.9	2.9	3	3
real	2s	0.7	1	0.9	0.9	1.6	1.9	2	2.1	2.2	1.6	1.1	1.3	1.6	1.8	1.8	2.1
	4s	1	1.3	1	1.7	2.2	2.3	2.3	2.4	2.5	2.1	1.8	2.1	2.3	2.3	2.4	2.5
	20s	1.2	1.5	1.7	2.1	2.4	2.5	2.5	2.6	2.6	2.3	2	2.3	2.5	2.5	2.6	2.6
*		0.9	1.2	0.8	1.1	1.5	1.8	2	2	2.2	1.7	1.2	1.4	1.7	1.8	1.9	2

It can be seen that doubling the distance of local permissions, i.e., the change from LS to LS2, yields only a small improvement in speed. The 2 s entry interval easily leads to congestion, especially in small parking areas and if VPS cannot provide frequent updates to AVs. This is visible by the overall low speed in Table 3 for layout Grid11 with 2 s entry time. In all other cases, both MVs about double the speed compared to LS and are only slightly affected by the worse connection. MV1 performs minimally better than MV2 with good connection, as it does not need to account for AVs moving without permission.

However, with longer connection losses, MV1 performs worse and even blocks the parking area. The fault *jam* will bring MV1 to a halt and need manual recovery, as AVs cannot get new permissions. For offline AVs, the VPS is not able to revoke permissions as they never expire. When using MV2, this is no issue and speed is only mildly affected even by permanent jams. AVs that do not manage to pass through the jam completely based on VPS permissions, can use their local sensors to escape it. A VPS that can detect and react to such possible rogue movement using infrastructure sensors, allows safety to remain ensured, when combined with permission lifetimes and rules that define if a vehicle may permit movement to itself.

In this simulation, the cloud based approaches could double the speed compared to using only local sensors, which will provide shorter wait times for users of such systems and help to avoid bottlenecks during rush hours by proving a higher throughput. Such a safe optimization could only be achieved by integrating the safety-analysis into the design process of the system.

The example analysis and the abstract simulation used for evaluation are focused on faults caused by connection problems. Real systems need to prepare for all kinds of faults and threats, e.g., uncertain sensors, failing actuators or malicious users. Nevertheless, they can be included in further iterations of the presented process and appropriate mitigation strategies can be developed. In overall, the results show that our approach improves efficiency in terms of average speed without compromising the CPSoS safety.

6 Conclusion

The advent of CPSoS brings along major advantages with respect to potential service-based applications but also bears great challenges with respect to meeting dependability requirements of safety-critical functions. By the examples of various industrial CPSoS use cases we could identify several common challenges. With the selected case study of an automated valet parking system, we show the performance optimization of a CPSoS while meeting the respective safety demands. Our monitoring and recovery solution further highlights the potential of such a performance optimization with respect to managing car parking under varying connection conditions. Thus, we showcase the potential for performance optimization of safety-critical applications in CPSoS.

In future work, we plan to target more extensive safety-critical applications of CPSoS like Mobility-as-a-Service solutions and additional methods to improve their performance, while meeting dependability requirements.

Acknowledgments. The research leading to these results has partially received funding from the Bavarian Ministry of Economic Affairs, Regional Development and Energy as Fraunhofer High Performance Center Secure Intelligent Systems.

References

1. Road Vehicles - Functional Safety. Technical Report ISO 26262:2018 (2018)
2. Road vehicles - Safety of the Intended Functionality. Technical Report ISO/PAS 21448 (2019)
3. Biggs, G., Juknevicius, T., Armonas, A., Post, K.: Integrating safety and reliability analysis into MBSE: overview of the new proposed OMG standard. INCOSE Int. Symp. **28**(1), 1322–1336 (2018). https://doi.org/10.1002/j.2334-5837.2018.00551.x
4. Brogi, A., Forti, S., Guerrero, C., Lera, I.: How to place your apps in the fog: state of the art and open challenges. Softw.: Pract. Exp. (2019). https://doi.org/10.1002/spe.2766
5. Clegg, K., Li, M., Stamp, D., Grigg, A., McDermid, J.: A SysML profile for fault trees—linking safety models to system design. In: SAFECOMP, pp. 85–93 (2019). https://doi.org/10.1007/978-3-030-26601-1_6
6. Engell, S., Paulen, R., Reniers, M.A., Sonntag, C., Thompson, H.: Core research and innovation areas in cyber-physical systems of systems. In: Berger, C., Mousavi, M.R. (eds.) CyPhy 2015. LNCS, vol. 9361, pp. 40–55. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25141-7_4
7. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Softw. Pract. Exp. **47**(9), 1275–1296 (2017). https://doi.org/10.1002/spe.2509
8. International Electrotechnical Commission: Hazard and Operability studies (HAZOP studies) - Application guide. Technical Report IEC 61882:2016
9. Ishigooka, T., Otsuka, S., Serizawa, K., Tsuchiya, R., Narisawa, F.: Graceful degradation design process for autonomous driving system. In: Romanovsky, A., Troubitsyna, E., Bitsch, F. (eds.) SAFECOMP 2019. LNCS, vol. 11698, pp. 19–34. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26601-1_2
10. Kam, H.R., Lee, S.-H., Park, T., Kim, C.-H.: RViz: a toolkit for real domain data visualization. Telecommun. Syst. **60**(2), 337–345 (2015). https://doi.org/10.1007/s11235-015-0034-5
11. Koopman, P., Wagner, M.: Challenges in autonomous vehicle testing and validation. SAE Int. J. Trans. Saf. **4**, 15–24 (2016). https://doi.org/10.4271/2016-01-0128
12. Kopetz, H., Bondavalli, A., Brancati, F., Frömel, B., Höftberger, O., Iacob, S.: Emergence in cyber-physical systems-of-systems (CPSoSs). In: Bondavalli, A., Bouchenak, S., Kopetz, H. (eds.) Cyber-Physical Systems of Systems. LNCS, vol. 10099, pp. 73–96. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47590-5_3
13. Laprie, J.C.: From Dependability to Resilience. DSN (2008)
14. Lauer, M., Amy, M., Fabre, J.C., Roy, M., Excoffon, W., Stoicescu, M.: Resilient computing on ROS using adaptive fault tolerance. J. Softw.: Evol. Process **30**(3), e1917 (2018). https://doi.org/10.1002/smr.1917
15. Ledmi, A., Bendjenna, H., Hemam, S.M.: Fault tolerance in distributed systems: a survey. In: 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS), pp. 1–5, October 2018. https://doi.org/10.1109/PAIS.2018.8598484

16. Ozeer, U., Etchevers, X., Letondeur, L., Ottogalli, F.G., Salaün, G., Vincent, J.M.: Resilience of stateful IoT applications in a dynamic fog environment. In: Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services - MobiQuitous 2018, pp. 332–341. ACM Press, New York (2018). https://doi.org/10.1145/3286978.3287007
17. Qayyum, T., Malik, A.W., Khan Khattak, M.A., Khalid, O., Khan, S.U.: FogNet-Sim++: a toolkit for modeling and simulation of distributed fog environment. IEEE Access **6**, 63570–63583 (2018). https://doi.org/10.1109/ACCESS.2018.2877696
18. Quigley, M., et al.: ROS: an open-source robot operating system. In: Proceedings of the IEEE International Conference on Robotics and Automation Workshop on Open Source Software, vol. 3, p. 6 (2009)
19. Riley, G.F., Henderson, T.R.: The *ns*-3 network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds) Modeling and Tools for Network Simulation, pp. 15–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3_2
20. Schleiss, P., Drabek, C., Weiss, G., Bauer, B.: Generic management of availability in fail-operational automotive systems. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10488, pp. 179–194. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_12
21. Schmittner, C., Griessnig, G., Ma, Z.: Status of the development of ISO/SAE 21434. In: Larrucea, X., Santamaria, I., O'Connor, R.V., Messnarz, R. (eds.) EuroSPI 2018. CCIS, vol. 896, pp. 504–513. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-97925-0_43
22. Schönemann, V.: Safety requirements and distribution of functions for automated valet parking. Dissertation, Technische Universität Darmstadt (2019)
23. Sen, K., Vardhan, A., Agha, G., Rosu, G.: Efficient decentralized monitoring of safety in distributed systems. In: Proceedings of the 26th International Conference on Software Engineering, pp. 418–427. IEEE Computer Society, Edinburgh (2004). https://doi.org/10.1109/ICSE.2004.1317464
24. Sonmez, C., Ozgovde, A., Ersoy, C.: EdgeCloudSim: an environment for performance evaluation of edge computing systems. Trans. Emerg. Telecommun. Technol. **29**(11), e3493 (2018). https://doi.org/10.1002/ett.3493
25. Thompson, N.C., Greenewald, K., Lee, K., Manso, G.F.: The Computational Limits of Deep Learning. arXiv:2007.05558 [cs, stat], July 2020
26. Törngren, M., Sellgren, U.: Complexity challenges in development of cyber-physical systems. In: Lohstroh, M., Derler, P., Sirjani, M. (eds.) Principles of Modeling. LNCS, vol. 10760, pp. 478–503. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95246-8_27
27. Varga, A., Hornig, R.: An overview of the OMNeT++ simulation environment. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, pp. 1–10. ICST, Marseille (2008)