Assuring Degradation Cascades of Car Platoons via Contracts

Irfan Sljivo[1], Barbara Gallina[1], and Bernhard Kaiser[2]

[1] Mälardalen University, Västerås, Sweden
{irfan.sljivo,barbara.gallina}@mdh.se
[2] Berner&Mattner Systemtechnik GmbH, Germany
bernhard.kaiser@berner-mattner.com

Abstract. Automated cooperation is arriving in practice, for instance in vehicular automation like platoon driving. The development and safety assurance of those systems poses new challenges, as the participating nodes are not known at design time; they engage in communication at runtime and the system behaviour can be distorted at any time by failures in some participant or in the communication itself. When running on a highway, simply switching off the function is not an option, as this would also result in hazardous situations. Graceful degradation offer a systematic approach to define a partial-order of less and less acceptable operation modes, of which the best achievable is selected in presence of failures. In this work we propose an approach for assurance of the degradation cascades based on mode-specific assertions, captured by assumption/guarantee contracts. More specifically, we share our experiences and methodology for specifying the contracts for both the nominal safe behaviour as well as the less safe but acceptable behaviour in presence of failures. Furthermore, we present an argument pattern for adequacy of the degradation cascades for meeting the global safety goals based on the contracts. We illustrate our approach by a car platooning case study.

1 Introduction

Cooperative systems represent the cornerstone of the fourth industrial revolution [1]. A typical example are vehicle platoons, where an automated fleet of vehicles join via Car2Car connection to achieve a cooperative function such as Cooperative Automated Cruise Control (CACC). Many of such systems are safety-critical. Accordingly, a technical safety concept is mandatory for discussing technical design solutions for all potential failure modes and their tolerance through detection and reaction mechanisms. When releasing the individual vehicle for road usage, an argument (called safety case) must be provided that the system is actually safe to the safety integrity level that has been claimed. The number of operation modes and of failure possibilities of the cooperative system as a whole explodes due to the combinatorics of operation and failure modes of all individual participants, plus failure modes of the communication link that can occur at any time. A safety argument must be designed at design time for each possible

configuration at runtime, and onboard-mechanisms must execute corresponding failure mode detection and reaction in every configuration. As the number of combinations would make this unmanageable, usage of abstraction techniques is necessary, focusing only on the relevant assertions on black-box-level.

Moreover, these systems are often safe-operational systems (e.g. a car platoon on a highway, driving automatically without the drivers being ready to react at short notice), which means that the reaction on failures cannot be as primitive as just switching off the function in one of the participating cars, but its partial functioning should be ensured instead. Dynamic system adaptation [2] is an approach to reconfigure the system to the best achievable operation mode in case of failures, thereby trading off between safety, availability and functionality provided. Which operation mode is the best can be determined by ordering the operation modes in a degradation cascade [3, 4]. An ordered set of rules determines when a certain operation mode or system configuration should be activated, triggered by failure detection, typically issued by model-based health monitors for sensors and other critical system parts.

Contract-based approaches have been frequently used for compositional verification and also offer means to specify and verify the reconfiguration rules in terms of contracts. They allow both checking of valid system configurations and checking refinement between the different hierarchical levels. A contract is a pair of assertions in form of assumptions and guarantees, where a component guarantees its own behaviour provided that the environment fulfils the assumptions. As reconfigurable components are characterised with different behaviours, we distinguish between strong and weak contracts [5]. A strong contract must hold in every environment, while a weak one is not required to hold in every environment. Only when, besides the strong assumptions, the weak assumptions are also met, the component offers the behaviour in the weak guarantees.

In our previous work we presented FLAR2SAF (Failure Logic Analysis Results to Safety Argument-Fragments) [6] – a partially tool-supported method that uses CHESS-FLA [7] to derive contracts and generate safety case argument-fragments. The basis for the argument generation is the connection of a contract with the corresponding safety requirement. Specifying requirements by stating *"X shall always happen"* is inadequate for degradation cascades where the requirements should describe a cascade stating what shall happen if for example X is not always possible [8]. We see weak contracts as a way to capture behaviour described in these complex safety requirements for degradation cascades.

In this work we propose systematic design and pattern-based safety assurance of degradation cascades using contracts. To derive contracts, we examine potential failures of initial system architecture using standard approaches, define failure detection mechanisms and the resulting operation mode changes through degradation cascade requirements. Based on the specified degradation cascade requirements and domain knowledge we derive degradation cascade contracts, which can be used to build an argument for assuring adequacy of degradation cascade to address the corresponding hazard. Finally, we illustrate the usage of the approach in a CACC case study.

The rest of the paper is organised as follows: In Section 2 we present background information. We present the assurance of degradation cascades using contracts in Section 3. In Section 4 we present the CACC case study. We present the related work in Section 5, and conclusions and future work in Section 6.

2 Background

In this section, we briefly recall the basic notions regarding component contracts and degradation cascades.

2.1 Assumption Guarantee Component Contracts

A traditional component contract $C = \langle A, G \rangle$ is composed of assumptions (A) on the environment of the component and guarantees (G) that are offered by the component if the assumptions are met. Strong $\langle A, G \rangle$ and weak $\langle B, H \rangle$ contracts [5] provide support for capturing variable behaviour of reusable components. The strong contract assumptions (A) are required to be satisfied in all contexts in which the component is used, hence the corresponding strong guarantees (G) are offered in all contexts in which the component can be used. The weak contract guarantees (H) are offered only when in addition to the strong assumptions, the corresponding weak assumptions (B) are satisfied as well.

Contracts can be used to (semi)-automatically instantiate existing safety case argument-patterns based on the SEooCMM metamodel [6] that captures the relations between the contracts, the supporting evidence and the safety requirements allocated on the component. Every contract is supported by one or more evidence items and every allocated safety requirement is addressed by at least one contract such that satisfaction and confidence in the associated contracts supports satisfaction of the corresponding requirement. We use Goal Structuring Notation (GSN) [9] – a graphical argumentation notation – to represent the argument-fragments. The GSN elements used in this paper are shown in Fig. 1.

2.2 Degradation cascades

Graceful degradation is seen as a way to improve dependability of a system by degrading its performance proportionally to the failures of its components [4]. Degradation cascades represent a partial order over a labelled set of operation modes where system degrades its performance based on the presence of certain failures, while always choosing the most convenient available mode at any time. As an ordering scheme, existing classifications like the SIL or ASIL (Automotive Safety Integrity Level) according to safety standards such as ISO 26262 or the Severity factor (S) known from Failure Mode and Effects Analysis (FMEA)may be used for labelling the operation modes in terms of safety criticality. The labelling can be denoted graphically by different colours ranging, for instance, from green (fully functional and safe) over yellow (degraded function, but still safe) and orange (emergency function, hazard of low severity) to red (hazardous).

Representing degradation cascades in terms of safety requirements is not as straightforward as stating *"The system shall do X"*. Rather, if X is not available, system should do something else by going to a degraded state. Furthermore, if that degraded state cannot be maintained, the system should further degrade its performance, until the final state is the uncontrollable failure that should be sufficiently unlikely to occur. Hence complex requirement structuring mechanisms are proposed for capturing degradation cascades where each degradation mode is represented by an if-else style "sub-requirement" [3]. Each of these "sub-requirements" addresses a single degradation mode by listing under which conditions should the particular degradation mode become active. For example, for a simple Lane Keeping Assist (LKA) system [3], the nominal requirement would be that *RQ1:"When active and no input failures the LKA system shall guide the vehicle in the middle of the lane with allowed tolerance of 0.5m"*. And its alternative requirement *RQ2:"If the system cannot achieve that (e.g., because it cannot detect the lane borders), then the vehicle shall keep in the middle of the neighbouring vehicles and issue an urgent take over"*.

3 Car Platooning Degradation Assurance via Contracts

In this section we first sketch a contract-based approach for design of degradation cascades for car platooning. Then we present the argumentation pattern for degradation modes based on the captured degradation mode contracts.

3.1 Contract-based Design of Degradation Cascades

Unlike for traditional cars, the term failure comprises for cooperative systems:

1. Technical Failures in the local car (e.g. sensor failures, actuator failures, controller failures);
2. Technical Failures in another participating car (remote failures), impairing the integrity of any information provided by this car;
3. Failures in the communication between any cars (e.g. message loss, message delay, message corruption).

The safety case will have to show that for any combination (!) of platoon configurations, environmental situations (listed and classified before) and failure modes (neglecting multiple failures, at first) the safety goal is not violated.

Combining our previous works on contract-based design [10], structured design of degradation cascades [8] and contract-based safety case argument generation [6], we propose a combined design and safety assurance of degradation cascades using component contracts. Just as in our previous works [10, 6], we follow the same generic safety assurance process using component contracts. We model the fault-free architecture, perform safety analysis to identify the needed extensions in terms of safety measures, and then derive contracts from results of such analyses. Finally we use the contracts to instantiate the corresponding argumentation pattern. We detail the process for the case of deriving the contracts for car platooning degradation cascades:

1. Model the local controller chain and the local operation state machine of each car for the failure-free case, using SysML and, if applicable, contracts (strong contracts for now), in a similar way as described in Section 3.3 of [10]. Validate the platoon use cases (e.g., join platoon) by simulation (building the product state-space would be unmanageable for humans, but could in the future offer the possibility for model checking);
2. Examine potential failures (local/remote/communication) and their consequences using standard approaches such as Failure Mode and Effects Analysis (FMEA) and define detection mechanisms (e.g. range checks for sensors, timeout and CRC checks for communication link). Failure classes can be used to structure this, and failures can be interpreted as violations of the initial contracts, as shown in Section 4 of [10]. Note: technically, this is implemented by observers or monitors as separate Simulink blocks, and captured by separate sub-state machines that report a Boolean failure state. Define safety mechanisms that change the operation mode (and therefore, the controller structure and/or the controller parameters, e.g. distance-to-predecessor setpoint) in case of detected failures. When doing so, try to fall back to the best performing, but yet safe operation mode, leading to a degradation cascade (denoted as a partial order of colours ranging from green to red, with which the operation modes are labelled, as shown for a local system in [8];
3. Collect degradation mode contracts for each local operation mode and adjust the parameters and controller configurations for each mode (and, thereby, the resulting guarantees) so that for each valid operation mode, the overall safety goal is implied.

3.2 Degradation Cascades Safety Argument Pattern

We use the degradation mode contracts to assure that the safety of the degradation cascade is adequate. We define the degradation cascades argument pattern (Fig. 1) that can be instantiated from such contracts using the technique from our previous work [6]. The argument assures that the degradation cascade has adequately addressed the causes leading to the corresponding hazards (*DegCascade* goal in Fig. 1). This means that the unreasonable risk of the hazard should be absent in each operation mode, both during nominal behaviour ($\{DMx\}$-*nomBeh*) and in presence of failures ($\{DMx\}$-*Str*). Hence, we develop the argument for each operation mode specified in the degradation cascade by looking into the identified failure combinations relevant for that particular mode. First, we assure that the mode is adequately safe under nominal conditions (when the identified relevant causes are absent). Then, we assure that increasing the system availability by switching to a degraded mode keeps the system acceptably safe when a relevant failure combination occurs ($\{DMx\}$-$\{failCombN\}$). In particular, we need to argue not only that the condition triggering the degradation (as described in the corresponding contract) is adequate ($\{failCombNContract\}Adeq$), but also that the target operation mode is adequate in presence of the particular failure combination ($\{DMy\}Acceptable$). We develop the lower-level parts of

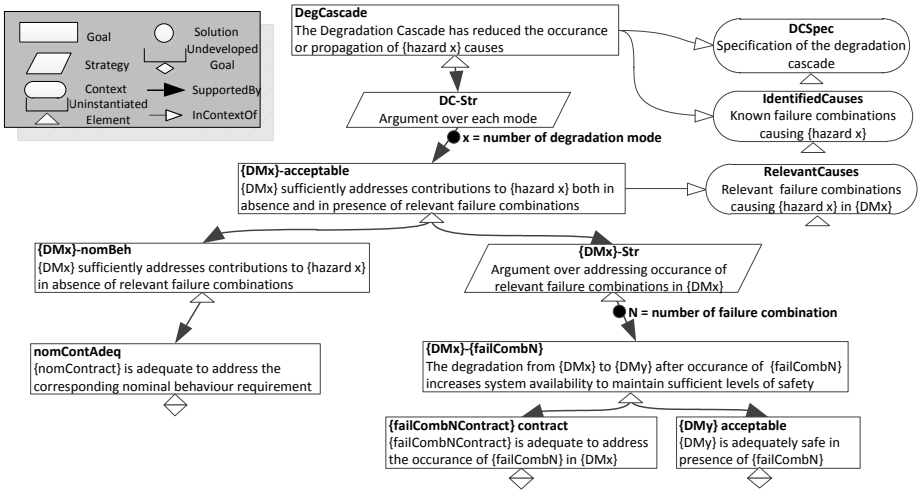

Fig. 1. Degradation Cascades (DC) Argument Pattern in GSN

the argument (*nomContAdeq* and {*failCombNContract*}*Contract*) related to the confidence in such contracts based on the contract satisfaction pattern [11].

4 CACC and Platooning Case Study

In this case study we use degradation cascade contracts on a CACC system. We first describe the system, and then apply the process described in Section 3.1.

4.1 CACC and Car Platooning

A typical example of a cooperative safety-critical system is CACC – smart cruise control guided by a predecessor vehicle via a Car2Car link, as well as vehicle platooning as an extension of CACC, where additional Car2Car connection is established to the leader vehicle (the first vehicle of the platoon) that coordinates the whole platoon. Fig. 2 explains the different operation modes. As a case study in the AMASS research project [12], we have built up a fleet of autonomous model cars in the scale 1:8 that can run autonomously and sense the road and any other cars or obstacles by means of camera and ultrasonic sensors. Additionally, the cars can establish a WIFI connection to one another at runtime to exchange Car2Car messages. When doing so, several cars can transit to CACC mode, and in a next step form a platoon where other cars can join in, thereby forming a system-of-systems (SoS). Several use cases have been modelled and implemented, such as “create platoon”, “join platoon”, “leave platoon” etc. Apart from manual driving (for model cars, this means: operated by a radio remote controller), there are the operation modes CC (cruise control, i.e. running alone with fixed speed), ACC (adaptive CC, i.e. perceiving the predecessor car with local distance sensors and adjusting speed accordingly), CACC (cooperative ACC, i.e. with

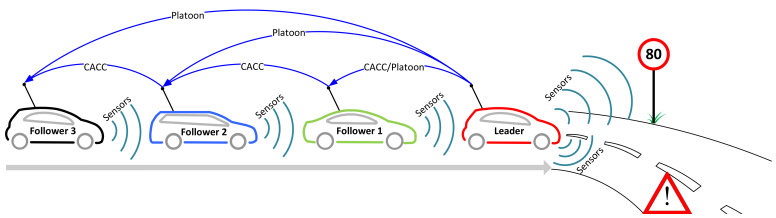

Fig. 2. Explanation of CACC and Platoon Driving

radio connection established to the predecessor car, which informs about its position, speed and manoeuvre intentions), and platoon (same as CACC, but all participants being informed via radio by the platoon leader vehicle, not just the immediate predecessor).

4.2 Safety Aspects of CACC and Platoon Driving

The SoS such as CACC and car platooning come with different hazards. We choose the rear collision as our running example. Note that, in contrast to today's road vehicles, safety standards such as ISO 26262 cannot be applied directly to vehicle platoons, because this standard allocates the overall safety responsibility to the system manufacturer, but a platoon has no manufacturer, as the participating cars come from different carmakers. However, global safety goals can be stated in a similar way as in traditional safety engineering, such as *"Any two cars shall always maintain a front-rear-distance of at least 2m to each other"* (of course, the scaling has to be adapted from real world to model cars). It has to be proven that the safety condition holds for each mode of operation, and for each expectable environmental situation (e.g. sudden strong braking of the leading vehicle, which can be constrained by an assumption about physically reasonable deceleration values), even in presence of failures.

To explain the process of failure analysis and degradation chain creation, we show in Fig. 3 a simplified excerpt of the application state machine of one particular vehicle (reduced to the case that this vehicle is not the platoon leader, but any of the follower cars). The abstraction (in comparison to the technically implemented state machine) leaves out technically necessary states (such as waiting for WLAN connection) and directs the attention to the overall operation modes, which form a degradation cascade (marked by the different colours). This abstraction is adequate for safety considerations, and also the reduction to the state machine of one single vehicle is appropriate for the hazard of rear collision.

In this state machine we can see some external events (operator commands, presence of another vehicles) as trigger events for operation mode transitions, but also some failure events (cf. Section 4 of [8]). For instance, the transition from manual drive to ACC is triggered by a user intervention (activation button pressed), the transition from ACC to CACC by Car2X engaged, i.e. a connection has been established and both vehicles have agreed upon who is preceding and who is following vehicle. The transitions back are the complementary events (e.g. deactivation button pressed), but also some failure events: When, for instance,

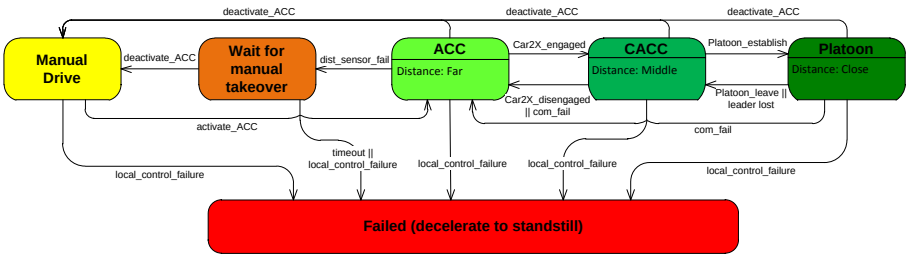

Fig. 3. Simplified Application State Machine Including Degradation

the communication by radio gets lost in CACC or platoon mode, the fall-back is ACC (just using the front distance sensor to adjust oneself with respect to the vehicle in front). As we will see, this includes in this case not so much a change of the speed and distance controller *structure*, but rather of some *parameter* (the distance setpoint), because the simultaneous information that the vehicle in front is about to brake is no longer available. The front distance sensor (ultrasonic based in case of the model car) also allows recognising the effect of a braking manoeuvre, but with a great deal of a delay, leading to a rear collision without sufficient safety distance. If the communication does not break down, but the predecessor or platoon leading vehicle reports an error via Car2X, a similar transition happens. When, as another example, the front sensor fails in ACC mode, the only remaining degradation level (above failed) is a manual takeover, so the vehicle goes into a transition state (orange coloured, because hazardous if pertaining too long) and prompts for takeover. If the takeover (user intervention) occurs in time, the car is back in manual mode (yellow, because safe, but not really a state of preference), but if a timeout occurs, the car changes to failed (red), which means that the car carefully decelerates to standstill. Of course, there are many other faults (all of them local faults) that lead to an immediate transition to the failed state, e.g. sensor problems with the wheel encoders or motor sensors, or any other kind of controller or actuator failures. In this case, no reasonable degradation is possible any more. Note that, for the purpose of safety analysis by simulation as described in [8], it may be useful to model more discrete states as shown in the figure (reflecting the technical implementation): when a mode change due to a failure occurs, there is a short time interval where the distance setpoint has already been increased, but the controller takes some time to increase the actual distance between the cars. Technically, this is not an extra state, just the setting time that every controller exhibits. For safety analysis, this is a state that must be considered (and coloured in orange, i.e. hazardous to some degree), because a braking manoeuvre of the predecessor car in this situation could lead to a rear crash. Therefore, an estimate for the frequency and duration of these states must be derived and compared to the acceptable hazard rates. Of course, the shown simplification to a single car does not consider effects like the chain-stability of the whole platoon (e.g., one car avoids the accident, but only by strong braking, which causes subsequent cars to crash). A part of these questions has been investigated by simulation in [13].

Table 1. The strong contract representing the overall safety goal

$\mathbf{A}_1$:	No car can decelerate with more than $8m/s^2$ AND A nominally performing car can decelerate with at least $5m/s^2$;
$\mathbf{G}_1$:	Maximum deceleration is within [$5m/s^2$,$8m/s^2$] AND (Platooning, CACC or ACC mode active) $\rightarrow$ The distance between the considered car and its predecessor car is always greater than 2m;

Table 2. A subset of the degradation cascade contracts for platoon and CACC modes

$\mathbf{B}_{P1}$:	Platoon active AND no local control failure AND no distance sensor failure AND no car2x failure AND Braking of the predecessor vehicle is recognised within 30ms;
$\mathbf{H}_{P1}$:	The distance to the predecessor vehicle is always greater than 20m AND A sudden braking manoeuvre of the preceding vehicle does never lead to a resulting distance of less than 2m;
$\mathbf{B}_{P2}$:	Platoon active AND no local control failure AND no distance sensor failure AND no car2prec failure AND car2leader failure;
$\mathbf{H}_{P2}$:	Transition to CACC mode within 10ms;
$\mathbf{B}_{P3}$:	Platoon active AND no local control failure AND no distance sensor failure AND car2x failure;
$\mathbf{H}_{P3}$:	Transition to ACC mode within 10ms;
$\mathbf{B}_{CACC1}$:	CACC active AND no local control failure AND no distance sensor failure AND no car2pred failure AND Braking of the predecessor vehicle is recognised within 60ms;
$\mathbf{H}_{CACC1}$:	The distance to the predecessor vehicle is always greater than 30m AND A sudden braking manoeuvre of the preceding vehicle does never lead to a resulting distance of less than 2m;
$\mathbf{B}_{CACC2}$:	CACC active AND no local control failure AND distance sensor failure;
$\mathbf{H}_{CACC2}$:	Transition to ACC mode within 10ms;

4.3 Specifying Degradation Cascade Contracts

Before specifying the degradation cascade contracts, we first state the overall degradation cascade safety goal in terms of a strong contract presented in Table 1. The overall safety goal contract guarantee should be implied by each mode. While the stated strong assumptions are the basis for calculations done when establishing the thresholds for each of the degradation modes. The safety goal here is a simplification, as the original Safety Goal deals with the distance between any two cars, may they be part of a platoon or not.

For the individual degradation modes, we have then collected sets of weak assumptions and guarantees. We have not formally proven the guarantees, but validated them by simulation. The parameters depend on the performance (accuracy and dead time, in particular) of the local sensors or the information transfer via Car2X, respectively, and could only be estimated conservatively. We present some simplified examples of the degradation mode contracts based on our domain knowledge regarding the controller structure in Table 2. A part of

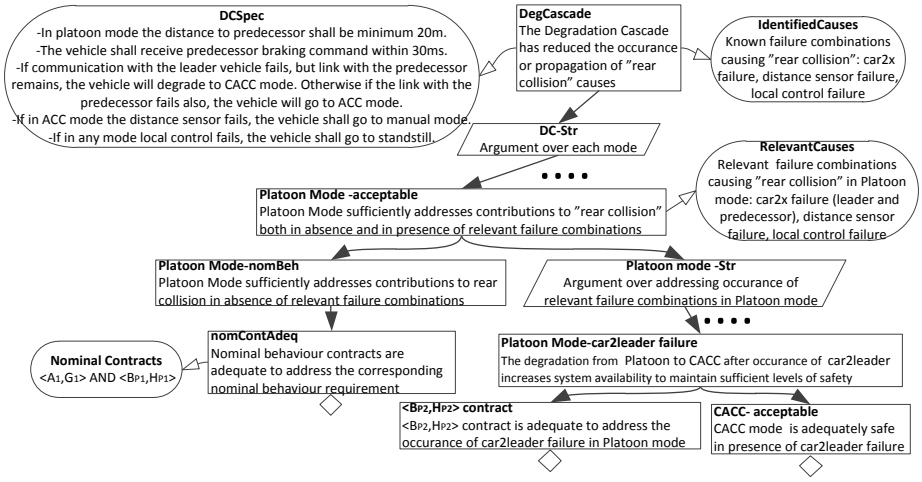

Fig. 4. Car Platooning Degradation Cascades Argument-fragment

the instantiated argument based on the presented pattern is shown in Fig. 4. The example covers only a portion of the platoon mode argument.

5 Related Work

Shelton et al. [4] present a framework for graceful degradation of distributed embedded systems based on the idea of configuration space that forms a product family architecture. Instead of specifying and designing degradation for every possible combination of failures individually, they propose a framework for focusing only on valid component configurations to reduce the number of failure combinations to examine. Similarly, in our work we use contracts to focus and specify only the valid configurations of the cooperative SoS and not all possible states. Schneider et al. [14] introduce Conditional Safety Certificates (ConSerts) specified by directed acyclic graphs that besides demands and guarantees also contain runtime evidence, gates and directed edges. To move part of assurance at runtime with ConSerts, it is important to formulate different ConSert variants at development-time such that the ConSert conditions can be resolved at runtime and the corresponding safety requirements in terms of guarantees established. Strong and weak contracts associated with requirements and the supporting evidence work in a similar way. In our work, we focus on the degradation cascades and propose how such conditional assurance can be achieved using contracts. Assumption/guarantee contracts that support specification of variable behaviour [15, 5] have been used to promote reuse of assurance artefacts. In this work, we utilise the possibility of specifying not design-time, but runtime variable behaviour such as behaviour exhibited by different degradation modes. Iliasov et al. [16] formally define notions of modes and their refinement in Event-B state-based formalism. These notions allow for describing system operation modes using assumptions to capture system conditions and guarantees

to express the behaviour expressed under those conditions. We define the contracts for degradation cascade in a similar fashion and use them as the basis for degradation cascade assurance.

6 Conclusion and Future Work

In this paper we have sketched a design and safety assurance approach for cooperative SoS exhibiting degradation cascades. More than traditional non-distributed systems, cooperative systems need to cope with not just the local failures, but also ones in other peers that are announced by the communication link, and in the communication link itself. Many of such systems can cause hazards and therefore need safety properties to be ensured, which involves the introduction of safety mechanisms. A total shutdown in case of any failure is often not acceptable, so a structured way of defining degradation cascades is mandatory, but hard to verify. To address these challenges, we have combined and extended some recent research contributions: the argument-fragment generation technique FLAR2SAF, a structured design approach for degradation cascades, and an approach of contract-based design. The particular thing about using contracts in our approach is that we use weak contract - originally proposed for facilitating the reuse of components in new environments - at runtime to define contract for different levels of degradation. Thereby, the process of selecting the best strategy to fulfil a given guarantee under varying assumptions is shifted from design time to runtime. Yet we can prepare the safety argument at design time by iteration over all possible degradation levels and arguing by assumption/guarantee matching that the guaranteed behaviour is acceptably safe. We have applied our approach to a fleet of autonomous model cars that perform CACC and platoon driving and successfully validated it by experiments. It should be noted that the approach is not only applicable to vehicle systems, but to any kind of cooperating cyber-physical systems, e.g., sensor networks or distributed automation systems. As a next step plan to formalise the contracts in OCRA language to prove the safety properties and the correct decomposition and allocation of contracts. To address verification of the atomic components (i.e. the implementation) and properties in the environment (e.g. that a certain distance between cars is sufficient to prevent collisions under all conditions), we plan to integrate other verification approaches such as model checking or simulation. To make our approach applicable to industries, we will need to build a tool chain that helps evaluating the possible configurations and their global guarantees at design time and to create software code for safety arbiters that evaluate the contracts at runtime on board the vehicle and select the appropriate degradation mode.

Acknowledgements. This work is supported by EU and VINNOVA via the ECSEL Joint Undertaking project AMASS (No 692474).

References

[1] Kagermann, H., Helbig, J., Hellinger, A., Wahlster, W.: Recommendations for Implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry. Forschungsunion (2013)

[2] Adler, R., Schaefer, I., Trapp, M., Poetzsch-Heffter, A.: Component-based Modeling and Verification of Dynamic Adaptation in Safety-critical Embedded Systems. ACM Transactions on Embedded Computing Systems **10**(2) (2011) 1–39

[3] Kaiser, B.: From "Safe State" to "Degradation Cascades" – Structured and Quantified Requirements for Automated Driving Systems. Presentation at VDA Automotive SYS. Berlin, Germany (2016)

[4] Shelton, C.P., Koopman, P., Nace, W.: A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems. In: 8th International Workshop on Object-Oriented Real-Time Dependable Systems, IEEE (2003) 156–163

[5] Sljivo, I., Gallina, B., Carlson, J., Hansson, H.: Strong and Weak Contract Formalism for Third-Party Component Reuse. In: 3rd International Workshop on Software Certification, IEEE (November 2013) 359–364

[6] Sljivo, I., Gallina, B., Carlson, J., Hansson, H., Puri, S.: A method to generate reusable safety case argument-fragments from compositional safety analysis. Journal of Systems and Software: Special Issue on Software Reuse (July 2016)

[7] Gallina, B., Javed, M., Muram, F., Punnekkat, S.: Model-driven Dependability Analysis Method for Component-based Architectures. In: 38th Euromicro Conference on Software Engineering and Advanced Applications, IEEE (September 2012) 233–240

[8] Kaiser, B., Nejad, B.M., Kusche, D., Schulte, H.: Systematic Design and Validation of Degradation Cascades for Safety-Relevant Systems. In: To Appear in The annual European Safety and Reliability Conference ESREL. (June 2017)

[9] Goal Structuring Notation Working Group: GSN Community Standard Version 1. Origin Consulting (York) Limited (2011)

[10] Kaiser, B., Weber, R., Oertel, M., Böde, E., Nejad, B.M., Zander, J.: Contract-Based Design of Embedded Systems Integrating Nominal Behavior and Safety. Complex Systems Informatics and Modeling Quarterly **4** (October 2015) 66–91

[11] Sljivo, I., Gallina, B., Carlson, J., Hansson, H.: Generation of Safety Case Argument-Fragments from Safety Contracts. In: 33rd International Conference on Computer Safety, Reliability, and Security. Volume 8666 of Lecture Notes in Computer Science., Springer (September 2014) 170–185

[12] ECSEL-JU-692474: AMASS – Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems. http://www.amass-ecsel.eu/

[13] Ghodratbaki, A.: Modellierung lose gekoppelter System-of-Systems am Beispiel eines Cooperative Adaptive Cruise Control (CACC) Fahrerassistenzsystems. Master's thesis (2017)

[14] Schneider, D., Trapp, M.: Conditional safety certification of open adaptive systems. TAAS **8**(2) (2013) 8:1–8:20

[15] Oertel, M., Schulze, M., Peikenkamp, T.: Reusing a Functional Safety Concept in Variable System Architectures. In: 7th International Workshop on Model-based Architecting and Construction of Embedded Systems. (September 2014) 16–25

[16] Iliasov, A., Romanovsky, A., Dotti, F.L.: Structuring specifications with modes. In: LADC, IEEE Computer Society (2009) 81–88