Cooperative Safety Critical CPS Platooning in SafeCOP

Samer Medawar, Detlef Scholle
Embeded Systems
Alten AB
Kista, Sweden
{samer.medawar,detlef.scholle}@alten.se

Irfan Šljivo
School of Innovation, Design, and Engineering
Mälardalen University
Västerås, Sweden
irfan.sljivo@mdh.se

Abstract— **This paper presents the platooning research within the Safe Cooperating Cyber-Physical Systems using Wireless Communication (SafeCOP) project. Cooperating Cyber-Physical Systems (CO-CPS) using wireless communication and having multiple stakeholders, dynamic system definitions (openness), and unpredictable operating environments, are the main application of SafeCOP. In addition to safety assurance methods and tools, SafeCOP devises a runtime manager architecture that detects irregular operation, hence, prompting a safe degraded mode in case of need.**

SafeCOP lays a safety and security umbrella over the usage of current wireless technologies, contributes to new standards and regulations by providing scientifically validated solutions to establish standards which also addresses cooperation and system-of-systems issues. SafeCOP addresses several use cases that solve customer related problems. However, in this paper we will present a use case that extract generic principles from the combination of the previous use cases to stimulate the European collaboration around the project objectives, and to collect general requirements for the SafeCOP solution, applicable across all the areas considered. We consider a CO-CPS composed of two or more systems moving in a platoon while cooperating in a safe function.

Keywords-component; cyber-physical systems; systems-of-systems; safety-assurance; wireless communication; platooning

I. INTRODUCTION

SafeCOP targets safety-critical systems that are required to provide a "safety case" – a well-documented body of evidence in form of a clear argument assuring that the system is acceptably safe. Building the safety case requires gathering the safety evidence during system development to ensure not only that identified failures have been addressed, but also that any unwanted interactions between the system parts as well as the environment have been managed. Such task is lengthy and generally not straightforward and leads tangible cost (time and money) increase in the order of at least 25% up to 1000% [1]. In CO-CPS this task is even more challenging as the system boundary (all the cooperating systems performing the safe function) and the environment are development [2]. Hence, SafeCOP aims to decrease cost and improve efficiency for crafting safety evidence for CO-CPS by developing safety assurance methods and tools combined with a runtime manager. SafeCOP seeks also global acceptance of these methodologies by contributing to the standardisation and regulation committees.

The SafeCOP ECSEL project was presented in [3] where the concept for safety assurance of CO-CPS is detailed together with the runtime manager function. Also, five use cases (UC)s, which are to solve customers concrete problems, were also presented. These UCs are briefly explained here:

- **UC1**: **Cooperative moving of empty hospital beds.** A two-robot autonomous bed mover would wheel smoothly through the densely occupied corridors of a hospital. The system will insure that the bed and the robots do not collide with the surrounding physical environment. Both the system failures and the system interactions with external emergencies are treated in the safety insurance case.
- **UC2: Cooperative bathymetry w/boat platoons.** This project develops a proof-of concept system (method and tools) for semi-autonomous boats and other vehicles cooperating to conduct bathymetry measurements for a portion of a port.
- **UC3: Vehicle control loss warning.** A Control Loss Warning system for a vehicle platoon is developed where the vehicles and road infrastructures are notified in case a vehicle in the platoon looses some functionality that affects the platoon.
- **UC4: Vehicles and roadside units interaction.** The roadside weather (RSW) is a device that collects measurements parameters for weather and traffic. Generally, these parameters are sent to a road administrator that further sends them to TV and radio station. SafeCOP will add the feature of the RWS sending the information directly to the passing vehicles. Also, the vehicles will collect the same data parameters and deliver them to the RWS. The runtime manager will monitor sequential measurements and checks.

The research leading to these results has been performed in the SafeCOP-project, that received funding from the ECSEL Joint Undertaking under grant agreements n°692529, and from National funding.

- **UC5: Vehicle to Infrastructure cooperation for traffic management.** In this use case the focus is laid on Intelligent Transport Systems (ITS) in order to improve the efficiency and safety of transportation.

A. *Contributions*

In this paper we present our initial results for the SafeCOP approach to continuous safety assurance using the notion of runtime manager. We detail the role of the runtime manager and identify which conditions it should monitor during runtime. Furthermore, we present how the data gathered by the runtime manager can be used for continuous safety assurance. We illustrate the continuous safety assurance with such runtime manager on the SafeCOP platooning use case.

II. SafeCOP Continuous Safety Assurance Approach

In this section we first present the general SafeCOP safety assurance concept (Figure 1) and then we present the SafeCOP continuous safety assurance approach using runtime manager.

A. *SafeCOP Safety Assurance Concept*

Assumption/guarantee contracts facilitate compositional verification and allow for independent development of components. The distinction on the strong and weak contracts is introduced to support specification of globally accepted and context/situation specific behaviours. This is especially relevant for reusable components that exhibit different behaviours in different systems, and there are some behaviours that should be enforced in every system. The strong contracts would be used for such behaviours for all systems, while the weak ones would be specific to only systems that satisfy the weak assumptions [4]. As the contracts capture safety-relevant behaviours, they are used during system development for generating system-specific safety case arguments that can be represented using Goal Structuring Notation (GSN) [5]. Just as the systems nowadays are rarely built from scratch but from pre-developed components, development of the safety case follows the same pattern. Research on safety cases in [6] focused on developing modular certification approaches, where a modular safety certificate is given to an individual subsystem (module) and thereafter these certificates are manually composed into a system certificate. When preparing a safety case for a traditional safety-critical system by composing it from pre-developed parts, the first condition we come across is whether the safety evidence is relevant for the particular environment and can it be reused or not.

Figure 1 shows two CPS systems whose behaviour is captured using assumption/guarantee contracts based on the allocated system and safety requirements. Each system has a cooperative subsystem responsible for forming a cooperative safety function. This cooperative system includes for example standardised cooperation protocol and runtime manager. As part of the traditional system safety case required by the standards, SafeCOP safety assurance concept includes a safety case module for the cooperative function. The static part of this module is decided during design time, but the dynamic safety

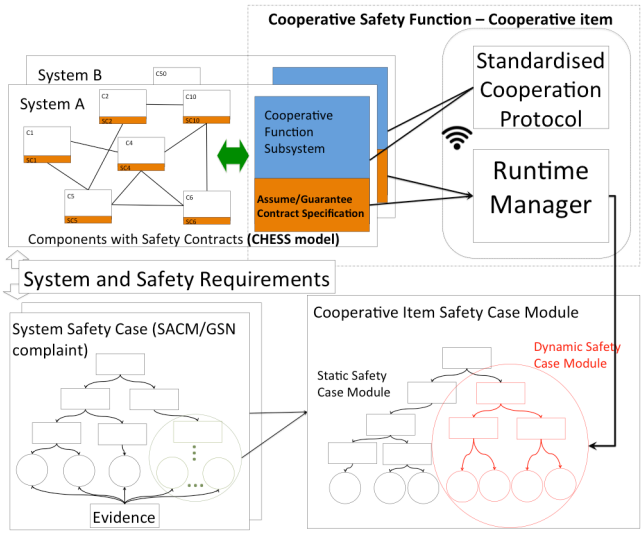

Figure 1: SafeCOP Safety Assurance Concept

case module is where the continuous safety assurance should take place.

B. *Continuous Safety Assurance Using Runtime Manager*

In open systems such as CO-CPS, it is not possible to fully assure a system before deployment, but due to the unknown environment, constant assurance is needed to maintain the confidence in the system. This means that the first condition in building a safety case is exhibited during design-time, when we compose the system from different components, and the second condition is during runtime, where the system comes in contact with unknown environments. The second condition in assuring the system relates to the runtime assurance claim whether the system is still sufficiently safe (whether the contracts are violated) in the current environment or not. We use the runtime manager to continuously assure the runtime assurance claim.

We include the runtime manager in SafeCOP architecture to evaluate contracts during runtime to achieve the adaptability of both the system behaviour and the safety case. We use strong and weak contract paradigm to facilitate checking of the second condition when building the assurance case. Continuous runtime manager checking of contract violations generates diagnostic data. We use this data derive either supporting evidence in the completeness of the contracts or to provide counter examples in their completeness, which can be resolved through update of the system and the corresponding contracts that have been invalidated at runtime.

The main role of the runtime manager for supporting runtime assurance is checking for contract violations. As contracts are specifications of behaviour of the system, then contract violations are the system failures. Since strong contracts must be satisfied always and by all environments, then their violation during runtime indicates that there was a failure in the environment, i.e., the behaviour guaranteed at design-time has been broken. On the other hand, if the strong contract assumptions are not violated, then the runtime manager should check if the system offers the promised

guaranteed behaviour. If the guaranteed behaviour is not provided, then an internal failure exists. Pinpointing the cause of the broken guarantees can be done by looking at the violations of the weak contracts. A weak contract is violated if provided the assumptions; it fails to offer the guaranteed behaviour.

The dual role of the runtime manager is on the one hand to report the failures of the real components compared to the modelled contract behaviour, and on the other hand to log all violations, and contribute to estimating the confidence in the specified contracts under the different environments the open system is exposed to.

III. PLATOONING UC

The UCs outlined in Section I come form specific customer requirements and aims to solve the problems encountered by these customers. The UC6 combines the experience and tools gained from the previous UCs to develop a cooperative safe critical communication and control system for platooning applications.

A. Relation to Other SafeCOP Demonstrators

We consider a CO-CPS composed of two or more systems that have to cooperate to perform a safety function. The cooperation relies on the wireless communication to reach a common goal. The cooperative safety function considered is that the systems move in a platoon and that they stop if one of the safety requirements is violated, e.g., the systems get too close to an obstacle or the wireless link brakes down. We consider two fall-back functions: the first function is a SafeCOP function, where the runtime manager detects a safety requirements violation and stops all the cooperating systems, and a more interesting and potentially useful fall-back function consisting of a safe park, where the CO-CPS systems, once a particular safety violation is detected, would independently go to a specified parking location, while keeping a safe distance from each other. The communication is based on 802.11p and the use case requires high integrity from the wireless communication.

Positive and negative experiences gained from integrating SafeCOP in the respective UCs' demonstrators are fed back into the technical work packages (WP)s to aid the iterative development of the SafeCOP system. This information will be used to evaluate the ease of integration of SafeCOP into the various types of systems used in our demonstrators. Conclusions drawn from this will be used to improve the integration-friendliness of SafeCOP components, with the goal of improving adoption of the SafeCOP system.

B. Platoon functionality architecture

Platooning functionality allows vehicles to cooperate by communicating with each other while in the platoon mode. The safety-criticality of these systems cannot be constrained to only a single vehicle, as each vehicle in the platoon depends on other vehicles. Hence, for a single vehicle to be sufficiently safe, the behaviour of the platoon as a multi-vehicle functionality should be sufficiently safe as well. In this UC we focus on the specific type of platooning intended for trucks. The advantage of being in the platoon is that the trucks would drive with an optimal distance between each other, which significantly improve the platoon fuel efficiency [7]. The information about the speed, position and acceleration of the nearby vehicles is faster and more accurate to receive directly from those vehicles than to use local sensors to identify the information. Hence, by using the remote data, platooning allows for closer gap between the vehicles.

Figure 2 illustrates the software architecture model of *platoonManager* in a single vehicle. It takes as inputs *remoteData* received from the nearby vehicles, *sensorDataOwn* received from the local sensors with its own information, and *sensorDataOther* received from the local sensors about the nearby vehicles. The *ownData* is transmitted to other vehicles, while the *accelerationCMD* instructs the corresponding actuators to increase or lower the vehicles acceleration. Each non-lead vehicle is equipped with a similar *platoonManager*.

The *platoonManager* system controls the motion of the vehicle based on the inter vehicle distance from the leader and the proceeding vehicle. The *commManager* is the software component in charge of collecting and sharing the motion information with other platoon members. The information from the other vehicles is received by the *commManager*, and together with the local sensor data, is forwarded to the *rajectoryModeller* component that calculates the current gap from the lead and proceeding vehicle as well as their estimated data, and forwards this information to *longitudinalControl* component. The *longitudinalControl* then decides the appropriate (de)acceleration command to keep the desired gap.

C. Safety Analyses and Contract Derivation

The failure logic analysis of the *platoonManager* system shows that when the information from the other vehicles is late, then subtle value failures are exhibited on the acceleration command output. If the information from other vehicles is completely lost, then greater coarse value failures are exhibited. To define what late and what omission means for our platoon, we consider a fixed messages scheduling policy form each vehicle. A soft deadline can be 50ms for each vehicle to provide data and hard deadline would be 100ms. We refer to missing a soft deadline as late failure and missing a hard deadline as omission failure. A runtime contract in such scenario may say, assuming there is no more than 2 omissions within 1 second, the system guarantees that it can maintain a safe distance at 5 meters. Safe distance may mean that if the front vehicle starts full braking when it dropped two messages that the following vehicle can after third message still safely stop without causing an accident. Such contract is inherently incomplete as it depends on the accuracy of the information received, different physical properties of the unknown vehicle, the weather conditions, road conditions etc. While the contract can be tested under many conditions, it cannot be tested under all possible conditions as new collaborating vehicles may be coming to market all the time. To maintain the safety assurance of such systems, we need to constantly check for validity of the

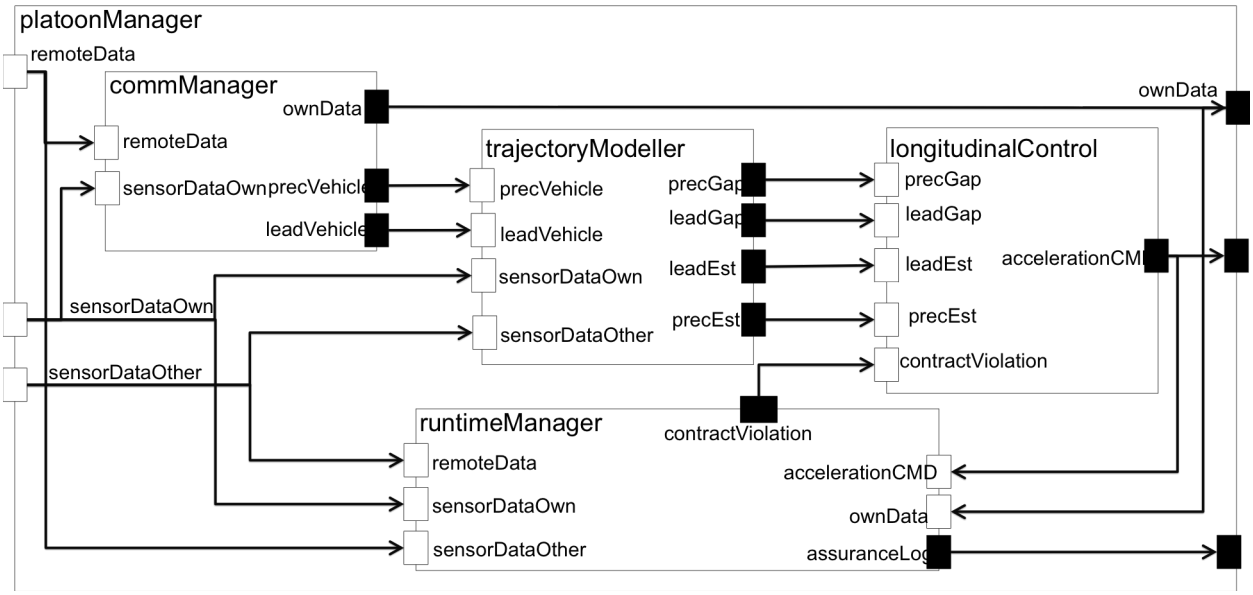

Figure 2: The platoon functionality architecture within a single (non-lead) vehicle

contracts against the different environments and evaluate the confidence in the contracts in the assurance case.

Based on the safety analysis, we specify a subset of the system contracts (Table 1). The strong contract requires the environment to provide information about speed, position and acceleration of the remote vehicle, as well as that every vehicle in the platoon must have maximum deceleration force between 5 and 8 m/s^2. This assumptions is required if the system is to avoid collision when a vehicle in front would to brake with full force. Since the time to stop in such case depends also on the freshness of the information received from other vehicles, then in the subsequent weak contracts we describe how the quality of the communication and local sensors affects the system performance. The distance to the proceeding vehicle is being adjusted accordingly, and when the *remoteData* link is unstable, then the vehicle switches to the Automated Cruise Control (ACC) mode that uses local sensor data instead of the data received from the other vehicles.

The argument-fragment in Figure 3 assures confidence in the contract <B1,H1> following the contract assurance pattern [4]. We extend the pattern by introducing the runtime manager assurance decomposing the contract completeness claim. For each of the different platoon contexts the runtime manager evaluates the completeness of the contract and feeds the results to the assurance argument where the contract completeness is either supported in the given context, or the goal serves as the counter evidence in contract completeness. Using colouring schemes such counter evidence or supports can be highlighted in the argument [8]. For example, the counter evidence can be highlighted with red indicating that contract is not sufficiently complete to be used in the given context, while green would be used to indicate support in its completeness in the given context.

Table 1: A subset of the platoonManager contracts

A_1:	*remoteData* contains (speed, position, acceleration) of the remote vehicle AND *minAcceleration* within [-8 m/s^2; -5 m/s^2];
G_1:	*accelerationCMD* $\geq$ -8m/s^2;
B_1:	*remoteData* not late or omitted more than 3 times in 1sec;
H_1:	*platoonManager* maintains the gap to the proceeding vehicle of minimum 5m;
B_2:	*remoteData* late or omitted between 3-6 times in 1sec;
H_2:	*platoonManager* maintains the gap to the proceeding vehicle of minimum 10m;
B_3:	*remoteData* late or omitted more than 6 times in 1sec AND *sensorDataOwn* not late or omitted more than 3 times in 1sec;
H_3:	*platoonManager* degrades to ACC mode and maintains the gap to the proceeding vehicle of minimum 20m;
B_4:	ACC mode active AND *sensorDataOwn* late or omitted more than 3 times in 1sec;
H_4:	*platoonManager* degrades to CC mode and maintains the gap to the proceeding vehicle of minimum 30m;

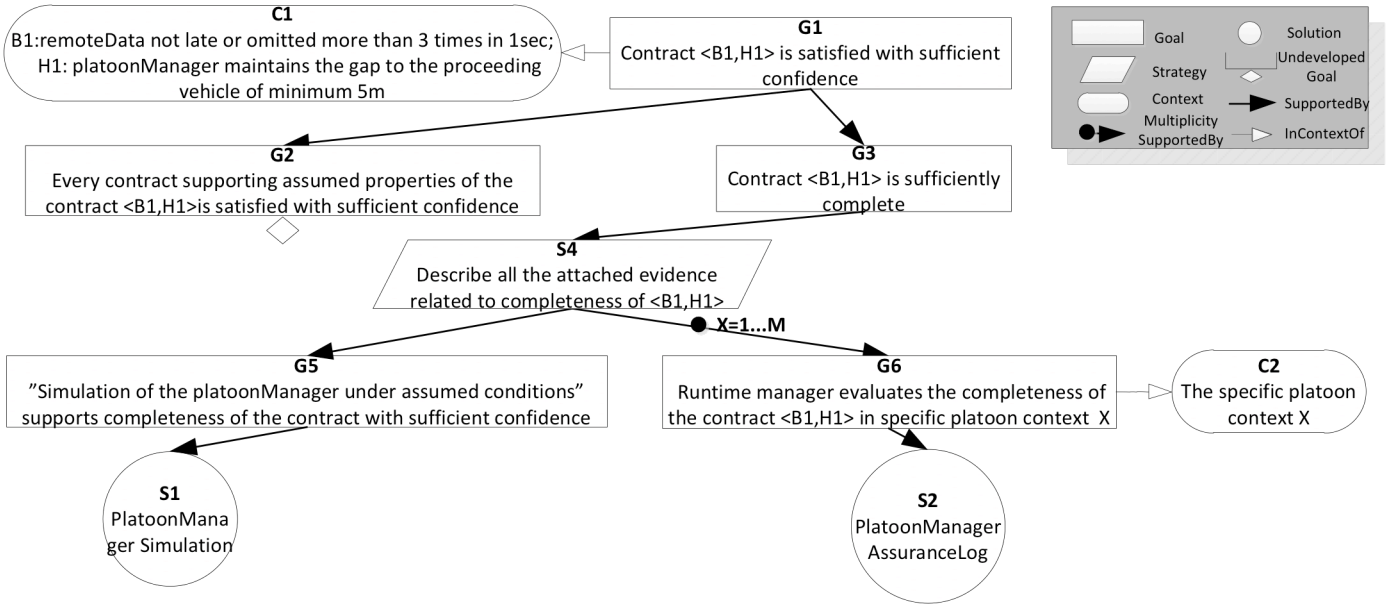

Figure 3: An argument-fragment assuring confidence in the contract <B1, H1> represented in GSN

IV. CONCLUSIONS

In this paper we have presented our initial SafeCOP results related to the role of the runtime manager in safety assurance of the truck platooning use case. To utilise the runtime manager we first need to specify contracts based on the safety analysis of both the local system as well as the overall cooperative safety function. Such contracts are first checked during design-time to establish and argue their validity. Since the context under which we check the contract changes in the cooperating cyber-physical systems such as truck platoons, we need to continuously check whether the contracts are violated and instruct the system how to behave in case of violations. The runtime manager plays the role of an advanced diagnostics component by checking whether the verified model described with contracts is in line with the actual system and its current environment. Violations of the contracts or reaching a situation not covered by the contracts should be adequately handled in the system design. We use the output from the runtime manager in the safety case to continuously evaluate the confidence in contracts.

In SafeCOP, we will not only incorporate the individual demonstrator evaluations but also we plan to perform a combined results analysis, which forms the final evaluation of SafeCOP. Some of the demonstrators e.g. the traffic management demonstrator will be evaluated in the same way with the SafeCOP solution a) included and b) absent. The differences in the results of these evaluations will give the direct evidence as to the value of SafeCOP in that particular instantiation. Furthermore, the hospital bed demonstrator develops two rather different perception systems in parallel, both of which provide information about the presence of people and obstacles in the area of interest. Comparisons between them can provide evidence that the SafeCOP safety element is relatively independent of the exact implementation of the perception system that is providing the information to the runtime manager. Other comparisons between individual demonstrator results will provide insight both into the actual functioning of the SafeCOP components and into the areas where improvements need to be made.

REFERENCES

[1] K. Nilsen, "Certification requirements for safety-critical software", RTC Magazine, 2004.

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

[3] P. Pop, D. Scholle, H. Hansson, G. Widforss, and M. Rosqvist, "The SafeCOP ECSEL project: Safe cooperating cyber-physical systems using wireless communication", In 2016 Euromicro Conference on Digital System Design (DSD), pages 532–538, Aug 2016.

[4] I. Sljivo, B. Gallina, J. Carlson and H. Hansson, "Generation of Safety Case Argument-Fragments from Safety Contracts", In 33rd International Conference on Computer Safety, Reliability, and Security, vol. 8666 of LNCS, pages 170–185. Springer, September 2014.

[5] GSN Community Standard Version 1. Origin Consulting (York) Limited, 2011.

[6] John Rushby and Paul S. Miner. "Modular Certification", Technical report, NASA Langley Research Center, US, 2002.

[7] A. Davila, "Report on fuel consumption, revision 13.0", SARTRE, Deliverable 4.3, January 2013.

[8] E. Denney, G. Pai and J. Pohl, "AdvoCATE: An Assurance Case Automation Toolset", in SafeComp Workshops, Vol. 7613 of LNCS, pages 8–21, Springer. September 2012.