Safe Cooperating Cyber-Physical Systems using Wireless Communication

Report type	Deliverable D2.2
Report name	Report on T2.2 on safety assurance conceptual model and its relations to the demonstrators and recommendations for safety committees
Dissemination level	PU
Report status:	Final version
Version number:	1.0
Date of preparation:	2018-03-28

Contributors: Carl Bergenhem (Qamcom), Ovidiu Drugan (DNVGL), Simen Eldevik (DNVGL), Hans Hansson (MDH), Omar Jaradat (MDH), Karl Meinke (KTH), Paul Pop (DTU), Sasikumar Punnekkat (MDH), Stefano Puri (Intecs), Irfan Sljivo (MDH), Henrik Thane (SIAB)

Revision history

0.1	2017-03-22	Initial Structure created by Paul & Irfan
0.2	2017-03-31	Structure revised and text modified by Sasi, Hans, Henrik & Irfan
0.3	2017-04-10	Incorporating review comments from Hans
0.4	2017-09-05	Incorporating review comments from Henrik
0.5		Final review of structure and draft among the partners
0.6	2018-02-20	Input from KTH, Intecs and MDH integrated in doc.
0.7	2018-03-05	Relations to other WPs, Conclusions added and editing (MDH)
0.8	2018-03-07	Input from DNVGL, DTU, SAFI, QAMCOM, CISTER and MDH integrated + polishing and formatting (MDH)
0.9	2018-03-20	Updates by MDH based on internal reviews by QAMCOM, GMV and SIAB
1.0	2018-03-28	Final release

Table of contents

1 Introduction

1.1 Overview of SafeCOP

1.2 Overview of the Safety Assurance Framework of SafeCOP (WP2)

A primary objective of SafeCOP is to develop a safety assurance framework for co-operating Cyber Physical Systems (co-CPS), which will facilitate their certification and market release. After an evaluation of the state of the art on safety assurance, WP2 will propose an assurance framework that can address the challenges of co-CPS by combining pre-release safety assurance with runtime monitoring. The basis for this framework will be a composable safety case, which contains "demands" placed on cooperative subsystems in order to provide safety "guarantees" for the cooperative safety function. We will also evaluate a safety analysis method called STAMP (Leveson, 2002), which is suitable for systems with a lot of interactions. This work package will also produce a set of scientifically-proven recommendations for the certification of co-CPS.

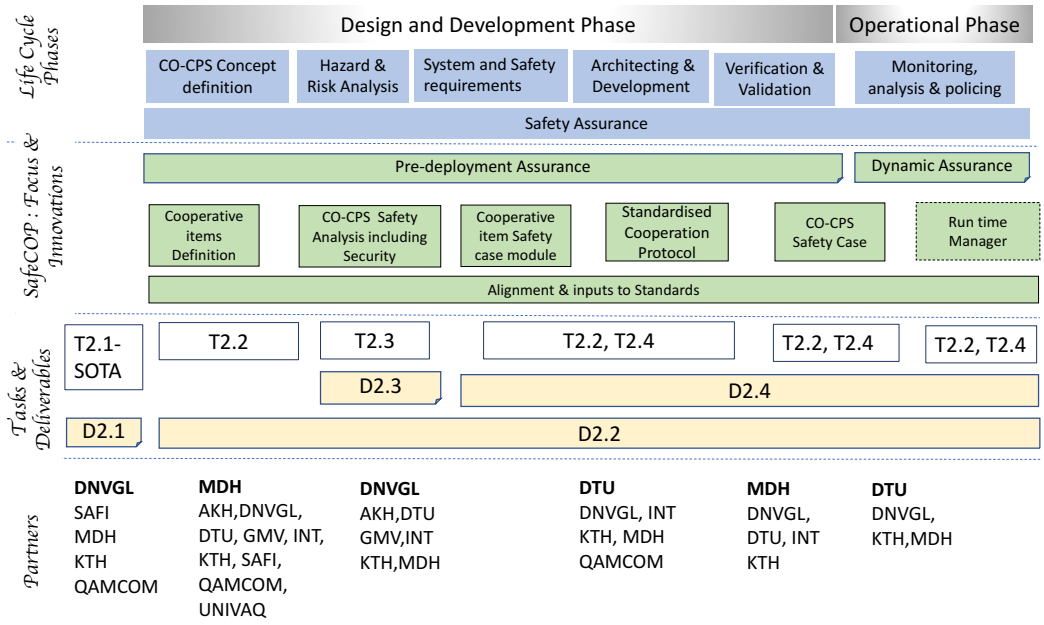

Figure 1: SafeCOP-WP2 (Safety Assurance Framework for co-CPS) Overview

Figure 1 presents an overview of the SafeCOP-WP2, indicating the main lifecycle activities in the development of a co-CPS and the various focus themes and innovations planned in SafeCOP. The figure also depicts various tasks, deliverables and main partners involved in relation to the main themes. Please note that WP2 will only address the safety relevant requirements on the run time manager whereas the development and elaboration through demonstrators will be addressed as part of WP4.

We now present a short description of the various tasks performed in WP2.

Task 2.1 State of the art on safety assurance

In Task 2.1 "State of the Art on Safety Assurance" we have evaluated and discussed a wide variety of technical approaches and methods that have been used or proposed to analyse system safety, hazards and risk. Many of the presented methodologies have been developed and used in different industries (i.e., originate from mechanical engineering) and predate the IT era. However, the cyber-

physical aspect of the Systems-of-Systems (SoS) considered in SafeCOP requires adaptation/re-interpretation of these methodologies from a software engineering perspective. The aim of the task was to create a shared consortium knowledge base on available practical frameworks and theoretical methodologies used or proposed to analyse system safety, hazards and risk. Additionally, it was used to identify significant omissions in current safety certification standards that fail to deal adequately with the distinctive features of co-CPS. The approaches we have surveyed in Task 2.1 include: simple hazard identification, quantitative risk modelling, prevention, detection and mitigation strategies, safety lifecycle models, and domain specific safety standards. The focus was on domain specific studies that are relevant to the SafeCOP demonstrators.

Task 2.2 Safety assurance conceptual model

This task develops the overall SafeCOP approach for certification of co-CPS. Based on the **state-of-the-art** (Task 2.1), methods applicable to certify co-CPS at design time are integrated end extended, and ideas of run time certification added to cope with the dynamics of co-CPS. The thus obtained **safety assurance conceptual model** is based on a **co-operative item** definition and the associated safety analysis and safety case and includes **conditional safety cases** for assurance of runtime components. It is conditional because it places demands on systems it interacts with and only then provides safety guarantees related to the cooperative safety function. The safety cases of several systems are composed both at design time (as **pre-deployment assurance**) and runtime (as **dynamic assurance**). At run time, a Runtime Manager will monitor that the "demand-guarantees" are satisfied before a component is accepted into the whole co-CPS system and is allowed to work with others, in addition to providing classical "black-box" recorder functionality. The approach proposed will be evaluated in the demonstrators in WP5.

Task 2.3 Safety analysis for co-CPS

In Task 2.3, Safety Analysis for co-CPS, we evaluate and propose some practical guidelines on how to perform a safety analysis for co-CPS. We focus the assessment of the systems on both safety and security aspects. It is based on the current state-of-the-art within safety assurance as described in the SafeCOP delivery D2.1 (SafeCOP, 2016).

The proposed safety assurance approach uses elements from the System Theoretic Accident Model and Process (STAMP; a description of STAMP analysis is included in SafeCOP deliverable D1.1), FAST and the CESAMES Systems Architecting Method (CESAM). The approach is also founded on a risk-based safety philosophy, where risk is understood as the consequences of an activity, with associated uncertainty. The guidelines outline the different stages of the safety analysis and includes examples on how it was used by the different use cases in SafeCOP. The main stages are:

- Operational Analysis: outlined how to establish the mission objectives and the mission constraints, i.e., define the targets and boundaries for the system performance.
- Functional Analysis: establish which functional requirements must be in place to ensure that the mission requirements (mission objectives and constraints) are fulfilled
- Resource and control structure: establish which resources and control structures that are needed to ensure that the identified functional requirements can be met.

We have also evaluated different tools to support a model-based design of systems to achieve the desired safety levels; and provided an overview of different safety analyses for co-CPS (reported in

D2.4), including tools for safety analysis, architectural modelling and analysis, assurance case modelling and their integration[1].

The safety assurance conceptual model will also be extended to cover security concerns; including taking the security work in WP3 into account. The main concern addressed is the lack of integration of safety and security work; typically, safety and security have separate standards, have independent processes, and are performed by separate groups of people. Specifically, security concerns are not covered in any details in safety standards, potentially resulting in systems that are successfully certified according to relevant safety standards, but that still are open for security threats that may jeopardize safety. This task will propose suggestions for integration of safety and security process with emphasis on introducing structured approaches to consider security concerns in the safety work.

Task 2.4 Conditional composable safety case models

The safety assurance is supported by tools which use safety case models. In SafeCOP, the safety case models are contract-based, and as mentioned earlier, they are conditional (due to the demands) and composable (both at design-time and runtime). Task 2.4 is concerned with proposing contract-based safety case models for the SafeCOP safety assurance concept. These are discussed in deliverable "D2.4 Report on T2.3 and T2.4 and its relations to demonstrators, and recommendations for safety committees".

1.3 Content, Aims and Relevance of this Report for the SafeCOP project

This report presents the SafeCOP safety assurance conceptual model, and provides thus important basis for WP3 (wireless networking) by providing general requirements and indicating assurance related information, WP4 (runtime system) by identifying the role of the runtime-related assurance in the overall approach, as well as the interactions between the run-time system and the assurance, WP5 (Use-cases) by providing a baseline and framework for the concrete assurance activities that will be performed within the use-cases, and WP6 by providing basis and input to standardization.

1.4 Overview of the Report

In Section 0, we present Safety Assurance conceptual model for co-CPS. Section 2 is subdivided into sections on the main challenges (Section 2.1), research questions (2.2), a general overview of the approach (2.3), additional detail on the idea of the approach (2.4) , model-based design supporting assurance case specification (2.5), our exploration of the most relevant safety standard in our context (i.e.) ISO26262 with respect to applicability in co-CPS safety certification as well as the process, mapping and extensions proposed (2.6), determining impact of system changed and how safety cases can be maintained to support continuous assurance (through-life cycle) (2.7) and overview of the SafeCOP run-time Manager (2.8) . Section 3 presents the relation of WP2 with other work packages followed by conclusions in Section 4. In addition, two appendices provide information regarding multi-stake holder assurance from a business perspective and brief introduction to temporal logic, respectively : Appendix A: A Business Model for selling components with safety certificates and Appendix B: Linear Time Temporal Logic (LTL).

[1] Related relevant information has also been collected and studied in our sibling project AMASS; reported in the AMASS state-of-the-art deliverables D3.1 "Baseline and requirements for architecture-driven assurance" (https://amass-ecsel.eu/sites/amass.drupal.pulsartecnalia.com/files/documents/D3.1_Baseline-and-Requirements-for-Architecture-Driven-Assurance_AMASS_final.pdf).

2 SafeCOP Safety Assurance Concept for co-CPS

2.1 The main challenges

When a system might harm humans or the environment (or are intended to mitigate or manage such harm), decision-makers (such as top management, assessors, certification authorities etc.) require pre-release safety assurance evidence that it manages risk acceptably (since such assurance provide liability-related evidence of "innocence" after an accident and could also be mandated by regulations). In most cases, developers must show that their processes and work products conform to a relevant standard; in some cases/domains developers prepare an explicit safety case combining the evidence with a safety argument (for the purpose of this document, we call the safety evidence a "safety case", even if explicit safety cases are not used – the work in SafeCOP applies also for such cases). The conceptual basis for certification is that the pre-release (design-time) evidence anticipates the possible circumstances that can arise from the interaction between the system and the environment, to show that these interactions do not pose an unacceptable risk. Certification is very expensive and can add a development cost overhead of 25 % to 100% (IBM Software Rational, 2010).

The development of co-CPS poses challenges that are not adequately addressed by existing practices. While careful safety-aware design and thorough safety assurance is required, no single manufacturer has design authority over or responsibility for the safety of a cooperative embedded system. Developing a safety critical system typically requires making design decisions that trade-off safety concerns, functionality, cost, and other considerations. Achieving adequately safe cooperative cyber-physical systems requires arriving at, realizing, and assuring a safe design even though participants in the design process are competitors reluctant to share all of their concerns or intricacies of designs with each other. Moreover, due to the cooperative and open nature, many circumstances which have to be covered by the pre-release safety assurance are difficult to anticipate at design time in the case of co-CPS.

As there are multiple stakeholders with responsibility for the different constituent systems, safety assurance need to be handled in a modular way, and able to cope with the inclusion of new constituent systems after the initial deployment.

Considering security is essential, as security concerns are not covered in detail in current safety standards, potentially resulting in systems that are successfully certified according to relevant safety standards, but that still are open to security threats that may jeopardize safety.

2.2 The research questions

In SafeCOP, the requirements coming from the 5 use cases are fed into three research work-packages that provide prototype solutions for the identified challenges. This is an iterative process: the technical solutions need to be evaluated through demonstrators, corresponding to the scenarios, and this generates the next input to the technical work. The requirements are important in supporting this process and clarifying the interrelation between demonstrators and technical work. The identification of research questions has been performed/refined through such an iterative process.
From the WP2 research perspective (Assurance of Safety and Security in co-CPS) the high-level question is 'Are the existing state-of-the-art and practice assurance techniques adequate for assuring safety and security of co-CPS and if not, how to extend the SOTA to address this issue?'. This can be further broken down to following important research questions and if required, how to provide directions for addressing the same:

RQ1. Do the **safety standards** support development and certification of co-CPS?

RQ2. Do the existing **pre-deployment safety/security analyses** provide support for multi-attribute co-CPS reasoning (here 'multi-attribute' means safety & security)?
RQ3. Are the **safety/security standards** adequate to support development and certification of **both attributes simultaneously**?
RQ4. Are the existing safety case assurance techniques (academic, as well as those mandated by applicable safety standards) adequate for **composable, conditional and continuous safety assurance**?
RQ5. Are existing runtime fault diagnostics techniques adequate for **dynamic safety assurance**?

It is imperative that one or more use-case driven requirements will directly/indirectly address these research questions and enable their evaluation through the demonstrators.

2.3 Overview of the approach

In subsequent discussions, we take the example of platooning to describe and clarify the challenges as well as the overall safety analysis requirements in the context of co-CPS. co-CPSs are essentially a subclass of System-of-Systems (SoS) with stringent requirement on their Safety and other extra functional properties often mandated directly or indirectly by domain specific standards and regulations. We will hence be using these two terms (co-CPS and SoS) interchangeably. A platoon typically consists of multiple vehicles made by different vendors which join or leave as per requirements. Even though each of the individual vehicles may be safety certified independently, there are a large set of emergent conditions which make the system of systems vulnerable (e.g. intrusive vehicle could compromise the system by compromising communicated data). Typically, the designer/integrator of such an SoS will need to define and follow a System life cycle with adherence to relevant standards. As depicted in Figure 2 following a V-model, the starting point is a generic SoS definition, followed by Hazard/Risk Analysis, Safety Functions definitions, and Safety goals & Safety requirements definitions. Once the system architecture has been defined, the hardware development and software development typically follow their own lifecycles. The right side of V-model includes verification and validation and culminates with the safety case. In the context of SoS the safety case is anticipated to include both a static part and a dynamic part.

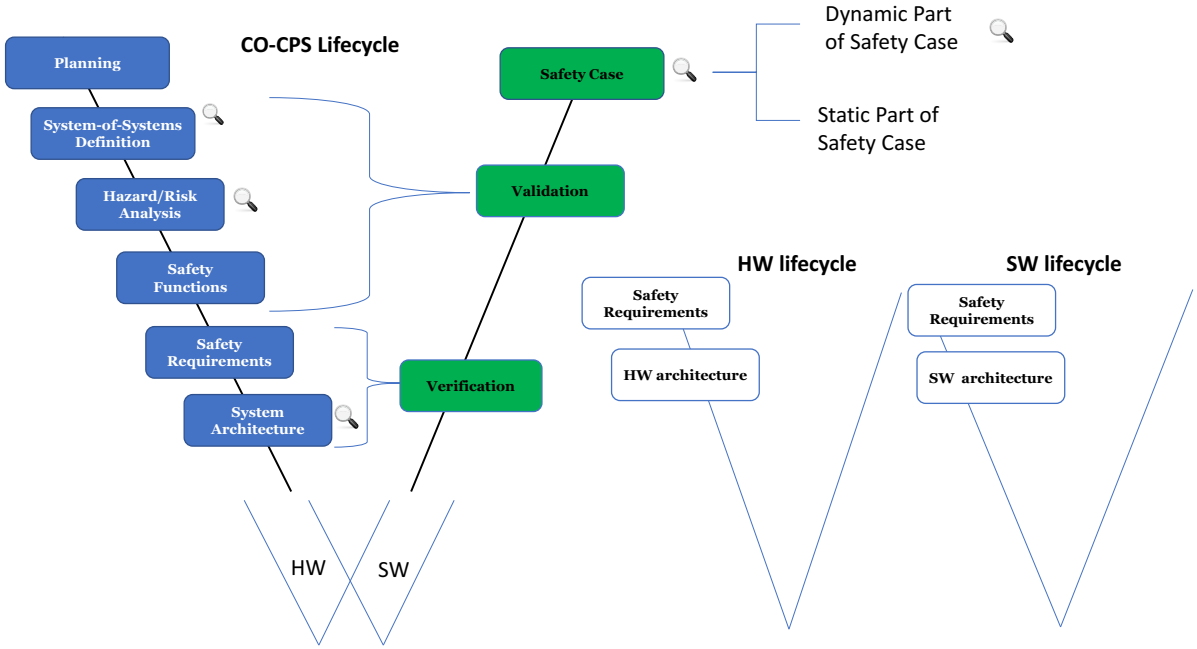

Figure 2: System of Systems(SoS) Life Cycle

We envisage that significant efforts are essential in SoS Definition, Hazard Analysis, System architecture and Safety Case in order to progress the state-of-art and state of practice in ensuring safety of SoS (highlighted in Figure 2 with a looking glass icon).

There will be many cooperative functions required to be defined and developed in SoS. Typical items in a vehicle are vehicle-level functions like Steering, Brakes, ABS, Cruise Control, etc., that have high coupling in terms of being interconnected via wired networks, sensors and actuators. These items in their vehicle context are usually assessed and certified according to ISO26262 before use. Items designed for a single vehicle, can hierarchically be built into more complex items (e.g., a cruise control built on top of existing items like Brakes, Throttle and elements like Speedometer, and Radar). Here the main innovation will be to adapt the concepts defined for tightly-coupled systems as envisaged in ISO26262 to a more loosely-coupled SoS environments.

Conceptually, we view the SoS as a single product ("a superset vehicle") consisting of a set of constituent systems (vehicles). The cooperative function is a loosely coupled Item (using wireless networks) within the SoS that relies on other items and elements that reside in different constituent systems (vehicles). The SoS can thus be described using a hierarchy of items in a similar fashion as for in-vehicle items, but with the difference that the SoS-level Item is loosely coupled via wireless networks.

We assume that each vehicle in the platoon has followed certification and has a vehicle level safety case attached to it. But just the safety case for each vehicle will not be sufficient for them to be accepted to join and collaborate in the platoon.

For safe functioning of a platoon or any other co-operating cyber-physical systems, one needs to establish a higher (SoS) level safety case, though the current standards do not address them explicitly. We envisage such safety cases to consist of both static (off-line) and dynamic(on-line) parts. The static part could be considered as the one we derived in the traditional way and have to be made before deployment and. The static part essentially relates to aspects of the individual vehicle (including its possible roles in the platoon), whereas the dynamic part is regarding cooperative functions implemented at the SoS level and admission control aspects. For traditional systems, safety requirements are assured in the context of a given environment. But for open and adaptive systems such as co-CPS, not all context variables can be defined and considered during system development, as the boundary of the environment cannot be fully established beforehand. Hence, there is need to evaluate their validity continuously, even during runtime (Medawar S., 2017). Since not necessarily everything about a requirement needs to be re-evaluated, but only certain aspects, we enable tagging of the related component contracts realizing those requirements with a "runtime" flag. A contract with a runtime flag should be evaluated during runtime, i.e., whether its assumptions are met, and whether the component implementing this contract actually provides the guarantees when the assumptions are met, as well as SoS safety requirement and emergent properties.

Monitoring and logging of all critical information during runtime as well as while allowing vehicles to join and leave the platoon are essential in platoons. This allows one to do post-mortem analysis if any untoward incident happens or safety is compromised. Typically, each individual aspirant vehicle will be subjected to certain verifications before their admittance to the platoon. These verifications will be based on (as well as be part of) the dynamic safety case. These verifications could for example be (a) to make sure that all have compatible versions of software or (b) to make sure that the certified vehicle software and hardware have not been modified inadvertently (or maliciously) etc.

2.4 The general idea

Figure 3 presents the SafeCOP safety assurance concept; introduced in (Pop 2016) and (Pop 2017). The approach in SafeCOP is to *restrict the behaviour* of the cooperative safety function at *runtime*, such that the *design-time* safety assurance evidence, with additional *monitoring at runtime*, is able to guarantee the safety requirements. Such an approach may require changes to the functional safety standards, hence a key objective of SafeCOP is to contribute to new standards and regulations. Standardization will be prompted by the SafeCOP project partners that are *safety assessors* (DNV GL, Safety Integrity, DTI), as well as members of standards committees. Additionally, the project is strengthened by an external advisory board, comprising people with vast safety assurance and security-related expertise. They will make sure that the innovations developed in SafeCOP are grounded in current certification and industrial practice and are aligned with the current efforts in the technical committees tasked with extending the certification standards.

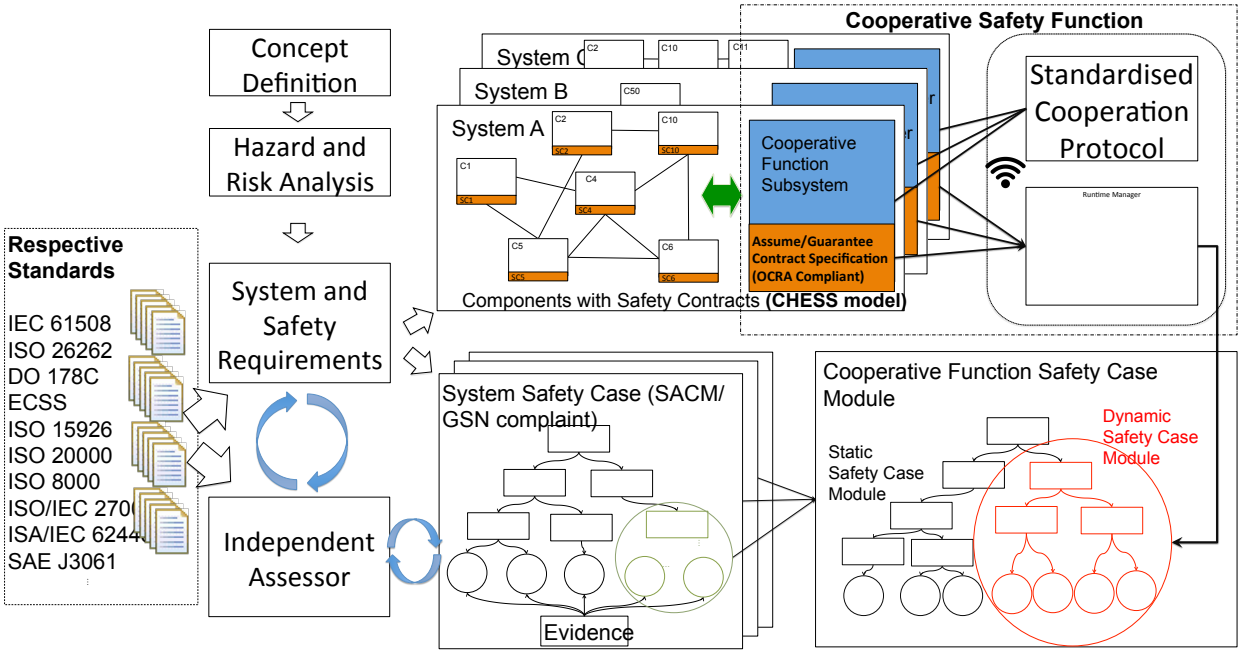

Figure 3: The SafeCOP safety assurance concept

As depicted in Figure 3, The SoS concept definition is the starting point followed by Hazard and risk analysis to decide upon the System and Safety goals. The Hazard analysis at the SoS level may not be a simple integration of outputs of the individual system hazard analysis, but will demand new perspectives to unearth emergent hazards as well (Baumgart 2017). Once the system and safety requirements are specified, activities related to the development of various subsystems and corresponding safety cases starts. System safety case and cooperative function safety cases are generated in an iterative manner based on evidences gather over development and V&V phases, which then are presented to an independent assessor (or certification body) to check their compliance with the relevant domain specific standards.

Figure 3 shows three CPS systems, A, B and C, developed by three different organizations. Note that the systems (actually Systems-of-Systems) addressed in SafeCOP may consist of several cooperating cyber-physical systems where each of those systems has its own accompanying safety case. Each system has a cooperative subsystem co-responsible for the cooperative safety function, re-

ferred to as a cooperative function in Figure 3. In SafeCOP we consider that the design, or even implementation, of certain subsystems of the cooperative functions are developed independently of the systems in which they are used (e.g., systems A, B, and C). Such independent development of safety-relevant subsystems is supported by different standards, e.g., in ISO 26262 through the notion of Safety Element out-of-Context (SEooC) - (Sivencrona, Johansson 2007). In SafeCOP we intend to use contract-based design (Benevniste, 2012) (Soderberg 2013) to facilitate the independent development of cooperative safety functions. We will build upon results on contract-based design already available in CHESS (CHESS, 2012b) (CHESS, 2016) (CHESS, 2018), an open source methodology and tool for model-driven development of high-integrity systems focused on component and contract-based design. CHESS is the result of different ARTEMIS projects such as CHESS (CHESS, 2012a), SafeCer (SafeCer, 2016) and CONCERTO (CONCERTO 2015) (CONCERTO, 2016).

In particular, the components in the CHESS toolset can be annotated with assumption-guarantee contracts and verified using the OCRA tool (OCRA 2016) for checking contract refinement (FoReVer 2014) (see Chapter 2.5 for more information on CHESS). In SafeCOP, we will use such contracts for both design-time and runtime contract checking. Our solution in SafeCOP is to enrich the CHESS toolset by providing support for development of co-CPS. More specifically, the tool will distinguish between the runtime and design time contracts and tightly couple the contracts with the corresponding safety assurance evidence to address the dynamic safety assurance nature of co-CPS. The communication between the different co-CPS in our cooperative function is wireless (e.g. Vehicle to Vehicle; V2V). The wireless communication subsystem is an example of an independently developed safety-relevant element that is a part of the cooperative function. SafeCOP will extend the current state-of-the-art wireless protocols by creating an application-level library and related API that acts as a "safety layer" on top of the existing protocols. If this API is used for the communication, we guarantee that the communication has "high integrity", i.e., trust is provided that the contents of messages are not corrupted either unintentionally or intentionally. This is needed because otherwise our cooperative function A cannot trust messages from the other systems (B and C) to implement its safety function. In providing such high-integrity communication, we will in addition to considering traditional safety concerns, reuse security results from other ARTEMIS projects such as DEWI and from the Cooperation Reference Technology Platform (CRTP), and our focus is on delivering a solution that is not susceptible to security threats, such as man-in-the-middle attacks Traditionally, an organization prepares their safety assurance evidence for the safety case at design-time. The notion of a cooperative function extends the scope of the safety case from a single system to other systems that are not necessarily known at design time. To provide additional assurance of such open systems, runtime safety assurance supported by runtime verification is also required. In SafeCOP, the safety case of the system A prepared by organisation A includes a safety case module related to the cooperative function, which covers not only the system A, but also the cooperating systems B and C. Since each organization is interested in protecting their Intellectual Property (IP), it is not always practically feasible to include the detailed evidence from other organizations (further reasoning related to such business aspects of safety components are provided in Appendix A). Hence, in SafeCOP, the cooperative function safety case module prepared by the organization A is done by composing the evidence of the cooperative subsystem A with the public evidence from the cooperative subsystems B and C (i.e., without exposing the IP of the organizations B and C). This is similar to modular certification allowed by certification standards, e.g., SEooC. The difference is

that modular certification typically does not hide the IP and does not address runtime safety assurance. We also refer to this as composing safety cases, since the safety case for the cooperative function will be based on the individual safety cases for subsystems A, B, and C.

2.5 Model based design supporting assurance case specification

Model based design is a well know practice to efficiently support CPS systems development, from requirement specification through system design, verification, validation and implementation. The assurance case definition can also benefit of the model-based design activities (architecture-driven assurance).

CHESS is a model driven methodology for the design, verification and implementation of cyber-physical system. CHESS comes with a dedicated modelling language (CHESSML), a UML/SysML/MARTE profile, offering a component-model contract-based approach, and an open source tool released under the Eclipse Polarsys ecosystem[2]. System components in CHESS have contracts associated, where each contract comes with assumption and guarantee properties. The guarantee tells about the behaviour of the component when it is executing in a given environment; such behaviour can be trusted only if the interaction with the environment behaves as specified by the contract's assumption property. A given contract can be tagged as strong or weak; basically, a component is not allowed to run in a context where the assumption of a strong contract does not hold. If the assumption of a weak contract does not hold for a component running in a given system, the component is still allowed to be part of the system but the guarantee coming with the weak contract itself cannot be taken into account, for instance it cannot be used as evidence to prove about some system level properties.

Contract assumption and guarantee properties represent the formalization of safety requirements (in CHESS, contracts can be traced to SysML requirements), so the contract properties inherit the criticality level of the corresponding requirements.

By using the LTL formal language for the expression of contracts assumptions and guarantees, CHESS allows to apply verification facilities about contracts by using seamless interoperability with the OCRA verification tool. Examples of the aforementioned facilities are formal verification of contracts refinement along the system components refinement and formal verification of components bindings by checking the satisfaction among components contracts assumptions vs guarantees properties (so to verify that what is assumed by a component is guaranteed by the collaborating components).

It is worth noting that in order to enable the usage of the aforementioned formal verification facilities as a mean to prove the correctness of the system with respect to considered safety requirements, evidences must be provided regarding the verification of the contracts implementation. In particular, it must be proved that a (software or hardware) component implementation behaves as specified by the associated contract's guarantee, by assuming that the environment behaves like specified in the contract's assumption. Such evidence must be derived according to the type and rigor of the test techniques that need to be applied, in particular depending on the criticality level associated to the formalized requirements and the applicable standards. Currently testing and verification techniques

[2] https://polarsys.org/

to check component implementation with respect to contract specification are not in the scope of the CHESS toolset support[3].

The information available in the CHESS model and the different verification activities performed on it allows to support the system assurance case definition, so the argumentation proving that the occurrence of unacceptable risks has been properly mitigated. In particular, in the assurance case, it is possible to argue about the safety of the system with respect to a given hazard by reasoning about the contracts provided in the system model and about the verification activities performed upon the contracts themselves; under some assumptions parts of such assurance case can also be derived in a semi-automatic way starting from the information attached to the contracts (Sljivo, 2016)(Sljivo 2018). This line of activity is related to multiple work products in the safety standards such as ISO26262 and spawns verification efforts at different levels in the V-Model. This way of structuring/deriving the assurance case is currently investigated in the AMASS ECSEL JU project. AMASS is proposing an environment (OpenCert) to model different information that can be of help for the different stakeholders of an assurance project. AMASS promotes the usage of CHESS for the architecture specification, so for the modelling of the components and the associated contracts; moreover, a SACM (OMG, 2016) like metamodel is used to be able to model the assurance case specification in a GSN way. Then, links between architectural entities and assurance case entities (e.g., between contracts, claims and evidences), are enabled by dedicated traceability tool support. The aforementioned architecture-driven assurance focused on component and contract-based approach is used as baseline in SafeCOP; as extension to this baseline, new requirements can be derived with respect to the modelling language capabilities, by considering to the information that should be addressed during the design of collaborating CPSs:

- **Support for collaborative scenarios**. For each collaborative function, a collaborative function model should be provided to analyse the functional requirements first; then a logical architecture (collaborative system model) should be identified by defining the roles (to be played at run-time by actual systems) that can be part of the co-CPS and the functions that they have to implement. A role is not a physical entity; it allows to focus on the set of properties an actual system should have in order to support the collaborative function only. The possible roles of a collaborative system model should be identified together with their interfaces (input and output ports, provided and required service ports). For instance, in the platooning example, a collaborative system model could comprise one role (with possibly multiple occurrence) played by the follower vehicle, one role for the leader, and input/output ports attached to the roles to represent the information exchange (e.g. about the speed, position and acceleration of the vehicles). Then valid interaction sequences (i.e. protocols) concerning interfaces usage should be provided, to define appropriate constraints on the interactions themselves (so to limit the occurrence of emergent and uncontrolled behaviour). The goal of the collaborative system model definition is to formalize the set of requirements that a given actual system should meet in order to be allowed to join a cooperative scenario related to that given collaborative function.
- **Support for run-time contracts**. Contract based design should be considered during the design of the collaborating functions, to then support the assurance cases related to the collaborative functions. Contract based design should be supported at the level of the collaborative system model; so, contracts constraining the behaviour expected/guaranteed on the declared interfaces could also be defined for each

[3] In the context of the AMASS Ecsel JU project there is an ongoing extension which regards the verification of the component contracts with respect to the state machine associated to the component itself. This feature could be used in the future to (partially) support the verification of the contract implementation by relying on an automatic generation of the component implementation (part of it) starting from the associated state machine behaviour

role of the collaborative system model. In case of co-CPS, the system is "built" at run-time, i.e. the systems composing the system-of-systems are composed dynamically. So, regarding the contract-based approach, the validation of the run-time contracts (i.e. contracts related to the collaborative functions) stating how each system can interact with the others cannot be provided at design time. What can be stated statically is the validation of the contracts out of any particular context, so by assuming that the assumptions of the contracts hold; the validity of these assumptions has to be proved at run-time. In the model, it should be possible to distinguish between static and run-time contracts; a run-time contract should be linked to the part of the assurance case which requires collection of evidences at runtime, to reflect that the contract cannot be (fully) proved statically.

- **Operational modes**. A collaborative function could be performed in different operational modes (for instance, in the platoon scenario, different modes could be implemented for the cruise control for platooning collaborative function, like adaptive cruise control and cooperative cruise control modes; having impact on e.g. the safe distance between the vehicles); in this case one collaboration scenario should be modelled for each operational mode, to show the interfaces and contracts involved in the given mode. Operational modes (states) should be modelled through a dedicated state machine, to be provided for the collaboration role/system, with the specification of the conditions enabling the transitions between different modes.
- **Link to the platform (run-time manager)**. The information about run-time contracts available in the model could then be used to automatically instrument the platform, i.e. to initialize the run-time manager with the contracts to be verified at run time. The run-time manager should evaluate the contracts according to the state of the system, i.e. the mode currently active, and according to the information coming from the environment, so to be able to verify the contracts assumptions.
- **Specify the safety behaviour to be executed in case a run-time contract becomes invalid at run-time** (for instance due to an internal/external failure or change in the environmental conditions). The occurrence of failures, and corresponding recovery actions, should be considered in the operational mode state machines, to then specify the recovery actions to be enabled. The detection of a failure regarding the validation of a contract (performed by the run-time manager), should be represented in the state machine as an event enabling some recovery/control actions and a mode transition (e.g. to activate a degradation mode).While performing a collaborative function, some though can be argued about the kind of recovery actions that can be enabled according to the kind of contract which has become invalid. If a strong contract of a system becomes invalid at run-time, then the system has to leave the cooperative mode, so the current enabled collaborative function has to be ended; it is worth noting that the termination of a cooperative function could be done by initiating an ad-hoc cooperative function, if possible, for instance to allow the system to leave the cooperative mode in a safe condition. If a weak run-time contract associated to a collaborative function is no more satisfied (environmental changes, internal or external failures), a change of the operational mode could be enabled to reconfigure the system. Due to a change of the environment condition, the weak contract(s) which become invalid could be replaced by different weak contract(s) which have been specified for the system. One possible goal is to continue running the same cooperative function but with different extra functional properties; for instance, in the platooning scenario, due to a change in the weather condition affecting the brake performance, the follower vehicle could decide to increase the minimum distance to be kept from the leader vehicle (at the cost of decreasing the performance of the platoon, so by increasing the fuel consumption). In case of reconfiguration and then by changing the set of contracts which become valid for the collaborative function, the dynamic part of the safety case should be rebuilt by considering the new set of active contracts.
- **Collaborative system design**. An actual collaborative system should be designed by declaring the supported collaborative system models and in particular the roles that it can play, and so by supporting the interfaces, operational modes and contracts associated to the implemented roles. In case the analysis of all the possible combinations of the configurations for a collaborative function is not feasible (for instance due the complexity of all the combinations), the run-time contracts could not be derived during the design and analysis of the collaborative models; in this case the collaborative model should at least

comprise the set of roles and the interfaces, the latter allowing the exchange of information between the different collaborative systems. In this case, a given system should be designed by guaranteeing a certain level of variability, represented by a set of weak run-time contracts associated to the collaborative functions. Then run-time support should be used to check about the compatibility of the run-time contracts for the involved collaborating systems, so to check the compliance between the assumption and guarantee of the contracts of the different systems.

- **A safety case should be modelled to prove that the safety goals are not violated by any collaborative function and operational mode**. The information available in the modelled collaborations should then be use at runtime. For instance, at run-time, in order to initiate a collaborative function or let a system join an initiated collaborative scenario, a check should be performed between the collaborating system in order to understand the collaboration/operational mode to be activated and the role played by each system. According to the activated collaboration-operational mode, the corresponding safety case statically defined has to be filled with the information (e.g. run-time contracts) coming with the given cooperative systems and with the evidences resulting by the validation of the run-time contracts.

2.6 Cooperative Safety Function – Loosely Coupled Item in ISO 26262 Terminology

In Section 2.4 we presented the general idea of our safety assurance framework, and in this section, we elaborate on that idea more concretely in the automotive domain with the vehicular standard ISO 26262. We are focusing on this standard since it provides a consistent terminology and since is relevant for the platooning example used in this report to illustrate our approach. Still, our approach can be straightforwardly generalized (mapped to) other safety standards, such as IEC61508:2010, ISO13849-1:2016, IEC 62061:2005. Functional safety in cooperative systems was also investigated in (Nilsson et al., 2013).

Conceptually, we view the SoS as a single product ("a superset vehicle") consisting of a set of constituent systems (vehicles). The cooperative function is a loosely coupled Item (using wireless networks) within the SoS that relies on other items and elements that reside in different constituent systems (vehicles). The SoS can thus be described using a hierarchy of items in a similar fashion as for in-vehicle items, but with the difference that the SoS-level Item is loosely coupled via wireless networks. Our assumption is that the base functionality of the constituent systems is already certified (e.g. the vehicles are certified according to the ISO26262 standard), and that the cooperative function has (at least conceptually) a two level realization and certification, with the first level as an add-on to the base functionality implemented in the constituent systems (i.e., a proxy with a wireless interface + control of local items) and the second level implemented in the SoS infrastructure outside the constituent systems (e.g., traffic congestion arbitration). Although possibly based on and relying on the base functionality of the constituent systems, the cooperative function essentially has a distributed implementation with cooperating elements in SoS infrastructure and the constituent systems.

By introducing the notion of Loosely Coupled items (LC-items) and using safety contracts we propose a modular partitioned approach for safety assurance consisting of the following steps:

1. Safety assurance/certification of the base functionality of the constituent systems according to current state of practice as required by applicable safety standards (e.g. ISO26262).

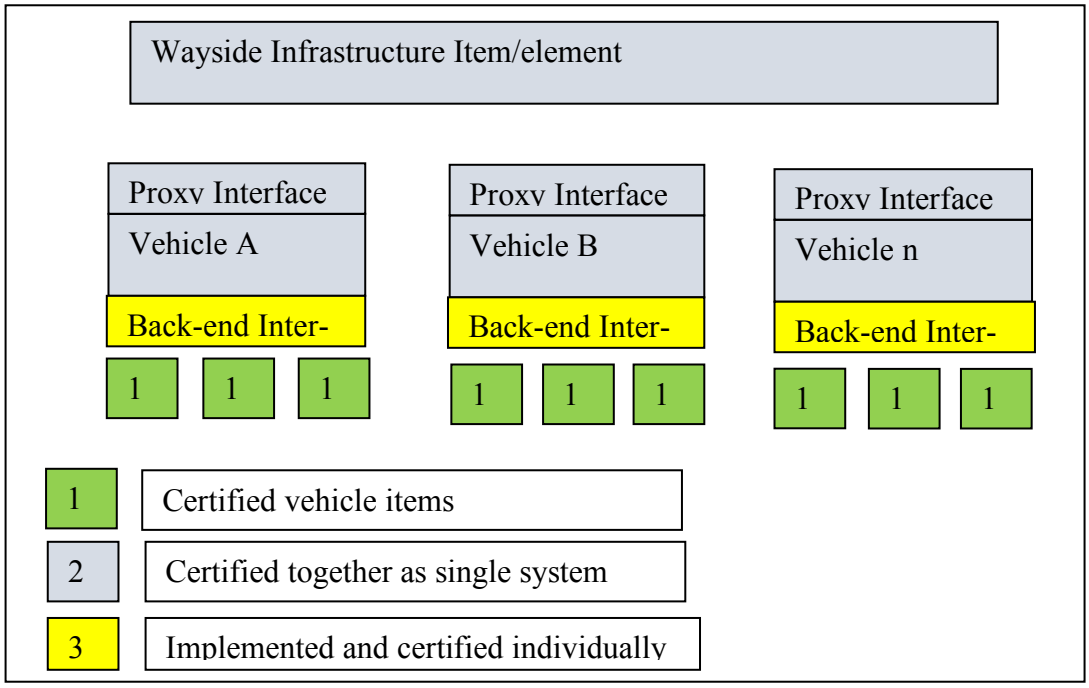

Figure 4 Platooning SoS LC-Item illustration

2. Safety assurance/certification of the cooperative function, which is considered as an LC-Item on the SoS level. This LC-Item will have a proxy interface and logic requiring the existence of a number of items in the SoS infrastructure and/or the constituent systems. Since this part of the LC-Item will run on many different constituent systems (vehicles) developed by different manufacturers it is likely that harmonization and/or standardization of the proxy interfaces as well as the local back-end interface for controlling the required items are necessary.
3. Safety contracts are used to specify the safety requirements on these items, i.e. the properties that these items are required to fulfil. These safety contracts are then used to support the safety assurance/validation/certification of the implementation of these interfaces, which will be necessary for each individual system (vehicle manufacturer).

Figure 4 presents the LC-item architecture where the LC item is distributed across vehicles and builds upon vehicle specific items. The corresponding safety case architecture is presented in Figure 5. The cooperative function (LC item) safety case module, as presented in Section 2.4, is composed of the common safety case module which addresses the part of the LC item certified together as a single system. The common module builds upon the parts implemented and certified individually (e.g., the back-end interface component), as well as the vehicle specific item modules. Since some information presented in the vehicle specific item modules are not to be shared outside the vehicle manufacturing company, public interface is specified and captured in the safety contracts between the “private” vehicle specific item modules and the common LC item module. The system safety case for a particular vehicle is composed of the cooperative function safety case module and the corresponding vehicle items module.

Safety contracts have been used to capture and validate context assumptions essential for the Safety Element out-of-Context concept from ISO 26262 to support integration of independently developed components (Sljivo 2015) (Sljivo 2016b). We utilize the contracts here in a similar way, i.e., to support the separation of concerns and independent development of the different vehicle items by ensuring that the context assumptions regarding the LC item are captured and validated in all vehicles.

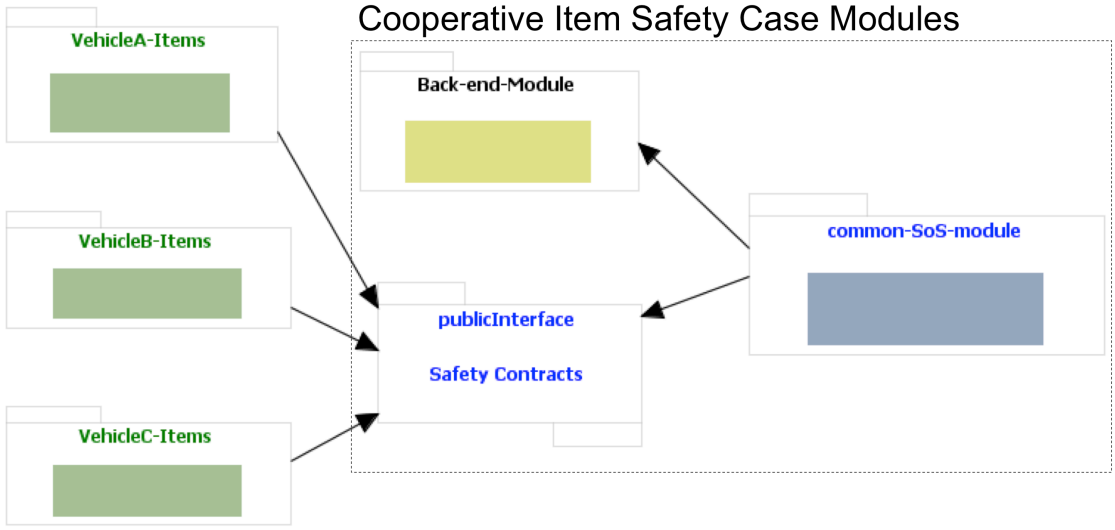

Figure 5:Platooning item Safety Case Architecture

2.7 Impact of changes and maintenance of Safety cases through co-CPS lifecycle

Safety assurance should provide justified confidence that all potential risks due to system failures are either eliminated or acceptably mitigated (Jaradat 2016). The development of safety cases has become common practice in many safety critical system domains to communicate and demonstrate the status of a system safety. For instance, to demonstrate the sufficiency and credibility of the required confidence in safety, safety cases shall always communicate the actual safe performance of a system and always contain only acceptable items of evidence that this system meets its safety requirements. Hence, safety cases are built as living documents that should always be maintained to justify the safety status of the associated system and evolve as these systems evolve.

However, safety critical systems are evolutionary and subject to preventive, perfective, corrective or adaptive changes during their lifecycle. Since the collected items of evidence are valid only in the operational and environmental context in which they are obtained or to which they apply, system changes might affect the operational assumptions which in turn can invalidate the available evidence and confidence. For example, there could be a divergence between the communicated system safety by the safety case and the system safety in actual operation. This divergence can undermine the confidence in safety and thus should be eliminated by a maintenance process through which developers decide the optimum of two possible options: 1) a corrective change so that the system safety in actual operation meet the communicated system safety by the safety case, or 2) an adaptive change so that the communicated system safety reflect the system safety in actual operation.

In this context, we proposed a framework (SANESAM) (Jaradat 2017b), presenting a sensitivity analysis-based technique which aims at measuring the ability of a system to contain a change (i.e., robustness) without the need to make a major re-design. The proposed technique exploits the safety

margins in the budgeted failure probabilities of events in a probabilistic fault-tree analysis to compensate for unaccounted deficits or changes due to maintenance. The technique utilises safety contracts to provide prescriptive data for what is needed to be revisited and verified to maintain system safety when changes happen. We also provided a detailed use case to explain more in-depth the application of the technique (Jaradat 2017).

In order to maintain the safety confidence under changes, system developers need to re-analyse and re-verify the system to generate new valid items of evidence. Identifying the effects of a particular change is a crucial step in any change management process as it enables system developers to estimate the required maintenance effort and reduce the cost by avoiding wider analyses and verification than strictly necessary.

We also provided a technique which uses runtime monitoring in a novel way to detect the divergence between the failure rates (which were used in the safety analyses) and the observed failure rates in the operational life. The technique utilises safety contracts to provide prescriptive data for what should be monitored, and what parts of the safety argument should be revisited to maintain system safety when a divergence is detected (Jaradat 2018).

2.8 SafeCOP Runtime Manager and its role in the Safety Assurance Framework

The SafeCOP safety assurance concept relies on a Runtime Manager to support the safety of the co-CPS. This section presents the SafeCOP Runtime Manager (SRM) and its **innovative concepts** intended to support the safety assurance process for addressing RQ5 in section 2.3:

- **Distributed real-time monitoring**, see Section 2.8.1 “Conceptual architecture”.
- **Automatically-derived safety requirements** based on formal models, see Section 2.8.2 “Modeling safety requirements”;
- **Correct-by-construction monitor generation**, see Section 2.8.4 “Automatic generation of monitors”;
- **Reference implementation based on isolation mechanisms**, see Section 2.8.5 “Reference implementation architecture”.

Safety critical systems in many areas, including cooperative safety critical systems such as the ones considered by SafeCOP, have increasing complexity, both in terms of functionality and interactions. Due to this complexity, it is challenging to demonstrate pre-release that they are adequately safe. To increase the dependability of such systems, the practice is to augment the systems with monitors (Goodloe, 2010): “A monitor observes the behaviour of a system and detects if it is consistent with a specification. […] online monitors […] check conformance to a specification at runtime (as opposed to offline, at a later time) and can therefore drive the system into a known good state if it is found to deviate from its specification. A monitor can provide additional confidence at runtime that the system satisfies its specifications.”

While the goal of verification and validation techniques, such as testing, debugging, and theorem proving is to ensure general correctness of programs, the intention of run-time monitoring is to determine whether the current execution (and system HW) meets the specified technical requirements. To achieve this goal, monitors collect the data of interest from the monitored systems, which can be used for further analysis by the user, certification authority, or the monitor itself. (Asadi, 2013) presents a survey and classification of monitors, and defines a monitor as a “tool that observes the behaviour of a system and determines if it is consistent with a given specification.” Runtime diagnostics is a part of most modern safety-critical systems (Sourdis et al., 2013). In SafeCOP, we extend the traditional runtime diagnostics with a *SafeCOP Runtime Manager* (SRM) for cooperative cyber-

physical systems. Complementing the diagnostics, the SRM works on an extended scope of variables that affect the runtime behaviour and establishes runtime certificates ensuring compatibility of the co-CPS. The SafeCOP Runtime Manager for co-CPS concept is discussed in the next section. SRM first ensures that all the cooperating systems contain a compatible cooperative item subsystem and that the configuration of the cooperating systems is within the predefined values established during out-of-context development of the cooperative item.

Several definitions are used in the literature for diagnostics and monitoring. We distinguish between the terms *monitor* and *manager* and the purpose of the definitions in this section is to ensure a consistency of terminology in the project.

- A safety monitor is responsible for data collection at runtime for safety assurance. Our monitor definition assumes that the monitor will not take actions, i.e., it only collects data[4]. Besides the typical data acquisition related to diagnostics, in SafeCOP the monitor will also collect safety-related data, which is used to periodically recheck the evidence related to the safety cases. Certifiers can decide to increase the confidence in safety case or decrease and revoke the certificate. As the run time monitoring can continuously add information to the safety case, we say that we have a dynamic safety case (Denney, 2015).
- A safety runtime manager is a mechanism that seeks to prevent undesired system states from being reached by detecting unsafe system states and triggering appropriate actions to bring the system back to a safe state. For example, it could bring the system into a safe stop, if one exists. A system is failsafe if it adopts "safe" output states in the event of failure and inability to recover. This manager definition is similar the monitor definition from (Goodloe, 2010).
- In addition, the monitored system is typically called System Under Observation (SUO).

The SafeCOP Runtime Manager (SRM) concept uses a Distributed-System Monitoring architecture (Goodloe, 2010), i.e., "monitors for a distributed system are other processes in the distributed system." The SRM is integrated with the SUO and has the following requirements, (Goodloe, 2010):

- **Functionality**: the manager does not change the functionality of the SUO unless the SUO violates its specification.
- **Schedulability**: the manager architecture does not cause the SUO to violate its hard real-time guarantees, unless the SUO violates its specification.
- **Reliability**: the reliability of the SUO in the context of the manager architecture is greater or equal to the reliability of the SUO alone.
- **Certifiability**: the manager architecture does not unduly require modifications to the source code or object code of the SUO.

2.8.1 Conceptual architecture

The conceptual architecture of the SafeCOP Runtime Manager is presented in Figure 6. On the right, we depict the *SafeCOP generic architecture* proposed in deliverable "D1.1 Requirements and Evaluation Metrics Baseline". We describe next each block in this generic architecture:

[4] Please note that our definition of monitor differs from the diagnostics functions/monitoring in standards, which additionally allows for taking action in response to monitored data.

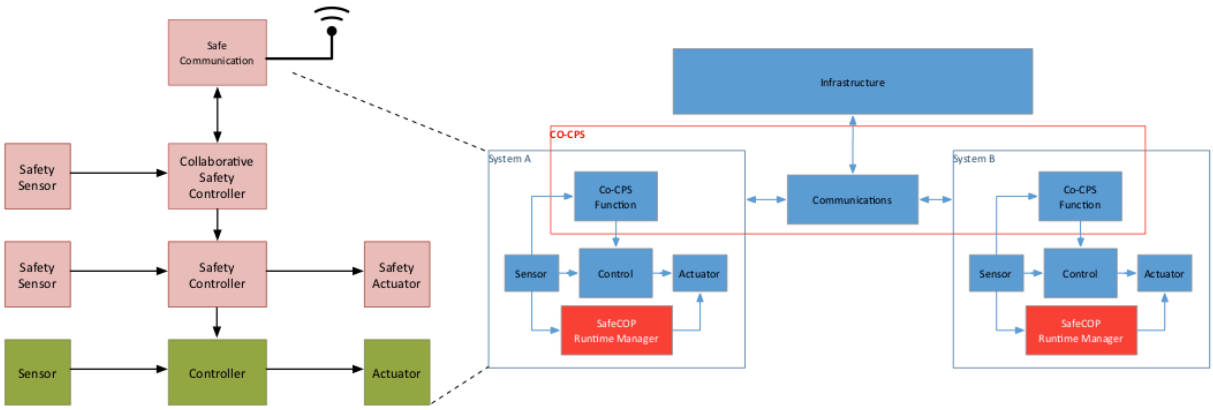

Figure 6: SafeCOP Runtime Manager concept

- **Infrastructure –** This block represents all the information databases about the external factors that could affect the system. All the requirements based on data clouds, environment, external systems, should be associated in there.
- **Communications –** This block represents the communication not only between different system nodes but also between each component of the system. All the requirements based on communication protocols, shared messages, communication failures, message format, or even hardware should be linked to this block.
- **co-CPS function –** Cooperative function under development by each use case. All the requirements based on cooperative functions being developed under each demonstrator should be linked here.
- **Sensor –** The sensor block represents all inputs to the system, such as internal vehicle sensors, or external data providers, like RSUs. All the requirements based on sensors and related data should be associated to this block.
- **Control –** This block represents the basic control over actuators, for example coming from device drivers. All the requirements related with control functions, parameter limits should be covered by Control block.
- **Actuator –** The actuator block represents all basic system output. The requirements related with actions and results should be covered here.
- **SafeCOP Runtime Manager.** This block has been called **Monitor** in deliverable D1.1. In this deliverable, we go into details about this block and explain its role in the SafeCOP Safety Assurance concept.
- Note that we have left out of the generic architecture the part related to **processes, methods and tools,** see D1.1 for details. The processes, methods and tools related to the SafeCOP Safety Assurance concept are presented in this deliverable and in D2.4, see Figure 3 for an overview.

On the left in Figure 6 we depict in more detail how one system out of the cooperating systems in a co-CPS is structured. Note that the SafeCOP Runtime Manager is an umbrella term that we use for the parts of the safety controllers which are focused on the collaborative aspects of safety. The SRM is actually implemented as safety controllers into each system, using a computer-based system consisting of hardware and software. Our SRM concept is compatible with the concepts in the functional safety standards (e.g., IEC 61508). Thus, each system relies on a *Controller* to implement its functionality (depicted in green). If the functionality is safety-related, then the controller is called a *Safety Controller*, depicted in red. A Safety Controller receives inputs from *Safety Sensors* and control the system via *Safety Actuators*. We call a Safety Controller that receive inputs form other systems via the wireless communication a *Collaborative Safety Controller*. Note that conceptually (and

in accordance to the functional safety standards) these are also Safety Controllers, just that it is useful to make this distinction for the case when inputs are coming from outside of the system.

The SRM concept is purposely high-level and compatible with the current standards: we plan to use it in each use case in SafeCOP and our use cases cover multiple areas. The implementation of the SRM is left for each use case. The implementation will consist of hardware and software safety functions that check safety-related conditions (events and states) at runtime and activate different safety functions to perform prevention or recovery actions. However, **in SafeCOP we propose that the Runtime Manager has the following innovative characteristics,** especially designed to support the safety assurance and to reduce the development, validation and verification, and assurance efforts:

- **Distributed real-time monitor**: The SRM extends the state-of-the-art in the area of real-time monitoring, i.e., monitoring not only the functionality (including providing "black-box" recording functionality), but also non-functional properties such as timing (Asadi, 2013). As mentioned earlier, we use a distributed-system architecture (Goodloe, 2010).
- **Automatically-derived safety requirements based on LTL**: The SRM receives as input from the Safety Assurance process (see Figure 3) the safety requirements that have to be monitored at runtime. In Section 2.8.2 "Modeling safety requirements" we propose to use an extension of the *Linear Temporal Logic* modeling formalism to model these safety requirements.
- **Correct-by-construction monitor generation.** Parts of the Safety Controllers within the SRM are automatically generated from a high-level formal "monitor specification language" that we propose in SafeCOP. Section 2.8.4 "Automatic generation of monitors" gives an overview of this work, which will be reported in more detail in deliverable D4.1 "Report on SafeCOP platform architecture, methods and runtime mechanisms".
- **Reference implementation based on isolation mechanisms.** The SafeCOP Runtime Manager (SRM) is presented at a conceptual level, leaving out the implementation details. The implementation of the wireless link is specified in WP3 as a "safety layer". In addition, we also define a reference implementation model for one given CPS (vehicle, robot, etc.) that is part of a co-CPS cooperation. Section 2.8.5 "Reference implementation architecture" provides a brief overview, and the details will be presented in deliverable D4.1 "Report on SafeCOP platform architecture, methods and runtime mechanisms".

2.8.2 Modeling safety requirements

Runtime verification is an emerging discipline that investigates methods and tools to enable the verification of program properties during the execution of the application. The goal is to complement static analysis approaches, in particular when static verification leads to the explosion of states. Non-functional properties, such as the ones present in real-time systems are an ideal target for this kind of verification methodology, as are usually out of the range of the power and expressiveness of classic static analyses.

Runtime verification is distinguishable from Runtime Monitoring due to its support on formal languages for specification of properties, and on formal models where those properties are matched against, and a verdict is made regarding the satisfiability of the formula by the trace of the target system's execution captured by the model is considering. Typically, such formal languages are instances of the well-known Linear Temporal Logic, initially proposed in (Pnueli,1977) and since subject to many extensions and intensively adopted also for the development of Model Checking solutions.

Modeling Safety Requirements using Linear Temporal Logic

Linear temporal logic (LTL) is a modal logic that has been widely used to model *functional* (aka. *behavioural*) *requirements* on reactive and embedded systems. A comprehensive introduction to LTL and its use is (Fisher 2011), and we present some basics in Appendix B.

LTL may be used at design time to capture safety critical system properties through a hazard analysis. For example, all the above LTL formulas can be related to the dynamic hazard of wireless communication packet-loss in a platoon. System requirements can be decomposed into sub-system and unit requirements within a tool such as Chess (ref elsewhere in this document). At any level in the system architecture, LTL requirements can be compared with system models using model-checking technology for design verification of safety critical properties. Finally, LTL requirements can be compared against actual software and hardware implementations of components, systems and even system-of-systems by the process of requirements testing. The KTH testing tool LBTest is being evaluated for this role of testing LTL requirements within SafeCOP on platooning UC 6.

An important theme within SafeCOP is to link risk management at design time, with risk management at product deployment time by means of runtime monitoring. An important observation here is that the same LTL modeling methodology that is used at design time can also be used at runtime. A non-invasive approach to monitoring is possible, whereby critical LTL formulas are embedded into runtime monitors that generate system notifications in real-time when they have been violated. From these hazard notifications, the system can be engineered to respond by falling back into a safe state. This approach has been developed by the runtime verification community in recent years, see e.g (Reinbacher 2014). The SafeCOP project makes a number of novel contributions to runtime monitoring, including automated (and hence reliable) monitor generation, and a quantitative approach to safety analysis of dynamic environmental factors that integrates design time testing with runtime monitoring.

2.8.3 Automatic derivation of safety requirements from the safety assurance process

The safety case relies on constraints that restrict the runtime behaviour of the safety function, in order to anticipate at design time the circumstances that can occur at runtime. These runtime constraints are derived from the safety requirements and are enforced by the SRM, which includes monitoring of the cooperative safety function. We use assumption guarantee contracts to capture these constraints. A contract is a pair of assertions, namely assumptions and guarantees, where an element described by the contract offers guarantees about its own behaviour given that its environment fulfils the assumptions (Benveniste, 2012). SRM checks such contracts during runtime, and generates runtime evidence. The dynamic part of the safety assurance case is built upon the runtime evidence. Since such evidence is tightly coupled with contracts, the resulting safety assurance case depends on the validity of the contract, i.e., both satisfaction of its assumptions and provision of the guarantees when those assumptions are met. We call this a “conditional safety case”: the safety requirements formalized by the contracts are guaranteed only if the related contracts are valid. For example, the integrity of cooperative subsystem A is guaranteed by system A’s *architecture* and safety mechanisms. The public safety evidence of the subsystem B is refined into a set of *demands* that have to be fulfilled by subsystem B in order to ensure its integrity. We assure the satisfaction of the safety requirements through such contracts and their accompanying runtime evidence by modelling a safety assurance case argument and indicating explicitly which are the dynamic parts of the argument that need to be revisited (the dynamic part of the safety case). The details on this will be presented in deliverable D2.4 “**Guideline on safety assurance**”. In SafeCOP, we will use an argumentation editor compliant with the OMG standard SACM (Structured Assurance Case Metamodel)

(OMG, 2016) for modelling the safety case. The SACM framework supports both main graphical assurance case notations: Goal Structuring Notation (GSN) (GSN, 2011), and Claims Arguments Evidence (CAE) (Adelard, 2016). The standardisation and portability offered by SACM enables frequent updates to the safety case with less effort, as it becomes easier to automatically update the dynamic safety assurance case.

2.8.4 Automatic generation of monitors

In the previous section, we have introduced the main ideas behind Runtime Verification and referred to formal languages (of temporal logics) that have been used to formally specify safety-assurance properties and that make them suited for being further explored and validated under the context of SafeCOP. Using formal languages to specify system properties is not only useful for verification purposes. They are also extremely valuable to enforce a tight correlation between requirements and the verification of their implementation. In runtime, this means that if monitors are generated from these formally specified requirements, they shall always make verdicts about their validation in a correct manner.

Monitor generation is therefore a key concept when adopting formal languages to specify and verify systems. The generation process is normally based on constructing finite machines that take as input finite traces resulting from the operation of the system under monitoring, and checking if those traces are safe, i.e., if the machines terminates in a safe (accepting) state once the trace if fully consumed. In the case of RMTLD, what is generated is a composition of non-recursive functions that follow the pattern of the original specification after it is subject to a static simplification process that reduces the overload of the monitors during runtime.

The current framework based on RMTLD, named the *rmtld3synth* tool is available in (rmtld, 2017) and a web-based interface is available in (rmtld2, 2017). Lastly, it is important that the code of the monitors can be generated for the C++ and OCaml languages, which due mainly to the former, is a facilitator for integration in new runtimes/OSs.

Although having a method to generate monitors in a correct-by-construction manner from RMTLD if of great interest as a yet another component for securing the wellness of operation of the target system (i.e., ensuring that all the checks are correctly performed), there is always the need to tune the target runtime system to ensure that events of interest for monitoring are correctly saved in a (set of) buffer(s). Furthermore, RMTLD does not focus on distributed system verification, and therefore distributed timed properties have to be subject to other type of analysis in order to be able to be produce a set of monitors that are (possibly) coupled to the distributed components under verification. Extension of support the generation of monitors specified in RMTLD onto ROS components and the exploration of techniques to use RMTLD to generate sets of monitors to capture distributed real-time properties are two topics that will be addressed in SafeCOP from a research angle. Indeed, requiring non-experts to specify safety-assurance requirements using logics like LTL, PLTL and RMTLD, is also challenging. These are mathematical languages, and not DSLs or event programming languages, which make them not intuitive until a considerably slow-growing learning process takes place. This is also a reason why so many formal methods are still only in the academia. In SafeCOP, we plan to explore the development of new/improvement of existing DSLs to specify properties for safety assurance systems in a way that their syntax are familiar to the engineer. One such example is the REVERT language, developed by Kochanthara et al. (Kochanthara, 2016). REVERT is a new specification language for real-time applications designed with usability and easiness in mind. REVERT is a combination of state machine, extended regular expressions,

Boolean expressions, and timing constraints. REVERT relies on external events to reason about traces. Properties on execution patterns or execution order of events, that must be enforced during the application run-time, are specified using extended regular expressions. To express timing constraints on a sequence of events we use three high-level operators: time, duration and jitter. These operators are then automatically converted to finite timed automata. An example of the template for specifying monitors in REVERT is given below

```
monitor m_i {
  observe { ev_1, ..., ev_l }
  variables {v_1 : type, ..., v_j : type }
  jobs{
    j_1{
     start: {ev_1, ev_2}
     suspend: {ev_3}
     resume: {ev_1}
     complete: {ev_6}
    },
    ....
    j_p{...}
  }

  nodes { n_1, ..., n_k }
  initial { n_q }

  node n_1 { init_1 prop_1 trans_1 }
  ...
  node n_k { init_k prop_k trans_k }
}
```

The generation of monitors is achieved through the following steps: 1) Generating an automaton for each transition; 2) Generating an automaton for each node n_i by applying a product operation on the automata obtained for each transition from n_i to any other node. As mentioned in section II, implicit priority is used to resolve potential conflicts on the final state; 3) Generating the monitor automaton by concatenating the automata of all nodes. The monitor automaton is then converted to XML format which can be used to produce code. Below we present the state-machine generated by the specification “failure(duration(j)<10)”, where j is specified using timed regular expressions. In SafeCOP we are interest in investigating the interplay between RMTLD and the language provided by REVERT, and understand if/how they can be combined to provide a more flexible way of specifying monitors whose properties they need to verify have different demands in expressivity.

If SRM detects abnormal behaviour in terms of contract violations it should react accordingly, e.g., if the contract demands are not satisfied, the cooperative safety functionality is disabled. SRM is responsible for the safety action execution. SafeCOP will analyse the requirements of the use cases and propose constraints based on the definition of the cooperative safety functions, such that sufficient levels of safety are guaranteed. The safety case needs to also consider the risk of requirement violations, and how to provide failsafe fallback mechanisms. Considered mechanisms, including safety under such scenarios, will introduce additional constraints on the systems involved.

The SRM has to know what to monitor; the “Verification and Validation” methods and tools will produce safety requirements that need to be monitored at runtime. Since many of co-CPS systems are fail operational, it is not acceptable to simply shut down the cooperative function, but the function needs to be provided even in presence of failures, even if that means degrading performance of the function. Once the SRM detects a safety violation, it will have to fall-back to a “degraded mode”. The requirements specification and the corresponding contracts need to be defined such that they enable specification of the degraded behaviour (Girs, 2017). Details on how to capture such behaviour is reported in more detail in deliverable D4.1 “Report on SafeCOP platform architecture,

methods and runtime mechanisms". The functionality of the "degraded mode" will have to be developed in the demonstrators, and it is specific to the function of the respective demonstrator. It is challenging to develop useful and failsafe fall-back functions, since often it is not appropriate to just stop (assuming there is a "safe stop" function).

2.8.5 Reference implementation architecture

The functional safety standard IEC 61508 recommends two monitoring approaches: "(1) the monitor and the monitored function in the same computer, with some guarantee of independence between them; and (2) the monitor and the monitored function in separate computers." The SRM concept presented here does not impose any implementation approaches. However, to support case (1) which is more challenging, in SafeCOP we define a reference implementation model for one given CPS.

During the engineering of a safety-critical system, the hazards are identified, and their severity is analyzed, the risks are assessed, and the appropriate risk control measures are introduced to reduce the risk to an acceptable level. A Safety-Integrity Level (SIL) captures the required level of risk reduction. SILs differ slightly among areas (for example, the avionics area uses five "Design Assurance Levels" (DAL), from DAL A to DAL E; in the automotive area there is the Automotive SIL, or ASIL). SILs are assigned to safety functions, from SIL 4 (most critical) to SIL 0 (non-critical). The SIL assigned to a function will dictate the development processes and certification procedures that have to be followed. SIL 0 functions are non-critical and can be developed using any methods. For SIL 1, a more systematic approach is needed, to the level required by quality management standards such as ISO 9001. SIL 2 is quite similar to SIL 1, but typically involves more reviewing and testing. SIL 3 is significantly more difficult. Certification standards will suggest specific methods to be followed and provide a checklist of techniques that are recommended to be applied. If "semi-formal" methods are acceptable in lower SILs, SIL 4 often requires formal methods, increasing further the difficulty and development costs associated to building safety-critical systems. SIL 4 is not possible to implement on a single chip due to redundancy requirements.

The current trend is towards the use of Multicores Processors (MCPs) that can improve SWaP (Size, Weight & Power Reduction) and performance, allowing the integration of multiple applications, but introduce resource contentions due to sharing of hardware resources (Wilhelm, 2012) such as, cores, on-chip network, memory sub-system, etc. With approach (1), the Runtime Manager is implemented as software running on MCP. This SRM functionality has to be "separated" from the functionality of the device, such that lower-criticality functions do not affect the functioning of the high-criticality RM.

Functions of different SILs have to be separated. Otherwise, for example, a lower-criticality function could write in the code or data area of a higher-criticality function, leading thus to a failure. The current practice to mixed-criticality systems is to physically separate the different criticality functions in different hardware components, so they cannot influence each other. If such an approach has worked in the past, the advent of multicores is, however, commonly regarded as a design challenge in the safety-critical area, as there are no established approaches to achieve certification. When several safety functions of different SILs share the same multicore, the standards require that the hardware and software be developed at the highest SIL among the SILs of the safety functions (for more details, see the standards), which is very expensive. This is because the current multicore architectures do not offer enough separation between the safety functions (Nowotsch, 2012). For

example, two tasks on separate cores might compete over the bus for access to the memory, increasing the worst-case execution times (WCETs), on which important timing guarantees are based. At the same time, there is an increased need for flexibility in the systems used in the safety-critical market. This need for flexibility puts new requirements on the customization and the upgradability of both the non-safety and safety-critical part. However, currently, any small change in the critical or non-critical functions, leads to an expensive recertification, at the highest SIL.

WP4 is concerned with the proposal of a Safety Runtime Manager reference architecture. We do not go into details in this deliverable (see WP4 deliverables for details), but instead we have provided the motivation behind this work and below we outline the main ideas behind this work. The reference architecture is based on a model that has two parts: (1) the *Wireless Communication Model* (which contains the safety layer, defined by WP3), which also specifies also the set of messages, bandwidth requirements, etc., depending on the communication mode and (2) the *Safety Runtime Manager CPS Implementation Model*. This runtime manager model has the following sub-models: (i) *Task and Message Models*, i.e., the SRM is implemented as real-time tasks which trigger safety actions. Regarding these models, specific functionality to be developed and demonstrated the use cases, and we plan to use "mode changes": a different set of tasks is activated depending on the communication mode, e.g., no cooperation vs. wifi-based cooperation vs. LTE-based cooperation. (ii) *Scheduling Model*: the tasks are running on top of an RTOS (depending on the use case), and we will model both non-preemptive static cyclic scheduling and preemptive fixed-priority scheduling policies. To capture the separation required for mixed-criticality functions, we also include a (iii) *Partitioning Model*, and in selected use cases we plan to use RT-hypervisor, e.g., XtratuM or PikeOS, besides physical separations. In these hypervisors, depending on the standard used, the the partitions are typically modeled as a partition schedule table. Finally, we will also use (iv) a CPS Platform Model that captures the details of the architecture, i.e., Single- vs. Multi-Core Processor architecture, in-vehicle network architecture.

Based on this reference architecture proposal from WP4, WP5 will implement selected safety functions by extending the platforms used in their respective domains, e.g., Robot Operating System (ROS[5]), C++ (Posix) integrated in Linux and possibliy also AUTOSAR[6]. Since these platforms may not provide separation mechanisms, WP4 will propose extensions that support the required separation.

2.8.6 Relation to the SafeCOP Safety Assurance concept

Safety requirements are the backbone of safety assurance. The goal of assurance would be to show that those requirements address all the unreasonable risk in the system, and that they are satisfied with sufficient confidence. In SafeCOP we use contracts to formalize and validate those requirements on the system models enriched with contracts. But not all requirements can be validated during design-time. Hence, SRM contains a *monitoring module* that performs *data acquisition* to collect violations of such contracts during runtime, which is then analysed and included in the safety assurance evidence regarding the requirements related to the reported contracts. In this context, we say that as a result of continuous assurance we have a "dynamic safety case" (Medawar, 2017), which is updated with information collected at runtime to evaluate confidence in the satisfaction of

[5] http://www.ros.org
[6] http://www.autosar.org

safety requirements. In this context we also say that the safety case is *incremental*, and it may support *provisional certificates* allowing usage in limited scenarios (e.g., initially only in-the-lab use). Through evidence collected at runtime, provisional certificates can be upgraded to cover more general usage scenarios. The qualification of SRM and the constraints are part of the design-time safety assurance evidence.

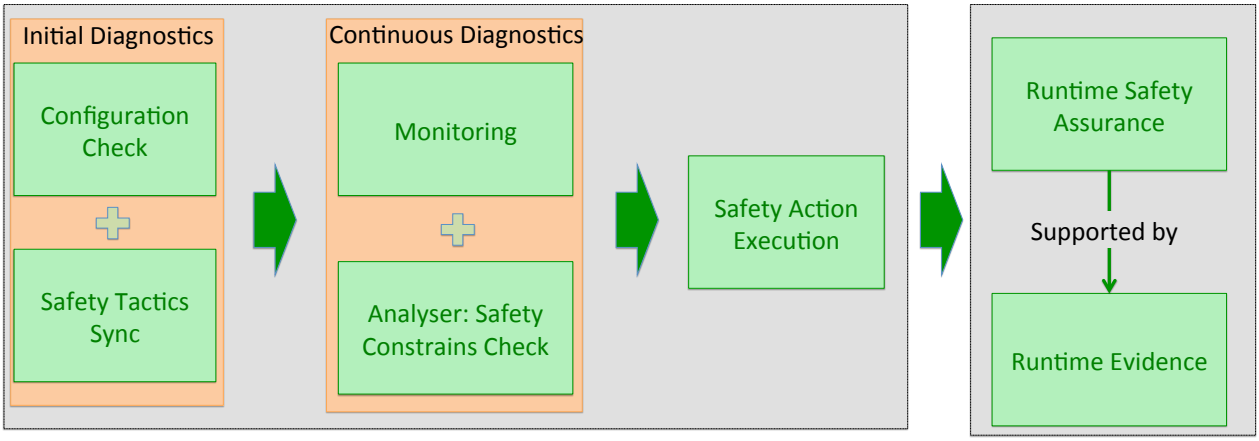

Figure 7: The SafeCOP model of diagnostics.

3 Relation with other work packages (WP3, WP4, WP5)

SafeCOP safety assurance framework targeting co-CPS (and eveolved in WP2) has the assumption of constituent systems communicating wirelessly. WP3 has the goal to extend the current wireless protocols (of interest and applicability in partner use cases), such that they provide the required levels of safety and security for co-CPS. The safety assurance framework supported by the SafeCOP platform is the focus of WP4, which consists of reference architecture for co-CPS and methods and tools for producing safety assurance evidence. The work in WP2, WP3 and WP4 happens in parallel, interacting on the issues related to the assurance framework, wireless protocol, architecture and methods and tools with constant sharing of knowledge and results to enrich each other's work.
Once a safe and secure communication solution is available in WP3, then the design of a distributed cooperation algorithms with safety-critical requirements could be attempted. Combined deliberations involving WP2, WP3 and WP4 are being conducted to make sure that a) such design is safe and secure b) is certifiable by the usage and specification of contract formalisms c) that these algorithms also identify and monitor key parameters as well as predict models of quality of communication to adapt the systems run-time behaviours in a safe and secure manner and d) they are implemented through the WP4 recommended reference architecture and tools.

4 Conclusions

4.1 Contributions

Related to the research questions (RQs) posed in Section 2.2, our conclusions are:

- **Current state-of-the-art and the state-of-practice defined by safety standards do not adequately support development and certification of co-CPS**. [relate to RQ1]
  Hence, the research conducted in WP2 is highly relevant and opens up themes worthy of standardization.

We have conducted an elaborate literature review and industrial survey to assess the state of the art and practice (results already presented in D2.1), which highlighted the inadequacy of the existing standards and practices. Section 2.6 describes cooperative safety function in the context of ISO26262, thus providing a potential way to address the co-CPS aspects by standards.

- The existing **pre-deployment safety/security analyses** do not provide support for multi-attribute co-CPS reasoning [relate to RQ2]
  We have performed a systematic literature review on safety/security co-analyses where we identified different analyses techniques proposed in the past 5 years and whether they target a single attribute, multiple attributes and in which way do they analyse the multiple attributes (Lisova 2018). We have also analyzed the inadequacy of existing hazard analysis techniques in dealing with SoS and proposed a new approach. As part of D2.4 there had been efforts on applying STAMP and relating it to the traditional analyses (evaluating which technique helps identify more hazards).
- **Current safety/security standards do not support development and certification of safety and security simultaneously.** [relate to RQ3]
  This issue is primarily reported on in SafeCOP deliverable D2.3.
- **The existing safety-case assurance do not address composability of safety evidences, nor conditional or continuous safety assurance**. [relate to RQ4]
  The SafeCOP safety-case assurance, presented in sections 2.3 and 2.4 introduces a two-level safety assurance, in which the assurance of the individual systems can be linked to the assurance at the co-operative function system-of-systems-level via contracts expressing assumptions and guarantees between the two levels, thereby enabling composability of safety evidence as well as conditional safety assurance. This concept has also resulted in publications and further submission are planned. Section 2.5 described SafeCOP perspective on linking model-based design with assurance cases. In Section 2.7 we briefly presented our contribution towards continuous safety assurance (through life-cycle) to facilitate maintenance of safety cases using safety contracts.
- **The existing runtime fault diagnostics techniques are neither specifically targeting dynamic/continuous safety assurance nor well-developed to address these issues**. [relate to RQ5]
  Section 2.8 outlines the SafeCOP runtime monitoring concept and its relation to safety-assurance; further details are provided in the SafeCOP WP4 deliverables.

The above contributions are of direct relevance to industry who are involved in design, development, verification and certification of co-CPS. Additionally, we investigated how the allied industry can potentially benefit from our results. In this context, we have outlined a new business model for selling components with safety certificates which is aligned with our composable safety assurance approach for co-CPS. Details of this business model is presented in Appendix A.

Specific connections and derivatives of WP2 w.r.t standardization and use cases are presented below.

4.2 Recommendations to standard committees

Based on a stress testing/impact analysis of the ISO26262 standard concerning its ability to handle co-CPS, our conclusion is that it should be straightforward to extend the scope of the standard to cover such scenarios. At least the following issues need to be adjusted/considered:

- The definition of "road vehicle" needs to be extended to cover cooperating vehicles (in the terminology of ISO DIS 26262:2018), i.e. "super vehicles" in our terminology above. Cooperative functions should then be defined for management of the "super vehicle" at the level of this "super vehicle".
- Based on the "super vehicle", cooperate items (cooperative steering, cooperative braking, etc.) can be defined, implemented and certified according to what is prescribed by the current standard.

- For liability or safety process there is additionally a need to clarify the top-level responsibility, i.e. it seems likely that there is a need to have an assigned stakeholder responsible for the top-level integration of the co-CPS and its safety assurance/certification.
- Due to the typically loosely coupled nature of cooperative vehicle functions, cybersecurity needs an increased attention (elaborated on in D2.3).

4.3 Demonstration

The WP2 results will be applied as the underpinning theory in some of the demonstration activities. Apart from specific examples elaborated as part of the research publications, all use cases have applied the "SafeCOP safety analysis" (based on STAMP) and plans include to integrate and make visible other aspects of the WP2 work in some of the use cases. In particular, UC5 and UC6 will apply the full tool-based methodology that we describe. One of the main bottlenecks here is that the WP2 research addresses issues across the breadth of the spectrum of activities throughout the lifecycle, whereas the use cases are more specific in focus and may not be able to justifiably capture the WP2 work as part of their scope. Considering this, we are now specifically focussing on the platooning use case (UC5.6) and working closely together to demonstrate WP2's results by a) providing guidelines on a tool chain that can be used to use and demonstrate the cooperative safety function and its distributed implementation providing traceability as well as appropriate contract-based formalisms to bind them together.

5 Bibliography

(26262) ISO 26262:2018 FDIS, International Standardization Organization. "Road vehicles – Functional safety". Final Draft International Standard 2018.

(Adelard, 2016) ASCAD: The Adelard Safety Case Development Manual. http://www.adelard.com/services/SafetyCaseStructuring/ 2016-12-23

(Asadi, 2013) Asadi, N., Saadatmand, M. and Sjödin, M., 2013, October. Run-time monitoring of timing constraints: A survey of methods and tools. In International Conference on Software Engineering Advances (ICSEA'13).

(Baumgart, 2017), S. Baumgart, J. Fröberg and S. Punnekkat, "Analyzing hazards in system-of-systems: Described in a quarry site automation context," *IEEE International Systems Conference (SysCon)*, Montreal, QC, 2017

(Benevniste, 2012) Benveniste A., Caillaud B., Nickovic D., Passerone R., Raclet J. B., Reinkmeier P., Sangiovanni-Vincentelli A., Damm W., Henzinger T., Larsen K.G. Contracts for System Design. Research report RR-8147. Inria. November 2012.

(Bhopal disaster) Bhopal disaster https://en.wikipedia.org/wiki/Bhopal_disaster, accessed 2018-03-12

(Bouyer,2010) Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. Inf. Comput. 208(2), 97–116 (2010)

(CHESS, 2018) CHESS toolset, Eclipse Polarsys website: https://www.polarsys.org/chess/

(CHESS, 2012b) A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi and T. Vardanega,"CHESS: A Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems", Proceedings of Automated Software Engineering (ASE) International Conference, Essen, July 2012

(CHESS, 2016) Mazzini, S., Favaro, J., Puri, S., Baracchi, L.: CHESS: an open source methodology and toolset for the development of critical systems. Third Workshop on Open Source Software for Model Driven Engineering (OSS4MDE 2016)

(CHESSML) https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf

(CHESS, 2012a) Composition with Guarantees for High-integrity Embedded Software Components Assembly (CHESS), ARTEMIS project. http://www.chess-project.org/, 2012

(CONCERTO, 2016) Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multi-core Systems, ARTEMIS project. http://www.concerto-project.org/, 2016

(CONCERTO, 2015) Baldovin A., Zovi A., Nelissen G. and Puri S.,"CONCERTO: A toolset for model-based engineering of avionics applications", Proc. of 20th International Conference on Reliable Software Technologies- Ada-Europe, Madrid, June 2015.
(Denney 2015) Denney, Ewen, Ganesh J. Pai and Ibrahim Habli. "Dynamic Safety Cases for Through-Life Safety Assurance." 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering
(Foord 2011) "TEN YEARS OF IEC 61508; HAS IT MADE ANY DIFFERENCE?" A. G. Foord and W. G. Gulland, 4-Sight Consulting, UK C. R. Howard, Istech Consulting Ltd, UK, 2011
(FoReVer, 2014) L. Baracchi, A. Cimatti, G. Garcia, S. Mazzini, S. Puri and S. Tonetta, "Requirements refinement and component reuse: the FoReVer contract-based approach", in A. Bagnato, I. R. Quadri, M. Rossi and l. S. Indrusiak, Editors Industry and Research Perspectives on Embedded System Design, IGI Global, 2014.
(Girs, 2017) Girs S., Sljivo I. and Jaradat O. Contract-Based Assurance for Wireless Cooperative Functions of Vehicular Systems. In 43rd Annual Conference of the IEEE Industrial Electronics Society. October 2017.
(Goodloe, 2010) Goodloe, Alwyn E., and Lee Pike. "Monitoring distributed real-time systems: A survey and future directions." (2010).
(GSN, 2011) GSN Community Standard Version 1. Technical report, Origin Consulting (York) Limited, November 2011.
(IBM Software Rational, 2010), IBM Software Rational. (2010). DO-178B compliance: turn an overhead expense into a competitive advantage. Aerospace and Defense: White paper.
(IEC 61508) IEC 61508:2010, Functional safety of electrical/electronic/ programmable electronic safety-related systems – Parts 1–7.
(Jaradat 2016), Omar Jaradat, Iain Bate , Systematic Maintenance of Safety Cases to Reduce Risk, 4th International Workshop on Assurance Cases for Software-intensive Systems (ASSURE), Sep 2016
(Jaradat 2017), Omar Jaradat, Iain Bate, Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases: An Example , MRTC Technical report, Apr 2017
(Jaradat 2017), Omar Jaradat, Iain Bate, Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases, European Dependable Computing Conference (EDCC), Sep 2017
(Jaradat 2018), Omar Jaradat, Sasikumar Punnekkat, Using Safety Contracts to Verify Design Assumptions During Runtime , Accepted for Ada-Europe, June 2018
(Joshi et al. 2017) Sanjay. L. Joshi , Bharat Deshpande , Sasikumar Punnekkat. An Industrial Survey on the Influence of Process and Product Attributes on Software Product Reliability. IEEE International Conference on Networks & Advances in Computational Technologies (NetACT), Oct 2017.
(Kochanthara, 2016) Kochanthara, S, Nelissen, G, Pereira, D, Purandare, R, "REVERT: A Monitor Generation Tool for Real-Time Systems", IEEE Real-Time Systems Symposium (RTSS 2016). 29, Nov to 2, Dec, 2016, RTSS@Work. Porto, Portugal.
(Koymans, 1990) Koymans, Ron: Specifying real-time properties with metric temporal logic. Real-Time Syst. 2(4), 255–299 (1990)
(Leveson 2002) Nancy G. Leveson: A New Accident Model for Engineering Safer Systems, ESD Working Papers;ESD-WP-2003-01.19-ESD Internal Symposium. http://hdl.handle.net/1721.1/102747
(Lisova 2018) Elena Lisova, Irfan Sljivo, Aida Causevic: Safety and Security Co-Analyses: A Systematic Literature Review, in submission for journal publication
(Medawar, 2017) Medawar S., Sljivo I. and Scholle D. Cooperative Safety Critical CPS Platooning in SafeCOP. In 5th EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems. June 2017.
(Nowotsch, 2012) Nowotsch, J., Paulitsch. M. Leveraging Multi-core Computing Architectures in Avionics. In European Dependable Computing Conference, pp.132-143, 2012.
(Nilsson et al., 2013) Josef Nilsson, Carl Bergenhem, Jan Jacobson, Jonny Vinter, "Functional Safety for Cooperative Systems", SAE World Congress 2013, 13AE-0139
(OCRA, 2016) OCRA: Othello Contracts Refinement Analysis. https://es-static.fbk.eu/ 2016-12-23
(OMG, 2016) Object Management Group (OMG). SACM: Structured Assurance Case Metamodel. Technical Report, Version 2.0, OMG, 2016. http://www.omg.org/spec/SACM/ 2016-12-23
(Ouaknine,2008) Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: FORMATS '08, pp. 1–13, Springer, Berlin (2008)
(Pedro, 2015) Pedro, A, Pereira, D, Pinho, L, Pinto, J, "Monitoring for a decidable fragment of MTLD", The 15th International Conference on Runtime Verification (RV'15). 22 to 25, Sep, 2015. Vienna, Austria.

(Pedro, 2017) Pedro, A, Pinto, J, Pereira, D, Pinho, L, "Runtime verification of autopilot systems using a fragment of MTL-∫", International Journal on Software Tools for Technology Transfer (STTT), Springer Berlin Heidelberg. 21, Aug, 2017, pp 1-17.

(Pnuelli, 1977) Pnueli, A., "The Temporal Logic of Programs". Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS'77), IEEE Computer Society, pp. 46-57, Washington, DC, USA, 1977.

(Pop 2016) Paul Pop, Detlef Scholle , Hans Hansson, Gunnar Widforss, Malin Rosqvist , The SafeCOP ECSEL Project -- Best Paper Award--, Euromicro Conference on Digital System Design (DSD), Aug 2016

(Pop 2017) Paul Pop, Detlef Scholle , Irfan Sljivo, Hans Hansson, Gunnar Widforss, Malin Rosqvist, Safe Cooperating Cyber-Physical Systems using Wireless Communication, Elsevier journal of Microprocessors and Microsystems (MICPRO), Jul 2017.

(Reinbacher 2014) Thomas Reinbacher, Kristin Yvonne Rozier, Johann Schumann: Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems. TACAS 2014: 357-372

(rmtld, 2017) Pedro, A, Pinto, J, Pereira, D, Pinho, L, "rmtld3synth: Runtime Verification toolchain for generation of monitors based on the restricted Metric Temporal Logic with Durations.", online: https://github.com/cistergit/rmtld3synth, last accessed: 02/2018

(rmtld2, 2017) Pedro, A, Pinto, J, Pereira, D, Pinho, L, "rmtld3synth web demonstrator", online: https://anmaped.github.io/rmtld3synth/, last accessed: 02/2018

(SafeCer, 2016) SafeCer project: "Safety Certification of Software-Intensive Systems with Reusable Com-ponents", http://safecer.eu/, 2016.

(Seveso disaster) Seveso disaster: https://en.wikipedia.org/wiki/Seveso_disaster, accessed 2018-03-12

(Simpson 2010) "Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety, IEC 61508 (2010 Edition) and Related Standards", David J. Smith and Kenneth G.L. Simpson

(Sivencrona, Johansson 2007) http://www.mynewsdesk.com/se/news/the-idea-of-iso26262-for-functional-safety-10227

(Sljivo, 2015) Irfan Sljivo, Barbara Gallina, Jan Carlson, Hans Hansson. Using Safety Contracts to Guide the Integration of Reusable Safety Elements within ISO 26262. The 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2015)

(Sljivo, 2016) 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. Volume 131. 570-590. Elsevier. July 2016.

(Sljivo, 2016b) Irfan Sljivo, Barbara Gallina, Jan Carlson, Hans Hansson. Configuration-aware Contracts, 4th International Workshop on Assurance Cases for Software-intensive Systems (ASSURE2016)

(Sljivo, 2018)Irfan Sljivo, Barbara Gallina, Jan Carlson, Hans Hansson, Stefano Puri. Tool-Supported Safety-Relevant Component Reuse: From Specification to Argumentation, Accepted for 23rd International Conference on Reliable Software Technologies -Ada-Europe 2018

(Soderberg 2013) Soderberg, Andreas, and Rolf Johansson. "Safety contract based design of software components." Software Reliability Engineering Workshops (ISSREW), IEEE International Symposium on. IEEE, 2013.

(Sourdis et al., 2013) Sourdis, C. Strydis, A. Armato, C.S. Bouganis, B. Falsafi, G.N. Gaydadjiev, S. Isaza, A. Malek, R. Mariani, D. Pnevmatikatos, and D.K. Pradhan. DeSyRe: On-demand system reliability. Microprocessors and Microsystems, 37(8), pp.981-1001, 2013.

(Wilhelm, 2012) R. Wilhelm and J. Reineke, "Embedded Systems: Many Cores—Many Problems," in Symposium on Industrial Embedded Systems, 2012.

Appendix A: A Business Model for selling components with safety certificates

This appendix outlines a business model for selling safety-related components. The goal is protection of the suppliers' intellectual property whilst providing confidence to the customer that the component is safe. The actors, e.g. supplier, customer etc., and their roles in the business model are described and motivated. The business model is described in an ideal scenario. Commentary is given where ideal assumptions differ from reality e.g. as found from experience. As this is an ideal model, it would require major effort (and expense) to use in a real scenario. However, the gains are clear. Also, it allows deficiencies in current practices to be pointed out and at least discussed. The business model assumes the use of a safety manual, in the interface to the customer; in the scope of components that are developed according to ISO 26262. Although this concept is not new (it is found in IEC 61508), it is not currently specified in ISO 26262. The ideal business model is contrasted with a fictive inferior model that is based on experience in the industry.

The background is the following: A supplier develops and offers a safety component. The offer is found by a potential customer through marketing and negotiations are made of a purchase. The material used for marketing the component is based on properties from the development of the component. The customer requests a specification of the component (beyond initial marketing material), e.g. its interface and properties, and evidence of conformance, e.g. to applicable safety standards. Since much of this data (specification sand evidence of conformance) is found in supplier documentation that also contains other information that the supplier should not expose, a challenge in this relationship is to control the exposure of the supplier's intellectual property to the customer. In the proposed business model this exposure is avoided by using a clear interface definition (a safety manual) and third-party certification of product safety to gain the customer's confidence without compromising the supplier intellectual property.

Traditionally system safety analysis has started with complete systems, not components, since the system is the scope. Functional safety standards have originated from this complete systems approach. For example, in aerospace, process and nuclear industry there is a long-standing tradition to estimate reliability, to analyze faults and to mitigate failures by system engineering. In the wake of several catastrophic incidents like the Seveso disaster and the tragic Bhopal disaster the need for an industrial standard to assess functional safety became an obvious priority. In 1998 the IEC 61508, the first international functional safety standard, was published and helped raise awareness of functional safety in the industry. Its lifecycle approach to functional safety (design, installation, operation and maintenance, human factors) has been the basis to standardization efforts in other fields, see (Foord 2011). The second edition of IEC 61508 was released in 2010. To allow for a component-oriented approach, e.g. ISO 26262 proposes the concept of SEooC - Safety Element out of Scope. A similar concept is available for IEC 61505:2010 : Pre-existing software element - (e.g. Route 1S: “development compliant to 61508”). See Part 3 - 7.4.2.12.

Problem definition and Proposal

Old and inferior business model

An automotive supplier would like to develop and sell a component. Since the requirements and context for the component is not actually known before a customer is known and involved, the development life cycle can, strictly speaking, not be initiated. When a customer appears, discussions can start and requirements for the component can be given; and development starts. When development is complete, the component is associated with all work products as mandated by ISO 26262, including a safety case for the component. The development process and component are subjected

to audit (fulfilment of process) and assessment (defensible technical claim for safety of the component). Most of these work products contain sensitive Intellectual Property (IP) of the component; which arguably should not be shown or passed to a customer. IP could include details about the design of the component. A typical unsatisfactory business model could be as is illustrated as Figure 8.

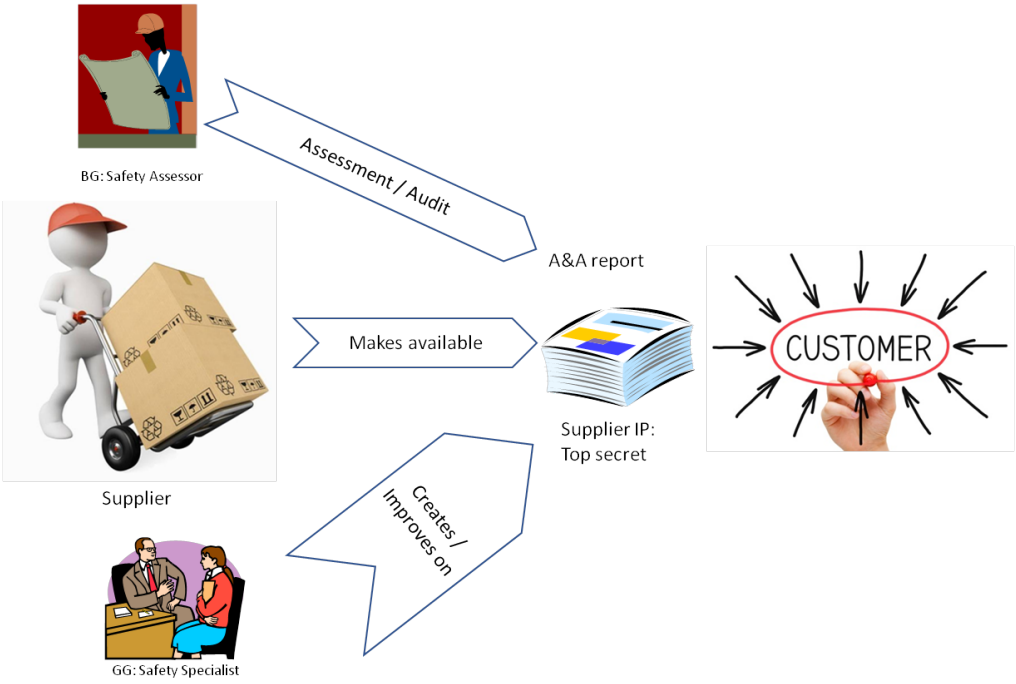

Figure 8: Inferior business model

The roles in the figure are:

- The **Supplier** develops the component and assigns a safety specialist (the "Good guy") to be part of the product development team. The supplier also assigns an independent third party (the "Bad guy") to performs audit and assessment.
- The **Good guy** (GG) is assigned by the supplier to perform development work of the component; such as FuSa-managing or safety engineering. The Good guy can also do pre-assessment, gap analysis and reviews, but will not be independent. Process development and training can also be performed. The Good Guy role could be fulfilled by a safety specialist consulting firm such as Qamcom Research and Technology AB.
- The **Bad guy** (BG) is assigned by the supplier (in some cases by the customer) to perform third party assessment and audit of the component. The bad guy is strictly independent to the supplier and the good guy, e.g. has not taken part in any development activity. This role could be fulfilled by the same type of form as for a GG, but provided that there has been no participation in development work.
- The **Customer**, typically an OEM, accepts the component based on acceptance of work products from the development. These are reviewed at joint meetings between the supplier and customer. In this review, the customer is given access to any requested work products, i.e. potentially secret IP.

With current practice, a customer for the component is found based on a RFI/RFQ procedure. In this, the supplier shows enough technical details and capability so that the customer trusts the technical capabilities and adequacy of the component. Technical details include e.g. safety case, but also

other technology centred documents such technical safety concept (TSC), requirement specifications and detailed system designs. The problem here is that of exposing IP to the customer. The customer could argue that refusal to do so could compromise the business deal. The supplier is in a weak position here.

The concept of good guy and bad guy can be illustrated as follows: The goal of the GG is to assist the supplier in developing the component. This includes both identifying gaps and also solving them. Conversely, the goal of the BG is to always offer an independent opinion. This implies that gaps can be identified, but the BG cannot aid in solving them; as this would compromise independence. In ISO 26262, confirmations measures (confirmation review, functional safety assessment and functional safety audit) require independence according to the ASIL level of the function. This implies that a GG (and also roles within the supplier development team) may be excluded from performing some of these tasks, as independence is not achieved.

A New and improved business model

We propose a business model where the supplier interfaces to the customer with a certificate safety and a safety manual. The certificate is signed and issued by an accredited independent 3rd party (e.g. certification body) that has reviewed details of the component. The certification process is in addition to functional safety assessment and audit, because the consequent report may expose IP. The safety manual defines the usage, interface, restrictions, and prerequisites of the component, and exposes no IP. Central to the business model is also the use of and trust in accredited parties and safety certificates. In the proposed business model, these are needed to claim independence between key parties. To understand the details of the business model, we first discuss the relation between the actors, and the generated work products, see Figure 9:

- The **Safety Assessor** (SA) is an accredited Bad Guy (e.g. independent inspection body). This implies that the party is accredited according to e.g. ISO/IEC 17020 to perform independent assessment according to a specific standard. ISO/IEC 17020 gives "General criteria for the operation of various types of bodies performing inspection"; basically, an extension to an existing quality management system. Accreditation implies that a certification or inspection body (in this case the safety assessor) has been assessed to demonstrate their competence, impartiality, and capability. In this setting, inspection (assessment) is done according to ISO 26262, e.g. this is what the SA is accredited for. The SA performs the assessment of the component being developed and issues a Functional Safety Assessment (FSA) Report (an ISO 26262 work product). This role could be fulfilled by an accredited company such as Safety Integrity AB.
- An **accredited FSA report** implies that the technical merits of the component to fulfil safety has been assessed. "Accredited" implies that the party performing the assessment has a quality management system that ensures correct procedure and handling of the assessment assignment, e.g. maintain quality of the assessment service. Note that the report still may contain sensitive IP, hence distribution should still be controlled.
- Accredited parties are subject to periodic assessment of their competence and conformance to their certification by national **Accreditation Bodies**. It is mandatory for every European member-state to have a single national Accreditation Body. Examples are SWEDAC in Sweden or UKAS in the United Kingdom. Accreditation is a confirmation that a party meets requirements to work with a particular standard; such as ISO 9001 or ISO 26262
- A **Certification Body** (CB) is an organisation accredited by a recognised accreditation body for its competence to perform assessment and issues a **safety certification**. Here the CB is an organisation with the business focused on certification, i.e. technical competence to assess a particular product is not in the scope of business. The role of the CB is motivated through added independence and trust

vis-á-vis the SA – who does have technical competence. The CB can oversee and "accredit" the assessment work of another party, such as the SA. Alternatively, the certificate could also be issued directly by an SA; but with no gain in independence and trust. Examples of certification bodies are: RISE-SP or DNV GL.

The gain for the **supplier** with this setup is stronger independence of the FSA report due to accreditation, and a new artefact, a **safety certificate.** With this artefact, the supplier can now make publicly available a statement (the certificate) that the component has been (successfully) subjected to audit and assessment by an accredited third party, without exposing any IP before or after a business relation is entered. Hence the certificate can be seen to guarantee the conclusion on the FSA report, i.e. acceptance that the component fulfils functional safety.

For life-cycle based automotive functional safety the first publication of ISO 26262 dates from 2011, (26262). Development of a component according to ISO 26262 can be according to different development categories: new development, modification, or SEooC (Sivencrona, Johansson 2007). The latter, a novel functional safety concept, was introduced to cater for reusable components that are developed not for use in a specific system (a certain vehicle in mind) but which are developed with the reuse of the component in different vehicles in mind e.g. an AUTOSAR communication stack or a microcontroller. A supplier develops a component (subsystem) as an SEooC, and makes assumptions about the context; what is provided (guarantees) and what is needed (assumes) by the

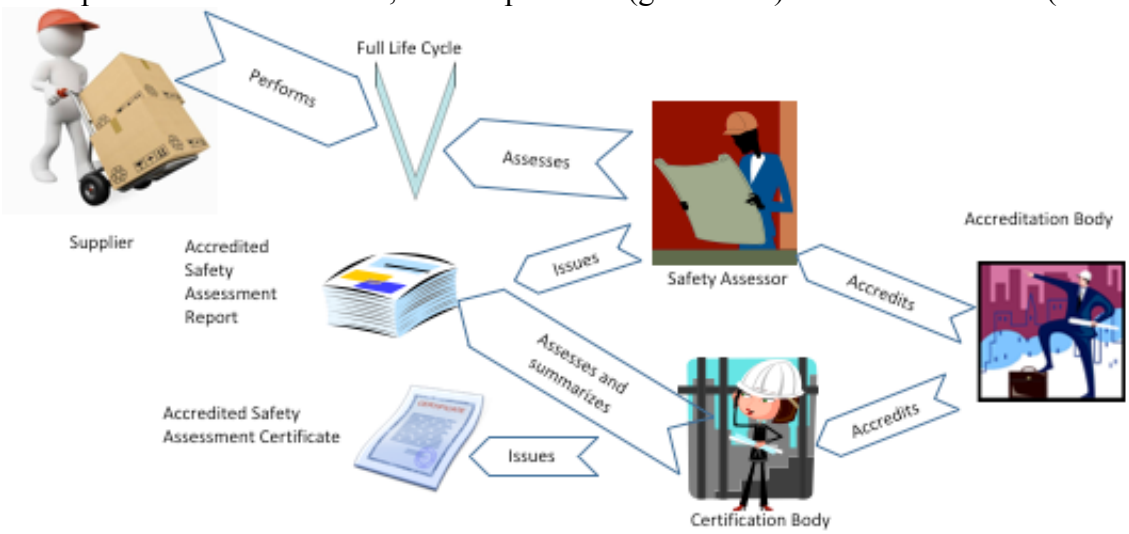

Figure 9: Improved business model – relations between actors during development.

component. This implies that outcomes of the concept phase need to be made such as item definition, hazard analysis and safety goals. When a customer for the component appears, a gap analysis between the true context and the assumed context is made and addressed. This implies that a component can be (almost) ready before a final customer is identified.

In itself such a SEooC component can never be assessed as safe unless it is evaluated in its final system at vehicle level, but the pre-qualification we consider here becomes much more valuable if business practices are established which allow the assumed context leveraged by the SEooC to be evaluated within the true context of a specific item (in the vehicle).

Hence, the prequalification of the SEooC component can be expressed with:

- a **Safety Certificate** from an accredited third party which has critically assessed the complete safety case (not only the safety manual) without tight binding to a specific item, i.e. final context. The certificate could be made publicly available, e.g. on the supplier home page.
- a **Safety Manual** containing the interface of the component. This can include a **Safety Contract** (Soderberg 2013) with assumptions and guarantees of the component that shall be fulfilled for the component to give correct service when included in an item. The Safety Manual could be made available to the customer first after negotiations with a customer have been entered, as it may contain sensitive details. Safety manuals are discussed in a later section.

The development of the component, according to the proposed business model, is illustrated in Figure 10. The key activities before negotiations take place with a customer are: component development, third party assessment and issue of certificate for the component.

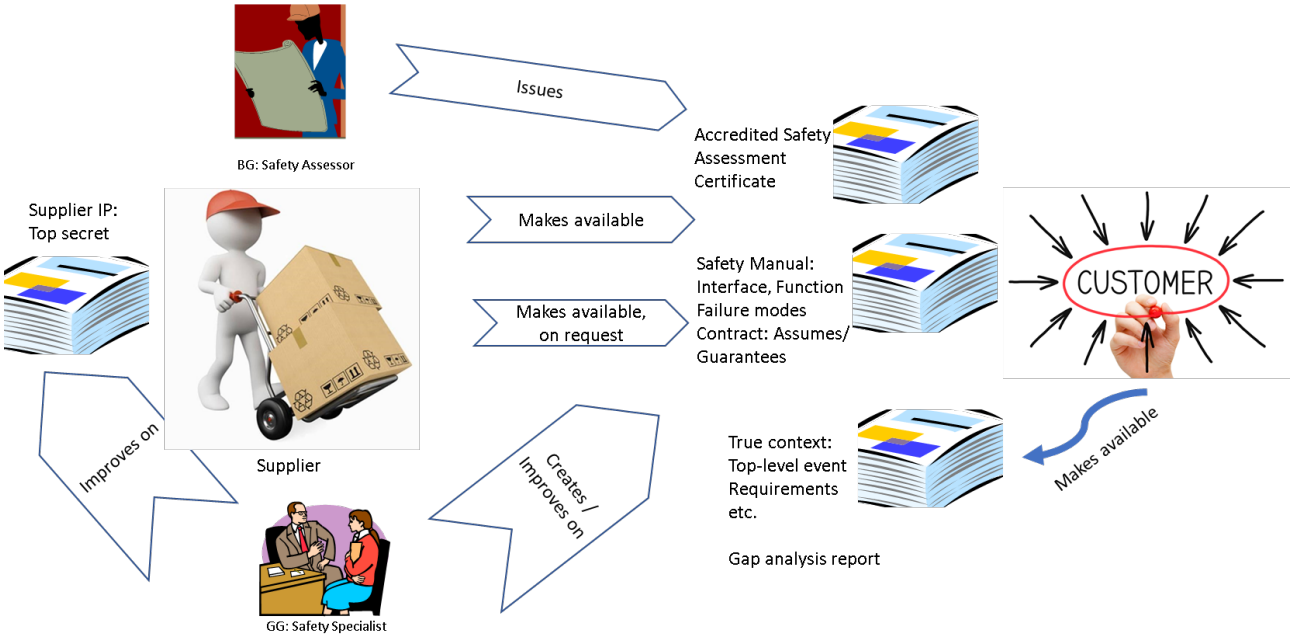

Figure 10: Roles in the improved business model

The roles in the figure of the proposed business model are:

- The Good Guy participates in the development of the component. This includes developing the **safety contract** for the component and **safety manual**.
- The Bad Guy is accredited and performs an independent functional safety assessment of the component and work products including the safety contract and safety manual. The BG issues the corresponding **FSA Report**, that will be part of the supplier´s top secret IP. The BG also issues a **safety certificate** as discussed above.
- The interface to the **Customer** is hence: the safety contract, safety manual and the safety certificate.
- The **true context,** e.g. requirements and context, is now known and a gap analysis can be performed. The supplier addresses and corrects the gaps.

When the component later is included in an item development there is a second step in which the component is assessed in an item-specific safety case which relies on the publicly available safety manual and safety certificate. For a review an assessor can still review any of the internal safety documentation (such as the TSC etc.) but the safety requirements shall be readily evaluated and validated by means documented in the safety manual.

How hard can it be?

What are the obstacles to accepting this model? For example, there may be conflicting goals, e.g. accidents can happen because what is fail-safe at the lower level (e.g. stopping the car) may become unsafe under certain conditions (e.g. another vehicle that rapidly approaching from behind). To gain confidence, the OEM therefore needs to be able to see at review how a function is implemented, i.e known specific technical details. Also, to check the maturity of safety practices at the supplier and technical capability (e.g. how tasks are scheduled to guarantee latency etc.). Even if probabilities are quoted and tested (e.g. detection or false alarm), the context in which these are tested are important, i.e. what assumptions have been made.

At Joint Review (JR) between the customer and supplier, detailed work products, e.g. the Safety Case, TSC, requirements, are normally available eyes-only. I.e. IP is partially protected. The grade of protection, however, becomes less if e.g. the review is conducted via telco or the participants in the meeting are not well-known to each other.

Potentially all these obstacles could be addressed by more details in a safety manual and including these issues in the third-party assessment. However, there will always arise questions and issues which could be answered by the OEM simply being allowed to “take a look behind the scene”. In the end, it is a question of trust. It is not certain that an OEM would trust the third party.

The SEooC concept states that assumptions can be made on the context so that development of the (system) component can take place. However, often there is are actual customers in mind, although the problem is only partially known. This leads to an SEooC that is less generic and has assumptions that ties it too closely to a particular customer. Further, since the problem is only partially known, the SEooC-development still needs to make assumptions. There will probably always be changes to the component e.g. for different OEMs. We expect that this implies recertification; which thus adds cost. A general comment on current practices is that it is difficult to know beforehand, when making assumptions on an SEooC, which requirements that will be present in the final context. It is a risk that any assumptions that are made, by a component, will not be possible to meet; either due to sensor limitations or final system architecture. The latter two are not known at the time that the SEooC is designed. These is a weakness in the application of the SEooC concept.

The safety manual in related practices

IEC 61508:2010 part 2 – Annex D gives requirements for the safety manual for a compliant component. The purpose of the safety manual is to document all the information, relating to a compliant component, which is required to enable the integration of the component into a safety-related system, or a subsystem or element, in compliance with the requirements of this standard. (Note that the terminology used in IEC 61508 may differ from ISO 26262)

Here we give examples of high-level contents of safety manuals for two types of components: hardware and software. The examples are cited from (Simpson 2010). In section "3.7 Safety Manuals" a manual for specific hardware items is given. Thus, instrumentation, PLCs and field devices will each need to be marketed with a safety manual. Contents can include (hardware):

- a detailed specification of the functions
- the hardware and/or software configuration
- failure modes of the item
- for every failure mode an estimated failure rate
- failure modes that are detected by internal diagnostics
- failure modes of the diagnostics

- the hardware fault tolerance
- proof test interval (if relevant).

In section "4.6 Safety Manuals") a safety manual for software is given. For specific reuseable software elements, such as items of code and software packages, content can include (software):

- A description of the element and its attributes
- Its configurations and all assumptions
- The minimum degree of knowledge expected of the integrator
- Degree of reliance placed on the element
- Installation instructions
- The reason for the release of the element
- Details of whether the pre-existing element has been subject to release to clear outstanding anomalies, or inclusion of additional functionality
- Outstanding anomalies
- Backward compatibility
- Compatibility with other systems
- A pre-existing element may be dependent upon a specially developed operating system
- The build standard should also be specified incorporating compiler identification and version, tools
- Details of the pre-existing element name(s) and description(s) should be given, including the version/issue/modification state
- Change control
- The mechanism by which the integrator can initiate a change request
- Interface constraints
- Details of any specific constraints, in particular user interface requirements shall be identified
- A justification of the element safety manual claims

Appendix B: Linear Time Temporal Logic (LTL)

Basic LTL

LTL extends basic propositional and first-order predicate logic with modalities that deal with time. These modalities or temporal operators may refer to the future time, e.g. `always(F)`, `eventually(F)`, `next(F)`, `until(F)` or the past time, e.g. `previously(F)`, `since(F)` etc. The underlying semantics of time in LTL is taken to be the discrete integer line (hence the term "linear time"). This model of time distinguishes LTL from other branching time modal logics. In practice, LTL is well suited to expressing black-box properties of embedded and reactive systems. Such systems can be modeled as some kind of state machine *M*. In this case, one step in LTL time is coupled to one transition in the state machine *M*, and through this relationship we can model the input/output behaviour of *M* in terms of timed stimuli and response produced by the state machine. LTL is well suited to modeling soft (or relative) real-time properties. Where the behaviour of *M* is synchronous, with respect to a clock, then LTL can also be used to model some hard-real-time properties. It is well known that in practise, LTL formulas mostly arise in certain specific stereotypical forms or *patterns*. Two common patterns are the *invariant*

always(F)
and the stimulus-response pattern
always(F -> next(G)).

In these patterns, `F` is a formula without any temporal operators (a pure propositional or predicate logic formula). Within SafeCOP, and the study of co-CPS, examples of such patterns can often be seen. For example: a first-order LTL safety invariant between successive vehicles `i` and `i+1` in a platoon (UC 6) is:

Always(tooFar > $distance_i$ & $distance_i$ > tooClose)

meaning: The distance between vehicles i and i+1 is never too far or too close. Another safety invariant is:

Always($distance_i$ > 0)

meaning: *Vehicles* `i` *and* `i+1` *never crash*. An example of the stimulus-response pattern is:

Always(Previously($timeHead_i$ < f_{HW}(packetLoss, platoonSize)) ⇒
$timeHead_i$ >= f(packetLoss, platoonSize))

meaning: *If time headway* $timeHead_i$ *between vehicles i and i+1 falls below a critical value* $f_{HW}()$*, it is restored in 1 time unit.* Here the critical value f_{HW}`( packetLoss, platoonSize)` is a function of the ambient packet loss within the wireless communication system, and also the overall size of the platoon.

LTL Extensions

Although LTL is a very powerful language able to capture several safety and liveness properties, it tends to be weak in capturing non-functional properties such as timing constraints. This is a big drawback when it comes to specify requirements of embedded systems, which has a clear impact on the verification of CPSs due to the strong presence of embedded software. Several extensions to LTL, and other temporal logics, have been proposed to address the issues with the specification of real-time requirements.

Timed temporal logics (Bouyer, 2010; Koymans, 1990) extend classic temporal logics with quantitative constructs, which makes them appropriate for reasoning about execution time requirements of safety assurance systems. It is, however, far from being straightforward to select an appropriate logic for reasoning about timing requirements of realistic systems, as exemplified by current autopilot applications. First of all, because too much expressive power may easily result in undecidability (Ouaknine,2008), this is famously illustrated by MTL, a real-time extension of LTL. A second difficulty is that, without guaranteeing the decidability of the logic, it might not be possible to devise an effective method for quantification removal. Moreover, the decidability result ensures that monitors synthesized from formulas will draw their verdicts and ideally terminate in a bounded amount of time, which is of outmost importance in safety assurance system when we are targeting execution time overloading of monitors.

Properties such as "the periodic task that performs the control loop has a duration per job no greater than 10 time units" or "the execution time that the monitor spends is less than 2 time units" cannot be specified, even in MTL, without knowing a priory the event triggering order of the system. In addition, formulas that can relate elapsed time for tasks such as "two tasks do not overload if the available load time checks a = b−10 or a < b+10" cannot be expressed in MTL.

In order to capture relevant properties involving durations, Pedro et al. introduced RMTLD (Pedro, 2015) a three-valued timed temporal logic that allows the synthesis of monitors that incrementally evaluate a system and draw a 3-valued verdict (yes, no, or unknown) in a deterministic bounded amount of time. The authors were motived by the paramount importance when specifying and developing real-time systems, mainly because the fundamental results about the reliability of this class of systems are related to ensuring that the execution time of the involved components does not miss some predetermined deadline. The concrete formal definition of this language is presented in (Pedro, 2015; Pedro, 2017), but as a short example we present below a formula written in the language of RMTLD.

$$(a \rightarrow ((a \vee b)\, \mathrm{U}_{<10}\, c)) \wedge \int^{10} c < 4$$

This specification intuitively states that "given an event *a*, the event *b* occurs until *c* occurs and, at the same time, the duration of *b* shall be less than four time units over the next 10 time units". A trace that would be validate by this specification, is for instance, *(a,2),(b,2),(a,1),(c,3),(a,3),(c,10)*. Another example, now assuming resource models, is the specification

$$\Box_{\leq\infty} \left(\mathrm{erenewal}(\omega) \overset{\bigcirc}{\Longrightarrow} (\Diamond_{=\pi}\, \mathrm{erenewal}(\omega)) \wedge \int^{\pi} \bigvee_{\tau_i \in \tau} evs^{+}(\omega, \tau_i) \leq \theta \right)$$

describing that for each occurrence of the event *erenewal(ω)* in the resource model, the duration of the other events until π time units does not overpasses the budget θ per period π, where the possible events are denoted by the term *evs*. As can be seen from this example, RMTLD is quite expressive in what time and duration constraints is concerned, which makes it deserve to be experimented with requirements of SafeCOP that address real-time properties. Furthermore, together with this language, the authors developed a correct-by-construction monitor generation algorithm, presented in (Pedro, 2015) and described in Section 2.8.4, and was validated in an autopilot scenario, described in detail in (Pedro, 2017).