Report type	Deliverable D2.4
Report name	Report on T2.3 and T2.4 and its relations to demonstrators, and recommendations for safety committees
Dissemination level	PU
Report status:	Final
Version number:	1.0
Date of preparation:	2018-03-31

Safe Cooperating Cyber-Physical Systems using Wireless Communication

Contributors (alphabetical)

Aitek Societa' per Azioni, Italy

Consiglio Nazionale Delle Ricerche, Italy

Danmarks tekniske universitet, Denmark

DNV GL, Norway

Instituto Superior De Engenharia do Porto, Portugal

Intecs SPA, Italy

Kungliga Tekniska Hoegskolan, Sweden

Maelardalens Hoegskolan, Sweden

Qamcom Research and Technology AB, Sweden

RoTechnology, Italy

Safety Integrity AB, Sweden

Universita Degli Studi Dell'Aquila, Italy

Revision history

0.1	2017-06-30	First draft, focused on safety analysis
0.2	2017-10-25	Circulated to the whole consortium for feedback and alignment
0.3	2018-02-16	Updated draft with details about the model-based tool support
0.4	2018-02-27	Final draft circulated for review among the partners
0.5	2018-03-12	Final quality check
1.0	2018-03-31	Final release

Table of contents

1 Introduction

1.1 Background

In recent years, a new class of systems has emerged, distinguished from traditional monolithic systems. In general, these systems combine characteristics of autonomous and independently developed components, often large and complex, in many cases geographically distributed, with emergent behaviour and evolutionary nature. We refer to these systems as systems-of-systems (SoS). The main idea in such systems is to provide a set of enhanced and improved services, based on the services provided by participating components. SoS often use some communication infrastructure, involve multiple stakeholders, dynamic systems definitions, and have unpredictable operating environments. In the literature, they are sometimes also referred to as Cooperative Open Cyber-Physical Systems (CO-CPS) since they combine cyber- and physical elements to provide mission-critical services.

CO-CPS provide solutions for several societal challenges. For example, use of cooperative robots can reduce the amount of the physical labour in hospitals, directly impacting the cost of the overall healthcare; cooperative boats may improve navigation safety; using vehicle-to-infrastructure and infrastructure-to-vehicle communication modes, one can deliver critical-up-to-date real-time road weather data to increase traffic safety. These are only a few among numerous examples where CO-CPS can be employed.

The emerging behaviour of complex CO-CPS arises when the fulfilment of a need, or a set of needs, requires the interaction of multiple systems. For example, instead of relying on one single system to meet a set of performance requirements, one could consider more encompassing solutions that meet a broader set of needs. It becomes a challenge to model and reason about such dynamic behaviours.

Since both humans and the environment are involved in CO-CPS, it might happen that the system causes harm to some of them. To prevent this, it is important to obtain design-time safety assurance evidence to guarantee that the system manages risk acceptably. Therefore, engineers must prepare safety cases, combining evidence with a safety argument, showing that their processes and work products conform to a relevant standard. The design-time evidence enables developers to foresee the possible safety-related problems that can arise from the interaction between the system and its environment, to show that these interactions do not pose an unacceptable risk. On the other hand, the safety engineering processes that consider certification are very expensive, and can add a development cost overhead of 25 to 100% (IBM, 2010).

SoS are constituted of interconnected systems, equipped with interfaces to the Internet, which potentially (Eames, 1999) (SRA, 2015) might endanger the system, as it exposes them to a wide range of security threats. If we just look at a modern car, equipped with the latest technology, simple security glitches might enable a potential attacker to take control over safety-related functionalities (ISO, 2010). In the early 1990s, researchers identified the need to harmonize activities related to safety and security issues. However, to the best of our knowledge, there is still no standardized approach to address safety and security in combination, see the references and discussions in the SafeCOP Deliverable D2.3 “Mitigation of security threats that may jeopardize safety”.

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

This deliverable reports on tasks T2.3 “Safety analysis for CO-CPS” and T2.4 “Conditional composable safety case models”. Deliverable D2.2 “Report on T2.2 on safety assurance conceptual model and its relations to the demonstrators and recommendations for safety committees” has presented an overview of the SafeCOP

Safety Assurance framework. In this deliverable, we go into the detail of the methods presented in the overview in D2.2. Note that the title of this deliverable mentions "relation to demonstrators"— this has been clarified instead in SafeCOP Deliverable D2.2 and "recommendations for safety committees"—which has also been mentioned in the SafeCOP Deliverable D2.2 and is also the topic of the SafeCOP Deliverable D6.3 "Dissemination, exploitation and certification report". Hence, instead of repeating here the text from D2.2, we refer instead the reader to:

- Section 3 of D2.2 for the "Relation with other work packages (WP3, WP4, WP5)";
- Section 4.1 of D2.2 for the "Contributions" of the R&D&I work we have performed on Safety Assurance;
- Section 4.2 of D2.2 for the "Recommendations to standard committees" and
- Section 4.3 of D2.2 for the relation to "Demonstration".

Thus, in this deliverable, we propose an approach for the safety assurance analysis of CO-CPS (Cooperating Open Cyber-Physical Systems), related to task T2.3 and we propose an approach for the modelling of the safety cases, related to task T2.4. The toolset that makes use of these models and supports the SafeCOP Safety Assurance framework is reported in WP4 "Platform and tool support for safety assurance". In relation to T2.3, this report gives practical guidelines on how to perform a safety analysis for CO-CPS. It is based on the current state-of-the-art within safety assurance as described in the SafeCOP Deliverable D2.1 (SafeCOP D2.1, 2016) (HSE, 2017), systems engineering principles (ISO, 2015) (Hafver, 2015), and current research performed by DNV GL in connection with the update of the recommended practice for technology (and systems) qualification (DNV GL, 2011) (Berner, 2016). In relation to T2.4, we present how we model the safety cases, which are conditional because they place demands on other systems/items, and composable. Our approach is to extend the safety case models developed in other ARTEMIS/ECSEL projects such as CHESS and CONCERTO using contract-based models and by providing support for development of CO-CPS.

1.3 Overview of the Report

Figure 1 presents the SafeCOP safety assurance concept as proposed in the SafeCOP Deliverable D2.2.

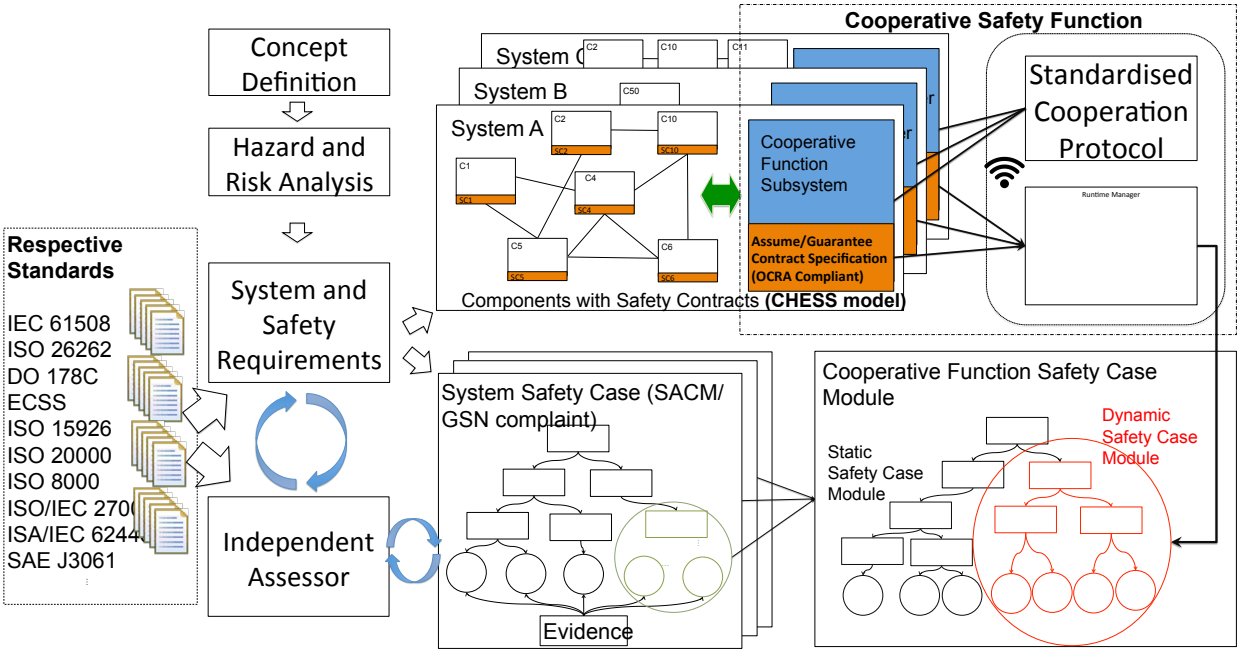

Figure 1: The SafeCOP safety assurance concept, from SafeCOP Deliverable D2.2

The CO-CPS addressed in SafeCOP consist of several 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, referred to as a cooperative function in Figure 1. Let us use the figure to present the overview of this report.

The left part of the figure (boxes labelled “Hazard and Risk Analysis”, “System and Safety Requirements” and “Independent Assessor”) is concerned with safety assurance analysis. Chapter 2 gives a brief overview of the recommended approach and principles on how to perform a safety assurance analysis of CO-CPS. It explains a hierarchical approach to ensure that the mission requirements of the system are appropriately allocated by the necessary functional architecture, which in turn is allocated to organizational and technical resources. The approach is based on STAMP/STPA (Leveson, 2011) (DNV GL, 2016a) and CESAM (CESAM, 2017) (Borza, 2011).

Chapter 3 outlines how to establish the mission objectives and the mission constraints, i.e. define the targets and boundaries for the system performance. Chapter 4 describes how the mission objectives and constraints can be broken down into functional requirements. Such functional requirements define the success of the mission, and can be used to identify hazards, i.e. how violations of constraints or failure to meet objectives may occur and potentially lead to accidents/losses.

Chapter 5 considers the physical resources (hardware, software, human) allocated to provide the required functions. The objective of the resource analysis is to check that adequate resources are in place, or establish which organizational and technical resources are needed, to ensure that the identified requirements can be met. By identifying which system resources are involved in providing a function or required to fulfil various functional requirements, it is possible to identify how functions may fail to be delivered and subsequently could lead to accidents or mission failure. The next step is to determine how such failures can occur and when they may be introduced, to determine how they can be handled and if the system is sufficiently safe (i.e., that the risk is tolerable). Some failures can be handled in the design phase, through alterations to the system architecture and control algorithms. Some failures can be prevented through continuous monitoring of the system and its environment during operation. Other failures can be prevented through imposing operational constraints/restrictions.

In SafeCOP we intend to use contract-based design to facilitate the independent development of cooperative safety functions. We will build upon results on contract-based design from ARTEMIS projects such as CHESS and CONCERTO, which focused on developing the CHESS-toolset—a tool for model-driven and component-based development of high-integrity systems. Our solution in SafeCOP is to enrich the CHESS-toolset by providing support for development of CO-CPS, see Chapter 6 for details. 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.

Throughout the process of assessing the safety of a CO-CPS it may be necessary to iterate part of, or the whole, process. The chapters of this document use examples from the SafeCOP use cases to illustrate the recommended approaches. For example, UC2 “Cooperative bathymetry with boat platoons” is used in Chapters 3-5 and UC5 “Vehicle-to-Infrastructure (V2I) cooperation for traffic management” is used in Chapter 6 and the full details of the application of SafeCOP Safety Analysis on UC5 is presented in the Appendix (Chapter 7).

2 Safety assurance of CO-OPS – Approach and principles

The safety assurance approach described in this document uses elements from the System Theoretic Accident Model and Process (STAMP), FAST and the CESAMES Systems Architecting Method (CESAM), elaborated in this chapter. The approach is also founded on a risk-based safety philosophy, where risk is understood as the consequences of an activity, with associated uncertainty.

In this context, the contribution of SafeCOP is that (i) we have shown the current safety analysis approaches used in the standards, see Chapter 7 (Appendix), are not capable of identifying the critical hazards arising from the cooperative nature of CO-CPS and (ii) we have proposed an innovative way of applying STAM and CESAMES elements to elicit the critical hazards specific to CO-CPS.

2.1 Risk and safety

Safety can be defined as “freedom from risk which is not tolerable” (ISO/IEC, 2014). Per this definition, to determine if an activity is safe, one requires a description and an evaluation of risk. The Society for Risk Analysis provides several conceptual definitions of risk, including: “risk is the consequences of activities and associated uncertainty”, and “risk is the potential for realization of unwanted, negative consequences of an event” (SRA, 2015). ISO defines risk as “the effect of uncertainty on objectives” (ISO, 2010). While emphasizing different aspects, the essence of all these definitions is that risk is a combination of consequences and associated uncertainty. Often, the consequences considered are restricted to negative consequences, where ‘negative’ is defined relative to some objectives.

By defining safety in terms of tolerable risk, safety is framed as a cost-benefit problem, where the negative effects of an activity are weighted against perceived benefits. This view on safety is articulated by the principle of ALARP: that risk should be maintained ‘as low as reasonably practicable’, i.e. that risk should be reduced until the sacrifices necessary to reduce it further become disproportionate to the risk to be avoided (HSE, 2017).

In order to compare risk, a set of quantitative or qualitative risk measures/metrics are typically used, such as risk matrices (i.e., indicating the likelihood and severity of consequences/events along the two axes), ‘risk intensity’ measures such as expected outcome, or curves showing the probability/frequency of consequences of a certain severity. Such risk measures/metrics allow comparison of risk and can be useful tools to help make priorities. However, it should be realized that any risk assessment relies on a series of assumptions and choices (Hafver, 2015), (Berner, 2016), (DNV GL, 2016a):

1. Firstly, when defining the scope of the risk assessment, deliberate or unconscious choices are made regarding which consequences are relevant to address, influenced by the objectives of the operator, requirements of regulators, mandate of risk assessors, etc.
2. Secondly, models are introduced to represent the processes and systems under consideration, and methods are selected to express and propagate uncertainty through these models, to predict the consequences of interest. To simplify the analysis, conditions that are known to vary over time may be represented as fixed in the analysis, e.g. by using expected or most likely values, or by considering ‘worst case’ scenarios.
3. Finally, the results of risk assessment are summarized in terms of selected risk measures (e.g. expected number of fatalities per year, frequencies of accidents over a certain magnitude, probability of event within certain time), which are sometimes displayed visually (e.g. in risk matrices, as distributions or as risk ‘level gauges’). A risk measure is some descriptor of the consequences and associated uncertainty.

This means that risk-measures used to describe or communicate risk only convey some aspects of risk, and do not convey what knowledge or which assumptions they are based on. Different assumptions and choices can lead to different descriptions of the same risk, and additional risk may 'hide' behind assumptions. Risk measures can be useful for ranking and comparing risks, but if critical assumptions are inaccurate or wrong, this could affect the ranking of risk and outcome of decisions. The outcome may also be sensitive to the choice of risk measure. How confident a decision-maker can be about decisions depends on how reasonable the underlying assumptions are, how well decision options can be differentiated when uncertainty about assumptions is taken into account, and how relevant the reported risk measures are for the decision at hand. To a decision-maker, confidence in results can be as important as the result itself, and, therefore, evaluation of the criticality of assumptions is an essential part of a risk assessment. A choice or assumption could be considered critical if realistic deviations from it could have significant effect on the results.

Hence, in addition to providing sensible risk measures to rank risks, it is important to identify and list assumptions made in a risk assessment, and to evaluate their criticality. Assumptions may be critical because our beliefs/predictions are sensitive to deviations from them, because we have reason to believe that there will be deviations from the assumptions, or because our knowledge is poor, and new knowledge therefore is expected to impact our judgement. An approach for evaluating assumption criticality along these three dimensions (sensitivity with respect to assumption, belief in deviation, and strength of knowledge) is presented in (Berner, 2016).

Figure 2 illustrates schematically how risk measures only cover a subset of the full risk picture. Figure 3 hints at the parts that are lost. Figure 4 shows how the strength of knowledge dimension can be accounted for when reporting risk.

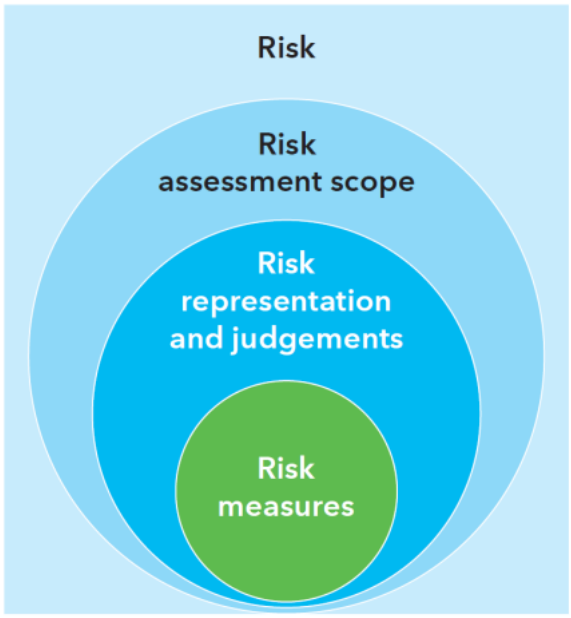

Figure 2: Risk measures do not provide a complete picture of risk, due to choices and assumptions made during the risk assessment process (DNV GL, 2016a).

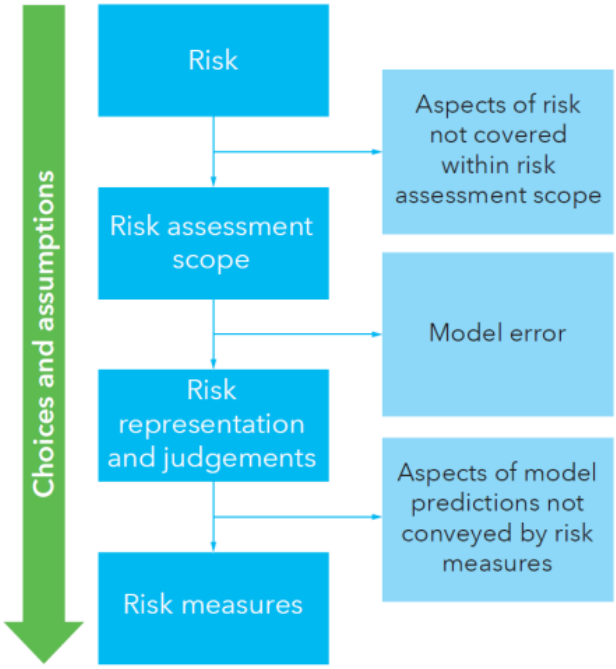

Figure 3: Assumptions made during a risk assessment, whose validity may change during operation (DNV GL, 2016a).

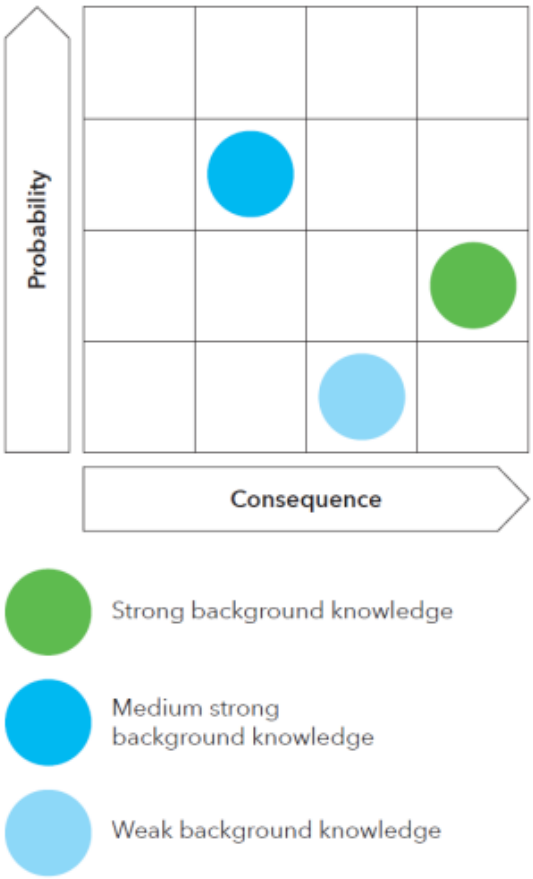

Figure 4: Example of a risk matrix providing an indication of the strength of the background knowledge that the computed probabilities and consequences are based on (DNV GL, 2016b).

2.2 STAMP and STPA

STAMP is a recent accident model that was first introduced by Nancy Leveson in 2003 (Leveson, 2003). This new causality model is based on systems theory focusing on enforcing behavioural safety constraints rather than preventing failures.

Briefly, systems theory is founded on two pillars, or pairs of ideas: i) emergence and hierarchy, ii) communication and control (Checkland, 1981). In the first pair of ideas, socio-technical complex systems are considered and assessed as a hierarchy of levels of organization, which starts from the less complex at the bottom and ends with the most complex at the top. Each level is characterized by emergent properties in the idea that at a given level complexity, and the properties and characteristics of that level, are irreducible (Leveson, 2011). The emergent properties are derived from the constraints upon the degree of freedom of the components on one level of the hierarchy. Based on systems theory and STAMP, safety can be obtained only in the context of a whole.

The second pillar of systems theory is that of communication and control. Control is considered as the imposition of constraints, in order to keep an activity at the desirable state. The four elements needed in order to control a process are (Leveson, 2011):

- Goal condition (set-point)
- Action condition (actuator)
- Model condition (model/algorithm of the controller)
- Observability condition (sensor)

as shown in Figure 5.

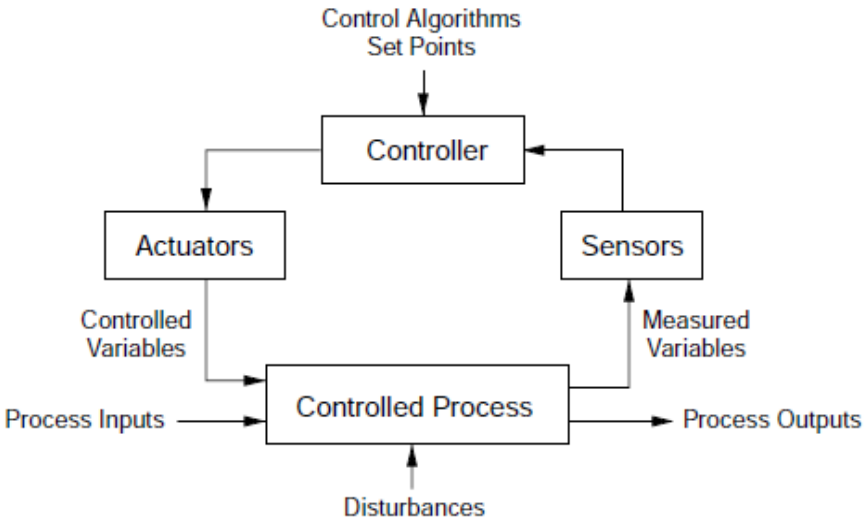

Figure 5: A standard control loop (Leveson, 2011).

STAMP is able to assess complex sociotechnical systems, which concern us today, by thinking of safety as a control problem rather than a reliability one. STAMP is assembled by three elements: safety constraints, hierarchical safety control structures, and process models. Each level in the hierarchy imposes constraints to the one below. A lack of control in one level influences the remaining levels directly or indirectly through the channels of communication (interfaces) between each other (Figure 6).

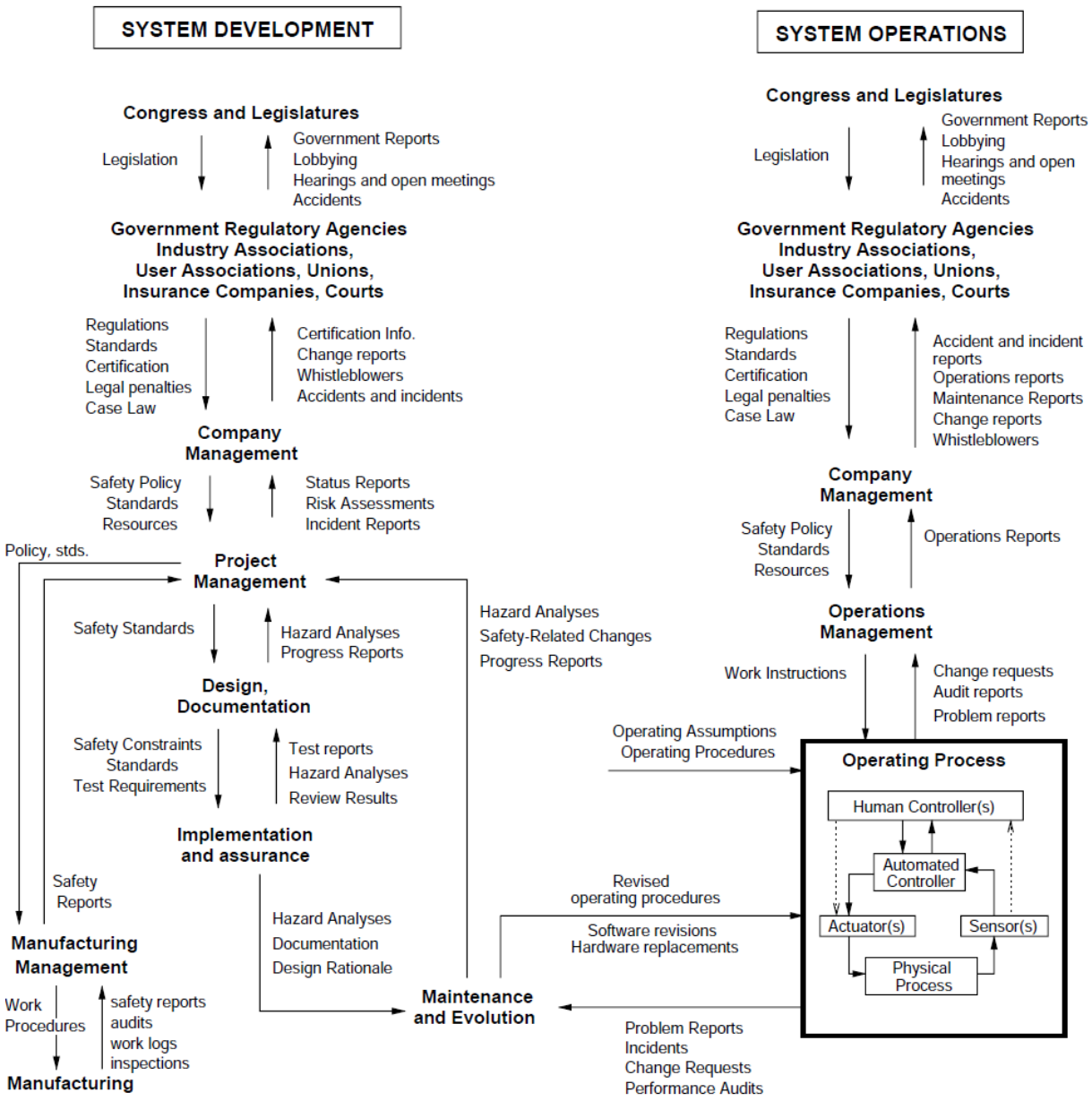

Figure 6: An example of general form of a model of sociotechnical control (Leveson, 2011).

STPA is a hazard analysis based on STAMP that focuses on identifying scenarios and causalities about how the constraints of the system can be violated. One of the main advantages of STPA is that it can be used both in design or operation. The framework is based on four mains steps:

2.2.1 Identification of Accidents, Hazards and Safety Constraints

Based on STAMP theory an accident is defined as:

> *"An undesired or unplanned event that results in a loss of human life, or human injury, property damage, environmental pollution, mission loss, etc."*

Accidents are defined based on the scope of the system risk assessment and its purpose. This can be done even by customers and government or general stakeholders and groups of experts. Accidents are not related only with the loss of human life, but to any kind of an undesirable performance and interaction that can lead

to any valuable loss. For example, in the oil and gas industry, stop in production due to blockage of a valve could be considered as an accident, although there is no loss of human life.

In this system, theoretic framework, hazards are considered as:

> *"A systems state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss)."*

For instance, in relation to the oil and gas example, a hazard could be the bad quality of the fluid passing through the valve: this, combined with environmental disturbances and hazardous conditions (extreme low temperatures), might lead to a loss or accident. Hazards are identified in a top-down concept, starting first from the system, and if they cannot be eliminated, they must be mapped in a subsystem or component level.

The last part of the first step of the STPA framework is the identification of the constraints. These are defined based on, and parallel to, the hazards and their main aim is to prevent a hazard from occurring. This is an important phase of the analysis since the failure modes of the system will be based on the failure of implementation of the constraints. In the oil and gas example, a constraint could be flow maintenance and controlling of the process quality.

2.2.2 Draw the control structure

This is a critical phase of the method where the control structure of the system is defined. This will reveal the interactions between components and operations, both on a high-level view of the system and the lower ones. The control structure could be applied either in the design phase or in the operations design when one control structure exists already. In the former case the design of the control connections and actions are based on the constraints, whereas in the latter case the control structure already exists, and it is just modelled. In some cases, it might be useful to start from scratch even in existing structures, aiming at revealing potential design modifications that might (further) reduce the hazards and failures in the system inherently.

When designing the control structure of a system one should define first what are the main components of the system and translate them into processes that support the whole performance of the system. These processes are linked to some controllers (either human or software) through sensors. Depending on the feedback and the "algorithms" of the controller, decisions are made about whether or not control actions are required. These actions are transferred to some actuators that are responsible to implement them on the process. All the aforementioned elements create a control loop similar to that shown in Figure 7. However, in a big and complex system, there will be multiple loops and interdependences among them.

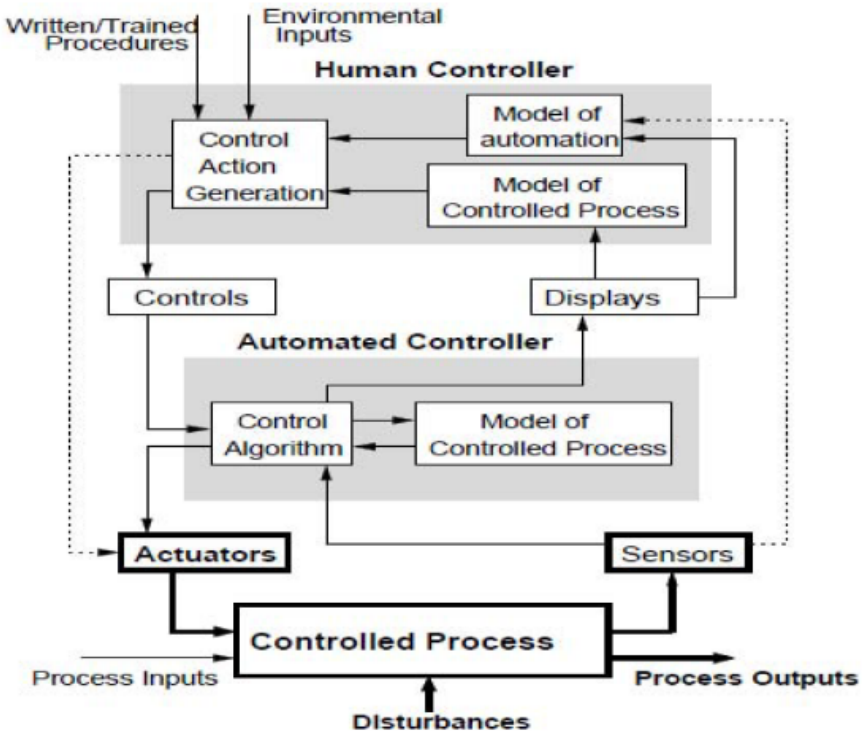

Figure 7: A control structure that combines a human controller with an automated controller, both controlling a physical process (Leveson, Engineering a Safer World: systems thinking applied to safety, 2011).

2.2.3 Identification of Unsafe Control Actions (UCA)

The main goal of STPA is to discover causalities on how hazards could occur. Hazards are due to lack of an adequate enforcement of the safety constraints. Based on the examined method, these can occur because:

- A control action required for safety is not provided or not followed.
- An unsafe control action is provided.
- A potentially safe control action is provided too early or too late, that is, at the wrong time or in the wrong sequence.
- A control action, required for safety, is stopped too early or applied for a too long time

In practice, UCA is performed by choosing a control action, combined with different states of the variables of the examined control process, and creating a context table that reveals all the unsafe control actions. Each unsafe control action is linked to the potential hazards and accidents that this can lead to. This provides a systematic way to analyse all the control actions. See example in Table 1.

Controller	Water Valve			Accident	Hazards		
				A1	H1.1,H1.2,H1.3		
Control Action	Open water valve			A2	H2.1,H2.2,H2.3		
				A3	H3.1,H3.2,H3.3		
				A4	H4.1,H4.2		
	Process Model Variables			Control Actions (CA) hazardous?			
	Water Level	Oil Level	Pressure	CA NOT provided	CA provided	CA provided too late/early	CA stopped too late/early
1	Over the maximum limit	Between the acceptable limits	Between the acceptable limits	H1.1	H3.2, H3.3	H2.3	Not hazardous
2	Over the maximum limit	Between the acceptable limits	Below the minimum level	H3.2, H3.3	H4.2	H1.1	H3.2,H3.3
3	Over the maximum limit	Below the minimum level	Between the acceptable limits	H4.2	H1.1, H3.2	H2.2	No hazardous
...							
n							

Table 1: Example of UCA table.

2.2.4 Identification of causal factors of UCA

In this step, one examines how the unsafe control actions, as identified in the previous step, can occur. In other words, the unsafe control actions should be linked to the different parts of the control loop of the process in order to examine which of them could cause unsafe control action (see Figure 8). In this way, the method can reveal gaps and lack of existing designs or suggest those that could enhance the safety of the system. One advantage of the method is that it can identify potential conflicts in case of multiple controllers on the same controlled processes.

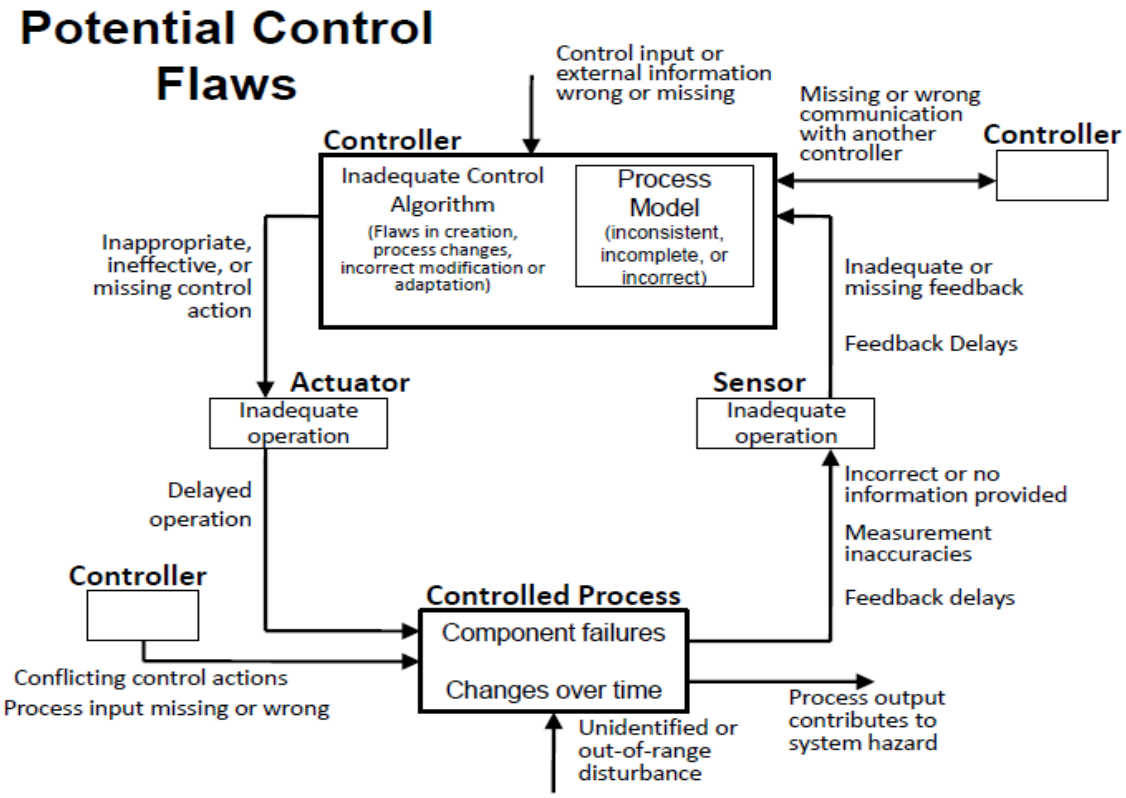

Figure 8: A classification of control flaws leading to hazards (Leveson, Engineering a Safer World: systems thinking applied to safety, 2011).

2.3 CESAM

CESAM is a system architecture and modelling framework, developed by The Center of Excellence on Systems Architecture, Management, Economy & Strategy (CESAMES). The following is a summary based on (CESAMES, 2017) (Borza, 2011).

In CESAM, a system is analysed from three different and complementary perspectives (summarized in Figure 9 and Table 2):

- *Operational perspective: Why (or for whom) does the system exist?*
  The operational perspective concerns the mission of the system (e.g., "System S shall do X to Y", where Y is external to the system S). Hence, operational analysis concerns the interactions between a system and external systems (i.e., the system's environment, including users), and requires the definition of the system boundaries. Definition of system boundaries is typically the first decision that must be made when designing or analysing a system.
- *Functional perspective: What is the system doing?*
  A function transforms a set of inputs into a set of outputs. The functional perspective concerns the input/output dynamics of a system (i.e. what is being done), without referring to the details of how this is achieved. In contrast to the operational perspective, the functional perspective only refers to the system of interest, without involving any external systems. (Note that the word function is often used with reference to technical systems, but here it is used in a broader sense, to also include processes or tasks.)
- *Constructional (resource) perspective: How is the system doing it?*

The constructional perspective refers to the concrete resources that forms the system, often termed components or elements of the system, including technical, software and human resources. Note that there is not necessarily a one-to-one relationship between function and component – the realization of a single function may depend on several components.

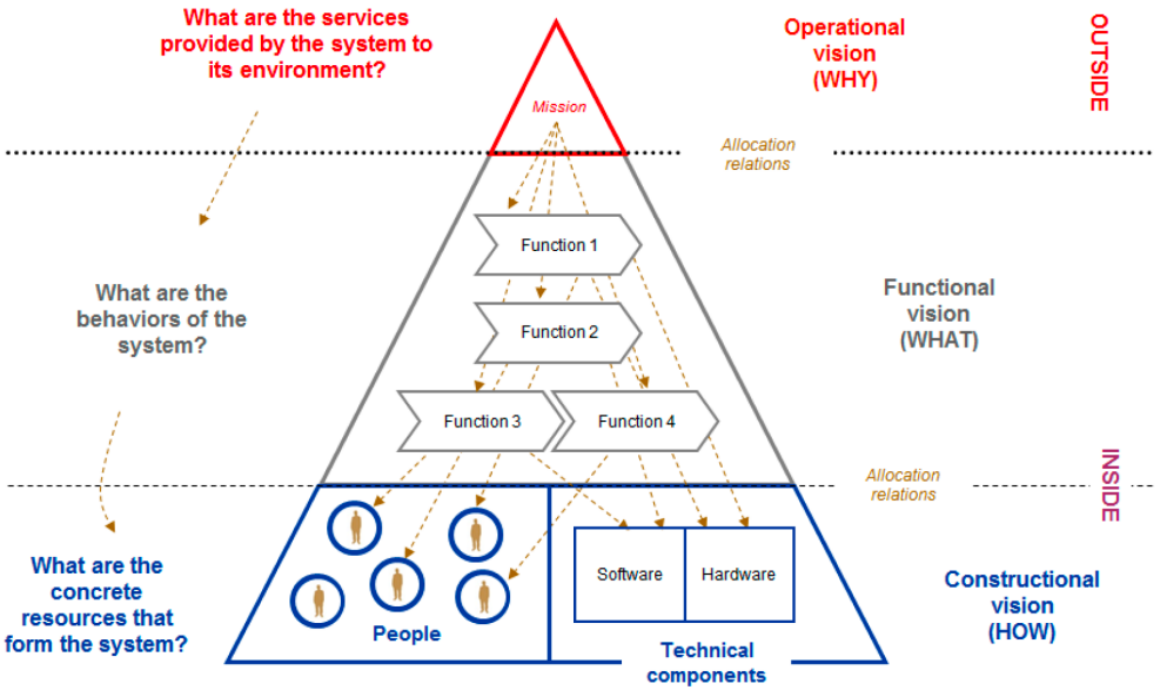

Figure 9: The CESAM systems architecture pyramid. from (CESAMES, 2017) (Borza, 2011).

Point of view	***Questions***	***Analysis***	***Keywords***	***Models***
Operational	For whom/what? Why?	Analysis of the environment of the system	Missions, use case, requirements, operational context, life cycle	Interactions of the system with its external environment
Functional	What?	Abstract analysis of the system	Function, task, process, mode	Abstract functions of the system
Physical	How?	Concrete analysis of the system	Component, part, architecture, configuration	Concrete components of the system

Table 2: Overview of the CESAM system perspectives, from (Torres, 2017).

2.4 FAST

The Functional Analysis System Technique (FAST) is a method for showing the logical relationship between functions of a system. FAST is part of the "Value Analysis" approach[1], and is a "*technique to develop a graphical representation showing the logical relationships between the functions of a project, product, process or service based on the questions 'How' and 'Why'*". Going from left to right in the FAST diagrams shows *how* a function is achieved through a set of other functions. Going from right to left answers *why* a function is needed. In addition, in the vertical direction, supporting functions may be listed and further expanded in the horizontal direction. An example is shown in Figure 10.

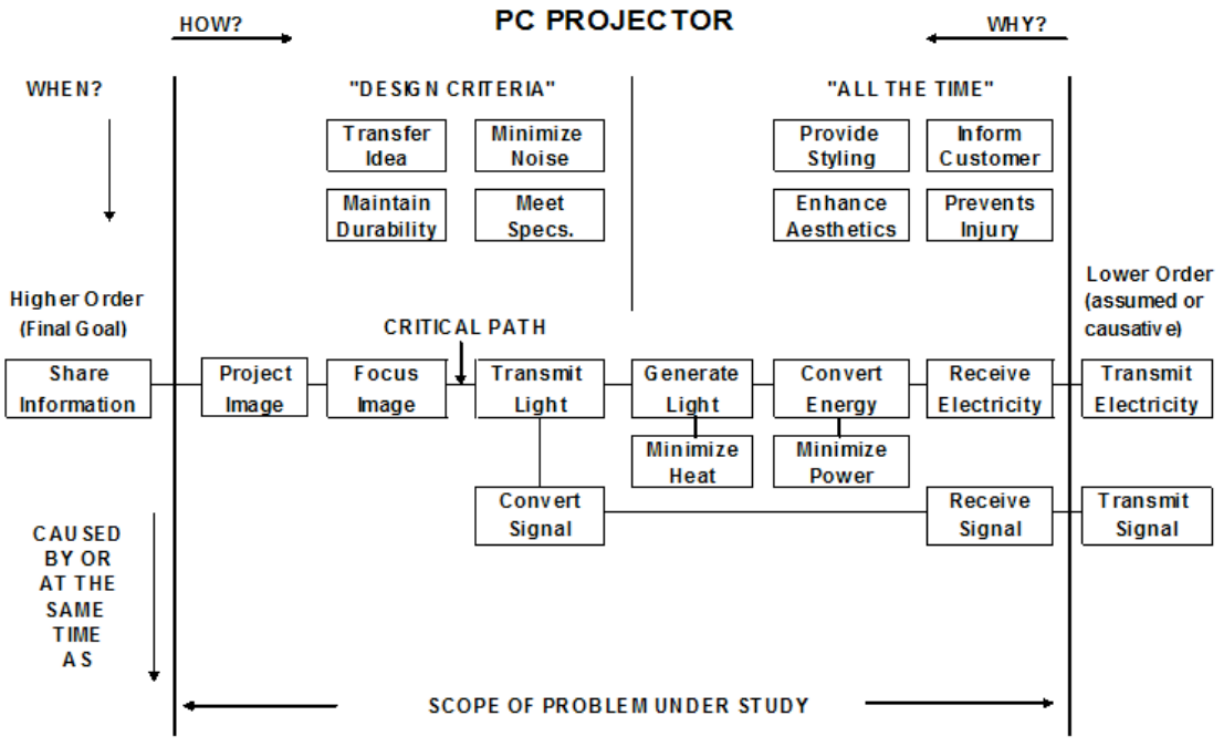

Figure 10: Example of a FAST diagram for a PC projector, from (Borza, 2011).

2.5 Overview of safety assurance approach

In this report the three CESAM system perspectives defined above will be used together with FAST and STAMP/STPA to assess safety in a three-step approach (where the three steps are inspired by CESAM):

1. Operational analysis (chapter 3): The purpose of the operational analysis is to identify/define mission requirements.
2. Functional analysis (chapter 4): The purpose of the functional analysis is to map out the functions required to fulfil mission requirements and avoid accidents. The breakdown of mission requirements into lower level functions is inspired by FAST (section 2.4). STAMP/STPA (section 2.2) is then used to identify the corresponding hazards and derive functional requirements. Resource and control struc-

[1] http://valueanalysis.ca/vadefinitions.php?section=definitions#Value%20Analysis

ture analysis (chapter 5): The objective of this step is to establish which resources and control structures that are needed to ensure that the identified functional requirements can be met, in order to avoid accidents or mission failure. The approach makes use of both traditional risk management approaches (section 2.1) STAMP/STPA (section 2.2) for identifying emergent failure modes inherent to complex CO-CPS.

Following this approach, mission requirements will be translated into functional requirements, which in turn sets requirements to the technical (hardware and software) and organizational resources used to deliver the functions.

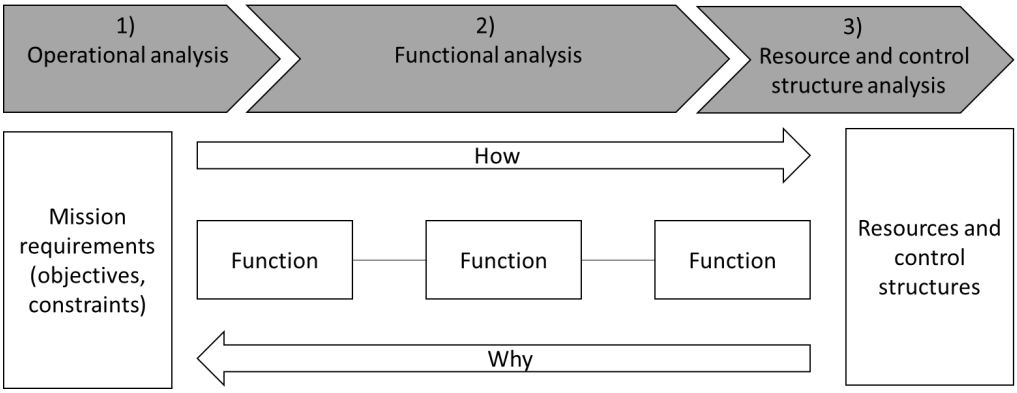

Figure 11: The steps of the proposed safety assurance approach.

3 Operational analysis

An activity will have a set of desired and unwanted consequences. Some of the consequences are known, and some are unknown. These consequences of an activity, with associated uncertainty, is called risk (SRA, 2015). Defining safety as 'tolerable risk' (ISO/IEC, 2014), the purpose of safety assurance is to ensure that the risk is tolerable. To this end, the objective of the operational analysis is to establish the mission (or system) requirements, including mission objectives (what we want to achieve) and mission constraints (what we want to avoid). The mission objectives reflect the purpose of the activity, while mission constraints define the boundaries of the system performance, to avoid or limit the extent of unwanted consequences to a tolerable level.

3.1 Establishing mission requirements

Defining the desired and unwanted consequences of an operation is a subjective judgement, but is often guided by regulatory requirements, company policies etc. Consequences may pertain to the system itself (e.g., damage to facilities), humans (e.g. injury or loss of life), the environment (pollution etc.) or the business (loss of production, financial losses, damage to reputation etc.).

Establishing the mission requirements comprises the definition of the top-level mission objectives (what the system should do) and mission constraints (what the system should not do).

Mission constraints may be divided into safety constraints or regulative constraints, related to constraints imposed by external parties, although regulative constraints are often based on safety considerations.

The following steps may help guide the establishment of mission requirements:

- Defining the operation and system under consideration and its boundaries (restrict analysis scope)
- Identifying stakeholders or external systems/environments that will be affected (both positively and negatively) by the operation.
- Identifying relevant regulations / policies and requirements (other imposed constraints)
- Identify acceptable performance of the system (e.g. quality, efficiency, reliability, etc.)

To ensure an exhaustive list of mission requirements, it can be useful to be generic and not too specific at this stage. The mission requirements will be broken down into functional requirements and requirements to the physical resources that make up the system in the following chapters.

Example Use Case 2 (UC2):

Background info:

UC2 is a typical bathymetry operation utilizing platooning of unmanned vessel(s) (USV) with a manned master vessel. The operation is often executed in shallow or near shore areas, where the marine traffic and human activity is higher. The reason behind one or several small USVs, operating over a larger area than one vessel can cover, is achieving a higher efficiency. Platooning at a relatively fixed position relative to the master vessel makes the overall safety in the operation high and the human captain and remote operator able to monitor and intervene in case of hazardous situations. As the USV's autonomous capacity increases, the strict platooning can be loosened up and the area of operation can be larger.

The operations of bathymetric surveys benefit from platooning a manned ship with one or several USVs, by safely increasing the efficiency. Near shore activity can be adversely affected by bathymetric surveys, but the increased efficiency of a platooning operation with small USVs under manned monitoring do at least not increase the negative effects. The normal requirements for marine crafts apply and there are less or no regulations for USVs. The bathymetric data are classified and the needed level of security must be implemented to safeguard these data.

Mission objectives:

- Map the seafloor

Mission constraints:

- Safety constraints:
  - Not harm anything/anyone (life, property or environment)
  - Not damage or loose USV (subset of above – included for clarity)
- Regulative constraints
  - Abide to laws, rules and regulations

3.2 Identifying unwanted consequences

Not meeting the mission objectives amounts to mission failure. Similarly, the failure to meet a safety constraint is often termed an accident, while failure to meet a regulative constraint may be termed a violation.

Mission failure, accidents and violations can be identified by negating the identified mission objectives and mission constraints.

Example Use Case 2 (UC2):

Mission failure:

- Seafloor not (adequately) mapped

Accidents:

- Life property or environment harmed
- USV damaged or lost

Violations:

- Laws/rules/regulations not followed

Once the unwanted consequences have been defined, it is also useful to rank them according to severity, as trade-offs sometimes need to be made when mission objectives and constraints become conflicting. This is a subjective judgement, but, for example, harm to life, property or the environment is usually considered more serious than mission failure or breach of rules. For the UC2 example above, it follows that it would be ok for the USV to not fulfil the mission requirement of data quality and coverage to avoid a collision. Similarly, it will not be ok to breach e.g. speed limits in traffic rules to reach performance requirements, but it will be ok to breach traffic rules to avoid a collision.

It is not necessarily possible to make a strict prioritization of mission requirements, as the ranking may be situation specific. However, it is important to recognise possible conflicts, and handle these as best as the situation allows. This will be addressed in more detail in Section 5.

4 Functional analysis

The objective of the functional analysis is to establish which functional requirements must be in place to ensure that the mission requirements (mission objectives and constraints) are fulfilled. It is a formal process of breaking the mission requirement down in specific functions that together will enable the system to not only to perform its mission, but also to do so within the defined mission constraints. This can be done using the FAST method described in section 2.4, together with STAMP/STPA, described in section 2.2, for identifying hazards.

4.1 Constructing function trees

The mission requirements may be considered as top-level system functions, and these functions may be broken down into sub-functions that are necessary to deliver the system functions. The sub-functions can be further broken down into sub-sub-functions etc. in a similar manner. A functional hierarchy may be constructed and represented as a tree graph, as illustrated in Figure 12, Figure 13, Figure 14 and Figure 15 for UC2.

The functional breakdown is inspired by the FAST method (Borza, 2011). Moving left to right in the tree corresponds to asking the question ‘how?’ Moving right to left in the tree corresponds to asking the question ‘why?’

Note that many of the functions appear several times and are linked to more than one mission requirement. Some functions are needed both for reaching the mission objective and to satisfy the mission constraints. This forces us to view mission accomplishment and safety in conjunction.

The breakdown of functions should continue to a granularity for which it is possible to establish and assign a treatable resource and control structure as detailed in Section 5.

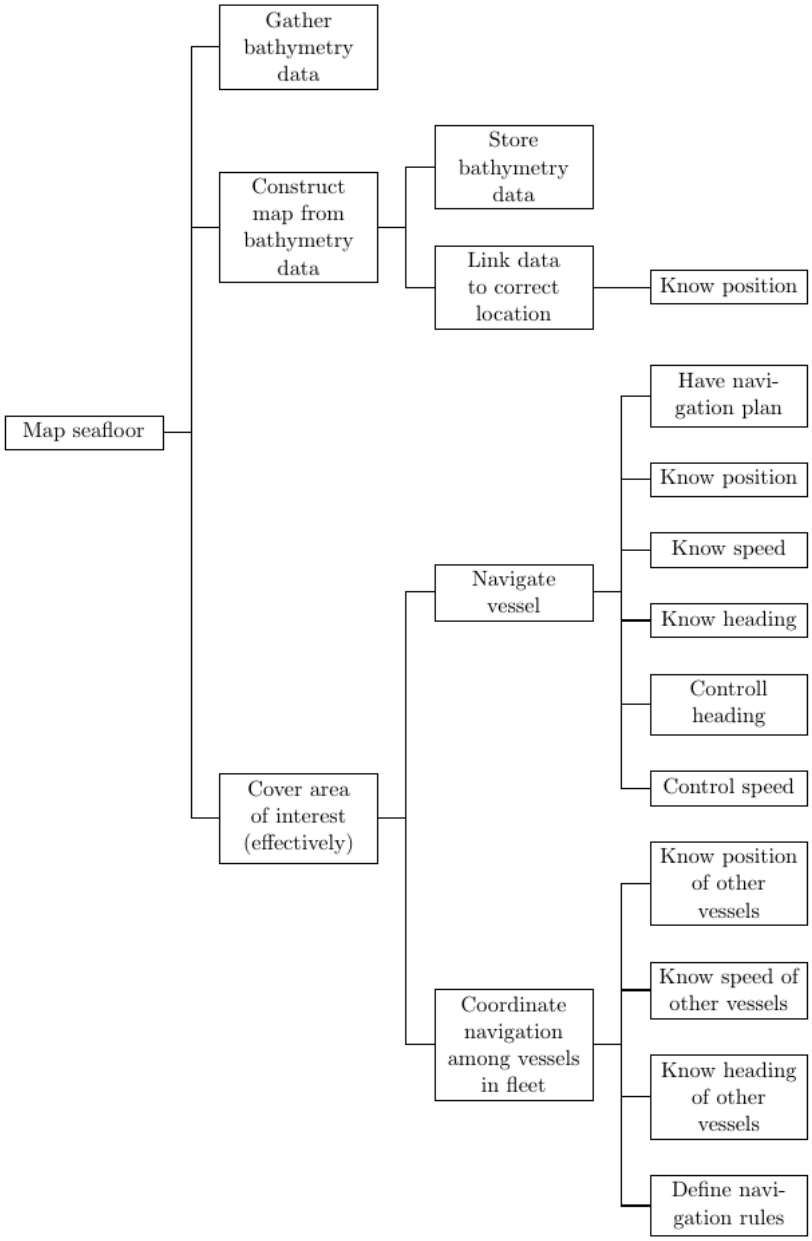

Figure 12: Functional break-down of the mission objective 'map the seafloor' for UC2.

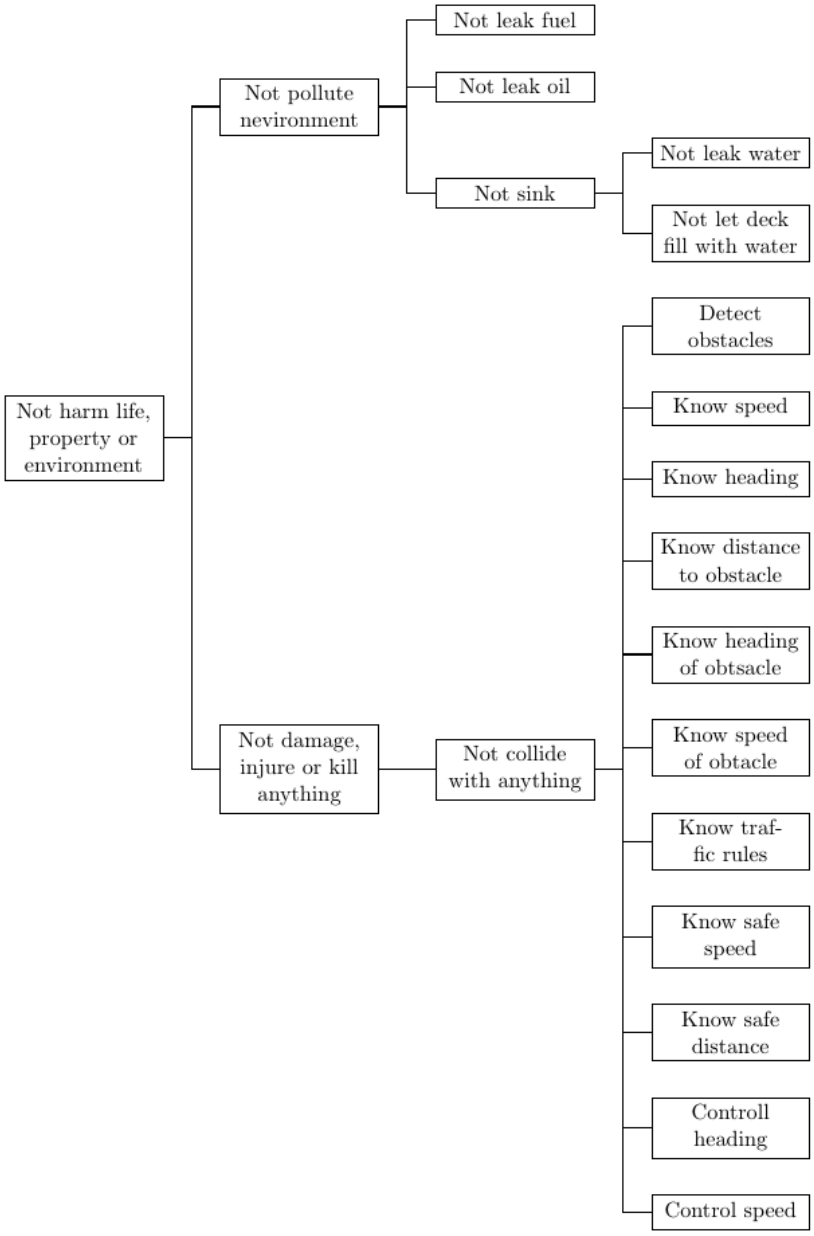

Figure 13: Functional break-down of the mission constraint 'not harm life, property or environment' for UC2.

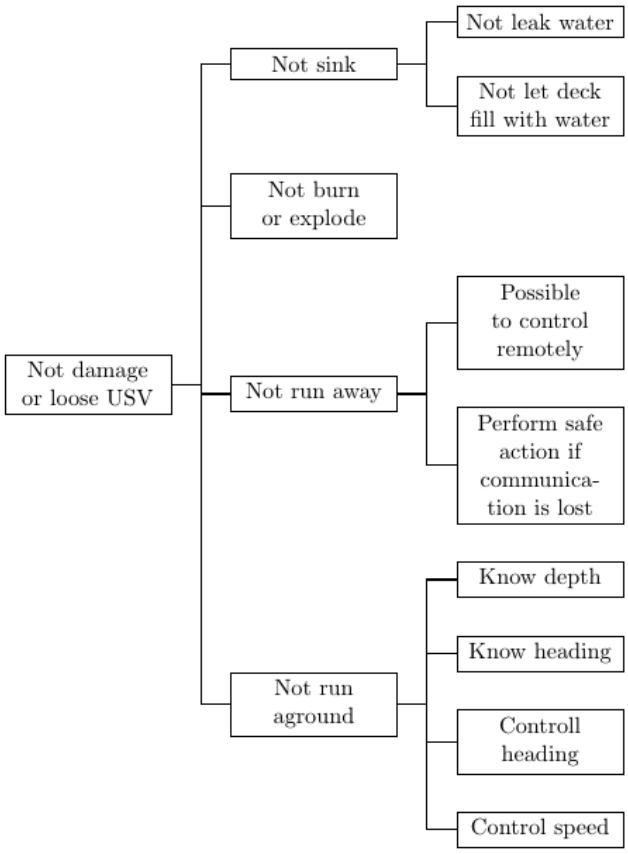

Figure 14: Functional break-down of the mission constraint 'not damage or loose USV' for UC2.

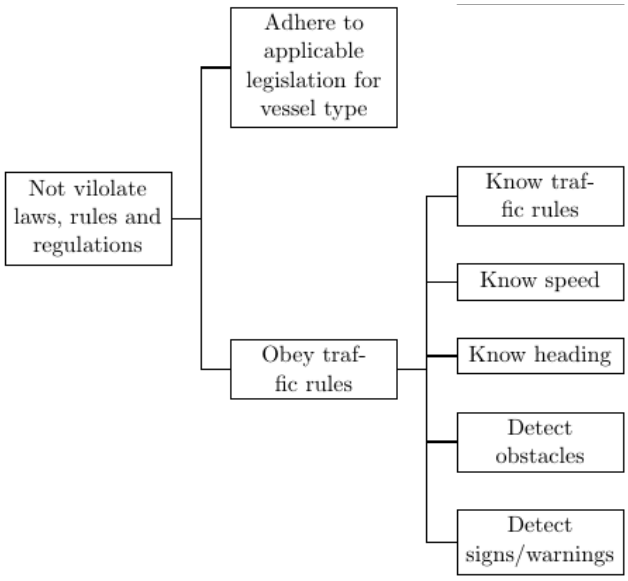

Figure 15: Functional break-down of the mission constraint 'not violate laws, rules and regulations' for UC2.

4.2 Hazard identification

Breaking the mission requirements into functions, allows one to identify hazards, i.e. scenarios that may lead to accidents or mission failure. This is in accordance with Leveson, which defines a hazard as "a system state

or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss).” (Leveson, 2011) (DNV GL, 2016a).

As an example, the hazard “not detecting obstacle” follows immediately from the identified function “detect obstacles”. In accordance with Leveson’s definition not detecting an obstacle does not necessarily imply an accident, but it could lead to an accident if the vessel is on collision course with the obstacle.

To avoid accidents, hazards should not occur. Hence, functions and associated hazards may be turned into functional requirements. For example, the hazard “not detecting obstacle” related to the function “detect obstacle” may be turned into a functional requirement of the form “obstacles should be detected in time to avoid collision”.

Below is an example of hazards and functional requirements identified for UC2.

Lev	T	ID	Description	Comment
0	A	01	***USV collides with master***	Accident
1	H	01.1	**Master has unsafe speed**	Scenario
2	C	01.1.1	*Master must have a safe speed to avoid USV collide with it*	Safety constraint or goal
3	FR	01.1.1.1	Safe speed of Master must be known in all situations	Control input needs
3	FR	01.1.1.2	Speed of Master must be controlled to avoid USV collision	Control needs
3	FR	01.1.1.3	Speed of Master must be known in all situations to avoid USV collision	Observability needs
1	H	01.2	**Master makes unsafe change of course or maneuvering**	Scenario
2	C	01.2.1	*Master must perform safe change of course or maneuvering*	Safety constraint or goal
3	FR	01.2.1.1	Safe change of course or maneuvering of Master must be known in all situations	Control input needs
3	FR	01.2.1.2	Course of Master must be controlled to avoid USV collision	Control needs
3	FR	01.2.1.3	Course of Master must be known at all times to avoid USV collision	Observability needs
1	H	01.3	**USV has unsafe speed and course risking collission with master**	Scenario
2	C	01.3.1	*USV must always have a safe speed and course to avoid collission with master*	Safety constraint or goal
3	FR	01.3.1.1	Safe speed and course of USV must always be known to avoid collission with master	Control input needs
3	FR	01.3.1.2	Speed and course of USV must always be controlled to avoid collission with master	Control needs
4	FR	01.3.1.2.1	Speed of USV must always be known to avoid collission with master	Observability needs
4	FR	01.3.1.2.2	Course of USV must always be known to avoid collission with Master	Observability needs
1	H	01.4	**USV has unsafe distance to Master**	Scenario
2	C	01.4.1	*USV must always have a safe distance to Master*	Safety constraint or goal
3	FR	01.4.1.1	Safe distance to master from USV must always be known to avoid collission	Control input needs
3	FR	01.4.1.2	Distance from USV to master must always be controlled to avoid collission	Control needs
4	FR	01.4.1.2.1	Distance from USV to master must always be known to avoid collission	Observability needs

Figure 16: STAMP example.

5 Resource- and control structure analysis

From the functional analysis in chapter 4, functional requirements were identified. Note that, on the functional level, no references to the actual systems or resources used to achieve these functions were needed. Functional dependencies were outlined, without reference to any physical entities or how they interact or exchange information.

The objective of the current chapter is to establish which resources and control structures that are needed to ensure that the identified functional requirements can be met. More specifically, the task is to evaluate

- if adequate resources and control mechanisms have been allocated to meet the functional requirements.
- whether the system is sufficiently safe with the available resources and implemented control mechanisms, or if additional resources or control measures are necessary to reduce the risk to a tolerable level.

In a design context, such analysis can be used as a formal process of allocating the necessary resources. For an existing system, it can be used to determine if scenarios can or should be either controlled, avoided, or accepted, based on cost benefit analysis and risk evaluation.

The proposed procedure outlined in this chapter consists of the following steps:

1. Identify the resources involved in delivering individual functions and draw up the system architecture, including the control structures and interactions among resources of the system and the environment
2. Identify flaws and determine how they can be handled, by analysing the system from different perspectives:
    a. Location perspective: Where in the system structure are the flaws?
    b. Time perspective: How does the flaw develop during the life cycle of the system?
    c. Controllability perspective: How predictable, detectable and controllable are the flaws?
    d. Risk perspective: Are the identified flaws acceptable, or should measures be taken to fix them?

At any point in the process of the safety assurance analysis, it may be necessary to iterate parts of, or the whole, process. As soon as discrepancies or ambiguity is identified, this need to be flagged, and the analysis need to be re-iterated to include the newly identified information. When the analysis is sufficiently granular, and all aspects of the safety has been considered, the remaining risk need to be communicated in clear and unambiguous terms, with recommendations which distinguish between how further risk can be reduced by either physical changes to the system (controlling the risk), or by increased knowledge (see e.g. (Eldevik, 2017)).

5.1 Drawing up the resource and control structure

Meeting the functional requirements determined from the analysis in Section 4 requires resources. This includes hardware, software, human and organizational elements that alone or in combination are responsible for delivering the required functions.

There are different ways of delivering the same functions. For example, there are many ways of propelling a vessel. For the proceeding analysis, a specific system architecture and resource allocation will be assumed,

and then analysed to identify possible flaws. In a design process, the system architecture may be modified base on identified flaws, in an iterative process. When analysing existing systems, the analysis may be used to identify needs for modifications or operational constraints.

As seen from the functional analysis of UC2 in Section 4, some functions can be related to several mission requirements. In such cases, it may be that the same resources can be used to deliver the function for several purposes. However, this introduces interdependencies that may increase risk. For example, in many industries it is required that the systems delivering safety-critical functions (i.e. related to safety constraints) are independent from production systems (i.e. used to deliver function related to achieving the mission objective). This illustrates that the system architecture used to deliver the functions has an impact on safety.

Following STAMP/STPA as described in section 2.2, a control structure diagram, indicating how the various resources in the system interact among themselves and with the environment, can be drawn up. A generic control structure is shown in Figure 17. On such a diagram, one may label the boxes, representing individual resources and the arrows between them, indicating how the resources interact or exchange information.

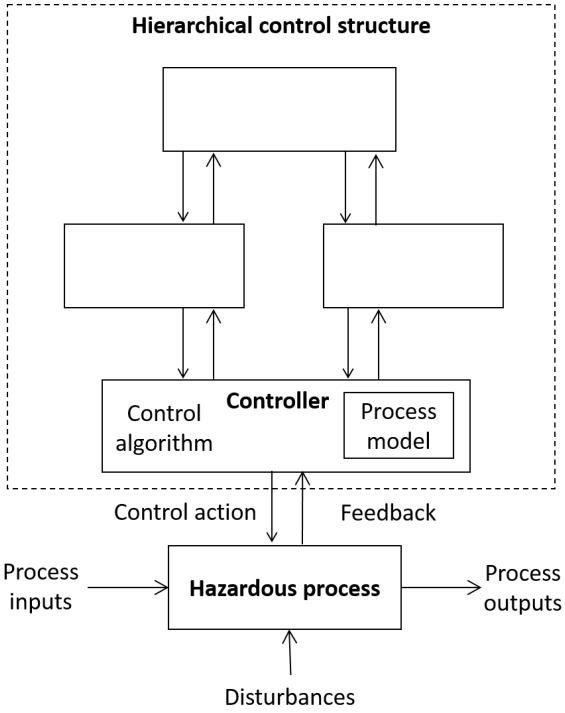

Figure 17: generic control structure, adapted from (Leveson, 2013)

In Figure 18: General control block component is the basic control structure we have used to design the system in UC2. The first design of the control structure for UC2 was done before we have started the STPA analysis in the use case and it is presented in Figure 19: First design of the control structure in UC 2. We have refined the control structure it after applying the STPA analysis and it is presented in Figure 20: Second design of the control structure in UC 2. We have introduced them to show the results of the first application of the STPA in the UC. However, the final control structure will change during the implementation of the systems and it will be presented in the final report presented describing UC2.

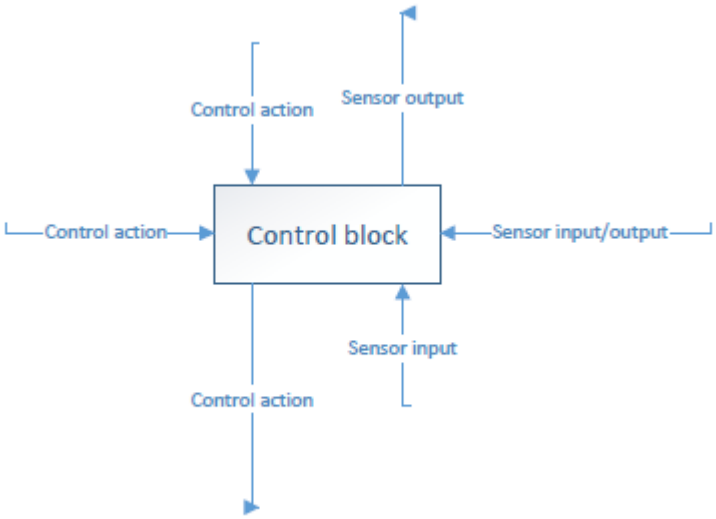

Figure 18: General control block component

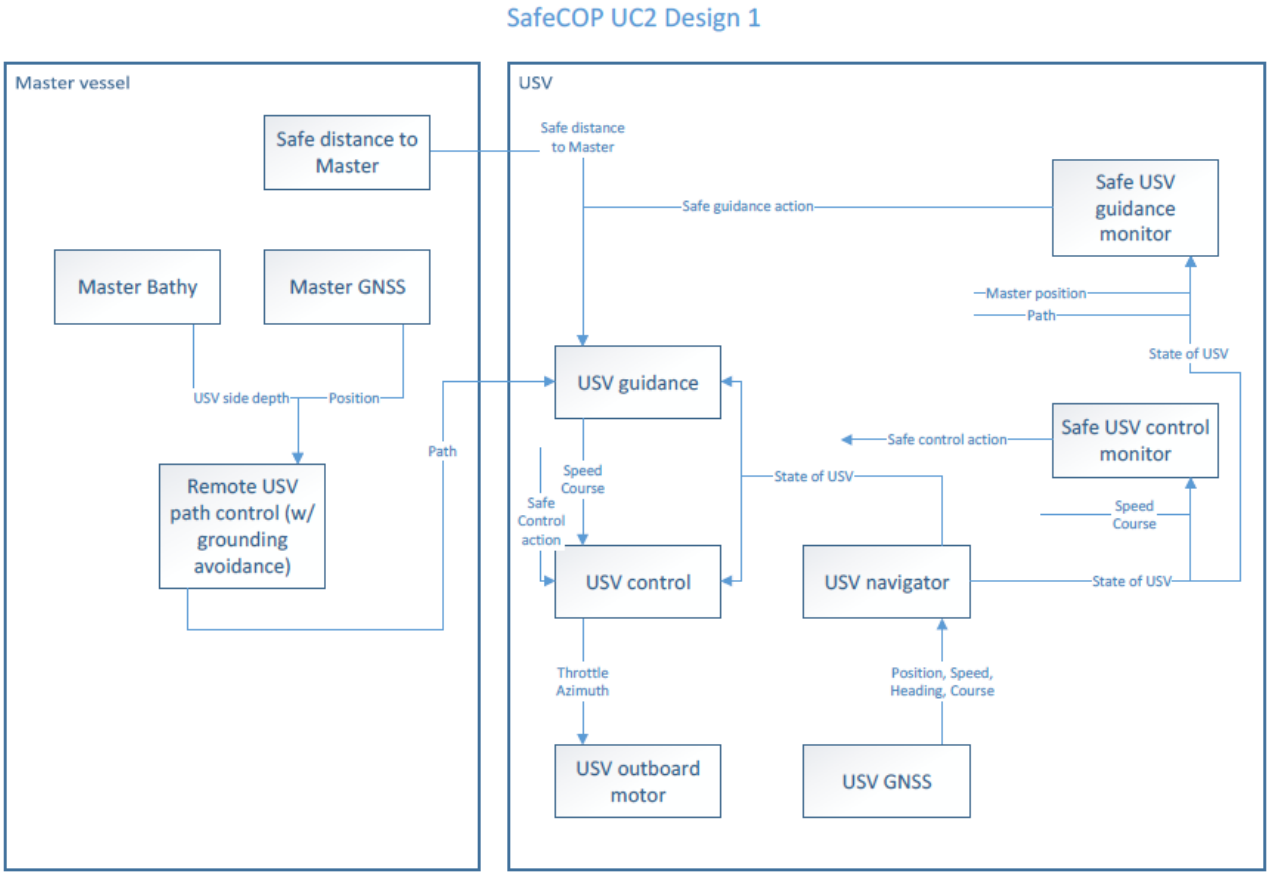

Figure 19: First design of the control structure in UC 2

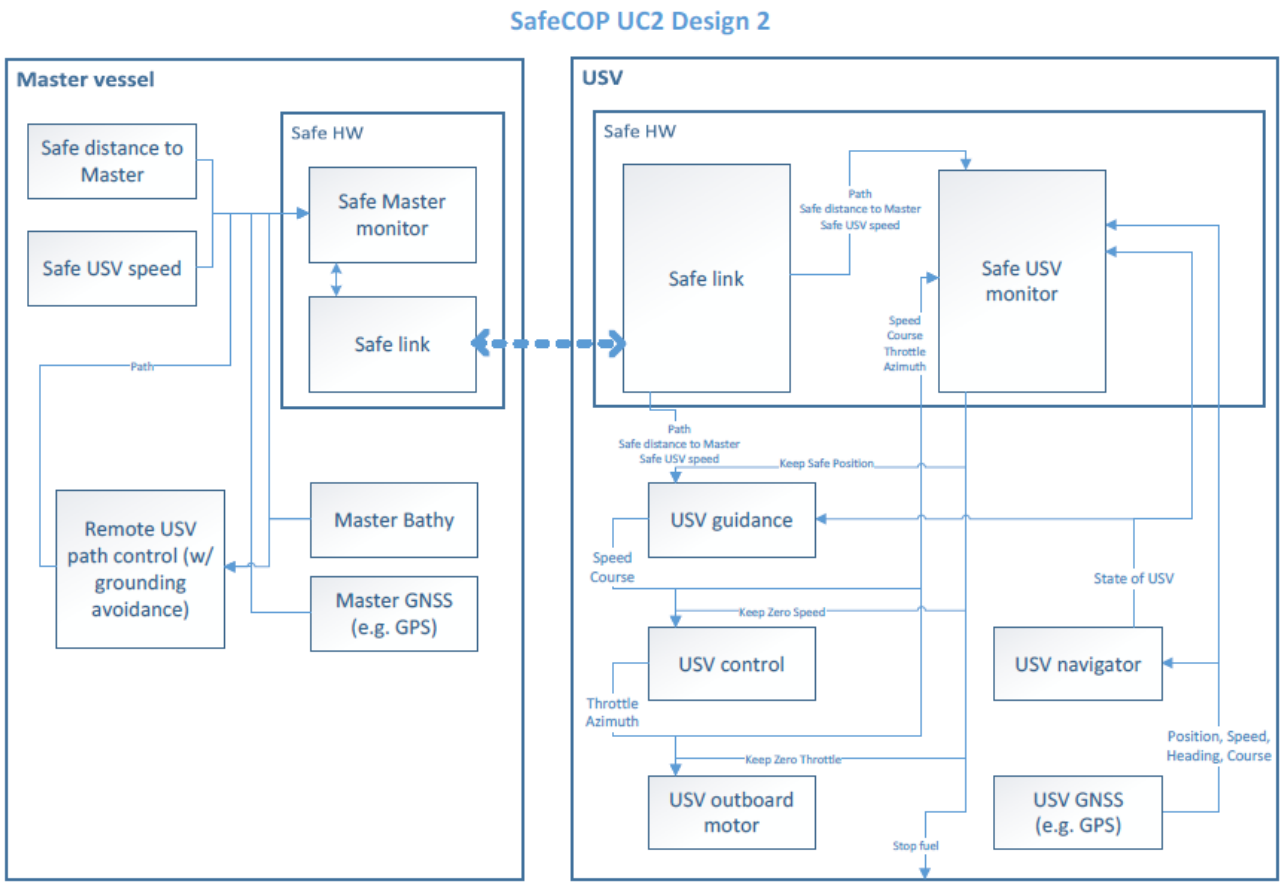

Figure 20: Second design of the control structure in UC 2

5.2 Identifying flaws and how they may be handled

5.2.1 Location perspective

Failures can be introduced by different entities or resources that make up the system. Failure to meet functional requirements may relate to human, organizational, hardware, software, or interfaces between- or co-operation among- the former. The origin of the failures will affect how they can be handled, and which resources are needed, to control them.

Note that a single component may serve several functions, and sometimes a collection of resources is needed to deliver a single function. Correspondingly, the failure of a function (constituting a hazard) may sometimes be due to failure in a single component, while other times it is due to interactions among components and not attributed to failure of any single component or collection of components.

In many cases a resource is designed to deliver a specific function, hence failure of the function might be directly linked to the state of the resource. However, for complex systems, a function is delivered by a set of resources working together towards a common goal, and failures on system level may not be attributable to a single part of the system (i.e. emerging failures).

One categorization of how resources may be linked to functions is listed below:

- A function may be absolutely dependent on a resource or combination of resources, so that requirements can only be met if the resource is present (e.g. a car cannot drive without wheels).
- It may be that a resource is required to deliver the function under a specific set of circumstances, but not always (e.g. lights on car may be needed to navigate a car from A to B in the dark).
- There may be redundancy in resources, so that a function can be fulfilled even if one or more resources are lost, or if the unavailability of the resource is temporary (e.g. a hybrid car can drive on petrol if the battery is charged out, if there is fuel on the tank).

A common practice is to evaluate failure and performance of individual components, e.g. as done in a FMECA, FTA, HAZOP etc. (see SafeCOP deliverable D2.1), to assess the system behaviour. This is a valid approach to see how individual or combination of individual resource's non-performance may lead to system failure. However, these methods are component focused, and may not be able to identify emergent failure modes.

To capture a larger range of flaws, including emergent failures, a method based on STAMP/STPA is recommended, where flaws are identified by means of the control structure diagram, as constructed in section 5.1. The structure can be used to assess the effect of inadequate control (e.g. missing-, conflicting-, too early-, or delayed control, etc.). Leveson summarize in (Leveson, 2013):

"(…) using the STAMP accident causation model, accidents occur when the safety control structure does not enforce the system safety constraints and hazardous states occur due to

1. *Unhandled environmental disturbances or conditions*
2. *Unhandled or uncontrolled component failures*
3. *Unsafe interactions among components*
4. *Inadequately coordinated control actions by multiple controllers"*

The analysis is done in the following steps:

1. Identify control actions (CA) from hierarchical control structure.
2. Identify unsafe control actions (UCA) related to each CA and based on the four categories above.
3. Identify scenarios (S) where UCAs could occur.
4. Formulate scenario requirements (SR) to mitigate/handle UCAs.

Examples of this procedure are shown in Figure 21 (to a relatively high design level, but clearly showing how STAMP/STPA contribute to requirements concerning safety monitoring and control).

Level	Type	ID	Description	Type
0	CTRL	03	***USV guidance***	Controller
1	CA	03.1	**USV speed**	Control Action
2	UCA	03.1.1	*Not provided when needed*	Unsafe Control Action
3	S	03.1.1.1	"USV speed" not set or provided when path provided	Scenario
4	SR	03.1.1.1.1	"USV speed" not provided if path is set should be deteced and handled (by Safe USV guidance monitor	Constraint
2	UCA	03.1.2	*Provided but not followed*	Unsafe Control Action
3	S	03.1.2.1	"USV speed" not received or used by "USV control"-controller	Scenario
4	SR	03.1.2.1.1	"USV speed" not reaching "USV controller" should be deteced and handled (by Safe USV control monitor ?)	Constraint
2	UCA	03.1.3	*Provided but not needed*	Unsafe Control Action
2	UCA	03.1.4	*Wrong magnitude/length provided*	Unsafe Control Action
3	S	03.1.4.1	"USV speed" zero (halt on path) when USV need to move away from master closing in	Scenario
4	SR	03.1.4.1.1	USV should not only try to stop, but actively move forward away from master ?	Constraint
3	S	03.1.4.2	"USV speed" zero or much lower than safe speed when path provided (platooning)	Scenario
4	SR	03.1.4.2.1	"USV speed" very low if path is set and USV is not doing master avoidance should be deteced and handled (by Safe USV guidance monitor ?)	Constraint
3	S	03.1.4.3	"USV speed" higher than safe speed	Scenario
4	SR	03.1.4.3.1	"USV speed" higher than safe speed should be limited or deteced and reduced (Safe USV guidance monitor)	Constraint
3	S	03.1.4.4	"USV speed" not zero when doing master avoidance	Scenario
4	SR	03.1.4.4.1	"USV speed" not zero when doing master avoidance must be detected and USV halted (Safe USV guidance monitor)	Constraint
2	UCA	03.1.5	*Wrong timing/sequence provided*	Unsafe Control Action

Figure 21: STPA example.

5.2.2 Time perspective

Flaws identified in the previous section may be introduced and develop into failures during different stages of the life-cycle of the system. In order to determine how the flaw can be handled, it is useful to ask the following questions:

- When is, the flaw introduced (e.g. in design, production/fabrication, transportation, installation, operation, maintenance, testing)?
- How does the degradation/failure progress (i.e. gradual or sudden)?
- Is the effect temporary, intermittent or permanent?

With regards to when a flaw is introduced, this determines when measures may be implemented to prevent it. Whether it is gradual or sudden determines if there is time to act once signs of a flaw or degradation appear so that can be fixed before turning critical. Such flaws may be more tolerable from a risk perspective (see section 5.2.4). Whether the fault is permanent or intermittent or only becomes problematic in certain circumstances also determines if operational restrictions can be used to avoid the effects of the flaw.

5.2.3 Controllability perspective

According to control theory, the ability to control a process depends on

- The ability to observe the process (observability need)
- The ability to influence the process (controllability needs)
- A defined objective (control input needs)
- An adequate model of the process, i.e. understanding of how to influence the system, based on observations, in order to reach the objective.

Control is can be exercised at different stages in the life cycle of a system:

- Control by design
  E.g. according to scenario requirements, standards, best practices, quality assurance, etc.
- Control in fabrication
  E.g. testing, quality assurance, procedural control, etc.
- Control in installation
  E.g. system testing, quality assurance, procedural control, etc.
- Control in operation
  E.g. through process control, monitoring, inspection and maintenance, repair, etc.

There may be more than one way to implement control. For example, a system to transport something from A to B can be designed to ensure a specific route from A to B (e.g. a train running on tracks), or it can be controlled in operation through a navigation system / driver.

5.2.4 Risk perspective

Identified flaws may not be critical if the consequence is small or the likelihood of the flaw developing into an accident or mission failure is low. STAMP/STPA is focused on identifying flaws and imposing additional control, but in reality, resources are limited, and implementation of extra control measures are judged according to their effects and costs compared to not doing anything. Rather than ensuring complete control, the question is whether the system is sufficiently controlled. ALARP, as low as reasonably practicable, is a principle applied in many industries, e.g. the oil and gas industry. ALARP implies that risk should be reduced until the sacrifices necessary to reduce it further become disproportionate to the risk to be avoided (HSE, 2017).

Depending on whether an identified scenario is controllable or not, will determine how it should be handled:

1. Not-controllable (by practical means):
   a. Consequence & Uncertainty evaluation (risk-based philosophy)
   b. Evaluate if risk is acceptable or not
2. Controllable (by design/static, by process/dynamic, social control/rules etc.) (Leveson, 2011)
   a. Evaluate cost/benefit of implementing control (based on Consequence & Uncertainty) and implement accordingly or go to list item 1.
   b. Evaluate degradation / reliability / failure of control (see list item 1 and reliability assessments)

6 Modelling support for CO-CPS

This section presents a model-based design supporting the safety assurance approach discussed in the previous sections. In particular, existing tools for safety analysis, architectural modelling and analysis, assurance case modelling and their extensions and integration are elaborated.

6.1 From safety analysis to system modelling

Modelling languages and supporting toolset are needed in order to help the safety architect to implement the safety assurance approach discussed in this document; in particular, with respect to hazard analysis and system architecture modelling phases, model-based support can be offered by XSTAMPP and CHESS tools.

In the following subsections, and introduction of XSTAMPP and CHESS is provided, together with guidelines about the usage of the current CHESS support covering the needs expressed in D2.2 and in the previous sections. Then areas of improvements related to XSTAMPP and CHESS usage in the context of SafeCOP for what regards hazard analysis and system modelling are discussed in section 6.1.5.

6.1.1 XSTAMPP

XSTAMPP (XSTAMPP, 2017) is an Eclipse based tool which supports the application of STAMP methods for safety analysis (e.g., STPA and CAST).

According to the needs of the STPA analysis process, XSTAMPP allows the modelling of the control structure (through the control structure diagram) of the system of interest. In particular, constructs to model controllers, actuators, controlled processes, sensors, control actions and connections between these entities are available. With regards to the controller, the modelling of its variables (e.g. internal variables, internal states and environmental variables) is also supported; variables for controllers can be used to obtain a better understanding of how unsafe control actions can occur and to derive refined safety requirements.

The aforementioned features allow a proper model-based support for the application of the STAMP methods and can support the resource- and control structure analysis step of the safety assurance approach presented in this document. It can also be used to document system goals, accidents, hazards and constraints, and how these are linked, as identified in the operational analysis and functional analysis.

6.1.2 CHESS

CHESS (Mazzini, 2016) is a model-based methodology for safety critical system design, analysis, with implementation support for what regards the software components through code generation. CHESS provides a modelling language, CHESSML (CHESSML, 2012), baselined on the UML/SysML and MARTE OMG standards; it also provides a set of views through which the system (System View), software (Component View), hardware related to software (Deployment View) and information related to supported analysis (Analysis View) can be modelled.

CHESS comes with a tool support which is based and extends the Papyrus Eclipse modelling editor. The tool allows to work with the CHESS modelling language and to apply the CHESS methodology in a dedicated environment.

One part of the CHESS methodology and modelling language regards the support for contract-based design, a hierarchical technique that provides formal support to the architectural decomposition of a system into

subsystems and subcomponents. The concept of contract, as sum of assumption and guarantee properties, can be modelled for a given system (or system component) to provide information about the behaviour of the system which is guaranteed to hold (contract guarantee) by assuming that the environment in which the system is instantiated behaves in a certain manner (contract assumption).

Requirements that need to be fulfilled by a given system/component have to be formalized by contracts assumptions and/or guarantees by using formal languages.

CHESS supports the modelling of the contracts, where a contract represents the formalization of a safety requirement. CHESS supports contract specification with LTL OCRA language[2] and the modelling of contracts refinement, i.e. the decomposition of the contracts along the decomposition of the system architecture. Moreover, CHESS supports different verification facilities related to contract-based design through a seamless integration with the OCRA formal verification tool[3] (OCRA, 2015), e.g. to prove that a contract's refinement is correct, or to verify that a given contract associated to a component is satisfied by the behavioural model (provided by a state machine) of the component itself. Some of these extensions are currently ongoing as part of the AMASS project.

6.1.3 Supporting the CESAM perspectives

The modelling of three operational perspectives identified by the CESAM methodology is supported by the current version of CHESS, in particular in its System View, by basically applying SysML support and CHESS contract-based extensions.

In general, during the whole modelling process, from the early phases - at a higher level of abstraction - down to the later phases - at a lower level of abstraction -, the system is described in terms of architectural components with their defined interfaces (e.g. input and output data ports) and related properties. During each phase the components are considered as black boxes until they are refined into new lower level components in the next phase.

The CESAM operational perspective is supported by use case diagrams, to clarify what the system should do, and block definition diagrams; in particular, the latter can be used to specify the system component interfaces; contracts associated to the system allow to formalize the safety requirements that the system has to fulfil and the expected interactions with the environment.

For reusability, the system can be designed in a way that the same functionalities can be provided with different extra-functional/quality properties; for instance, the same functionalities regarding a control action could be provided with different timing constraints according to different assumptions on the environment. So, different contracts (see the concept of *weak* contracts explained in section 6.2.2) have to be provided for the system, one for each possible different extra functional property.

For what regards the functional analysis, the modelling of the CESAM functional perspective, and also the logical relationship between functions of a system addressed by the FAST technique, are supported by SysML Block Definition and Internal Block Diagrams through which the functionalities (UML Activities), their dependencies and possibly their hierarchical decomposition and dependencies, can be represented. Figure 22 and Figure 23 below represent an example of the usage of the aforementioned diagrams modelling a functionality

[2] https://es.fbk.eu/tools/ocra/download/OCRA_Language_User_Guide.pdf

[3] https://ocra.fbk.eu/

related to the optimization of the traffic flow through the adoption of a Green Light Optimal Speed (GLOSA) solution (this is one of the functionality addressed by UC5 SafeCOP use case; it is worth noting that the models represented here have been just inspired by the UC5 use case. UC5 actual results will be reported in the corresponding D5.7 and D5.8 deliverables). An activity can be represented in a block definition diagram with its decomposition in fine-grained activities. Then the data flow between the activities can be detailed in the internal block diagram, by using typed in-out ports, to represent activities dependencies.

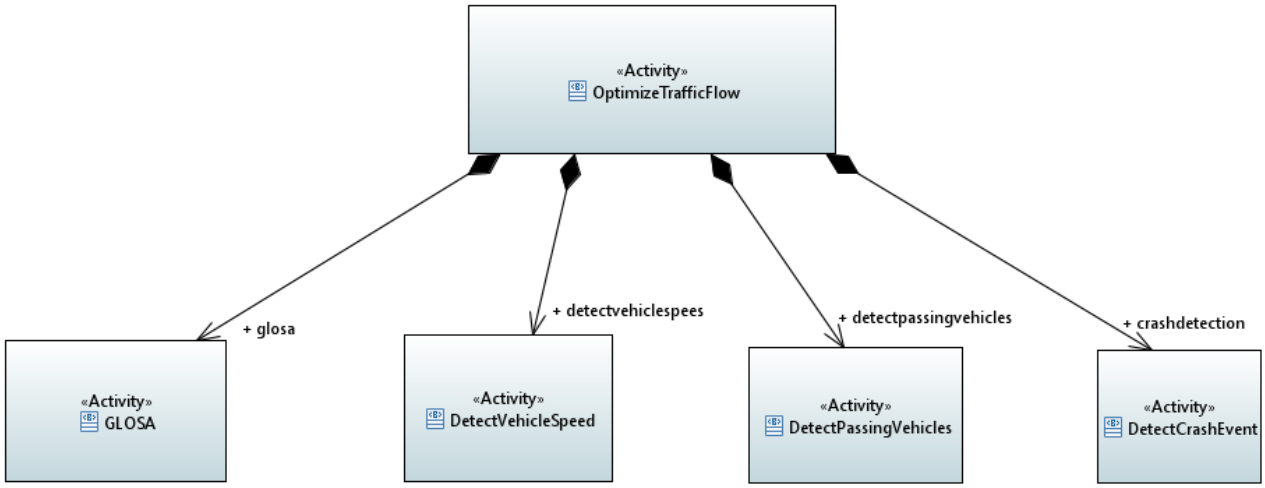

Figure 22: Functional perspective, activity decomposition

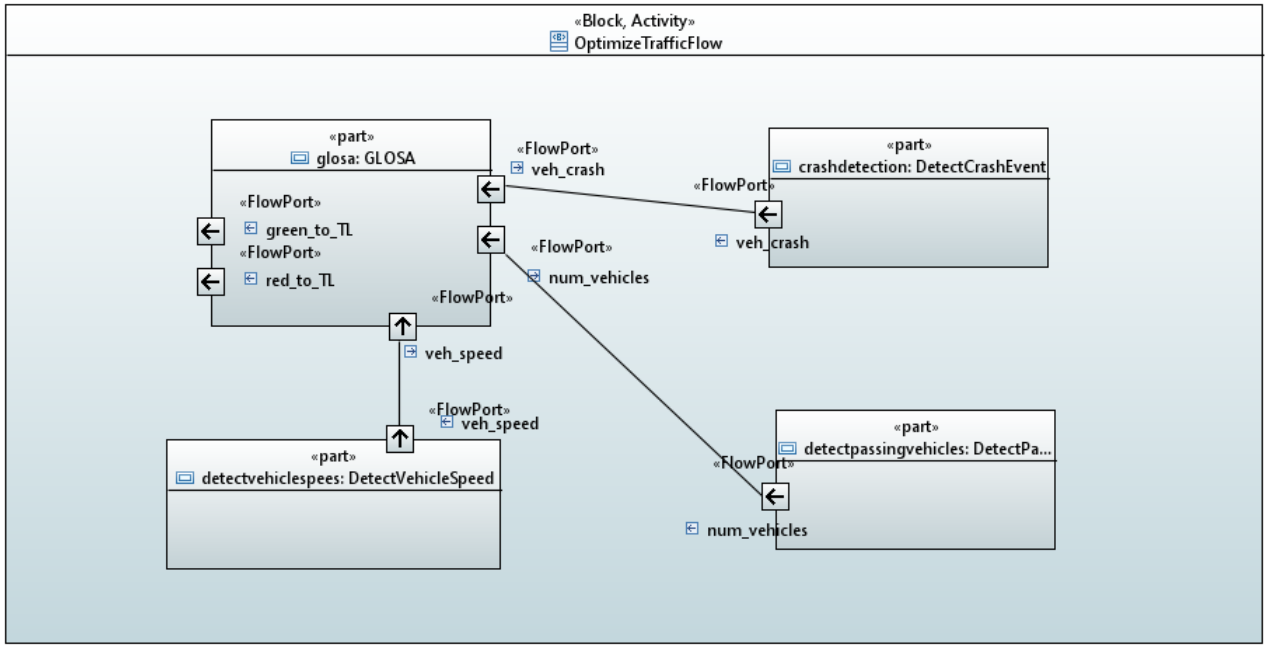

Figure 23: Functional perspective, data flow

The modelling of the constructional perspective at system level is also supported by the usage of the Block Definition and Internal Block Diagrams; the SysML Block construct is used to represent a logical or physical architectural entity. Figure *24* below shows a set of architectural entities (in blue) building the traffic management system in charge to realize the functionalities (in orange) identified in the functional perspective; the functionalities to system architectural mapping is modelled by using the Allocated dependencies.

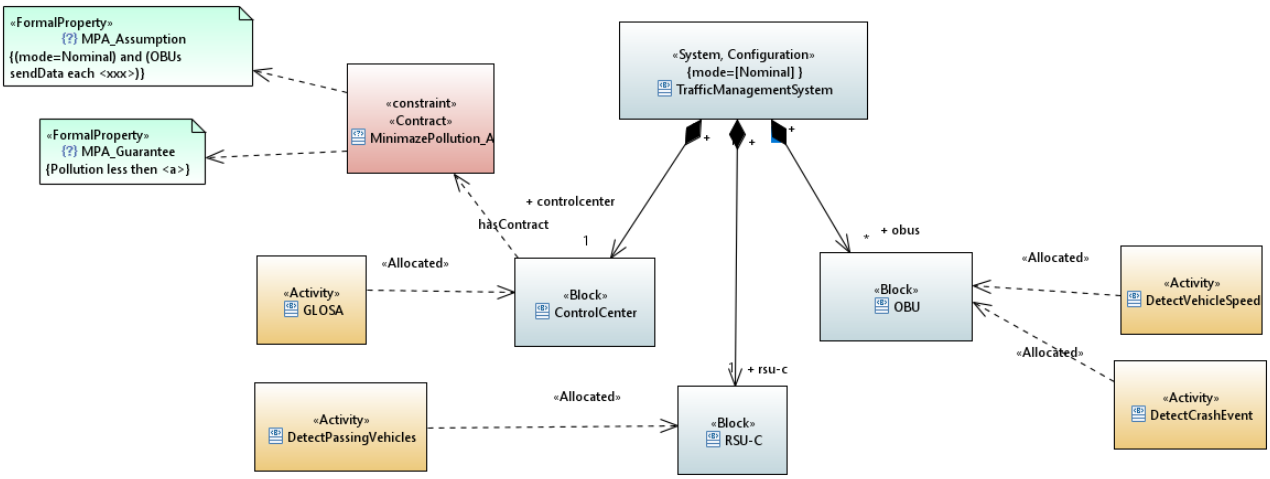

Figure 24: constructional perspective

The contract-based approach promoted by CHESS is applicable during all the different modelling phases. Each system (functional or architectural) component can have contracts associated representing the formalization of the requirements coming from the safety analysis. When a component is decomposed, its contracts are refined with contracts of the sub-components: if the refinement steps are proven correct (by using the analysis enabled by the CHESS to OCRA tool integration), any implementation of the leaf components that satisfies the component contracts can be used to implement the system. The verification is therefore compositional and early verification of partial architecture is possible.

Figure *24* shows an example of contract (in pink) associated to the ControlCenter elaborating some properties about the level of pollution to be guaranteed (MPA_Guarantee FormalProperty); to accomplish the goal, the contract requires (MPA_Assumption FormalProperty) that the on-board units (OBUs) are able to send data with a given frequency (note that for the sake of simplicity, the contract assumption and guarantee properties are expressed here in a semi-formal language).

It is worth noting that for the functional and constructional perspectives, the model describes a *vertical* refinement with the decomposition of components into subcomponents and the refinement of contracts into a collection of contracts over subcomponents. Such vertical refinement is subject to formal verification (with OCRA) and is key-point in the overall verification process.

When stepping from one conceptual level to the next (e.g. from the functional perspective to the constructional perspective), on the other hand, a new architecture model is created and links (SysML <<allocated>> dependency, see Figure *24*) are created to maintain the connection between corresponding entities in the two different architectures with different decomposition structure and contracts, mainly for the sake of traceability. This process is referred to as *horizontal* refinement, it is not subject to formal verification.

6.1.4 Modelling CO-CPS and collaborating functions

As discussed in SafeCOP D2.2, the modelling and analysis of scenarios plays a key role in the context of model-based design for CO-CPS.

This section provides information about how the different possible configurations in which the CO-CPS can run can be modelled in CHESS.

The different states in which the system can operate can be modelled by using the OMG MARTE support for modes modelling. Figure 25 shows the usage of stereotyped ModeBehavior state machine which defines two possible operational modes (Mode states) in which the Traffic Management System previously introduced can run, nominal and degraded, the latter for instance representing a system state where the road-side camera unit (RSU_C) is no more available.

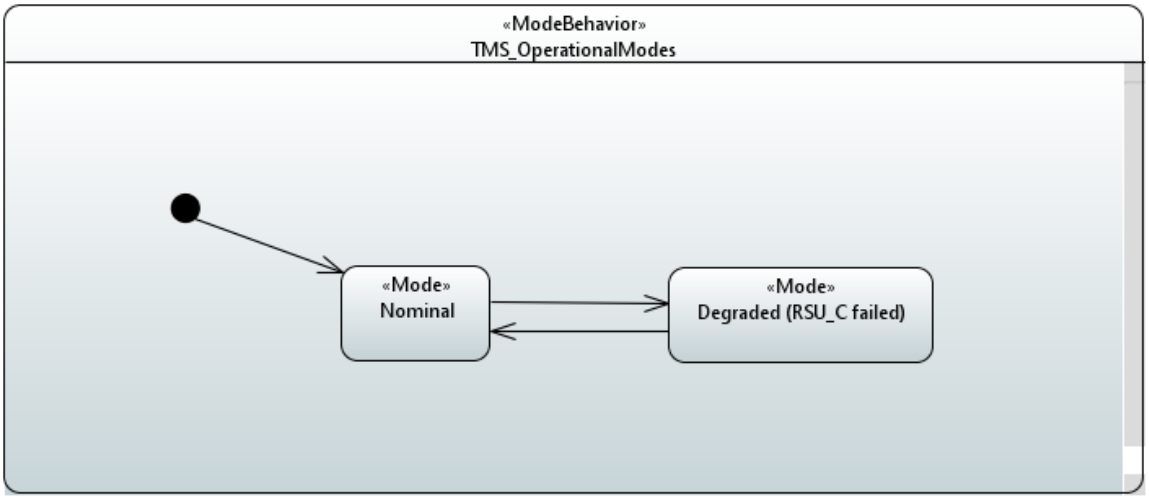

Figure 25: Modelling Operational Modes

MARTE support for modelling configuration is used here to depict the different structures of the systems and the roles that are required to perform the collaborating function in the particular operational mode: this support will be evaluated in the context of the case studies application.

Figure *24* and Figure 26 show how configurations could be represented: the configuration appears as a dedicated entity stereotyped with MARTE Configuration stereotype, for which the set of active system elements are represented. For each configuration, the allocation of activities to the internal parts (i.e. the configuration roles) can be specified, so different configurations can have different allocations of the activities (Figure 26 shows that with respect to the nominal case depicted in Figure *24*, in the degraded case a different GLOSA activity *GLOSA_v2* is allocated to the ControlCenter).

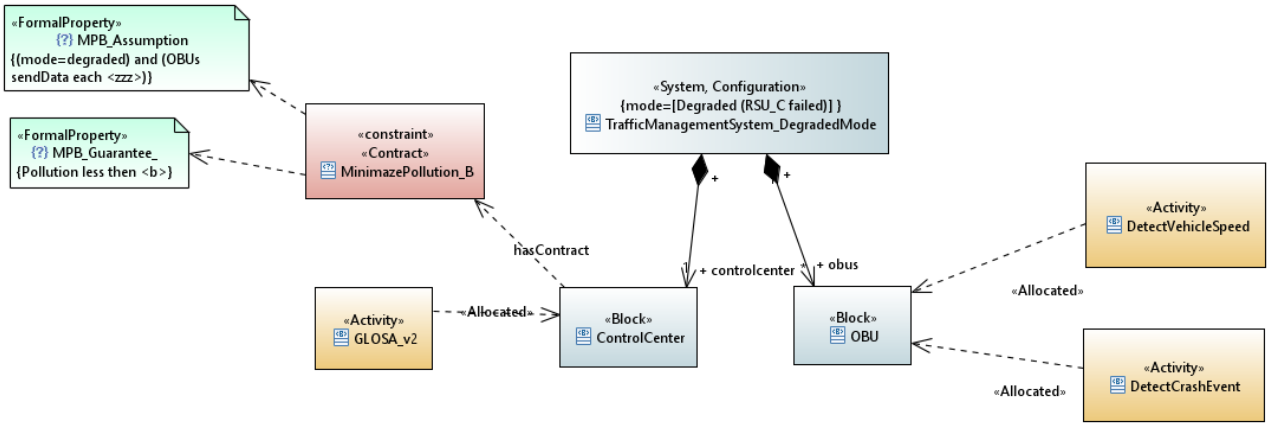

Figure 26: Modelling configuration

Internal block diagrams are provided for each system configuration to detail the information that is exchanged between the involved roles; it is worth noting that each configuration can be different with respect to the involved roles, their interfaces and the connections between them. For instance, in the provided example, the ControlCenter in the nominal mode will exchange information with the RSU_C and the OBUs, while in the degraded mode it will communicate with the OBUs only.

Different safety requirements can be derived from the different operational modes; they can be expressed through contracts and associated to the roles. Figure 26 shows that a different contract is enabled for the ControlCenter in the degraded mode; in this mode, we can imagine that, with respect to the nominal case, a higher (but still safe) threshold for the pollution can be guaranteed.

The ModeBehaviour state machine depicted in Figure 25 is used to declare the modes in which the system can execute. Then, ModeBehaviour state machines have to be provided for the roles of the configuration, as refinement of the ModeBehaviour represented in Figure 25, to detail how modes transitions are actually enabled. For instance, in the above example, we can suppose that the ControlCenter has a contract assuming that, in the nominal mode, data is sent by the RSU_C with a given frequency; the transition to the degraded mode could be enabled by an event representing the violation of this contract at run-time (to be checked by the run-time manager).

6.1.5 Areas of investigation

6.1.5.1 Modelling CO-CPS configurations

As illustrated in the previous chapter, MARTE modelling language already provides some support for the representation of system configurations by using the ModeBehaviour, Mode and Configuration stereotypes. However, this support comes with some limitations, indeed for instance it requires to model one system block for each configuration, and so to model the decomposition of each system block to show the entities which are active in the corresponding configuration; the ideal solution would be to have a single composite block representing the system and model the different decomposition/configurations for it, to avoid possible duplication of information in the model. Indeed, as example, suppose to have a collaboration of sub-blocks which is in common with two system configurations: with the current MARTE support the collaboration have to model two times in the internal block diagrams of the two system blocks representing the two configurations.

These identified limitations will be further investigated in the context of the case studies modelling examples. Then, extensions addressing these limitations will be evaluated in the context of WP4.

6.1.5.2 XSTAMPP and CHESS

To support the application of the safety assurance approach presented in this document, we propose an integration between the XSTAMPP and CHESS methodologies and corresponding tool; while STAMP can enrich the CHESS methodology and tool with dedicated support for safety analysis, CHESS offers a richer set of constructs to properly model and refine the system in charge to satisfy the safety requirements derived through STAMP and XSTAMPP. In particular the CHESS contract-based approach supported and envisaged by CHESS can be of help, to formalize the safety requirements, to enable analysis techniques for them and then to support the definition and generation of the assurance case, for what regards its static part but also its dynamic one (see section 6.2).

The information that can be exchanged between XSTAMPP and CHESS is the following:

- Safety requirements: safety requirements are derived through safety analysis and represented in XSTAMPP. Once defined, safety requirements have to be formalized in CHESS and represented in a form of components contracts.
  One interesting feature available through XSTAMPP regards the possibility to automatically derive the LTL formal representation of the safety requirements which have been derived via the application of the STAMP methodology. The possibility to use this LTL representation to automatically fill the contracts modelled in CHESS is under investigation.
- Control structure: traceability information between the control structure addressed in XSTAMPP and the CHESS system model could be provided.

Mechanisms and features supporting the integration between XSTAMPP and CHESS, in particular related to the aforementioned information, are currently under development in WP4. For instance, one promising technology enabling the management of traceability models comes from the Eclipse Capra project, the latter already used in the context of the AMASS project to manage traceability links between architectural and assurance case related entities.

6.2 Contract-driven modular assurance

Contract-based design in component-based software engineering relies on describing behaviours of components in terms of contracts. A contract is a pair of assertions namely assumptions and guarantees, such that the component offers the guarantees about its own behaviour, given that the environment in which it is deployed fulfils the assumptions (Benveniste, 2012). Such contracts are envisaged to support compositional verification of a system as well as reuse and independent development of components. The nature of compositional verification is that confidence in the contracts of a composite component is supported by the confidence in the contracts of its subcomponents. To support design of compositional systems modelled in CHESS, contracts should be specified on each level of composition. Since not every sub-component contract supports every contract of the composite component, the notion of decomposition is used to indicate the dependencies between the contracts on the different component abstraction levels. Contract refinement represents the backbone of checking the component decomposition. Informally, a set of contracts of the sub-components refines a contract of the composite component if: (i) the assumptions of all sub-component contracts are met by the other sub-components and the environment defined by the assumptions of the composite component contract; and (ii) the sub-component contracts deployed in the environment defined by the composite contract assumptions imply the composite contract guarantees (Cimatti, 2014).

To perform contract-driven modular assurance, tool support for the following activities is needed: system modelling, contract checking engine and safety case modelling. We turn to AMASS platform that includes

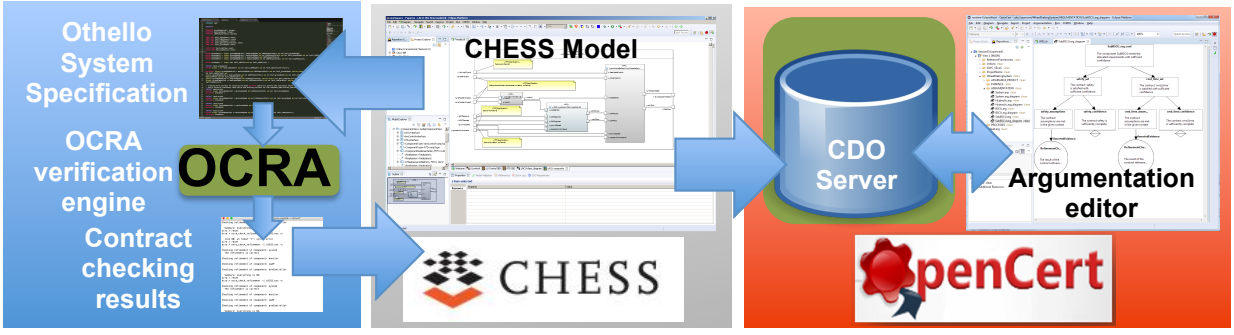

Figure 27: The overview of the tool information flow

CHESS[4] as the tool for system modelling, OCRA[5] as a contract checking engine and OpenCert[6] as an assurance case modelling tool. The overview of the interaction of the three tools is shown in Figure 27. In the subsequent sections, we first introduce the background information related to the requirement assurance via component contracts and contract driven support for reuse and adaptation developed through several projects such as SafeCer, SYNOPSIS, CHESS, and AMASS. Then, we present the SafeCOP contribution to the contract-driven assurance approach, in particular to provide support for dynamic/runtime assurance.

6.2.1 Requirements assurance via component contracts

Since safety assurance is driven by safety requirements that are allocated to different components of the system, the corresponding contracts of those components are envisaged to formalize the allocated require-

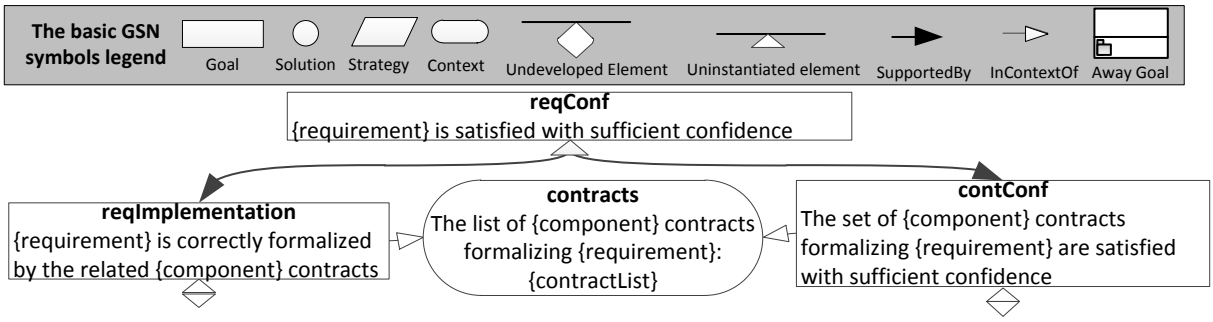

Figure 28: Contract-driven requirement satisfaction assurance argument pattern

ments. In that scenario, each requirement is formalized or realized by a set of contracts. Establishing the validity of such requirements then boils down to checking the consistency and refinement of the contracts. But something more is needed to assure that the requirement is met by the system. To assure that a system satisfies a given safety requirement based on the related contract, we need to provide evidence that the contract correctly realizes the requirement (often said that its guarantees formalize the requirement) and evidence that the contract is satisfied with sufficient confidence in the given system context. We refer to this argument strategy as the contract-based requirements satisfaction pattern (Figure 28).

While compositional verification of a system using, contracts establishes validity of a particular requirement on the system model in terms of contracts, confidence that the system implementation actually behaves according to the contracts should also be assured. Hence, to drive the system assurance using contracts we have associated assurance assets with each contract. Those assets can be different kinds of evidence that increase confidence that the component (i.e., the implementation of the contracts) behaves according to the contract, i.e., that the component deployed in any environment that satisfies the contract assumptions exhibits the behaviours specified in the corresponding contract guarantees. To argue that a contract is satisfied with sufficient confidence we need to assure that the component actually behaves according to the contract, and that the environment in which the component is deployed satisfies the contract assumptions (Sljivo, 2015). But when we deal with hierarchical systems where contracts are defined on each hierarchical level with well-defined decomposition conditions, then to argue that the composite component behaves according to the contract, we should explicitly argue over the component decomposition (Figure 29).

[4] CHESS Eclipse Polarsys project, https://www.polarsys.org/chess/start.html

[5] OCRA Tool, https://ocra.fbk.eu/

[6] OpenCert Eclipse Polarsys project, https://www.polarsys.org/projects/polarsys.opencert

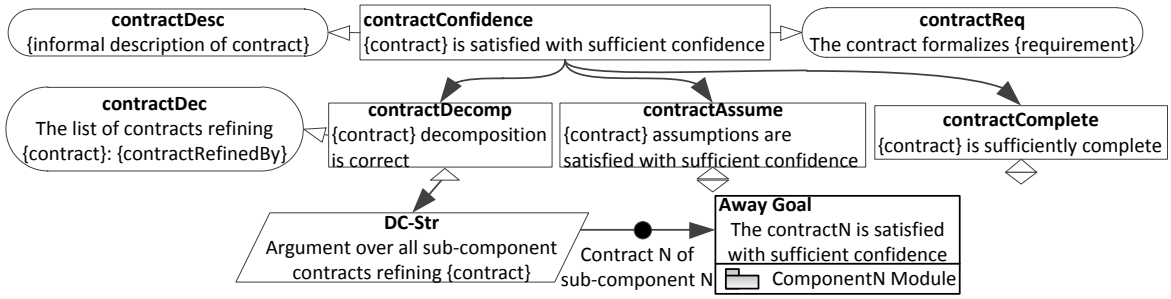

Figure 29: Contract satisfaction assurance argument pattern

The contract-driven assurance case structure assumes that an assurance module is created for each system component where the contract-satisfaction patterns are instantiated to argue over each contract of the component. A separate module that depends on those component modules is created for arguing over satisfaction of requirements. The contract-based requirements satisfaction pattern is used to build that argument module and connect that module to the corresponding component modules. The structure of the assurance case is shown in Figure 30.

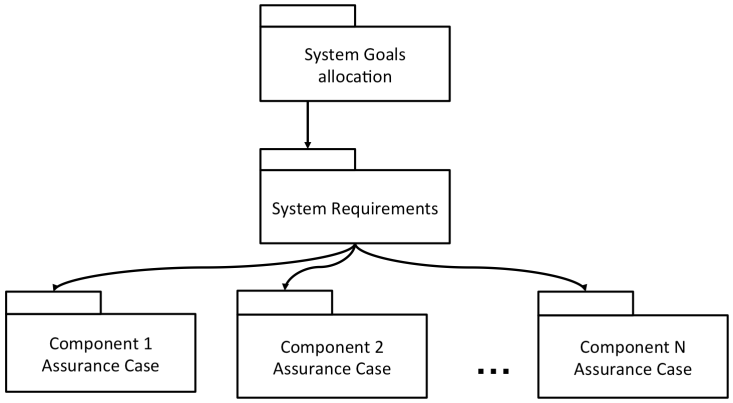

Figure 30: Assurance case structure

6.2.2 Contract-driven support for assurance adaptation and reuse

Reuse is intrinsic to contract-based design. It enables checking if a component can be reused in a particular system, i.e., whether the system meets its demands and whether the component meets the demands of the system. The support for reuse in contract-based design has been mainly focused on components (i.e., implementations of contracts) and not reusable components as implementations of a set of contracts for different environments that may or may not be satisfiable together. While traditionally we would define the environment in which the system will perform and develop the system to be acceptably safe in the context of that environment, with open and adaptive systems it is becoming challenging to define an environment to develop the system in as the environments often change in such open and adaptive systems.

The variability on the contract level in terms of strong and weak contracts allows for handling varying component environments of reusable components (Sljivo, 2015). On the one hand, the strong contracts are those whose assumptions should be met by every context in which the component is reused, hence its guarantees are always offered by the component. On the other hand, the weak contract assumptions do not need to be

satisfied by every context in which the component is reused, but when they are met, only then the component offers the corresponding weak guarantees.

Reusable components, just as components in open and adaptive systems, need to be ready for certification and assurance in different environments. Checking the validity of their contract specification should tell us whether the component can be reused and which parts of it can be reused. Similarly, for open and adaptive systems, checking the contracts should tell us whether the needed contracts have been fulfilled for the component to continue guaranteeing its behaviours needed for maintaining the system safety. Depending on which weak contract is met in the given environment we can then identify which assurance information (and certificates) are valid in the given environment. For example, the assurance in one environment may include one set of contract-satisfaction arguments, and a different set in another environment, depending on the validity of the related contracts. To automate the contract-driven assurance and reuse by providing tool support, a generic Safety Element out-of-Context MetaModel (SEooCMM) that defines relationships between reusable components such as SEooC, contracts, requirements and assurance assets has been proposed (Sljivo,2016). CHESS has been extended to support this metamodel by allowing modelling of both strong and weak contracts, and their association with the related requirements and assurance assets. Furthermore, CHESS and Opencert have been extended with the automatic instantiation of the contract satisfaction argument pattern from the system model compliant with SEooCMM.

6.2.3 Contract-driven support for dynamic assurance

In this section, we add the SafeCOP flavour to the contract-driven assurance approach developed in earlier projects and add support for dynamic/runtime assurance. We first introduce the basic notions, then we discuss the needed meta-model extensions, tool support status, and finally, we present an application example.

6.2.3.1 The basic support for dynamic/continuous assurance: "Runtime" contracts, runtime evidence and dynamic asserted relationships

In assurance of traditional systems, requirements are assured in the context of a given environment. But in 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, 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. When assuring the contract satisfaction this means that not the whole contract-related argument needs to be re-evaluated, but only "contractAssume" and "contractComplete" claims of the contract-satisfaction argument pattern. To model that in the safety case we introduce the notion of *runtime/dynamic evidence*. Such evidence is dynamic, it changes during runtime and at some point, it can be supporting the connected claims, i.e., increasing their confidence, while at sometimes it can be decreasing their confidence and acting as counter example. Hence, we also introduce the notion of *"dynamic" asserted evidence relationship*, which connects such runtime evidence with claims in the system. The dynamic asserted evidence relationship adapts with the changes to the runtime evidence. It should be specified with a condition on the dynamic properties of the runtime evidence it is related to. When that condition is met, the dynamic asserted evidence relationship acts as supported by relationship, where it shows that the claim is supported by the underlying runtime evidence. When the condition is false, the dynamic asserted evidence relationship acts as a counter example

relationship. We refer to the claim that is connected with a dynamic asserted evidence relationship as a "dynamic" claim.

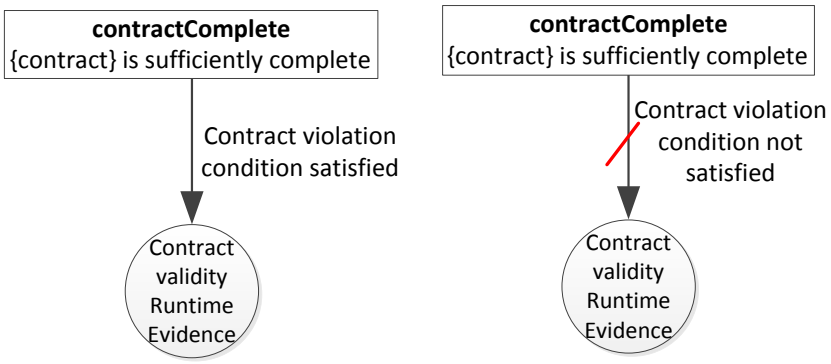

Figure 31: The usage of the dynamic asserted evidence relationship in the contract-satisfaction pattern

In the context of the contract-satisfaction pattern we deem that the "contractComplete" claim should be dynamic, as it aims at showing that the component offers the guarantees whenever the assumptions are met. Furthermore, the "contractAssume" claim would be also dynamic, as the contract assumptions might get invalidated during runtime, which may decrease the confidence in that claim, hence in the contract satisfaction, and in turn, that in the confidence in the satisfaction of the related safety requirement. Figure 31 shows an example of the usage of the dynamic asserted evidence relationship for the "contractComplete" claim in the contract-satisfaction argument pattern. The dynamic evidence in this case is "Contract validity runtime evidence", which may report the contract violation rate.

6.2.3.2 Extending the system and assurance meta-models for dynamic assurance

To support the tagging of contracts with the runtime flag, we adapt the system modelling meta-model to accommodate for the change (Figure 32). We add the propertyStatus attribute to both contracts and the formalExpressions that can be either standalone or a part of contract in form of assumptions and guaran-

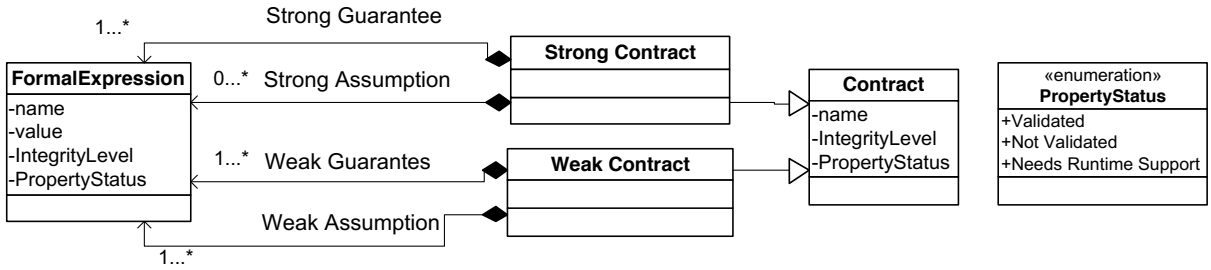

Figure 32: Contract-related part of the system meta-model

tees. The "*needs runtime support*" status indicates that the validity of the corresponding property should be evaluated during runtime. In the case of contracts, it means that the implication "assumption implies guarantee" should be evaluated, while in the case of a standalone formalExpression, and then the expression itself should be evaluated during runtime.

In the context of AMASS, we have associated the contracts and formalExpressions with assurance artefacts with support for artefact versioning, as facilitated through SACM Artefact Package Diagram (Figure 33). The

artefact modelling support in SACM is sufficient for modelling of the runtime/dynamic evidence for continuous assurance. The version and date attributes can be used to distinguish between different evidence instances and save history. The property associated to the artefact can be used to set the values of the dynamic properties that characterize the runtime evidence. For example, in the case of runtime evidence for contracts, a dynamic property can be the contract violation rate, which tracks the rate of contract violations, i.e., when the assumptions are fulfilled, and guarantees are not provided. The violation rates of the different runtime properties can be useful when evaluating confidence in the safety requirements supported by those properties.

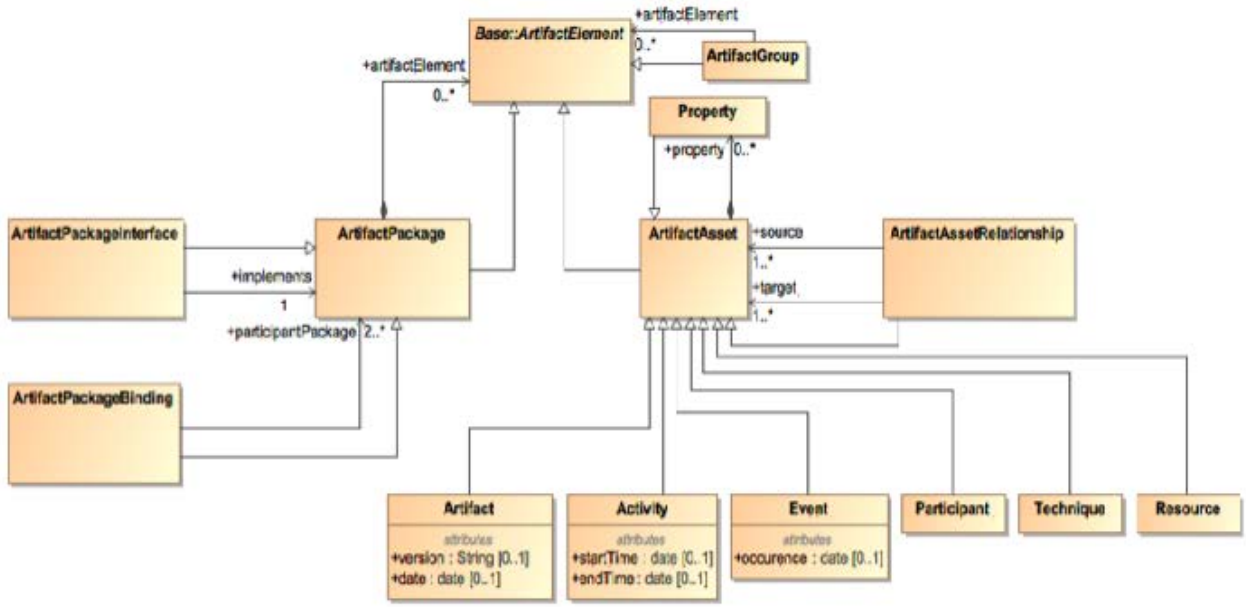

Figure 33: The SACM Artifact Package Diagram

To support the confidence evaluation on the assurance case modelling side, we adapt the existing SACM Argumentation Package Diagram to add support for the dynamic assertion relationships. As shown in Figure 34, the assertedRelationship in SACM has the *isCounter* Boolean attribute, which can be used to set the nature of the asserted relationship, whether the target element is supporting or disproving the source claim. To support the dynamic asserted relationship, we propose to model the isCounter attribute as an expression over the property of the target element, rather than just static Boolean flag. If the target element doesn't have a specified property, then isCounter can act as a flag, but if the property of the target is specified, then the isCounter can be an expression over that property. For example, if the property of an evidence supporting a contract states the contract violation rate, then the isCounter expression of the corresponding dynamic assertion can compare the violation rate value against a threshold established to indicate if the evidence is actually supporting or disproving the confidence in the corresponding contract.

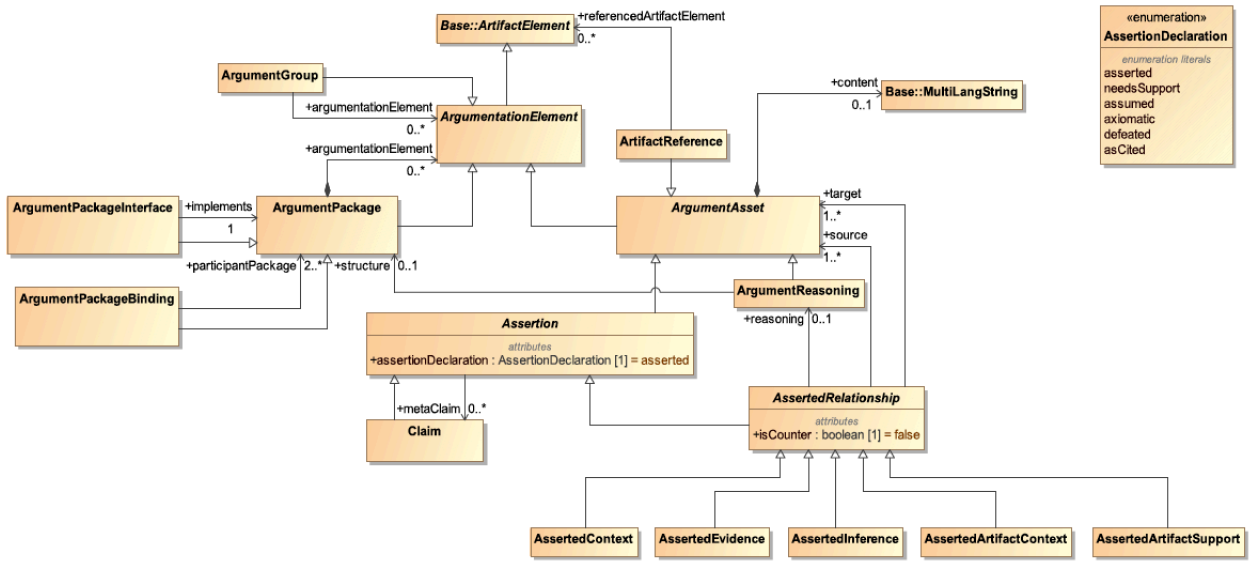

Figure 34: The SACM Argumentation Package Diagram

6.2.3.3 Adapting the argument structure for dynamic assurance

We structure the assurance case accordingly to separate the dynamic parts that need to be re-evaluated continuously, and the static parts that have been evaluated with sufficient confidence (Figure 35). We make the separation on the component assurance level such that each contract that is flagged as runtime is part of the Component dynamic argument, and each contract that is flagged as validated is part of the component static argument.

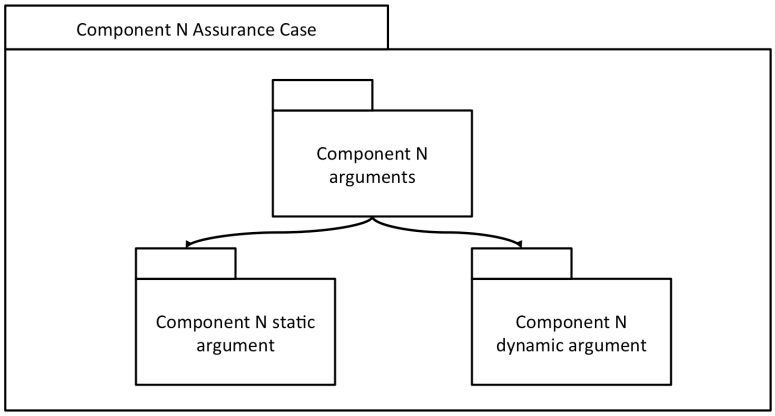

Figure 35: Dynamic component assurance case structure

6.2.3.4 The current tool support

CHESS supports flagging of the contracts via the contract status attribute of the ContractProperty stereotype. Currently, CHESS and Opencert support automated generation of component modules based on the contract-satisfaction argument patterns. Since there is already modelling support for flagging contracts that require runtime support, we are extending the CHESS/Opencert environment to support the separation of component assurance modules on dynamic and static part. Furthermore, since Opencert implements

SACM, we plan to investigate how to implement the support for the dynamic asserted evidence relationship that requires changing the nature of the isCounter attribute of an assertedRelationship element.

6.2.3.5 An application example: Car Platooning example

6.2.3.5.1 Platoon functional architecture

Platooning functionality allows vehicles to cooperate by communicating with each other while in the platoon mode. The safety-criticality of these systems cannot be constrained to only a single vehicle, as each vehicle in the platoon depends on other vehicles. Hence, for a single vehicle to be sufficiently safe, the behaviour of the platoon as a multi-vehicle functionality should be sufficiently safe as well. In this UC we focus on the specific type of platooning intended for trucks. The advantage of being in the platoon is that the trucks would drive with an optimal distance between each other, which significantly improve the platoon fuel efficiency (Davila, 2013). The information about the speed, position and acceleration of the nearby vehicles is faster and more accurate to receive directly from those vehicles than to use local sensors to identify the information. Hence, by using the remote data, platooning allows for closer gap between the vehicles.

Figure 36 illustrates the software architecture model of *platoonManager* in a single vehicle. It takes as inputs *remoteData* received from the nearby vehicles, *sensorDataOwn* received from the local sensors with its own information, and *sensorDataOther* received from the local sensors about the nearby vehicles. The *ownData* is transmitted to other vehicles, while the *accelerationCMD* instructs the corresponding actuators to increase or lower the vehicles acceleration. Each non-lead vehicle is equipped with a similar *platoonManager*.

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

6.2.3.5.2 Safety Analyses and Contract Derivation

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

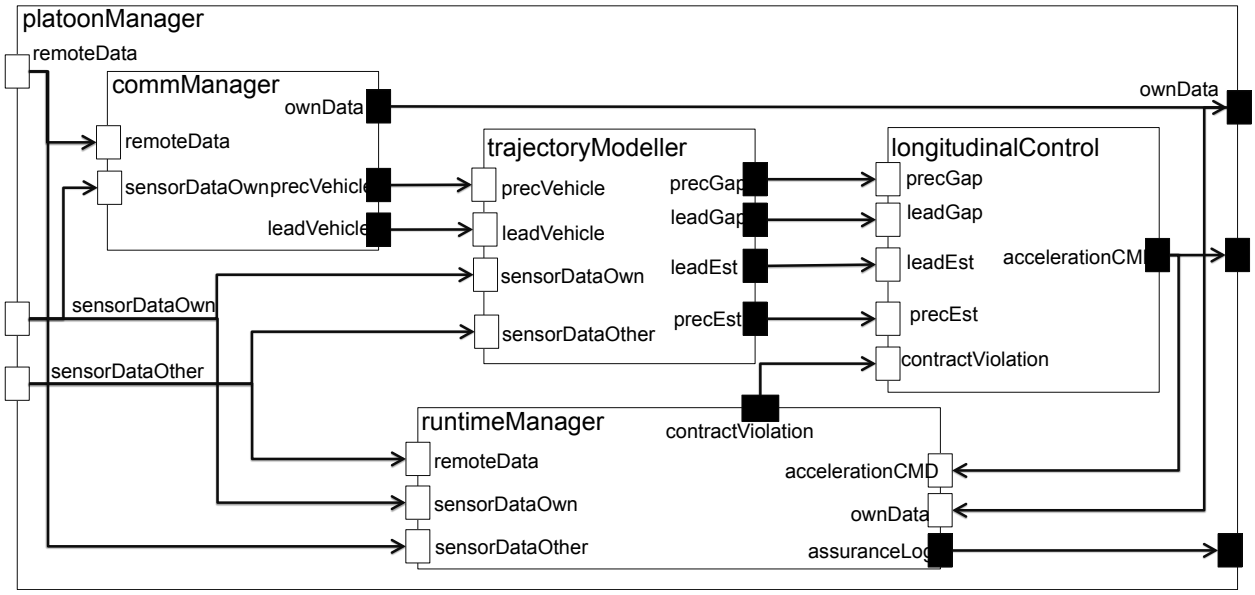

Figure 36: The platoon functionality architecture within a single (non-lead) vehicle

To maintain the safety assurance of such systems, we need to constantly check for validity of the contracts against the different environments and evaluate the confidence in the contracts in the assurance case.

Based on the safety analysis, we specify a subset of the system contracts (Table 3). The strong contract requires the environment to provide information about speed, position and acceleration of the remote vehicle, as well as that every vehicle in the platoon must have maximum deceleration force between 5 and 8 m/s^2. This assumption is required if the system is to avoid collision when a vehicle in front would to brake with full force. Since the time to stop in such case depends also on the freshness of the information received from other vehicles, then in the subsequent weak contracts we describe how the quality of the communication and local sensors affects the system performance. The distance to the proceeding vehicle is being adjusted accordingly, and when the *remoteData* link is unstable, then the vehicle switches to the Automated Cruise Control (ACC) mode that uses local sensor data instead of the data received from the other vehicles. In this scenario, we specify that the weak contracts need runtime support.

The argument-fragment in Figure 37: An argument-fragment assuring confidence in the contract <B1, H1> represented in GSN assures confidence in the contract <B1, H1> following the contract-driven requirements assurance pattern and focusing on the contract completeness branch. We develop the completeness claim by introducing the runtime manager assurance decomposition strategy. For each of the different platoon contexts the runtime manager evaluates the completeness of the contract and feeds the results to the assurance argument where the contract completeness is either supported in the given context, or the goal serves as the counter evidence in contract completeness. For example, the assertion connection between the goal G6 and the solution S2 is a dynamic assertion connection with the isCounter condition over the violation rate of the <B1, H1> contract.

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

Table 3: A subset of the platoon Manager contracts.

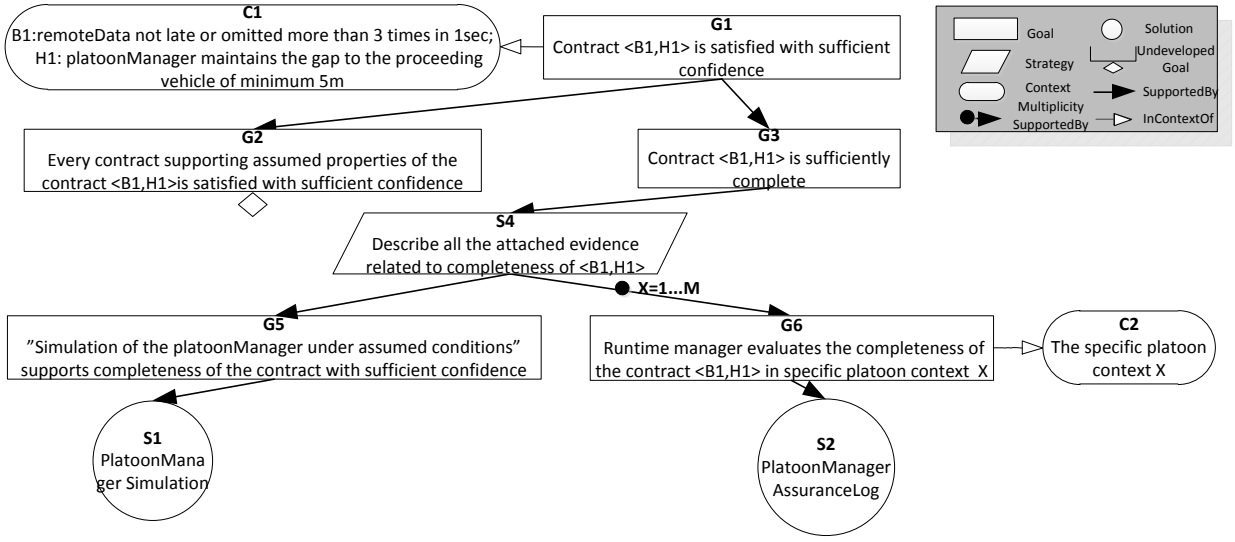

Figure 37: An argument-fragment assuring confidence in the contract <B1, H1> represented in GSN

6.3 Link to run-time support

More and more runtime environments require safety assurance mechanisms that allow to identify, during operation, that all functions are being executed as specified in a functional and non-functional safe manner. Therefore, there is an increasingly need to have monitors that are able to gather information from the runtime environment and its running applications and analyse those data to produce verdicts about their correctness. This, however, brings a challenge to designers and developers: how to produce such observers - usually known as monitors - in a correct way by design, ideally supported by contracts established during design or compilation time, and in a way that these monitors have bounded overhead that do not compromise the original requirements of the runtime environment and applications.

Most of the work being in this direction is being carried out in the discipline of Runtime Verification (RV), where formal languages and methods are used to enable the unambiguous specification of properties and where specialized algorithms were developed to synthesize automatically the corresponding structure and code for the monitors that will observe the system and make verdicts about correctness and safety. Unfortunately, very few of the works focus on real-time embedded systems, rendering them not too much useful for the current and next generation of real-time embedded systems. Still, the works that consider this spectrum of systems have produced promising results. Here, the focus will be on one of such work, namely the RMTLD3Synth which is a framework composed of a timed temporal logic to enable the definition of formal specifications and the associated algorithm to synthesize monitors from specifications, targeting hard real-time embedded systems (although it can be used in virtually any situation where the properties to be verified by the monitors are related to the duration of events, assuming that the necessary instrumentation is performed in the target runtime system).

The RMTLD3Synth framework is based on the RMTLD3 logic, a three-valued timed temporal logic that is expressive enough to capture relevant properties of behaviours that can be described in terms of events occurring in some specified time duration. Its three-valued semantics dictates the fact that, given a finite trace extracted from the run-time system's execution, sometimes it is not possible to make a decision given the absence of data produced by the system (this information might be used by the runtime system to inform running applications that something may be potentially wrong, and therefore trigger some preventive actions in due time, or simply the monitor has to wait for further traces of execution to make a final verdict). The syntax of RMTLD3 is given by the following formal grammar:

$$\eta ::= \alpha \mid x \mid \eta_1 \circ \eta_2 \mid \int^{\eta} \varphi$$

$$\varphi ::= true \mid p \mid \eta_1 < \eta_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \, \mathrm{U}_{\sim\gamma} \, \varphi_2 \mid \varphi_1 \, \mathrm{S}_{\sim\gamma} \, \varphi_2 \mid \exists x \, \varphi$$

In a nutshell, the meaning of each of the constructors is as follows. The meta-variable η denotes terms, whereas the meta variables φ denote formulas. Terms are either a real-valued constant α, a variable *x*, the addition and multiplication operation between two terms, or a duration term $\int^{\eta}\varphi$ meaning the duration of the formula φ in the interval $[0,\eta]$. In the case of formulas, these are either the true Boolean constant, some proposition *p*, the less-than relation over two terms, disjunction of two formulas, negation, the "until" formula meaning that φ_1 holds until φ_2 holds during the interval γ, the "since" formula meaning that φ_1 holds since φ_2 holds during the interval γ, and a formula stating that there exists some valuation for the variable *x* such that φ holds for that value. An example specification, intuitively meaning the property that "the duration of event c within 9 time units is less than 2" can be written as $\int^9 c < 2$. Further complex specifications taking advantage of the expressive power of the logic are available in the online demonstrator (Pedro-2, 2018).

In what comes to the actual monitor generation capabilities of RMTLD3Synth, the underlying algorithm has a static phase that simplifies RMTLD3 specification in order to remove existential operators. This phase uses modern SMT solvers to obtain values that satisfy those existential operators which otherwise would not be possible to deploy in runtime time, since searching for those values would naturally compromise the execution times of monitors and either cause exaggerated interference in the monitored applications or render these monitors useless in case they could not find appropriate values during their temporal execution windows. The generation of the monitors from these simplified specifications is then carried on via a translation into lambda functions that, when composed, satisfy the intended semantics, guaranteeing as well bounded

execution times. The full details on the monitor generation algorithm are available in (Pedro, 2015) and a validation of the approach on a UAV flight control system are available in (Pedro, 2017).

During this section, we have described the RMTLD3Synth tool. This tool is available online in (Pedro-1, 2018) and a small demonstrator is available, also online (Pedro-2, 2018). We wrap up the description with a visual scheme of the process, notably its linking to a runtime monitoring architecture presented in (Nelissen, 2015) and which was specifically designed to provide support for runtime verification frameworks such as the one of RMTLD3Synth. The overall process starts from the establishment of a requirement stating that "if some event *a* takes place, then either *a* or *b* continue two occur until *c* happens during 10 time units, and during that time, the event *c* does not take more than 4 time units". This is formally encoded via the following the RMTLD formula:

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

The encoded formula is then used as input for the RMTLD3Synth tool, which generates the corresponding code, assuming an instrumentation of the target application/run-time environment in order to capture the events of interest (in this case, the events *a*, *b*, and *c*). The image below shows, schematically, the overall process of monitoring applications/run-time environments using the RMTLD3Synth tool.

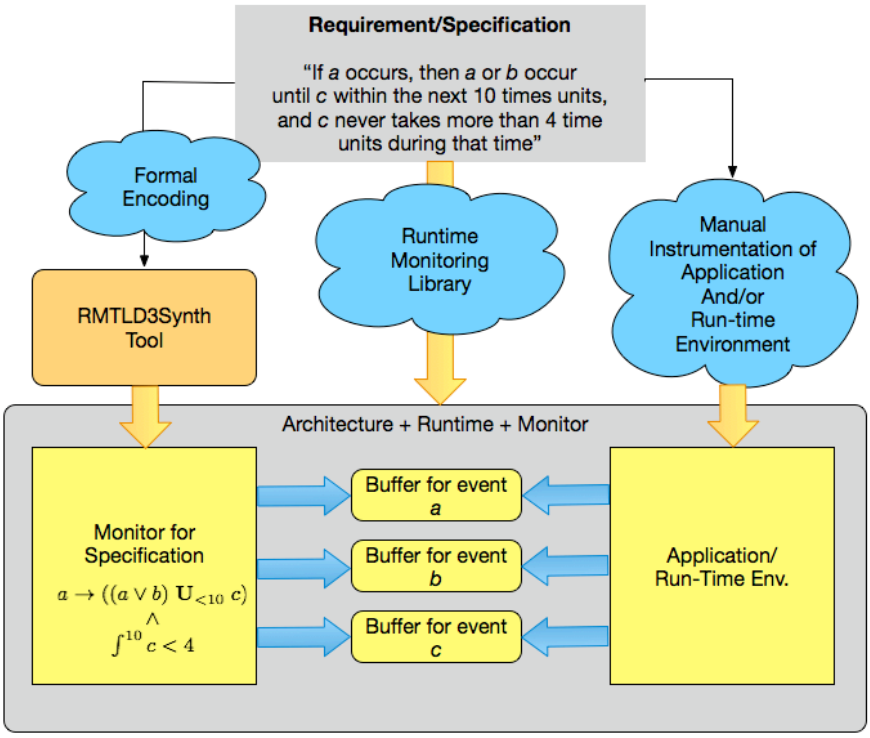

Figure 38 Adding monitoring support via RMTLD3Synth

For the running example, the Figure below shows the code generation process using the online RMTLD3Synth tool (which works as a simple wrapper using a translation from OCaml to JavaScript, although the actual tool which is a console application). Next, we show a snapshot of the code generated by the tools for exemplification purposes. It includes the code for the actual monitor (an instance of C++ class named RMTL_monitor) and the remaining support code.

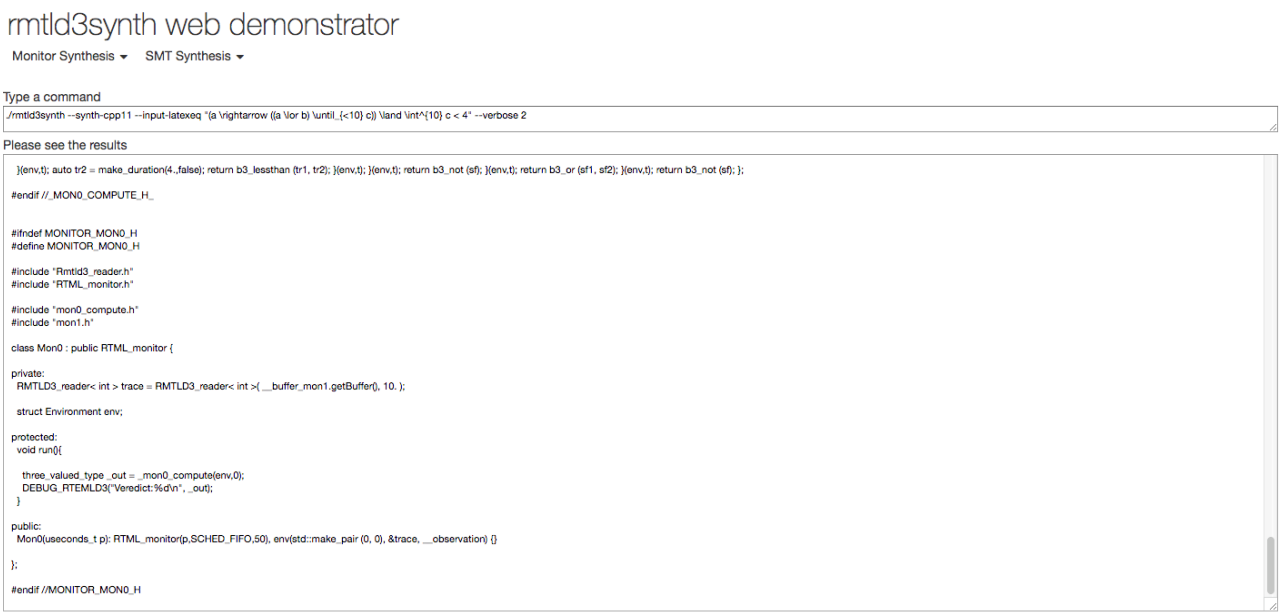

Figure 39 Snapshot of the usage of the RMTLD3Synth web interface.

The generation of monitors via the tools is already mature, although integration with larger development frameworks is still ongoing and expected to be considerably developed within the scope of SafeCOP. Indeed, one of the main drawbacks of the tool (as happens with many of the tools that implement formal languages/methods) is the interface for users. Indeed, with RMTLD3Syth, the kind of user to take advantage of its features should be someone with expert knowledge in temporal logics, which is normally out of reach of CPS engineers. Bringing RMTLD3Synth closer to the level of industrial production is one of the goals in this project, by developing an easier to use DSL with clear syntax and also aiming at the connection with modelling tools, namely the CHESS framework.

Bibliography

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

Berner, C. & Flage, R., 2016. Strengthening quantitative risk assessments by systematic treatment of uncertain assumptions. *Reliability Engineering & System Safety*, Volume 151, pp. 46-59.

Borza, J., 2011. *FAST diagrams: The foundation for Creating Effective Function Models.* [Online] Available at: https://aitriz.org/documents/TRIZCON/Proceedings/2011-06_FAST-Diagrams-The-Foundation-for-Creating-Effective-Function-Models.pdf

Cinatti A., Dorigatti M., Tonetta S. OCRA: Othello Contracts Refinement Analysis.[Online] Available at: https://es.fbk.eu/tools/ocra/download/OCRA_Language_User_Guide.pdf

Cimatti, A., Tonetta, S. 2014. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming*, 97(3), pp. 333–348.

Checkland, P., 1981. *Systems Thinking, Systems Practice.* New York: John Wiley & Sons.

CHESS Eclipse Polarsys project. [Online] Available at: https://www.polarsys.org/chess/start.html

Center of Excellence on Systems Architecture, Management, Economy & Strategy (CESAMES), 2017. *CESAM: CESAMES Systems Architecting Method - Pocket guide.*

CHESSML - CHESS Modelling Language. [Online]. Available at: https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf

DNV GL, 2011. *Recommended Practise RP-A203 - Technology Qualification.*

DNV GL, 2016a. *Enabling confidence - Adressing uncertainty in risk assessments.*

DNV GL, 2016b. *DNVGL-ST-0373 standard: Hardware in the Loop Testing (HIL).*

Eames, D. P. & Jonathan, M., 1999. *The Integration of Safety and Security Requirements.* Toulouse, Springer Berlin Heidelberg, pp. 468-480.

Eldevik, S., Hafver, A., Jakopanec, I. & Pedersen, F. B., 2017. *Risk, Uncertainty, and "What if?" - A practical view on uncertainty and risk in the knowledge- and physical domain.* Portoroz, Slovenai.

Hafver, A. et al., 2015. *Risk - from concept to decision making, In Safety and Reliability of Complex Engineered Systems, ESREL.* Zurich.

Health and Safety Executive UK (HSE), n.d. *ALARP at a glance.* [Online] Available at: http://www.hse.gov.uk/risk/theory/alarpglance.htm [Accessed 17 April 2017].

IBM Software Rational, 2010. *DO-178B compliance: turn an overhead expense into a competitive advantage,* Aerospace and Defense: White paper.

International Organization for Standardization (ISO), 2010. *ISO 31000 - Risk management*.

ISO, 2015. *ISO15288:2015 - Systems and software engineering - System life cycle processes.*

ISO/IEC Guide 51:2014:, n.d. Safety aspects - Guidelines for their inclusion in standards.

Leveson, N., 2003. A new accident model for engineering safer systems. Safety Science, 42(4), pp. 237-270.

Leveson, N., 2011. *Engineering a Safer World: systems thinking applied to safety.* Cambridge , Massachusetts: The MIT Press.

Leveson, N., 2013. *An STPA primer.*

Mazzini, S. ,. F. J. P. ,. B., 2016. CHESS: an open source methodology and toolset for the devel-opment of critical systems.

Medawar S., Sljivo I. and Scholle D., 2017. Cooperative Safety Critical CPS Platooning in SafeCOP. *In 5th EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems*.

OCRA, 2015. [Online] Available at: https://ocra.fbk.eu/

OpenCert Eclipse Polarsys project. [Online] Available at: https://www.polarsys.org/projects/polarsys.opencert

SafeCOP D2.1 Deliverabe, 2016. *State-of-the-art on safety assurance,* SafeCOP Consortium.

Sljivo I., 2015. Facilitating reuse of safety case artefacts using safety contracts. *Licentiate thesis*. Mälardalen University.

Sljivo I., Gallina B., Carlson J., Hansson H., Puri S., 2016. A Method to Generate Reusable Safety Case Argument-Fragments from Compositional Safety Analysis. *Journal of Systems and Software: Special Issue on Software Reuse*, 131, pp. 570-590.

Society for Risk Analysis (SRA), 2015. *SRA Glossary.* [Online]
Available at: http://www.sra.org/sites/default/files/pdf/SRA-glossary-approved22june2015-x.pdf

Torres, R. J. A., 2017. *Master's thesis: Hybrid Modelling for Dynamic Reliability Assessment in Subsea Oil and Gas Processing Systems.*

XSTAMPP, 2017. [Online] Available at: http://www.xstampp.de/

(Sulaman, S., Beer, A., Felderer, M. et al.) Facilitating reuse of safety case artefacts using safety contracts. Software Qual J. https://doi.org/10.1007/s11219-017-9396-0 (2017)

(Sotomayor Martínez, Rodrigo) System theoretic process analysis of electric power steering for automotive applications. Master thesis. Massachusetts Institute of Technology (2015)

(N. Leveson) Engineering a Safer World: Systems Thinking Applied to Safety (2011)

(Ioannis M. Dokas, John Feehan, Syed Imran) EWaSAP: An early warning sign identification approach based on a systemic hazard analysis. Cork Constraint Computation Centre, University College Cork,Cork,Ireland, Safety Science 58 (2013)

A. Davila, “Report on fuel consumption, revision 13.0”, SARTRE, Deliverable 4.3, January 2013.

(Pedro, 2015) Pedro, A., Pinto, J.S., Pereira, D., Pinho, L.. M., 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.S., Pereira, D., Pinho, L..M., 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.

(Pedro-1, 2018) Pedro, A., Pinto, J.S., Pereira, D., Pinho, L..M., Rmtld3synth - Runtime Verification toolchain for generation of monitors based on the restricted Metric Temporal Logic with Durations. Available online: https://github.com/cistergit/rmtld3synth

(Pedro-2, 2018) Pedro, A., Pinto, J.S., Pereira, D., Pinho, L..M., rmtld3synth web demonstrator. Available online: https://anmaped.github.io/rmtld3synth/

7 Appendix: Use Case UC5 SafeCOP Safety Analysis results

We are currently applying the framework and corresponding methods and tools presented in deliverables D2.2 and D2.4 to the Use Cases in the project. We have shown throughout this deliverable, using examples, how the methods can be applied. We have used in Chapters 3-5 as an example Use Case UC2 “Cooperative bathymetry with boat platoons”. In this appendix, we have decided to include the most mature result we have so far on the application of the SafeCOP Safety Analysis, namely the results obtained on UC5 “Vehicle-to-Infrastructure (V2I) cooperation for traffic management”.

7.1 Introduction

In order to provide a qualitative feedback on the newer systemic approaches described in Section 2.2, two different approaches for Hazard and Risk Analysis (HRA) will be applied on the same system to better estimate the disparity (if any) of the potential identifiable hazards, flaws and items in the early steps of the design process that could jeopardize the system safety and possibly affect the reliability and the performance of its components. For the purpose of this study, the ISO 26262 standard will be used to support the “traditional” HRA analysis in order to provide both a reference risk classification scheme and a context for comparison with the STAMP methodology.

7.2 Hazard and Risk Analysis

The Hazard and Risk Analysis (HRA) is the initial step of all design processes that deal with functional safety. The two-main reference standard for functional safety are IEC 61508 and ISO 26262, the former being older and generic, the latter newer and specific of the automotive domain. As Figure 40 shows, the HRA step is performed at the beginning of the safety lifecycle.

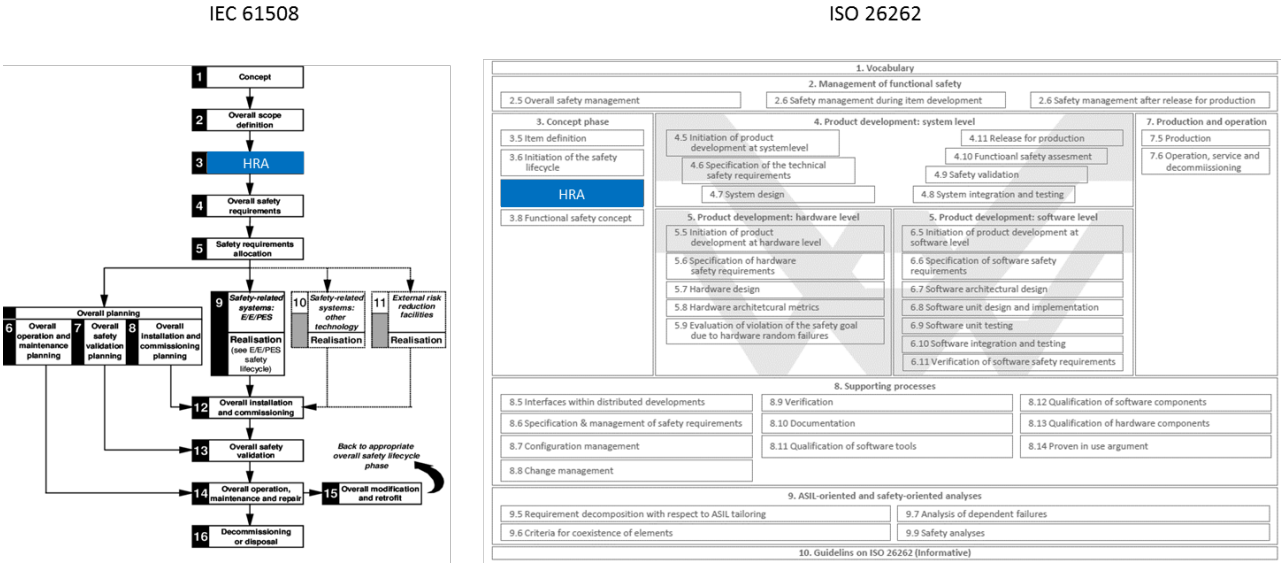

Figure 40: HRA in standard safety lifecycles (see the respective standards for the details in the figures).

Dealing with functional safety means in first place to identify the “risks” of the equipment under control (EUC) and the associated safety function. In this context, and according to the abovementioned standards, the risk is specific function of the frequency (or likelihood) of a hazardous event and the severity of the consequences of that event. When the risk associated to a certain hazard is not acceptable, then one or more

"safety functions" shall be applied to reduce the initial risk to a tolerable level. A safety function is a function intended to achieve or to maintain a "safe state" for the equipment under control, i.e. to maintain the system in a state where the hazardous event will not lead to dangerous consequences. The "measure" of the risk reduction level is called "Safety Integrity Level" (SIL, according to IEC 61508), or, specifically for the automotive industry by the "Automotive Safety Integrity Level" (ASIL, ISO 26262). Alternatively, the safety integrity level (SIL or ASIL) can also be defined as the probability of the safety-related system performing the required safety functions under all stated conditions within a stated period of time (safety process time).

Though very similar, the concepts of SIL and ASIL refer to different probabilities but can be mapped one to another according to the scheme in Table 4: Mapping between SIL and ASIL.

IEC 61508	N/A	SIL 1	SIL 2		SIL 3	SIL 4
ISO 26262	QM	ASIL A	ASIL B	ASIL C	ASIL D	N/A

Table 4: Mapping between SIL and ASIL.

Determining the necessary integrity level associated to each safety function is the starting point for the specification and design of all the safety related portions of the system. The standards, in fact, provide a set of technical guidelines indicating the documents, the analyses, the techniques and the tools to be used for specification and for hardware/software design based on the specific integrity level. As an example, a safety function with ASIL D integrity level shall be documented and modelled using a formal language, while for lower integrity levels semiformal description are acceptable.

The approach proposed by ISO 26262 for HRA and adopted for individual subsystems of the use-case, is organized in the four phases described in the following.

Situation analysis and hazard identification. The operational situations (operating modes) and the environmental or external conditions in which the system may undergo a malfunctioning are to be considered as the malfunctioning behavior may trigger potential hazards. These situations and the corresponding potential hazards shall be described and evaluated. Then, the potential hazards are determined at system level through brainstorming and/or other techniques such as field studies. The consequences of these potential hazards are to be identified, documented and systematically collected in a specific document. The operating modes identified for all subsystems are listed in Table 5:

Id	**Name**	**Description**
[S1]	Off	The system is powered down.
[S2]	Inactive	The system is not active and is not required to perform its functions.
[S3]	Isolated	The system is fully operational, but is isolated from the rest of the infrastructure and vehicles. This may happen when the other to the V2I infrastructure elements are not connected, or no vehicles are connected.
[S4]	Connected	The system is fully operational and is reliably communicating with the rest of the infrastructure.

Table 5: Operating modes.

To describe the external and environmental conditions where the systems are operating, the following aspects have been considered: traffic conditions, environmental condition, road type, vehicle speed and camera condition. For each of these aspects different qualitative levels have been defined as shown in Table 6.

Aspect	Values/Levels
Traffic condition	Low, High
Environmental condition	Normal, Wet/Icy, Low visibility
Road type	Urban, Extra-urban
Vehicle Speed	Slow (< 30km/h), Medium (30 - 80km/h), Fast (> 80km/h)
Camera condition	Normal, Covered/Moved

Table 6: External aspects for HRA.

This first phase is completed when all the hazards have been identified in all possible combinations of operating modes and external conditions.

Hazard classification. This step has the goal of classifying the identified potential hazards based on the estimation of three factors: "Severity", "Exposure", and "Controllability". The severity expresses the severity of potential harm to the driver, the passengers, occupants of other vehicles, and pedestrians and is classified according to the four levels reported in Table 7. The exposure expresses the likelihood of the hazardous event given the operational and external conditions specified. Exposure is expressed according to the levels reported in Table 8. Finally, the controllability expresses an evaluation of the possibility of avoidance of a specific harm, i.e. an estimation of the probability that the driver or other impacted persons are able to gain control of the hazardous event that is arising and are able to avoid the specific harm. Controllability is classified according to the levels summarized in Table 9.

Level	Severity
S0	No injuries
S1	Light and moderate injuries
S2	Severe and life-threatening injuries (survival probable)
S3	Life-threatening injuries (survival uncertain), fatal injuries

Table 7: Hazard severity levels

Level	Exposure
E0	Incredible
E1	Very low probability
E2	Low probability
E3	Medium probability
E4	High probability

Table 8: Hazard exposure levels

Level	Controllability

C0	Controllable in general
C1	Simply controllable
C2	Normally controllable
C3	Difficult to control or uncontrollable

Table 9: Hazard controllability levels

ASIL determination. When severity, exposure and controllability of each hazard have been defined, the integrity level is determined according to Table 10, taken from the ISO 26262 standard. It is worth noting that a similar table for determining the SIL level is provided by the IEC 61508 standard.

		C0	**C1**	**C2**	**C3**
S0	**E0**	QM	QM	QM	QM
	E1	QM	QM	QM	QM
	E2	QM	QM	QM	QM
	E3	QM	QM	QM	QM
	E4	QM	QM	QM	QM
S1	**E0**	QM	QM	QM	QM
	E1	QM	QM	QM	QM
	E2	QM	QM	QM	QM
	E3	QM	QM	QM	ASIL-A
	E4	QM	QM	ASIL-A	ASIL-B
S2	**E0**	QM	QM	QM	QM
	E1	QM	QM	QM	QM
	E2	QM	QM	QM	ASIL-A
	E3	QM	QM	ASIL-A	ASIL-B
	E4	QM	ASIL-A	ASIL-B	ASIL-C
S3	**E0**	QM	QM	QM	QM
	E1	QM	QM	QM	ASIL-A
	E2	QM	QM	ASIL-A	ASIL-B
	E3	QM	ASIL-A	ASIL-B	ASIL-C

	E4	QM	ASIL-B	ASIL-C	ASIL-D

Table 10: Integrity level determination matrix

Safety goal formulation. Finally, a "safety goal" is determined for each hazardous event evaluated in the hazard analysis. By definition, safety goals are top-level safety requirements. They lead to the functional safety requirements needed to avoid an unreasonable risk for each potential hazard. Safety goals are not expressed in terms of technological solutions, but in terms of functional objectives. The ASIL determined for the hazardous event is then assigned to the corresponding safety goal. A potential hazard may have more than one safety goal, and if similar safety goals are determined, they can be combined into one safety goal that will be assigned the highest ASIL of the similar goals.

The output of the HRA conducted according to the process describe above is thus a list of hazards, each associated with its safety integrity level and its safety goal.

7.3 STPA/EWaSAP

As already described in Section 2.2, STPA is a tool developed to include the causal factors identified by the STAMP methodology with a top-down approach. As outlined in Section 4, the application scenario at hand has a particular focus on special control actions that enforce control over different sub-system components even by non-controller actors.

For this reason, an extension of STPA analysis called *Early Warning Sign Analysis based on STPA* (EWaSAP) is included to add awareness actions, enabling controllers to transmit warnings and alerts that justify the presence of flaws and vulnerabilities in their controlled process based on the process models they possess.

This model is used as a complementary tool to find factors that are out of the pool of the possible perceived data that traditional hazard analyses are unable to detect due to limitations of the inner nature of sequential accident models, such as:

- Managerial deficiencies
- Safety culture flaws
- Undesirable behaviours and interactions of system components
- Software flaws
- System changes due to evolution and adaptation that affect safety

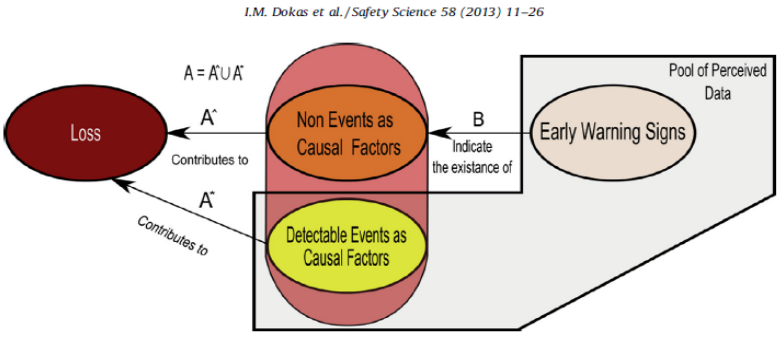

Figure 41: EWaSAP justification model

The analysis provides for three main steps:

- **Step 1**: find anyone/anything outside the system who need to be informed about perceived progress status (e.g. emergencies operators)
- **Step 2**: identify useful tools (e.g. sensory devices) belong to systems outside the one in focus and establish synergy
- **Step 3**: Enforce Internal Awareness Actions

The complete sequence of steps to correctly apply the model is described in Table 11:

Execution sequence	STPA steps[a]	EWaSAP steps	EWaSAP steps description
1	STPA (1)		
2		EW (1)	Aim: Decide if there is anyone outside the system in focus (i.e. emergency responders) who needs to be informed about the perceived progress of the hazard or about its occurrence
3	STPA (2 a, b, c)		
4		EW (2)	Aim: Identify useful sensory services (i.e. video surveillance cameras pointing at the system in focus) installed in or possessed by systems outside of the system in focus and establish synergy
		a	For each top level safety constraint identify those signs which indicate its violation
		b	Find those systems in the surrounding environment with sensors capable of perceiving the signs defined in EW(2a) and request to establish synergy. The objective is to have the surrounding systems agree on the adoption of appropriate awareness actions, so that to be able of transmitting voluntary/synergetic early warnings to the appropriate recipients as per EW(1) about the occurrence of a top level safety constraint violation
5	STPA (3 a, b)		
6		EW (3)	Aim: Enforce Internal Awareness Actions
		a	Describe what needs to be monitored, and what type of features/capabilities the sensors must have so that to make the appropriate controllers capable of perceiving: (a) The signs indicating the occurrence of the flaw (b) The violation of the assumptions made during the design of the system
		b	After the design trade-offs and when it is known which sensors have been selected for installation into the system, define which patterns of perceived data indicate the occurrence of the flaw and/or the violation of its designing assumptions
		c	Update the process models of the controllers with appropriate awareness and control actions, which should be enforced based on the perceived early warning signs, so that to warn about, adapt to, or eliminate the causal factor to the loss which is present in the system
		d	For each perceived warning sign of the previous step, define its meta-data/attribute values (i.e. how the message will be coded/written by the transmitter) as to ensure that it will be perceived and ultimately understood by the appropriate controller/s
7	STPA (4)		

Table 11: EWaSAP steps as add-ons to STPA and their logical sequence of execution

7.4 V2I scenario

The goal of the V2I scenario developed within the use-case UC5 is twofold, namely:

- To reduce traffic congestions by means of a wide-area traffic light management optimization, and, consequently to reduce pollution. This goal is not safety-related.
- To reduce the risk of accidents at road crossings by a real-time management of traffic lights and I2V control of the speed/breaking of vehicles in the surrounding of the crossing. This goal is safety-related and must be approached by combining traditional safety approaches (IEC-61508, ISO-26262) with a novel run-time safety assurance methodology.

The reference overall system view is described in Figure 42.

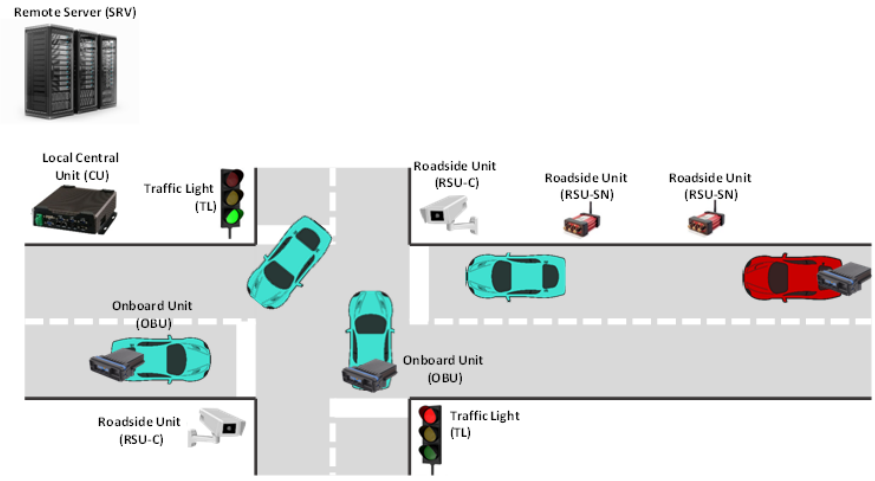

Figure 42: Overall system view of the V2I scenario

Here follows a brief description of the system components:

OBU. The on-board unit will be adapted to support redundant acceleration sensors, to implement integrity checks by means of diagnostic functions and to communicate with the CU according to the MQTT application protocol already defined.

RSU-C. The Road Side Unit equipped with camera and video monitoring system (RSU-C) will be adapted to support redundancy in the video acquisition and in the video analytics processes. Moreover, it implements integrity checks by means of self-diagnostic functions. More specifically, it is composed of a couple of cameras monitoring the same area (acquisition redundancy) and two video analytics processes (processing redundancy).

RSU-SN. The road side unit - sensor network will be formed by the following elements: two temperature sensor nodes, two light sensors, one sink node with no sensing capabilities and one SBC acting as gateway towards the V2I infrastructure. Each sensor node will provide the measure of temperature and ambient light through its sensor-board. The measurements coming from the sensor nodes are encrypted using the hybrid cryptography scheme and sent to a special sensor node (the sink node) connected to the SBC via a UART port.

TL. The traffic light is directly controlled by the LCU and acts as an actuator of the Control Centre (which is the entire set of decision taken by the SRC and the LCU).

LCU. The local decisions and actions are taken from the Local Control Unit (CU) and concern real-time and potentially safety-critical actions (e.g. messages to vehicles and traffic light control in case of a hazardous condition).

SRV. Longer term decisions and control actions are managed, with a lower priority, by the remote server(SRV).

The high-level distributed and cooperative architecture shown in Figure 43 has been identified. In this figure, the red paths represent safety-critical real-time data being exchanged locally (e.g. Ethernet, WiFi, 802.11.p, 5G) and intended also (not necessarily only) for the realization of the safety function. The green paths, on the other hand, indicate data being exchanged though a cellular wide-area network for non-safety-critical purposes, namely for traffic management.

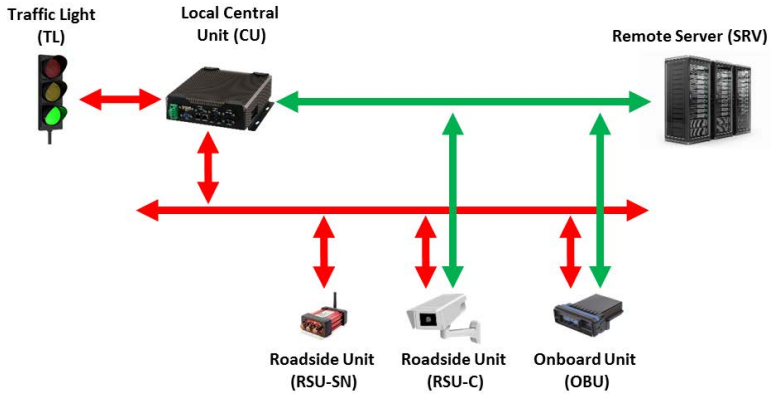

Figure 43: Reference architecture for the V2I application

7.5 Results

The following three sections summarize the results that have been obtained by the adoption of the traditional ISO 26262 hazard and risk analysis approach and the STAMP/STPA approach.

7.5.1 ISO 26262 HRA Results

In the initial phases of the analysis, each component identified in Section 4 is analysed to extract the goals and the individual functions as emerged from the requirements. The steps taken to complete the process are as follows:

1. Describe each subsystem in general terms and express the requirements as high-level functions
2. Identify potentially critical hazards
3. Identify the vehicle-related and environmental condition to consider during the HRA
4. Define the safety goals and their integrity level
5. Define the safe-states and safe-operational modes of the system

The main goal of the infrastructure is both to manage a traffic lights and exchange information with vehicles in order to:

[G1] Reduce traffic. This goal shall be achieved by:

[G1.1] Indicating to individual vehicles the optimal driving speed

[G1.2] Control the semaphore based on instantaneous traffic conditions

[G2] Minimize pollution. This goal shall be achieved by

[G2.1] Reducing the waiting time for the green light

[G3] Prevent hazardous situations. This goal is achieved by

[G3.1] Identifying specific “Vehicle Events” (see description below)

[G3.2] Notifying the identified events to the rest of the system

[G3.3] Controlling the braking system of individual vehicles

Such goals are achieved in a cooperative scenario, where all information circulating among the different actors of the V2I infrastructure, and vehicles is combined to compute the decisions and the actuation to be taken. Based on the items defined in the initial phase, the following “Roads Measures” are considered:

[M1] Number of vehicles. Number of vehicles crossing the intersection captured by the RSU-C camera.

[M2] Classification. Number of vehicles per class (bike, car, truck). captured by the RSU-C camera.

[M3] Direction. Number of vehicles classified according to their direction, captured by the RSU-C camera.

[M4] Speed. Speed is sent by vehicles or from the RSU-C camera.

[M5] Acceleration. Acceleration is sent by vehicles.

[M6] Position. Position is sent by vehicles.

[M7] Heading. Direction of a vehicle is sent by vehicles.

[M8] Temperature. Temperature is sent by vehicles or RSU-SN.

[M9] Humidity. Humidity is sent by RSU-SN.

[M10] Fog. Fog is sent by RSU-SN, RSU-C.

[M11] Ice. Ice is sent by RSU-SN.

To give an example of the level of detail of the analysis performed for each item, the Local Control Unit is considered. The analysis starts with the identification of the LCU high-level functional requirements:

[R1] Collect notification of all the events and measures from RSUs, and OBUs, and TL.

[R2] Check consistency of the information received by different sources (OBUs, RSUs, TL) (ref UC5042, UC5046, UC50472)

[R3] When a dangerous situation is detected (by reception of events, check of consistency, or analysis of measures), communicate to OBUs to warn the driver, to reduce speed or to completely break, and, eventually, to TL to move to permanent red, in some direction.

[R4] Depending on the changes of traffic conditions, send information to the TL to control the duration of the three phases (green, yellow, red) with respect to a reference direction (e.g. blinking mode when the traffic is very low, remove a permanent red, or decrease/increase the green/red period) (ref UC5038, UC5039)

[R5] Communication avoidance to vehicles to slow down or to break, or to TL blink, or to move to permanent red, or yellow in some direction, in absence of a dangerous situation (ref UC5041)

As already anticipated in the general description of the HRA approach, each item is characterized by an operating mode. For this reason, the LCU also is describe in terms of the following operating modes:

[S1] Off. The system is powered down.

[S2] Inactive. The system is not active and is not required to perform its functions.

[S3] Isolated. The system is fully operational but is isolated from the rest of the infrastructure and vehicles. This may happen when the other to the V2I infrastructure elements are not connected, or no vehicles are connected.

[S4] Connected. The system is fully operational and is reliably communicating with the rest of the infrastructure.

Since the LCU is the subsystem implementing the major part of the core application logic and, as a result of its computation, is responsible to perform actuations, it is crucial to consider such actuations in the HRA. LCU actuations, in fact, can lead to dangerous hazards. The actuations considered are the following:

[A1] WARNING to OBU

[A2] SLOWDOWN to OBU

[A3] BRAKE to OBU

[A4] RED to TL

[A5] YELLOW to TL

[A6] GREEN to TL

[A7] FLASHING to TL

The functions performed by the LCU are the following:

[F1] Acquire event sand measures from OBUs and RSUs,

[F2] Notify actions to all the OBUs and TL according to the event severity.

[F3] Check the consistency of the measures received and elaborate the status of the traffic conditions

[F4] If the presence of inconsistent situation is detected, or the V2I infrastructure is not operational, notify action to TL to blink.

[F5] If the presence of dangerous situations is detected (e. g. inattentive driving, high traffic, ice, or fog), notify a corresponding message to all the OBUs, according to the severity of the situation.

[F6] According to changes traffic conditions, control the duration of the TL phases

Finally, considering measures received by the LCU, the functions that it is supposed to perform and the actuation that it is capable of controlling, the following hazards have been identified:

[H0] The L-CU fails to acquire events

[H1] The L-CU fails to acquire measures

[H2] The L-CU fails to notify a dangerous situation to OBUs

[H3] The L-CU notifies a warning action to OBUs, in absence of danger

[H4] The L-CU notifies a slowdown action to OBUs, in absence of danger

[H5] The L-CU notifies a brake action to OBUs, in absence of danger

[H6] The L-CU fails to detect an inattentive driving situation

[H7] The L-CU fails to detect high traffic situation

[H8] The L-CU fails to detect icy road situation

[H9] The L-CU fails to detect fog situation

[H10] The L-CU communicates a wrong message to TL to increase GREEN phasing, for a direction, in a dangerous condition

[H11] The L-CU communicates a wrong phasing message to TL, in a dangerous condition

[H12] The L-CU communicates a wrong BLINKING message to TL, in a dangerous condition

This same procedure has been followed for the OBU, the RSU-C and the RSU-SN, leading the results summarized below in Table 12, Table 13, Table 14. Figure 44: Example of HRA worksheet shows an example of the worksheet used for the HRA process.

Hazard description	Operating mo	Road	Speed	Environment	Exposure comment	E	Severity comment	S	Controllability comment	C	ASIL	Safety goal
The OBU warns the driver of a potential danger while there is no actual danger	Inactive	Not relevant	Not relevant	Not relevant	The system shall be inactive	E0	To be inactive the vehicle is key-off, thus is not moving	S0	The vehicle need not to be controlled	C0	QM	None (Quality Management)
	Isolated	Not relevant	Not relevant	Not relevant	The system is not connected and shall thus receive no notifications from infrastuctre	E1	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C2	QM	
	Connected	Low	< 30 Km/h	Normal	In low traffic conditions, a misdetection of a dangerous condition is rare	E1	At low speed injuries can be limited	S0	The driver can ignore the warning	C0	QM	
				Wet or icy road	In low traffic conditions, a misdetection of a dangerous condition is rare	E1	At low speed injuries can be limited	S0	The driver can ignore the warning	C1	QM	
				Low visibility	Low visibility degrades camera-based detection algorithms	E2	At low speed injuries can be limited	S0	The driver can ignore the warning	C0	QM	
			30 - 80 Km/h	Normal	In low traffic conditions, a misdetection of a dangerous condition is rare	E1	At medium speed injuries can be limited	S0	The driver can ignore the warning	C0	QM	
				Wet or icy road	In low traffic conditions, a misdetection of a dangerous condition is rare	E1	At medium speed injuries can be limited	S0	The driver can ignore the warning	C1	QM	
				Low visibility	Low visibility degrades camera-based detection algorithms	E2	At medium speed injuries can be limited	S0	The driver can ignore the warning	C0	QM	
			> 80 Km/h	Normal	In low traffic conditions, a misdetection of a dangerous condition is rare	E1	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C1	QM	
				Wet or icy road	In low traffic conditions, a misdetection of a dangerous condition is rare	E1	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C2	QM	
				Low visibility	Low visibility degrades camera-based detection algorithms	E2	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C1	QM	
		Heavy	< 30 Km/h	Normal	Heavy traffic raises the probability of misdetection	E2	At low speed injuries can be limited	S1	The driver can ignore the warning	C0	QM	
				Wet or icy road	Heavy traffic raises the probability of misdetection	E2	At low speed injuries can be limited	S1	The driver can ignore the warning	C1	QM	
				Low visibility	Heavy traffic raises the probability of misdetection, further increased by low visibility	E3	At low speed injuries can be limited	S1	The driver can ignore the warning	C0	QM	
			30 - 80 Km/h	Normal	Heavy traffic raises the probability of misdetection	E2	At medium speed injuries can be limited	S1	The driver can ignore the warning	C0	QM	
				Wet or icy road	Heavy traffic raises the probability of misdetection	E2	At medium speed injuries can be limited	S1	The driver can ignore the warning	C1	QM	
				Low visibility	Heavy traffic raises the probability of misdetection, further increased by low visibility	E3	At medium speed injuries can be limited	S1	The driver can ignore the warning	C0	QM	
			> 80 Km/h	Normal	Heavy traffic raises the probability of misdetection	E2	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C1	QM	
				Wet or icy road	Heavy traffic raises the probability of misdetection	E2	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C1	QM	
				Low visibility	Heavy traffic raises the probability of misdetection, further increased by low visibility	E3	At high speed injuries can be severe	S2	The driver can be distracted by the warning	C1	QM	

Figure 44: Example of HRA worksheet

ID	Hazard Description	Safety Goal	ASIL	Safe state
H0	The OBU warns the driver of a potential danger while there is no actual danger	None	QM	N/A
H1	The OBU notifies the infrastructure of a potential danger while there is no actual danger	None	QM	N/A
H2	The OBU fails to warn the driver about the presence of danger	None	QM	N/A
H3	The OBU fails to notify the infrastructure about the presence of a danger	The OBU shall guarantee notification of the presence of a danger to the infrastructure	ASIL-A	Inhibit transmission to the infrastructure
H9	The OBU communicates wrong dynamic or operational state information to the infrastructure	The OBU shall transmit correct dynamic and operational state data to the infrastructure	ASIL-A	Inhibit transmission to the infrastructure
H4	The OBU slows down the car while there is no actual danger	When Isolated or Inactive, the OBU shall never slow down the vehicle	ASIL-B	Inhibit commands to the Control CAN Bus
H6	The OBU brakes while there is no actual danger	When Isolated or Inactive, the OBU shall never brake the vehicle	ASIL-C	Inhibit commands to the Control CAN Bus
H4	The OBU slows down the car while there is no actual danger	When Connected the OBU shall not slow down the vehicle in absence of a danger	ASIL-B	Inhibit commands to the Control CAN Bus
H6	The OBU brakes while there is no actual danger	When Connected the OBU shall not brake the vehicle in absence of a danger	ASIL-C	Inhibit commands to the Control CAN Bus
H5	The OBU fails to slow down the car in presence of a danger	None	QM	N/A
H7	The OBU fails to brake in presence of a danger	None	QM	N/A
H8	The OBU fails to communicate dynamic or operational state information to the infrastructure	None	QM	N/A

Table 12: HRA results for the on-board unit (OBU)

ID	Hazard Description	Safety Goal	ASIL	Safe state
H0	The RSU-C fails to detect a car accident or a stationary vehicle in a dangerous position	The RSU-C shall guarantee that there are no car accident or stationary vehicles if it has detected none	ASIL-B	Notify a ASIL reduction and inhibit any other transmission to the CU
H2	The RSU-C fails to detect a vehicle moving along a forbidden direction	The RSU-C shall guarantee that there are no vehicle moving along forbidden direction if it has detected none	ASIL-B	
H4	The RSU-C fails to detect a pedestrian in a dangerous position	The RSU-C shall guarantee that there are no pedestrian in a dangerous position if it has detected none	ASIL-B	
H1	The RSU-C detects a car accident or a stationary vehicle in a dangerous position while there is none	The RSU-C shall detect car accidents or stationary vehicles in a dangerous position	ASIL-B	
H3	The RSU-C detects a vehicle moving along a forbidden direction while there is none	The RSU-C shall detect a vehicle moving along a forbidden direction	ASIL-B	
H5	The RSU-C detects a pedestrian in a dangerous position while there is none	The RSU-C shall not detect a car accident or a stationary vehicle in a dangerous position while there is none	ASIL-B	
H6	The RSU-C detects a wrong number of vehicles	None.	QM	N/A
H7	The RSU-C fails to send its operational state information	None	QM	N/A

Table 13: HRA results for the road-side camera (RSU-C)

ID	Hazard Description	Safety Goal	ASIL	Safe state
H0	The RSU-SN sends measurements associated to a potential danger while there is no actual danger	None	QM	N/A

H1	The RSU-SN sends measurements associated to a safe condition while there is potential danger	The RSU-SN shall inhibit transmission when measures are uncertain	ASIL-B	Inhibit transmission to the infrastructure
H2	The RSU-SN WSN fail to communicate sensor data to the SBC	None	QM	N/A
H3	The RSU-SN SBC fails to communicate with the CC	None	QM	N/A

Table 14: HRA results for the road-side wireless sensor network (RSU-SN)

7.5.2 STAMP/STPA Results

Following the STAMP guidelines, the analysis starts by identifying the goals of the system (described in Section 4) and the system accidents and hazards from a high-level perspective. It has to be noted that the definition of what it is to be considered during the assessment as an accident or an unacceptable loss in the system has to be made before any other consideration which is safety relevant.

The following list is an example of high-level system accidents defined during STPA preliminary phase:

[A-1] Vehicle occupants are injured/killed during operation

[A-1.1] Collision between two or more vehicles

[A-1.2] Collision between vehicle and a moving body

[A-1.3] Collision between vehicle and a static body

[A-2] Vehicle damages occurs (economic loss)

[A-3] Infrastructure damages occurs

After the accidents are defined, the system-level hazards are identified:

[H1] A vehicle in the crossing where V2I operates passes the street when the traffic light for its lane is red

[H2] A vehicle stops or does not move when it is in the middle of the crossing where V2I operates

[H3] A vehicle does not obey the minimum separation distance between itself and any possible obstacle in the crossing where V2I operates

[H4] A vehicle drives off road in the crossing where V2I operates (I.E. in the pavement)

[H5] A vehicle parks in a forbidden slot on the road where V2I operates

[H6] A vehicle drives in the wrong way of march in the crossing where V2I operates

[H7] A vehicle attempts to pass by a road section when its physical dimensions are excessive/disproportional

[H8] A vehicle within the V2I network operates outside the operative area

[H9] V2I integrity level is violated

[H10] V2I causes or contributes to a near miss collision between multiple vehicles

[H11] V2I provides commands that contradicts the commands of the Law enforcement

It is useful to link the identified hazards and accidents right away to have an idea of the goals and the effort to be applied on the system from a safety perspective. This relation is reported in Table 15.

		Accidents		
		A1	**A2**	**A3**
Hazards	**H1**	X	X	X
	H2	X	X	
	H3	X	X	X
	H4	X	X	X
	H5	X	X	X
	H6	X	X	X
	H7	X	X	X
	H8	X	X	X
	H9			X
	H10	X		
	H11		X	

Table 15 Relation between hazards and accidents

All of the information above mentioned are translated into safety constraints that can be used during the engineering process of the system, as shown in Table 16:

Hazard	**Safety Constraint**
A vehicle in the crossing where V2I operates passes the street when the traffic light for its lane is red	Vehicles must never violate the red-light order to cross the reference lane
A vehicle stops or does not move when it is in the middle of the crossing where V2I operates	Vehicles must always keep the crossing free
A vehicle does not obey the minimum separation distance between itself and any possible obstacle in the crossing where V2I operates	Vehicles must never violate the minimum separation requirements between objects or bodies
A vehicle drives off road in the crossing where V2I operates (I.E. in the pavement)	Vehicles must always operate in the expected road fragments within the V2I area

A vehicle parks in a forbidden slot on the road where V2I operates	Vehicles must always use the expected parking slots when they are not operative
A vehicle drives in the wrong way of march in the crossing where V2I operates	Vehicles must always drive in the appropriate lane with the respect of the right direction of travel
A vehicle attempts to pass by a road section when its physical dimensions are excessive/disproportional	V2I must provide effective warnings and appropriate alerts to the vehicles which do not meet the dimensions and weight requirements to cross a certain road fragment within an acceptable time frame
A vehicle within the V2I network operates outside the operative area	V2I must provide effective warnings and appropriate alerts to the vehicles when entering/exiting the system operative area within an acceptable time frame
V2I integrity level is violated	Effective countermeasures and detection tools must be applied in the system in order to preserve its operational state from external intrusions.
V2I causes or contributes to a near miss collision between multiple vehicles	
	V2I must provide effective warnings and appropriate alerts about potentially dangerous threats within an acceptable time frame

Table 16: Hazards and related constraints

To keep up with the holistic view of the system expected by the STAMP methodology, the next step is to provide a high-level control structure to allow the designers to be conscious of all the interactions between the system components and their role. Figure 40 shows the V2I control structure:

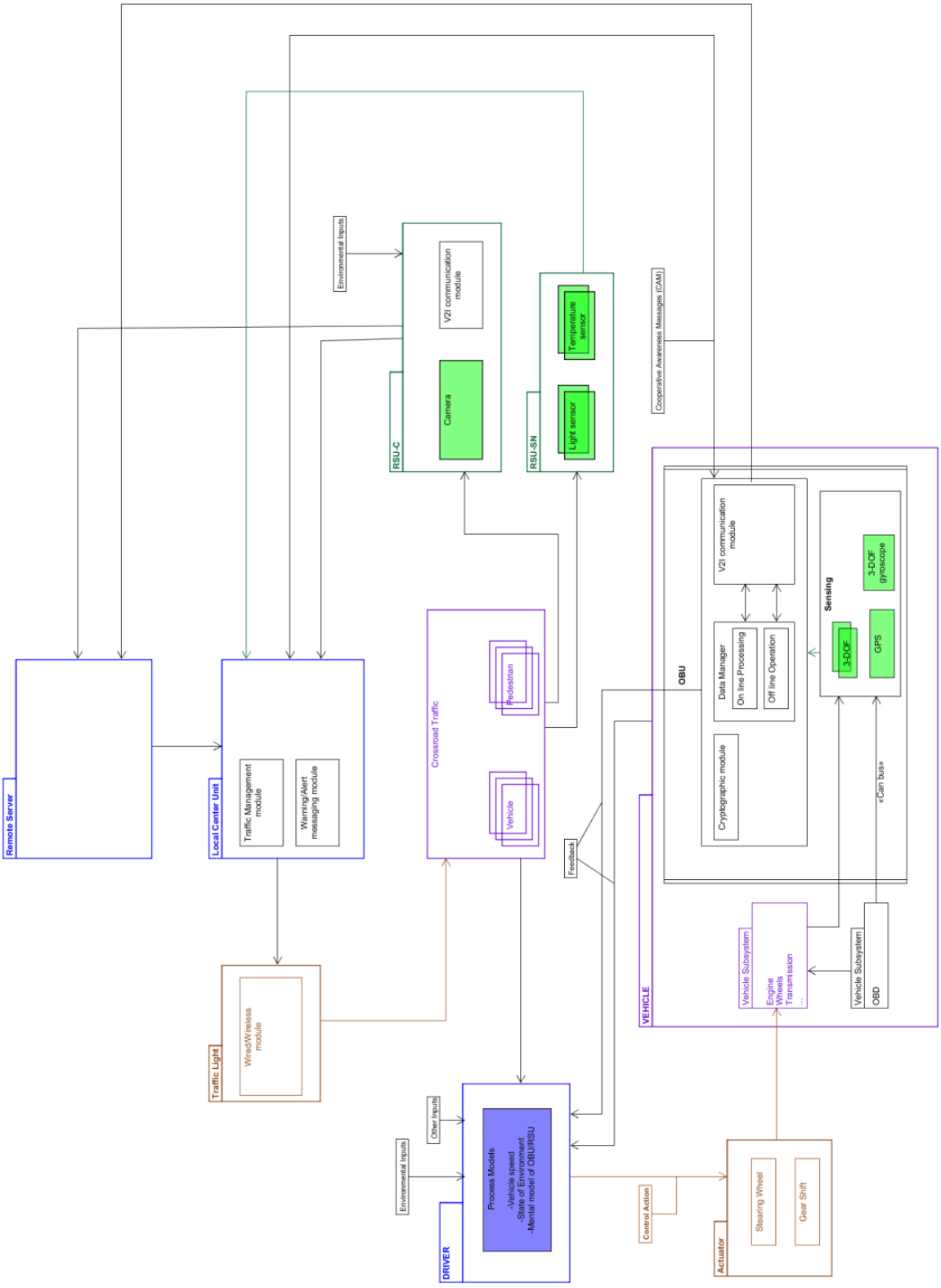

Figure 45 High level view of the system's control structure

Once the preliminary phase is completed, STPA step 1 provides the guidelines to identify unsafe control actions; for convenience, a general form of the reference table is used to keep track of all the items. The next Figure 41 shows an example of unsafe control actions related to the LCU component.

Local Central Unit

CONTROL ACTION	PROVIDED	NOT PROVIDED	WRONG TIMING/ORDER	APPLIED TOO LONG/ STOPPED TOO SOON
Red Light	Controller issues a red light command in the wrong lane Controller issues a red light command in a lane when a law enforcement vehicle or an ambulance is passing by Controller issues a red light command while the green light on the oppposite lane of the intersection is on	The red light command is not provided in the right combination with the other lanes. The red light command is not provided while a vehicle or an obstacle is stationary in the middle of the crossroad The red light command is not provided while a distant driver accelerates dangerously while green light is active on the opposite lane	Red light command is issued before the orange command. The command is not provided in the right combination with the pedestrian light commands.	The command is applied over the maximium treshold value *(TBM)* The command is applied for less than the minimum treshold value *(TBM)*
Green light	Controller issues a green light command while green light is active on the opposite direction of travel Controller issues a green light command while a vehicle or an obstacle is stationary in the middle of the crossroad	Command is not provided when there is free space in a lane to relieve traffic congestion Command is not provided when the crossroad is free	Green light command is issued after the orange command. The command is not provided in the right combination with the pedestrian light commands.	The command is applied over the maximium treshold value *(TBM)* The command is applied for less than the minimum treshold value *(TBM)*
Yellow light	Controller issues a yellow light command while a vehicle tries to cross the intersection at high speed The command is issued while the drivers speed requires a sudden brake and there is another vehicle behind Controller issues a yellow light command while a vehicle or an obstacle is stationary in the middle of the crossroad	Command is not provided between the green light command and the red light command	Yellow light command is issued after the red light command The command is not provided in the right combination with the pedestrian light commands.	The command is applied over the maximium treshold value *(TBM)* The command is applied for less than the minimum treshold value *(TBM)*
Pedestrian Green	The command is issued in contrast with the other crossroad lanes	The command is not provided in the right combination with the vehicles lanes	The command is issued after the orange command. The command is not provided in the right combination with the vehicles light commands.	The command is applied over the maximium treshold value (TBM) The command is applied for less than the minimum treshold value (TBM)
Pedestrian Red	The command is issued in contrast with the other crossroad lanes	The command is not provided in the right combination with the vehicles lanes	The command is issued before the orange command. The command is not provided in the right combination with the vehicles light commands.	The command is applied over the maximium treshold value (TBM) The command is applied for less than the minimum treshold value (TBM)
Pedestrian Yellow	The command is issued in contrast with the other crossroad lanes	Command is not provided between the green light command and the red light command	Yellow light command is issued after the red light command The command is not provided in the right combination with the vehicles light commands.	The command is applied over the maximium treshold value (TBM) The command is applied for less than the minimum treshold value (TBM) The command is stopped before the displayed timer for pedestrian elapses
Blinking	Controller issues a Blinking command during the rush-hour traffic	Command is not provided when there is an unsupported traffic configuration Command is not provided in case of the system needs to notify a malfunctioning status to the users Command is not provided while the system status is off	The command is applied with delay after the system has gone offline	The command is still applied when the system has come up back on-line The command is stopped while the system is still off-line
Break	The controller issues a break command while there are no real dangers or reasons to decelerate/stop the vehicl The command is issued while the vehicle is in the middle of a curve	The system doesn't notify a set of dangerous conditions ahead of the driver (i.e. ice on the road, accidents, obstacles on the road, ec...) (False negative)	The command is issued when the car runs at high speed before the slow down command The command is issued in rough weather conditions/ degraded road conditions before the slow down command The command is signaled with delay in presence of hazardous conditions	The command is still applied while in safe state The command is stoppped while in unsafe state
Slow Down	The controller issues a slow down command while there are no real dangers or reasons to decelerate in a high velocity area (False positive)	Command is not provided when a vehicle is transitioning to a lower speed area The system doesn't notify a set of dangerous conditions ahead of the driver (i.e. ice on the road, accidents, obstacles on the road, ec...) (False negative)	The command is issued when it is necessary to quickly free the area.	The command is still applied while in safe state The command is stoppped while in unsafe state
Video Warning	The system notifies the driver of unexistent warnings (False positive messages) The provided video warning is not adapted to a color blind driver The provided video warning is displayed in a non visible spot to the driver	The system doesn't notify a fault/alert to the driver in case of system malfunctions (False negative) The system doesn't notify the alert/warning messages to the driver issued by the LCU/RSV	Video warnings/alerts with lower priority are shown befor the ones with higher priority The video warnings are overlapped with other warning/alert messages	The video warning message is displayed for a brief interval making the alert/warning unintelligible The time between diffrent messages is below minimum treshold (TBM)
Sound Warning	The volume of the sound warning is above the maximum audio treshold *(TBM)* The volume of the sound warning is below the minimum audio treshold *(TBM)* The system notifies the driver of unexistent warnings (False positive messages)	The system doesn't notify a fault/alert to the driver in case of system malfunctions (False negative) The system doesn't notify the alert/warning messages to the driver issued by the LCU/RSV	The controller notifies the driver with a sound warning that overlaps with higher priority messages	The command is applied for an interval that is too short to be comprehnsible (below minimum treshold) *(TBM)*

Figure 46 Example of STPA worksheet

7.5.3 Comparison

First of all, the comparison involves the conceptual approach of the two methods. To perform such a comparison, the ISO 26262 and the STPA methods have been decomposed in logically subsequent and self-contained steps, trying to highlight similarities and differences. This comparison, focusing on the HRA according to ISO 26262 and STAMP/STPA is summarized in Table 17.

ISO 26262 - HRA	**STPA – Preliminary & Step 1**	**Comments**
Item definition. The system is described at a high level of abstraction identifying its boundaries and the main safety-related functions	System overview and description. Identification of the field of application	Both approaches start with an high-level system description. The ISO 26262 standard requires more formal approach.
HRA preparation: identification of the external conditions, the system operating modes and identification of a set of high-level hazards		STPA does not consider a formal step to define the operating modes and the external conditions that might affect hazardous events.
	Identification of an initial set o high-level accidents and hazards and their linkage Acquisition of the functional control diagram of the system to be analyzed as a whole	STPA requires a fine-grained analysis. HRA, on the other hands, concentrates on high-level hazards, deferring to subsequent phases a more detailed architectural and functional analysis.
Risk analysis. Identification of the relation between external conditions and the probability of occurrence, the severity of the consequences and the controllability of each hazard.	Identification of how each potential hazardous control action could occur	This steps concentrates on the hazard analysis. One method (HRA) is oriented on a statistical analysis and evaluation of the consequences, the other (STPA) is more focused on the control actions causing the hazards.
Determine the integrity level emerging from the analysis of each hazard		STPA is based on a qualitative analysis, while HRA uses a semi-quantitative approach such as risk graph or risk matrix.
Define the safety functions and safe states associated to each safety critical hazard		

Table 17Comparison of ISO 26262 and STPA Step 1

Until this point the two methodologies, though from different perspectives are rather similar and can be compared as summarized above. The main difference is that HRA provide a quantitative integrity requirement associated to the hazards, while STPA only lists a set of hazards. In summary thus, the output of both methods consists in:

- A set of system-level hazards, for both approaches
- A set of high-level requirements (safety functions) and their integrity requirements (HRA only)

These are the starting point for the subsequent phases. Concerning such phases, shortly summarized in Table 18, it must be noted that STPA itself offers the possibility to be pushed further (Step 2) by increasing the level of detail of the system description and including subsystem and component-level details. According to the ISO 26262 approach, on the other hand, subsequent steps require different analysis tools such as FMEA/FMEDA (Failure Mode and Effect Analysis), FTA (Fault Tree Analysis), SCA (Software Criticality Analysis) and so on.

ISO 26262 –Next steps	**STPA – Step 2**	**Comments**
Detailed analysis such as FMEA/FMEDA, FTA, SCA requiring a more detailed view of the system architecture	Identification of potential hazards caused by inadequate control commands	
	Define controls and countermeasures if they do not exist or evaluate existing ones	

Table 18 Comparison of ISO 26262 and STPA - Step 2

The comparison of the two approaches, limited to STPA - Step 1, concerns thus the identification of hazards only. It is worth noting that hazards identified by the HRA approach are item-specific, while those elicited using STPA are expressed at the system-level. For this reason, hazards related to the A-TLS have not been analyzed during HRA, since it is not part of the system being designed, while it is included in the STPA analysis since the system is considered as a whole. Table 19 summarizes the hazard counts.

	Subsystem-Level		**System-Level**
HRA	OBU	10	N/A
	RSU-SN	4	
	RSU-C	8	
	L-CU	13	

	A-TLS	Not considered	
STPA	N/A	N/A	11

Table 19 Number of hazards