PADERBORN UNIVERSITY
The University for the Information Society

Faculty for Computer Science, Electrical Engineering and Mathematics
Heinz Nixdorf Institute and Department of Computer Science
Research Group Software Engineering

Safety Requirements Engineering for Early SIL Tailoring

PhD Thesis

submitted in partial fulfillment
of the requirements for the degree of
“Doktor der Naturwissenschaften (Dr. rer. nat.)”

by
Markus Fockel

Supervised by:
Prof. Dr. Eric Bodden

Paderborn, December 2018

ABSTRACT

The high degree of innovation in mechatronic systems domains leads to so-called cyber-physical systems (CPS) that are characterized by their complex functionality and communication with their surroundings. The safety-criticality of such systems is categorized into so-called safety integrity levels (SIL) that are defined by safety standards like ISO 26262. A determined SIL not only describes the risk of potential harm, it also dictates the required degree of rigor to be applied in the development of a system to prevent hazards or mitigate their consequences. A high SIL requires the application of safety measures with a high degree of rigor in all phases of development, and, thus, implies high safety effort. SIL tailoring is a means to reduce safety effort by assigning subsystems with a lower SIL if they are separated from more critical subsystems or fulfill redundant safety requirements.

To plan the required safety effort, SIL tailoring possibilities should be identified as early as possible, i.e., already during requirements analysis. Due to the complexity of CPS, it is difficult to elicit and document safety requirements that establish valid SIL tailoring possibilities. The validity of SIL tailorings has to be analyzed based on failure propagation paths through the system. However, it is error-prone and time-consuming to determine failure propagation paths based on requirements that are specified in informal language and undergo frequent changes. In addition, the validity of applied SIL tailorings has to be assured by arguments compiled in a so-called safety case. If requirements and applied SIL tailorings undergo frequent changes, the safety case easily turns inconsistent and requires high maintenance effort. Existing research approaches work on the design level rather than requirements, and target only parts of these challenges. No existing approach provides a seamless SIL tailoring process on the requirements level.

The contribution of this thesis is a systematic, tool-supported SIL tailoring process applied in safety requirements engineering that copes with the mentioned challenges. The process uses a model- and scenario-based formal requirements specification language, and provides a catalog of requirement patterns to support the specification of unambiguous and consistent safety requirements. Based on these formal requirements, automatically, failure propagation models are generated and SILs allocated to subsystems. This minimizes the safety analysis effort to a review task. Finally, a safety case with arguments for the validity of the applied SIL tailorings is automatically derived from the generated analysis results to automate its maintenance for consistency.

The SIL tailoring process was evaluated by a case study with two cases from the automotive domain. The results show that the process is applicable to realistic examples and reduces the effort for the safety manager.

ZUSAMMENFASSUNG

Der hohe Grad an Innovation in mechatronischen Systemen führt zu sogenannten Cyber-Physical Systems (CPS). Diese sind durch komplexe Funktionalität und Kommunikation mit ihrer Umgebung charakterisiert. Wie sicherheitskritisch solche Systeme (bzgl. Safety) sind, wird durch sogenannte Sicherheits-Integritätslevel (SIL) kategorisiert, die durch Normen wie der ISO 26262 definiert werden. Ein bestimmter SIL beschreibt nicht nur die Höhe des Gefährdungsrisikos, sondern diktiert auch den erforderlichen Grad an Sorgfalt bei der Entwicklung des Systems, um Gefahren zu verhindern oder abzumildern. Ein hoher SIL erfordert die Anwendung von Safety-Maßnahmen mit einem hohen Sorgfaltsgrad in allen Phasen der Entwicklung und impliziert daher einen hohen Safety-Aufwand. SIL-Tailoring ist ein Mittel um den Safety-Aufwand zu reduzieren, indem man Subsystemen geringere SILs zuordnet, falls sie von kritischeren Subsystemen getrennt sind oder redundante Safety-Anforderungen erfüllen.

Um den nötigen Safety-Aufwand zu planen, sollten Möglichkeiten für SIL-Tailoring so früh wie möglich identifiziert werden – d.h. bereits in der Anforderungsanalyse. Durch die Komplexität von CPS, ist es schwierig Safety-Anforderungen zu ermitteln und zu dokumentieren, die valide Möglichkeiten für SIL-Tailoring eröffnen. Die Validität von SIL-Tailorings wird bestimmt, indem die Pfade analysiert werden, auf denen Fehler durch das System propagieren. Allerdings ist es fehleranfällig und zeitaufwändig solche Pfade basierend auf Anforderungen zu identifizieren, die informell beschrieben sind und häufigen Änderungen unterliegen. Zusätzlich muss die Validität von angewendeten SIL-Tailorings durch Argumente begründet werden, die im sogenannten Safety Case zusammengestellt werden. Wenn Anforderungen und SIL-Tailorings häufigen Änderungen unterliegen, wird der Safety Case leicht inkonsistent und erfordert hohen Pflegeaufwand. Existierende Forschungsansätze arbeiten auf dem Design-Level statt auf Anforderungen und adressieren nur Teile der genannten Herausforderungen. Kein existierender Ansatz stellt einen nahtlosen SIL-Tailoring-Prozess auf Anforderungsebene bereit.

Der Beitrag dieser Dissertation ist ein systematischer, werkzeugunterstützter SIL-Tailoring-Prozess, der im Safety Requirements Engineering angewendet wird und die genannten Herausforderungen bewältigt. Der Prozess nutzt eine modell- und szenariobasierte, formale Sprache zur Anforderungsspezifikation und stellt einen Katalog von Anforderungsmustern bereit. Dies unterstützt die Spezifikation von eindeutigen und konsistenten Safety-Anforderungen. Basierend auf diesen formalen Anforderungen werden Fehlerpropagierungsmodelle automatisch generiert und Subsystemen automatisch SILs zugeordnet. Das minimiert den Sicherheitsanalyseaufwand auf eine Review-Aufgabe. Schließlich wird aus den generierten Analyseergebnissen automatisch ein Safety Case mit

Argumenten für die Validität von angewendeten SIL-Tailorings abgeleitet. Dadurch wird die Safety-Case-Pflege automatisiert.

Der SIL-Tailoring-Prozess wurde durch eine Fallstudie mit zwei Fällen aus der Automobilbranche evaluiert. Die Ergebnisse zeigen, dass der Prozess auf realistische Beispiele anwendbar ist und den Aufwand des Safety-Managers reduziert.

DANKSAGUNG

Es gibt viele Menschen, die auf unterschiedlichste Weise zu dieser Arbeit beigetragen haben. An dieser Stelle möchte ich all diesen Menschen danken.

Zuerst möchte ich mich bei Wilhelm Schäfer bedanken, der mich dabei unterstützte, den Weg zur Promotion zu beschreiten. Eric Bodden möchte ich dafür danken, dass er bereitwillig übernahm und mich auf der Zielgeraden dieses Weges begleitete. Ich danke Eric Bodden und Joel Greenyer für die Anfertigung ihrer Gutachten und auch den weiteren Mitgliedern meiner Prüfungskommission, Roman Dumitrescu, Matthias Meyer und Stefan Sauer für ihre Bereitschaft, an meiner Verteidigung teilzunehmen.

Weiterhin danke ich allen ehemaligen und aktuellen Kolleginnen und Kollegen am Fraunhofer IEM und in der Fachgruppe Softwaretechnik. Hervorheben möchte ich dabei Jörg Holtmann, Jens Frieben, David Schubert, David Schmelter und Thorsten Koch. Jörg danke ich für die Zusammenarbeit in der Forschung und die Erhaltung der Kaffeepausentradition. Darüber hinaus möchte ich Jörg, Jens und David Schubert für die jeweils geteilte Zeit im gemeinsamen Büro und die erheiternden Momente danken. Außerdem danke ich Jörg, David Schmelter und Thorsten für die zahlreichen Diskussionen in unserer RE-Expertenrunde. Matthias Meyer danke ich dafür, dass er als Abteilungsleiter immer hinter mir stand. Für das Lösen von organisatorischen und technischen Herausforderungen möchte ich mich bei der Verwaltung am Fraunhofer IEM und in der Fachgruppe bedanken, insbesondere bei Vera Meyer und Jürgen Maniera.

Ruslan Bernijazov, Patrick Enste, Frederik Knust, Conrad Neumann und Roman Trentinaglia danke ich für die Zusammenarbeit im Rahmen ihrer SHK-Tätigkeit, Abschlussarbeit oder Projektgruppe.

Schließlich danke ich insbesondere auch meiner gesamten Familie. Meine Eltern haben mich stets auf dem Bildungsweg zu diesem Ziel gefördert und unterstützt. Annemarie danke ich für ihre Liebe und ihr Verständnis für all die kostbare Zeit, die ich in diese Dissertation investiert habe.

CONTENTS

1

INTRODUCTION

Systems that were purely mechanical in the past, have evolved into mechatronic systems that are comprised of mechanical, electric/electronic, and software parts. In the automotive domain, today 90 percent of innovation is realized by electronics and software [Inv14]. The number of *electronic control units* (ECUs) used in cars rapidly increased since 1985 and reached 70 to 100 ECUs per car in 2009 [Cha09; EJ09]. This high degree of innovation leads to so-called *cyber-physical systems* [Aca11] that are characterized by their complex functionality and communication with other systems and surroundings. Exemplary for this development is the increasing complexity of *advanced driver assistance systems* (ADAS) that are realized by connecting ECUs which previously had separate functionality [WHL^{+}16]. These ADAS make use of huge amounts of sensory data and Vehicle-to-X communication [Ins12] to automatically take decisions and control brakes or steering. Such systems are not only complex but also highly safety-critical. If an ADAS unintentionally steers or fails to brake when necessary, people are in danger. Especially, if drivers fully trust and rely on the correct functioning of the systems (like Tesla's Autopilot [Tes16]).

To make safe and profound decisions, ADAS analyze the current driving situation based on the readings of several sensors and cameras. This integration of data, referred to as sensor fusion, requires a lot processing power to gain results in real-time. Thus, there is a trend towards central, powerful ECUs as the brain for decision-making [Bai15; Kus16]. These central ECUs realize functions with different levels of safety-criticality and, by that, form so-called *mixed-criticality* systems. For example, in an emergency braking system, the function activating the brakes is more safety-critical than the function that warns the following traffic by activating the hazard lights. A failure of the hazard lights shall not cause a failure of the brakes. So it is important that functions of different safety-criticality do not interfere with one another.

A function's safety-criticality stems from the hazards that are caused by its possible failures. Each hazard is categorized based on its severity and possibility of occurrence. The safety standard for automotive systems ISO 26262 [Int11b] categorizes safety-criticality in so-called *Automotive Safety Integrity Levels* (ASIL) ranging from ASIL D to ASIL A (in decreasing order) and QM for non-safety-critical elements. Safety standards for other domains have similar safety integrity levels (SIL) to categorize safety-criticality (e.g., DAL for aerospace systems [SAE10] or AgPL for agricultural vehicles [Int10b]). Accordingly, the concepts presented in this thesis are transferable although the automotive domain is used as a running example.

The safety-criticality (i.e., the determined SIL) not only describes the risk of potential harm to people. It also dictates the required degree of rigor to be applied in the development of a system to prevent the hazards or mitigate their consequences [WC12]. Based on the SIL, the safety standards prescribe what safety measures have to be applied. These are safety mechanisms to be integrated into the system (e.g., watchdogs or redundant components) and safety activities to be executed in different development phases (e.g., performing an FMEA [Int06a] or work product reviews). It is the responsibility of the *safety manager* [Int11a] to make sure these safety measures are fulfilled to the required degree.

The development of mechatronic systems like ADAS requires a strong interplay of different engineering disciplines. The systems consist of mechanics, electrics/electronics, and software. To consolidate the disciplines, development processes based on the V model, like VDI 2206 [VDI04] for mechatronic systems in general and Automotive SPICE [Aut10] for embedded systems in the automotive industry, are followed. Figure 1.1 depicts the left side of a V model based on those standards. Development begins with the discipline-spanning system-level phases system requirements analysis and system architectural design. Afterwards, development continues in discipline-specific sub-processes for mechanics, electrics/electronics (E/E), and software (SW). For example, in the software process again phases for requirements analysis (SW requirements analysis) and design (SW design) are executed, focusing on the system's software parts. The right side of the V-model follows with testing and integration phases.

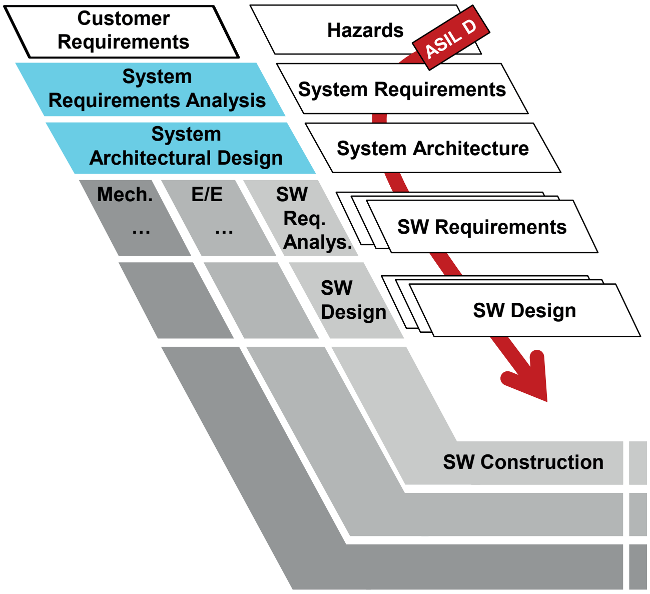


Figure 1.1: Interdisciplinary development process with work products and ASIL propagation

An ASIL that is determined for a hazard propagates through all following work products (cf. the red arrow in Figure 1.1). It propagates to the system requirements containing safety requirements to prevent the hazard, on to the subsystems of the system architecture that satisfy those safety requirements, on to the discipline-specific safety requirements (e.g., the SW requirements) that refine the system-level safety requirements, on to the discipline-specific designs (e.g., SW design), etc. Thus, a high ASIL requires the application of safety measures with a high degree of rigor in all phases of development. A high ASIL accordingly implies a high safety effort. Additionally, if system failures that would cause a hazard, are detected late in the process, this causes expensive iterations to repeat safety measures or apply additional ones. Hence, the safety manager's goal is to find possible failures and plan safety measures as early as possible. Due to the complexity and mixed-criticality of ADAS and cyber-physical systems in general, reaching this goal is challenging, and may depend on the expertise and experience of the safety manager.

As an example of a complex, mixed-criticality system, we use an ADAS from the automotive domain that is introduced in the next section.

1.1 Advanced Driver Assistance System EBEAS

Autonomous Emergency Braking systems automatically detect an imminent crash with a front vehicle, warn the driver, and also automatically brake to prevent or mitigate the crash. These systems are part of the Euro NCAP test for vehicle safety since 2014 [Eur14].

Autonomous Emergency Steering systems automatically support the driver's steering to keep the vehicle under control when the driver decides to evade an obstacle. Examples of such systems have been presented by automotive suppliers like Continental [Tut10] and ZF [ZF 16]. These assistance systems have not yet made it into series production.

The *Emergency Braking & Evasion Assistance System* (EBEAS) combines the two systems and automatically decides to brake or evade based on coordination with the surrounding vehicles via Vehicle-to-Vehicle (V2V) communication [HFK$^+$16].

Figure 1.2 sketches the system's application scenario. When the leading vehicle suddenly performs an emergency braking because of obstacles on the road, it automatically warns the following vehicle ego via V2V communication. ego then has to decide to either brake as well or evade to the neighboring lane. This decision, on the one hand, is based on the vehicle's sensor information and, on the other hand, on V2V communication. ego communicates with the nearby vehicles following and overtaking: If following can safely brake and come to a standstill without a crash, ego will decide to brake. Otherwise, ego will ask overtaking whether it can safely brake such that ego can evade to its lane to solve the situation. If evading is also no option, ego will perform an emergency braking and warn following. This will not prevent a crash but mitigate the consequences and is assumed to be less severe than evading and crashing into overtaking.

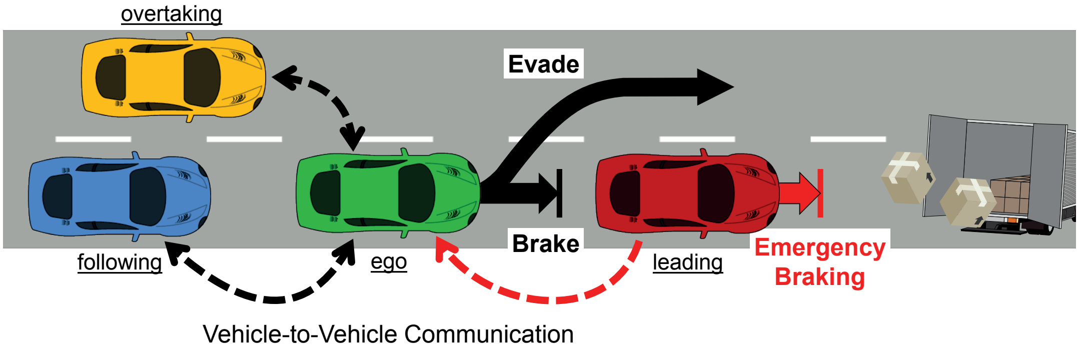


Figure 1.2: EBEAS scenario

The basic principle underlying this system is to first detect and analyze the critical situation, then make a decision to brake or evade, and finally execute that decision. There are existing, series production systems that gather information that can be used to detect a critical situation or are able to control braking or steering. Figure 1.3 shows the EBEAS and the existing systems that it has to interact with. The Adaptive Cruise Control is connected to radar sensors in the front of the vehicle and can detect the distance and speed of the leading vehicle. The Electronic Stability Control uses wheel speed sensors to determine the ego vehicle's speed and controls the hydraulic braking system. The Vehicle2X Communication uses antennas to exchange information with other vehicles (or infrastructure). The Active Front Steering can electro-mechanically actuate the steering column for the EBEAS (or for automatic parking). The EBEAS is also connected to further systems to detect the following and overtaking vehicle, to detect lane markings, to determine the ego vehicle's driving lane, and to control the turn signals and brake lights.

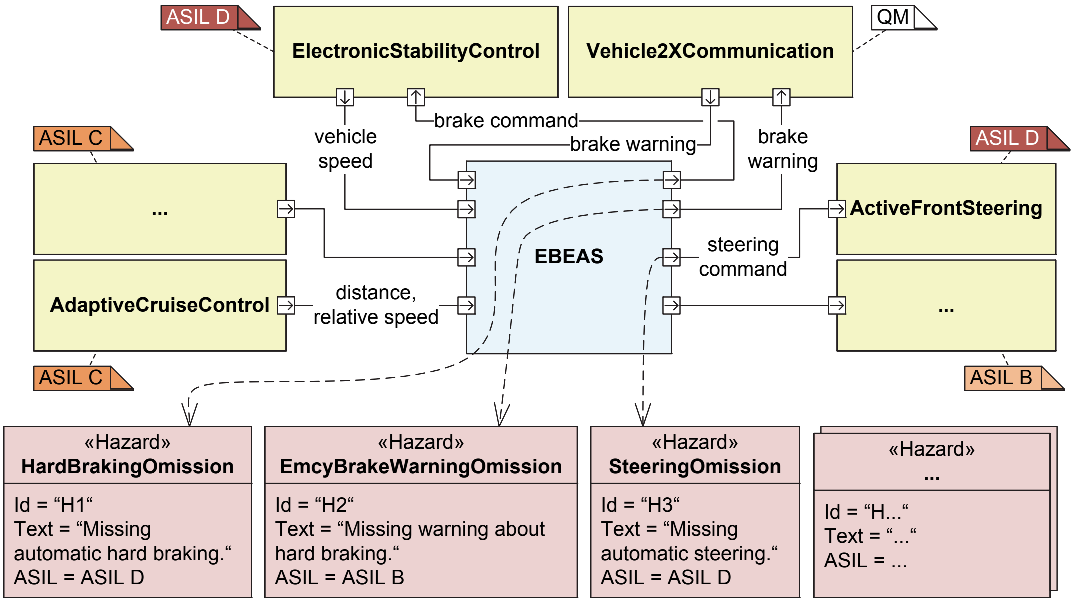


Figure 1.3: EBEAS ECU, interacting systems, and hazards

This shows that an ADAS like this is a complex system comprised of the interaction of a number of ECUs, sensors, and actuators. The different information gathered from the sensors and V2V messages has to be combined (sensor fusion) to get a thorough understanding of the situation.

The system is highly safety-critical (ASIL D), as it interferes with brakes and steering. However, it also is a mixed-criticality system because the interacting systems like the Adaptive Cruise Control or the Vehicle2X Communication contain preexisting functionality with lower criticality. In Figure 1.3 the surrounding preexisting systems are annotated with predetermined ASIL values. These stem from hazard analyses based on their prior usage without an EBEAS [Cot13; Cro15]. The Electronic Stability Control and the Active Front Steering are ASIL D systems as they control brakes and steering. The Adaptive Cruise Control and some other existing systems that do not directly interfere with brakes or steering are ranked ASIL C. The Vehicle2X Communication is used to display warnings about upcoming road hazards via the infotainment system to the driver. In its prior usage, it does not interfere with vehicle control systems. Thus, it is not considered safety-critical and assigned with QM.

Some hazards that are caused by possible failures of the EBEAS are shown in the bottom of Figure 1.3. The first hazard H1 is the missing automatic hard braking. It occurs if the EBEAS fails to decide to brake when it should, such that the Electronic Stability Control is not signaled to activate the braking maneuver. Hazard H3 analogously describes the missing automatic steering for an evasion maneuver. Both of these hazards are assigned with the highest ASIL value of ASIL D, as they regard the highly safety-critical brakes and steering. Hazard H2 is the missing warning of the following vehicle about a hard braking once the EBEAS decided to perform an emergency braking maneuver. As this is only a warning to a following vehicle that still could detect the critical situation on its own, this hazard is assigned with ASIL B only.

The EBEAS has to be developed with the degree of rigor of the highest ASIL that is assigned to one of its hazards. Thus, the EBEAS is an ASIL D system. However, as there is at least one hazard (H2) with a lower ASIL value, parts of the EBEAS (e.g., the communication functionality) might possibly be developed with less rigor. So, the EBEAS is a mixed-criticality system in itself.

To develop a safe EBEAS with reasonable time and effort, the safety manager has two goals: to early on make sure

- that the predetermined ASIL values assigned to the surrounding systems do not need to be increased, and
- that as much as possible of the EBEAS itself can be developed in accordance with a low ASIL value.

1.2 Problem Statement

A high ASIL requires the application of safety measures with a high degree of rigor. A high ASIL accordingly implies a high safety effort. However, less critical parts of the system should not require the same high safety effort. If parts of the safety requirements (and the subsystems satisfying them) can be assigned with a lower ASIL, this could reduce the safety effort because less rigorous safety measures are required [APW$^+$14]. To reduce the number of high ASIL requirements and subsystems, the safety manager can apply so-called *ASIL tailoring* [Int11h]. It is a means to

- *separate* subsystems (incl. their safety requirements) with different ASILs and to
- *decompose* safety requirements into redundant safety requirements with lower ASILs.

For example, if the non-safety-critical infotainment system of the vehicle is separated from the safety-critical EBEAS, it may keep its low ASIL (i.e., QM). If the sensing of the distance to the leading vehicle can be decomposed onto two redundant and independent sensors, they both may be assigned with a lower ASIL than a single sensor.

The resulting system has to stay safe although the planned safety effort is reduced. Accordingly, the validity of each applied tailoring has to be assured by safety analysis (e.g., FTA or FMEA): Separated subsystems may not cause more critical subsystems to fail, and decomposed subsystems may not fail for the same reason. This can be argued by analyzing the propagation paths of the identified possible failures.

The ASIL tailoring rules are explicitly defined in ISO 26262 for the automotive domain [Int11h]. The underlying principles are transferable to other mechatronic systems domains (e.g., aerospace [SAE10]), and always the validity of each tailoring has to be assured by safety analysis.

The separation and decomposition of safety-critical functionality for ASIL tailoring can be applied in different phases of development. The earlier in the development process it is applied, the less effort for safety measures is required in following development phases (cf. ASIL propagation in Figure 1.1). Existing research approaches are applied in the phases of system architectural design or software design [APW$^+$14; MAL$^+$12]. They calculate an ASIL allocation to subsystems of a technical architecture with known failure propagation and safety mechanisms already in place. They assume that ASIL tailoring measures (separation and decomposition) have already been applied – both on requirements and architectural level.

However, ASIL tailoring is not a design task in the responsibility of system designers [She96] or software architects. It is the responsibility of the safety manager and should be performed during requirements engineering on functional, technical, and software/hardware safety requirements [Int11h; WC12]. The reason is that ASIL tailoring is not about reducing the probability of random (hardware) failures, it considers deterministic failures that are reduced by systematic changes. The solution space for such changes is higher on the requirements level. For example, ASIL tailoring by decomposition does not necessarily require introducing additional, redundant subsystems. It can also be realized by decomposing a safety requirement into requirements that are satisfied by independent subsystems that exist in the system anyways: The EBEAS requirement to detect the hard braking of the leading vehicle can be satisfied by the Vehicle2X communication receiving a warning message and by the adaptive cruise control sensing the sudden decrease of distance. The Vehicle2X communication and the adaptive cruise control are both part of the vehicle anyways. Hence, this requirements decomposition avoids the effort for safety measures like adding additional sensors or antennas.

The safety manager's goal is to find possible failures and plan safety measures as early as possible to reduce expensive development iterations and achieve foreseeable safety effort. A high ASIL requires high effort for safety measures because it propagates through all development phases (cf. Figure 1.1). ASIL tailoring is a means to reduce the safety effort in following development phases. Therefore, the safety manager's goal is to apply ASIL tailoring as early as possible, i.e., already on the requirements level. However, this goal is hindered by the following challenges.

C1: Complexity of Mixed-Criticality Systems

ADAS are complex systems that have to realize many requirements. For example, the EBEAS (s. Section 1.1) makes use of huge amounts of sensory data and vehicle-to-X communication to automatically take decisions and control brakes and steering. The EBEAS directly communicates with about eight other ECUs, which are again connected to further ECUs, sensors, and actuators that comprise the EBEAS functionality. In addition, complexity also stems from the different driving situations, number and position of other vehicles, environmental conditions, etc. that have to be considered. Moreover, ADAS are developed interdisciplinary. The requirements have to be realized by mechanics, hardware, and software. This is a source of complexity which introduces dependent failures, crossing the disciplines (e.g., a software failure leading to a hardware failure), which are difficult to identify. Altogether, the complexity of ADAS puts a challenge on thoroughly identifying failures and their propagation through the system [HWK$^+$14; ZRH16]. However, the identified failures are a prerequisite for planning safety measures and applying ASIL tailoring.

ADAS are mixed-criticality systems that span multiple subsystems of a vehicle. Some ECUs realize functions for different ADAS with different levels of safety-criticality. The ECUs that are connected to the EBEAS ECU realize driver assistance systems, like the adaptive cruise control, with their own safety-criticality levels (cf. predetermined ASIL values in Figure 1.3) which differ from the safety-criticality of the EBEAS. The EBEAS ECU contains subfunctions of different criticality (cf. different ASIL values of the hazards in Figure 1.3) and, thus, is a mixed-criticality system in itself. Furthermore, there is a trend towards central, powerful ECUs that provide information in real-time to ADAS of different criticality and, thus, form mixed-criticality systems. In addition, the centralization onto mixed-criticality ECUs is inevitable because of the limited physical space for ECUs in a vehicle. The mixed-criticality requires the application of ASIL tailoring to ensure the safety of the overall vehicle by separating subsystems of lower safety-criticality and decreasing the number and complexity of highly safety-critical subsystems.

To cope with complexity of ADAS and cyber-physical systems in general, systems engineering methods [Int08; Int15] are used that deal with interdisciplinary development, especially in the early system requirements analysis and system architectural design phases (cf. Figure 1.1). Unfortunately, the processes of systems engineering and safety engineering are not well-integrated. They use different vocabulary, paradigms, and tools [HWK$^+$14]. Moreover, requirements engineering is considered to be a part of systems engineering, but hazard analysis and other safety analyses are considered to be a part of safety engineering [HWK$^+$14]. This bad integration hinders identification of failure propagation paths and ASIL tailoring possibilities on the requirements level.

The complexity and mixed-criticality of systems also infers complex, mixed-criticality requirements specifications which are typically only specified in informal language. Informal language is prone to ambiguity, incompleteness, and inconsistency. Together with the bad integration of systems and safety engineering processes, this leads to error-prone and incomplete safety analysis and ASIL tailoring results. What the safety manager needs is "having a requirements specification in a form that is understandable, amenable to analysis, and precise enough" [HWK$^+$14].

C2: Safety Requirements Engineering Dilemma

To apply valid ASIL tailoring, the possible system failures and their propagation paths through the system need to be known. Unfortunately, there is a problem known as the *safety requirements engineering dilemma* [Ber98]. It states that a system's possible failures and resulting hazards can best be found late in the development process, but are ideally already known during requirements engineering. In the later phases of the development process most system details are known (i.e., during SW construction or the testing and integration phases). So it is easier to identify possible failures. However, changing the system to prevent or mitigate a failure, then causes expensive development iterations (e.g., changing system and SW requirements, changing SW/HW design, changing code and electronics). So, failures ideally would be found early, such that the safety manager can plan safety measures, ASIL tailoring, and resulting safety effort from the beginning and avoid iterations.

In the automotive domain, a functional architecture of the system is used to identify failure propagation paths through the system early [MFH15]. In the avionics domain, a safety analysis called Functional Hazard Analysis [SSK14] is applied, which "should be accomplished as early as possible in the systems engineering process" [Dep12]. Such safety analyses are conducted manually. Thus, for complex systems, they are prone to errors, based on experienced expert's knowledge, and time-consuming. Additionally, on the earlier requirements level, safety analyses are prone to incompleteness. Whenever new information about failures is gained, the safety analyses have to be repeated. Therefore, in practice, safety analyses are currently not performed on the requirements level, but rather on later, more stable work products.

To reduce the manual effort, existing research approaches conduct automated safety analyses on the technical system architecture [CJL$^+$08; Dor14]. However, automated safety analysis on a requirements level, especially for ASIL tailoring on that level, has not yet been considered. In general, safety analysis automation is seen as an open challenge [HWK$^+$14].

The validity of applied ASIL tailoring measures has to be assured by safety analysis. As safety analysis is not conducted on the requirements level in practice, also ASIL tailoring is not applied on the requirements level. As existing research approaches conduct automated safety analyses on technical architectures only, also existing ASIL tailoring approaches work on technical architectures only [APW$^+$14; MAL$^+$12]. However, the safety manager needs support in *early* safety analysis and ASIL tailoring on the requirements level.

C3: Safety Case Construction and Maintenance

Safety standards like the ISO 26262 require a so-called *safety case* [Int11c], which provides the "argument that the safety requirements for an item are complete and satisfied by evidence compiled from work products of the safety activities during development" [Int11a]. The safety manager has to argue the validity of each applied ASIL tailoring in the safety case. For the EBEAS, evidence could be safety analysis results that show that the subsystems with different assigned ASILs are non-interfering and that decomposed subsystems do not fail for the same reason.

As safety analyses are time-consuming and error-prone in early development stages (cf. previous challenges), ASIL tailoring is applied on technical architectures rather than functional models of the requirements analysis phase. Consequently, safety cases are not built in that phase either. Building a safety case with arguments for early ASIL tailoring requires traceability throughout the whole development process (from the initial hazards to the safety requirements, to introduced safety mechanisms, to safety analysis results). The industry already identified the need to start building the safety case from the beginning of the development process, but still misses supporting methods and tools [ZRH16]. Reverse engineering the safety case in the end of development is a manual, tedious, and error-prone task that may identify missing safety measures and, thus, cause expensive process iterations. In research, approaches exist that describe ways to document a safety case [Kel98; Ass18] and patterns to apply in a safety case [KM97; DP13]. However, there is no approach that automatically constructs and maintains the safety case according to an applied ASIL tailoring. In fact, safety case automation was identified as open challenge

[HWK+14]. Altogether, the safety manager needs automated support in safety case maintenance, including ASIL tailoring arguments.

1.3 Overview of the Solution

In order to tackle the challenges identified in the previous section, we propose a systematic, tool-supported ASIL tailoring process working on *functional safety requirements* [Int11d] in the system requirements analysis phase. The tailoring process is integrated into a systems engineering method [Int08; Int15] that deals with the interdisciplinary development of complex systems like ADAS and cyber-physical systems in general.

To cope with complexity, abstraction from details is a common approach. Therefore, development processes like Automotive SPICE [Aut10] and safety standards like ISO 26262 [Int11e] include a discipline-spanning system architecture (cf. Figure 1.1) prior to discipline-specific architectures. According to ISO 26262, the system architecture shall be modular, have adequate level of granularity, be simple, and avoid unnecessary complexity [Int11e]. Because of the increasing complexity of ADAS and cyber-physical systems in general, a further abstraction level is proposed to focus on a system's functionality prior to its technical realization [LW10][VEF+12][EAS13][GRS14, Sect. 4.1]. This *function hierarchy* is used to bridge the gap between requirements and technical system architecture. Pahl et al. describe the function hierarchy as a means to decompose a system's complex functionality into subfunctions, to focus on the essential, to negotiate with stakeholders, and to come up with possible structures of the system architecture [PBF+06].

In order to tackle the challenge of complexity of mixed-criticality systems (C1), our process uses such a functional abstraction of the system under development (SUD) in form of a function hierarchy. It is used to structure and decompose the requirements (esp. the safety requirements), as suggested by the requirements engineering standard ISO 29148 [Int11i]. In addition, the requirements are specified in a graphical, scenario-based modeling language to improve understandability of requirements specifications and cope with the complexity of different application scenarios (e.g., EBEAS driving scenarios with and without an overtaking vehicle). Furthermore, the language is formal, and thus, precise and amenable to analysis (e.g., for requirements validation [BGP13] and verification [GBC+13]).

To reduce the safety requirements engineering dilemma (C2), our tool-support automatically identifies failure propagation paths in the function hierarchy based on the requirements specified in the formal modeling language. Hence, the effort for repeated, manual safety analyses on the requirements level is minimized.

To support the construction and maintenance of a safety case from the beginning of development (C3), in our approach, a model-based safety argument is automatically generated and updated whenever an ASIL tailoring is applied on the function hierarchy.

Figure 1.4 shows the process steps and created/required work products of the ASIL tailoring process as a UML activity diagram [Obj15b]. The left of the figure is annotated with the corresponding early system phases of the V model (cf. Figure 1.1) in which the process steps are executed and the responsible roles.

In the early system phases, systems engineering methods are applied. Thus, the ASIL tailoring process needs to be integrated with such a method. As an example of an established model-based systems engineering method, we use *CONSENS* [GRS14, Sect. 4.1] in this

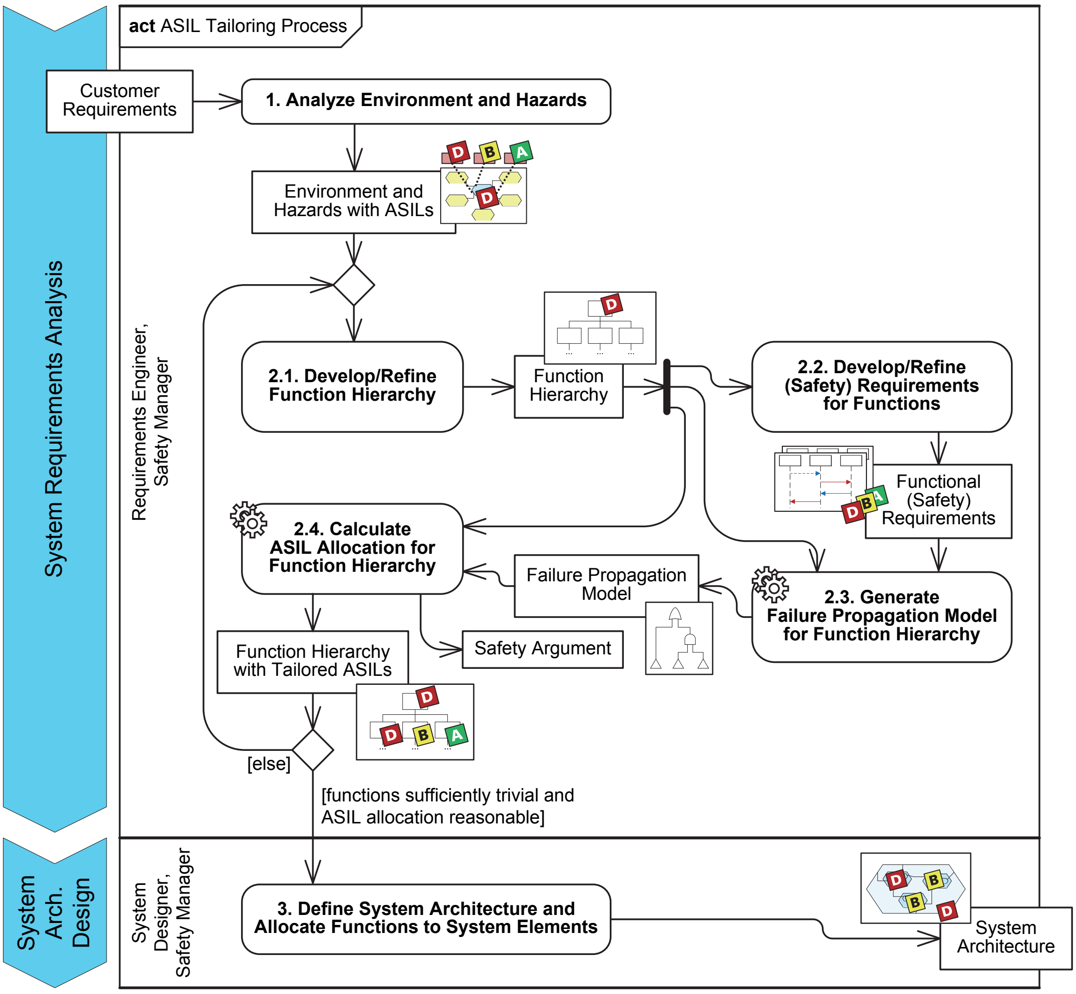

act ASIL Tailoring Process
System Requirements Analysis
Requirements Engineer, Safety Manager
Customer Requirements
1. Analyze Environment and Hazards
Environment and Hazards with ASILs
2.1. Develop/Refine Function Hierarchy
Function Hierarchy
2.2. Develop/Refine (Safety) Requirements for Functions
Functional (Safety) Requirements
2.3. Generate Failure Propagation Model for Function Hierarchy
Failure Propagation Model
2.4. Calculate ASIL Allocation for Function Hierarchy
Safety Argument
Function Hierarchy with Tailored ASILs
[else]
[functions sufficiently trivial and ASIL allocation reasonable]
System Arch. Design
System Designer, Safety Manager
3. Define System Architecture and Allocate Functions to System Elements
System Architecture
D
B
A

Figure 1.4: Overview of ASIL tailoring process steps and work products

thesis. In Step 1 of the ASIL tailoring process, the CONSENS environment model for the SUD is specified and analyzed for hazards. The environment model describes the interfaces of the SUD and its surrounding systems in the context that it has to interact with. In addition, we specify the hazards (incl. their ASIL value) that could be caused by failures occurring on the interfaces of the SUD. The hazards are identified through a hazard analysis on the SUD as a black-box and its context [Int11d]. Figure 1.3 sketches an example of an environment model with hazards for the EBEAS.

In Step 2.1 the CONSENS Function Hierarchy is developed. It describes a break down of the overall functionality of the SUD, represented by a root function, into sub-functions until technical realizations can be found. Each function describes how a sub-task of the overall system is fulfilled by producing outputs based on given inputs. We extend the CONSENS function hierarchy to specify this information flow through the functions as a structural basis for the formal requirements and failure propagation paths.

In Step 2.2, the requirements (esp. the safety requirements) of each function are specified using a formal modeling language. We use *Modal Sequence Diagrams* (MSDs) that are a formal, scenario-based requirements language based on UML sequence diagrams [HM08; Gre11; HFK^{+}16]. Requirements specified in this language can automatically be processed to identify failure propagation paths.

To find possible ASIL tailoring solutions and to verify that an applied tailoring meets its safety requirements, safety analyses have to be performed. As requirements are iteratively refined while developing the function hierarchy and applying ASIL tailoring, and because of the safety requirements engineering dilemma, the safety analyses should require only reasonable effort. Thus, in the automated Step 2.3, the safety manager generates a Failure Propagation Model based on fault trees to be used for safety analysis. The failure propagation model is generated from the structure of the function hierarchy and the behavior described in the formal functional (safety) requirements. The failure propagation model is used to calculate valid, tailored ASIL values for the functions of the function hierarchy in the automated Step 2.4. Also in Step 2.4, a safety argument is constructed that connects and traces the work products of the ASIL tailoring process to maintain a safety case in form of a model using the *Goal Structuring Notation* (GSN) [Ass18] as suggested by ISO 26262 [Int12b].

The steps 2.1 to 2.4 are iterated until the tailored ASILs of the functions meet the safety manager's expectations and the detail level of the functions is sufficiently trivial to be realized by technical elements of the System Architecture.

In the final Step 3, the system architecture is defined by the *system designer* [She96] (as CONSENS active structure) and the functions are allocated to their realizing system elements. Afterward, the functional safety requirements of the functions are refined into *technical safety requirements* [Int11e] that have to be satisfied by the system elements that the functions have been allocated to. Following, safety analyses on the system architecture have to be performed to show that the ASIL tailoring on functional level remains valid on the system level.

1.4 Thesis Structure

This thesis is structured as follows. Chapter 2 provides the foundations for the following chapters. Chapter 3 describes the ASIL tailoring process in detail. In Chapter 4, the language and method for specifying formal functional (safety) requirements and the integration with natural language requirements are explained. Chapter 5 goes into detail on the generation of the failure propagation model and its usage for ASIL allocation to functions. Chapter 6 describes the construction of model-based safety arguments. The prototypical implementation and evaluation of the approach are explained in Chapter 7. Chapter 8 concludes the thesis. In addition, Appendix A contains details on the models used for the evaluation and Appendix B on my paper contributions.

2

FOUNDATIONS

This chapter introduces the terms, languages, and methods that are necessary to understand the concepts presented in this thesis. Section 2.1 introduces the development process for cyber-physical systems underlying the approach of this thesis. In Section 2.2, the automotive safety standard ISO 26262 is introduced. Section 2.3 gives on overview of other safety standards and their safety integrity levels. In Section 2.4, the basics of safety analysis using fault trees are explained. Section 2.5 describes the two ASIL tailoring measures defined by ISO 26262. The sections 2.6, 2.7, and 2.8 give an overview of the model-based systems engineering method CONSENS, Modal Sequence Diagrams, and the Goal Structuring Notation, respectively.

2.1 Development Process for Safe Cyber-Physical Systems

Cyber-physical systems [Aca11] are characterized by their complex functionality and communication with other systems and surroundings. They have evolved from mechatronic systems that are comprised of mechanical, electric/electronic, and software parts. The development of such complex systems requires the application of *systems engineering* as an interdisciplinary approach and means to enable their successful realization [Int15; Int08]. Figure 2.1 shows the development process for mechatronic systems of VDI 2206 [VDI04] following the V model. Especially in its phase of system design interdisciplinary systems engineering methods like CONSENS (cf. Section 2.6) are applied. Based on requirements a system architecture is designed that forms the common ground for development in discipline-specific design. The discipline-specific design is split up into the disciplines of mechanical engineering for mechanical system parts, electrical engineering for electric/electronic system parts, and information technology for software parts. Afterward, system-level test and integration is done in the phase of system integration resulting in the final product.

The complexity of cyber-physical systems also implies complexity of their requirements. Hence, their development requires the application of thorough *requirements engineering* [Int11i; IG14]: “Requirements engineering is concerned with discovering, eliciting, developing, analyzing, determining verification methods, validating, communicating, documenting, and managing requirements” [Int11i]. Automotive SPICE [Aut10] is a specialization of the general process maturity model SPICE [Int12a] for the automotive domain. Its development process is shown in Figure 2.2. In contrast to VDI 2206, it decomposes the phase of system design into two phases to explicitly account for requirements engineering: Its phase of ENG.2 system requirements analysis precedes the phase of

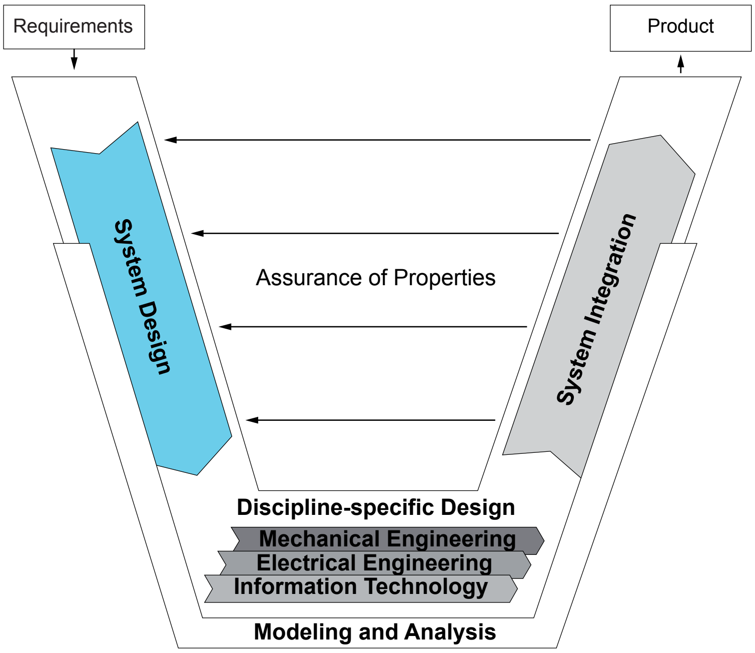


Figure 2.1: VDI 2206 V model [VDI04]

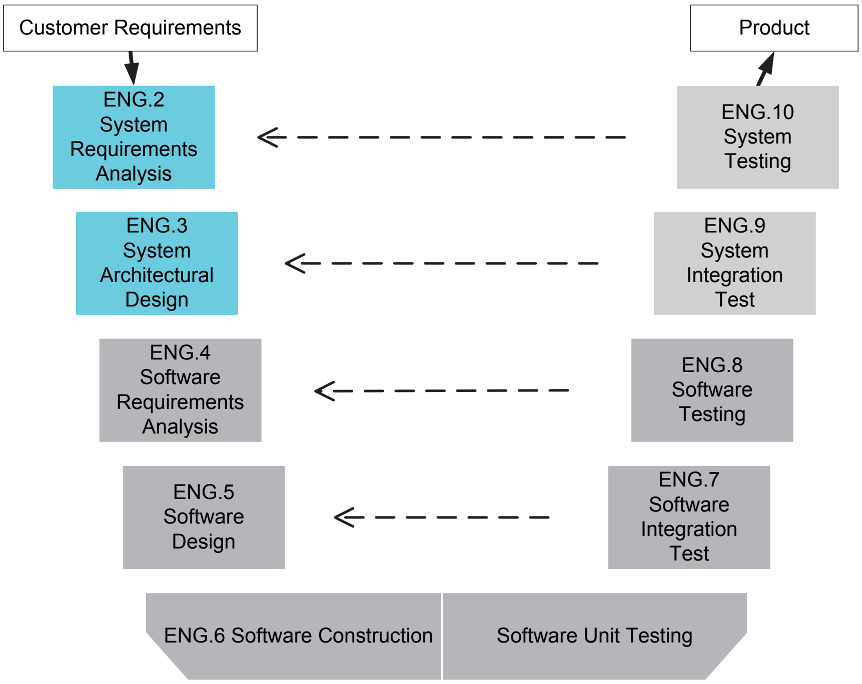

Customer Requirements
Product
ENG.2 System Requirements Analysis
ENG.10 System Testing
ENG.3 System Architectural Design
ENG.9 System Integration Test
ENG.4 Software Requirements Analysis
ENG.8 Software Testing
ENG.5 Software Design
ENG.7 Software Integration Test
ENG.6 Software Construction
Software Unit Testing

Figure 2.2: Automotive SPICE V model

ENG.3 system architectural design. After these two system-level phases, Automotive SPICE focuses on the software development phases ENG.4 to ENG.8 (cf. information technology in VDI 2206). Finally, it also splits the phase called system integration in VDI 2206 into two phases ENG.9 system integration test and ENG.10 system testing, in order to distinguish tests against the system architecture and against the system requirements as specified in the first two phases on the left side of the V model.

The development of safe cyber-physical systems requires the application of *safety engineering* throughout the whole development process: "System safety engineering is a compilation of engineering analyses and management practices that control dangerous situations" [Bah14]. This are activities to

- identify the hazards in a system,
- determine the underlying causes of those hazards,
- develop engineering or management controls to either eliminate the hazards or mitigate their consequences,
- verify that the controls are adequate and in place, and
- monitor the system after it has been changed and modify further as needed [Bah14].

Figure 2.3 shows an excerpt of the development process as prescribed by the automotive safety standard ISO 26262 [Int11f] (cf. Section 2.2). The excerpt focuses on the system and software phases related to Automotive SPICE as shown in Figure 2.2. The number to the left of each phase references the according section of the standard (3-x [Int11d], 4-x [Int11e], 6-x [Int11f]). In phases 3-8 and 4-6 system-level safety requirements are specified (i.e., functional safety requirements and technical safety requirements, cf. Section 2.2). Hence, these phases can be integrated with phase ENG.2 of Automotive SPICE. The phase 4-7 adds safety engineering measures to the system architectural design phase ENG.3. The phase 4-8 adds safety engineering measures to the system-level test and integration phases ENG.9 and ENG.10. The phases 6-6 to 6-11 add safety engineering measures to the software development phases ENG.4 to ENG.8.

In conclusion, the development of safe cyber-physical systems requires sophisticated methods for systems, requirements, and safety engineering in sync with development processes as demanded by standards like VDI 2206, Automotive SPICE, and ISO 26262. Figure 2.4 shows an interdisciplinary development process that integrates the views of those three standards. Customer requirements are the initial input to the development process and specifically to the system requirements analysis phase. The output of that phase are system requirements that include the functional and technical safety requirements of ISO 26262. Output of the system architectural design is the system architecture that forms the basis for the discipline-specific phases of mechanical, electric/electronic (E/E), and software engineering. The discipline-specific integration and test phases are followed by system integration test and system test. Finally, the cyber-physical system is the product of the development process.

In such a development process many persons are involved. They assume one or several roles that are responsible for the tasks of the different development phases. Figure 2.5 shows the roles that are relevant for the remainder of this thesis.

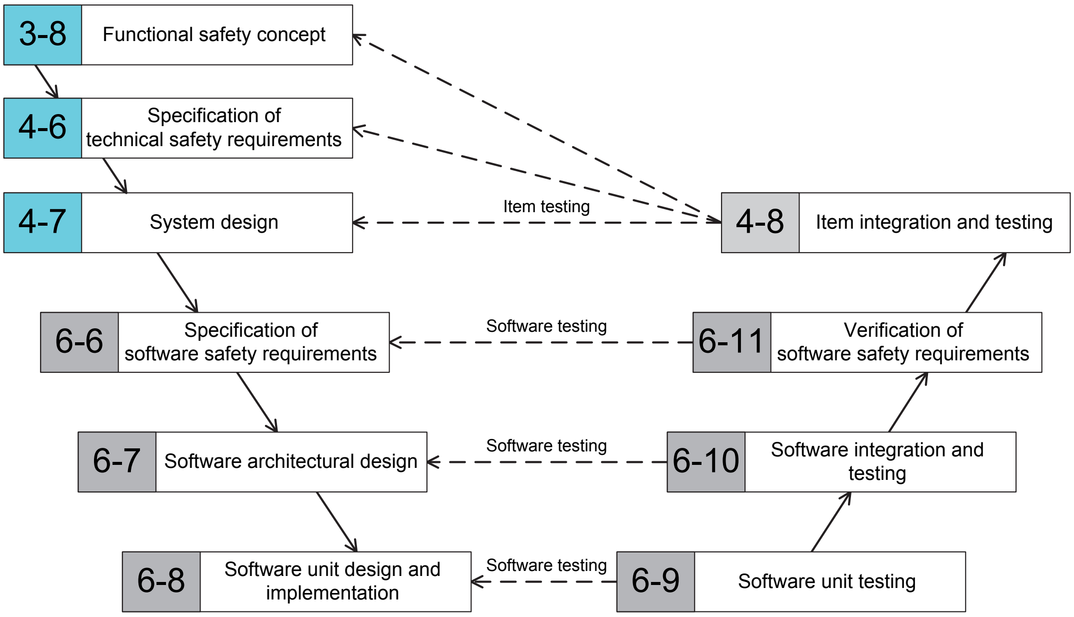

3-8
Functional safety concept
4-6
Specification of technical safety requirements
4-7
System design
Item testing
4-8
Item integration and testing
6-6
Specification of software safety requirements
Software testing
6-11
Verification of software safety requirements
6-7
Software architectural design
Software testing
6-10
Software integration and testing
6-8
Software unit design and implementation
Software testing
6-9
Software unit testing

Figure 2.3: ISO 26262 V model

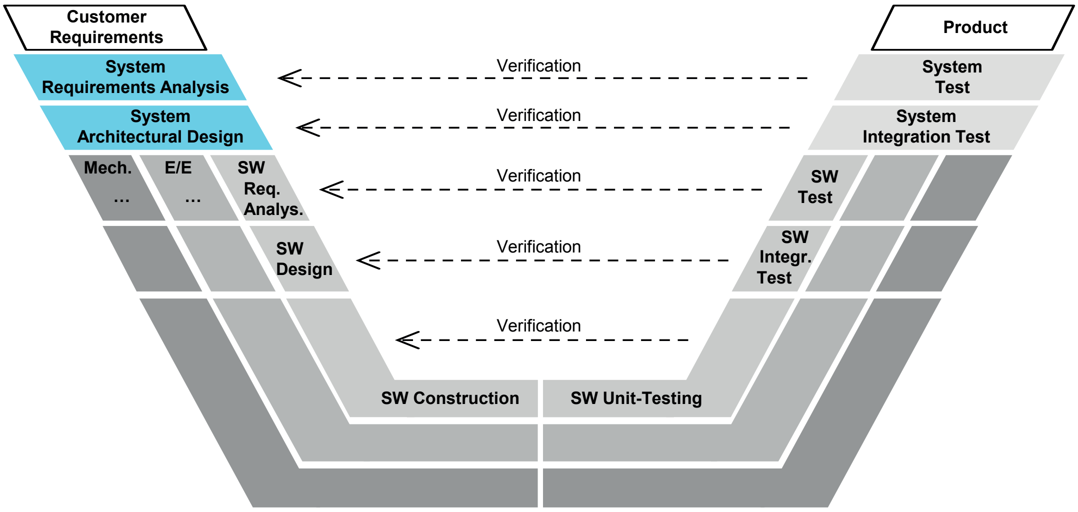


Figure 2.4: Interdisciplinary development process for cyber-physical systems

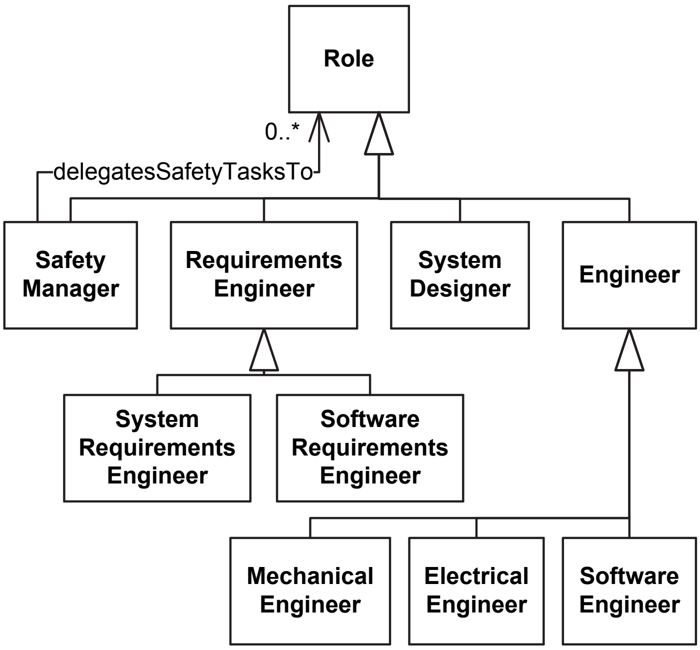


Figure 2.5: Roles in the development process

Definition 2.1 (Safety Manager)
"The safety manager shall be responsible for the planning and coordination of the functional safety activities in the development phases of the safety lifecycle" [Int11c]. "The safety manager can delegate tasks to persons that possess the required skills, competences and qualifications" [Int11c]. The safety manager is responsible for the safety activities (Def. 2.8; e.g., safety analysis) and safety work products (e.g., hazards, safety requirements, and safety case).

In this thesis, whenever we refer to the safety manager doing something, like e.g., writing safety requirements, it is possible that he/she actually delegates that task to a requirements engineer.

Definition 2.2 (Requirements Engineer)
The requirements engineer is a person who – in collaboration with stakeholders – elicits, documents, validates, and manages requirements [IG14].

Definition 2.3 (System Requirements Engineer)
The system requirements engineer is a requirements engineer that is responsible for system-level requirements engineering (i.e., the system requirements analysis phase) and work products (e.g., functional requirements and function hierarchy). We consider this role a synonym for the *requirements owner* defined by Sheard [She96] and used in the integrated system and software development process for cyber-physical systems of Holtmann et al. [HBM+16].

Definition 2.4 (Software Requirements Engineer)
The software requirements engineer is a requirements engineer that is responsible for software-specific requirements engineering (i.e., the software requirements analysis phase) as described by Holtmann et al. [HBM+16].

Definition 2.5 (System Designer)
The system designer is responsible for the phase of system architectural design and the system architecture as defined by Sheard [She96] and used by Holtmann et al. [HBM$^+$16].

For simplicity, we compile all discipline-specific tasks into the roles mechanical engineer, electrical engineer, and software engineer in this thesis (except for the software requirements engineer, cf. Definition 2.4). We consider these roles responsible for discipline-specific design, implementation, and test.

2.2 Automotive Safety Standard ISO 26262

"ISO 26262 is the adaptation of IEC 61508 [Int10a] to comply with needs specific to the application sector of electrical and/or electronic (E/E) systems within road vehicles. ISO 26262 is intended to be applied to safety-related systems that include one or more electrical and/or electronic (E/E) systems and that are installed in series production passenger cars with a maximum gross vehicle mass up to 3.5 t." [Int11a]

In general, ISO 26262 specifies what safety measures (cf. Definition 2.6) have to be taken to develop, produce, and operate a safe vehicle.

Definition 2.6 (Safety Measure)
A safety measure is a safety activity (Def. 2.8) or safety mechanism (Def. 2.7) to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their harmful effects [Int11a].

Definition 2.7 (Safety Mechanism)
A safety mechanism is a technical solution implemented by E/E functions or elements, or by other technologies, to detect faults or control failures in order to achieve or maintain a safe state [Int11a].

Definition 2.8 (Safety Activity)
A safety activity is an activity performed in one or more subphases of the safety life cycle [Int11a]. Examples are the HARA (Def. 2.12) and safety analyses (Def. 2.9) like FMEA.

Definition 2.9 (Safety Analysis)
"The objective of safety analyses is to examine the consequences of faults and failures on the functions, behaviour and design of items and elements. Safety analyses also provide information on conditions and causes that could lead to the violation of a safety goal or safety requirement." [Int11h]
Safety analyses include FMEA and FTA [Int11h].

ISO 26262 consists of the nine normative Parts 1 to 9 and an informative Part 10. Part 1 defines the terms used throughout the standard. In Part 2, a safety life cycle from the concept phase (Part 3), over development (Parts 4 to 6), to the phases of production, operation, service, and decommissioning (Part 7) is defined. Part 8 specifies requirements on supporting processes like configuration and change management. Part 9 contains requirements on safety analyses and defines the rules for ASIL tailoring (cf. Section 2.5). The informative Part 10 provides a guideline on the standard, e.g., concerning its understanding and the specification of a safety case (cf. Definition 2.10).

DEFINITION 2.10 (SAFETY CASE)
The safety case is "an argument that the safety requirements for an item are complete and satisfied by evidence compiled from work products of the safety activities [Def. 2.8] during development" [Int11a]. "The safety case should progressively compile the work products that are generated during the safety lifecycle" [Int11c].

The left of Figure 2.6 shows an excerpt from the concept and development phases of ISO 26262 annotated with the respective part (and section) number. In ISO 26262, the system under development is called *item*. During item definition the system boundary is defined, i.e., what is part of the item and what is part of its environment. Afterward, during hazard analysis & risk assessment (cf. Definition 2.12), hazards (cf. Definition 2.11) of the item are identified, and their risk is determined and specified in form of an ASIL value (Automotive Safety Integrity Level). ASIL values are ranging from ASIL D to ASIL A (in decreasing order) and QM for non-safety-critical elements.

DEFINITION 2.11 (HAZARD)
A hazard is a potential source of physical injury or damage to the health of persons [Int11a].

DEFINITION 2.12 (HAZARD ANALYSIS & RISK ASSESSMENT)
Hazard Analysis & Risk Assessment (HARA) is "a method to identify and categorize hazardous events of items and to specify safety goals and ASILs related to the prevention or mitigation of the associated hazards in order to avoid unreasonable risk" [Int11a].

In general, the risk of a hazard is defined as the combination of its likelihood of occurrence and the severity if the hazard occurs (Risk = Likelihood × Severity). In ISO 26262, the likelihood is approximated by exposure (e.g., how often is the system in situations where the hazard could occur) and controllability (in how far can the hazard be avoided by timely reactions of involved persons). Hence, the ASIL of a hazard is determined by the three metrics exposure, controllability, and severity:
ASIL = Risk = Likelihood × Severity = (Exposure × Non-Controllability) × Severity.

From the identified hazards, so-called Safety Goals are derived that represent top-level requirements to prevent or mitigate the hazards. Each safety goal inherits the ASIL assigned to the corresponding hazard. During the definition of the functional safety concept, the safety goals are refined into Functional Safety Requirements (FSRs) specified on a Preliminary Architecture. Each functional safety requirement inherits the maximum ASIL of the safety goals it refines.

During product development on system level, the functional safety requirements are refined into Technical Safety Requirements (TSRs) that specify requirements on the discipline-spanning System Design. Each technical safety requirement inherits the maximum ASIL of the functional safety requirements it refines. During product development on hardware level, technical safety requirements are refined into Hardware Safety Requirements that specify requirements on the Hardware Detailed Design. Each hardware safety requirement inherits the maximum ASIL of the technical safety requirements it refines. Similarly, during product development on software level, technical safety requirements are refined into Software Safety Requirements that specify requirements on the Software Architectural Design. Each software safety requirement inherits the maximum ASIL of the technical safety requirements it refines.

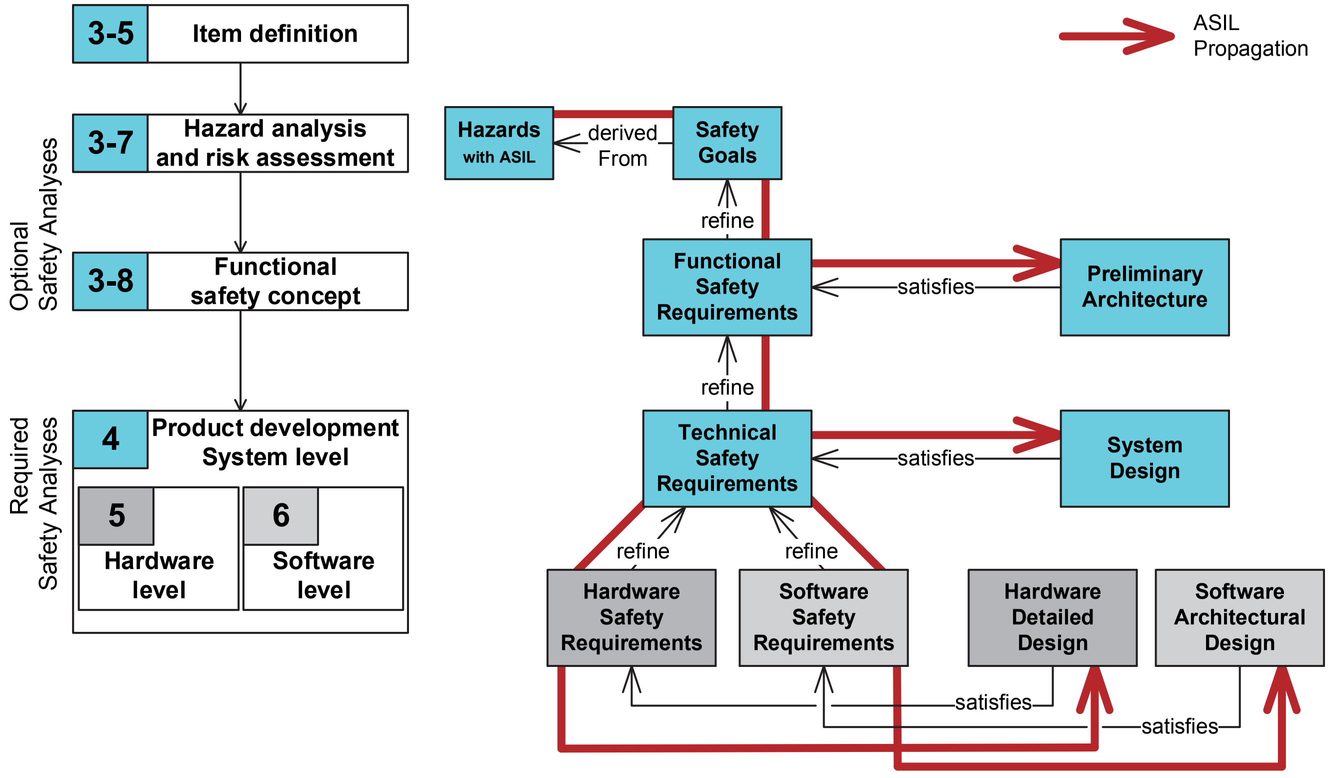


Figure 2.6: Excerpt of ISO 26262 safety life cycle phases with work products and ASIL propagation

Safety analyses support the safety manager to identify failures and failure propagation paths through the system, and, thus, to define corresponding safety requirements. ISO 26262 requires to apply safety analyses in the product development phases 4 to 6. In the earlier phases 3-5 to 3-8 they are only considered optional.

As a refinement of Figure 1.1 on page 2, the red, thick arrows in Figure 2.6 highlight the ASIL propagation. Each ASIL value is initially determined for an identified hazard, and propagates over the different safety requirement abstraction levels to the corresponding system-level and discipline-specific designs. Based on the ASIL of a requirement or design element, ISO 26262 requires the application of safety measures (cf. Definition 2.6) of different sophistication and effort. Hence, a high ASIL that propagates through all phases results in a high safety effort.

The term "functional"

The term "functional" has several meanings in the different fields we touch in this thesis: We have to distinguish *functional safety*, *functional safety requirement*, and *(non-)functional requirement*. Functional safety is concerned with safe system behavior (cf. Definition 2.13). Functional safety requirements (cf. Definition 2.14), technical safety requirements, and hardware/software safety requirements denote the three safety requirement abstraction levels used in ISO 26262 (cf. Figure 2.6). Orthogonal to that definition, in requirements engineering, functional requirements (cf. Definition 2.15) and non-functional requirements (cf. Definition 2.16) are terms used to distinguish requirements on behavior and requirements on quality [IG14].

DEFINITION 2.13 (FUNCTIONAL SAFETY)
Functional safety is the absence of unreasonable risk due to hazards caused by malfunctioning behavior of a system [Int11a].

DEFINITION 2.14 (FUNCTIONAL SAFETY REQUIREMENT)
A functional safety requirement is the specification of an implementation-independent safety behavior, or an implementation-independent safety measure (Def. 2.6) [Int11a].

DEFINITION 2.15 (FUNCTIONAL REQUIREMENT)
A functional requirement is "a requirement concerning a result of behavior that shall be provided by a function of a system" [IG14].

DEFINITION 2.16 (NON-FUNCTIONAL REQUIREMENT)
A non-functional requirement is "a quality requirement or a constraint" [IG14].

In this thesis, we use the abbreviated, general term "safety" instead of "functional safety" (cf. Definition 2.13) to not confuse the reader with the other meanings of "functional". We use the term "functional" to refer to the abstraction level of *functional* safety requirements (cf. Definition 2.14) and corresponding functions of a preliminary architecture (e.g., CONSENS function hierarchy, cf. Section 2.6). We use the term "functional requirement" to refer to non-safety requirements on the ISO 26262 abstraction level of *functional* safety requirements. In conclusion, in this thesis, functional requirements are requirements on the functional abstraction level that can be related to behavior, quality, or constraints, and functional safety requirements are the same but specifically concerned with safety.

2.3 Safety Integrity Levels in other Safety Standards

ISO 26262 is a safety standard specifically for the automotive domain. It is derived from the base standard IEC 61508 [Int10a]. IEC 61508 defines five safety integrity levels (SIL) ranging from SIL 0 (non-safety-critical) to SIL 4 (highly safety-critical). ISO 26262 also defines five safety integrity levels, called Automotive Safety Integrity Levels (ASIL), ranging from QM, over ASIL A, to ASIL D.

Many domains have derived their own safety standard from IEC 61508. The railway domain derived their standard EN 50128 [CEN11] adopting the same five SILs of IEC 61508. So did mechanical and plant engineering with IEC 62061 [Int05]. The safety standard for agricultural vehicles ISO 25119 [Int10b] is also derived from IEC 61508 but uses their own five safety integrity levels, called Agriculture Performance Levels (AgPL), ranging from AgPL a to AgPL e. There are further domains with standards derived from IEC 61508.

Apart from domains basing on IEC 61508 there are other domains that defined safety standards on their own. For instance, in the aerospace domain ARP4754A defines Development Assurance Levels (DAL) [SAE10]. These are also comprised of five values ranging from DAL E (lowest) to DAL A (highest).

Blanquart et al. compared the safety standards of different domains [BAB$^+$12]. Each standard they considered uses some kind of safety integrity levels and the risk assessment is based on (a derivation of) the basic principle Risk $=$ Likelihood $\times$ Severity.

2.4 Safety Analysis using Fault Trees

The objective of safety analysis is to examine the consequences of failures on the functions, behavior and design of a system and its elements (cf. Definition 2.9). In this thesis, we consider the following kinds of failures:

DEFINITION 2.17 (SINGLE-POINT FAILURE)
A single-point failure is a failure that results from one fault and leads directly to the violation of a safety goal (i.e., the occurrence of a hazard) [Int11a].

DEFINITION 2.18 (MULTIPLE-POINT FAILURE)
A multiple-point failure is a failure resulting from the combination of several independent faults that leads directly to the violation of a safety goal (i.e., the occurrence of a hazard) [Int11a].

DEFINITION 2.19 (DUAL-POINT FAILURE)
A dual-point failure is a multiple-point failure resulting from the combination of two independent faults that leads directly to the violation of a safety goal (i.e., the occurrence of a hazard) [Int11a].

DEFINITION 2.20 (DEPENDENT FAILURES)
Dependent failures are failures whose probability of simultaneous or successive occurrence cannot be expressed as the simple product of the unconditional probabilities of each of them ($P_{AB} \neq P_A \times P_B$). [Int11a]

DEFINITION 2.21 (CASCADING FAILURE)
A cascading failure (CF) is a dependent failure (Def. 2.20) of an element causing one or more other elements to fail [Int11a].

DEFINITION 2.22 (COMMON CAUSE FAILURES)
Common cause failures (CCFs) are dependent failures (Def. 2.20) of two or more elements resulting from a single specific event or root cause [Int11a].

Fault trees can be used to specify the correlation of subsystem failures and events leading to a failure or hazard of the overall system. The following Section 2.4.1 briefly introduces their notation and use. Section 2.4.2 explains an adaptation of the fault tree notation for use in hierarchically structured models like component-oriented architectures.

2.4.1 Fault Tree Analysis (FTA)

Fault tree analysis (FTA) [Int06b] is a deductive safety analysis used to identify the possible causes of an event (e.g., a failure). Figure 2.7 shows an example fault tree. Its root node Omission of activateEmergencyBraking is a *top-event* denoted by a rectangle. It represents the failure that the activation of emergency braking is omitted. In deductive fault tree analysis, this event is decomposed into sub-events that lead to its occurrence. Beside events, fault trees can also contain *OR gates* and *AND gates*. In the figure, the top-event occurs if the *basic event* Crash (denoted by a circle) occurs OR the events Omission of reachingLastPointToBrake AND Omission of receivedEmcyBrakeWarning occur simultaneously. The latter two events are top-events in other fault trees that are referenced in this fault tree by the triangle notation. On the contrary, basic events are events that are not refined by any subtree of gates and events.

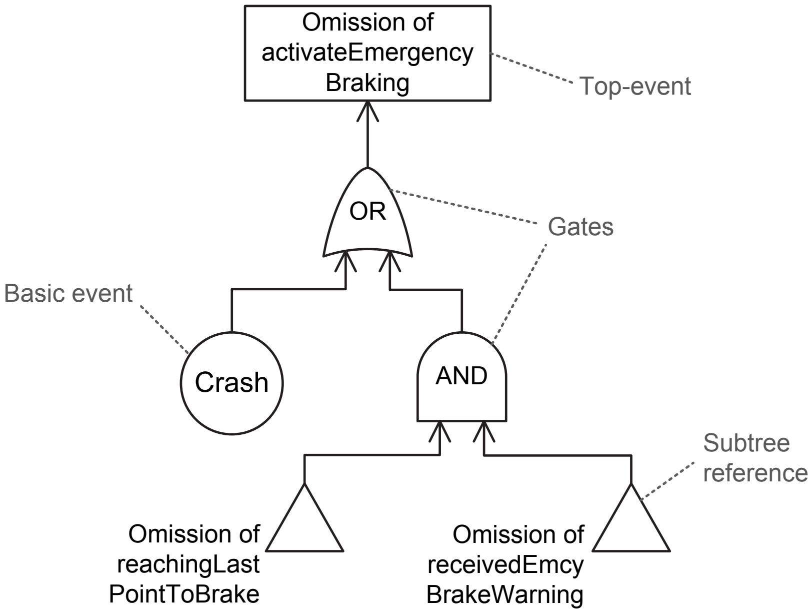

Omission of activateEmergency Braking
Top-event
OR
Gates
Basic event
Crash
AND
Subtree reference
Omission of reachingLast PointToBrake
Omission of receivedEmcy BrakeWarning

Figure 2.7: Example fault free

Once a fault tree is specified, it is analyzed for so-called *minimal cut sets*. They describe minimal sets of fault tree leaf nodes (typically basic events) whose simultaneous occurrence leads to the occurrence of the top-event. Figure 2.7 contains the two minimal cut sets {Crash} and {Omission of reachingLastPointToBrake, Omission of receivedEmcyBrakeWarning}.

2.4.2 Component Fault Trees (CFTs)

In classical fault tree analysis, for each hazard of a system, a fault tree with the hazard occurrence as top-event is specified. If fault tree diagrams get too large to grasp, subtrees are described in separate fault trees and referenced by the triangle notation (cf. Figure 2.7). The classical fault tree notation has no defined traceability to a system, its components, or component interfaces. Furthermore, fault trees relating to the same system usually have common subtrees whose consistency has to be maintained if the system changes.

To overcome these disadvantages, Kaiser et al. developed *Component Fault Trees* (CFTs) that decompose classical fault trees into reusable components that can directly relate to components of a system [KLM03; HTZ$^+$12]. Figure 2.8 shows a CFT representation of the fault tree from Figure 2.7. A CFT is represented by a rectangle with the keyword "CFT" in the top left followed by the name of the CFT (e.g., MakeDecisionCFT in Figure 2.8). The internal CFT syntax closely follows the syntax of classical fault trees. The triangle notation is used to connect different CFTs. Like UML ports, the triangles are placed on the border of the CFT. So-called *input failure modes* (similar to subtree references in classical fault trees) are denoted by white triangles, and output failure modes (relating to input failure modes of other CFTs) by black triangles. Figure 2.8 shows the advantage of this notation: The crash and omission input failure modes result in the output failure mode Omission of activateEmergencyBraking, as denoted in the classical fault tree of Figure 2.7. However, they also result in the output failure mode Omission of sendEmcyBrakeWarning. In classical fault tree analysis, this would need to be specified by two separate fault trees that only differ in their top-event and have to be kept in sync.

2.5 ASIL Tailoring

As a high ASIL requires a high degree of rigor that implies high development effort, there is a demand to reduce the amount of high ASIL requirements and subsystems. ASIL tailoring is a means to reduce the ASIL of parts of the system by separation and decomposition. The automotive safety standard ISO 26262 defines rules that a tailoring has to obey [Int11h]. The rules describe under what conditions a specific ASIL value of a safety requirement (and its satisfying subsystem) may be reduced and to what value. Safety standards for other domains (cf. Section 2.3) have similar rules, e.g., ARP4754A for aerospace.

The following sections describe the two ASIL tailoring measures *separation* (Section 2.5.1) and *decomposition* (Section 2.5.2) and the according rules by example.

2.5.1 Separation

The first ASIL tailoring option to reduce the development effort is to separate subsystems with different levels of safety-criticality (i.e., with different ASIL values). If one can make sure that a less safety-critical subsystem will not interfere with a more safety-critical subsystem, i.e., that a failure of the less critical subsystem will not cause a failure of the more critical subsystem violating a more critical safety requirement (i.e., causing a more critical hazard), the two subsystems are considered separate and may have different ASIL values assigned [Int11h, Clause 6].

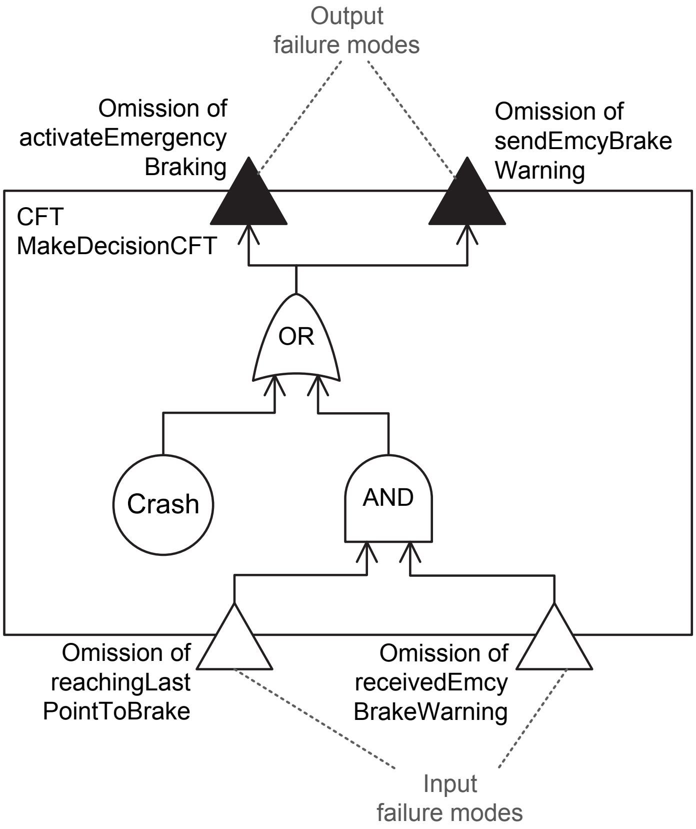


Figure 2.8: Example Component Fault Tree

Definition 2.23 (Freedom from Interference)
An element a is free of interference from an element b if there are no cascading failures (Def. 2.21) from b to a that could lead to the violation of a safety requirement [Int11a].

Definition 2.24 (ASIL Separation)
ASIL separation is the act of apportioning safety requirements to sub-elements of an element, with the objective of reducing the ASIL of sub-elements that do not interfere (Def. 2.23) with sub-elements with higher ASIL safety requirements.

Figure 2.9 depicts the separation of the EBEAS and the Body Control Module. The latter controls the brake lights and turn signals and does not send any information to the EBEAS. Thus it seems likely, that it's malfunction will not cause the EBEAS to fail as well.

Table 2.1 sketches the safety requirements for the two systems. The Body Control Module by itself only causes ASIL B hazards, and thus, has to fulfill ASIL B safety requirements only (SR_1 ff.). The EBEAS is highly safety-critial (cf., ASIL D hazards in Figure 1.3 on page 4), and thus, has to fulfill some ASIL D requirements (SR_{42} ff.). For the EBEAS scenario, the Body Control Module is connected to the EBEAS. In general, this would require to increase the ASIL value of the Body Control Module to ASIL D as well. This would increase the safety effort significantly, as the whole Body Control Module would need to be redeveloped applying all the safety measures required for ASIL D systems. If the safety analysis shows that no failure of the Body Control Module propagates to the EBEAS leading

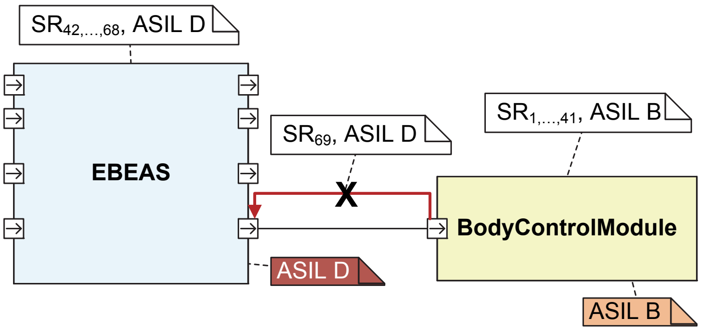


Figure 2.9: ASIL tailoring measure: Separation

to the violation of ASIL C or D requirements (safety requirement SR_{69}), the ASIL of the Body Control Module does not need to be increased.

Table 2.1: Requirements for separation

ID	Description	ASIL
SR_1 ...	The Body Control Module shall... ...	ASIL B ...
SR_{42} ...	The EBEAS shall... ...	ASIL D ...
SR_{69}	There shall not be any cascading failures from the Body Control Module to the EBEAS.	ASIL D

2.5.2 Decomposition

The second ASIL tailoring option to reduce the development effort is to decompose a system's highly safety-critical requirement into redundant, less critical safety requirements that are realized by independent subsystems [Int11h, Clause 5]. By adding redundancy in this way, two systems would have to fail at the same time (dual-point-failure), in order for the addressed hazard to occur.

Definition 2.25 (Independence)
Two or more elements are independent if there are no dependent failures (Def. 2.20) between them that could lead to the violation of a safety requirement [Int11a].

Definition 2.26 (ASIL Decomposition)
ASIL decomposition is the act of apportioning safety requirements redundantly to sufficiently independent elements (Def. 2.25), with the objective of reducing the ASIL of the redundant safety requirements that are allocated to the corresponding elements [Int11a].

Figure 2.10 and Table 2.2 depict an example of an ASIL D requirement SR_2 that is decomposed into lower ASIL requirements $SR_{2.1}$ and $SR_{2.2}$ in order to keep the ASIL of the two systems Adaptive Cruise Control and Vehicle2X Communication lower than the ASIL of the EBEAS. The EBEAS can cause the hazards of missing brake activation and missing steering for evasion (cf., hazards H1 and H3 in Figure 1.3 on page 4). The according safety requirement SR_1 shall avoid these hazards and thus inherits their ASIL value of ASIL D. If the EBEAS shall activate the brakes or steering, it needs to get the information whether the leading vehicle is braking to make a decision. This is stated in requirement SR_2 which accordingly also inherits ASIL D. The leading vehicle sends an emergency brake warning via a vehicle-to-vehicle message. So, SR_2 would be realized by the Vehicle2X Communication that thus would inherit the ASIL D of the requirement. As described in Section 1.1, the Vehicle2X Communication has a predetermined ASIL value of QM. So, the increase to ASIL D would result in high development effort. In addition to the vehicle-to-vehicle message, the braking of the leading vehicle can also be detected by the sensors of the Adaptive Cruise Control. This means that there are two redundant sources of information. This fact can be exploited for requirements decomposition with respect to ASIL tailoring. The requirement SR_2 is decomposed into the two redundant requirements $SR_{2.1}$ and $SR_{2.2}$ that are allocated to the Vehicle2X Communication and the Adaptive Cruise Control, respectively. Consequently, the two systems would have to fail at the same time, in order for one of the hazards H1 and H3 to occur (cf., the AND-gate in Figure 2.10). To make sure that the two systems do not fail for the same reason, the three safety requirements $SR_{2.3}$ to $SR_{2.5}$ are added.

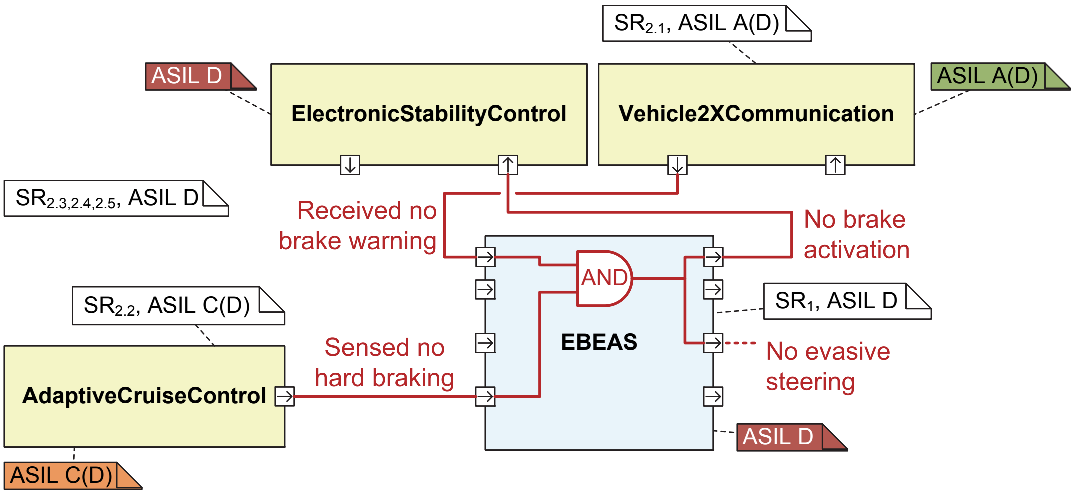


Figure 2.10: ASIL tailoring measure: Introduction of redundancy

As there are two systems that redundantly satisfy the original safety requirement SR_2, the safety standard ISO 26262 allows to develop each of the systems with a lower degree of rigor than the original requirement's ASIL value requires. But in sum, the applied rigor has to still meet the original ASIL. This can be shown by applying simple arithmetics: each ASIL value is assigned with a number from 0 to 4 and the sum of the decomposed ASIL values needs to reach the original ASIL value [APW$^+$14]. In the example, $SR_{2.1}$ is assigned with the ASIL value ASIL A and $SR_{2.2}$ with ASIL C, which adds up to ASIL D $(1 + 3 = 4)$. When applying this decomposition ASIL tailoring method, ISO 26262 still requires to apply some safety measures in accordance with the original ASIL, especially in the integration phases.

Table 2.2: Requirements decomposition for ASIL tailoring

ID	Description	ASIL
SR_1	If the EBEAS determines the leading vehicle's hard braking, the EBEAS shall execute an emergency braking or evasion maneuver.	ASIL D
SR_2	If the leading vehicle brakes hard, the EBEAS shall be informed.	ASIL D
$SR_{2.1}$	If the Vehicle2X Communication receives a hard braking warning, it shall inform the EBEAS.	ASIL A(D)
$SR_{2.2}$	If the Adaptive Cruise Control senses the leading vehicle's hard braking, it shall inform the EBEAS.	ASIL C(D)
$SR_{2.3}$	There shall not be any cascading failures from the Vehicle2X Communication to the Adaptive Cruise Control.	ASIL D
$SR_{2.4}$	There shall not be any cascading failures from the Adaptive Cruise Control to the Vehicle2X Communication.	ASIL D
$SR_{2.5}$	There shall not be any common cause failures of the Vehicle2X Communication and the Adaptive Cruise Control.	ASIL D

Thus, the original ASIL value always has to be stated in parentheses (cf., Figure 2.10 and Table 2.2).

In summary, the example contains ASIL D hazards and according safety requirements for the EBEAS. The Vehicle2X Communication provides information, whose omission can lead to those hazards occurring and thus would be assigned with ASIL D as well. The predetermined ASIL value of the Vehicle2X Communication is only QM, so ASIL D would result in a high development effort. The Adaptive Cruise Control can redundantly provide the same information required by the EBEAS. Its predetermined ASIL value is ASIL C. By decomposing the original ASIL D requirement SR_2 into an ASIL A(D) requirement for the Vehicle2X Communication and an ASIL C(D) requirement for the Adaptive Cruise Control, the latter stays ASIL C and the ASIL value of the Vehicle2X Communication is only increased one level to ASIL A. Thus, the overall development effort is reduced.

2.6 Model-based Systems Engineering with CONSENS

The CONceptual design Specification technique for the ENgineering of complex Systems (CONSENS) is a model-based systems engineering language and method [GRS14, Section 4.1]. Figure 2.11 sketches its so-called *partial models* and the process of their creation. A detailed description of the CONSENS process including the transition to software engineering is specified by Holtmann et al. [HBM$^+$16]. Firstly, the system's environment, its application scenarios, and requirements are specified. These work products merge into a function hierarchy specifying the functions the system shall provide. Based on the function hierarchy, the active structure is developed that specifies the system's internal structure. Afterward, the system's behavior and shape is specified. Eventually, the partial models serve as input to the mechanical engineering, electrical engineering, and information technology and their discipline-specific work products (cf. Figure 2.1 on page 14).

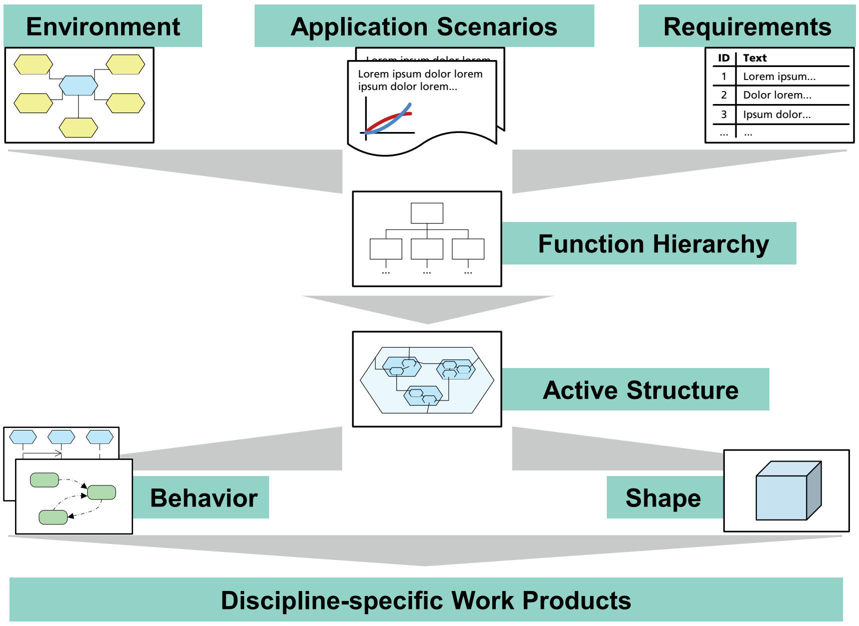


Figure 2.11: Simplified CONSENS process with partial models

Instead of the plain CONSENS language, we use SysML4CONSENS [IKD$^+$13; KDH$^+$13] that allows to specify CONSENS partial models based on the Systems Modeling Language (SysML) [Obj15a]. SysML provides no systems engineering method but is based on UML. Hence, using SysML4CONSENS enables a conceptual integration of the CONSENS method with other UML-based methods used in this thesis, and easier technical integration using established and extensible UML tool-support (e.g., Eclipse Papyrus[1]). Figure 2.12 shows the elements of the SysML4CONSENS profile used in this thesis. We explain the elements using the following examples of CONSENS partial models.

Environment

The partial model *environment* is used to specify the system boundary, i.e., the context of the system. Figure 2.13 sketches the environment of the EBEAS. The SysML block definition diagram (bdd) Environment Types specifies types of system and environment elements, their ports, and their interfaces. The SysML internal block diagram (ibd) Environment specifies interactions and connections between the system and its surrounding elements. The system under development is marked by the SysML4CONSENS stereotypes System Template and System Exemplar, respectively. The elements of the system's environment are marked by the SysML4CONSENS stereotypes Environment Element Template and Environment Element Exemplar, respectively. In Figure 2.13, the EBEAS is the system under development. It is screwed to the Vehicle Body via a Screw Connection marked as SysML4CONSENS Mechanical Connection. The EBEAS communicates with the Electronic Stability Control via a bus system called ADAS Bus. This is realized by exchange of electronic signals represented by the SysML4CONSENS Energy Flow Specification named Adas Bus Signals. Accordingly, both ECUs are connected to the ADAS Bus via ports typed by that energy

[1] www.eclipse.org/papyrus

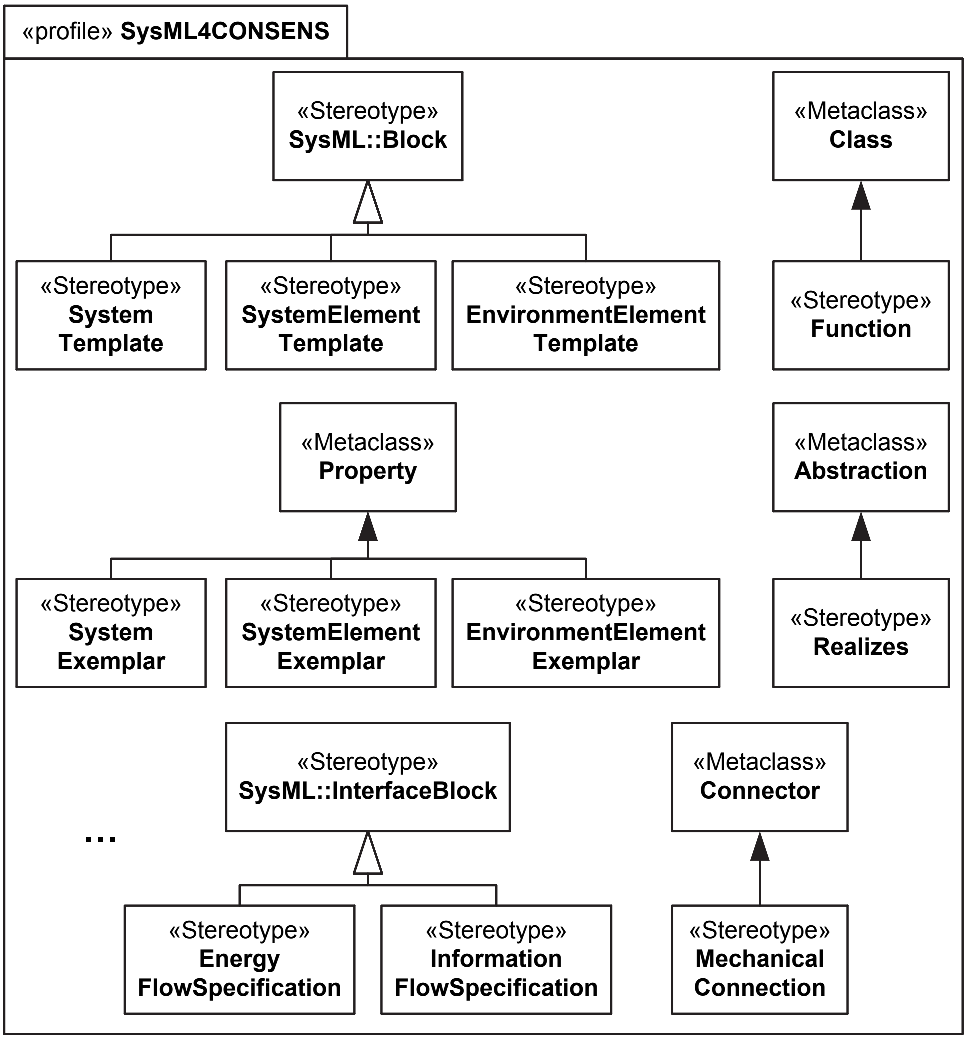

«profile» SysML4CONSENS
«Stereotype» SysML::Block
«Metaclass» Class
«Stereotype» System Template
«Stereotype» SystemElement Template
«Stereotype» EnvironmentElement Template
«Stereotype» Function
«Metaclass» Property
«Metaclass» Abstraction
«Stereotype» System Exemplar
«Stereotype» SystemElement Exemplar
«Stereotype» EnvironmentElement Exemplar
«Stereotype» Realizes
«Stereotype» SysML::InterfaceBlock
«Metaclass» Connector
...
«Stereotype» Energy FlowSpecification
«Stereotype» Information FlowSpecification
«Stereotype» Mechanical Connection

Figure 2.12: Excerpt of SysML4CONSENS profile

flow specification. The electronic bus signals are used to exchange information between the EBEAS and the Electronic Stability Control. This is specified by direct port connections typed by SysML4CONSENS Information Flow Specifications. The Electronic Stability Control sends the current velocity to the EBEAS and the EBEAS sends braking requests to the Electronic Stability Control.

Application Scenarios

The partial model *application scenarios* is used to specify use cases of the system under development by informal texts and figures. They do not describe any internal structure or behavior of the system but its externally visible behavior.

We consider application scenarios an abstract and informal specification of goals for the system under development that could be the result of requirements elicitation workshops with the customer. We see them as part of customer requirements specifications that are the input

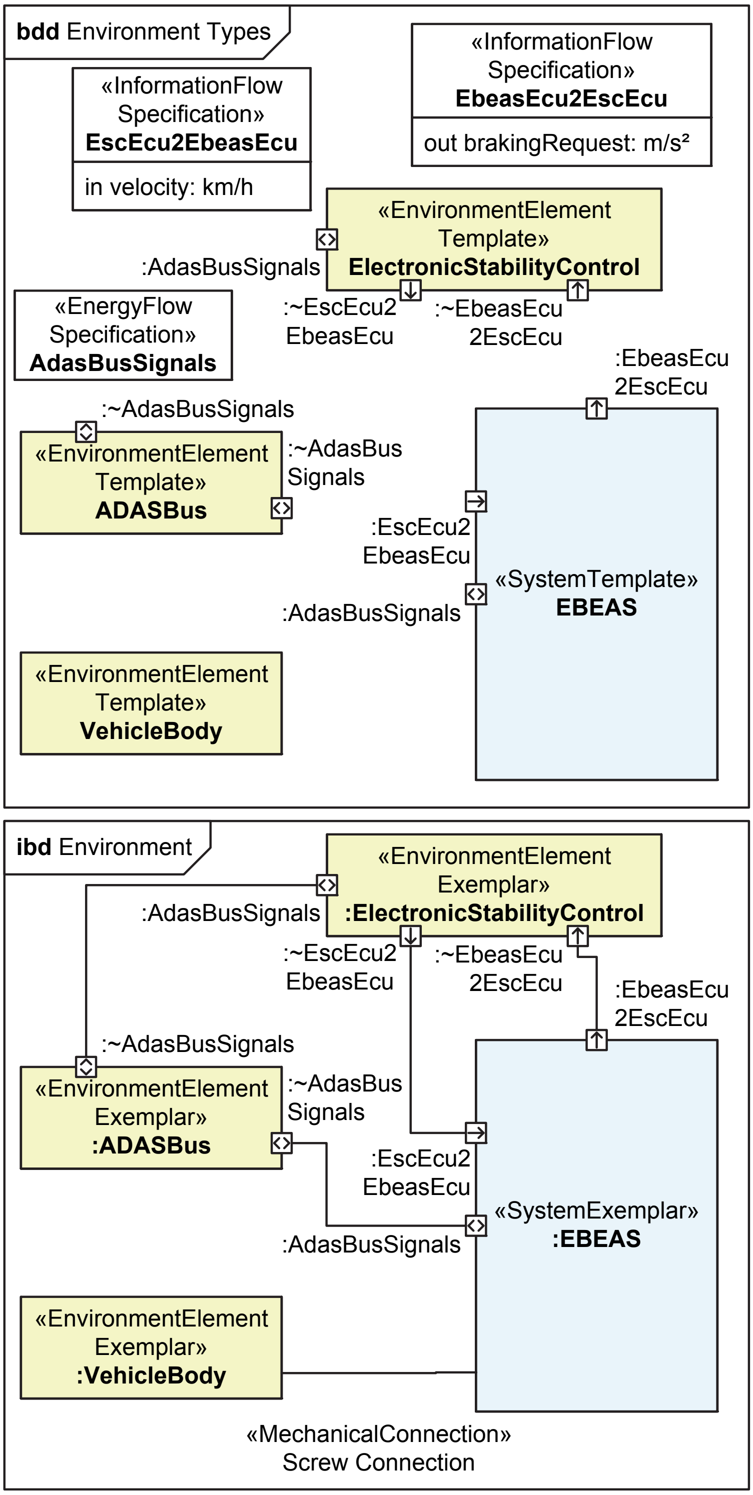


Figure 2.13: Example of CONSENS environment

to our approach (cf. Figure 1.4 on page 10). Hence, we do not use application scenarios in this thesis.

Requirements

The partial model *requirements* is a list of textual requirements that refer to elements of other partial models (e.g., functions or system elements).

We do not use this partial model in this thesis. Instead, we specify requirements as Modal Sequence Diagrams (cf. Section 2.7 and Chapter 4). However, we also integrate our approach with textual requirements that could be seen as instantiation of this partial model (cf. Section 4.5).

Function Hierarchy

The partial model *function hierarchy* is used to decompose the functionality that the system under development shall provide. The functionality is described by functions that are decomposed level by level, until they are sufficiently trivial to implement.

Figure 2.14 shows the function hierarchy for the system EBEAS from Figure 2.13. Functions are represented by UML classes marked with the SysML4CONSENS stereotype Function. The root node FEBEAS represent the overall functionality of the EBEAS. It is decomposed into four functions:

- The function Analyze Situation shall analyze the vehicles surrounding and velocity,
- the function Make Decision shall make the decision to brake or evade,
- the function Communicate with other Vehicles shall provide the vehicle-to-vehicle communication, and
- the function Ensure Passenger Safety shall activate the emergency braking or evasion.

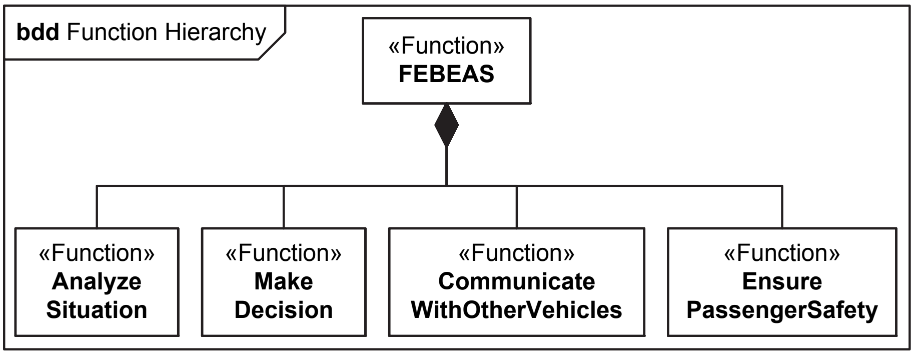

bdd Function Hierarchy
«Function» FEBEAS
«Function» Analyze Situation
«Function» Make Decision
«Function» Communicate WithOtherVehicles
«Function» Ensure PassengerSafety

Figure 2.14: Example of CONSENS function hierarchy

Active Structure

The partial model *active structure* is used to specify the internal structure of the system under development. Figure 2.15 shows the active structure of the system EBEAS from Figure 2.13. Elements inside the system under development are called *system element* by CONSENS. Accordingly, they are marked with the SysML4CONSENS stereotype System Element Exemplar (and System Element Template in a corresponding bdd not shown in the figure). The EBEAS contains the cores Lockstep Cores and Performance Core and the bus interfaces ADAS Bus Interfaces and V2X Bus Interface. Their connections are specified using ports and SysML4CONSENS flow specifications as it is done in the partial model *environment* (the details are omitted in Figure 2.15).

In addition, the system elements realize the leaf functions of the function hierarchy. This is specified by UML abstractions with the SysML4CONSENS stereotype Realizes. In Figure 2.15, the system element Lockstep Cores realizes the function Make Decision from Figure 2.14.

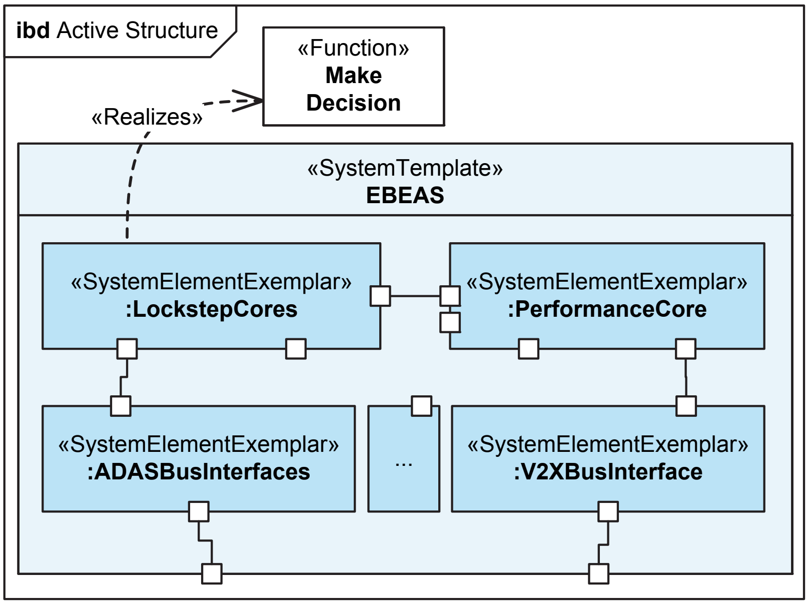


Figure 2.15: Example of CONSENS active structure

Behavior

The partial model *behavior* is used to specify the technical behavior of the overall system and its environment and the internal system elements. This can be done by UML sequence diagrams and state machines.

We specify communication of the overall system with its environment on the functional abstraction level using the function hierarchy root node and functional representations of the environment (cf. Chapter 4). Moreover, we do not specify internal system element communication, as our approach ends with the specification of a system architecture represented as active structure. Hence, we do not use the behavior partial model in this thesis.

Shape

The partial model *shape* is used to specify the three dimensional shape of the system under development. This is not realized by the SysML4CONSENS profile but has to be done in a CAD/CAM tool used by mechanical engineering.

This partial model is not used in this thesis because our approach ends with a system architecture specified as active structure.

2.7 Modal Sequence Diagrams (MSDs)

Modal Sequence Diagrams (MSDs) are a formal, model- and scenario-based behavior specification language based on UML sequence diagrams [HM08]. Harel and Maoz defined MSDs to specify provisional and required sequences of events (e.g., message exchange) [HM08]. Greenyer et al. added language features to specify real-time behavior [Gre11; BGH$^+$14]. Holtmann and Meyer adapted MSDs to obey the communication rules of hierarchically structured component-oriented architectures (e.g., information hiding concepts using ports and interfaces) [HM13]. In this thesis, we follow the MSD requirements language definition compiled in the technical report of Holtmann et al. [HFK$^+$16].

Figure 2.16 shows an example of a so-called *MSD specification* for the EBEAS. It describes message-based interaction behavior of a set of objects. It consists of UML classes, UML collaborations, and MSDs. Classes define the types of objects. Their operations define the messages an object can receive. In the figure, the EBEAS provides two operations and the other two ECUs each provide one. Collaborations are used to specify different situations, application scenarios, or use-cases of a system. In the figure, the collaboration EBEAS Collaboration contains one role for each type defined in the class diagram. Another collaboration based on the same class diagram could, for instance, leave out the Vehicle2X Communication to specify behavior of the EBEAS without communication. For each collaboration a set of MSDs can be specified. Each of their lifelines represents a role from the corresponding collaboration.

MSD specifications can be validated by simulation with the so-called *play-out* algorithm [BGP13]. For that, MSDs add the modalities of *execution kind* and *temperature* to messages of UML sequence diagrams. Monitored messages are represented by a dashed line, executed messages by a solid line. Cold messages are represented by a blue line, hot messages by a red line. For readability without color-printing, we annotate the execution kind and temperature on the side of an MSD using the letters “c”/“h” for cold/hot and “m”/”e” for monitored/executed.

The terms execution kind and temperature are defined based on the play-out of an MSD specification (i.e., the execution of a set of diagrams). From the point of view of one MSD, a message with execution kind *monitored* can be observed during the execution of the MSD but its occurrence is not required. An *executed* message, on the contrary, is required to occur during the execution of an MSD. If it is not sent/received, a liveness violation occurs (i.e., a *liveness property* is violated [Lam77]). In that case, we say the MSD is violated, in this thesis. A message with temperature *cold* may be sent/received after any preceding and before any subsequent messages of the same MSD, but its execution is not required to occur in this order (it is not strict). If any other message of the same diagram occurs when the cold message is expected, a cold violation occurs and the MSD execution is terminated. In that case, we say the MSD is discarded, in this thesis. A *hot* message, on the contrary, has to strictly occur in the order as specified in the MSD. If any other message of the same diagram occurs when the hot message is expected, a hot violation occurs that is a violation of a *safety property* [Lam77]. In that case, we say the MSD is violated, in this thesis.

The play-out algorithm uses so-called *cuts* to describe the execution state of an MSD specification. Each MSD has its own cut that progresses from top to bottom and advances whenever a message of the diagram occurs. The overall system state is defined by the set of

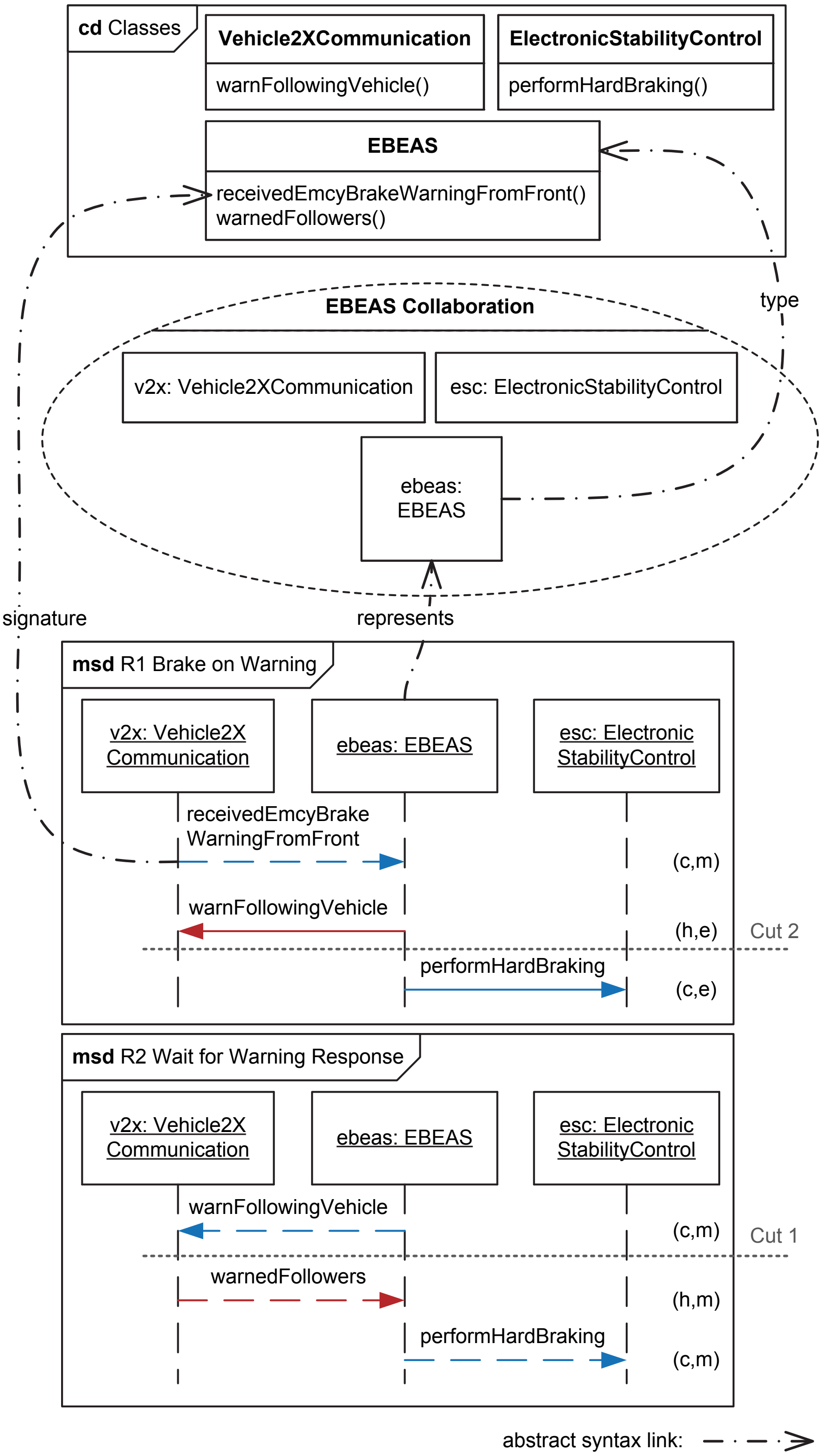


Figure 2.16: Example of an MSD specification

all cuts. Each MSD starts in cut 0 located at the top of the MSD above the first message. The first message of an MSD is called *minimal message* and is always cold and monitored.

Figure 2.16 contains the two MSDs R1 and R2. R1 specifies the following behavior: if the EBEAS is informed about a received emergency brake warning from a vehicle in front, it warns the following vehicle and afterward performs a hard braking maneuver. R2 specifies the behavior that the EBEAS waits for the Vehicle2X Communication to have warned following vehicles before braking. In the beginning, both MSDs are inactive. If the message receivedEmcyBrakeWarningFromFront occurs, the MSD R1 is activated and its cut advances from above that minimal message (cut 0) to cut 1 above the message warnFollowingVehicle. In this state, both messages warnFollowingVehicle and performHardBraking have to occur eventually because they are specified as executed. However, because the message warnFollowingVehicle is hot, it has to occur before performHardBraking. If it occurs, the cut advances to cut 2 above performHardBraking. In addition, the MSD R2 is activated and its cut advances to cut 1 because warnFollowingVehicle is its minimal message. This state is depicted by the two cuts in Figure 2.16. In this state, R1 requires performHardBraking to occur next. However, R2 prohibits this because it also contains that message but its cut is above the hot message warnedFollowers. Thus, the Vehicle2X Communication has to send warnedFollowers before the EBEAS may send performHardBraking. Once warnedFollowers occurred, the cut of both MSDs is before the message performHardBraking and it may be sent. Then, both cuts reached the end of the respective MSD, and the MSDs become inactive until they are again activated by the occurrence of their minimal message.

Additionally to play-out simulation, MSD specifications can be formally verified for consistency by synthesizing global controllers [GBC$^+$13]. If the MSD R1 in Figure 2.16 also contained the message warnedFollowers and the message performHardBraking was hot, the cuts shown in both MSDs could not progress. R1 would require the hot performHardBraking to occur before warnedFollowers may occur, and R2 still requires the hot warnedFollowers to occur before performHardBraking. This would be an inconsistency in the MSD specification that is found by the controller synthesis.

In addition to messages, MSDs also can contain *conditions* (represented as hexagon, cf. Figure 3.9 on page 49). A condition contains a boolean expression that is evaluated when the MSD execution reaches the condition. The execution kind of conditions is fixed to their temperature: a cold condition is always monitored and a hot condition is always executed. For specifying real-time behavior, MSDs can also contain so-called *clock conditions*. These conditions are denoted by an hourglass in their top right (cf. Figure 4.12 on page 79). A clock condition contains a boolean expression concerning a clock variable that holds the elapsed time. If the MSD execution reaches a cold (clock) condition and its expression evaluates to false, a cold violation occurs and the MSD is discarded. If the MSD execution reaches a hot (clock) condition and its expression never evaluates to true, a liveness violation occurs. If the MSD execution reaches a hot (clock) condition and any other message of the same diagram occurs before its expression evaluates to true, a hot violation occurs. Both cases are a violation of the MSD.

2.8 Goal Structuring Notation (GSN)

The Goal Structuring Notation (GSN) is used to specify model-based argument hierarchies [Kel98; Ass18]. This graphical notation shall support the understanding of complex arguments and reduce the problem of ambiguity in informal textual arguments. GSN arguments follow a tree-like structure of goals that are refined by subgoals using certain argument strategies. The leaf goals are supported by evidence in form of solutions or marked as undeveloped.

Figure 2.17 shows an example argument in Goal Structuring Notation. It argues that all hazards of a system have been mitigated (cf. root goal Goal 1). This goal is in the context of the identified hazards H1 and H2. The hazards are referenced by the *context* Hazards, denoted by a rectangle with rounded sides. Context nodes are linked via "in context of" links depicted with a white arrow head.

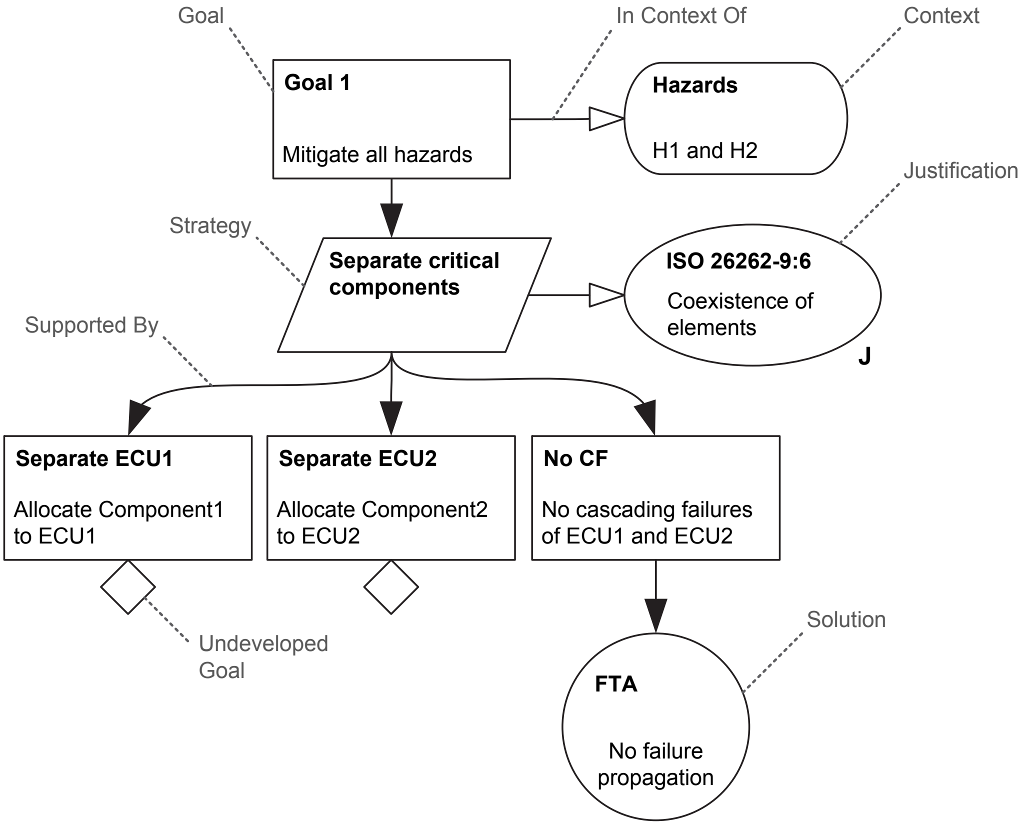


Figure 2.17: Example of an argument in Goal Structuring Notation

Goal 1 is reached by separating safety-critical components. This is specified via the *strategy* Separate critical components denoted by a parallelogram. Strategies and goals are linked via "supported by" links depicted with a black arrow head. The applicability of strategies can be argued via *justifications* denoted as ellipse with a "J" in the lower right. Justifications are linked via an "in context of" link. In the figure, the applicability of the separation strategy is argued via the coexistence of elements section of the standard ISO 26262. The strategy Separate critical components is supported by the three subgoals

Separate ECU1, Separate ECU2, and No CF. The first two goals argue that two software components Comp1 and Comp2 are allocated to separate ECUs of a vehicle. The third goal No CF argues that there are no cascading failures (CF) between the two ECUs. The first two goals are marked as *undeveloped* via a diamond below the rectangle. Undeveloped goals represent sub-arguments that need to be supported by further subgoals, strategies, and/or solutions. The goal No CF is supported by the *solution* FTA denoted by a circle. This solution shall represent the result of a fault tree analysis, assuring that there is no failure propagation between the two ECUs.

3

ASIL Tailoring Process on Functional Safety Requirements

To apply ASIL tailoring on functional safety requirements, we propose a systematic ASIL tailoring process. That process is described in this chapter. It is embedded into the earliest phases *system requirements analysis* and *system architectural design* of a development process complying to Automotive SPICE and ISO 26262 (cf. Section 2.1).

Concepts presented in this chapter have been published in [Foc16].

This chapter is structured as follows. First, Section 3.1 summarizes the scientific contributions of the ASIL tailoring process. Section 3.2 gives an overview of its process steps and work products. Section 3.3 describes the step of analyzing the environment and hazards. Section 3.4 describes the steps of specifying functions and functional (safety) requirements. Section 3.5 describes the steps of safety analysis and ASIL calculation on functions. Section 3.6 describes the step of allocating functions to the system architecture. Section 3.7 lists assumptions and limitations of the process. The final Section 3.9 concludes this chapter.

3.1 Contributions

The contributions of this chapter can be summarized as follows:

- The presented ASIL tailoring process integrates concepts of model-based safety engineering with model-based requirements and systems engineering. This is a step towards well-integrated processes for these domains as required for handling complexity (cf. Challenge C1 in Section 1.2).

- The ASIL tailoring process is embedded into a standard development process compliant to Automotive SPICE (cf. Section 2.1).

- The process is a concrete solution fulfilling the requirements posed by the automotive safety standard ISO 26262 concerning safety requirements engineering and ASIL tailoring (cf. Sections 2.2 and 2.5).

- The process applies ASIL tailoring in the earliest possible phase of the development process (i.e., on functional requirements level before the system architecture is defined). This early planning of ASIL tailoring measures is required due to the rising complexity and mixed-criticality of ADAS (cf. Challenge C1 in Section 1.2).

3.2 Overview of Process Steps and Work Products

The ASIL tailoring process is embedded into the model-based systems engineering method *CONSENS* (cf. Section 2.6). In CONSENS, based on customer requirements, the environment of the SUD is specified, followed by requirements, the function hierarchy, and the active structure. CONSENS does neither consider safety nor define in detail how the function hierarchy and requirements are developed. Our process adds the identification of hazards on the environment, the interleaved development and refinement of the function hierarchy and corresponding functional (safety) requirements, and ASIL tailoring steps. Figure 3.1 recapitulates the ASIL tailoring process from Figure 1.4 on page 10 with its process steps and flow of work products. The relations between the work products are shown in Figure 3.2.

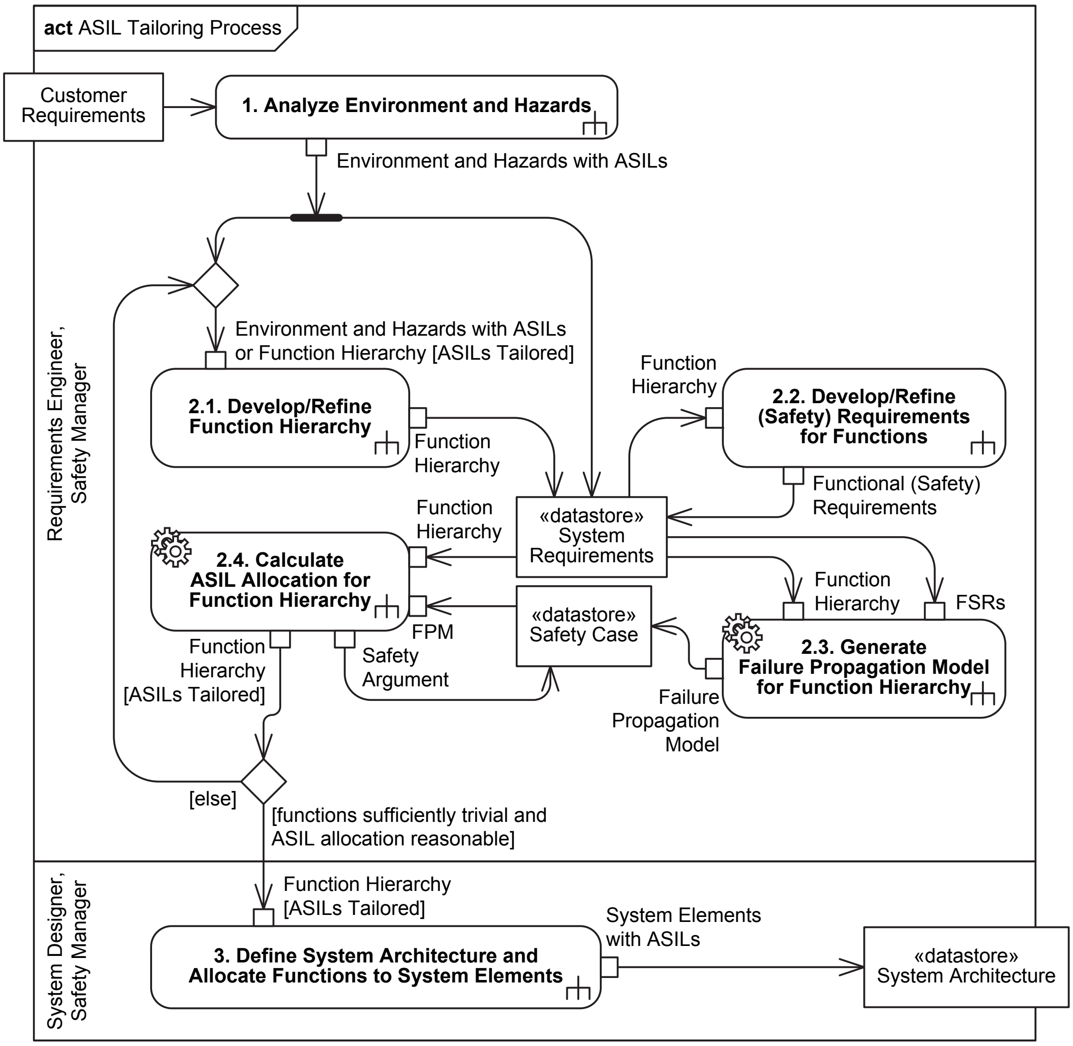

act ASIL Tailoring Process
Customer Requirements
1. Analyze Environment and Hazards
Environment and Hazards with ASILs
Environment and Hazards with ASILs or Function Hierarchy [ASILs Tailored]
Requirements Engineer, Safety Manager
2.1. Develop/Refine Function Hierarchy
Function Hierarchy
Function Hierarchy
2.2. Develop/Refine (Safety) Requirements for Functions
Functional (Safety) Requirements
Function Hierarchy
«datastore» System Requirements
2.4. Calculate ASIL Allocation for Function Hierarchy
FPM
«datastore» Safety Case
Function Hierarchy
FSRs
2.3. Generate Failure Propagation Model for Function Hierarchy
Failure Propagation Model
Function Hierarchy [ASILs Tailored]
Safety Argument
[else]
[functions sufficiently trivial and ASIL allocation reasonable]
System Designer, Safety Manager
Function Hierarchy [ASILs Tailored]
3. Define System Architecture and Allocate Functions to System Elements
System Elements with ASILs
«datastore» System Architecture

Figure 3.1: ASIL tailoring process steps and work products

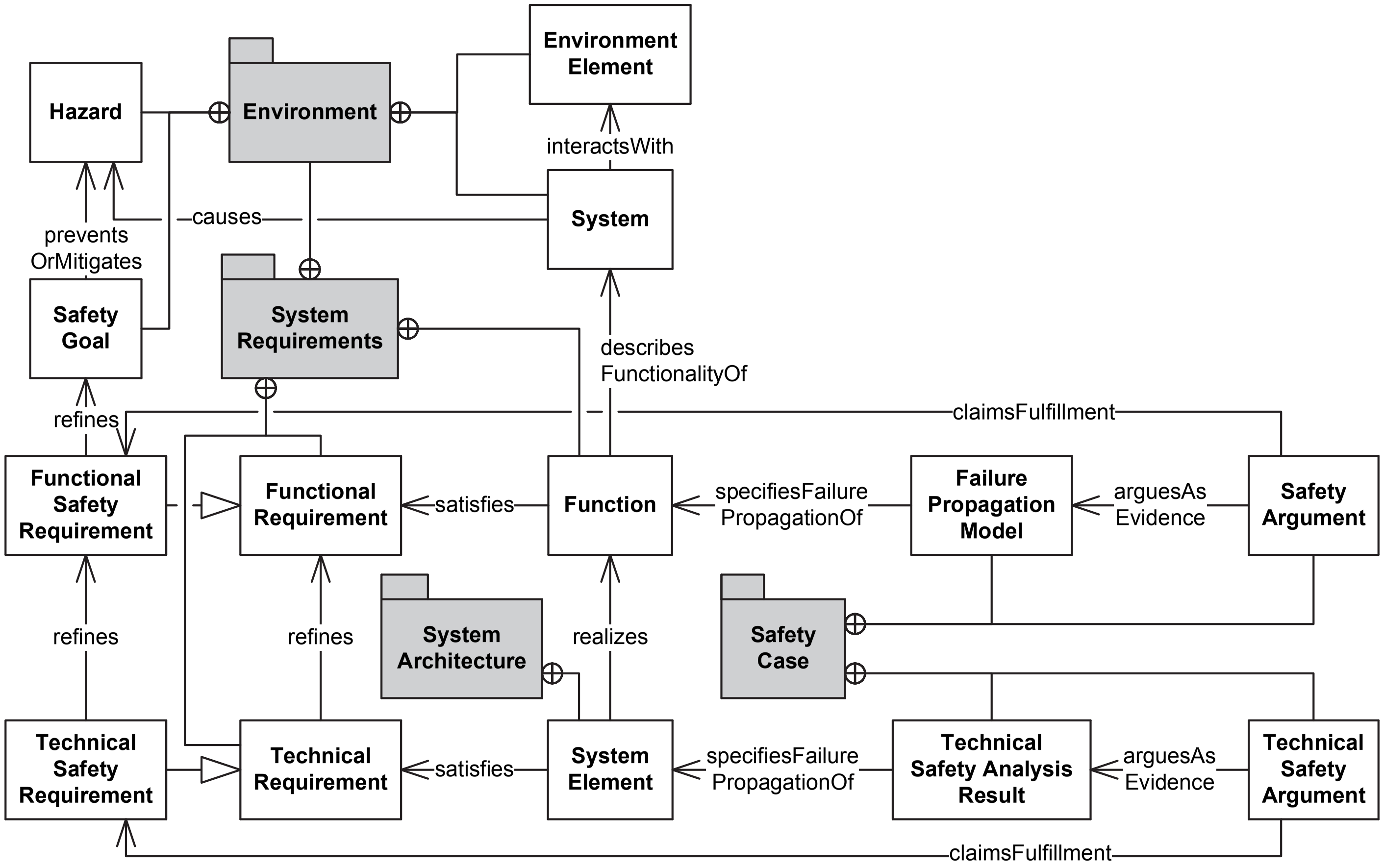


Figure 3.2: ASIL tailoring process work product relations

In Step 1 of the ASIL tailoring process, the CONSENS environment model for the SUD is specified and analyzed for hazards. The environment describes the interfaces between the system and its surrounding environment elements that it has to interact with (cf. Figure 3.2). In addition, we specify the hazards that could be caused by failures occurring on the outgoing interfaces (ports) of the system and their safety goals. The hazards and their ASILs are identified through a hazard analysis (cf. Definition 2.12 on page 19) on the system as a black-box and its environment [Int11d]. Step 1 is described in detail in Section 3.3.

In Step 2.1, the CONSENS function hierarchy is developed. It describes a breakdown of the overall functionality of the SUD, represented by a root function, into sub-functions until technical realizations can be found. Each function describes how a set of inputs is used to generate outputs to fulfill a sub-task of the overall system. Inputs and outputs are also decomposed onto sub-functions. In Step 2.2, the functional requirements and functional safety requirements of each function are specified using a formal modeling language. We use *Modal Sequence Diagrams* (MSDs) that are a formal, scenario-based requirements language based on UML sequence diagrams (cf. Section 2.7). Requirements specified in this language can automatically be processed to identify failure propagation paths. Step 2.1 and Step 2.2 are described in further detail in Section 3.4.

To find possible ASIL tailoring solutions and to verify that an applied tailoring is valid, safety analyses have to be performed. Thus, in the automated Step 2.3, the safety manager generates a failure propagation model to be used for safety analysis on the function hierarchy. We use *Component Fault Trees* (CFTs) that allow to specify decomposed fault trees for component-oriented models like the CONSENS function hierarchy (cf. Section 2.4.2). The failure propagation model is generated from the structure of the function hierarchy and the behavior described in the formal functional (safety) requirements. The failure propagation model is used to calculate valid, tailored ASIL values for the functions of the function hierarchy in the automated Step 2.4. The work products of the ASIL tailoring process are connected and traced to construct and maintain a safety argument in form of a model using the *Goal Structuring Notation* (GSN) as suggested by ISO 26262 [Int12b] (cf. Section 2.8). Step 2.3 and Step 2.4 are described in further detail in Section 3.5.

The steps 2.1 to 2.4 are iterated until the tailored ASILs of the functions meet the safety manager's expectations and the detail level of the functions is sufficiently trivial to be realized by technical system elements of the system architecture. In the final Step 3, the system architecture is defined by the system designer (as CONSENS active structure) and the functions are allocated to their realizing system elements. The functional (safety) requirements of the functions have to be refined into technical (safety) requirements that are satisfied by the system elements (cf. Figure 3.2). Afterward, safety analysis has to be repeated on the technical level of the system architecture to show that the ASIL tailoring on functional level remains valid on the system level. The result accordingly has to be documented in a technical safety argument (cf. Figure 3.2). Step 3 is described in detail in Section 3.6.

3.3 Analyzing the Environment and Hazards

In Step 1 of the ASIL tailoring process, the environment of the SUD is specified and used to identify possible hazards and their criticality (i.e., ASIL). Figure 3.3 shows the sub-actions of Step 1.

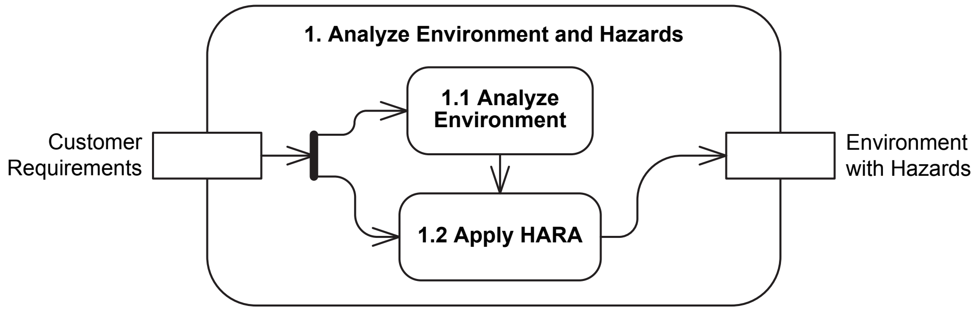


Figure 3.3: Process Step 1 - Analyze Environment and Hazards

Step 1.1 is an instantiation of the item definition phase of ISO 26262 (cf. phase 3-5 in Figure 2.6 on page 20). The requirements engineer uses the customer requirements to define the boundary of the system under development (i.e., the item) and to identify the environment elements that the system has to interact with.

Step 1.2 corresponds to the *hazard analysis and risk assessment* phase (HARA) of ISO 26262 (cf. phase 3-7 in Figure 2.6 on page 20). Here, the safety manager uses the customer requirements and the specified environment to identify hazards that could be caused by the system.

The customer requirements are the input for the whole development process and specifically for the system requirements analysis phase (cf. Section 2.1). They have to be satisfied by so-called system requirements which are the output of the system requirements analysis phase. The system requirements contain different types of requirements (cf. Figure 3.2). For each hazard a top-level safety requirement called safety goal is defined. The safety goals are later on refined into functional safety requirements in Step 2.1.

Figure 3.4 shows the environment with hazards for the EBEAS as refinement of the sketch in Figure 1.3 on page 4. The system EBEAS receives information from the environment elements Electronic Stability Control, Vehicle2X Communication, and Adaptive Cruise Control. The information received from the latter is depicted in the information flow specification AccEcu2EbeasEcu. The Adaptive Cruise Control notifies the EBEAS if an obstacle or a vehicle was detected (obstacleDetected, frontVehicleDetected), and sends the time in milliseconds until the detected object will be reached (timeToObstacle, timegapToFrontVehicle).

The EBEAS sends information to the environment elements Electronic Stability Control, Vehicle2X Communication, and Active Front Steering. These output interfaces are the locations where failures of the EBEAS can cause hazards.

The hazards are identified via HARA (cf. Section 2.2) in Step 1.2 and depicted in the bottom of Figure 3.4. A failure of the EBEAS could lead to a missing automatic hard braking of the vehicle via the Electronic Stability Control that controls the brakes (cf. H1 Hard Braking Omission). Similarly, a failure could lead to a missing automatic evasive steering via the

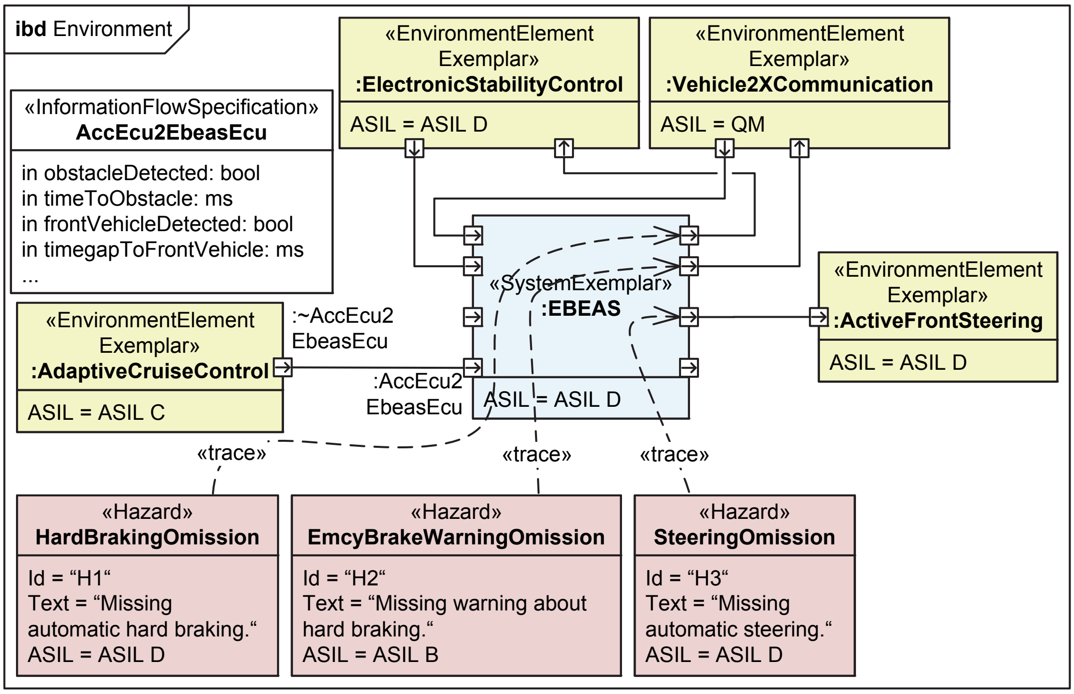


Figure 3.4: EBEAS system, environment, and hazards

Active Front Steering (cf. H3 Steering Omission). Furthermore, a failure of the EBEAS could lead to a missing warning about the hard braking of the vehicle to following vehicles via the Vehicle2X Communication (cf. H2 EmcyBrakeWarningOmission).

After the hazards were identified, their safety-criticality (i.e., ASIL) is determined as described in Section 2.2. The EBEAS is assigned with ASIL D as it has to be developed with the degree of rigor required for the hazard with the highest criticality (although there is hazard H2 with a lower ASIL). The surrounding environment elements in Figure 3.4 are assigned with predetermined ASIL values based on their previous usage without an EBEAS (cf. Section 1.1). If no ASIL tailoring can be applied, all environment elements would have to be assigned with the high ASIL of the EBEAS to assure system safety.

The specification of hazards and ASIL assignments in Figure 3.4 is enabled by the UML profile shown in Figure 3.5. Hazards are specified as specialization of SysML requirements. They thus inherit the attributes Id and Text as seen in Figure 3.4. Hazards additionally have the attributes of severity, exposure, and controllability as defined by ISO 26262 that together define the hazard's ASIL (cf. Section 2.2).

To assign an ASIL to structural elements like systems, environment elements, etc., the profile defines the stereotype SafetyClassifiedElement. It extends UML NamedElement to be applicable to structural SysML elements on type and part level and to MSD requirements. Safety Classified Elements can have two ASIL values: The ASIL attribute specifies the final determined ASIL of an element after ASIL tailorings have been applied. If ASIL decomposition was applied, the initialASIL specifies the maximum, undecomposed ASIL inherited from the hazards. In ISO 26262 this value is appended to the decomposed ASIL in parentheses (e.g., "ASIL B(D)") [Int11h]. For easier automatic computation these two values

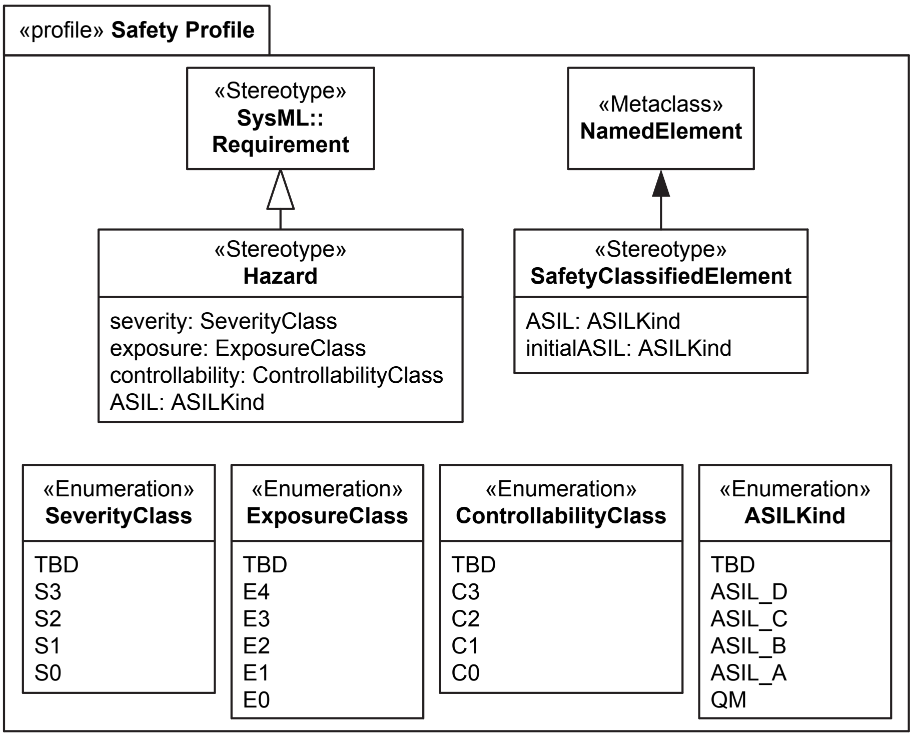


Figure 3.5: UML Profile for hazards and safety classified elements

are split in the profile. In the diagrams of this thesis, the parenthesis notation of ISO 26262 is appended to the ASIL attribute for better readability. The stereotype SafetyClassifiedElement is omitted in diagrams of this thesis for better readability, but the ASIL attribute is shown in a stereotype compartment or curly braces.

In addition, each attribute defined in the profile can be assigned a value of TBD (to be determined) to mark it as not yet fixed.

3.4 Specifying Functions and Requirements

In Steps 2.1 and 2.2 of the ASIL tailoring process, the function hierarchy and its (safety) requirements are developed. This section describes the process steps and the final work products produced in Steps 2.1 and 2.2. The detailed method how the work products are created is described in Chapter 4.

In Step 2.1, the function hierarchy is developed and refined. Figure 3.6 shows the subactions of Step 2.1. After the environment and hazards were specified in Step 1, the top-level function hierarchy is derived from the environment in Step 2.1.a). This action consists of the specification of the root function of the function hierarchy representing the full functionality of the SUD, and the functional abstraction of the SUD's technical interfaces to the environment into input and output information. The top-level function hierarchy is afterward used as input for Step 2.2. In following iterations of the Steps 2.1 to 2.4, the function hierarchy is refined in Step 2.1.b). This refinement is the decomposition of the functionality of single functions (like the top-level root function) into smaller functions on the next level of the function hierarchy, including their input and output information. Furthermore, the structure of the function hierarchy can be revised based on the ASIL tailoring results of Step 2.4.

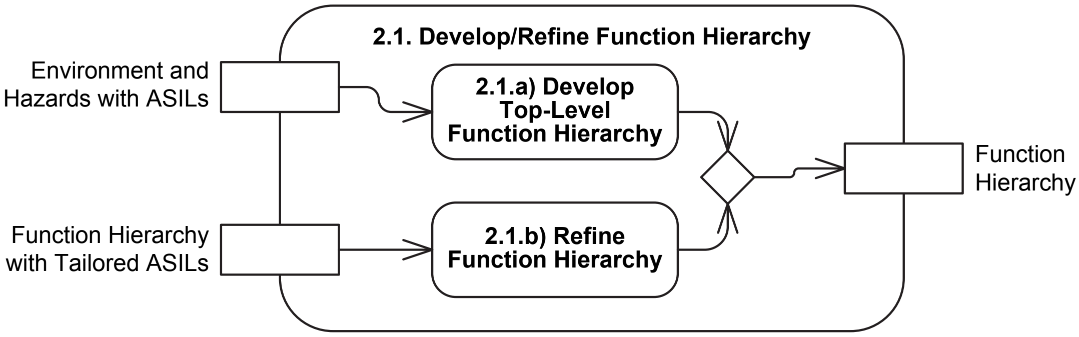

2.1. Develop/Refine Function Hierarchy
Environment and Hazards with ASILs
Function Hierarchy with Tailored ASILs
2.1.a) Develop Top-Level Function Hierarchy
2.1.b) Refine Function Hierarchy
Function Hierarchy

Figure 3.6: Process Step 2.1 - Develop/Refine Function Hierarchy

In Step 2.2, the functional (safety) requirements for the functions of the function hierarchy are developed and specified in form of MSDs (cf. Section 2.7). Figure 3.7 shows the subactions of Step 2.2. After the top-level function hierarchy was specified in Step 2.1.a), in Step 2.2.a) the according (safety) requirements for the top-level function are developed. In following iterations, the top-level requirements are refined adhering to the refinement of the function hierarchy in Step 2.1.b).

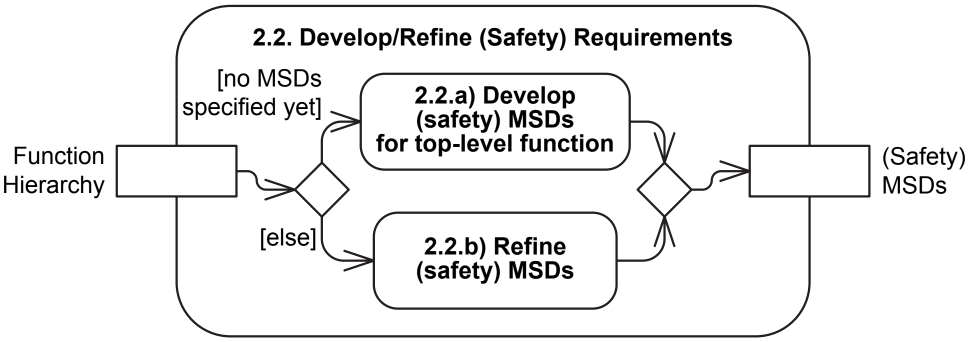

2.2. Develop/Refine (Safety) Requirements
[no MSDs specified yet]
[else]
Function Hierarchy
2.2.a) Develop (safety) MSDs for top-level function
2.2.b) Refine (safety) MSDs
(Safety) MSDs

Figure 3.7: Process Step 2.2 - Develop/Refine (Safety) Requirements for Functions

Figure 3.8 depicts an excerpt of the function hierarchy as result of Step 2.1. The figures 3.9 and 3.10 show functional safety requirements (FSRs) for functions of the function hierarchy as result of Step 2.2.

The top-level function FEBEAS in Figure 3.8 represents the whole functionality of the SUD EBEAS. Thus, it also inherits its ASIL value ASIL D. Its ports represent the input and output information received from and sent to its environment (cf. Figure 3.4). For instance, the port p1 typed by the interface Acc2Ebeas represents the information received from the adaptive cruise control. The top-level function FEBEAS with its ports and interfaces is developed in Step 2.1.a).

In Step 2.1.b), the top-level function FEBEAS is decomposed into the four functions Analyze Situation, Make Decision, Communicate With Other Vehicles, and Ensure

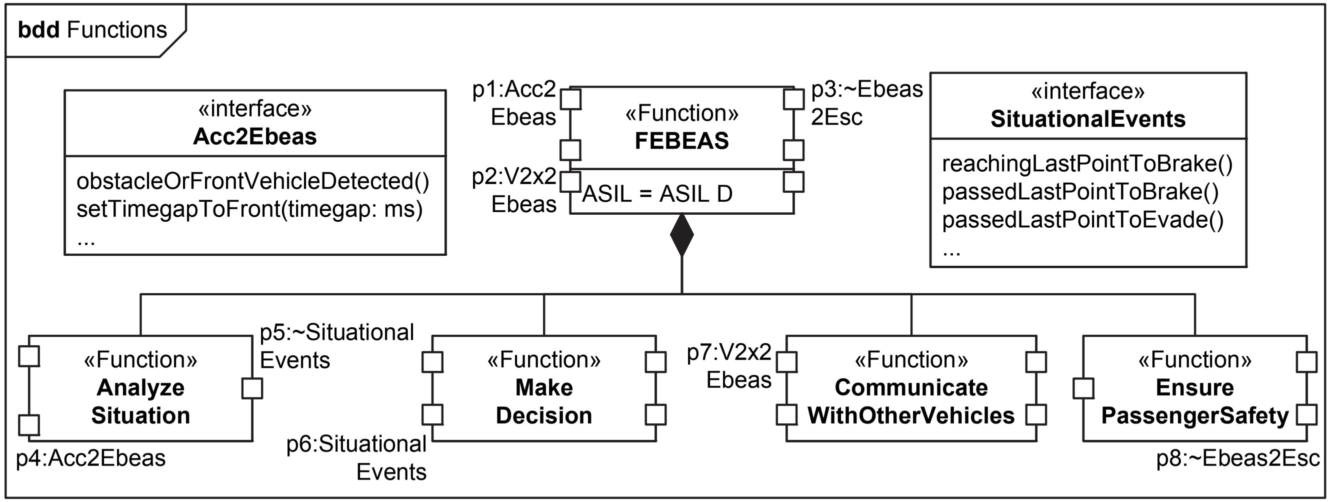


Figure 3.8: EBEAS function hierarchy

Passenger Safety. The purpose of the function Analyze Situation is to encapsulate the requirements for analyzing the current driving situation of the vehicle (e.g., distance to surrounding vehicles and ego vehicle speed). The function Make Decision describes requirements on making the decision to brake or evade based on the information provided by the functions Analyze Situation and Communicate With Other Vehicles. The latter's purpose is to provide information received from other vehicles via Vehicle-to-Vehicle communication and to send information to those vehicles. The function Ensure Passenger Safety encapsulates requirements concerning the execution of the decision made by Make Decision, i.e., activating the brakes or the evasive steering maneuver.

The information received/sent by the top-level function is decomposed among its subfunctions. It can be directly delegated to one function (like the port p1 to port p4) or decomposed onto different functions with separate ports. The decomposed functions also can exchange additional information (like Situational Events between p5 and p6) that is required to fulfill the superordinate functionality.

Interfaces in the function hierarchy describe a high-level information exchange as an abstraction from technical details like sample time, resolution, etc. The interface Acc2Ebeas is a functional abstraction of the technical interface AccEcu2EbeasEcu in Figure 3.4. For example, it merges the technically separated messages for detection of obstacles and vehicles into one information. The interface SituationalEvents describes information sent from the function Analyze Situation to the function Make Decision. The information exchanged is abstracted to events like reaching or passing the last point to safely brake. Technically, this information is complex to determine and consists of different sensor inputs in specific time intervals.

The requirements for each function of the function hierarchy are specified as MSDs in Step 2.2. The following figures show some examples. The top MSD in Figure 3.9 specifies a functional safety requirement FSR2.1 for the function Analyze Situation. If the time timegap until the leading vehicle would be hit is still long enough to safely brake (i.e., longer than time4Braking) but shorter than that time plus a certain threshold brakeThreshold, then the function Analyze Situation shall inform the function Make Decision that the last point to safely brake is about to be reached (cf. message reachingLastPointToBrake).

The bottom MSD in Figure 3.9 specifies a functional safety requirement FSR1.1 for the function Communicate With Other Vehicles. If the function receives information about an emergency brake warning from the vehicle in front, it shall inform the Make Decision function.

The two MSDs in Figure 3.10 specify functional safety requirements for the function Make Decision. The top MSD describes the requirement that if the Make Decision function is informed about an emergency brake warning from the vehicle in front (cf. FSR1.1 in Figure 3.9), it shall decide to activate the emergency braking procedure of the function Ensure Passenger Safety. Redundantly to this requirement, the bottom MSD in Figure 3.10 describes the requirement that the Make Decision function shall make the same decision if it is informed about reaching the last point to safely brake by the function Analyze Situation (cf. FSR2.1 in Figure 3.9).

The specification of the function hierarchy and the functional (safety) requirements as resulting from the Steps 2.1 and 2.2 is the input for the safety analysis on the functional (safety) requirements in the following Steps 2.3 and 2.4.

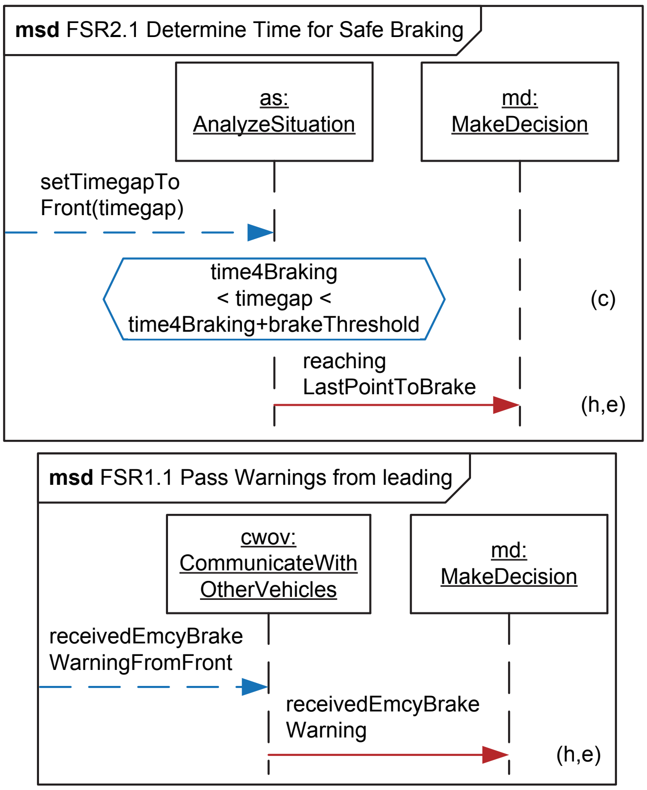

msd FSR2.1 Determine Time for Safe Braking
as: AnalyzeSituation
md: MakeDecision
setTimegapTo Front(timegap)
time4Braking < timegap < time4Braking+brakeThreshold
(c)
reaching LastPointToBrake
(h,e)
msd FSR1.1 Pass Warnings from leading
cwov: CommunicateWith OtherVehicles
md: MakeDecision
receivedEmcyBrake WarningFromFront
receivedEmcyBrake Warning
(h,e)

Figure 3.9: Requirements for functions Analyze Situation and Communicate With Other Vehicles

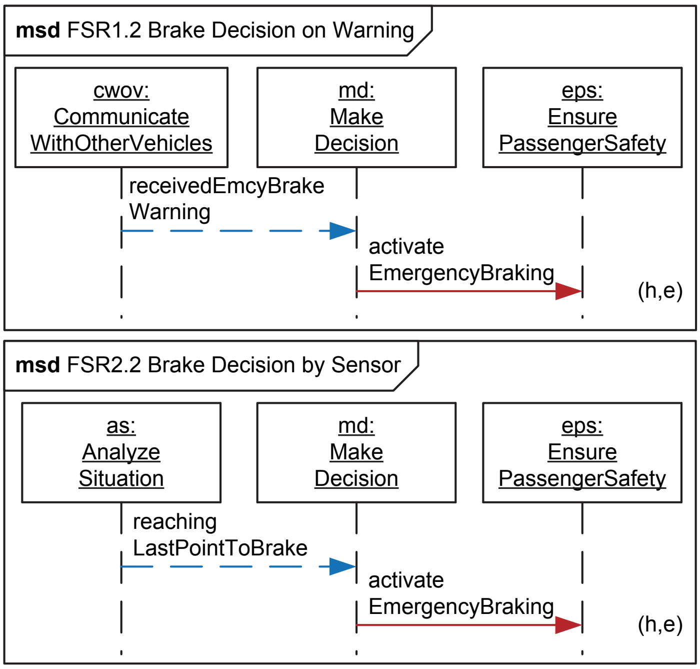

msd FSR1.2 Brake Decision on Warning
cwov: Communicate WithOtherVehicles
md: Make Decision
eps: Ensure PassengerSafety
receivedEmcyBrake Warning
activate EmergencyBraking
(h,e)
msd FSR2.2 Brake Decision by Sensor
as: Analyze Situation
md: Make Decision
eps: Ensure PassengerSafety
reaching LastPointToBrake
activate EmergencyBraking
(h,e)

Figure 3.10: Requirements for function Make Decision

3.5 Safety Analysis and ASIL Allocation

In Steps 2.3 and 2.4 of the ASIL tailoring process, a safety analysis is performed on the function hierarchy to allocate tailored ASIL values to the functions. Together with the ASIL allocation to functions, a safety argument is developed. This section describes the process steps and the final work products produced in Steps 2.3 and 2.4. The detailed methods how the work products are created are described in Chapters 5 and 6.

In Step 2.3, a failure propagation model in form of CFTs (cf. Section 2.4.2) is generated from the function hierarchy and the functional (safety) requirements. Figure 3.11 shows the sub-actions of Step 2.3. Initially, the safety manager has to specify which output failure modes of the top-level function lead to a certain hazard by a so-called Hazard CFT in Step 2.3.1. Afterward and in all following iterations, a failure propagation model in form of interleaved CFTs is automatically generated from the function hierarchy and its (safety) requirements in Step 2.3.2. These CFTs are generated from the hierarchical structure of the functions and the requirements specified as MSDs. In addition, they are connected to the manually specified Hazard CFTs. This enables safety analyses to find failure propagation paths through the function hierarchy leading to a hazard.

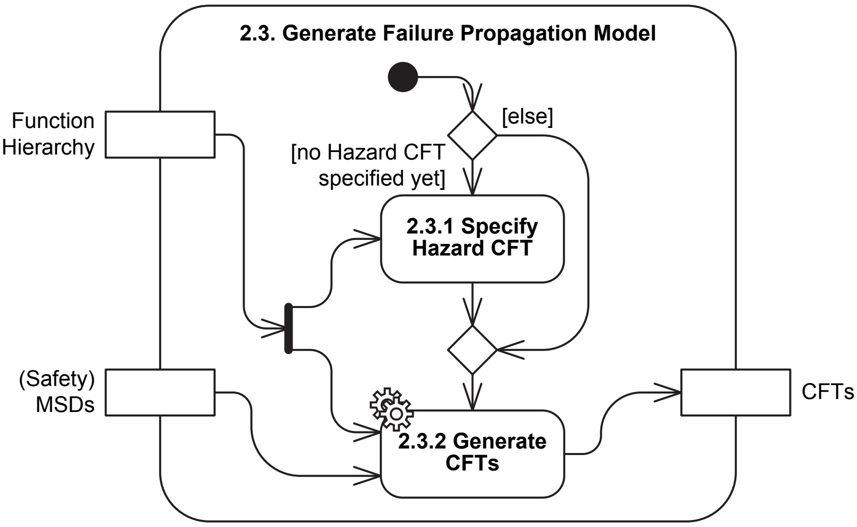


Figure 3.11: Process Step 2.3 - Generate Failure Propagation Model for Function Hierarchy

In Step 2.4, the failure propagation model is used to calculate valid ASIL tailorings and allocate ASILs to the functions of the function hierarchy. Figure 3.12 shows the automated sub-actions of Step 2.4. First, in Step 2.4.1 the ASIL of each hazard in Hazard CFTs is propagated along the failure propagation paths of the failure propagation model and allocated to input/output failure modes of the different sub-CFTs. Where possible, the ASIL is decomposed or reduced obeying the ASIL tailoring rules (cf. Section 2.5). As the CFTs mirror the structure of the function hierarchy, afterward in Step 2.4.2, the highest ASIL of each CFT is allocated to its corresponding function of the function hierarchy. Finally, in Step 2.4.3, the safety argument, in form of a GSN model (cf. Section 2.8), assuring the validity of applied ASIL tailorings in Step 2.4.1 is generated. This is done based on the calculated ASIL propagation paths and ASIL allocation of the Steps 2.4.1 and 2.4.2.

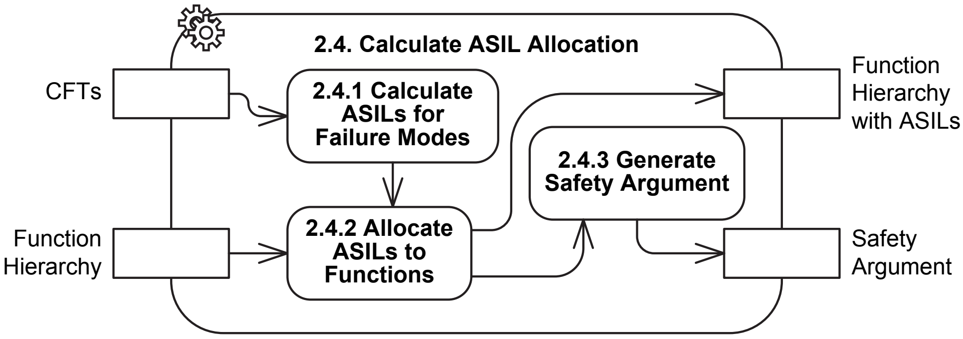

2.4. Calculate ASIL Allocation
CFTs
Function Hierarchy
2.4.1 Calculate ASILs for Failure Modes
2.4.2 Allocate ASILs to Functions
2.4.3 Generate Safety Argument
Function Hierarchy with ASILs
Safety Argument

Figure 3.12: Process Step 2.4 - Calculate ASIL Allocation for Function Hierarchy

Figure 3.13 depicts the failure propagation model for the function hierarchy in form of CFTs as result of Step 2.3. Figure 3.14 shows the function hierarchy with the resulting ASIL allocation as result of Step 2.4.

In the failure propagation model, the hazard Hard Braking Omission from Figure 3.4 is represented by the fault tree event :HardBrakingOmission inside the Hazard CFT HardBrakingOmissionCFT (cf. Figure 3.13). This Hazard CFT is the result of Step 2.3.1. The hazard is caused by an omission failure of the function FEBEAS represented by the CFT FEBEASCFT (automatically generated in Step 2.3.2). The omission failure propagates through the function Ensure Passenger Safety (details omitted in Figure 3.13) and is caused by an omission failure of the information activateEmergencyBraking of the function Make Decision.

The failure propagation of the CFT MakeDecisionCFT is automatically derived from the requirements in Figure 3.10. The information activateEmergencyBraking is not sent by the function Make Decision, if the function has an internal failure :Crash (e.g., because the system element realizing the function fails), or if the information reachingLastPointToBrake and receivedEmcyBrakeWarning are both omitted.

The information reachingLastPointToBrake is sent by the function Analyze Situation. So, the omission failure reachingLastPointToBrake:O is propagating from its CFT AnalyzeSituationCFT. Similarly, the omission failure receivedEmcyBrakeWarning:O is caused by the CFT CommunicateWithOtherVehiclesCFT.

The ASIL values annotated to failure modes, gates, and CFTs in Figure 3.13 result from Step 2.4.1. The hazard Hard Braking Omission is an ASIL D hazard. So, ASIL D propagates along the failure propagation path of the according omission failure into FEBEASCFT, through EnsurePassengerSafetyCFT, and into MakeDecisionCFT. After the AND gate the ASIL is decomposed, because both input failures reachingLastPointToBrake:O and receivedEmcyBrakeWarning:O have to occur at the same time, in order for the information activateEmergencyBraking not to be sent by Make Decision. According to the ASIL tailoring rules, the ASILs of the two CFTs AnalyzeSituationCFT and CommunicateWithOtherVehiclesCFT can be reduced to ASIL B(D) (cf. Section 2.5.2).

Figure 3.14 shows the function hierarchy from Figure 3.8 after Step 2.4.2. ASILs are allocated to the sub-functions as prescribed by the failure propagation model in Figure 3.13. The top-level function FEBEAS keeps the original ASIL D from the hazard Hard Braking Omission. Ensure Passenger Safety and Make Decision remain ASIL D functions as

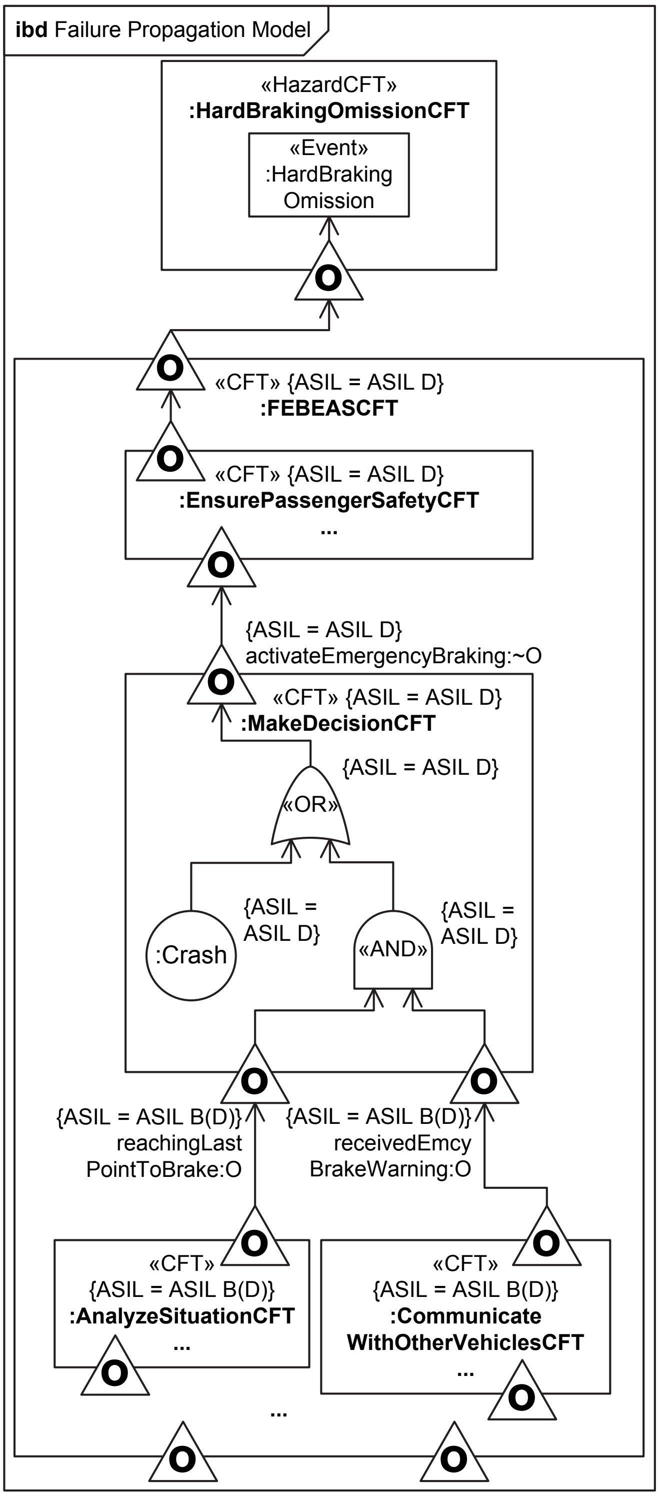


Figure 3.13: Failure propagation and ASIL allocation of function hierarchy (focused on Make Decision)

well. However, the ASILs of the functions Analyze Situation and Communicate With Other Vehicles could be reduced to ASIL B(D).

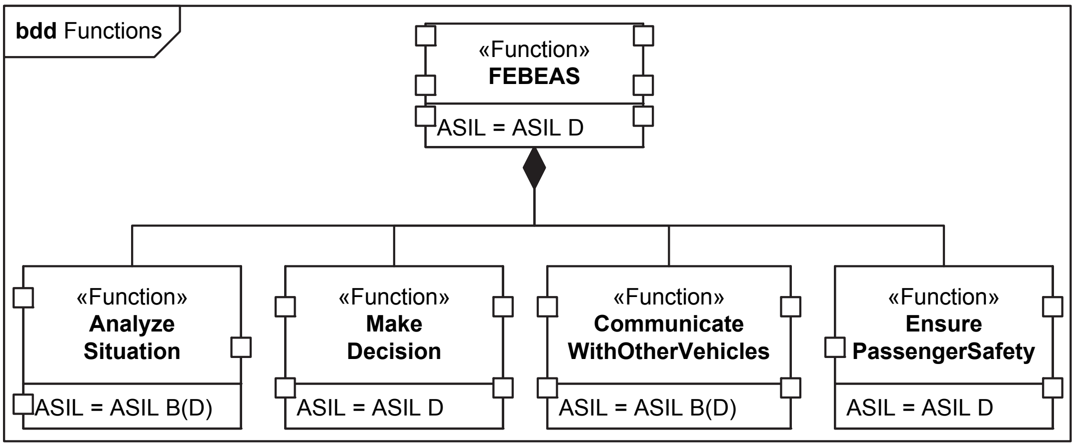


Figure 3.14: EBEAS function hierarchy with allocated ASILs

Based on the ASIL allocation to functions, a safety argument is automatically derived in Step 2.4.3. The safety argument for the calculated ASIL allocation is specified as GSN model in Figure 3.15. The hazards H1 and H3 from Figure 3.4 are addressed by the safety goal SG1 which is refined by functional safety requirements FSR1 and others. FSR1 is decomposed into the requirements FSR1.2, FSR2.2, and the tailored requirements FSR1.1 and FSR2.1 (cf. Figures 3.9 and 3.10).

FSR1.2 and FSR2.2 each make sure that the brake decision is made in the case of a hard braking of the leading vehicle. Thus, they are the reason for the AND gate in the CFT in Figure 3.13 that indicates a possibility of ASIL decomposition. This ASIL tailoring technique was applied on the requirements FSR1.1 and FSR2.1 that redundantly make sure that the information needed to make the brake decision is provided to the function Make Decision (i.e., the requirement FSR1.2 or FSR2.2).

For valid ASIL decomposition upon the functions Analyze Situation and Communicate With Other Vehicles that the requirements FSR1.2 and FSR2.2 belong to, the functions have to be independent (cf. Section 2.5.2). Thus, the three requirements FSR1.5, FSR1.6, and FSR1.7 for absence of cascading and common cause failures are added. That these requirements are fulfilled is shown by the CFT FEBEASCFT in Figure 3.13: there is no failure propagation path from one function's CFT to the other and no common failure mode as input to both function CFTs.

By executing the automated steps 2.3 and 2.4, ASIL values are allocated to the functions of the function hierarchy and tailored where possible. The validity of applied ASIL tailorings is assured by safety analysis on a failure propagation model and documented in a safety argument. The applied ASIL tailoring can then be reviewed by the safety manager. He might adapt and refine the function hierarchy and safety requirements to gain better ASIL tailoring results in following iterations of the steps 2.1 to 2.4. Once the function hierarchy and its ASIL allocation are considered final, the functions have to be allocated to system elements of the system architecture that realize their functionality in Step 3.

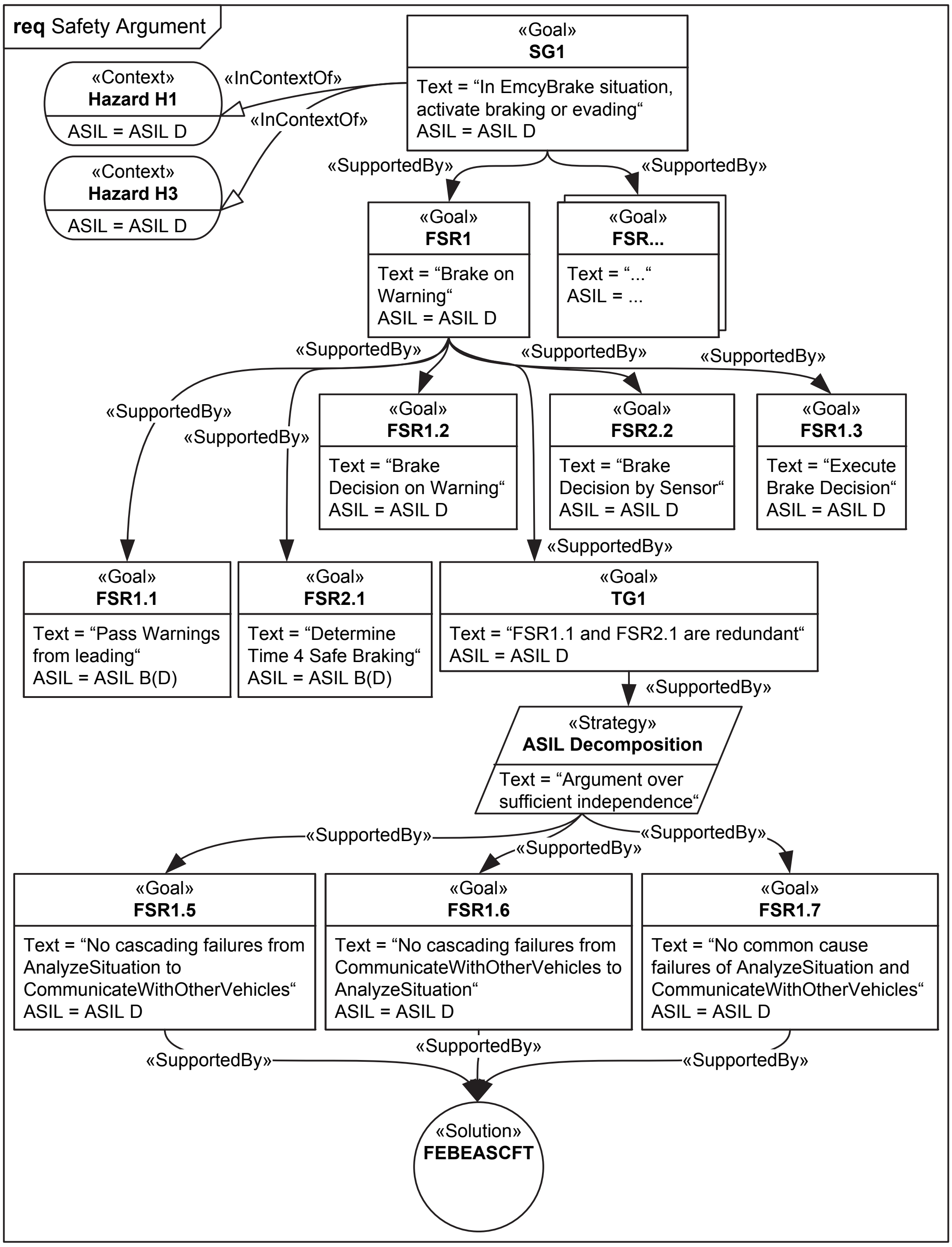


Figure 3.15: Safety argument for ASIL allocation

3.6 Allocating Functions to System Architecture

In Step 3 of the ASIL tailoring process, the functions of the function hierarchy are allocated to parts of the system architecture that realize them technically. In addition, the applied ASIL tailoring on the functional level is checked for validity on the technical level after allocation. Figure 3.16 shows the sub-actions of Step 3.

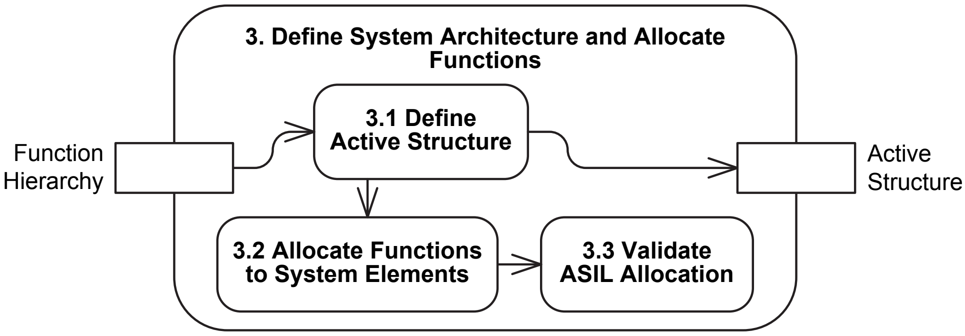

3. Define System Architecture and Allocate Functions
Function Hierarchy
3.1 Define Active Structure
Active Structure
3.2 Allocate Functions to System Elements
3.3 Validate ASIL Allocation

Figure 3.16: Process Step 3 - Define System Architecture and Allocate Functions to System Elements

In Step 3.1, the system designer defines the system architecture in form of a CONSENS active structure (cf. Section 2.6) based on the function hierarchy. This step is following the CONSENS systems engineering method.

In Step 3.2, the system designer links the system elements of the system architecture to the functions of the function hierarchy whose functionality (i.e., functional requirements) they realize to maintain traceability from requirements to realization [FHM12]. The system elements inherit the highest ASIL value of the functions they realize. In conjunction, the safety manager refines the functional safety requirements addressing the functions into technical safety requirements addressing the system elements.

In Step 3.3, the safety manager performs a safety analysis similar to that on the function hierarchy (cf. Section 3.5) on the system architecture to validate that the inherited ASIL values of the system elements are valid and independence claims hold. A safety analysis on CONSENS active structures is presented by Gausemeier et al. [GPD+09].

Figure 3.17 shows the system architecture of the EBEAS and the allocated functions of the function hierarchy from Figure 3.14. The interior of the EBEAS is defined in Step 3.1 by the system designer and allocated to the realized functions in Step 3.2. The ASIL B(D) functionality of the function Communicate With Other Vehicles is realized by software on the core :PerformanceCore and a bus interface :V2XBusInterface to the Vehicle2X Communication. The highly safety-critical ASIL D functionality of the functions Make Decision and Ensure Passenger Safety is realized on separated cores running in lockstep mode (:LockstepCores). The function Ensure Passenger Safety requires communication with actuator ECUs and thus is also partly realized by the separated redundant bus interfaces :ADASBusInterfaces. Safety analysis has shown that the function Analyze Situation can be developed ASIL B(D). However, it requires communication with the functionality provided by the Adaptive Cruise Control that is connected to the same bus as the more critical ECUs. Thus, the function is realized by the :ADASBusInterfaces and :LockstepCore as well. This may result in the function having to be realized according to ASIL D if it cannot be safely separated inside the two system elements (cf. ASIL tailoring by separation in Section 2.5.1). The safety analysis on the system architecture in Step 3.3 provides assurance for that. The Adaptive Cruise Control is assigned with ASIL B(D), because it only provides inputs for the Analyze Situation function and does not interfere with the ASIL D functionality.

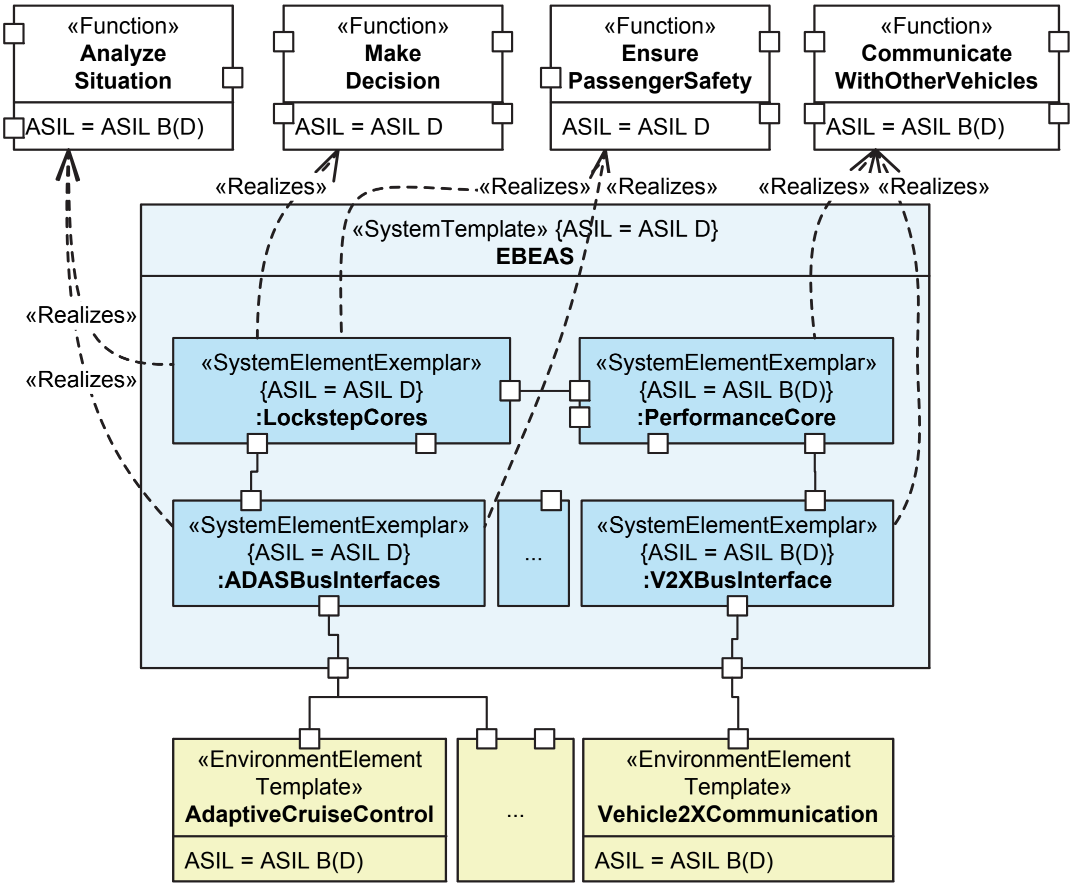


Figure 3.17: EBEAS system architecture and function allocation

By allocating the functions of the function hierarchy to the system architecture the traceability of functional (safety) requirements to the system architecture is maintained. Furthermore, the ASILs allocated to the functions provide the system designer with information how low the ASIL of system elements could be if no technical constraints (e.g., connection to buses) jeopardize the independence claims.

After the presented ASIL tailoring process is finished in Step 3 the discipline-specific development starts (cf. Section 2.1) where the technical (safety) requirements are refined into hardware and software (safety) requirements and the system architecture is refined into discipline-specific designs like the software architecture. On those work products, again safety analyses have to be repeated to validate that independence claims hold.

3.7 Assumptions & Limitations

The safety-criticality tailoring process presented in this chapter is applied to the automotive domain. It uses the terminology of automotive-specific standards and is embedded into an automotive-specific development process. However, safety-criticality levels like the ASIL also exist in other mechatronic systems domains (cf. Section 2.3). These domains have safety standards similar to ISO 26262 with comparable concepts and terminology. In fact, many of these standards are based on the same base standard IEC 61508 [Int10a]. The used automotive-specific development process is compliant to Automotive SPICE [Aut10] which is a specialization of the general software process maturity model SPICE [Int12a]. Thus, we assume that the presented tailoring process is also applicable to other mechatronic systems domains.

The tailoring process is embedded in a development process following the V model compliant to Automotive SPICE (cf. Section 2.1). An embedding into agile development processes like SCRUM [Sch97] has not been considered. However, there exists work discussing the applicability of SPICE to agile processes [LF09]. So, the migration to agile should be possible.

The automotive safety standard ISO 26262 has to be obeyed in the development of embedded systems consisting of software and hardware (electrics/electronics). The development of mechanics is not considered. Consequently, the ASIL tailoring process is limited to tailoring among functions that are realized by software or hardware.

ISO 26262 specifies requirements for the development of a so-called *Safety Element out of Context* (SEooC). “An SEooC is a safety-related element which is not developed for a specific item. This means it is not developed in the context of a particular vehicle” [Int12b]. A typical example of an SEooC is a platform project of a supplier that develops a system that shall be delivered to different customers with only slight changes to its interfaces. For such a system the concrete environment is not known initially. Thus, assumptions about the environment and specifically the hazards, failure propagation, and allocated ASIL values have to be made. In the presented process we assume that the environment of systems interacting with the SUD is known. We did not take SEooC development into account.

3.8 Related Work

Existing approaches for ASIL tailoring are applied in the phases of system architectural design or software design [APW$^+$14; APW$^+$13; PLB$^+$10][MAL$^+$12]. They calculate an ASIL allocation to subsystems of a technical architecture with known failure propagation and safety mechanisms already in place. They assume that ASIL tailoring measures (separation and decomposition) have already been applied – both on requirements and architectural level. Moreover, they do not consider the documentation of a corresponding safety argument.

Based on the EAST-ADL [EAS13] a safety engineering process has been defined [SCL$^+$10]. It distinguishes a functional and technical level but does not integrate an ASIL tailoring process on functional safety requirements.

Beckers et al. present a process for derivation of functional safety requirements [BCF$^+$14]. They specify textual requirements in UML class attributes using a UML profile (not extending SysML). They do not use a function hierarchy. The functional safety requirements are arbitrarily structured and directly connected to safety goals. ASIL decomposition is part of their process but as a completely manual step.

In summary, the existing approaches do not provide a process for ASIL tailoring in the requirements analysis phase. They rather apply ASIL calculation and allocation on system and software architectures based on the manually specified failure propagation of subsystems. In addition, the approaches do not consider the documentation of safety arguments for applied ASIL tailorings.

3.9 Conclusion

This chapter introduces the ASIL tailoring process working on functional safety requirements. Figure 3.18 gives an overview of the EBEAS example work products in their final state after the process was executed. The first step of the process is to specify the system's environment, identify hazards that the system can cause and determine their ASIL values. In Figure 3.18, the top-left diagram Environment sketches the environment and hazards of the EBEAS. There is an ASIL D hazard H1 named Hard Braking Omission that is traced to an outgoing port of the SUD.

Afterward, safety requirements are derived for the functions that the system shall realize. The Function Hierarchy diagram in Figure 3.18 shows the break down of the overall functionality FEBEAS of the SUD into sub-functions like Make Decision. The diagram FSR1.2 Brake Decision on Warning shows an example of a safety requirement for the function Make Decision. If it receives the information of an emergency brake warning receivedEmcyBrakeWarning, it shall decide to activate Emergency Braking.

Based on the structure of the function hierarchy and their requirements (especially functional safety requirements), an automated safety analysis is performed to allocate tailored ASIL values to the functions. The Failure Propagation Model diagram in Figure 3.18 describes which failures propagating through the function hierarchy lead to the Hard Braking Omission hazard. The CFT Make Decision CFT in the middle of the diagram describes the failure propagation through the function Make Decision. It shows that the function has to Crash or two input failures have to occur at the same time (cf. the AND gate) in order for an output failure to lead to the hazard. The AND gate in the CFT Make Decision CFT of Figure 3.18 indicates a possibility for ASIL tailoring. Only if its two ingoing failures occur simultaneously, an outgoing failure will propagate on to the OR gate. The analysis of the failure propagation model provides assurance that the ASIL D propagating from the hazard Hard Braking Omission can be decomposed into the two ASIL B(D) input failures of Make Decision CFT. Thus, the two connected CFTs and their corresponding functions inherit that lower ASIL.

The validity of applied ASIL tailorings is automatically documented in form of a model-based safety argument in a safety case. The Safety Argument diagram in Figure 3.18 documents an argument for the validity of the applied ASIL tailoring. It connects the hazard H1, the refined and decomposed safety requirements (i.e., MSDs), and the evidence for safety requirement fulfillment in form of CFTs.

The function hierarchy and the functional safety requirements are refined by iteration and finally allocated to the technical system architecture. In the example System Architecture diagram of Figure 3.18 the EBEAS is comprised of four system elements. They are

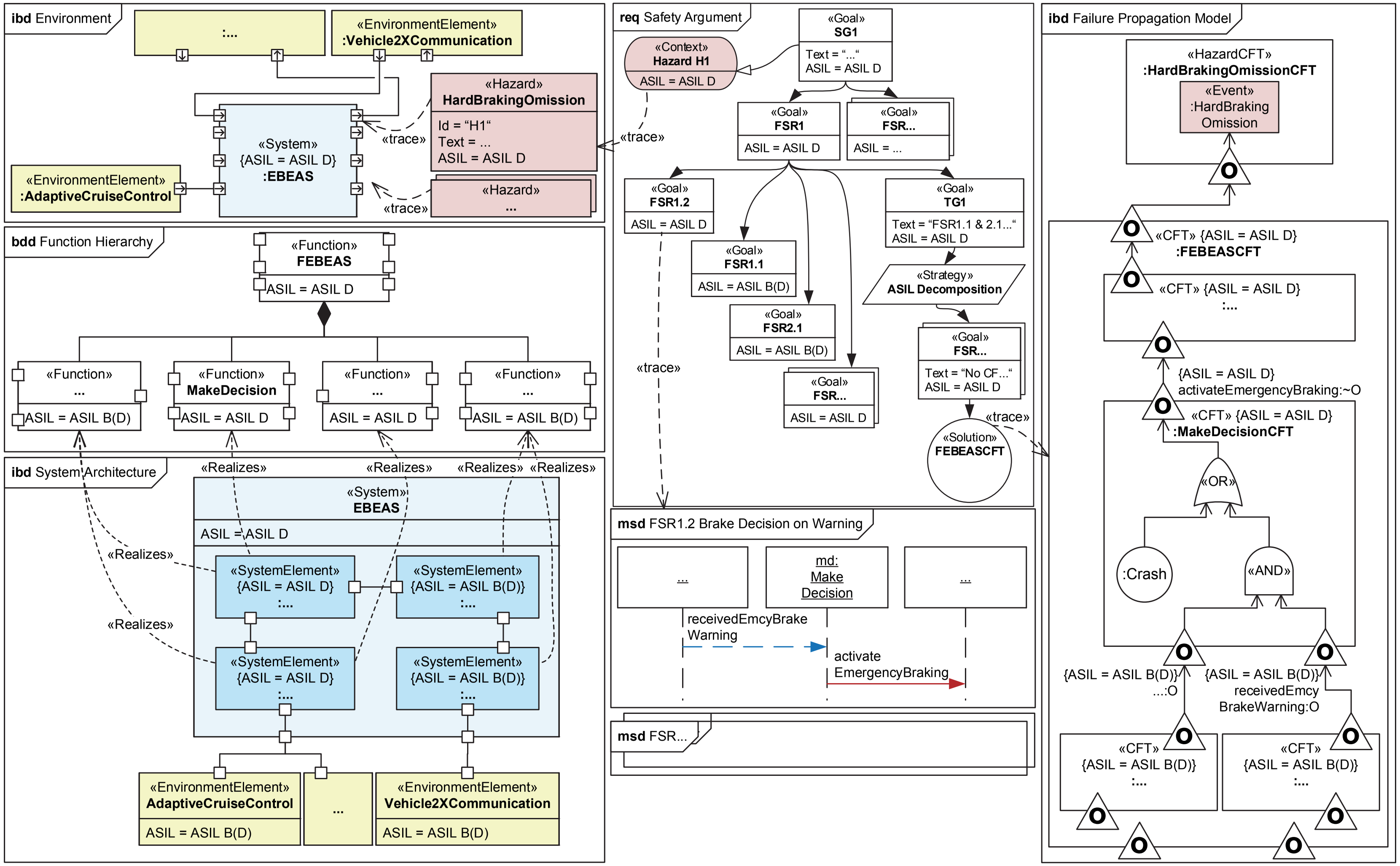


Figure 3.18: Sketch of ASIL tailoring process work products

connected to the functions that they realize and inherit the highest ASIL of the realized functions.

This systematic ASIL tailoring process allows to plan safety measures and required effort early in the development process (i.e. during system requirements analysis). Furthermore, its automated steps remove error-prone and time-consuming manual tasks of the safety manager. This fosters the application of ASIL tailoring already during requirements analysis which is required due to the increasing complexity of mixed-criticality systems.

In the following Chapter 4, the method for specifying the function hierarchy and corresponding functional safety requirements is described in detail. Chapter 5 contains the details of the automated safety analysis and ASIL allocation on the function hierarchy. In Chapter 6 the automatic documentation of safety arguments is elaborated.

4

Specifying Formal Functional Safety Requirements

In Steps 2.1 and 2.2 of the ASIL tailoring process (cf. Figure 4.1), the function hierarchy and its (safety) requirements are developed and refined. This chapter describes the details of these process steps and the used methods.

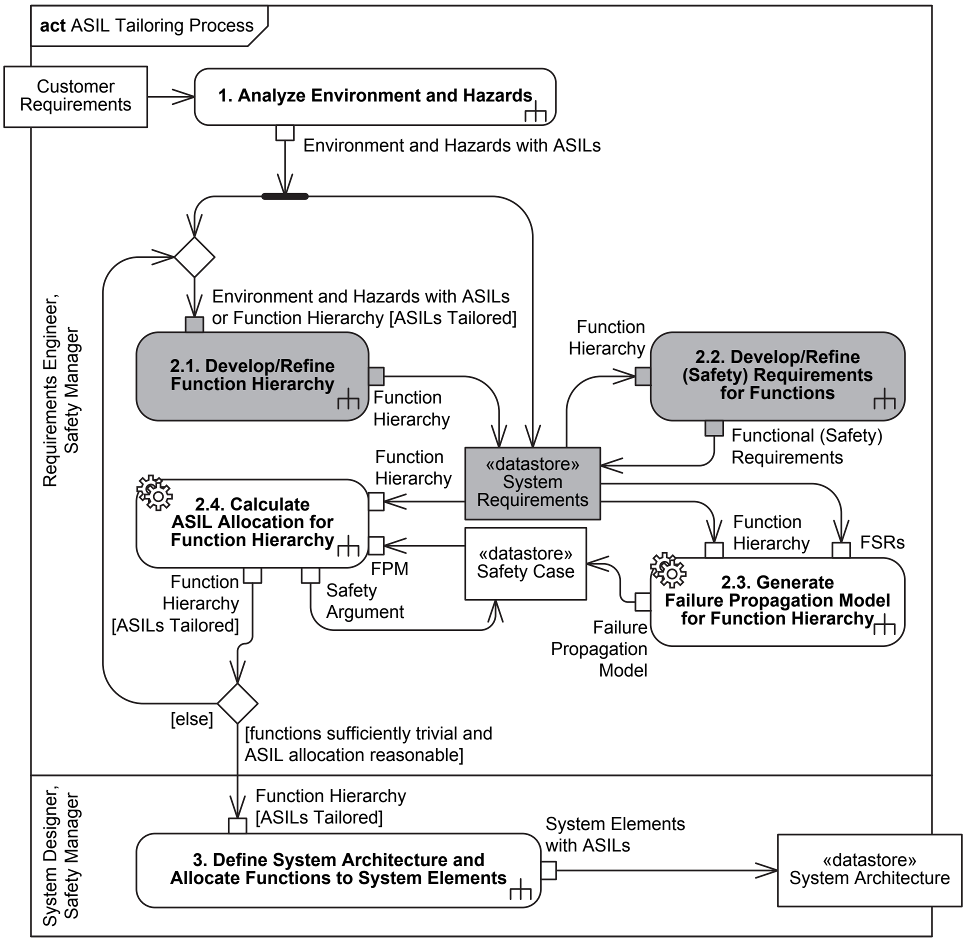

act ASIL Tailoring Process
Customer Requirements
1. Analyze Environment and Hazards
Environment and Hazards with ASILs
Environment and Hazards with ASILs or Function Hierarchy [ASILs Tailored]
Requirements Engineer, Safety Manager
2.1. Develop/Refine Function Hierarchy
Function Hierarchy
Function Hierarchy
2.2. Develop/Refine (Safety) Requirements for Functions
Functional (Safety) Requirements
Function Hierarchy
«datastore» System Requirements
2.4. Calculate ASIL Allocation for Function Hierarchy
FPM
«datastore» Safety Case
Function Hierarchy
FSRs
2.3. Generate Failure Propagation Model for Function Hierarchy
Failure Propagation Model
Function Hierarchy [ASILs Tailored]
Safety Argument
[else]
[functions sufficiently trivial and ASIL allocation reasonable]
System Designer, Safety Manager
Function Hierarchy [ASILs Tailored]
3. Define System Architecture and Allocate Functions to System Elements
System Elements with ASILs
«datastore» System Architecture

Figure 4.1: ASIL tailoring process with highlighted contents of Chapter 4

The function hierarchy describes and decomposes the system's functionality. Requirements on that functionality (functional requirements and functional safety requirements) are specified using Modal Sequence Diagrams. These requirements consider the occurrence, order, and timing of information exchange between functions. Requirements on the functions of other classes can be specified in natural language and are linked to the functions. Altogether, the function hierarchy and corresponding (safety) requirements are consolidated in a system requirements specification.

This chapter is structured as follows. First, Section 4.1 summarizes the scientific contributions of the formal specification of functional (safety) requirements. Section 4.2 describes how those requirements and related work products fit in a system requirements specification. Section 4.3 contains the details of the ASIL tailoring process steps 2.1 and 2.2. Section 4.4 explains the semantics and use of Modal Sequence Diagrams for specifying different classes of functional (safety) requirements. Section 4.5 describes the integration of MSD-based and natural language requirements. Section 4.6 lists assumptions and limitations of the formal specification of functional (safety) requirements. In Section 4.7, related work is discussed. The final Section 4.8 concludes this chapter.

4.1 Contributions

The contributions of this chapter can be summarized as follows:

- A model- and scenario-based, hierarchically decomposed specification of functional safety requirements to cope with the complexity of CPS functionality and requirements (cf. Challenge C1 in Section 1.2).
- A formal specification of functional safety requirements that enables automated safety analysis and ASIL allocation on requirements level (cf. Challenge C2 in Section 1.2).
- A catalog of formal, model-based requirement patterns for functional requirements concerning chronological succession, real-time, and safety. It supports requirements engineers and safety managers in building a high-quality requirements specification for a safe system.
- The integration of model-based safety requirements with natural language safety requirements that fosters the completeness and consistency of requirements specifications whilst providing flexibility to specify requirements in the respectively most suitable way.

4.2 System Requirements Specification Contents

The ASIL tailoring process presented in Chapter 3 is integrated into the SPICE phase *system requirements analysis*. The purpose of that phase is to "transform the defined customer requirements into a set of desired system technical requirements that will guide the design of the system" [Aut10]. This set of requirements shall be documented in a so called *System Requirements Specification* (SyRS). The required contents and their structure are defined in the requirements engineering standard ISO 29148 [Int11i]. Hence, those work products of the ASIL tailoring process that we consider part of the system requirements have to adhere to this standard as well. Figure 4.2 shows how the requirements work products used in the ASIL tailoring process are structured in accordance with ISO 29148 and ISO 26262.

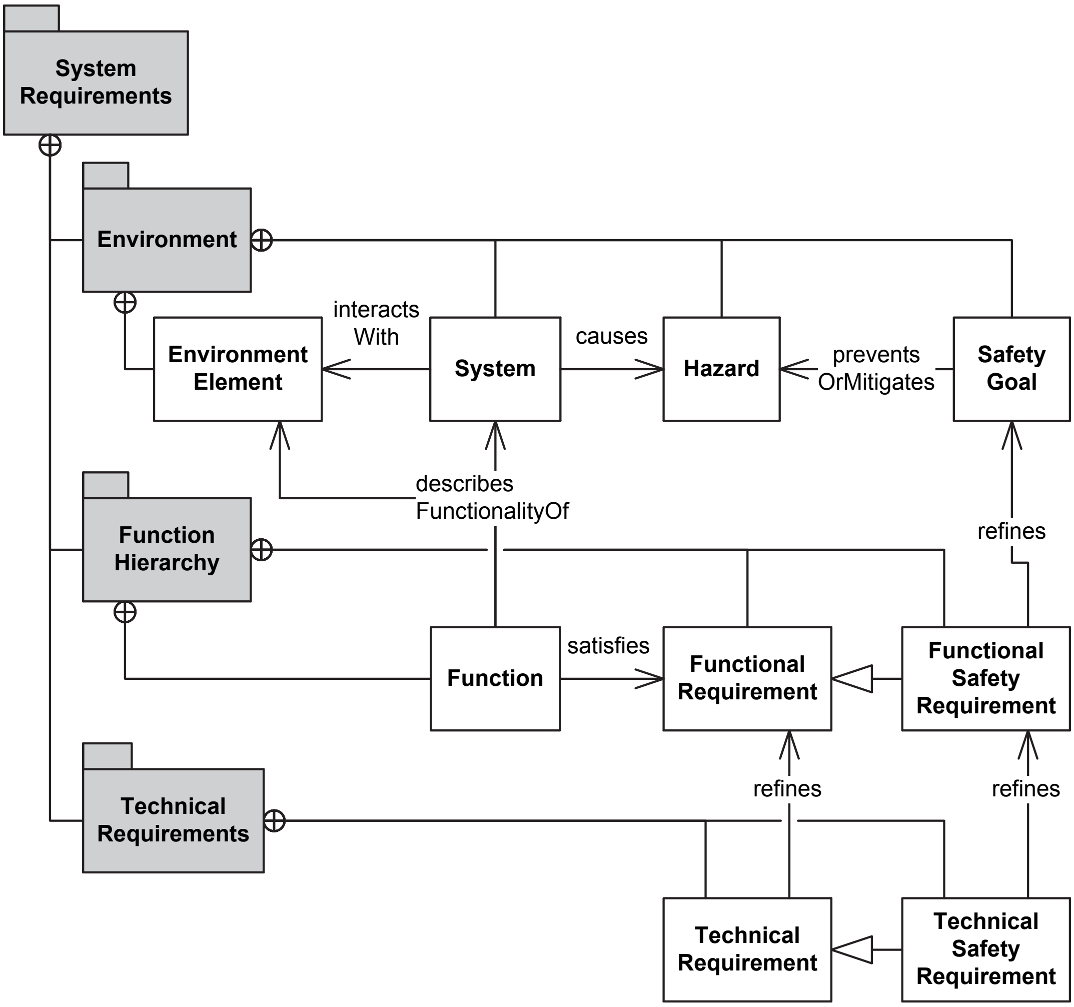


Figure 4.2: Contents of the System Requirements Specification

ISO 29148 describes a so-called *system context* that shall contain "the major elements of the system [...] and how they interact [...], defining all significant interfaces crossing the system's boundaries" [Int11i]. The environment specified in Step 1 of the ASIL tailoring process (cf. Section 3.3) specifies the system's boundary, its technical interfaces, and surrounding environment elements that it has to interact with (cf. Section 2.6). Thus, the Enviroment fits as system context.

ISO 26262 distinguishes between requirements on different abstraction levels (cf. Section 2.2). We consider safety goals, functional safety requirements and technical safety requirements as part of the SyRS. Safety goals are top-level safety requirements specified for each identified hazard that lead to the functional safety requirements [Int11d]. Hazards are caused by failures on the boundary of the system. Thus, we document safety goals and hazards together with the environment.

ISO 29148 defines so-called *system functions* that shall describe “major system capabilities, conditions, and constraints” [Int11i]. We specify system capabilities as functions in the function hierarchy.

ISO 29148 prescribes no dedicated section for safety requirements, and ISO 26262 states that safety requirements can be documented together with other requirements if they are “unambiguously identifiable as safety requirements” (e.g., by an ASIL attribute) [Int11g]. Thus, we document safety requirements as part of the SyRS together with other requirements.

We specify functional safety requirements together with other functional requirements as MSDs structured by the functions of the function hierarchy. This matches the organizational approach for requirements structuring *functional hierarchy* suggested by ISO 29148 [Int11i].

Technical safety requirements shall be allocated to elements of the system architecture [Int11e]. We document technical safety requirements together with other technical requirements textually, and group them in a *system requirements* section (that we call Technical Requirements in Figure 4.2) as suggested by ISO 29148.

4.3 Systematic Development and Refinement of Functional Safety Requirements

This section describes the details about the substeps of the Steps 2.1 and 2.2 of the ASIL tailoring process. In Step 2.1, the function hierarchy is developed and refined. Figure 4.3 shows the sub-actions of Step 2.1. In Step 2.2, the functional (safety) requirements for the functions of the function hierarchy are developed and specified in form of Modal Sequence Diagrams (cf. Section 2.7). Figure 4.4 shows the sub-actions of Step 2.2. After the environment and hazards were specified in Step 1, the top-level function hierarchy is derived from the environment in Step 2.1.a). This action consists of the specification of the root function of the function hierarchy representing the full functionality of the SUD, and the functional abstraction of the SUD’s technical interfaces to the environment into input and output information. The top-level function hierarchy is afterward used as input for Step 2.2.a). In Step 2.2.a), the according (safety) requirements for the top-level function are developed. In following iterations of the process steps, the function hierarchy is refined in Step 2.1.b). This refinement is the decomposition of the functionality of single functions (like the top-level root function) into smaller functions on the next level of the function hierarchy, including their input and output information. Furthermore, the structure of the function hierarchy can be revised based on the ASIL tailoring results of Step 2.4 (cf. Chapter 5). Afterward, in Step 2.2.b), the functional (safety) requirements are refined adhering to the refinement of the function hierarchy.

Section 4.3.1 explains the systematic derivation of the top-level function hierarchy from the environment (Step 2.1.a)). Section 4.3.2 describes the specification of (safety) requirements on the function hierarchy using MSDs (Step 2.2.a)). Section 4.3.3 details the refinement steps for the function hierarchy and requirements (Steps 2.1.b) and 2.2.b)).

4.3.1 Deriving the Top-Level Function Hierarchy from the Environment

The CONSENS environment model specifies the system’s boundary, its technical interfaces, and surrounding elements that it has to interact with (cf. Section 2.6). The system is

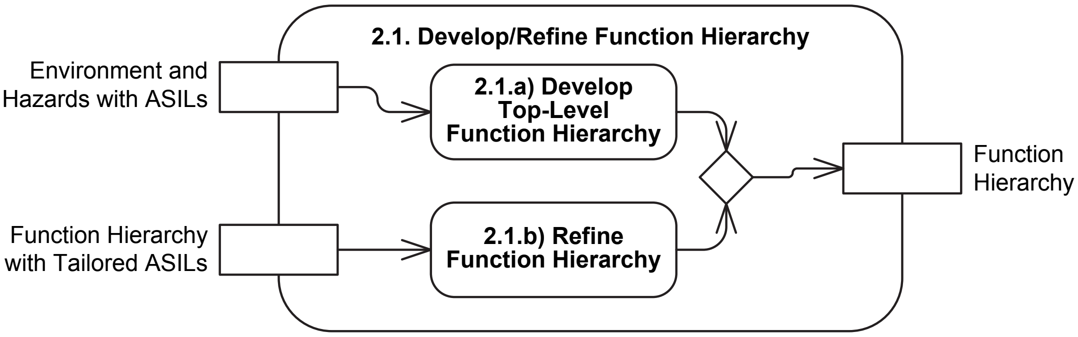

2.1. Develop/Refine Function Hierarchy
Environment and Hazards with ASILs
Function Hierarchy with Tailored ASILs
2.1.a) Develop Top-Level Function Hierarchy
2.1.b) Refine Function Hierarchy
Function Hierarchy

Figure 4.3: Process Step 2.1 - Develop/Refine Function Hierarchy

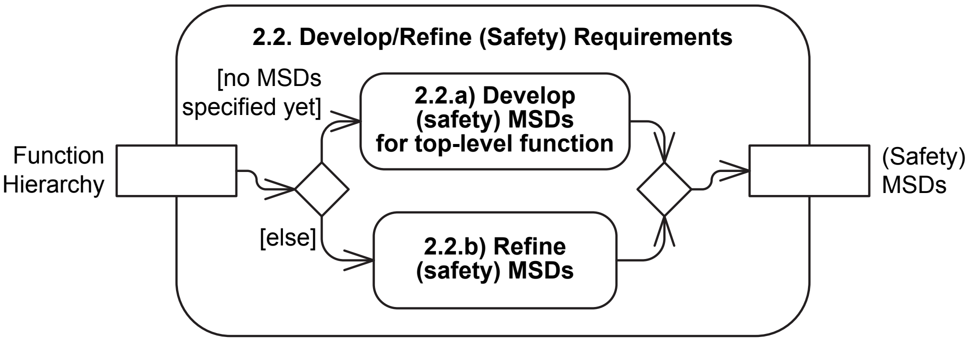

2.2. Develop/Refine (Safety) Requirements
[no MSDs specified yet]
[else]
Function Hierarchy
2.2.a) Develop (safety) MSDs for top-level function
2.2.b) Refine (safety) MSDs
(Safety) MSDs

Figure 4.4: Process Step 2.2 - Develop/Refine (Safety) Requirements for Functions

considered as a black box. Its internals are specified during the system architectural design phase in the system architecture. To come up with an adequate system architecture that fulfills all requirements, we use a function hierarchy as an intermediate step. It is a means to cope with complexity by decomposing the required functionality that has to be realized by the system and its architecture.

Figure 4.5 shows the detailed environment of the EBEAS (refinement of Figure 3.4 on page 44) as result of ASIL Tailoring Process Step 1 (cf. Section 3.3). The internal block diagram in the bottom shows the system and its connections to the environment elements via ports typed by the interfaces shown in the block definition diagram in the top.

The Vehicle2X Communication sends and receives messages to and from other vehicles surrounding the ego vehicle. Information that is relevant for the EBEAS is passed on to it (e.g., an emergency brake warning from a vehicle in front). In addition, the EBEAS commands the Vehicle2X Communication to send messages to certain vehicles (e.g., emergency brake or evade requests and warnings). The Adaptive Cruise Control informs the EBEAS about detected obstacles and vehicles in front, and continuously provides information about the time it takes to reach those. The Electronic Stability Control continuously provides the EBEAS with information about the current ego vehicle velocity and the maximal possible deceleration (cf. maxDecel of interface EscEcu2EbeasEcu in Figure 4.5). The Electronic Stability Control controls the vehicle's brakes. Hence, in case the EBEAS decides to execute emergency braking, it commands the Electronic Stability Control to perform a hard braking. The Active Front Steering can control the steering column. So, in case the EBEAS decides to

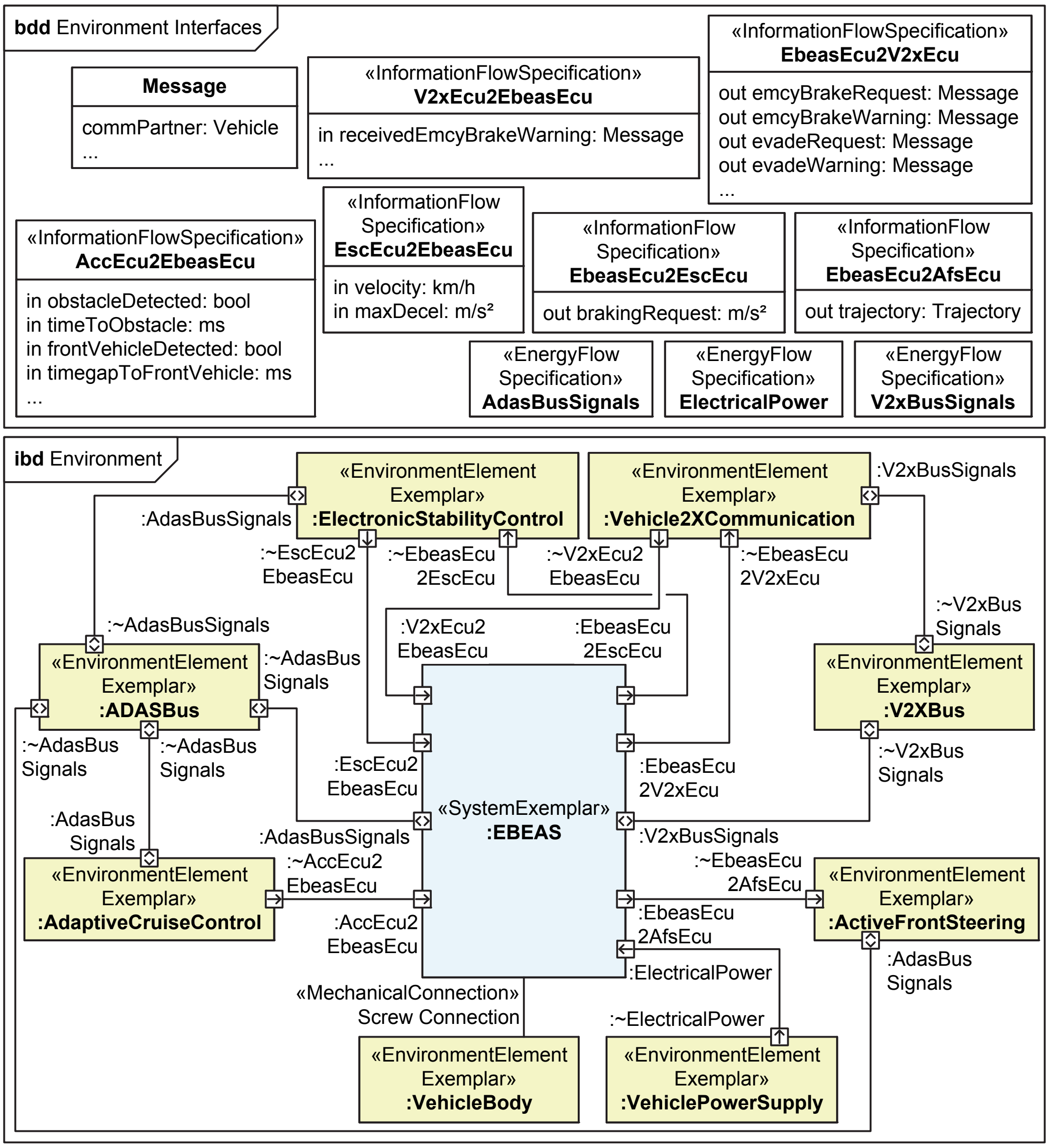


Figure 4.5: Environment of the EBEAS with Interfaces

evade, it commands the Active Front Steering to steer continuously according to a calculated trajectory to perform a lane change maneuver.

Technically, the Vehicle2X Communication is connected to the EBEAS via the V2X Bus (cf. energy flow V2xBusSignals). So, all information exchange specified as information flow between the Vehicle2X Communication and the EBEAS is actually transferred over that bus. All other ECUs are connected to the ADAS Bus. Thus, their information exchange is transfered over that bus. The EBEAS receives its electrical power (cf. energy flow ElectricalPower) from the Vehicle Power Supply. Moreover, the EBEAS is mechanically connected to the Vehicle Body via screws (cf. mechanical connection Screw Connection).

Deriving the top-level function hierarchy from the environment (cf. Step 2.1.a) in Figure 4.3) consists of the specification of the root function of the function hierarchy representing the full functionality of the SUD, and the functional abstraction of the SUD's technical environment interfaces into input and output information. To support the derivation process, the CONSENS function hierarchy (cf. Section 2.6) had to be extended: The CONSENS function hierarchy does not specify ports or interfaces for functions. This means, there is no direct traceability from the technical ports and interfaces in the environment model to the functions of the function hierarchy. In addition, information flow through the function hierarchy cannot be analyzed (e.g., for failure propagation). Thus, we add the use of SysML ports and interfaces to the function hierarchy. Furthermore, the CONSENS function hierarchy does not specify functions on the same level as the root function to represent the functionality of environment elements. However, this is needed to specify model-based requirements on the behavior of the root function based on input and output information from/to the environment's functionality. Thus, we also use CONSENS functions as a functional abstraction of the environment elements that communicate with the root function via ports and interfaces.

The derivation of the top-level function hierarchy with these extensions consists of the following steps. For illustration, Figure 4.6 shows the top-level function hierarchy of the EBEAS derived from the environment in Figure 4.5.

1. **Derive Root Function:** The SUD (the CONSENS system template) is represented by a CONSENS function. Its name is prefixed with an 'F' to distinguish it from the system template. In the example, the EBEAS is represented by the root function FEBEAS.

2. **Derive Functions for Environment Elements:** Each environment element that communicates with the SUD via information flow is added as a top-level function (with an 'F'-prefixed name). In the example, this results in the ADAS Bus, V2X Bus, Vehicle Power Supply, and Vehicle Body not being transformed.

3. **Derive Ports:** Each port of the system that is typed by an information flow is represented by a port on the system's root function. The ports of the environment elements that are connected to these system ports are also transformed to ports of the corresponding functions. This results in technical ports like energy flow from buses or power supply and mechanical screw connections being omitted.

4. **Derive Interfaces:** Finally, the information flow interfaces have to be abstracted to functional interfaces as types for the function ports. Whilst the previous steps could be automated, this step requires the requirements engineer's expertise. In the

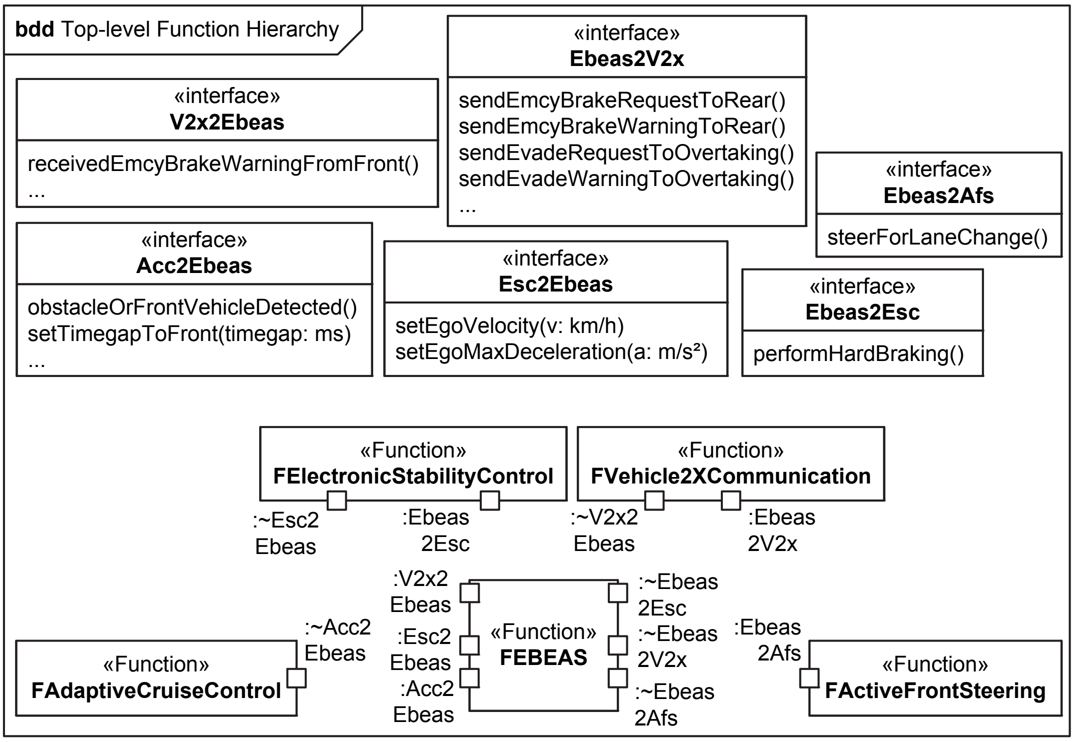


Figure 4.6: Top-level Function Hierarchy of the EBEAS

example's environment, the V2X messages have a parameter :Vehicle to specify what other vehicle around the ego vehicle is communicating. In the function hierarchy that parameter is omitted and the messages are instead appended with "front", "rear", and "overtaking" to describe what role the communicating vehicle has in the EBEAS scenario, as only messages from those vehicles are relevant for the EBEAS. Also, the information from the Adaptive Cruise Control that an obstacle or a vehicle was detected is merged to one (cf.obstacleOrFrontVehicleDetected() in Figure 4.6). Furthermore, continuously sent signals are abstracted to events: The function FEBEAS does not sent a trajectory but a simplified command steerForLaneChange() to the Active Front Steering.

Functional (safety) requirements for the top-level root function and each sub-function are afterwards specified in Step 2.2 (cf. Figure 4.4). The details, how these requirements are specified using MSDs are described in Sections 4.3.2 and 4.4. In following iterations of the ASIL tailoring process, sub-levels of the function hierarchy are derived by decomposing the complex functionality of the top-level root function (cf. Step 2.1.b) in Figure 4.3) and its requirements. This refinement is described in Section 4.3.3.

4.3.2 Structure of MSD Specifications for Functional Safety Requirements

Figure 4.7 shows the relation between the function hierarchy and its requirements specified as MSDs. The top of the figure shows an excerpt of the top-level function hierarchy of the EBEAS from Figure 4.6 as a SysML block definition diagram. It describes the functions and their required and provided information. This is specified via ports typed by interfaces that define information as UML operations. The function FEBEAS has a port p2 that is typed by the interface V2x2Ebeas. The interface defines the information receivedEmcyBrakeWarningFromFront.

The middle of Figure 4.7 shows a SysML collaboration required for MSD specifications (cf. Section 2.7). It is derived from the top-level function hierarchy and used to specify the allowed communication paths between the functions (represented as SysML parts). For instance, there is a connector between the port p1 of the function v2x and the port p2 of the function ebeas. It specifies that the information defined in the interface V2x2Ebeas may be sent by the function v2x to the function ebeas.

The bottom of Figure 4.7 shows an example MSD describing a requirement on the function FEBEAS. If the function ebeas is informed by the function v2x about a received emergency brake warning from the front vehicle and the last point to brake has not yet been passed, the function ebeas shall command the function esc to perform a hard braking maneuver. The cold condition in the MSD uses the boolean attribute passedLastPointToBrake of the function FEBEAS as defined in the block definition diagram.

A lifeline of an MSD represents a part from the collaboration that has a block from the function hierarchy as type (e.g., lifeline ebeas represents part ebeas of type FEBEAS). A message in an MSD (e.g., receivedEmcyBrakeWarningFromFront) obeys the signature of a UML operation from an interface (e.g., V2x2Ebeas) of the function hierarchy. Additionally, a message represents information flow from one function to another over a connector from the collaboration (e.g., from v2x to ebeas via p1 to p2).

4.3.3 Refining the Function Hierarchy and Safety Requirements

Requirements on the top-level function of a system under development are specified with MSDs as described in the previous section and as shown in Figure 4.7. To cope with the complexity of cyber-physical systems and their requirements, the top-level function is decomposed into smaller functions with decreased size. Accordingly, also the top-level requirements have to be broken down onto the smaller functions. To realize this, we decompose the function hierarchy using SysML block definition diagrams and internal block diagrams. The requirements specified as MSDs are decomposed following the structure defined by those diagrams based on work by Holtmann and Meyer [HM13]. This results in a function hierarchy with decomposed requirements specified as MSDs.

Figure 4.8 shows the structure and relation of the function hierarchy and its functional (safety) requirements in form of MSDs. The left column shows the function hierarchy with its functions and interfaces. The center column contains the function connections on the different hierarchy levels and the internal structure of each function. The right column depicts the functional (safety) requirements for the functions on the different hierarchy levels.

The top row of Figure 4.8 shows the top-level of the function hierarchy and its requirements. Hence, the top left contains (an excerpt of) the top-level function hierarchy that is derived from the environment model (cf. Section 4.3.1). The top center shows the collaboration required for the MSD specification (cf. Section 4.3.2). It specifies the connectors between the ports of the top-level function FEBEAS and its environment. The top right shows an example of a requirement on the top-level function FEBEAS. The lifelines

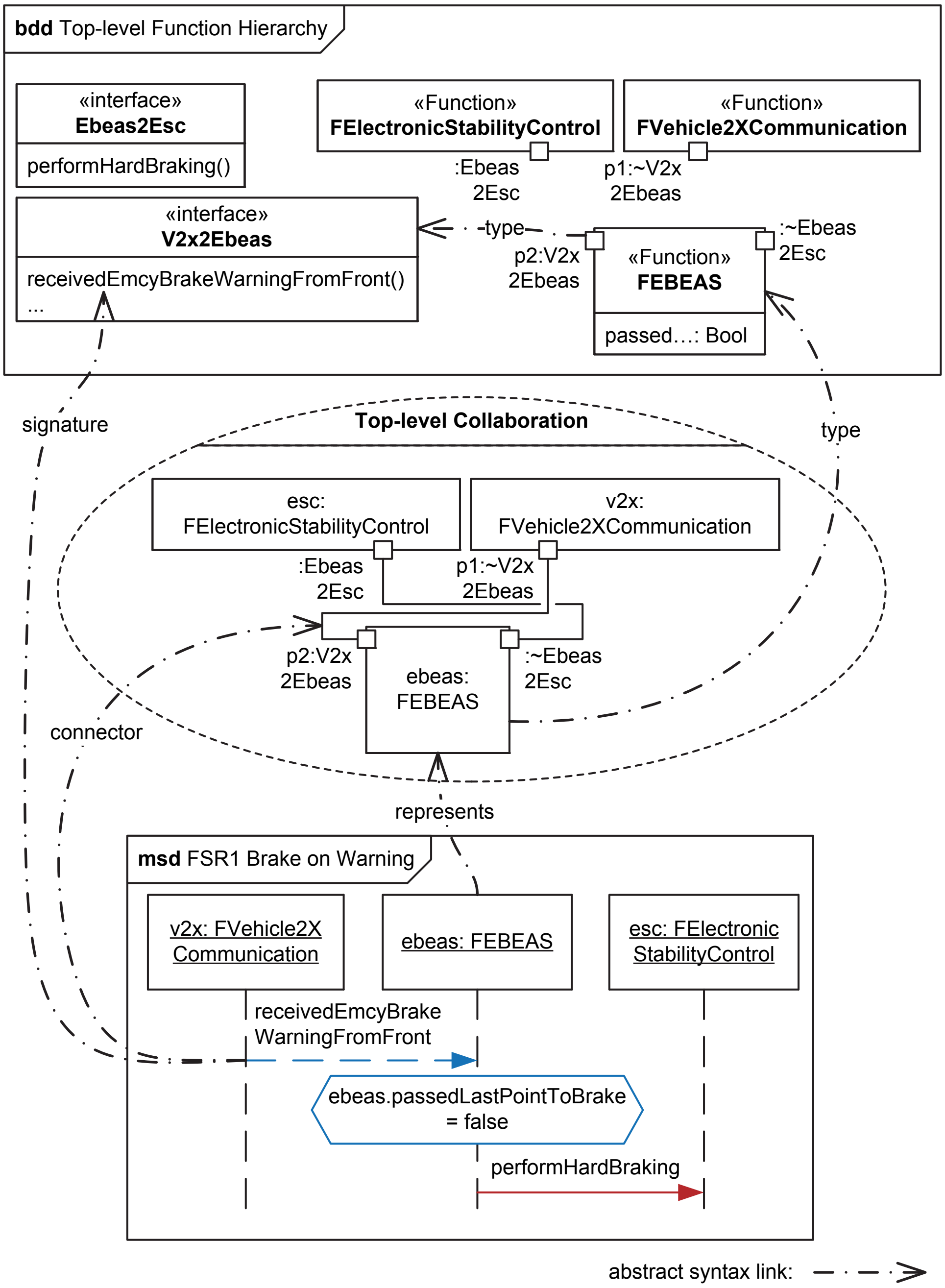
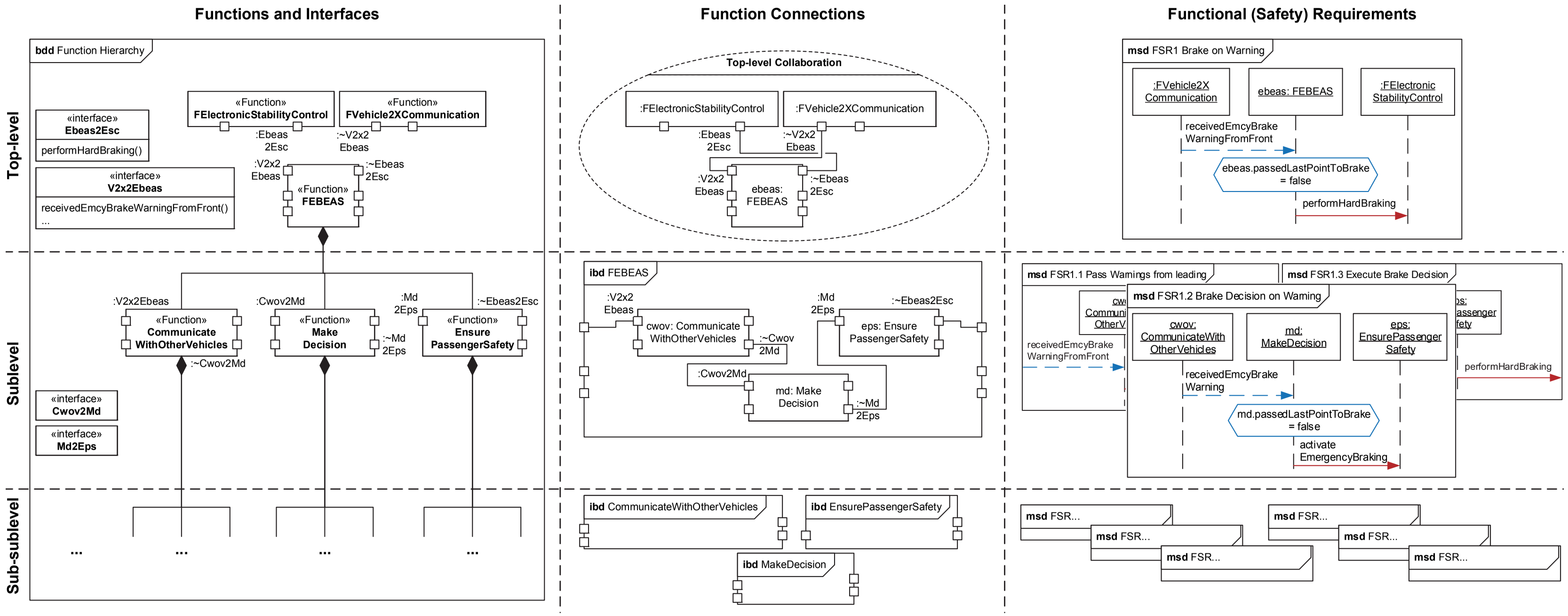


Figure 4.7: Relations between function hierarchy and MSD specification



Functions and Interfaces
Function Connections
Functional (Safety) Requirements
Top-level
Sublevel
Sub-sublevel
bdd Function Hierarchy
«interface» Ebeas2Esc
performHardBraking()
«interface» V2x2Ebeas
receivedEmcyBrakeWarningFromFront()
...
«Function» FElectronicStabilityControl
«Function» FVehicle2XCommunication
:Ebeas 2Esc
:~V2x2 Ebeas
:V2x2 Ebeas
:~Ebeas 2Esc
«Function» FEBEAS
:V2x2Ebeas
:Cwov2Md
:Md 2Eps
:~Ebeas2Esc
«Function» Communicate WithOtherVehicles
«Function» Make Decision
«Function» Ensure PassengerSafety
:~Md 2Eps
:~Cwov2Md
«interface» Cwov2Md
«interface» Md2Eps
...
Top-level Collaboration
:FElectronicStabilityControl
:FVehicle2XCommunication
ebeas: FEBEAS
ibd FEBEAS
cwov: Communicate WithOtherVehicles
:~Cwov 2Md
eps: Ensure PassengerSafety
:Cwov2Md
md: Make Decision
:~Md 2Eps
ibd CommunicateWithOtherVehicles
ibd EnsurePassengerSafety
ibd MakeDecision
msd FSR1 Brake on Warning
:FVehicle2X Communication
ebeas: FEBEAS
:FElectronic StabilityControl
receivedEmcyBrake WarningFromFront
ebeas.passedLastPointToBrake = false
performHardBraking
msd FSR1.1 Pass Warnings from leading
msd FSR1.3 Execute Brake Decision
msd FSR1.2 Brake Decision on Warning
cwov: CommunicateWith OtherVehicles
md: MakeDecision
eps: EnsurePassenger Safety
receivedEmcyBrake Warning
md.passedLastPointToBrake = false
activate EmergencyBraking
msd FSR...

Figure 4.8: Hierarchical structure of functions and requirements (based on [HM13])

represent the parts in the collaboration and the messages represent information flowing over connectors in the collaboration. For example, the message performHardBraking between the lifelines ebeas: FEBEAS and :FElectronicStabilityControl flows over the connector between the ports typed by the interface Ebeas2Esc.

The middle and bottom rows of Figure 4.8 depict sub-levels of the function hierarchy and requirements resulting from the refinement of the required functionality. The middle left shows three functions that decompose the top-level function FEBEAS. The ports of the top-level function are delegated down to the subfunctions (e.g., the port :V2x2Ebeas down to Communicate With Other Vehicles) and new ports and according interfaces are added to describe information flow between the subfunctions (e.g., the two ports typed by the interface Cwov2Md of the two functions Communicate With Other Vehicles and Make Decision). The internal block diagram in the center specifies the internal structure of the function FEBEAS (i.e., the connections between its subfunctions). The middle right shows the decomposition of the requirement on the top-level function into three requirements on the subfunctions. Each function and requirement of the first sublevel can be decomposed into further sub-levels in the same way as indicated by the bottom row of Figure 4.8.

This structure is built row by row and from left to right, during Steps 2.1 and 2.2 of the ASIL tailoring process (cf. Figure 4.3 and 4.4). These steps are based on the requirements engineering process described in [FH14]. The top-level function hierarchy in the top-left is developed in Step 2.1.a) as explained in Section 4.3.1. The collaboration in the top-center and the top-level functional (safety) requirements in the top-right are specified in Step 2.2.a) as described in Section 4.3.2. Afterward in Step 2.1.b), the top-level function hierarchy is refined into subfunctions and their connections as shown in the middle-left and center. After the structure of the function hierarchy has been refined, also the top-level functional (safety) requirements are refined in Step 2.2.b) as shown in the middle-right. Further sub-levels as depicted in the bottom row of Figure 4.8 are specified by iterating Steps 2.1.b) and 2.2.b).

4.4 Specifying Functional Safety Requirements with MSDs

As outlined in Section 4.3, we use MSDs to describe functional requirements and functional safety requirements. ISO 26262 *highly recommends* the use of semi-formal notations (e.g., UML) and semi-formal verification (e.g., "by executable models") as methods to specify and verify safety requirements of ASIL C and ASIL D systems [Int11g]. As MSDs refine semi-formal UML sequence diagrams to a formal notation, their concrete syntax is still close to that semi-formal notation. In addition, an MSD specification is an executable model that can be executed by the play-out algorithm (cf. Section 2.7) for semi-formal verification. The use of formal notations (e.g., Z [Int02]) and formal verification is *recommended* for ASILs B to D [Int11g]. MSDs have completely defined syntax and semantics, and thus, are considered a formal notation by ISO 26262 [Int11a]. In addition, MSDs and the language they are based on (Life Sequence Charts) can be used for formal verification [LLN$^+$09][LYZ$^+$11]. Also, Bitsch assembled a classification of typical safety requirements from industry [Bit00; Bit01]. He proposes to also use formal sequence diagram like notations for specifying safety requirements. Furthermore, MSDs have been successfully used to specify requirements in a case study in the automotive industry [GHM$^+$15]. For all these reasons, we use MSDs in this thesis as a notation for safety requirements.

In the following Section 4.4.1, we recapitulate the MSD semantics from the foundations (cf. Section 2.7) with focus on formulation of functional requirements. This is the formal basis for Section 4.4.2, where we specify MSD requirement patterns for well-known classes of chronological succession, real-time, and safety requirements.

4.4.1 MSD Semantics for Requirements

By following the model structure as described in Section 4.3, we use MSDs to describe functional requirements and functional safety requirements on the function hierarchy. On this level, requirements are formulated on an abstraction level that uses event-triggered, discrete information flow. Technical realization (e.g., in form of continuous signals) is explicitly kept undefined and simplified to events (e.g., rising or falling signal edges). In general, we use MSDs to describe if-then requirements. *If* some information flow sequence occurs or some condition holds, *then* some information flow shall (not) happen or some condition shall hold.

Figure 4.9 provides an overview of the semantics of the central MSD constructs *execution kind* and *temperature* (cf. Section 2.7) and how we interpret them for requirements specification. From the point of view of one MSD a monitored message can be observed during the execution of the MSD but its occurrence is not required. An executed message, on the contrary, is required to occur during the execution of an MSD. If it is not sent/received, a liveness violation occurs. In that case, we say the MSD is violated and, thus, the requirement is not fulfilled. A cold message may be sent/received after any preceding and before any subsequent messages of the same MSD, but its execution is not required to occur in this order (it is not strict). If any other message of the same diagram occurs when the cold message is expected, a cold violation occurs. In that case, we say the MSD is discarded (but the requirement is not violated). A hot message, on the contrary, has to strictly occur in the order as specified in the MSD. If any other message of the same diagram occurs when the hot message is expected, a hot violation occurs. In that case, we say the MSD is violated and, thus, the requirement is not fulfilled. In addition to messages, MSDs also can contain conditions. Cold violations of conditions lead to the MSD being discarded (but the requirement is not violated), and hot/liveness violations lead to the MSD being violated and the requirement being not fulfilled.

In Figure 4.9, the execution kind is annotated in the top over the two columns and the temperature on the left of the two rows. Each diagram is connected to a note describing its requirement (part) in natural language. The first message of an MSD is always monitored and cold. It is the trigger that starts the execution of the diagram. Accordingly, the first message a in all diagrams of Figure 4.9 is monitored and cold. The execution kind of conditions is fixed to their temperature. Hence, there is only one example MSD for a cold and one for a hot condition in the figure.

Monitored & Cold

The top left of Figure 4.9 shows two MSDs with monitored and cold semantics. The top MSD contains two monitored and cold messages a and b. As they are both monitored, there is no requirement for them to occur. In addition, they are both cold and, thus, not required to occur in the order specified. In conclusion, this diagram shows a possible execution of the two messages and describes no requirement at all. As sketched by the text in the note

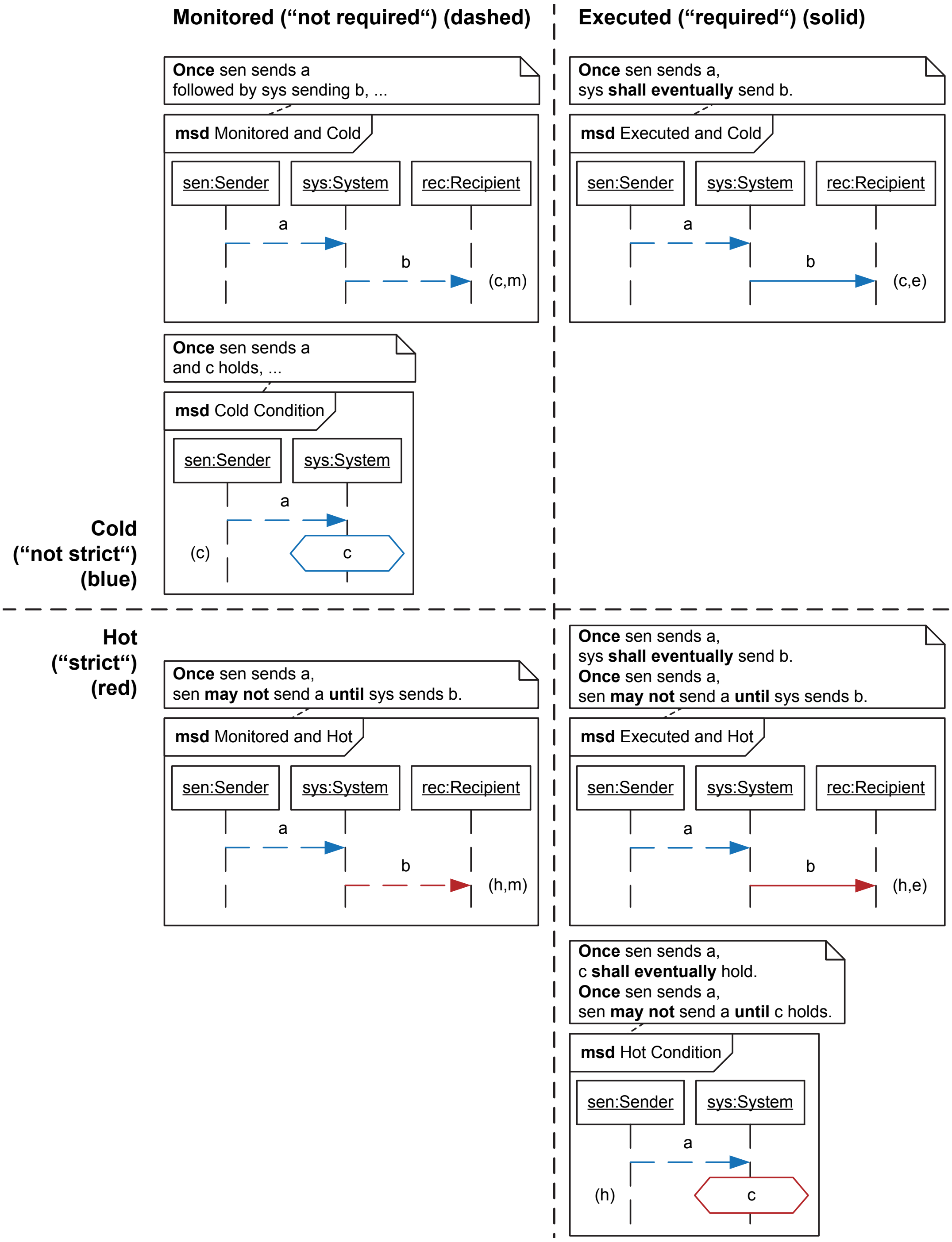


Figure 4.9: MSD semantics overview

annotated to the diagram, this diagram forms the if-condition of a requirement and could be extended by an executed or hot message to add the then-statement.

The bottom MSD in the top left of Figure 4.9 contains a cold condition c. Cold conditions are always executed immediately after their preceding message. Thus, in the diagram c holds together with a. However, as the condition is monitored it is not required to hold, and as it is cold it also is not required to hold together with (immediately after) a. Thus, this diagram describes no requirement, but only the if-condition of a requirement that could be extended. This fact is also described by the connected note in the same way as for the MSD above.

In conclusion, elements of an MSD that are monitored and cold can be used to describe the if-condition of a requirement but specify no requirement by themselves.

Executed & Cold

The top right of Figure 4.9 shows an MSD with executed and cold semantics. Specifically, the message b is executed and cold. This means, that if a occurs, b is required to occur as well. As there are no timing constructs in this MSD, an arbitrary long time may pass until b occurs but it has to occur eventually. Furthermore, arbitrary other messages may occur before b because it is cold (not strict). In fact, even the message a may occur again. In conclusion, an executed message always is part of the then-statement of a requirement as it is required to be sent/received.

Monitored & Hot

The bottom left of Figure 4.9 shows an MSD with monitored and hot semantics. Specifically, the message b is monitored and hot. This means, that if a occurs, b is required to follow before a may occur again. However, b is not required to occur because it is monitored. As there are no timing constructs in this MSD, an arbitrary long time may pass until b occurs but during that time a is not allowed to occur. Furthermore, any other message that is not part of this MSD may occur before b (but not a again). In conclusion, a hot message always is part of the then-statement of a requirement as it puts a restriction on the order of messages being sent/received.

Executed & Hot

The bottom right of Figure 4.9 shows two MSDs with executed and hot semantics. In the top MSD, the message b is executed and hot. Thus, it combines the requirements specified in the executed & cold MSD and the monitored & hot MSD. If a occurs, b is required to occur as well. As there are no timing constructs in this MSD, an arbitrary long time may pass until b occurs but it has to occur eventually. In addition, b is required to occur before a may occur again. Furthermore, any other message that is not part of this MSD may occur before b (but not a again).

The bottom MSD in the bottom right of Figure 4.9 contains a hot condition c. Contrary to cold conditions, hot conditions are not always evaluated immediately after their preceding message but they have to evaluate to true eventually (they have the execution kind executed). Thus, in the diagram, if a occurs, c is required to hold after a eventually. As there are no timing constructs in this MSD, an arbitrary long time may pass until c finally holds. In addition, because the condition is hot, c is required to hold before a may occur again. Furthermore, any other message that is not part of this MSD may occur before c holds (but not a again).

In conclusion, elements of an MSD that are executed and hot are always part of the then-statement of a requirement and combine the required and strict execution of a message or the fulfillment of a condition.

4.4.2 Functional (Safety) Requirement Classes

We use MSDs to specify functional safety requirements and general functional requirements. Thus, these types of requirements have to be expressible with the MSD language. Dwyer et al. assembled typical classes of general requirements on chronological succession of propositions (e.g., conditions that must hold) [DAC99]. Konrad and Cheng assembled classes of general real-time requirements [KC05]. Bitsch assembled classes of typical safety requirements [Bit00; Bit01]. All three sources based their requirement classes on experience from industry. Hence, we assume they cover the majority of (safety) requirements and our approach is applicable for typical industry projects if we can express their classes using MSDs. Consequently, we take these sources of requirement classes and show how they can be expressed with MSDs. Concepts presented in this section have been published in [FHK$^+$18].

All three sources provide examples for their requirement classes using Computation Tree Logic (CTL)[EC82] or Timed CTL [ACD93] for real-time, respectively. MSDs are based on Life Sequence Charts that can be translated to CTL [KHP$^+$05]. Timed MSDs and Life Sequence Charts can be translated to timed automata [BGH$^+$14; LLN$^+$09]. We translated the chronological succession and safety requirement classes to MSDs based on the CTL formulas provided by Dwyer et al.[1] and Bitsch [Bit00], respectively. The real-time classes were translated based on timed automata specified in [ZLG10] that were derived from the TCTL formulas of Konrad and Cheng. (T)CTL propositions typically argue over conditions that shall hold in certain states of a system. In this thesis we focus on requirements about the information exchange between system functions. This information exchange is specified using the scenario- and message-based MSDs. Each message exchange between two functions is an event. Thus, we adapt the requirement classes to focus on events instead of state-conditions where possible.

Table 4.1 shows the classification of general requirements based on the chronological succession classes by Dwyer et al. and real-time requirement classes by Konrad and Cheng. Each class is appended with an example requirement sketch. The variable u denotes a condition, k a numeric time value, and n an integer value. All other variables denote events. Each of the general requirement classes is subdivided into the five scopes shown in Figure 4.10:

- Globally: Always, an event is required to (not) occur or a condition is required to hold.
- Before r: Before an event r, an event is required to (not) occur or a condition is required to hold.
- After q: After an event q, an event is required to (not) occur or a condition to hold.
- Between q and r: Between two events q and r, an event is required to (not) occur or a condition is required to hold.
- After q until r: After an event q and until an event r, an event is required to (not) occur or a condition is required to hold.

[1] http://patterns.projects.cis.ksu.edu/documentation/patterns/ctl.shtml (last accessed 14.05.2016)

Table 4.1: General requirement classes (based on [DAC99; KC05])

General Requirement Pattern Class	Real-Time	Count
Occurrence		
Existence p shall eventually occur.	-	5
Absence p shall never occur.	-	5
Bounded Existence p shall occur at most n times.	-	5
Bounded Recurrence p shall occur at least every k time units.	x	5
Universality u shall always hold.	-	5
Minimum Duration Once u becomes satisfied, it shall hold for at least k time units.	x	5
Maximum Duration Once u becomes satisfied, it shall hold for less than k time units.	x	5
Order		
Response Once p occurs, s shall eventually occur.	-	5
Bounded Response Once p occurs, s shall occur after at most k time units.	x	5
Bounded Invariance Once p occurs, u shall hold for at least k time units.	x	5
Response Chain 1-n Once p occurs, s shall eventually occur and be succeeded by t. (n=2)	-	5
Response Chain n-1 Once s occurs and is succeeded by t, p shall eventually occur after t. (n=2)	-	5
Constrained Chain Once p occurs, s shall eventually occur and be succeeded by t, but z shall not occur between s and t.	-	5
Precedence Once p occurs, s shall previously have occurred.	-	5
Precedence Chain 1-n Once s occurs and is succeeded by t, p shall have occurred before s. (n=2)	-	5
Precedence Chain n-1 Once p occurs, t shall previously have occurred, preceded by s. (n=2)	-	5
	Total patterns:	80

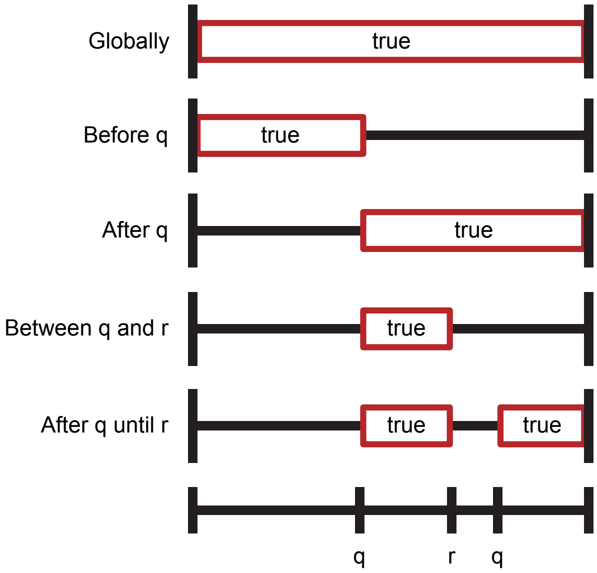


Figure 4.10: General requirement scopes from [DAC99]

Textually, the five scopes can be prepended to the example sentences for the requirement classes shown in Table 4.1. For the Existence class with “After q” scope this would be “After q, p shall eventually occur”. Figure 4.11 shows the MSD representation of this pattern and its application for the example requirement “After the function FEBEAS receives the information receivedEmcyBrakeWarningFromFront, it shall send the information performHardBraking to the function FElectronicStabilityControl”.

In the pattern MSD, the initial message q is monitored and cold. It describes the requirement’s if-condition – the “After q” scope. If q occurs, p has to occur. If q does not occur, no requirement on p is made. If q occurs, p shall eventually follow. This is specified by the executed and cold message p. It is not hot but cold, because q may occur arbitrarily often before p finally occurs (p does not have to follow strictly after every q). For the example requirement, this means that FEBEAS can be informed about the emergency braking of its front vehicle several times before it decides to perform a hard braking maneuver. To enforce a strict and timely response to the emergency brake warning, the real-time pattern Bounded Response with scope Globally can be used: “Globally, once p occurs, s shall occur after at most k time units”. In this pattern, “Globally” enforces that s has to occur strictly after every p. Figure 4.12 shows this pattern and the refined example requirement as MSDs.

In the pattern MSD, the initial message p is monitored and cold (just like q in Figure 4.11). It describes the requirement’s if-condition – the “Once p occurs”. If p occurs, s has to occur. If p does not occur, no requirement on s is made. p is followed by the message s that is executed and hot. If p occurs, s shall follow (it is executed just like p in Figure 4.11). s is hot because it shall occur strictly after p and before the following clock condition that states that not more than k time units passed since the initial p occurred. If the clock condition is reached after more than k time units passed, the MSD is violated and the requirement not fulfilled. The right MSD in Figure 4.12 shows the pattern application for the refined example requirement “Globally, once the function FEBEAS receives the information

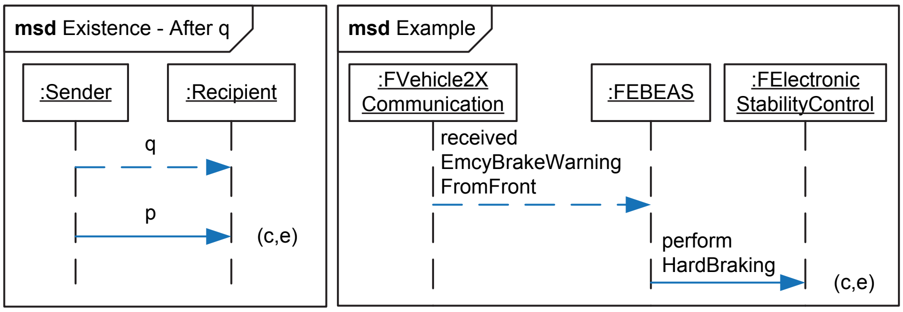

msd Existence - After q
:Sender
:Recipient
q
p
(c,e)
msd Example
:FVehicle2X Communication
:FEBEAS
:FElectronic StabilityControl
received EmcyBrakeWarning FromFront
perform HardBraking
(c,e)

Figure 4.11: General occurrence pattern “Existence (After q)” as MSD

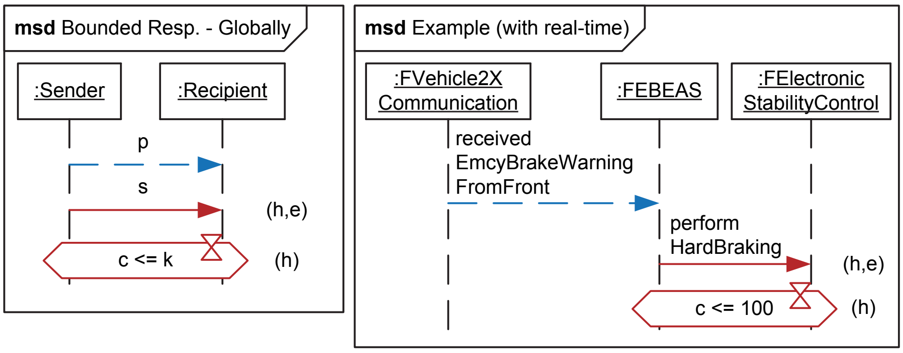

msd Bounded Resp. - Globally
:Sender
:Recipient
p
s
(h,e)
c <= k
(h)
msd Example (with real-time)
:FVehicle2X Communication
:FEBEAS
:FElectronic StabilityControl
received EmcyBrakeWarning FromFront
perform HardBraking
(h,e)
c <= 100
(h)

Figure 4.12: General real-time pattern “Bounded Response (Globally)” as MSD

receivedEmcyBrakeWarningFromFront, it shall send the information performHardBraking to the function FElectronicStabilityControl after at most 100 time units”.

Table 4.2 shows the classification of safety requirements by Bitsch. Each class is appended with an example requirement sketch. The variables a, b, c, and p denote events, the variable u a condition, and the variable T an integer time value. The classes Static and General Access Guarantee contain requirements describing global invariants for the system. The three chronological succession classes contain requirements on events that have to (not) occur or conditions that have to hold after/until a certain event. The class Explicit Time contains requirements stating that events have to (not) occur or conditions have to hold within or after a certain time frame. Bitsch divided this class into two subclasses for event-triggered and time-triggered formalisms. In this thesis, we use MSDs on the function hierarchy as an event-triggered formalism and, thus, only consider that class.

The chronological succession and real-time safety requirement classes from Table 4.2 are subdivided into four categories:

a) Necessary: In the given time frame an event is required to occur or a condition is required to hold.

b) Permitted: Only in the given time frame an event may occur or a condition may hold (but is not required).

c) Necessary & Permitted: In the given time frame an event is required to occur or a condition is required to hold and not allowed to occur/hold outside that time frame.

d) Conditional Guarantee: In the given time frame it must be possible for an event to occur or a condition to hold if necessary.

Table 4.2: Safety requirement classes (based on [Bit00; Bit01])

Safety Requirement Pattern Class	**Count**
Global	
Static Globally, u shall always hold.	2
General Access Guarantee Globally, p shall eventually occur (if requested).	2
Chronological Succession	
Beginning of Validity After a occurs, it is necessary/permitted/possible that ...	20
Duration of Validity Before b occurs, it is necessary/permitted that ...	4
Beginning & Duration of Validity After a until c, it is necessary/permitted that ...	14
Real-Time	
Explicit Time It is necessary/permitted/possible that after/within T time units ...	10
Total patterns:	52

The combination of the classes and categories forms the requirement scopes as depicted in Figures 4.13 and 4.14. The first figure shows the scopes for the three chronological succession classes and the latter for the real-time class.

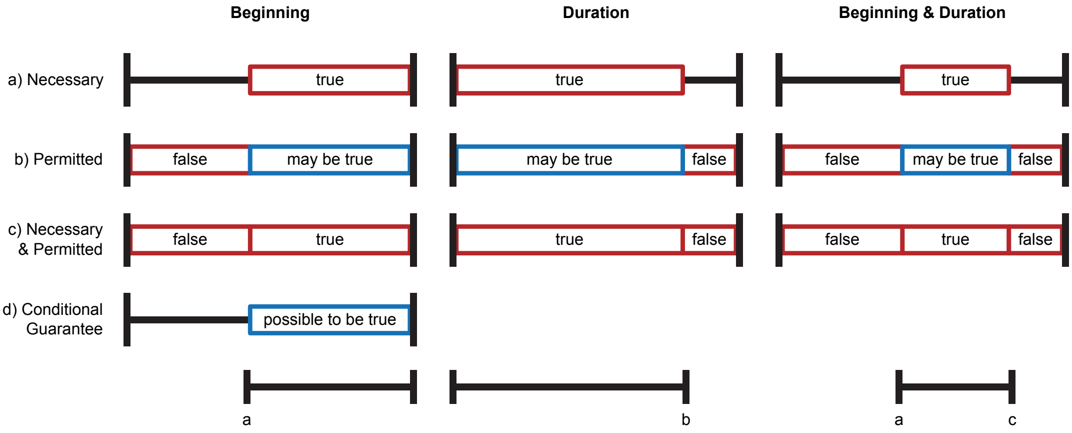


Figure 4.13: Chronological succession safety requirement scopes based on [Bit00; Bit01]

A complete textual example pattern from the safety requirement class Beginning with category "Necessary & Permitted" is "After a occurs, it is necessary that b eventually occurs and b is not permitted to occur before". Figure 4.15 shows the MSD representation of this pattern and its application for the example requirement "After the function FEBEAS sends the information sendEmcyBrakeWarningToRear, it is necessary that it sends the information performHardBraking and it may not send this information before sendEmcyBrakeWarningToRear". This phrasing is difficult to understand and violates the

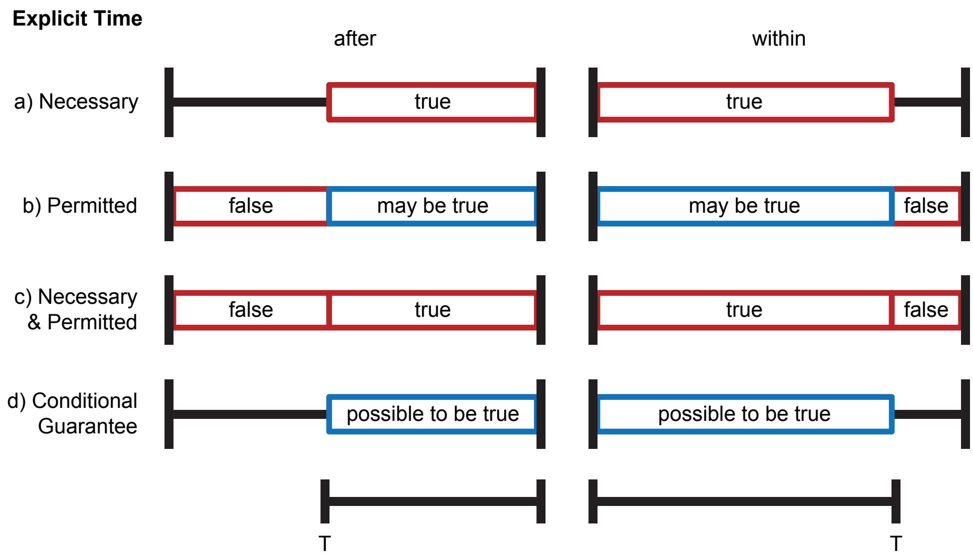


Figure 4.14: Real-time safety requirement scopes based on [Bit00; Bit01]

requirement quality characteristic that each requirement shall be singular and not use any conjunctions like “and” [Int11i]. Thus, we split the pattern into two MSDs shown in the top of Figure 4.15.

The first MSD specifies the permission part of the pattern: “Before a occurs, b is not permitted to occur”. The MSD specifies the event trace that is not allowed to occur and “the way out”. The initial message start represents the system start (e.g., the vehicle’s ignition). After system start, b is not allowed to occur as long as no a occurred. This is specified by a monitored and cold message b, followed by a hot condition false, and a monitored and cold message a. Because all messages are monitored and cold, none of them is required to occur and there is no requirement on their order. However, if b occurs after start the MSD is violated and the permission requirement is not fulfilled because the expression of the hot condition always evaluates to false. The only exception is, if a occurs after start when b is expected. In that case, a cold violation occurs and the MSD is discarded. Nevertheless, the permission requirement is still fulfilled because no b occurred before a, and the MSD is no longer required, as b is now permitted to occur. The second MSD specifies the necessity part of the pattern: “After a occurs, it is necessary that b eventually occurs”. b is specified as executed and cold because it is required to occur after a but a may occur several times before b finally occurs.

The two MSDs in the bottom of Figure 4.15 show the application of the pattern to the example requirement. Once an obstacle or a front vehicle is detected (represented by the message obstacleOrFrontVehicleDetected) the functionality of the EBEAS is started. Then, the EBEAS may not perform a hard braking maneuver until it warned the rear vehicle (represented by the message sendEmcyBrakeWarningToRear to the function FVehicle2XCommunication). Once it warned the rear vehicle, it shall eventually perform the hard braking.

By comparing the safety requirement pattern shown in Figure 4.15 with the MSDs of the general requirement classes, we realized that the safety pattern is a combination of the

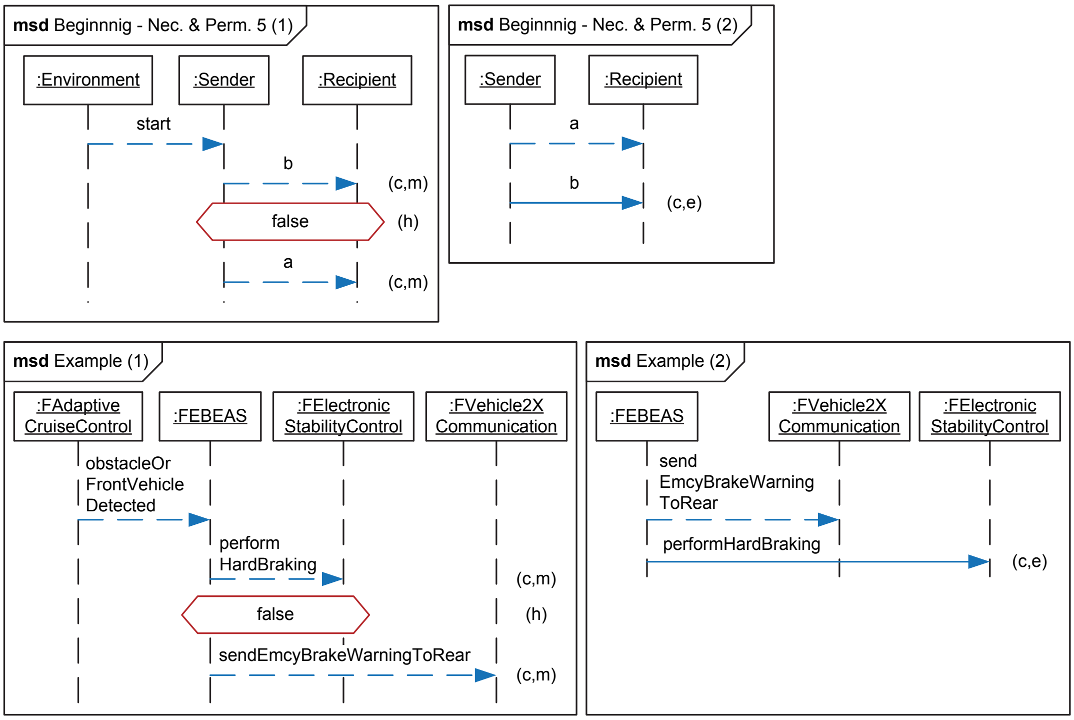

msd Beginnnig - Nec. & Perm. 5 (1)
:Environment
:Sender
:Recipient
start
b
(c,m)
false
(h)
a
(c,m)
msd Beginnnig - Nec. & Perm. 5 (2)
:Sender
:Recipient
a
b
(c,e)
msd Example (1)
:FAdaptive CruiseControl
:FEBEAS
:FElectronic StabilityControl
:FVehicle2X Communication
obstacleOr FrontVehicle Detected
perform HardBraking
(c,m)
false
(h)
sendEmcyBrakeWarningToRear
(c,m)
msd Example (2)
:FEBEAS
:FVehicle2X Communication
:FElectronic StabilityControl
send EmcyBrakeWarning ToRear
performHardBraking
(c,e)

Figure 4.15: Safety pattern “Beginning - Necessary & Permitted 5” as MSDs

two patterns Absence (After q until r) and Existence (After q). This mapping is depicted in Figure 4.16.

We found that all global and chronological succession safety requirement classes and a subset of the explicit time class can be expressed by (a combination of) the general requirement classes of Dwyer et al. and Konrad and Cheng. Table 4.3 shows the complete mapping between the requirement classes. For example, the requirement patterns of the classes Static and General Access Guarantee can be specified by a combination of patterns of the classes Universality, Existence, and Response with the scope Globally.

Unfortunately, most of the requirement patterns from the explicit time class could not be mapped to patterns of Dwyer et al. or Konrad and Cheng. Figure 4.17 shows the safety pattern “Explicit Time - Permitted 2” as an example of that. In textual form it is phrased as “Only after T time units it is permitted that a occurs”. Figure 4.17 shows the MSD representation of this pattern and its application for the example requirement “Only 200 time units after the function FEBEAS sent the information sendEvadeWarningToOvertaking, it is permitted that it sends the information steerForLaneChange”.

In the pattern MSD, the initial message startTimer is monitored and cold. It specifies the event after which the time has to be measured. The message is followed by a hot clock condition c > T. It describes the requirement’s if-condition – only if T time units passed. If startTimer occurs and T time units pass, a is allowed to occur. a is a monitored and cold message after the clock condition. It is not required to occur but if it occurs, it may not occur between startTimer and the clock condition evaluating to true. This order is encoded by the clock condition being hot and, thus, being required to occur strictly before a.

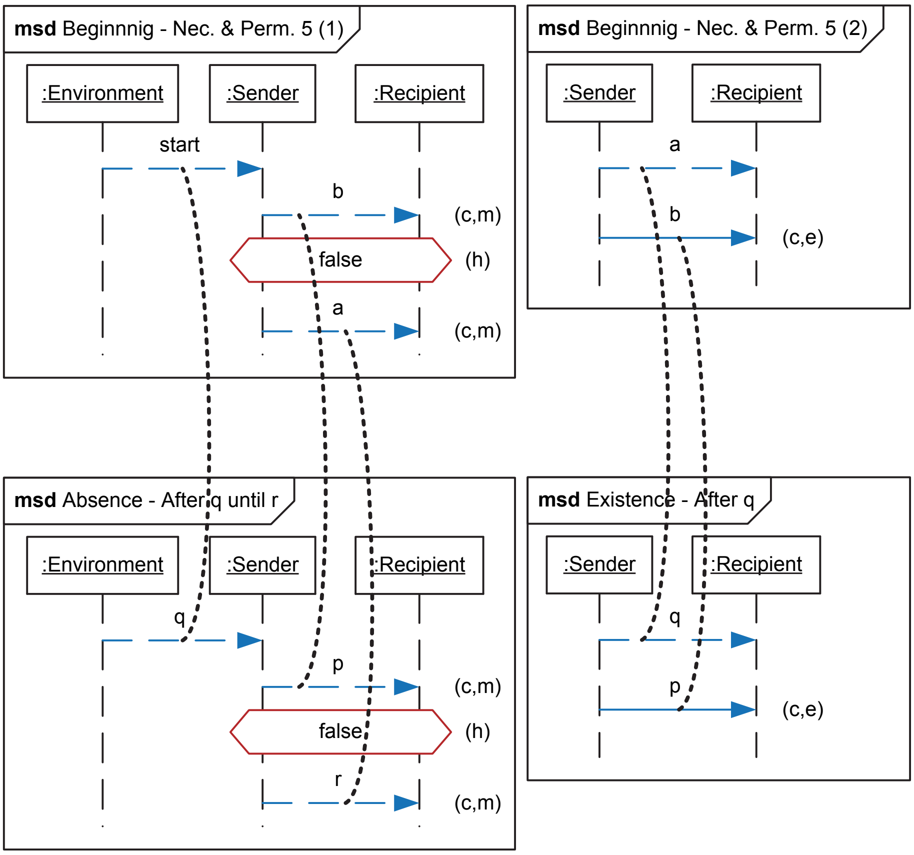


Figure 4.16: Safety pattern "Beginning - Necessary & Permitted 5" mapped to Absence (After q until r) and Existence (After q)

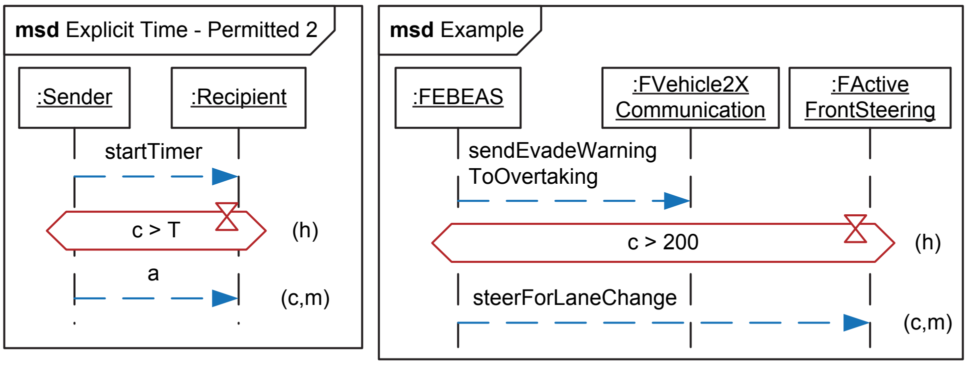


Figure 4.17: Safety pattern "Explicit Time - Permitted 2" ("Time-constrained Absence 2") as MSD

Table 4.3: Relation of requirement classes

Safety	General
Static, General Access Guarantee	Universality, Existence, Response (Globally)
Beginning	
a) Necessary	Universality, Existence (After q)
b) Permitted	Absence (After q, After q until r)
c) Necessary & Permitted	Absence (After q until r), Universality, Existence (After q)
d) Conditional Guarantee	Response, Absence (After q)
Duration	
a) Necessary	Universality (Before r)
b) Permitted	Universality (After q)
c) Necessary & Permitted	Universality (Before r, After q)
Beginning & Duration	
a) Necessary	Universality, Existence, Absence (After q until r)
b) Permitted	Absence (After q, After q until r), Universality (After q)
c) Necessary & Permitted	Universality, Existence, Absence (After q until r)
Explicit Time	
a) Necessary	Bounded Response (Globally), -
b) Permitted	-
c) Necessary & Permitted	-
d) Conditional Guarantee	Bounded Response (Globally), -

The right MSD in Figure 4.17 shows the application of the pattern to the example requirement. Once the EBEAS sends a warning that it will evade (represented by the message sendEvadeWarningToOvertaking), 200 time units have to pass before the EBEAS may actually start to evade (represented by the message steerForLaneChange).

Independently from Bitsch, Autili et al. identified two real-time pattern classes that are missing in the catalog of Konrad and Cheng [AGL+15]. They call those classes "Time-constrained Existence" and "Time-constrained Absence". We identified these as the same as Bitsch's "Explicit Time - Necessary" and "Explicit Time - Permitted", respectively. We find the names of Autili et al. more expressive and use those in the remainder of this thesis. The class "Explicit Time - Necessary & Permitted" is a combination of the two that we thus call "Time-constrained Absence & Existence". The patterns of the class "Explicit Time - Conditional Guarantee" can be mapped to patterns of the class "Explicit Time - Necessary" (cf. [FHK+17] for details) and, thus, is not used in the following.

Table 4.4 shows the final set of requirement classes merged from Dwyer et al., Konrad and Cheng, and Bitsch. The safety classes are mapped to the other classes where possible and otherwise listed using the names proposed by Autili et al.. For each class it is annotated from what source it originates, whether it is a real-time class, whether it is usable to specify safety requirements (i.e. originating or mapped from Bitsch), and the number of contained patterns. The complete catalog of requirement patterns of the different classes, including the detailed mapping of the safety requirement classes, can be found in [FHK$^+$17]. The catalog consists of 86 distinct requirement patterns with MSD examples (55 non-real-time and 31 real-time patterns).

Table 4.4: Merged requirement pattern classes

Pattern Class		Source	Real-Time	Safety	Count
Occurrence					
1.	Existence	[DAC99]	-	x	5
2.	Bounded Existence	[DAC99]	-	-	5
3.	Time-constrained Existence	[Bit00]	x	x	2
4.	Bounded Recurrence	[KC05]	x	-	5
5.	Universality	[DAC99]	-	x	5
6.	Minimum Duration	[KC05]	x	-	5
7.	Maximum Duration	[KC05]	x	-	5
8.	Absence	[DAC99]	-	x	5
9.	Time-constrained Absence	[Bit00]	x	x	2
10.	Time-constrained Absence & Existence	[Bit00]	x	x	2
Order					
11.	Response	[DAC99]	-	x	5
12.	Bounded Response	[KC05]	x	x	5
13.	Bounded Invariance	[KC05]	x	-	5
14.	Response Chain 1-n	[DAC99]	-	-	5
15.	Response Chain n-1	[DAC99]	-	-	5
16.	Constrained Chain	[DAC99]	-	-	5
17.	Precedence	[DAC99]	-	-	5
18.	Precedence Chain 1-n	[DAC99]	-	-	5
19.	Precedence Chain n-1	[DAC99]	-	-	5
				Total patterns:	86

4.5 Integrating MBRE and NLRE for Safety Requirements Engineering

Model-based requirements engineering (MBRE) as used in the previous sections has many advantages over natural language requirements engineering (NLRE). For instance, their semi-formal or formal semantics allow model-based requirements to be automatically processed for verification and validation. On the contrary, informal natural language often introduces ambiguity. Nevertheless, natural language can be used for any type of requirement whilst model-based languages are tailored to specific types of requirements and need to be learned before they can be used correctly. Thus, ISO 26262 proclaims that safety requirements shall be specified "by an appropriate combination of" natural language and "informal/semi-formal/formal notations for requirements specification" [Int11g]. Following this requirement, in this thesis, we use the formal, model-based language Modal Sequence

Diagrams, and show in this section, how it can be integrated with semi-formal and informal natural language.

In previous work, we presented a combined MBRE-NLRE approach [FH14; FH15] [FHM14; DFH$^+$13]. It contains a so-called *analysis model* specified in a SysML block definition diagram that similarly to the CONSENS function hierarchy decomposes the required functionality of a system. In addition, textual requirement patterns in form of a semi-formal controlled natural language (CNL) are used to specify the analysis model and its requirements in natural language. A bidirectional transformation between the analysis model and the natural language requirements keeps both representations in sync.

In this thesis, we adapt that approach to the used CONSENS function hierarchy extended by ports and interfaces (cf. Section 4.3). Figure 4.18 sketches the resulting work products and their relations. The top left represents the model-based function hierarchy as explained in Section 4.3. The bottom left shows the model-based (safety) requirements specified as MSDs whose lifelines represent functions of the function hierarchy (cf. Section 4.4). The top right is a textual representation of the function hierarchy specified in a CNL. This representation can automatically be derived from its model-based equivalent using a bidirectional transformation [FH14]. This transformation allows to synchronize changes in either representation with the other. The bottom right depicts textual requirements that have no model-based representation but still address functions of the function hierarchy. This requirements representation is especially intended for types of requirements that are not expressible with MSDs.

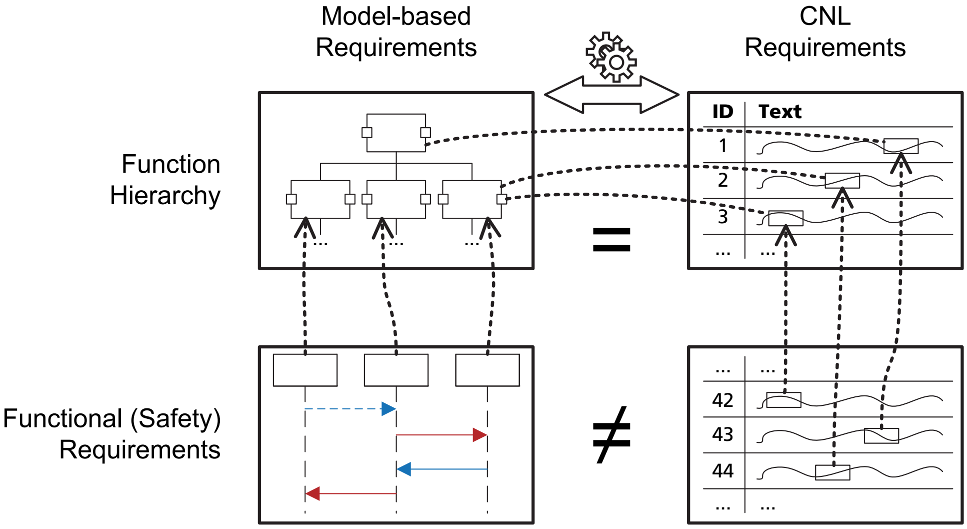


Figure 4.18: Overview of MBRE-NLRE integration

All information contained in a model-based function hierarchy (as explained in Section 4.3) can be expressed in natural language using the eight requirement patterns RP 1 to RP 8 listed below. They are adapted from requirement patterns published in [FHM14] to match the terms used in CONSENS.

Requirement Patterns for Function Hierarchies

RP 1: The top-level function is called *FUNCTION*.

RP 2: The function *FUNCTION* has the following purpose: *PURPOSE*.

RP 3: The functional device *FUNCTIONALDEVICE* has the following purpose: *PURPOSE*.

RP 4: The following information is received by the function *FUNCTION* [from the functional device *FUNCTIONALDEVICE*]: *INFORMATIONLIST*.

RP 5: The following information is sent from the function *FUNCTION* [to the functional device *FUNCTIONALDEVICE*]: *INFORMATIONLIST*.

RP 6: The function *FUNCTION* is a subfunction of the function *PARENTFUNCTION*.

RP 7: The following information is used by the function *FUNCTION*: *INFORMATIONLIST*.

RP 8: The following information is created by the function *FUNCTION*: *INFORMATIONLIST*.

Requirement pattern RP 1 is used to specify the top-level root function of a function hierarchy. RP 2 can be applied to summarize a function's purpose. RP 3 does the same for functions representing environment elements. RP 4 allows to specify information that the top-level function receives from environment functions. RP 5 similarly specifies information the top-level function sends to the environment. RP 6 is used to specify a subfunction of a (top-level) function. RP 7 is used to specify input information that a subfunction uses. RP 8 similarly is used to specify output information that a subfunction creates.

Figure 4.19 shows an example excerpt from the model-based function hierarchy of the EBEAS. The following requirements Req-1 to Req-18 form the CNL representation of that diagram. The textual chapter structure of the CNL requirements follows the model structure of the function hierarchy. This adheres to the organizational approach "Functional Hierarchy" suggested by ISO 29148 for high-quality requirements specifications [Int11i].

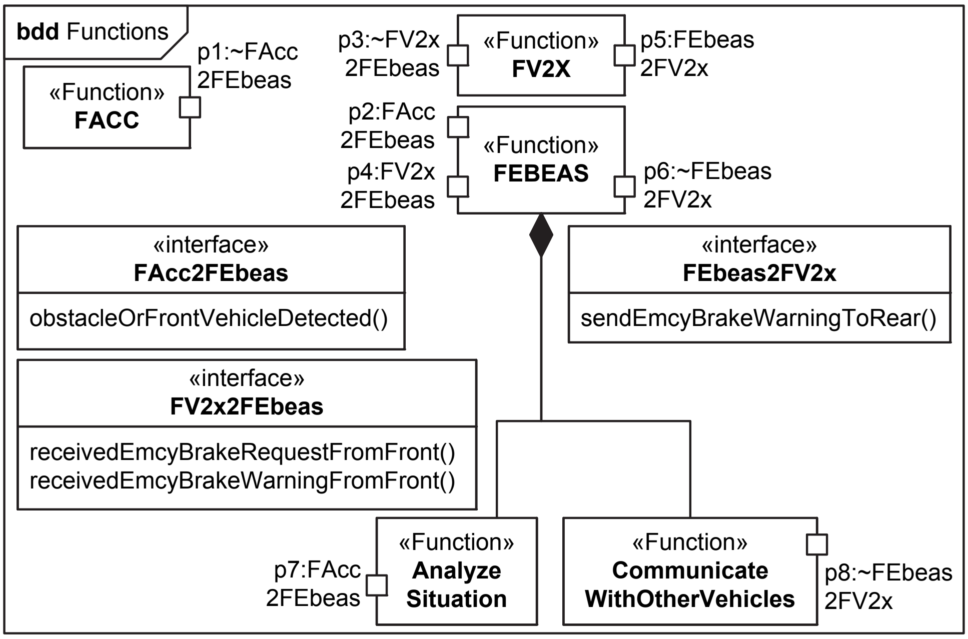


Figure 4.19: Example excerpt from model-based function hierarchy for EBEAS

CNL Requirements for Function Hierarchy in Figure 4.19

Req-1: **1 FEBEAS**
Req-2: The top-level function is called FEBEAS.
Req-3: **1.1 External Elements**
Req-4: The functional device FACC has the following purpose: “It provides obstacle and front vehicle information via sensors”.
Req-5: The functional device FV2X has the following purpose: “It sends and receives V2V messages”.
Req-6: **1.2 Inputs & Outputs**
Req-7: The following information is received by the function FEBEAS from the functional device FACC: obstacleOrFrontVehicleDetected.
Req-8: The following information is received by the function FEBEAS from the functional device FV2X: receivedEmcyBrakeRequestFromFront and receivedEmcyBrakeWarningFromFront.
Req-9: The following information is sent from the function FEBEAS to the functional device FV2X: sendEmcyBrakeWarningToRear.
Req-10: **1.3 Subfunctions**
Req-11: **1.3.1 Analyze Situation**
Req-12: The function AnalyzeSituation is a subfunction of the function FEBEAS.
Req-13: **1.3.1.1 Inputs & Outputs**
Req-14: The following information is used by the function AnalyzeSituation: obstacleOrFrontVehicleDetected.
Req-15: **1.3.2 Communicate With Other Vehicles**
Req-16: The function CommunicateWithOtherVehicles is a subfunction of the function FEBEAS.
Req-17: **1.3.2.1 Inputs & Outputs**
Req-18: The following information is created by the function CommunicateWithOtherVehicles: sendEmcyBrakeWarningToRear.

Model-based (safety) requirements on the functions are specified using MSDs as described in Section 4.4. In future work, textual patterns could be defined for the MSD patterns defined in Section 4.4.2. This could be based on existing textual patterns [KC05; AGL$^+$15] and requirements languages [GGK$^+$16]. In addition to the requirement patterns described in Section 4.4.2, there are requirements that cannot be specified using MSDs. Examples are requirements on the independence of functions (i.e., no cascading or common cause failures as specified in Figure 3.15 on page 54). However, these can be specified using natural language. Examples of according textual patterns and their application are shown below (RP 9 to RP 11 and Req-19 to Req-22).

Additional Requirement Patterns for Functions

RP 9: There shall not be any cascading failures from the function *FUNCTION1* to *FUNCTION2*.
RP 10: There shall not be any common cause failures of the functions *FUNCTIONLIST*.
RP 11: The function *FUNCTION* shall *FREETEXT*.

Additional CNL Requirements for Function Hierarchy in Figure 4.19

Req-19: There shall not be any cascading failures from the function AnalyzeSituation to CommunicateWithOtherVehicles.

Req-20: There shall not be any cascading failures from the function CommunicateWithOtherVehicles to AnalyzeSituation.

Req-21: There shall not be any common cause failures of the functions AnalyzeSituation and CommunicateWithOtherVehicles.

Req-22: The function AnalyzeSituation shall . . .

4.6 Assumptions & Limitations

Assumptions

For the derivation of the top-level function hierarchy from the environment (cf. Section 4.3.1), we assume that the relevant input and output information is explicitly specified as CONSENS information flow (e.g., not implicitly hidden as energy flow).

Some of the requirement patterns of the MSD pattern catalog (cf. Section 4.4.2) are subject to assumptions as documented in the corresponding technical report [FHK+17].

Limitations

ISO 26262 defines the concept of *Safety Element out of Context* (SEooC) (cf. Section 3.7). If a system like the EBEAS is not developed for a specific vehicle (i.e., context or environment), assumptions about its environment have to be made. Once the system is integrated into a vehicle, these assumptions have to be checked for validity. An MSD specification can contain so-called *environment assumption* MSDs [BGP13]. These are used to specify assumptions about the behavior of the environment. Thus, SEooC might also be tailored using the ASIL tailoring process of this thesis, but they were not specifically considered.

If the MSD specification is hierarchically built up for the function hierarchy as described in Section 4.3.3, a parallel simulation (play-out) of multiple function hierarchy levels is currently not possible. The followed approach of Holtmann and Meyer requires the use of UML interaction references to refine an MSD over different hierarchy levels [HM13]. This constrains the refinement such that one MSD cannot be refined into a combination of multiple MSDs but has to be refined into a single MSD. Hence, we assume that each function hierarchy level is simulated separately. In the end, the lowest level of functions specifies the full functionality and only these functions are actually allocated to the system architecture. Thus, we argue that this constraint is acceptable. However, in future work, the play-out maybe can be adapted to remove this constraint, e.g., by adopting the work of Atir et al. [AHK+08].

The integration of MBRE and NLRE described in Section 4.5 supports the bidirectional synchronization of the function hierarchy (model and text). The requirements specified with MSDs are not synchronized with a textual representation. However, in future work, textual patterns could be defined for the MSD patterns defined in Section 4.4.2. This could be based on existing textual patterns [KC05; AGL+15] and requirements languages [GGK+16].

4.7 Related Work

We divide the work related to our method for specifying functional (safety) requirements into three categories: work related to the use of a function hierarchy (Section 4.7.1), to the use of MSDs (Section 4.7.2), and to the integration of MSDs and natural language (Section 4.7.3).

4.7.1 Function Hierarchies

We use the so-called function hierarchy that is part of CONSENS [GRS14, Section 4.1] to decompose the complex functionality of cyber-physical systems and to structure the functional (safety) requirements. ISO 29148 suggests to use a functional structure to decompose requirements [Int11i]. Other model-based systems engineering approaches also use a functional abstraction of the system under development.

SYSMOD/FAS

SYSMOD [Wei14] and FAS [LW10] combined form a model-based systems engineering method based on SysML. SYSMOD provides means to specify the system context, requirements, use cases and a physical architecture. FAS is embedded, and derives a functional architecture from the SYSMOD use cases as basis for the SYSMOD physical architecture [LW14]. The functions are derived by grouping use cases with high cohesion. The aim of the functional architecture is a solid model that abstracts from variations in the physical architecture.

The SYSMOD system context is similar to the CONSENS environment and the physical architecture to the CONSENS active structure. In contrast to how the CONSENS function hierarchy is used in this thesis, SYSMOD/FAS links functions to use cases and not to functional requirements directly.

SPES

The SPES Modeling Framework [BDH$^+$12] is a development methodology for software-intensive embedded systems. It is not based on a specific language like SysML. It comprises four viewpoints: the requirements, functional, logical, and technical viewpoint. The requirements viewpoint is used to specify model-based requirements. The functional viewpoint starts with a functional black-box model consisting of user functions derived from the requirements. It is followed by a functional white-box model to decompose user functions and the according system behavior [VEF$^+$12]. The logical viewpoint specifies the logical components of the system that are realized by software or hardware in the technical viewpoint.

In previous work, we tailored the SPES modeling framework to the automotive domain and applied and evaluated it [FHH$^+$12].

In contrast to CONSENS, SPES does not contain a specific environment or system context viewpoint. This information is spread across the four existing viewpoints. The functional viewpoint is similar to the CONSENS function hierarchy, but consists of two sub-models (black-box and white-box). The logical viewpoint is related to the CONSENS active structure. The technical viewpoint describes discipline-specific details like software scheduling and deployment. Hence, there is no equivalent in CONSENS.

EAST-ADL

EAST-ADL is an architecture description language for automotive electronic systems [EAS13; BLH$^+$13]. It is comprised of different modeling layers starting from vehicle features, refined into a Functional Analysis Architecture, refined into a Functional Design Architecture, and ending with the software and hardware specification for the automotive specific AUTOSAR framework (Automotive Open System Architecture)[2]. Orthogonal to these layers EAST-ADL also contains environment, requirements, and dependability aspects. EAST-ADL provides a custom metamodel that is also available as UML profile [EAS10].

In contrast to CONSENS, EAST-ADL's environment only contains elements in the system's context and not the system itself. The elements of the environment are connected to elements in the functional analysis and the functional design architectures. In EAST-ADL, requirements are specified as nodes with text like in SysML. They can be hierarchically structured but that is not required.

The functional analysis architecture is related to the CONSENS function hierarchy. It contains functions that can be hierarchically decomposed and that interact via connected ports. These ports can specify flows or client-server interactions. In this thesis, we extend the CONSENS function hierarchy by ports that specify abstract information flow, and do not consider whether it is realized as energy flow or operations called by software. Furthermore, the functional analysis architecture also contains special types of functions to model sensors and actuators. In contrast to EAST-ADL, we do not specify these details in the function hierarchy but in the following CONSENS active structure. The latter is similar to EAST-ADL's functional design architecture where decisions what is to be realized in software or hardware are documented.

As EAST-ADL is specifically developed for the automotive domain (especially AUTOSAR) its dependability part provides means to annotate ASILs to elements and to model failure propagation that can be analyzed by an external safety analysis tool [CJL$^+$08].

Summary

The three approaches SYSMOD/FAS, SPES, and EAST-ADL all have commonalities and slight differences to CONSENS and the way its function hierarchy is used in this thesis. All three provide a functional view of the system. In contrast to this thesis, SYSMOD/FAS does not link the functions to requirements. The SPES functional viewpoint consists of two distinct sub-models for a black-box and white-box view and is not based on a standard modeling language like SysML. EAST-ADL is specifically designed for the automotive domain, and integrates technical interface information into its functional analysis architecture that we explicitly abstract from.

SPES and EAST-ADL provide safety modeling and analysis means. However, only EAST-ADL integrates aspects for ASIL allocation. Although we use the automotive domain as running example in this thesis, our approach is designed to work for other safety-relevant domains as well.

4.7.2 Formal Functional Safety Requirements

ISO 26262 highly recommends the use of semi-formal notations (e.g., UML) and semi-formal verification (e.g., "by executable models") [Int11g]. The use of formal notations

[2] www.autosar.org

and formal verification is recommended by ISO 26262 [Int11g]. We use MSDs to specify graphical, model- and scenario-based, formal functional (safety) requirements. MSDs integrate with the semi-formal UML by a profile, and can be executed for semi-formal verification via play-out.

The Scenario Modeling Language (SML) is a textual, scenario-based formal notation [GGK$^+$16]. It is based on MSDs, and supports semi-formal verification via play-out. In contrast to MSDs, it is not a graphical language, and it has not been shown that it can express the requirement patterns from Section 4.4.2.

Computation Tree Logic (CTL) is a textual formal notation [EC82] that is widely used for formal verification. It can express all the non-real-time requirement patterns (cf. Table 4.4). Timed CTL is a formal notation that extends CTL with real-time capabilities [ACD93]. It can express all the real-time patterns. In contrast to MSDs, CTL and TCTL are neither graphical nor scenario-based languages. Accordingly, they are not integrated with semi-formal notations like UML and do not support semi-formal verification.

Property Sequence Charts (PSC) are a graphical, scenario-based, formal notation [AIP07]. They can be translated to CTL and express the general non-real-time patterns (cf. Table 4.1). Timed PSCs are a real-time extension of PSCs that can express the general real-time patterns [ZLG10]. In contrast to MSDs, it has not been shown that (T)PSCs can also express the safety patterns (cf. Table 4.2). In addition, PSCs and TPSCs are not integrated with a semi-formal notation like UML and do not support semi-formal verification.

4.7.3 MBRE-NLRE Integration for Safety

We integrate model-based safety requirements engineering with natural language safety requirements engineering by synchronizing the function hierarchy in its model-based representation with a textual representation in controlled natural language (CNL). The requirement patterns introduced in Section 4.4.2 are expressed using MSDs, and requirements that cannot be expressed using MSDs are specified in CNL.

Gordon and Harel generate Life Sequence Charts (basis of MSDs) from a CNL but provide no transformation in the other direction [GH09]. A general survey on approaches that generate models from natural language requirements has been conducted by Yue et al. [YBL11]. A survey for the reverse direction has been presented by Nicolás et al. [NT09]. Both surveys did not find true synchronization approaches that transform in both directions. Furthermore, no approach specifically focuses safety requirements engineering.

4.8 Conclusion

This chapter describes the details how functional (safety) requirements are specified and refined in the ASIL tailoring process introduced in Chapter 3. Section 4.2 specifies the contents of a requirements specification. The functional (safety) requirements are derived from the system's environment and systematically structured and refined as described in Section 4.3. What types of requirements are relevant and how these can be expressed using Modal Sequence Diagrams is explained in Section 4.4. For requirements that cannot be expressed with MSDs, Section 4.5 specifies an integration of model-based and natural-language-based safety requirements engineering.

The model- and scenario-based, hierarchically decomposed specification of functional safety requirements provides a means to cope with the complexity of CPS functionality and requirements (cf. Challenge C1 in Section 1.2). The formal specification of functional safety requirements with MSDs enables the automated safety analysis and ASIL allocation on requirements level (cf. Chapter 5). The catalog of formal, model-based MSD requirement patterns for functional (safety) requirements supports requirements engineers and safety managers in building a high-quality requirements specification for a safe system. The integration of model-based requirements with natural language requirements fosters the completeness and consistency of requirements specifications whilst providing flexibility to specify requirements in the respectively most suitable way.

The following Chapter 5 contains the details of the automated safety analysis and ASIL allocation that is based on the function hierarchy and MSD requirements described in this chapter. In Chapter 6 an automatic documentation of safety arguments is elaborated that links the hazards from the environment, the functions from the function hierarchy and the corresponding MSD requirements.

5

SAFETY ANALYSIS AND ASIL ALLOCATION ON FUNCTIONAL SAFETY REQUIREMENTS

In Steps 2.3 and 2.4 of the ASIL tailoring process (cf. Figure 5.1), a safety analysis is performed on the function hierarchy to allocate tailored ASIL values to the functions. This chapter describes the details of these process steps and the used methods.

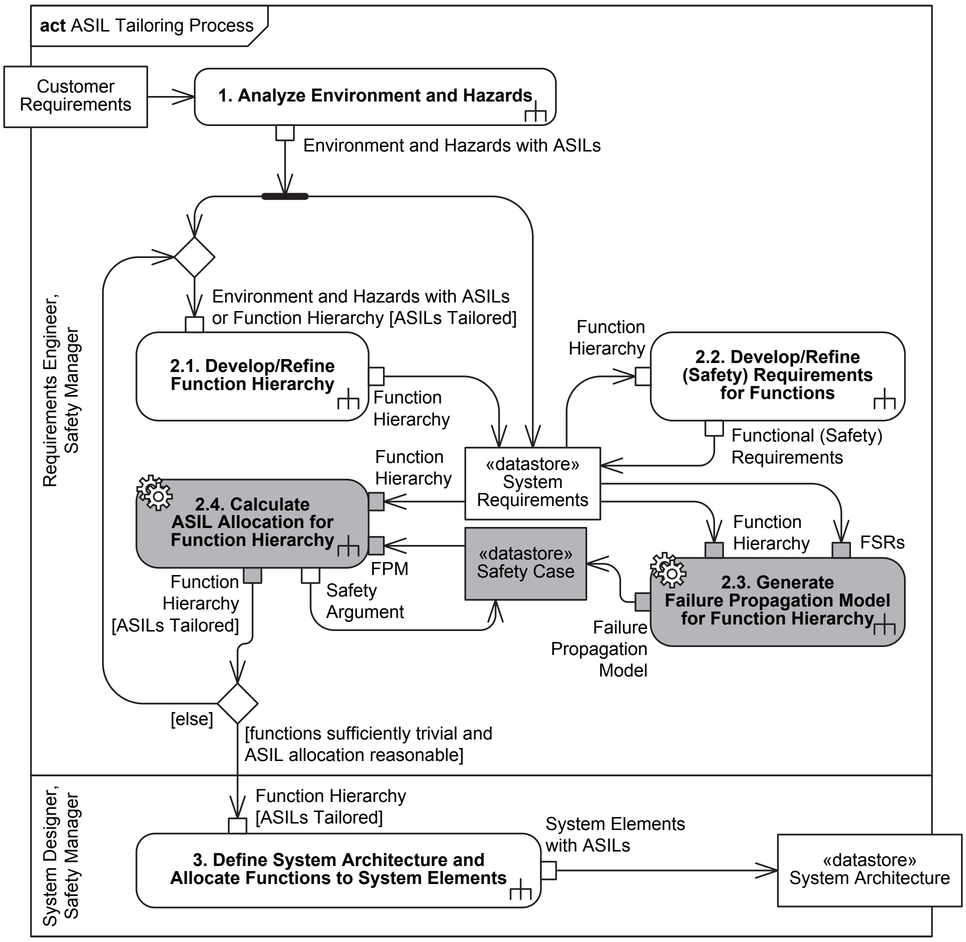

act ASIL Tailoring Process
Customer Requirements
1. Analyze Environment and Hazards
Environment and Hazards with ASILs
Environment and Hazards with ASILs or Function Hierarchy [ASILs Tailored]
Requirements Engineer, Safety Manager
2.1. Develop/Refine Function Hierarchy
Function Hierarchy
Function Hierarchy
2.2. Develop/Refine (Safety) Requirements for Functions
Functional (Safety) Requirements
Function Hierarchy
«datastore» System Requirements
2.4. Calculate ASIL Allocation for Function Hierarchy
FPM
«datastore» Safety Case
Function Hierarchy
FSRs
2.3. Generate Failure Propagation Model for Function Hierarchy
Function Hierarchy [ASILs Tailored]
Safety Argument
Failure Propagation Model
[else]
[functions sufficiently trivial and ASIL allocation reasonable]
System Designer, Safety Manager
Function Hierarchy [ASILs Tailored]
3. Define System Architecture and Allocate Functions to System Elements
System Elements with ASILs
«datastore» System Architecture

Figure 5.1: ASIL tailoring process with highlighted contents of Chapter 5

This chapter is structured as follows. First, Section 5.1 summarizes the scientific contributions of safety analysis and ASIL allocation on functional safety requirements level. Section 5.2 explains Step 2.3 where a failure propagation model is generated from the function hierarchy and the functional (safety) requirements. In Section 5.3, Step 2.4 is described where the failure propagation model is used to calculate valid ASIL tailorings and allocate ASILs to the functions of the function hierarchy. Section 5.4 lists assumptions and limitations of the presented safety analysis and ASIL allocation methods. In Section 5.5, related work is discussed. The final Section 5.6 concludes this chapter.

5.1 Contributions

The contributions of this chapter can be summarized as follows:

- Definition of a failure propagation meta model with static semantics for specifying different types of failures and their propagation through functions and functional (safety) requirements to the occurrence of hazards.
- Translation of ISO 26262's informal ASIL tailoring rules to the failure propagation meta model to enable automatic reasoning about validity of applied ASIL tailoring measures.
- Automatic derivation of failure propagation models from formal requirements to reduce the effort and mistakes in safety analysis on functional requirements (Challenge C2 in Section 1.2).
- Automatic ASIL allocation to functions and functional (safety) requirements to reduce the effort and mistakes in ASIL allocation and tailoring on functional requirements (Challenge C2 in Section 1.2).
- Support for early identification of ASIL tailoring options, decision-making, and planning of safety measures (Challenges C1 and C2 in Section 1.2).

5.2 Safety Analysis on Functional Requirements

In Step 1 of the ASIL tailoring process, hazards and their criticalities (ASIL) are identified (cf. Section 3.3). Hazards are caused by failures of the system. Hence, safety measures have to be taken to avoid failures or mitigate their consequences. In Steps 2.1 and 2.2 of the ASIL tailoring process, the functionality of the system under development is decomposed as function hierarchy, and corresponding requirements on the functions are specified as Modal Sequence Diagrams (cf. Chapter 4). This supports the requirements engineer in handling system complexity during analysis and validation of safety requirements.

Depending on a hazard's ASIL, safety measures require a different level of effort. To plan safety efforts for individual system functions, it is required to know which hazards they are involved in. Consequently, the failure propagation from failures of a function through the function hierarchy to the hazards has to be determined.

A well-established method to determine failure propagation paths from a hazard down to its root cause failures is the Fault Tree Analysis (FTA, cf. Section 2.4.1). It is suitable for the analysis of systems that are complex (cf. Challenge C1 in Section 1.2) and newly developed from scratch (e.g., CPS like autonomous driving). Furthermore, it is also suitable for analyzing combinations of failures and for partitioning and allocating safety requirements. Other analysis methods like Event Tree Analysis [Eri05], FMEA [Int06a], or HAZOP [Int01] are considered unsuitable or are not recommended for this combination of purposes. Markov analysis [Int06c] and petri-net analysis are suitable but require higher expertise and are less accepted than FTA. [Int03, Table 2]

Traditional fault trees are not strongly connected to the model that is analyzed. Specifically, they are not necessarily structured consistent to the model and provide no means to follow the hierarchical structure of a function hierarchy. Hence, we use Component Fault Trees (cf. Section 2.4.2) in this thesis that add these features to traditional fault trees. In this way, we gain a direct traceability between failures and inputs/outputs of functions.

In Step 2.3, a failure propagation model in form of Component Fault Trees is generated from the function hierarchy and the functional (safety) requirements. Figure 5.2 shows the sub-actions of Step 2.3. Initially, in Step 2.3.1, the safety manager specifies a so-called Hazard CFT for each hazard. It defines what (combinations of) top-level function failures cause the respective hazard. Afterward, in Step 2.3.2, CFTs for the sub-functions of the function hierarchy are automatically generated based on the requirements specified as MSDs. These CFTs describe the failure propagation through the functions to the manually specified failures of the Hazard CFTs. When changes to the function hierarchy or MSDs are made in further iterations of the ASIL tailoring process, only the automated Substep 2.3.2 needs to be repeated.

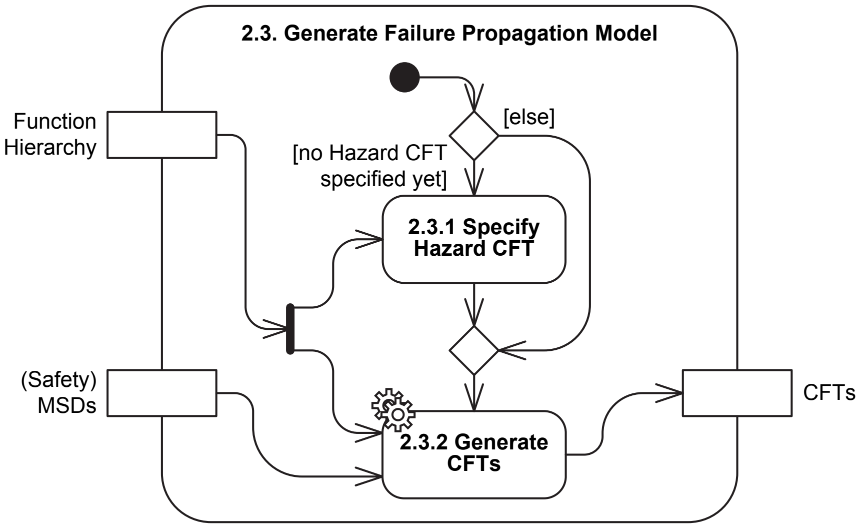


Figure 5.2: Process Step 2.3 - Generate Failure Propagation Model for Function Hierarchy

The following Section 5.2.1 describes how CFTs and different types of failures are specified in this thesis. The details of Step 2.3.1 and Hazard CFTs are explained in Section 5.2.2. The CFT generation of Step 2.3.2 is elaborated in Section 5.2.3.

5.2.1 Component Fault Tree Meta Model and Profile

To be able to automatically reason about the failure propagation of a function hierarchy for ASIL tailoring, we need a definition of the CFT language. Figure 5.3 shows the meta model for CFTs with modeling constraints (C_1 to C_{10}) as used in this thesis. This meta model is an extension of the CFT meta model presented in [HTZ$^+$12]. The existing elements are depicted with gray background color and the new elements and constraints have a white background. CFTs can be hierarchically structured, such that a CFT can contain subCFTs. Furthermore, a CFT can contain a set of Gates, i.e., Input Failure Modes, Output Failure Modes, Basic Events, Or Gates and And Gates (cf. Section 2.4.2).

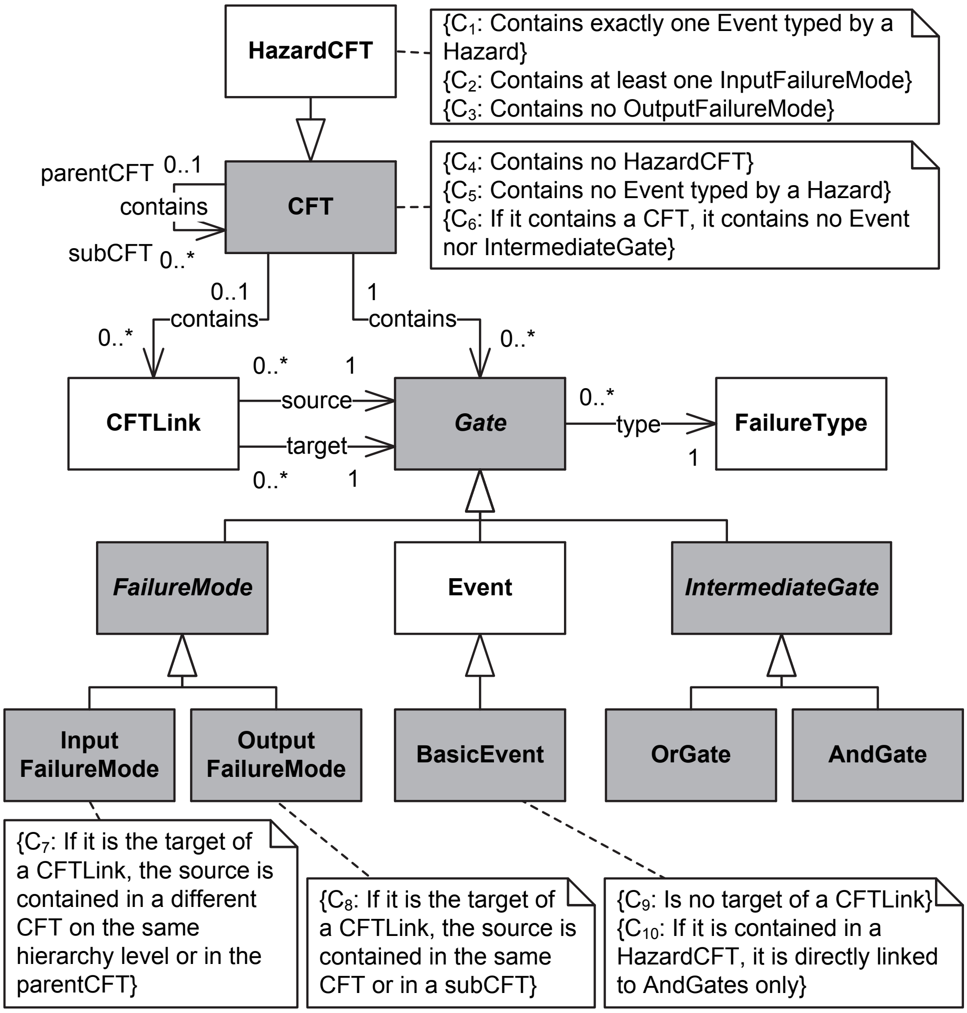

HazardCFT
{C1: Contains exactly one Event typed by a Hazard}
{C2: Contains at least one InputFailureMode}
{C3: Contains no OutputFailureMode}
parentCFT 0..1
contains
subCFT 0..*
CFT
{C4: Contains no HazardCFT}
{C5: Contains no Event typed by a Hazard}
{C6: If it contains a CFT, it contains no Event nor IntermediateGate}
0..1
contains
0..*
1
contains
0..*
0..*
1
CFTLink
source
target
0..*
1
Gate
0..*
type
1
FailureType
FailureMode
Event
IntermediateGate
Input FailureMode
Output FailureMode
BasicEvent
OrGate
AndGate
{C7: If it is the target of a CFTLink, the source is contained in a different CFT on the same hierarchy level or in the parentCFT}
{C8: If it is the target of a CFTLink, the source is contained in the same CFT or in a subCFT}
{C9: Is no target of a CFTLink}
{C10: If it is contained in a HazardCFT, it is directly linked to AndGates only}

Figure 5.3: Extended meta model for Component Fault Trees (based on [HTZ$^+$12])

In addition to these elements, we add Events. In traditional fault trees, they are used to specify the main event (i.e., the fault tree root node) whose causes are analyzed by a FTA. In this thesis, hazards are the main events whose causes are analyzed. We specify a CFT for each function of the function hierarchy. As a hazard is not part of the system's functionality, we specify a special Hazard CFT for each hazard. Each Hazard CFT contains an Event typed by the hazard and specifies what Output Failure Modes of the top-level CFT cause the hazard (cf. constraints C_1 to C_6 and C_{10}). In contrast to [HTZ$^+$12], we specify the connections between Gates explicitly in the meta model as CFT Link. This allows to specify constraints, in what situations different types of Gates may be linked (cf. constraints C_7 to C_9).

Furthermore, we add Failure Types as type for Gates. Thereby, failures like omission and commission can be distinguished. Figure 5.4 shows the failure types used in this thesis. The failure types are based on the failure type hierarchy of Giese et al. who distinguish service, value, and timing failures [GTS04] (also defined in [ALR$^+$04]):

- Omission (O): An expected event does not occur.
- Commission (C): An event occurs unexpectedly or unintentionally.
- Crash (Cr): A function stops to deliver service (e.g., software crashes).
- Value (V): An event occurs with an unexpected content (e.g., parameter value).
- Timing Early (TE) / Late (TL): An event occurs too early / too late.

For a better diagram overview, we use the abbreviations shown below each failure type in this thesis, and also depict them inside the triangular failure modes of the CFT concrete syntax.

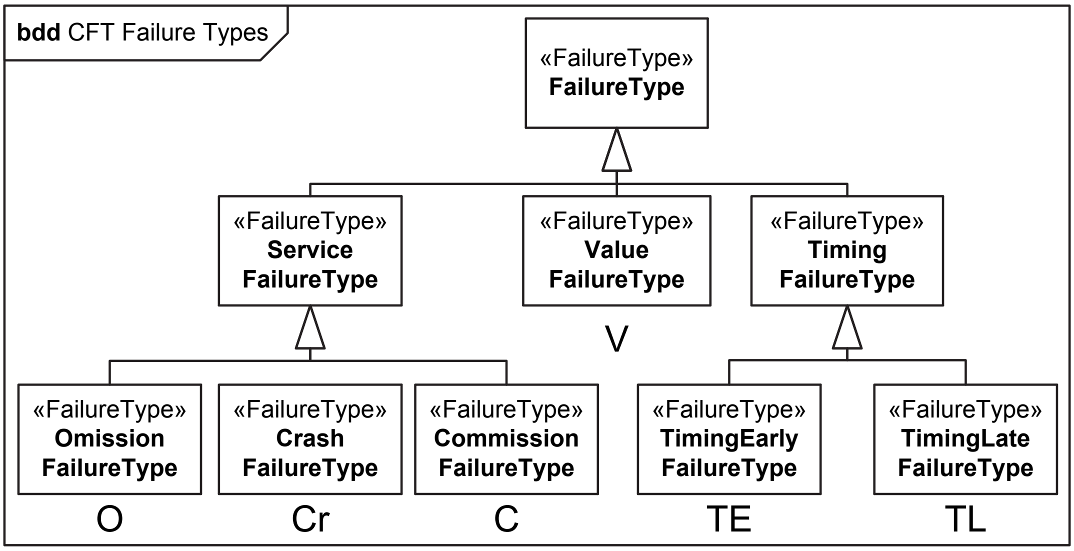

bdd CFT Failure Types
«FailureType» FailureType
«FailureType» Service FailureType
«FailureType» Value FailureType
V
«FailureType» Timing FailureType
«FailureType» Omission FailureType
O
«FailureType» Crash FailureType
Cr
«FailureType» Commission FailureType
C
«FailureType» TimingEarly FailureType
TE
«FailureType» TimingLate FailureType
TL

Figure 5.4: Failure type library model for gates

The input models for generating the CFT failure propagation model are specified in SysML/UML. The function hierarchy is specified using the SysML4CONSENS profile and MSDs using a profile for UML sequence diagrams. To model failure propagation in the same language with easy traceability to the other models and to be able to use the same tool, we consequently specify a UML profile for Component Fault Trees. Figure 5.5 shows the CFT profile definition derived from the meta model in Figure 5.3, and Figure 5.6 shows an example CFT in abstract syntax (bare application of the CFT UML profile) and in the concrete syntax used throughout this thesis (an integration of the CFT syntax and UML). A CFT is a special UML Class whose Ports represent Failure Modes. Output failure modes are specified as conjugated ports, input failure modes are not conjugated. Events, Basic Events, OR-gates, and AND-gates are specialized UML Properties. All gates are connected via directed CFT Links specializing UML Dependencies. The CFT meta model constraints

C_1 to C_{10} are realized as OCL constraints in the profile. In the CFT meta model, gates are typed by failure types. When using the UML profile, a Gate's failure type is specified by the UML type of its Port or Property. The failure types shown in Figure 5.4 are specified using the UML profile. We specify the failure types in a library model, not as part of the UML profile, to make them extensible for different application domains (e.g., [MZH$^+$16]).

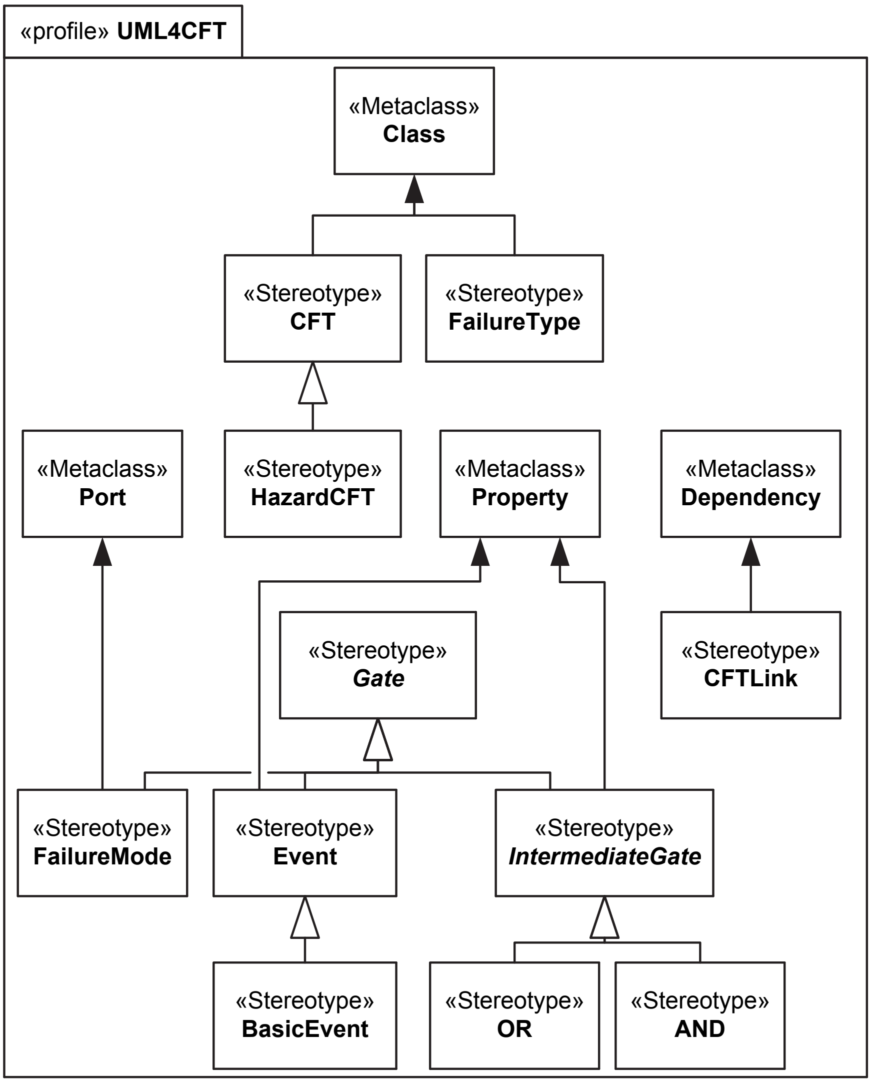


Figure 5.5: UML Profile for Component Fault Trees

A CFT specifies the failure propagation of a function. The CFT Make Decision CFT in Figure 5.6 specifies the failure propagation through the function Make Decision. Its failure modes represent failures of the corresponding function's input and output information grouped by the function's port interfaces. The failure mode activateEmergencyBraking:∼O specifies that the function Make Decision does not send the information activateEmergencyBraking over its port p4 when expected. It is caused either by a crash :Cr of the function or by the two omission failures reachingLastPointToBrake:O and receivedEmcyBrakeWarning:O occurring at once (specified by the combining AND-gate).

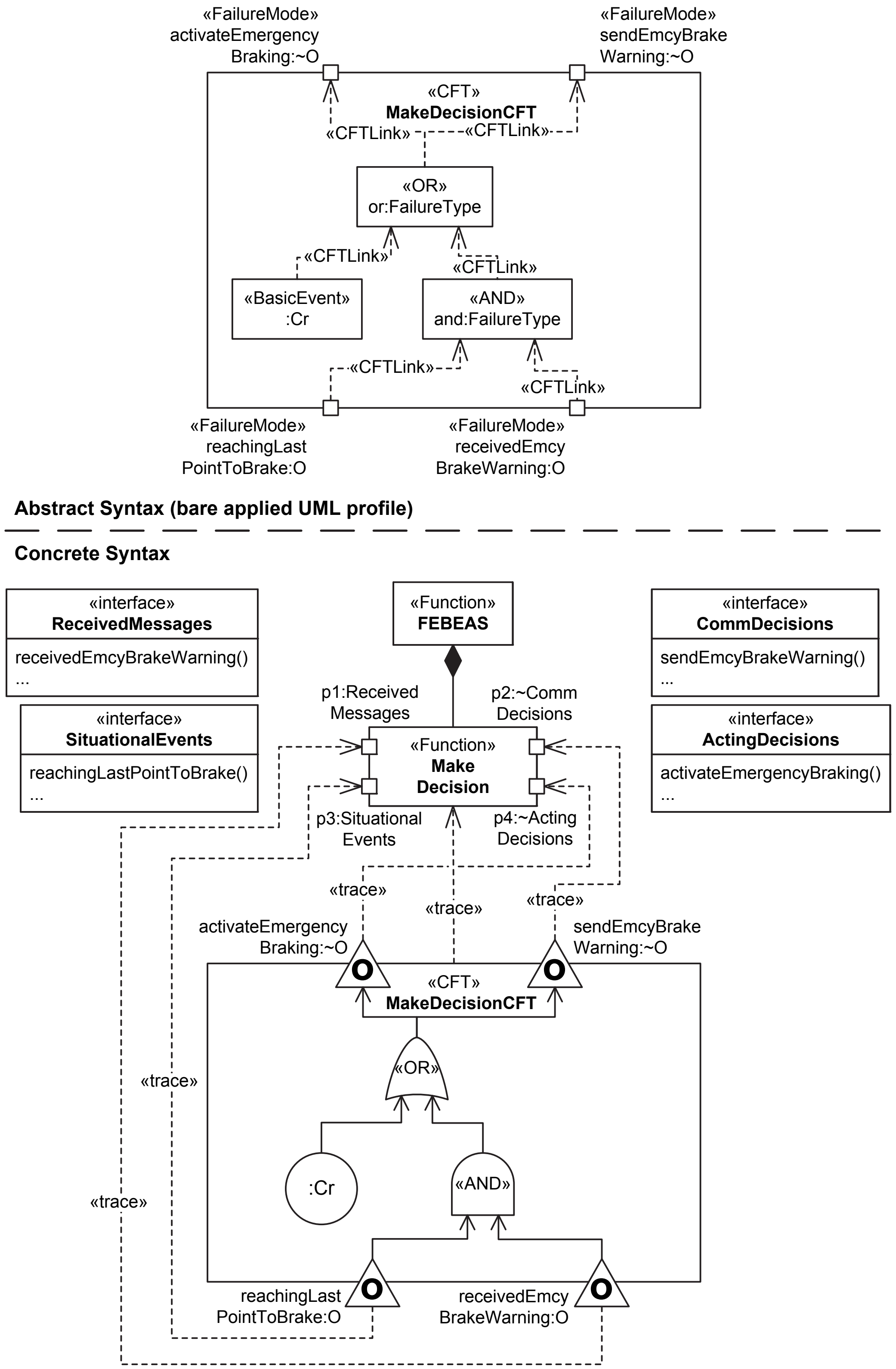


Figure 5.6: Example CFT for function Make Decision in concrete and abstract syntax

5.2.2 Linking Hazards to Failures

In Step 1 of the ASIL tailoring process, hazards are specified, and in Step 2.1, the function hierarchy is specified. Hazards occur if the system under development fails to deliver its service, i.e., does not provide its functionality as expected. So, in terms of the function hierarchy, a hazard occurs if an output information of the top-level function has a failure (e.g., is omitted when expected).

Figure 5.7 shows an example hazard from Step 1 and its relation to CFTs and the function hierarchy from Step 2.1. The hazard Hard Braking Omission describes that the vehicle does not perform a hard braking in case of an obstacle, because the EBEAS fails to send a respective brake request to the ESC (cf. Figure 3.4 on page 44). In Step 2.3.1 of the tailoring process, this relation between hazard and brake request failure is specified by the safety manager in form of the Hazard CFT Hard Braking Omission CFT. It contains a CFT event :HardBrakingOmission that represents the occurrence of the hazard (because its type is set to the hazard). The brake request is represented as the information performHardBraking sent by the top-level function FEBEAS. Therefore, the safety manager specifies its failure as input omission failure mode performHardBraking:O of the Hazard CFT, and links it to the CFT event representing the hazard occurrence.

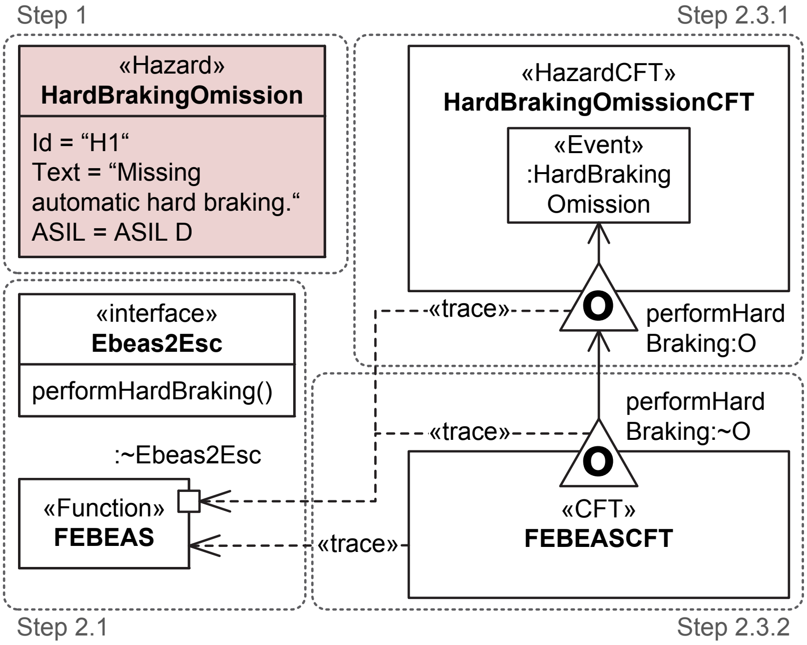


Figure 5.7: Example Hazard CFT for hazard Hard Braking Omission

Hazard CFTs can contain the same gates as CFTs (except for output failure modes). If a hazard is caused by different failures or only by certain combinations of failures, this can be specified by using OR and AND gates. The CFTs for the function hierarchy (FEBEAS CFT in Figure 5.7) with their failure modes and internal structure are automatically generated and linked to the failure modes of Hazard CFTs in Step 2.3.2 of the tailoring process. The details of this step are described in the following section.

5.2.3 Generating Component Fault Trees

We express the failure propagation through the function hierarchy by Component Fault Trees. In Step 2.3.2 of the ASIL tailoring process, they are automatically derived from the functional (safety) requirements specified as Modal Sequence Diagrams (cf. Section 4.4). Figure 5.8 sketches this CFT generation by an example. The top diagram is an MSD that specifies the requirement that the function Make Decision shall decide to activate emergency braking once it is informed that the vehicle is reaching the last point to brake. The bottom diagram shows the function's corresponding CFT Make Decision CFT with its failure modes and internal elements generated from the MSD above.

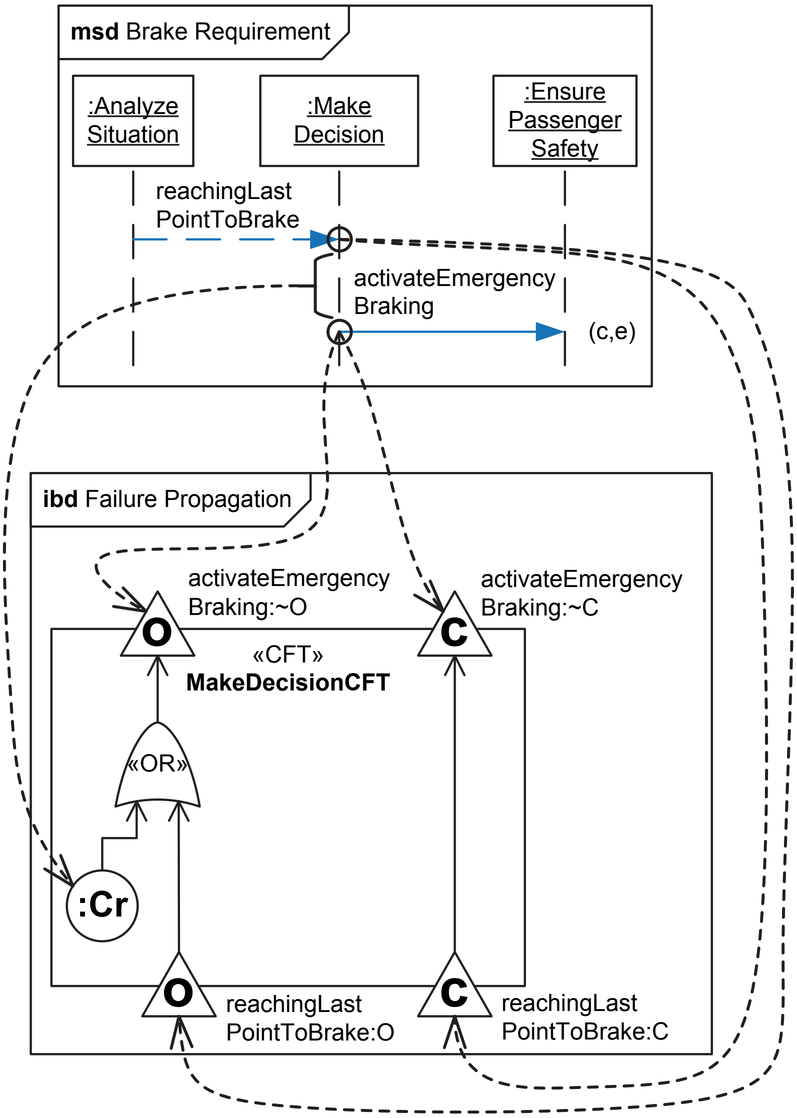

msd Brake Requirement
:Analyze Situation
:Make Decision
:Ensure Passenger Safety
reachingLast PointToBrake
activateEmergency Braking
(c,e)
ibd Failure Propagation
activateEmergency Braking:~O
activateEmergency Braking:~C
O
C
«CFT»
MakeDecisionCFT
«OR»
:Cr
O
C
reachingLast PointToBrake:O
reachingLast PointToBrake:C

Figure 5.8: Example of CFT generation for function Make Decision

If the function Make Decision fails when the vehicle reaches the last point to brake, it will not send the information activateEmergencyBraking. In other words, the information activateEmergencyBraking is omitted (O) if the function crashes (Cr) between the receiving of the information reachingLastPointToBrake and the sending of activateEmergencyBraking. This is specified by the basic event :Cr linked to the output failure mode activateEmergencyBraking:∼O. In addition, if the information reachingLastPointToBrake is not received by Make Decision, the function will not send the information activateEmergencyBraking. This is specified by the input failure mode reachingLastPointToBrake:O linked to activateEmergencyBraking:∼O. As both the crash and the input failure cause the output omission failure individually (single-point failures), they are connected to the output failure via an OR gate.

If the information reachingLastPointToBrake is received unexpectedly by Make Decision because Analyze Situation sends it unintentionally (C), Make Decision will send the information activateEmergencyBraking as normal reaction adhering to the requirement. This propagates the commission failure from Analyze Situation through Make Decision. It is specified by the input failure mode reachingLastPointToBrake:C linked to the output failure mode activateEmergencyBraking:∼C.

Section 4.4.2 lists patterns of functional requirements and functional safety requirements expressible with MSDs. These patterns were consolidated from different sources that identified them in industry requirements specifications. Hence, we assume these patterns represent all relevant types of functional (safety) requirements. Consequently, we assume that all MSD specifications, that we derive CFTs from, solely consist of MSDs applying those patterns. From a systematic analysis of all MSD requirement patterns we derived rules for the generation of CFTs as sketched in Figure 5.8. For omission and commission failures we identified eight cases of MSDs in the patterns that we group into *positive* and *negative* MSDs. Positive MSDs specify information exchange that *shall* occur. Negative MSDs specify information exchange that *must not* occur. Negative MSDs can be identified by a contained *hot-false-condition*, i.e., a hot condition with the expression “false” that can never evaluate to true. Figure 5.9 shows the four positive MSD cases and the respective CFT elements. Figure 5.10 shows the four negative MSD cases and the respective CFT elements.

Case 1 specifies the requirement excerpt “If the considered lifeline :L sends the information q, followed by sending/receiving arbitrary further information, then :L has to send the information p”. In this case, :L might crash between sending q and p resulting in an omission failure of p. Case 1 results in no commission failure of p because :L receives no information that could have a commission failure that would propagate to p. A commission failure of p would have the same cause as the commission failure of q, and that cause is not visible in this case’s MSD.

Case 2 is similar to Case 1 with the difference that initially the considered lifeline :L *receives* the information q. In that case, again :L might crash between q and p resulting in an omission failure of p. In addition, that omission might also be caused by an omission of the ingoing information q. If q has a commission failure, it will propagate through L to a commission failure of p, because that information would be correctly sent according to the MSD requirement.

Case 3 specifies the requirement excerpt “If the considered lifeline :L sends/receives one or more information, followed by receiving the information s, followed by sending/receiving arbitrary further information, then :L has to send the information p”. This case describes the fact that a required information p (then-part of the requirement) would not be sent by :L if an information on its prechart (if-part of the requirement) is not received by :L. The prechart

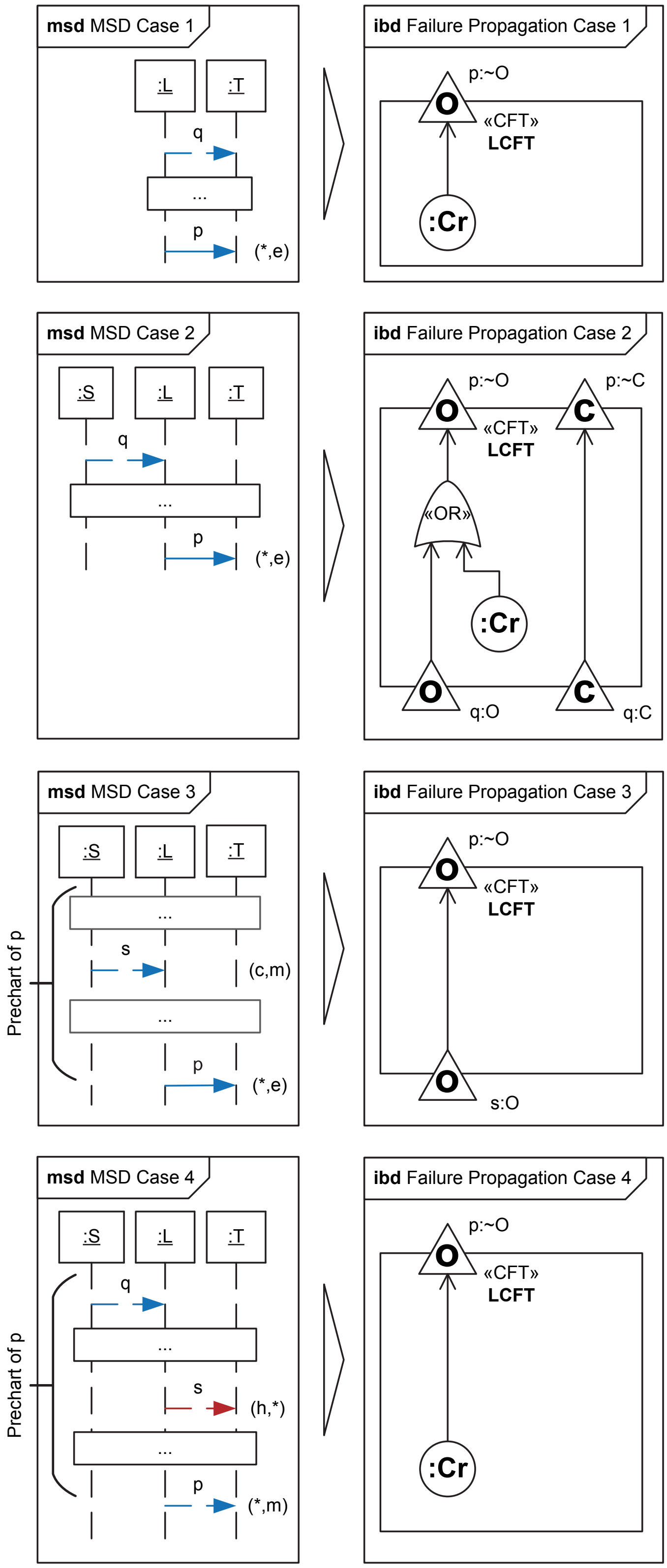


Figure 5.9: CFT generation cases for positive MSDs

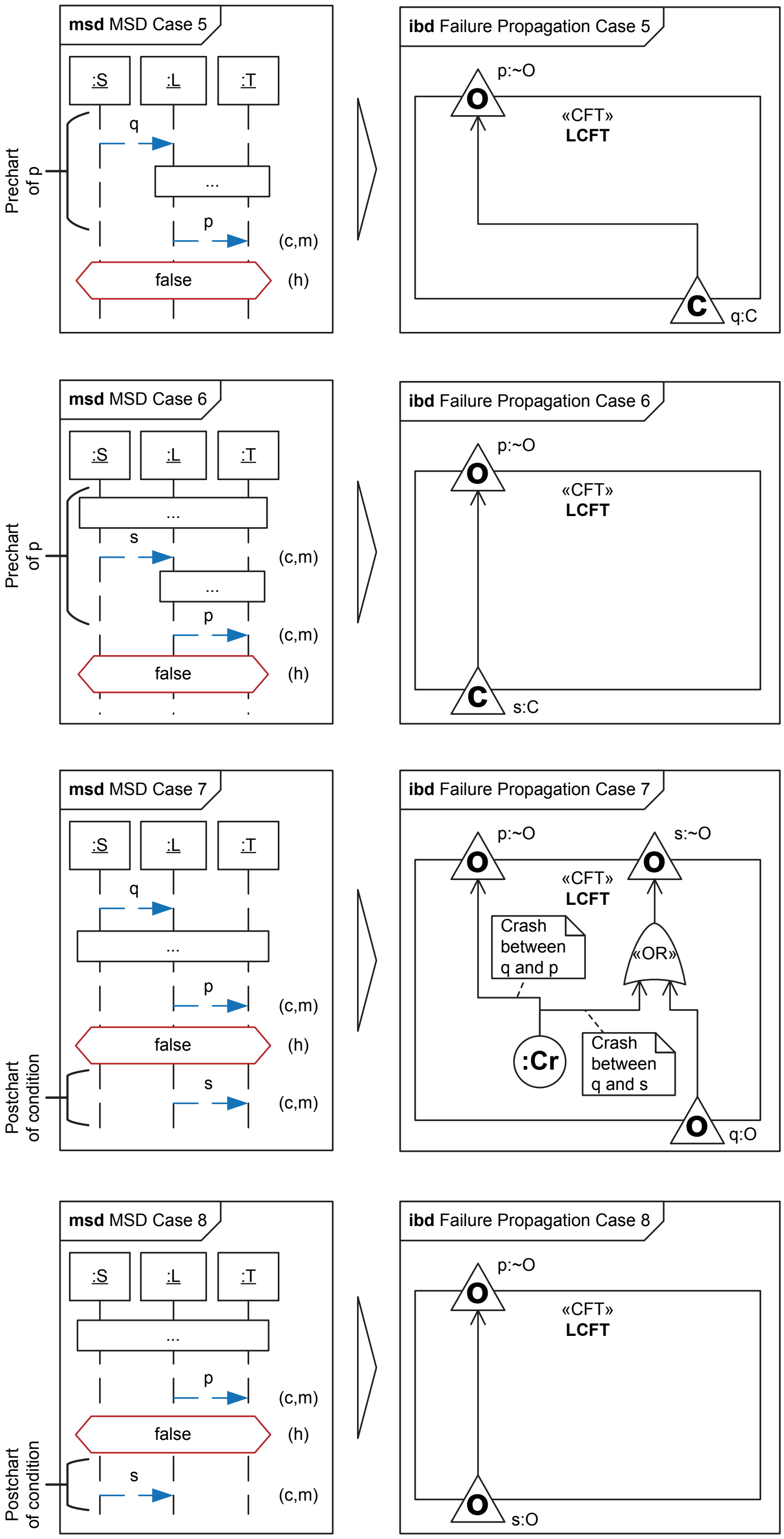


Figure 5.10: CFT generation cases for negative MSDs

of an MSD element consists of all elements above it on all lifelines. Case 3 results in no commission failure of p because s is not the minimal message of the MSD that triggers the evaluation of the requirement. The commission of s without the commission of the MSD's minimal message would not lead to p being sent unintentionally.

Case 4 specifies the requirement excerpt "If the considered lifeline :L receives the information q, followed by sending/receiving arbitrary further information, followed by sending the information s in strict order related to all other information in the MSD, followed by sending/receiving arbitrary further information, then the information p may be sent by :L". In this case, :L might crash between sending q and p resulting in an omission failure of p (although p is not required because it is monitored). If p is required by some other MSD, it is blocked by this MSD once the execution reaches s. Then, p is not allowed to occur until s occurred. If :L crashes before s occurs, p will not be sent. Case 4 results in no commission failure of p because it is not required to occur (not executed but monitored), it is only restricted.

The Cases 1 to 4 prescribe the rules for generating CFTs from positive MSDs. These rules are aggregated as pseudo code in Algorithm 5.1. This algorithm was applied in Figure 5.8. The top diagram in the figure is a Case 2 MSD. The failure mode activateEmergencyBraking:∼O and the the basic event :Cr are created and linked by line 6 of Algorithm 5.1. The failure mode reachingLastPointToBrake:O is created and linked to the already created failure mode activateEmergencyBraking:∼O by line 9. The failure modes activateEmergencyBraking:∼C and reachingLastPointToBrake:C are created and linked by line 10.

Algorithm 5.1 Creation of O and C failures from positive MSDs

```
 1: $m$ := the considered positive MSD
 2: for all lifelines $l$ of $m$ do
 3:     for all messages $p$ sent by $l$ do
 4:         if $p$ is executed then
 5:             if the minimal message of $m$ is received/sent by $l$ then      ▷ Cases 1 and 2
 6:                 Create failure mode $p$:∼O caused by basic event :Cr of $l$
 7:             end if
 8:             if the minimal message $q$ of $m$ is received by $l$ then         ▷ Case 2
 9:                 Create failure mode $p$:∼O caused by failure mode $q$:O
10:                 Create failure mode $p$:∼C caused by failure mode $q$:C
11:             end if
12:             for all (c,m) messages $s$ received by $l$ on the prechart of $p$ do   ▷ Case 3
13:                 Create failure mode $p$:∼O caused by failure mode $s$:O
14:             end for
15:         else                                                           ▷ $p$ is monitored
16:             if the prechart of $p$ contains a hot message sent by $l$ then    ▷ Case 4
17:                 Create failure mode $p$:∼O caused by basic event :Cr of $l$
18:             end if
19:         end if
20:     end for
21: end for
```

The four cases for negative MSDs are shown in Figure 5.10. Case 5 specifies the requirement excerpt "If the considered lifeline :L receives the information q, followed by sending arbitrary further information, then :L must not send the information p". In this case, the information p might be unintentionally forbidden if the last input to :L q (which is also the minimal message) is unexpectedly received by :L. Therefore, a commission failure of q results in an omission failure of p.

Case 6 specifies the requirement excerpt "If the considered lifeline :L sends/receives one or more information, followed by receiving the information s, followed by sending arbitrary further information, then :L must not send the information p". This case is similar to Case 5, except that s is not the minimal message but any other last input to :L on the prechart of p. Accordingly, the omission failure of p is caused by this last input s instead of a minimal message that is possibly received by :L.

Case 7 specifies the requirement excerpt "If the considered lifeline :L receives the information q, followed by sending/receiving arbitrary further information, then :L must not send the information p unless it sends s before". This case describes the scenario that p is forbidden after the minimal message q until any information s from the postchart of the hot-false-condition is sent. The postchart of an MSD element consists of all elements below it on all lifelines. In this case, p is only unintentionally forbidden if no information s from the postchart of the hot-false-condition is sent. That is only the case if L crashes. Hence, an omission failure of p is caused by a crash of L. Any information s on the postchart of the hot-false-condition might be omitted if the minimal message q is not occurring and triggering this requirement or if :L crashes before it can send s.

Case 8 specifies the requirement excerpt "If the considered lifeline :L sends/receives one or more information, then :L must not send the information p until it receives s". In this case, p might be forbidden longer than intended if any information s on the postchart of the hot-false-condition is unexpectedly not received. Hence, an omission failure of p is caused by an omission failure of s.

All negative MSD cases do not lead to output commission failures because they are only restricting information exchange (i.e., negative MSDs contain no executed messages). The Cases 5 to 8 prescribe the rules for generating CFTs from negative MSDs. These rules are aggregated as pseudo code in Algorithm 5.2.

Algorithms 5.1 and 5.2 create failure modes, basic events, and their connections treating every input failure mode and basic event as single-point failure. However, some requirements may be redundant such that two or more input failures have to occur at the same time for an output failure to occur (multiple-point failure). Single-point failures are merged by an OR gate, whereas multiple-point failures have to be merged by an AND gate. Hence, we apply another transformation step (Algorithm 5.3) to identify redundant requirements and update the CFT internals accordingly.

Figure 5.11 shows an example of redundant requirements, the resulting CFT after execution of Algorithm 5.1, and the final CFT with integrated AND gates by Algorithm 5.3. The first MSD G'$_1$ specifies the requirement that mf shall be sent if mg1 is received. The second MSD G'$_2$ specifies that mf shall be sent if mg2 is received. So, these two MSDs form redundant requirements with the two inputs mg1 and mg2 that redundantly require the sending of mf. The third MSD G'$_3$ specifies the requirement that mf shall be sent if mg3 is received and followed by mg4. This is a third redundant requirement with the difference that

Algorithm 5.2 Creation of O and C failures from negative MSDs

```
$m$ := the considered negative MSD
$c$ := the hot-false-condition of $m$
$p$ := the message directly followed by $c$
$l$ := the lifeline $p$ is sent by
if postchart of $c$ is empty then
    for all messages $s$ received by $l$ on the prechart of $p$ do    ▷ Cases 5 and 6
        if $s$ is the last ingoing message of $l$ on the prechart of $p$ then
            Create failure mode $p$:∼O caused by failure mode $s$:C
        end if
    end for
else    ▷ There are messages after $c$
    if the postchart of $c$ contains messages sent by $l$ then    ▷ Case 7
        Create failure mode $p$:∼O caused by basic event :Cr of $l$
        for all messages $s$ on the postchart of $c$ sent by $l$ do
            Create failure mode $s$:∼O caused by basic event :Cr of $l$
            if the minimal message $q$ of $m$ is received by $l$ then
                Create failure mode $s$:∼O caused by failure mode $q$:O
            end if
        end for
    end if
    for all messages $s$ on the postchart of $c$ received by $l$ do    ▷ Case 8
        Create failure mode $p$:∼O caused by failure mode $s$:O
    end for
end if
```

Algorithm 5.3 Creation of AND gates for ∼O failures

```
for all ∼O failures $f$ of a CFT $c$ do
    $I$ := all input failure modes of $c$ on the path to $f$
    $G$ := failure modes of $I$ grouped by MSD that they originate from
    if $G$ contains more than one group then    ▷ AND gate(s) required
        Remove the current links of all failure modes $g \in G$ towards $f$
        $A := \{\}$    ▷ The set of created AND gates
        $P := \prod_{G' \in G} G'$
        for all cartesian products of failure modes $P' \in P$ do
            Link all $g \in P'$ to a new AND gate $a$
            Add $a$ to $A$
        end for
        Link all $a \in A$ to output failure mode $f$ (or its preceding OR gate if existing)
    end if
end for
```

the combination of the two inputs mg3 and mg4 is redundant to the inputs mg1 and mg2 from the other MSDs.

Algorithm 5.1 identifies Case 2 in all three MSDs and additionally Case 3 in MSD G'$_3$. Therefore, its result is the intermediate CFT in the middle of Figure 5.11 with a crash basic event and four input omission failure modes for each of the four input messages. Each of these failures could be involved in an omission of the output mf, and thus they are all connected to the output omission failure mf:$\sim$O via an OR gate.

Algorithm 5.3 separates the four input failure modes into three groups according to the MSDs that they originate from. Because each MSD is redundant to the others, a failure from each group of failure modes has to occur simultaneously for the output failure to occur. Hence, the algorithm creates an AND gate for each combination (cartesian product) of redundant failure modes. The CFT in the bottom of Figure 5.11 shows the result of Algorithm 5.3 which is the final result of the CFT generation with consideration of redundant requirements.

The prototypical implementation of the algorithms is described in Section 7.1. That implementation was used to validate that the transformation rules (the eight cases) are complete and to verify the correctness of the algorithms by testing the implementation against the patterns of the MSD pattern catalog (cf. Section 4.4.2) and the expected CFT outputs.

5.3 ASIL Allocation on the Function Hierarchy

The function hierarchy decomposes the functionality of the system under development and structures the functional (safety) requirements. The CFTs generated from the function hierarchy and its requirements specify the failure propagation from input and basic failures of the system to output failures causing hazards. Different hazards have different criticality specified by their ASIL value. In general, each function has to be developed in accordance with the maximum ASIL of all identified hazards. Thereby making sure that it is developed with the required degree of rigor to avoid or mitigate the hazards. If ASIL tailoring shall be applied to develop a function with lower degree of rigor, the validity of the tailoring has to be checked.

To fulfill these requirements and find a valid ASIL allocation to functions, we calculate the ASIL propagation from hazards over the failure modes of the CFTs. Afterward, we derive the ASIL of each function based on the ASILs allocated to its corresponding CFT. All this is done automatically in Step 2.4 of the ASIL tailoring process. Figure 5.12 shows the sub-actions of that step.

In Step 2.4.1, ASILs are calculated for each failure mode of the CFTs based on the hazards that they are involved in. The details of this step are described in the following Section 5.3.1. Afterward, in Step 2.4.2, ASILs are allocated to functions and their requirements based on the ASILs calculated for the CFT failure modes. The details of this step are explained in Section 5.3.2. When changes to the function hierarchy or requirements are made in further iterations of the ASIL tailoring process, these automated steps can be repeated. In Step 2.4.3 the validity of the ASIL allocation is documented. This is explained in Chapter 6.

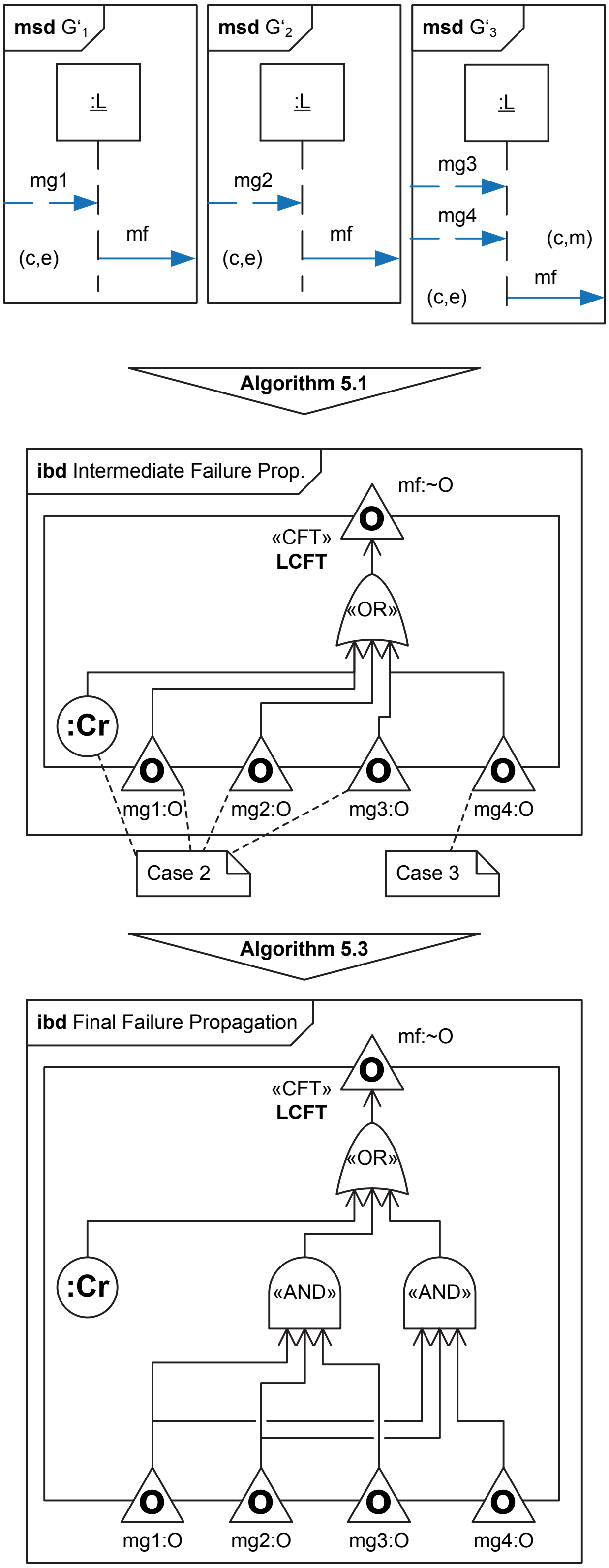


Figure 5.11: Example of CFT generation for ∼O failure and redundant MSDs

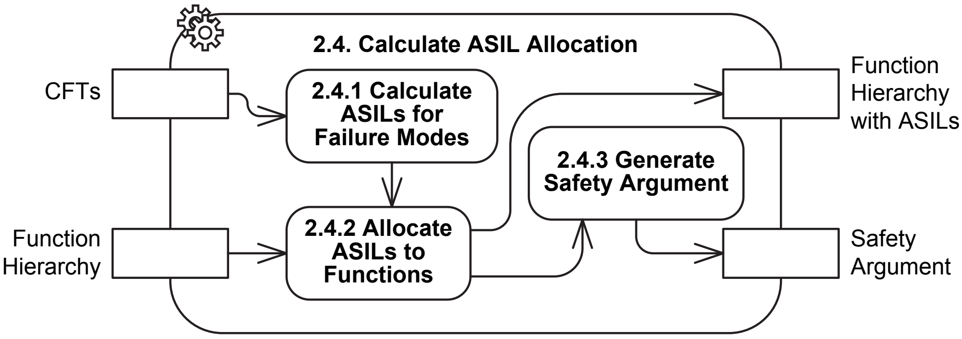

2.4. Calculate ASIL Allocation
CFTs
Function Hierarchy
2.4.1 Calculate ASILs for Failure Modes
2.4.2 Allocate ASILs to Functions
2.4.3 Generate Safety Argument
Function Hierarchy with ASILs
Safety Argument

Figure 5.12: Process Step 2.4 - Calculate ASIL Allocation for Function Hierarchy

5.3.1 Calculating ASILs on CFTs

The failure propagation model consists of hierarchically structured CFTs that represent the failure propagation of the function hierarchy. In addition, the model contains Hazard CFTs that specify what failures of the top-level CFT cause hazards. Figure 5.13 shows a sketch of a failure propagation model. The hazard h1 is caused by failure o1 of CFT c1. The hazard h2 is caused by failure o2 of CFT c1. Each hazard has an ASIL value that specifies its safety-criticality. In Figure 5.13, h1 has ASIL D and h2 has ASIL C. Thus, by default, the CFT c1 and all its sub-CFTs inherit the maximum ASIL of both hazards, i.e., ASIL D. If ASIL tailoring is applied, some CFTs can be allocated with a lower ASIL than others (cf. c3 and c5). This ASIL reduction has to follow the rules defined by ISO 26262.

5.3.1.1 ASIL Tailoring Rules

ASIL tailoring can be done by applying one of the two measures *separation* and *decomposition*. Both measures are applied on requirements. In separation, safety requirements with lower criticality are separated from requirements with higher criticality by allocating them to different functions, such that the more critical function is free of interference from the less critical function (cf. Definition 2.24 on page 25). In decomposition, a safety requirement with higher criticality is decomposed into a set of redundant requirements that are allocated to sufficiently independent functions (cf. Definition 2.26 on page 26). Hence, each redundant requirement may be considered less critical because all requirements have to fail at once to violate the original requirement.

ASIL tailoring by separation or decomposition has to obey the rules of ISO 26262 described in Section 2.5. In the following, we transfer those rules to the elements of the CFT failure propagation meta model. Each CFT specifies the failure propagation of a function. The failure propagation inside a CFT is specified by a set of linked gates and is derived from the requirements as explained in Section 5.2.3. If a CFT's output failure mode occurs, the corresponding requirement of a function is violated. Definition 5.1 specifies the basic elements of a failure propagation model and their relations.

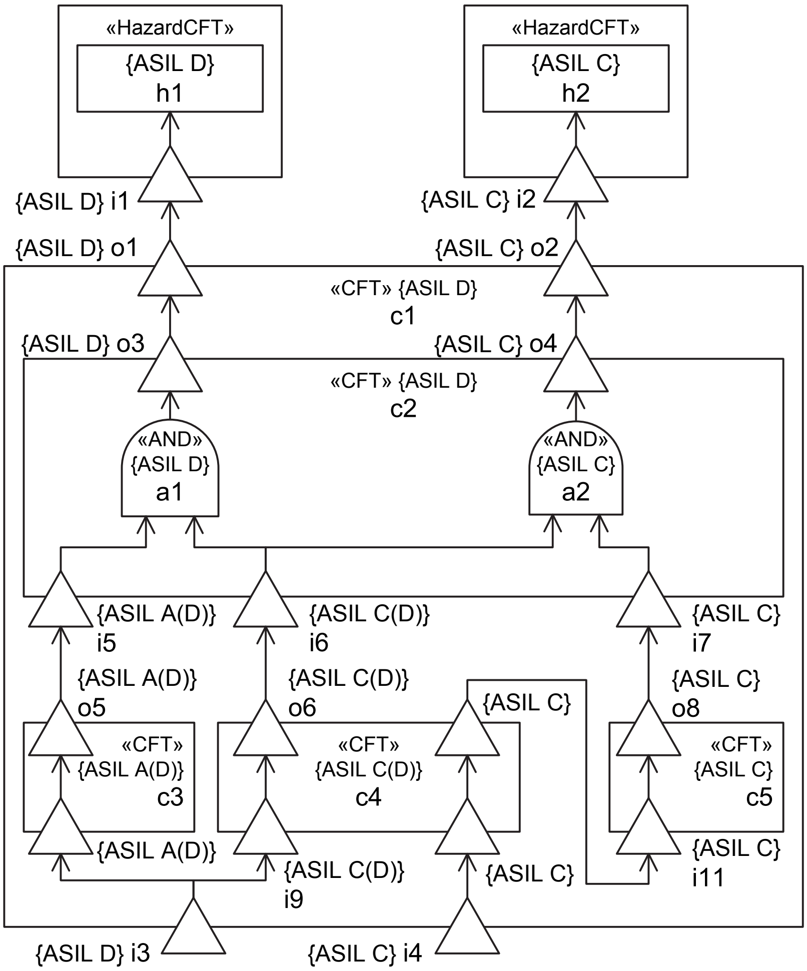


Figure 5.13: Example of failure propagation model with allocated ASILs

Definition 5.1 (CFT Elements)

A failure propagation model (cf. Figure 5.3) consists of a set of CFTs C and a set of gates G that are contained in those CFTs (G_c). Gates consist of intermediate gates, failure modes F, and events (including events H representing hazards). For failure modes of a CFT c, we distinguish input failure modes I_c and output failure modes O_c. Two gates lie on a failure propagation path ($\rightsquigarrow$) if they are directly connected via a CFT link or indirectly via a set of gates and connecting CFT links leading towards a hazard. Each gate g can be assigned with an ASIL value s_g. ASILs are represented by the integer values 0 (QM) to 4 (ASIL D). Each CFT c is assigned with the maximum ASIL of all failure modes it contains s_c.

$$\begin{aligned}
C &= \text{Set of all CFTs}\\
G &= \text{Set of all gates}\\
G_c &= \{g \in G \mid g \text{ contained in CFT } c \in C\}\\
F &= \{f \in G \mid f \text{ being a failure mode}\}\\
H &= \{h \in G \mid h \text{ being an event typed by a hazard}\}\\
I_c &= \{i \in G_c \mid i \text{ being an input failure mode}\}\\
O_c &= \{o \in G_c \mid o \text{ being an output failure mode}\}\\
\rightsquigarrow : G \rightarrow G &= \text{failure propagation path between two gates}\\
S &= \text{Set of all ASIL values assigned to gates and CFTs in } (C \cup G)\\
&\quad \text{with } s_g \in S \text{ being the ASIL} \in \{0,1,2,3,4\} \text{ assigned to gate } g \in G\\
&\quad \text{and } s_c \in S \ = \max\{s_f \mid f \in (F \cap G_c)\} \text{ being the ASIL assigned to CFT } c \in C
\end{aligned}$$

Separation Rules

Separation is defined using the terms *cascading failure* (cf. Definition 2.21 on page 22) and *freedom from interference* (cf. Definition 2.23 on page 24). A function a is separate from a function b if b is free of interference from a concerning all safety requirements with higher criticality than a's requirements. The function b is free of interference from a if there are no failures of a that lead (or cascade) to failures of b. To define these rules for valid separation on CFTs, we need to define the above terms on the CFT meta model.

DEFINITION 5.2 (CFT CASCADING FAILURE)
An output failure mode o_1 is a cascading failure from a CFT c_1 to a CFT c_2 if there exists a failure propagation path from o_1 to an input failure mode i_2 of CFT c_2.

$$\mathrm{CF}_{c_1,c_2} = \{o_1 \in O_{c_1} \mid \exists\, i_2 \in I_{c_2} : o_1 \rightsquigarrow i_2\}$$

DEFINITION 5.3 (CFT FREEDOM FROM INTERFERENCE)
A CFT c_1 interferes with a CFT c_2 concerning a gate g_3 if there exists a cascading failure o_1 from c_1 to c_2 leading to g_3.

$$\mathrm{Inter}_{c_1,c_2} = \{g_3 \in G \backslash G_{c_1} \mid \exists\, o_1 \in \mathrm{CF}_{c_1,c_2}, i_2 \in I_{c_2} : o_1 \rightsquigarrow i_2 \rightsquigarrow g_3\}$$

DEFINITION 5.4 (CFT SEPARATION)
A CFT c_1 is separate from a CFT c_2 if c_1 does not interfere with c_2 concerning any output failure mode o_2 of c_2 with an ASIL s_{o_2} higher than c_1's ASIL s_{c_1}.

$$\mathrm{Sep}_{c_2} = \{c_1 \in C \mid \neg\exists\, o_2 \in O_{c_2} : o_2 \in \mathrm{Inter}_{c_1,c_2} \wedge s_{o_2} > s_{c_1}\}$$

In Figure 5.13, the CFT c3 is separate from the CFT c5 because c3 does not interfere with c5 concerning o8. There are no cascading failures from c3 to c5. Thus, c3 may have any lower ASIL allocated than c5 (QM instead of ASIL C in this example). The CFT c5 is separate from the CFT c4 because c5 does not interfere with c4 concerning any of its output failure modes. There are no cascading failures from c5 to c4. Hence, c5 may have any lower ASIL allocated than c4 (ASIL C instead of D).

c5 is separate from c2. First, because c5 does not interfere with c2 concerning o3. There is no cascading failure from c5 to o3. Thus, c5 may have a lower ASIL allocated than o3. Second, there is a cascading failure o4 from c5 to c2's output failure mode o4, but it is not on a failure propagation path leading to o3. Thus, c5 may have a lower ASIL allocated than o3 but not lower than o4 (ASIL C instead of D).

Decomposition Rules

Decomposition is defined using the terms *common cause failure* (cf. Definition 2.22 on page 23) and *independence* (cf. Definition 2.25 on page 26). A safety requirement may be decomposed into two redundant requirements of two functions a and b if a and b are sufficiently independent. Independence requires freedom from interference between a and b in both directions concerning the requirement. In addition, it requires the absence of common cause failures of a and b (i.e., failures of both functions with the same cause). According to the definition in ISO 26262, valid decomposition requires *sufficient* independence (cf. Definition 2.26 on page 26). In our interpretation, this means that if common cause failures exist, decomposition may still be applied but the common cause has to be safeguarded with the degree of rigor before decomposition was applied (i.e., the original ASIL). To define these rules for valid decomposition on CFTs, we need to define the above terms on the CFT meta model.

DEFINITION 5.5 (CFT COMMON CAUSE FAILURE)
A failure mode f is the cause of two common cause failures (cf. Def. 2.22) i_1 and i_2 of two resp. CFTs c_1 and c_2 if there exists a failure propagation path from f to the input failure mode i_1 of c_1 and a path from f to the input failure mode i_2 of c_2.

$$\text{CCF}_{c_1,c_2} = \{f \in F \backslash (G_{c_1} \cup G_{c_2}) \mid \exists\, i_1 \in I_{c_1}, i_2 \in I_{c_2} : f \rightsquigarrow i_1 \wedge f \rightsquigarrow i_2\}$$

DEFINITION 5.6 (CFT SUFFICIENT INDEPENDENCE)
A CFT c_1 is sufficiently independent from a CFT c_2 concerning a gate g_3 if c_1 does not interfere with c_2 concerning g_3 and vice versa, and if there is no failure mode f that is the common cause of failures of both CFTs leading to g_3 with an ASIL s_f lower than g_3's ASIL s_{g_3}.

$$\begin{aligned}\text{SufIndep}_{c_1,c_2} = \{g_3 \in G \mid\ & g_3 \notin (\text{Inter}_{c_1,c_2} \cup\ \text{Inter}_{c_2,c_1}) \wedge \\ & (\neg\exists\, f \in \text{CCF}_{c_1,c_2} : f \rightsquigarrow g_3 \wedge s_f < s_{g_3})\}\end{aligned}$$

DEFINITION 5.7 (CFT DECOMPOSITION)
A CFT c_1 is part of a decomposition concerning an output failure mode o_3 of a CFT c_3 if c_1 interferes with c_3 concerning an AND gate a_3 of c_3 on a path to o_3, there are other CFTs that interfere with a_3, and c_1 is sufficiently independent from those CFTs. The sum of ASILs of all CFTs that are part of a decomposition concerning the same output failure mode o_3 must be as high as the ASIL of o_3 or higher.

$$\begin{aligned}\text{Decomp}_{c_1,c_3} = \{o_3 \in \text{Inter}_{c_1,c_3} \mid\ & \exists\ \text{AND gate}\ a_3 \in G_{c_3} : a_3 \in \text{Inter}_{c_1,c_3} \wedge a_3 \rightsquigarrow o_3 \wedge \\ & (\exists\, c_2 \in C \backslash \{c_1\} : a_3 \in \text{Inter}_{c_2,c_3}) \wedge \\ & (\forall\, c_2 \in C \backslash \{c_1\}\ \text{with}\ a_3 \in \text{Inter}_{c_2,c_3} : o_3 \in \text{SufIndep}_{c_1,c_2})\}\end{aligned}$$

$$\sum_{c \in D} s_c \geq s_{o_3} \qquad \forall\, c_3 \in C, o_3 \in O_{c_3}, D = \{c \in C \backslash \{c_3\} \mid o_3 \in \text{Decomp}_{c,c_3}\}$$

In Figure 5.13, valid decomposition was applied on CFTs c3 and c4 concerning o3 of c2 because c3 and c4 are sufficiently independent. There are no cascading failures from c3 to c4 nor vice versa. So, there is freedom from interference between them. Failure mode i3 of CFT c1 leads to common cause failures of c3 and c4 and is allocated with ASIL D of o3 to reach sufficient independence. In conclusion, c3 and c4 may have lower ASILs assigned than c2 but their sum must meet ASIL D (in the figure D(4) = A(1) + C(3)). For the CFTs c4 and c5 no valid decomposition is possible because they are not sufficiently independent. c5 is not free of interference from c4, as there is a cascading failure of c4 leading to i11.

Conclusion
In conclusion, if a CFT has a lower ASIL than others, ASIL tailoring was applied (i.e., separation or decomposition).

PROPOSITION 5.1 (CFT ASIL TAILORING)
A CFT c_1 may have a lower ASIL s_{c_1} than a CFT c_2 if c_1 is separate from c_2 or if c_1 is part of a decomposition concerning an output failure mode of c_2.

$$s_{c_1} < s_{c_2} \;\Rightarrow\; c_1 \in \mathrm{Sep}_{c_2} \vee \mathrm{Decomp}_{c_1,c_2} \neq \emptyset$$

The ASIL allocation to the CFTs of the example in Figure 5.13 is one solution that obeys the ASIL tailoring rules. There is another valid solution where c3 and c4 are allocated with different ASILs. c3 and c4 are part of an ASIL decomposition. Thus, the sum of the ASILs of o5 and o6 must be greater or equal to ASIL D of o3. This allows five permutations of ASIL allocations to c3/c4: ASIL A(D)/ASIL C(D), ASIL C(D)/ASIL A(D), ASIL B(D)/ASIL B(D), QM(D)/ASIL D(D), and ASIL D(D)/QM(D). As i11 is allocated with ASIL C and there is a cascading failure of c4 leading to i11, c4 is constrained to an ASIL greater or equal to ASIL C. Hence, only two of the ASIL decomposition permutations are possible: ASIL A(D)/ASIL C(D) as shown in Figure 5.13, and QM(D)/ASIL D(D).

This shows that the ASIL tailoring rules open a constrained space of different ASIL allocations. To find allocations that obey all constraints, and thus are valid for the whole failure propagation model, the rules have to be checked for all gates and CFTs.

5.3.1.2 ASIL Allocation as Optimization Problem

As shown in the previous section, there can be more than one valid ASIL allocation for a given failure propagation model. Each CFT represents the failure propagation of a function. Consequently, each function inherits the ASIL allocated to its corresponding CFT. The ASIL of a function dictates the required safety effort for realizing the function. The goal of the safety engineer is to find an ASIL allocation to functions that reduces the overall safety effort as much as possible. The search for an ASIL allocation that obeys the ASIL tailoring rules and has minimal safety effort can be expressed as an optimization problem, i.e., an integer linear program (ILP).

The following ILP (P) is an example program derived from the failure propagation model shown in Figure 5.13. The objective (P.1) specifies the goal to minimize the sum of ASILs of all CFTs. The two constraints (P.2) and (P.3) specify the fixed ASIL values ASIL D and ASIL C of the two hazard events h1 and h2. The constraints (P.4) to (P.6) specify that each CFT's ASIL is the maximum of its failure mode ASILs (cf. Definition 5.1). The constraints from (P.2) to (P.6) represent static rules for ASIL allocation. The following constraints from (P.7) onwards apply the ASIL tailoring rules to the given failure propagation model.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{5} s_{c_j} && && \text{(P.1)}\\
\text{subject to} \quad & s_{h_1} = 4 && \triangleright \text{ Fixed ASIL constraints} && \text{(P.2)}\\
& s_{h_2} = 3 && && \text{(P.3)}\\
& s_{c_1} \geq \max\{s_{o_1}, s_{o_2}, s_{i_3}, s_{i_4}\} && \triangleright \text{ CFT constraints} && \text{(P.4)}\\
& \dots && && \text{(P.5)}\\
& s_{c_5} \geq \max\{s_{o_8}, s_{i_{11}}\} && && \text{(P.6)}\\
& s_{i_6} \geq s_{a_2} && \triangleright \text{ Prohibited ASIL decomposition} && \text{(P.7)}\\
& s_{i_7} \geq s_{a_2} && && \text{(P.8)}\\
& s_{i_3} \geq s_{a_1} && \triangleright \text{ Constrained ASIL decomposition} && \text{(P.9)}\\
& s_{i_5} + s_{i_6} \geq s_{a_1} && \triangleright \text{ AND gate constraints} && \text{(P.10)}\\
& s_{i_6} + s_{i_7} \geq s_{a_2} && && \text{(P.11)}\\
& s_{i_1} \geq s_{h_1} && \triangleright \text{ Default gate constraints} && \text{(P.12)}\\
& s_{o_1} \geq s_{i_1} && && \text{(P.13)}\\
& s_{o_3} \geq s_{o_1} && && \text{(P.14)}\\
& s_{a_1} \geq s_{o_3} && && \text{(P.15)}\\
& \dots && && \text{(P.16)}\\
& s_k \in \{0,1,2,3,4\} \quad \forall k && && \text{(P.17)}
\end{aligned}
$$

The following Figures 5.14 to 5.16 describe patterns in a failure propagation model that, if identified, lead to constraints as from (P.7) onwards. The patterns are specified in abstract syntax conforming to the CFT meta model in Figure 5.3 with two additional relations *pathTo* and *pathToFirst* defined as follows:

$$
\begin{aligned}
\text{pathTo} : G \rightarrow G &= \text{reverse failure propagation path between two gates}\\
&\ \text{with } g_1 \text{ pathTo } g_2 \Leftrightarrow g_2 \rightsquigarrow g_1\\
\text{pathToFirst} : G \rightarrow F &= \text{reverse failure propagation path to the first non-delegating}\\
&\ \text{failure mode of a type}\\
&\ \text{with } g \text{ pathToFirst } f_1 \Leftrightarrow\\
&\ g \text{ pathTo } f_1 \wedge (\neg\exists\ f_2 \text{ of same type as } f_1 : g \text{ pathTo } f_2 \text{ pathTo } f_1)
\end{aligned}
$$

For an input failure mode i_1 and an output failure mode o_2, i_1 pathToFirst o_2 means that o_2 causes i_1 and the failure propagation path between them consists either of a single CFT link or solely of failure modes connected via delegating CFT links that bridge CFT hierarchy levels between the two failure modes.

Figure 5.14 describes two default constraints between gates. Figure 5.14 a) shows the pattern of a gate g2 that is directly linked to a gate g1 that is no AND gate. This represents the default model excerpt of cascading failure modes (cf. Definition 5.2), events, and OR gates. In this case, the gate g2 inherits the ASIL of gate g1 or another linked gate with a higher ASIL. This constraint is required to enforce the ASIL separation rules. For the example failure propagation model in Figure 5.13, the pattern of Figure 5.14 a) leads to the constraint (P.12) and following. These constraints are derived from the model by following the path form the event h1 to the AND gate a1.

Figure 5.14 b) shows the pattern of a set of gates g that are directly linked to an AND gate a. This represents the basic model excerpt required for ASIL decomposition. In this case, the sum of ASILs of all gates g equals the ASIL of the AND gate a or is higher if this ASIL decomposition is influenced by other constraints (e.g., Figure 5.15). For the example in Figure 5.13, the pattern of Figure 5.14 b) leads to the constraints (P.10) and (P.11) for the two AND gates a1 and a2.

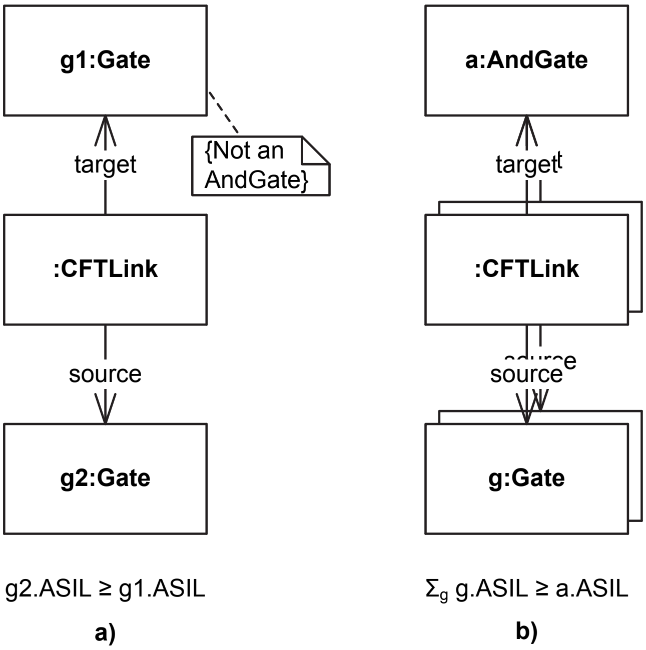
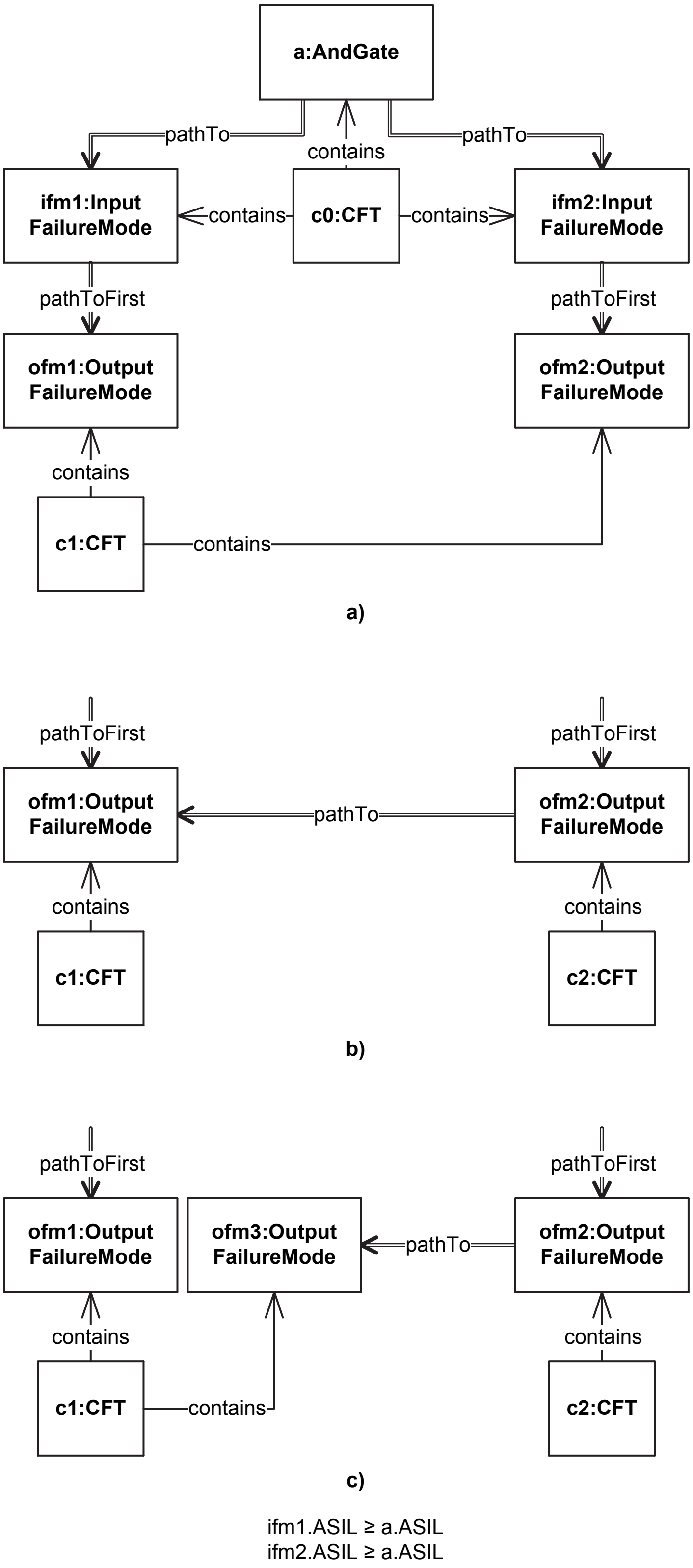


Figure 5.14: Default ASIL allocation constraints

Figure 5.15 describes three patterns that prohibit the application of ASIL decomposition (cf. Definition 5.7). All three patterns have a common upper part: A CFT c0 containing two input failure modes ifm1 and ifm2 that lie on failure propagation paths to an AND gate a, and that are caused by two output failure modes ofm1 and ofm2, respectively. The AND gate a inherits the ASIL of an output failure mode of its CFT c0 by the pattern in Figure 5.14 a) (a being mapped to g2). If ASIL decomposition concerning a is prohibited, the ASIL of input failure modes of c0 lying on failure propagation paths to a may not be reduced as specified by the pattern in Figure 5.14 b). Instead, the input failure modes ifm1 and ifm2 have to inherit the ASIL of a or any higher ASIL, similar to the pattern in Figure 5.14 a).

In addition to the common upper part, Figure 5.15 a) contains a single CFT c1 that contains both output failure modes ofm1 and ofm2. In this case, ASIL decomposition concerning



a)

b)

c)

Figure 5.15: Prohibited ASIL decomposition constraints

a is prohibited because c1 causes both input failures ifm1 and ifm2. There is only one corresponding function, and thus no redundancy.

Figure 5.15 b) extends the common part by two CFTs c1 and c2 that contain the output failure mode ofm1 and ofm2, respectively. Additionally, there is a failure propagation path from ofm1 to ofm2. In this case, ASIL decomposition concerning a is prohibited because c2 is not free of interference from c1. ofm1 is a failure that cascades to c2's ofm2. Accordingly, there is no sufficient independence between c1 and c2.

Figure 5.15 c) contains the same elements as b) extended by a second output failure mode ofm3 contained in CFT c1. In contrast to b), there is no failure propagation path from ofm1 to ofm2 but from ofm3 to ofm2. In this case, ASIL decomposition concerning a is prohibited for the same reason as in b): ofm3 is a cascading failure to c2's ofm2. For the example in Figure 5.13, the pattern of Figure 5.15 c) leads to the constraints (P.7) and (P.8). These constraints are derived from the cascading failure between the CFTs c4 and c5.

Figure 5.16 describes the pattern that constrains ASIL decomposition because of common cause failures (cf. Definition 5.6). It shows the same upper part as Figure 5.15: a CFT c0 containing two input failure modes ifm1 and ifm2 that lie on failure propagation paths to an AND gate a, and that are caused by two output failure modes ofm1 and ofm2, respectively.

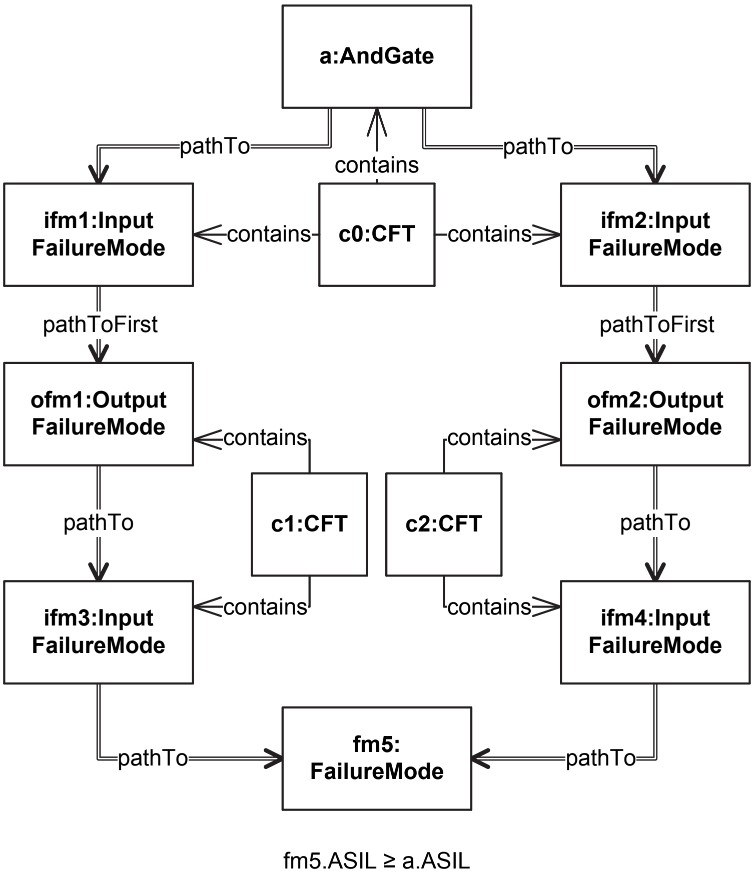


Figure 5.16: Constrained ASIL decomposition constraint

In addition, ofm1 is contained in a CFT c1 that also contains an input failure mode ifm3 lying on a failure propagation path to ofm1. Similarly, ofm2 is contained in a CFT c2 with an input failure mode ifm4. Furthermore, there is a failure mode fm5 that lies on a failure propagation path to both ifm3 and ifm4. In this case, the CFTs c1 and c2 are free from interference but ifm3 and ifm4 are common cause failures (both caused by fm5). Hence, c1 and c2 are only sufficiently independent if fm5 is allocated with the same ASIL as a or a higher ASIL. If this constraint is fulfilled, ASIL decomposition concerning a is allowed. For the example in Figure 5.13, the pattern of Figure 5.16 leads to the constraint (P.9). It is derived from the failure i3 that is a common cause for the CFTs c3 and c4 concerning a1.

Algorithm 5.4 sketches how ILP constraints are derived from a failure propagation model based on the patterns from Figures 5.14 to 5.16. Combined with the ASIL minimization objective (P.1) the ILP can be passed to an ILP solver. Afterward, the resulting solution's variables contain the ASILs to be allocated to gates and CFTs of the failure propagation model.

5.3.2 Allocating ASILs to Functions and Functional Safety Requirements

ASILs are assigned to hazards to categorize their safety-criticality. Functional safety requirements are specified to prevent or mitigate hazards and inherit the addressed hazard's ASIL. Functions are used to decompose the required functionality and group the functional (safety) requirements. During decomposition and refinement of functions and requirements, the corresponding ASILs can be lowered for sub-functions and -requirements by applying ASIL tailoring measures. The set of applied, valid ASIL tailoring measures dictates the allocation of ASILs to functions and functional safety requirements.

Figure 5.17 shows all model elements that can be assigned with an ASIL and their ASIL allocation constraints. Each Hazard's safety-criticality is specified by an ASIL. The failure propagation model consists of CFTs that represent the failure propagation of the function hierarchy. The ASIL of a hazard propagates along the paths of CFT Gates that are involved in the occurrence of the hazard and is reduced where valid ASIL tailoring is applied. This allocation of ASILs to gates and CFTs follows the rules and constraints described in Section 5.3.1. As each CFT represents the failure propagation of a Function, each function inherits the ASIL of its corresponding CFT. Safety requirements on functions are specified using MSDs (cf. Chapter 4). The failure propagation of each function is derived from its MSDs as described in Section 5.2.3. A failure in the fulfillment of an MSD requirement leads to an output failure mode of the corresponding function's CFT. Therefore, each MSD inherits the maximum ASIL of all output failure modes that were derived from it.

Figure 5.18 shows an example result of ASIL allocation to MSD requirements. ASILs are annotated on model elements using the stereotype Safety Classified Element from the safety profile (cf. Figure 3.5 on page 45). The diagram in the top shows an excerpt of a failure propagation model after a valid ASIL allocation was calculated by an ILP solver (cf. Section 5.3.1.2). There are two Hazard CFTs with events representing the ASIL D hazard Hard Braking Omission and the ASIL B hazard Emcy Brake Warning Omssion from Figure 3.4 on page 44. There is a failure propagation path from the sub-CFT Make Decision CFT to each Hazard CFT (illustrated by a dashed CFT link). The ASIL of both hazards propagates down to the two output failure modes of the CFT and on to an AND gate where

Algorithm 5.4 Collecting ASIL Allocation Constraints

```
for all Events $e$ typed by a Hazard do    ▷ Fixed ASIL constraints from Hazards
    Add constraint: $e$.ASIL = <ASIL value of Hazard>
end for
for all FailureModes $f$ not contained in a HazardCFT do    ▷ CFT constraints
    $c$ := CFT containing $f$
    Add constraint: $c$.ASIL $\geq f$.ASIL
end for
for all Gates $g_0$ do    ▷ Default ASIL constraints
    if $g_0$ is an AndGate then
        $A$ := all Gates directly succeeding $g_0$
        Add constraint: $\sum_{g \in A} g$.ASIL $\geq g_0$.ASIL
    else
        for all Gates $g_1$ directly succeeding $g_0$ do
            Add constraint: $g_1$.ASIL $\geq g_0$.ASIL
        end for
    end if
end for
for all AndGates $a$ do    ▷ Prohibited and constrained ASIL decomposition constraints
    $c$ := CFT containing $a$
    $I$ := All InputFailureModes contained in $c$ and succeeding $a$
    for all pairs of InputFailureModes $(i_1, i_2)$ in $I \times I$ with $i_1 \neq i_2$ do
        $o_1$ := First non-delegating OutputFailureMode succeeding $i_1$
        $o_2$ := First non-delegating OutputFailureMode succeeding $i_2$
        if $o_1$ or $o_2$ does not exist then
            continue for loop    ▷ $i_1$ or $i_2$ is not caused by a function of the SuD
        end if
        $c_1$ := CFT containing $o_1$
        $c_2$ := CFT containing $o_2$
        if ($c_1 = c_2$) or ($c_1$ contains an OutputFailureMode succeeding $o_2$) or ($c_2$ contains an OutputFailureMode succeeding $o_1$) then    ▷ Prohibited ASIL decomposition
            Add constraint: $i_1$.ASIL $\geq a$.ASIL
            Add constraint: $i_2$.ASIL $\geq a$.ASIL
        else
            $J_1$ := All InputFailureModes contained in $c_1$ succeeding $o_1$
            $J_2$ := All InputFailureModes contained in $c_2$ succeeding $o_2$
            for all pairs of InputFailureModes $(j_1, j_2)$ in $J_1 \times J_2$ with $j_1 \neq j_2$ do
                if $j_1$ and $j_2$ are both succeeded by the same FailureMode $f$ then    ▷ Constrained ASIL decomposition
                    Add constraint: $f$.ASIL $\geq a$.ASIL
                end if
            end for
        end if
    end for
end for
```

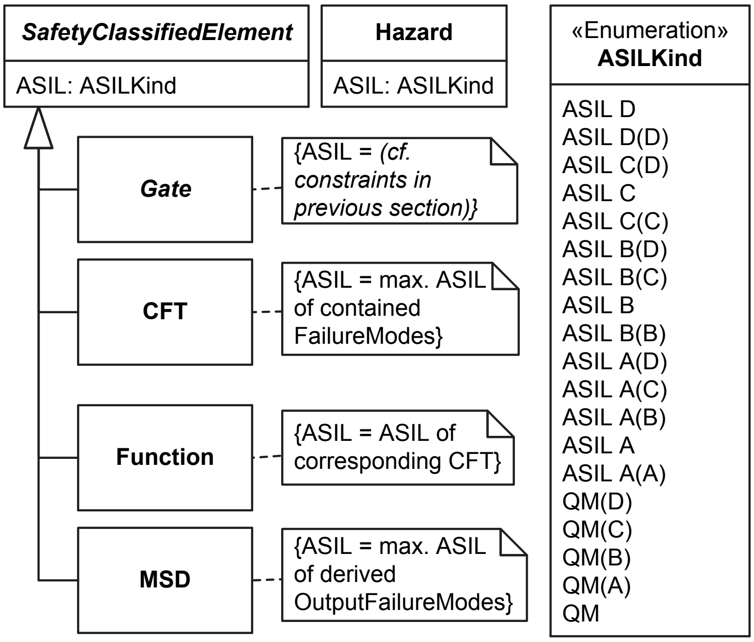


Figure 5.17: Meta model elements considered by ASIL allocation and their ASIL relations

valid ASIL decomposition was applied. Hence, the two input failure mode's ASILs were reduced to ASIL B(D). The CFT inherits ASIL D as maximum ASIL of its failure modes. Consequently the function Make Decision whose failure propagation is represented by the CFT inherits ASIL D as well.

The failure propagation model was derived from the MSDs shown in the bottom of the figure. The two Brake Requirements require the sending of the information activateEmergencyBraking and are traced to the corresponding omission output failure mode of Make Decision CFT. The two Warn Requirements require the sending of the information sendEmcyBrakeWarning and are traced to the other omission output failure mode. The two sketched Pre Requirements require the sending of the information needed by the other requirements and are traced to corresponding omission output failure modes that lead to the input failure modes of the CFT shown in the top diagram. The CFTs containing those output failure modes are not shown in the figure. Each MSD inherits the ASIL of the output failure mode that was derived from it.

5.3.3 Application to other Safety-Critical Domains

Different domains have different safety standards (cf. Section 2.3) but they all have SILs to classify the safety-criticality of hazards. The SIL of a hazard classifies its safety-criticality by the risk of its occurrence and is determined during hazard analysis and risk assessment (cf. Section 2.2). The risk of a hazard is defined as the combination of its probability of occurrence and the severity if it occurs (Risk = Probability × Severity). A hazard occurs if one or more failures of a system's functions occur. Hence, the probability of hazard occurrence is dependent on the probability of function failures. If a hazard h is caused by a single failure f, the hazard probability P_h is equal to the failure probability P_f, and the

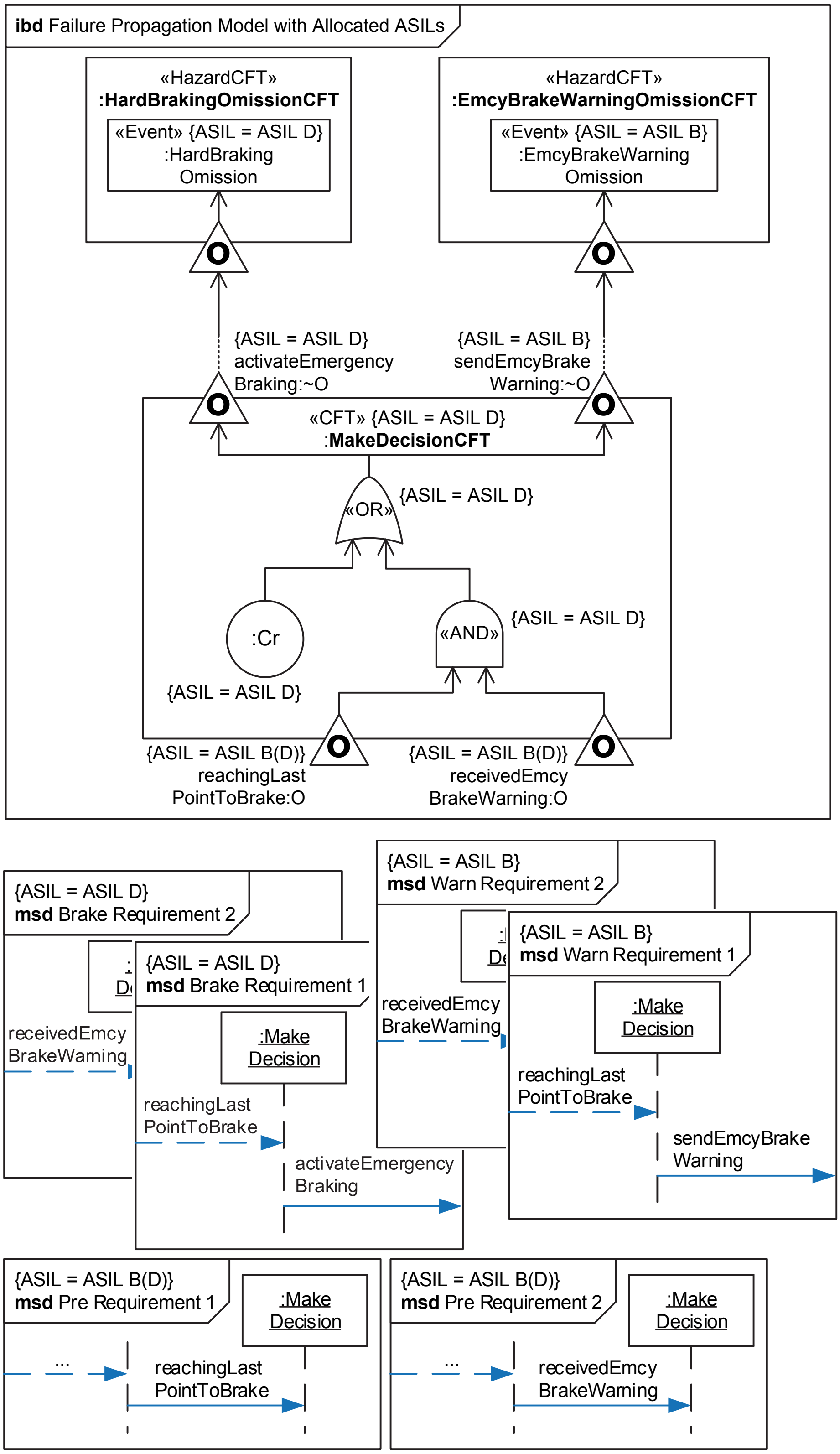


Figure 5.18: Example failure propagation model and ASIL allocation to MSDs

severity of that failure f is equal to the severity of the caused hazard h. Thus, the function causing that failure has the same risk as h and inherits its SIL.

In addition to that simple rule, in the automotive domain, ISO 26262 explicitly defines rules for SIL tailoring (cf. Section 2.5). The argument behind the ASIL tailoring rule Separation is that if a function f_1 cannot cause a more critical function f_2 to fail, it does not contribute to f_2's probability of failure, i.e., risk. Thus, f_1 has a lower risk and may be assigned with a lower ASIL than f_2 as specified in Definition 5.4.

The argument behind the ASIL tailoring rule Decomposition is as follows: If a hazard h only occurs if two failures f_1 and f_2 occur simultaneously, the hazard probability P_h is equal to the probability of both failures occurring simultaneously $P_{f_1 f_2}$. If the two failures are caused by independent functions (such that $P_{f_1 f_2} = P_{f_1} \times P_{f_2}$), the two functions share the risk of h. In this case, the risk of h (its ASIL) may be arbitrarily distributed among the two functions as specified in Definition 5.7 ($P_h = P_{f_1} \times P_{f_2} \Rightarrow s_{f_1} + s_{f_2} = s_h$).

Not all safety standards provide explicit SIL tailoring rules. We claim that the reasoning behind ASIL tailoring can be adopted for other domains. Separation is applicable to any domain: If a function cannot cause a hazard, there is no reason for it to be developed to the degree of rigor required for that hazard's risk. Decomposition rules need to be individually defined and their validity argued if the domain standard provides no explicit rules.

In aerospace, ARP4754A specifies the concept of DAL decomposition [SAE10]. Similarly to the ASIL, the DAL (Development Assurance Level) represents the required degree of rigor during development of a system. It is also categorized into five values. The values range from DAL A (highest) to DAL E (lowest). The DAL decomposition rules can be summarized as follows [RPS+17]:

- Option 1: One of the functions that are part of a decomposition concerning the same failure f must keep the DAL of f, all other functions may be assigned with a DAL up to two levels lower than f's DAL.
- Option 2: Two of the functions that are part of a decomposition concerning the same failure f are assigned with a DAL one level lower than f's DAL, all other functions may be assigned with a DAL up to two levels lower than f's DAL.

Definition 5.8 (DAL Decomposition)

The sum of DALs of all CFTs that are part of a decomposition concerning an output failure mode o_3 must be greater or equal to the number of involved CFTs D times the DAL that is two levels lower than o_3's DAL s_{o_3} plus 2. The DAL of each of the involved CFTs D must be greater or equal to the DAL that is two levels lower than o_3's DAL s_{o_3}.

$$\begin{aligned}\sum_{c \in D} s_c &\geq |D|(s_{o_3} - 2) + 2 \quad && \forall\, c_3 \in C, o_3 \in O_{c_3}, D = \{c \in C \backslash \{c_3\} \mid o_3 \in \text{Decomp}_{c,c_3}\} \\ s_c &\geq s_{o_3} - 2 && \forall\, c \in D\end{aligned}$$

To find valid DAL allocations to CFTs, safety classified elements must be allocated with DAL A to DAL E instead of ASIL D to QM. The ILP constraints remain the same as for ASIL allocation, except the default constraint for AND gates (cf. Figure 5.14 b)). It has to be exchanged with the following two constraints:

$$\begin{aligned}\sum_{g \in D} g.\mathrm{DAL} &\geq |D|(a.\mathrm{DAL} - 2) + 2 && \text{with } D = \{g \in G \mid a \text{ pathToFirst } g\} \\ g.\mathrm{DAL} &\geq a.\mathrm{DAL} - 2 && \forall\, g \in D\end{aligned}$$

From the definitions of ASIL decomposition and DAL decomposition one can see that there is a lower risk tolerance in the aerospace domain than in the automotive domain. In ASIL decomposition, the decomposed function SILs have to add up to the original SIL. Even in the extreme case of ASIL D decomposition, one function may be considered non-safety-critical (QM(D)). In DAL decomposition, the decomposed function SILs have to add up to more than the original SIL, and no function may have a SIL lower than two levels below the original SIL.

The automated ASIL allocation presented in this chapter can be applied to both domain standards. Hence, we claim that the (A)SIL tailoring process as a whole is applicable to embedded systems domains and the development of safety-critical cyber-physical systems.

5.4 Assumptions & Limitations

Assumptions

For the derivation of the failure propagation model from requirements, we assume that all requirement MSDs are instances of the requirement patterns of Section 4.4.2.

Limitations

We use CFTs for the failure propagation model that are based on traditional fault trees and inherit their limitations (cf. [Int03]): Traditional fault trees are not considered applicable for failures that are dependent on (1) the current state of the system (e.g., initialization, running, degraded, diagnosis), or (2) the order or timing of events. We mitigate these shortcomings by our use of MSDs: (1) ASIL tailoring requires the analysis of cascading and common cause failures (cf. Section 5.3.1.1) that exist in any state of the system. MSD specifications can be grouped by use cases that can represent different system states. Hence, we can generate a failure propagation model for each system state and compare their ASIL allocation results to find a valid solution for all system states. (2) MSDs support specifying and analyzing the order and timing of events (cf. Section 2.7). As we need to consider any system state for ASIL tailoring, we generate omission and commission failure modes if there is at least one order of events leading to such a failure. Furthermore, we use failure types for early and late timing failures that can be derived from timed MSDs.

The current implementation for generating CFTs does not consider any MSD conditions other than hot false conditions, no negative fragments, and no message parameters. Hence, failure modes typed by timing or value failure types are not generated.

The ILP we use for ASIL allocation calculates only one optimal allocation result although there may be more than one optimal result that the safety manager could choose from (cf. conclusion in Section 5.3.1.1). However, one could use additional constraints to specify fixed ASILs for certain functions that might be required due to reuse of legacy components with a specific ASIL or project cost constraints. Moreover, instead of an ILP solver, we could use meta heuristics that are more scalable for bigger failure propagation models. However, the implementation that we use for deriving allocation constraints is independent from any concrete solver, such that we could use different ILP solver implementations and also meta heuristics.

5.5 Related Work

We divide the work related to our method of safety analysis and ASIL allocation into two categories: work related to deriving failure propagation models from requirements (Section 5.5.1), and to allocating ASILs to functions (Section 5.5.2).

5.5.1 Generating Failure Propagation Models

HiP-HOPS is a failure propagation analysis solution that generates fault trees from component-based architectures [PM99]. The fault trees are generated from the structure of the components and their connections. Each component has to be annotated with its internal failure specification. HiP-HOPS does not distinguish different failure types.

McKelvin et al. present an approach that derives fault trees from software deployments to ECUs [MEP$^+$05]. The purpose is to check for valid redundancy of ECUs. The software is specified in their own language called Fault Tolerant Data Flow. A fault tree is derived from the SW/HW deployment model. The approach supports only the omission failure type.

Lauer et al. derive fault trees from software architectures and their deployment to ECUs [LGP11]. Their objective is to ease the comparison of different architecture alternatives. Behavior is specified by UML activities that are allocated to software components. Additionally, they define a UML profile to specify failures of components. The fault trees are derived from component failures and control flow of activities. The generated fault trees do not distinguish different failure types.

Mader et al. generate fault trees and FMEA tables from an EAST-ADL model [MAL$^+$11]. They aim at safety analysis to show how safety goals can be violated by failures of the modeled embedded system. For that, an error model is specified for each system component. The fault trees and FMEA tables are derived from that error models. The error model does not differentiate between different types of failures.

Priesterjahn et al. derive failure propagation models from software behavior specifications with the objective to analyze the timing of failure propagations [PHS13]. The behavior is specified by timed automata. The failure propagation model is specified as timed petri net that is derived from the state transitions and their timing conditions. Consequently, the approach focuses on timing failures.

All mentioned approaches focus on software and hardware architectures, whereas we target requirements specifications in this thesis. Furthermore, all approaches except [PHS13] require the manual specification of failures and merely connect those local failure conditions to global failure propagation models (i.e., fault trees) for the overall system. In this thesis, failures and failure propagations are completely generated from requirements specified as MSDs.

5.5.2 ASIL Allocation

Safar developed an approach for ASIL calculation on fault trees [Saf17]. The ASIL values are allocated to the failure modes of the fault trees. The fault tree structure and the ASIL decomposition rules are specified in a textual syntax for Satisfiability Modulo Theories. The ASIL allocation result is not transfered to a system or requirements model.

Mader et al. present an ASIL allocation approach for EAST-ADL models [MAL+12]. The approach uses traditional fault trees, one for each hazard of the system. Allocation constraints are specified over minimal cut sets of the fault trees. One optimal ASIL allocation to system components is calculated by an ILP solver that respects a safety manager's ASIL preferences for certain components. The approach does not use a complete failure propagation model (all failure modes of all fault trees) for ASIL allocation but only allocates ASILs to components. Thus, the safety manager cannot see ASIL propagation paths that would support identifying and deciding on system changes to gain a better (e.g., cheaper) ASIL allocation result.

Dhouibi et al. allocate ASILs to EAST-ADL elements based on minimal cut sets of fault trees [DPS+14]. They use a system of linear equations to calculate all possible ASIL allocations to the elements (not to failure modes). The identification of an optimal allocation is left to the safety manager.

Papadopoulos et al. also developed an approach for ASIL allocation to EAST-ADL models [PLB+10]. It uses HiP-HOPS [PM99] and presents all valid and optimal permutations of ASIL allocations (without safety manager preferences). In contrast to the approaches of Mader et al. and Dhouibi et al., ASILs are not allocated to components but to their failure modes. The approach was improved by Azevedo et al. by using a meta heuristic to reduce the result permutations [APW+13]. Their allocation result is not necessarily optimal but the computation is more scalable than solving an ILP. In 2014, they added the use of a cost heuristic that specifies a cost value for each ASIL [APW+14]. This allows to express the relative savings of reducing from one ASIL to another.

The existing approaches for ASIL tailoring are applied in the phases of system architectural design or software design. They calculate an ASIL allocation to subsystems of a technical architecture with known failure propagation. They assume that ASIL tailoring measures (separation and decomposition) have already been applied – both on requirements and architectural level – only the permutation of ASILs is left to decide. Furthermore, there is no ASIL allocation to requirements although ASIL decomposition is defined on requirements (cf. Definition 2.26). Moreover, the approaches do not consider independence (cf. Definition 2.25) nor sufficient independence (cf. Definition 5.6). Instead, they leave the analysis of dependent failures to the safety manager.

The ASIL allocation approach in this thesis, allocates ASILs to all failure modes of the failure propagation model and to corresponding requirements. This supports the safety manager in deciding on safety requirements changes for better ASIL allocation results. Furthermore, the rules for sufficient independence are considered by additional allocation constraints.

5.6 Conclusion

This chapter describes the details about automated safety analysis and ASIL allocation on functional (safety) requirements as part of the ASIL tailoring process introduced in Chapter 3. Section 5.2 explains the used failure propagation meta model, the manual linkage of hazards and top-level failures, and the automatic generation of CFTs. Section 5.3 specifies the ASIL tailoring rules on CFTs for separation and decomposition, and how ASILs are allocated to functions and functional (safety) requirements by solving an ILP. Moreover, Section 5.3.3 shows the applicability to more than the automotive domain.

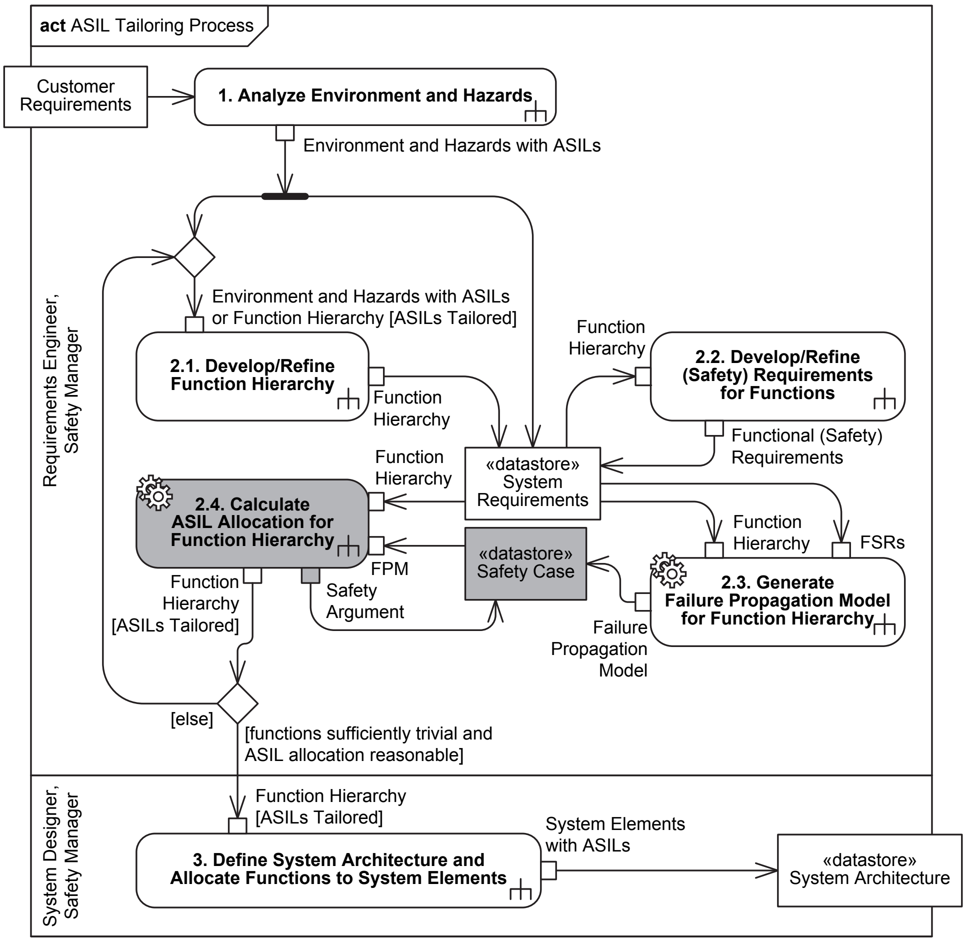
The automatic derivation of a failure propagation model from functions and corresponding (safety) requirements supports early safety analysis by saving time and reducing manually introduced errors. This is a step towards solving the safety requirements engineering dilemma (cf. Challenge C2 in Section 1.2). Moreover, the automatic allocation of ASILs to functions and requirements that inherently checks for valid application of ASIL tailoring reduces manual effort and mistakes, and supports early planning of ASIL tailoring measures and safety effort.

The following Chapter 6 adds an automatic documentation of safety arguments that link hazards, safety analysis results, functions, and corresponding requirements to assure valid ASIL tailoring.

6

DOCUMENTING ASIL TAILORING ARGUMENTS

In Step 2.4 of the ASIL tailoring process (cf. Figure 6.1), together with the ASIL allocation, a safety argument is developed that provides assurance that all hazards are addressed by functional safety requirements and the ASIL tailoring is valid.



act ASIL Tailoring Process
Customer Requirements
1. Analyze Environment and Hazards
Environment and Hazards with ASILs
Requirements Engineer, Safety Manager
Environment and Hazards with ASILs or Function Hierarchy [ASILs Tailored]
2.1. Develop/Refine Function Hierarchy
Function Hierarchy
Function Hierarchy
2.2. Develop/Refine (Safety) Requirements for Functions
Functional (Safety) Requirements
«datastore» System Requirements
Function Hierarchy
2.4. Calculate ASIL Allocation for Function Hierarchy
FPM
«datastore» Safety Case
Function Hierarchy
FSRs
2.3. Generate Failure Propagation Model for Function Hierarchy
Function Hierarchy [ASILs Tailored]
Safety Argument
Failure Propagation Model
[else]
[functions sufficiently trivial and ASIL allocation reasonable]
System Designer, Safety Manager
Function Hierarchy [ASILs Tailored]
System Elements with ASILs
3. Define System Architecture and Allocate Functions to System Elements
«datastore» System Architecture

Figure 6.1: ASIL tailoring process with highlighted contents of Chapter 6

In this chapter, the details how this safety argument is derived from the function hierarchy, functional safety requirements, and failure propagation model is described.

This chapter is structured as follows. First, Section 6.1 summarizes the scientific contributions of automated documentation of ASIL tailoring arguments. Section 6.2 explains the used safety argument language, and how the arguments are derived from an ASIL allocated failure propagation model and related requirements. Section 6.3 lists assumptions and limitations of the generation of safety arguments. In Section 6.4, related work is discussed. The final Section 6.5 concludes this chapter.

6.1 Contributions

The contributions of this chapter can be summarized as follows:

- A method and tool-support for safety case construction embedded in an ASIL tailoring process to foster model-based safety assessment.
- Support for traceability from safety arguments to related work products (i.e., hazards, safety requirements, functions, and safety evidence) in order to improve safety case impact analysis and maintenance.
- Automated safety case construction (including traceability) to reduce manual effort and errors.
- Automated derivation of validity arguments for applied ASIL tailorings.
- Support for building the safety case from the beginning of development (i.e., functional safety requirements analysis, cf. Challenge C3 in Section 1.2).

6.2 Safety Case Construction

There is no guarantee for 100% safety. Hence, the risk of hazards is determined, and measures are taken to reduce the risk as low as reasonably possible. Safety standards like ISO 26262 define SILs that categorize the risk of hazards into different levels. For each SIL they list state-of-practice measures that should be taken to prevent a hazard's occurrence or mitigate its consequences.

As safety cannot be proven, confidence that the risk has been reduced as low as reasonably possible, has to be provided by other means. The so-called *safety case* shall provide this confidence by compiling safety-relevant work products and linking them by a safety argument (cf. Section 2.2). For each hazard, a safety argument shall link derived safety requirements with evidence for their fulfillment and contribution to hazard mitigation. This also includes argumentation for the validity of applied ASIL tailorings. Thus, the safety case provides an overview of system safety, and is the central entry point for safety assessments.

In practice, safety arguments are usually written as free text documents [Kel98]. Free text arguments can be ambiguous and usually require lots of cross-references. Thus, these documents are difficult to analyze for completeness and validity of arguments. Especially in early development phases, requirements change regularly, and it is time-consuming and error-prone to keep the free text arguments in sync and consistent with linked work products.

Hence, safety arguments are specified late in the development process where changes are few [Kel98]. If mistakes in the arguments and missing safety measures are discovered late in the development, this causes expensive process iterations to integrate additional measures and update the safety case. The industry already identified the need to start building the safety case from the beginning of the development process, but still misses supporting methods and tools [ZRH16].

To overcome the problems of safety argument ambiguity and structural complexity, the use of model-based languages is proposed by research [Kel98; CJL^{+}08] and ISO 26262 [Int12b]. Therefore, in this thesis, we chose the Goal Structuring Notation (GSN, cf. Section 2.8) to model safety arguments, as it is suggested in ISO 26262 [Int12b]. As part of the ASIL tailoring process, we automatically generate a GSN safety argument that can be regenerated whenever an ASIL tailoring is applied on functional requirements, to support the construction and maintenance of a safety case from the beginning of development.

This approach fosters an unambiguous overview of the safety-relevant work products and navigation through the safety case by a graphical and model-based notation. The automatic generation avoids manually introduced errors and saves time because it can be repeated whenever changes occur. In addition, it simplifies impact analyses by automatically maintaining the traceability to other safety-related work products.

The following Section 6.2.1 describes how GSN safety arguments are specified in this thesis. The details of safety argument generation are elaborated in Section 6.2.2.

6.2.1 Safety Arguments in Goal Structuring Notation Profile

The safety case links safety requirements with safety evidence by safety arguments. In ISO 26262, safety requirements exist at different levels, starting from safety goals that are derived from hazards, over functional safety requirements that are refined into technical safety requirements down to hardware and software safety requirements (cf. Section 2.2). Safety evidence are safety analysis results, like FMEA tables and fault trees, that show absence of failures or independence of subsystems. On technical, software, and hardware level, also test results that verify safety requirement fulfillment, can be safety evidence. Safety arguments specify why given safety evidence contributes to the fulfillment of safety requirements (i.e., mitigation of hazards) and why applied ASIL tailoring measures are valid.

On functional level, the safety case contains the CFT failure propagation model as safety evidence and GSN safety arguments that link the CFTs to functional safety requirements in form of MSDs (cf. Figure 3.2 on page 41).

In this thesis, we specify GSN safety arguments as UML models using a profile that extends the SysML. This way, all work products of the ASIL tailoring process are based on a single modeling language (the UML), which makes it easier to learn and understand their use. Furthermore, this enables the use of a single modeling environment. In addition, this supports the use of a common traceability concept, which is essential for safety case navigation.

Figure 6.2 shows the elements of our GSN profile. All GSN nodes are represented by UML classes with specialized SysML Requirement stereotypes. The nodes are linked by specialized UML Trace stereotypes. Figure 6.3 shows the application of the profile for an example safety argument. It specifies how an ASIL D hazard H1 is mitigated by a Safety

Goal 1 and refining functional safety requirements (FSRs) with valid application of ASIL Decomposition and ASIL Separation.

As explained in Section 2.8, the main nodes of GSN models are Goals, Strategies, and Solutions. Goals represent safety requirements or subparts of a safety argument that are claimed to hold. In Figure 6.3, Safety Goal 1 and the several FSRs are examples of safety requirements specified as goals. The two Tailoring Goals are examples of argument subparts. They specify the goal of the two ASIL tailorings.

Strategies represent implications that form the line of argument. In adherence to the concrete syntax of GSN, we denote strategies as parallelograms. In Figure 6.3, two applied ASIL tailorings are specified by the strategies ASIL Decomposition and ASIL Separation.

Solutions represent safety evidence. To adhere to the concrete syntax of GSN, we denote solutions as circles. In Figure 6.3, the solution FEBEASCFT represents a failure propagation model that provides evidence for the connected claimed goals.

Goals and Strategies refer to subgoals, strategies, and solutions (evidence) that support/refine their argument by SupportedBy traces. These links specify the line of argument which is usually visualized in a tree layout from top to bottom. To adhere to the concrete syntax of GSN, we denote this link type by a rounded arrow with a solid head and annotated with the SupportedBy stereotype.

Goals and Strategies can be marked as undeveloped by the respective stereotype's attribute isUndeveloped. In original GSN syntax, this is denoted by a diamond below the element. We do not adopt this syntax to avoid confusion with UML aggregations. Undeveloped elements need further refinement by supporting goals and evidence. In Figure 6.3, the goals FSR1.1, FSR2.1, FSR1.2, FSR2.2, and FSR1.3 are marked as undeveloped because they are not linked to further supporting goals (e.g., refining functional or technical safety requirements) or solutions.

Additionally, Goals and Strategies can be elaborated by Contexts, Justifications, and Assumptions. Akin to the concrete syntax of GSN, we denote these as rectangles with rounded corners. These elements are linked by InContextOf traces. We denote this link type by a rounded arrow with a narrow, hollow head and annotated with the InContextOf stereotype, to adhere to the concrete syntax of GSN. In Figure 6.3, Safety Goal 1 is linked to the context Hazard H1 to specify which hazard it addresses. The two strategies ASIL Decomposition and ASIL Separation are linked to justifications that name the corresponding clauses of ISO 26262 that define and allow the two ASIL tailoring measures.

The example safety argument in Figure 6.3 is derived from the safety relevant work products shown in Figure 6.4. To support traceability from a GSN safety argument to the corresponding work products, contexts, goals, and solutions are linked to hazards, functions, MSD requirements, and CFTs via UML traces. Table 6.1 lists the traces for the example safety argument in Figure 6.3. The context Hazard H1 is traced to the hazard H1 referred to by the event in the Hazard CFT HCFT. To be able to show the ASIL of traced work products, elements of the GSN safety argument are also applied with the SafetyClassifiedElement stereotype of the Safety Profile (cf. Figure 3.5 on page 45). Hence, the context Hazard H1 inherits ASIL D from the Hazard CFT's event. The Safety Goal 1 is traced to the omission failure of the information brake shall be prevented or mitigated because it directly leads to the occurrence of the hazard H1. The goals FSR1 and FSR2 are traced to the MSDs with the same name. These MSDs specify requirements on the top-level function FEBEAS whose CFT is directly connected to the Hazard CFT. Similarly, the five supporting goals FSR1.1 to

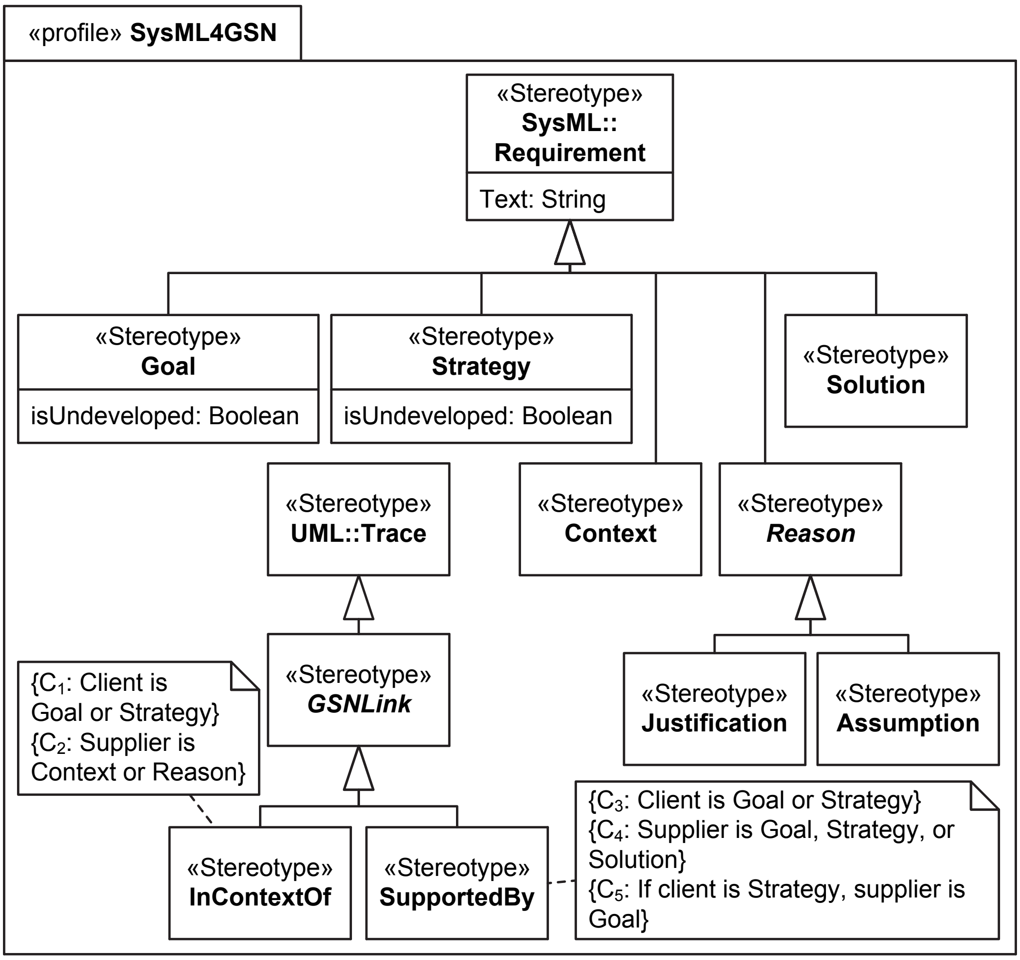


Figure 6.2: UML Profile for the Goal Structuring Notation with constraints

FSR1.3 are traced to MSDs that specify requirements on the decomposed functions whose CFTs lie on failure propagation paths to the omission failure of brake.

From the CFT model one can see that ASCFT and CWOVCFT are part of an ASIL decomposition concerning the brake failure. The corresponding MSDs FSR1.1 and FSR2.1 specify redundant requirements with lowered ASIL. Hence, the Tailoring Goal 1 is traced to those MSDs to argue the application of ASIL decomposition by the supporting strategy ASIL Decomposition and the following goals on absence of cascading and common cause failures FSR1.4 to FSR1.6. The solution FEBEASCFT traces to the CFT with the same name that shows the absence of those failures between the functions AS and CWOV.

The CFT LCFT has no failure modes that lie on a failure propagation path to the omission of brake. Hence, its corresponding function L is separate from the other functions on the same hierarchy level. This application of ASIL separation is argued by the Tailoring Goal 2 that is traced to L and its supporting ASIL Separation strategy and goal FSR1.7. The goal specifies the required absence of cascading failures to the other functions and is supported by the solution FEBEASCFT that references the CFT with that name as evidence.

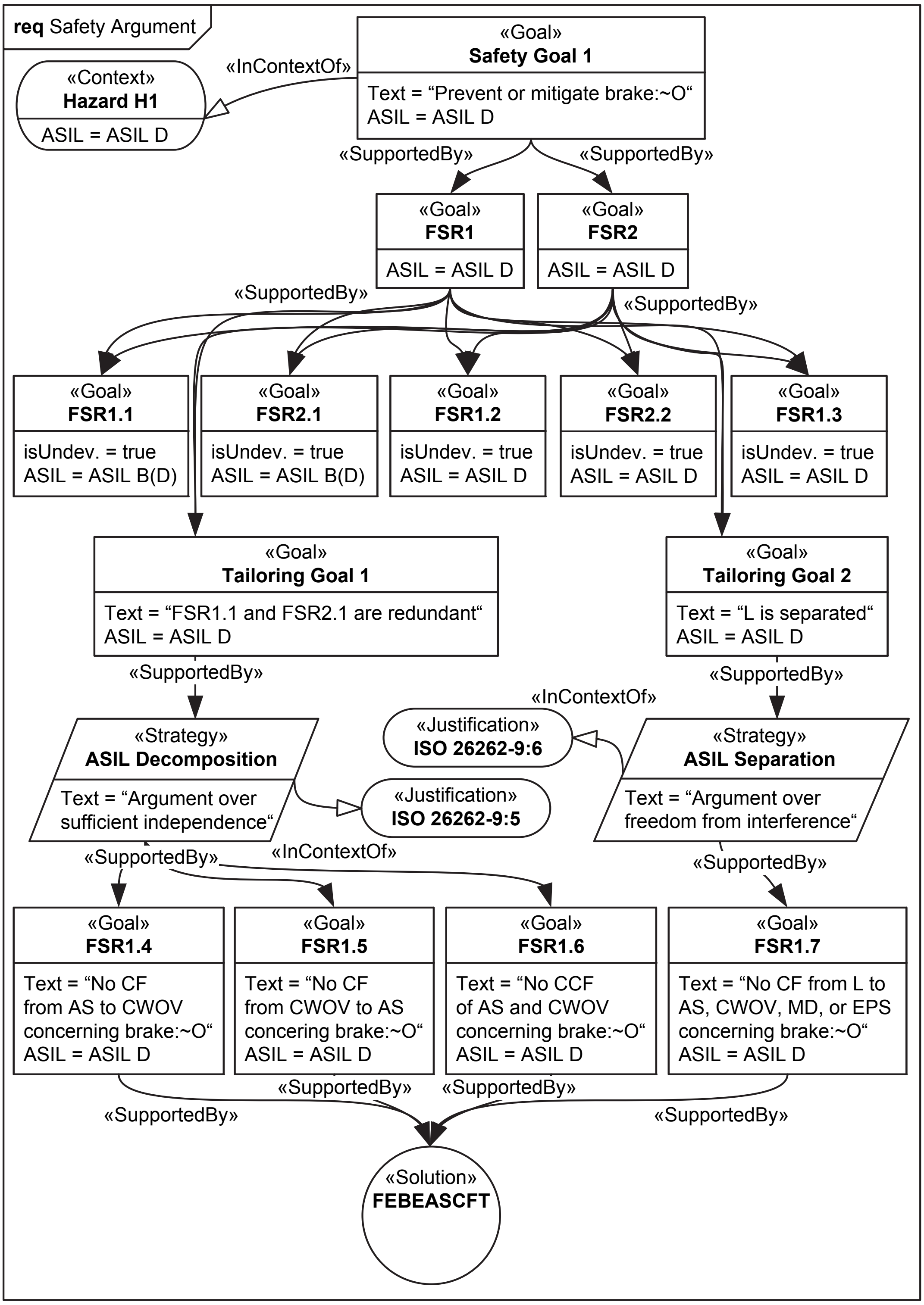


Figure 6.3: Example GSN safety argument for hazard H1

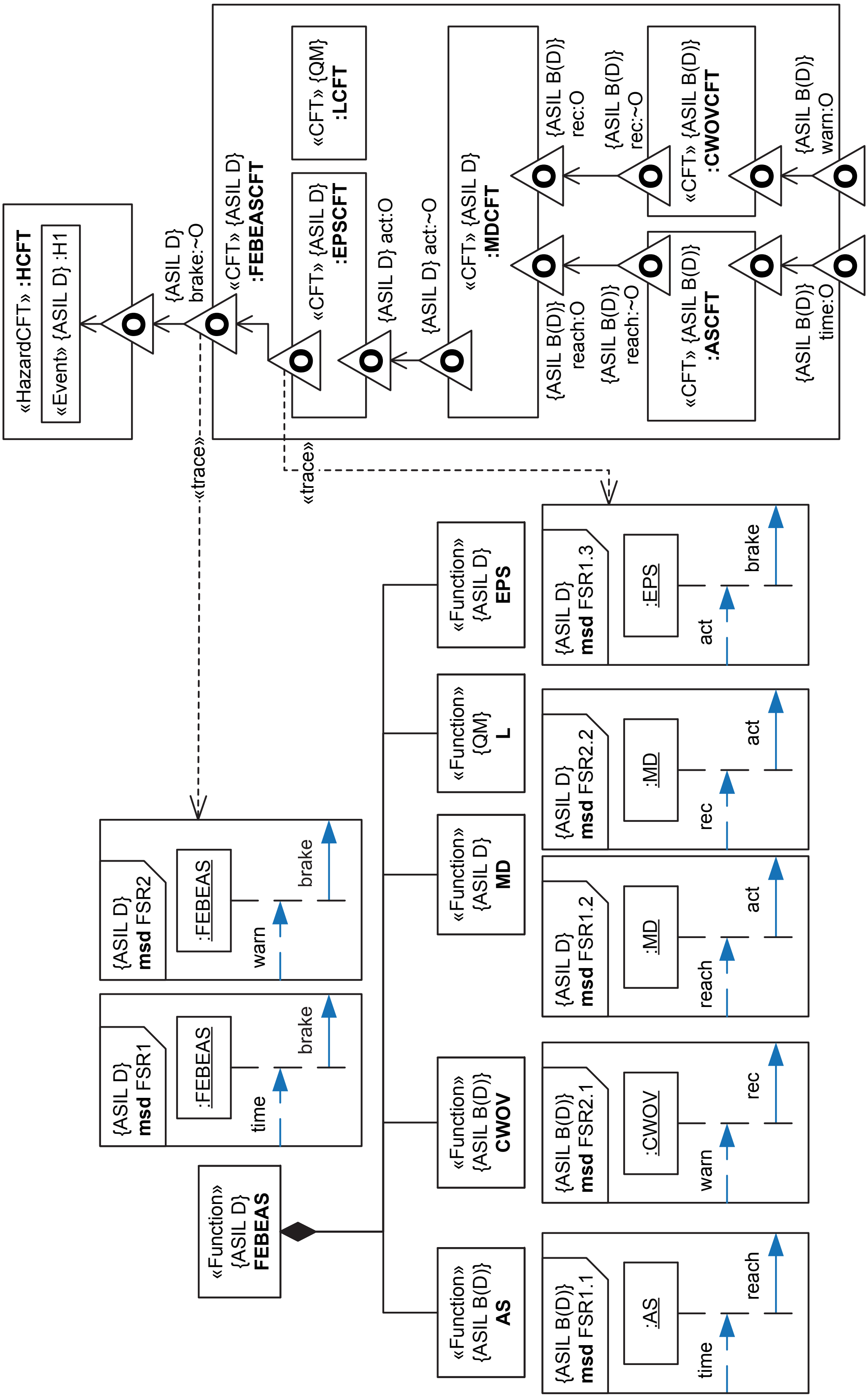


Figure 6.4: Example function hierarchy, MSDs, and CFT

Table 6.1: Traceability links of GSN safety argument

GSN Source Element	Target Element	Target Work Product
«Context» Hazard H1	«Hazard» H1	Environment model
«Goal» Safety Goal 1	«FailureMode» brake:∼O	CFT model
«Goal» FSR1	MSD FSR1	MSD specification
«Goal» FSR2	MSD FSR2	MSD specification
«Goal» FSR1.1	MSD FSR1.1	MSD specification
«Goal» FSR2.1	MSD FSR2.1	MSD specification
«Goal» FSR1.2	MSD FSR1.2	MSD specification
«Goal» FSR2.2	MSD FSR2.2	MSD specification
«Goal» FSR1.3	MSD FSR1.3	MSD specification
«Goal» Tailoring Goal 1	MSDs FSR1.1 and FSR2.1	MSD specification
«Goal» Tailoring Goal 2	«Function» L	Function hierarchy
«Solution» FEBEASCFT	«CFT» FEBEASCFT	CFT model

6.2.2 Generating Safety Arguments

Safety arguments for valid ASIL tailoring on functional safety requirements inherently have a strong relation to the work products of the ASIL tailoring process presented in previous chapters of this thesis. The example safety argument and its traceability table shown in the previous section demonstrate this claim. Thus, we automatically derive safety arguments from the failure propagation model, the function hierarchy, and the functional safety requirements to avoid manually introduced errors, save time, and maintain the safety case in sync with functional safety requirements and applied ASIL tailoring measures.

The root node of a safety argument is always a hazard. The safety argument has to show how the hazard is prevented or mitigated and why the taken measures are valid. Hence, for each identified hazard, a safety argument has to be build up, that links the hazard with derived safety requirements and their refinement. We call this the default requirement hierarchy. If a requirement in that hierarchy is assigned with a lower ASIL than the hazard, ASIL tailoring was applied. The validity of such a tailoring has to be argued within the safety argument and underpinned by safety evidence (cf. ASIL tailoring rules in Section 5.3.1.1).

Algorithm 6.1 specifies the automatic derivation of GSN safety arguments from CFT failure propagation models. It is based on the bachelor's thesis of Trentinaglia [Tre18]. The failure propagation model contains a Hazard CFT for each hazard. Hence, a safety argument is generated from each Hazard CFT. First, the root nodes of the argument are generated. Then, the default argument hierarchy is derived from the requirement hierarchy. Afterward, that argument hierarchy is adapted where ASIL separation or ASIL decomposition was applied on CFTs. These sub-steps are specified in the following Algorithms 6.2 to 6.5 and visualized in Figures 6.5 to 6.8. In the figures, elements with gray background already exist before application of the respective algorithm, and terms in angle brackets are placeholders for element names or ASIL values.

Figure 6.5 sketches the generation rule for the root node of a safety argument specified in Algorithm 6.2. For each hazard <H>, a Hazard CFT exists in the failure propagation model. A context linked to a goal is generated for <H>. The context represents the hazard that is

Algorithm 6.1 Generation of GSN safety arguments

```
for all HazardCFTs $c_h$ do
    GSNmodel $G$ := GenerateSafetyArgumentRoot($c_h$)              ▷ Algo. 6.2 and Fig. 6.5
    GenerateDefaultArgumentHierarchy($G$, $c_h$)                   ▷ Algo. 6.3 and Fig. 6.6
    for all OutputFailureModes $f$ on failure propagation paths to $c_h$ do
        GenerateSeparationArgument($G$, $f$)                       ▷ Algo. 6.4 and Fig. 6.7
        GenerateDecompositionArgument($G$, $f$)                    ▷ Algo. 6.5 and Fig. 6.8
    end for
end for
```

also represented by the event inside the Hazard CFT. It is annotated with the ASIL <ASILX> of the hazard (i.e., the event). The generated goal Safety Goal 1 represents the safety goal that is derived from the hazard. It states the requirement that all failures <P> to <Q> of the top-level function <F> have to be prevented or mitigated that lead to the hazard (i.e., lie on a failure propagation path to the Hazard CFT's event). As the safety goal is directly derived from the hazard, it is annotated with its ASIL <ASILX>.

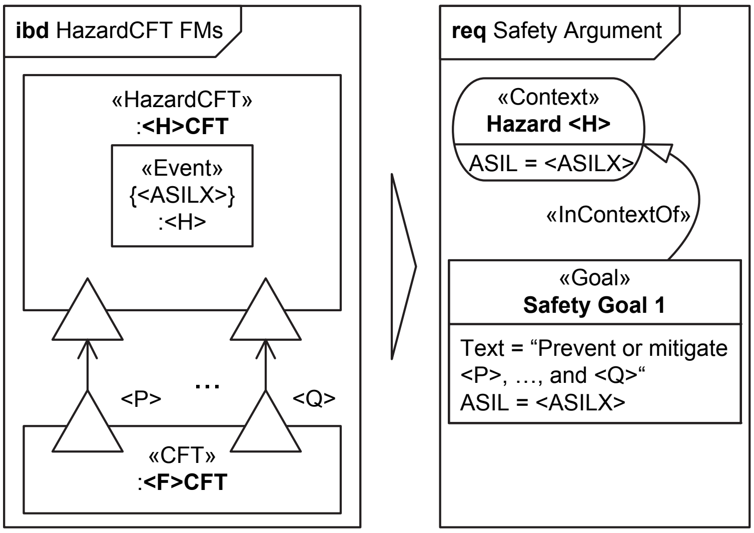


Figure 6.5: GSN generation for hazard

Figure 6.6 sketches the generation rule for the default requirement hierarchy specified in Algorithm 6.3. The CFTs of the failure propagation model follow the structure of the function hierarchy. Concerning the CFTs in the figure, the functions F2 and F3 are children of the function F1. For each function of the function hierarchy, MSDs specify requirements. An MSD is traced by the failure modes of a function's CFT that are caused by violation of the requirement. In the figure, MSD <R1> specifies a requirement on the function F1 and causes the failure mode p$\sim$.

The default requirement hierarchy in the GSN safety argument follows the failure propagation paths starting from the hazard down to the deepest CFT hierarchy level. Starting from the hazard's safety goal (e.g., Safety Goal 1) all MSDs that are traced by failure modes on the paths are mapped to a goal with the same name. Each of those goals is annotated with

Algorithm 6.2 GSN generation for hazard

```
function GENERATESAFETYARGUMENTROOT(HazardCFT $c_h$)
    Create an empty GSNmodel $G$
    Add a context $t_h$ for the hazard that the event of $c_h$ is typed by
    Set the ASIL of the context $t_h$ to $s_x$ of the event/hazard
    Add a goal $g_s$ with ASIL $s_x$ of the event/hazard
    Let $O$ be the set of OutputFailureModes directly linked to InputFailureModes of $c_h$
    Set $g_s$'s text to "Prevent or mitigate <List of all $o \in O$>"
    Connect the goal $g_s$ to the context $t_h$ via an InContextOf link
    return $G$
end function
```

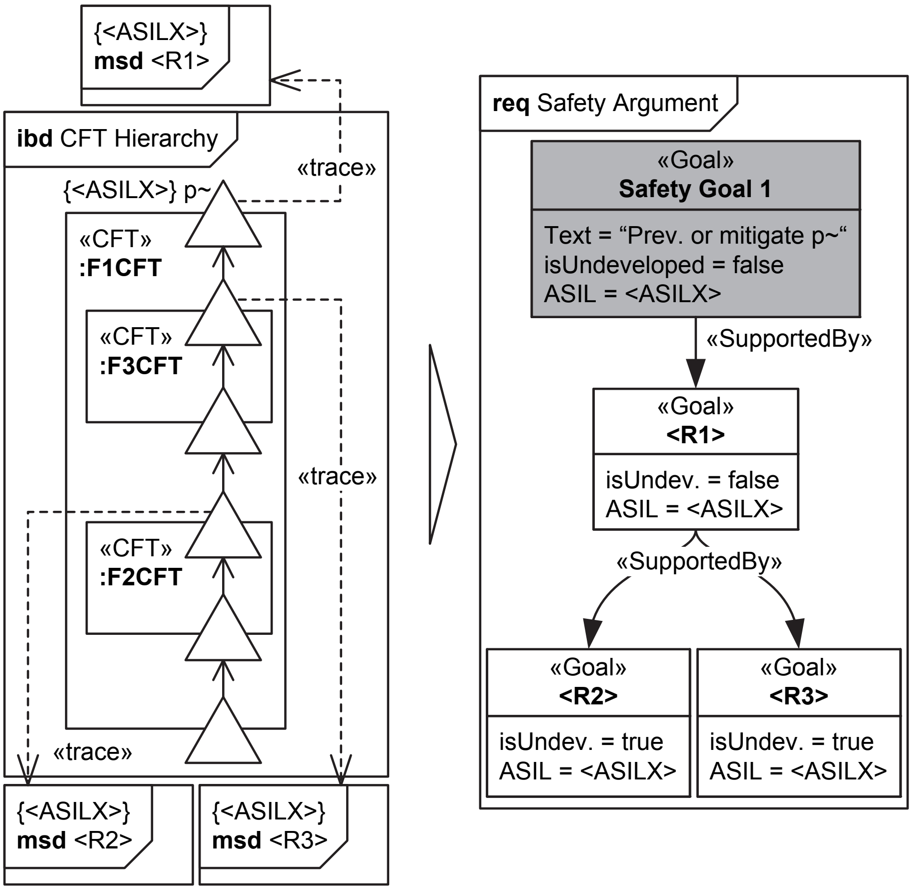


Figure 6.6: GSN generation for default requirement hierarchy

the ASIL of the corresponding MSD. All goals referring to MSDs on the same function/CFT hierarchy level, are on the same goal hierarchy level. In the figure, the MSDs <R2> and <R3> are on the same hierarchy level below MSD <R1> because F2 and F3 are children of F1. Thus, the corresponding goals <R2> and <R3> are on the same hierarchy level below goal <R1>. The goal attribute isUndeveloped is set to false for each goal that has subgoals and to true for each goal without subgoals (e.g., <R2> and <R3>).

Algorithm 6.3 GSN generation for default requirement hierarchy

```
function GENERATEDEFAULTARGUMENTHIERARCHY(GSNmodel $G$, HazardCFT $h$)
    for all InputFailureModes $f$ of HazardCFT $h$ do
        for all OutputFailureModes $p$ directly linked to $f$ do
            for all MSDs $m_{r_1}$ traced from $p$ do
                Add a goal $g_{r_1}$ with the ASIL $s_x$ of the MSD $m_{r_1}$
                Connect the existing safety goal $g_s$ to $g_{r_1}$ via a SupportedBy link
                Set isUndeveloped to false for goal $g_s$
                Set isUndeveloped to true for goal $g_{r_1}$
            end for
            RECURSIVEGENDEFAULTHIERARCHY($p$, $p$'s CFT)
        end for
    end for
end function
function RECURSIVEGENDEFAULTHIERARCHY(OutputFailureMode $p$, CFT $c$)
    for all OutputFailureModes $o$ of direct subCFTs of $c$ on paths to $p$ do
        for all MSDs $m_{r_2}$ traced from $o$ do
            Add a Goal $g_{r_2}$ with the ASIL $s_x$ of the MSD $m_{r_2}$
            for all MSDs $m_{r_1}$ traced from $p$ do
                if $m_{r_1}$ is a parent MSD of $m_{r_2}$ concerning $p$ then
                    Connect the goal $g_{r_1}$ derived from $m_{r_1}$ to $g_{r_2}$ via a SupportedBy link
                    Set isUndeveloped to false for goal $g_{r_1}$
                end if
            end for
            Set isUndeveloped to true for goal $g_{r_2}$
        end for
        RECURSIVEGENDEFAULTHIERARCHY($o$, $o$'s CFT)
    end for
end function
```

Figure 6.7 sketches the generation rule for ASIL Separation arguments specified in Algorithm 6.4. If a CFT <F3>CFT has a lower ASIL than another CFT <F2>CFT and it has no output failure modes on a path to a higher ASIL output failure mode of <F2>CFT, ASIL separation was applied (cf. Definition 5.4 in Section 5.3.1.1). In the figure, the CFT <F3>CFT is assigned with an ASIL <ASILZ> that is lower than the ASIL <ASILX> of the CFT <F2>CFT. The failure mode q of CFT <F2>CFT is on a failure propagation path to the failure mode <P> of <F1>CFT. The CFT <F3>CFT has no failure mode that is on a failure propagation path to either q nor <P>. Hence, its ASIL <ASILZ> is lower than the ASILs assigned to the two failure modes. The failure mode <P> originates from the requirement <R1>. <R2> refines this requirement and is the source of q .

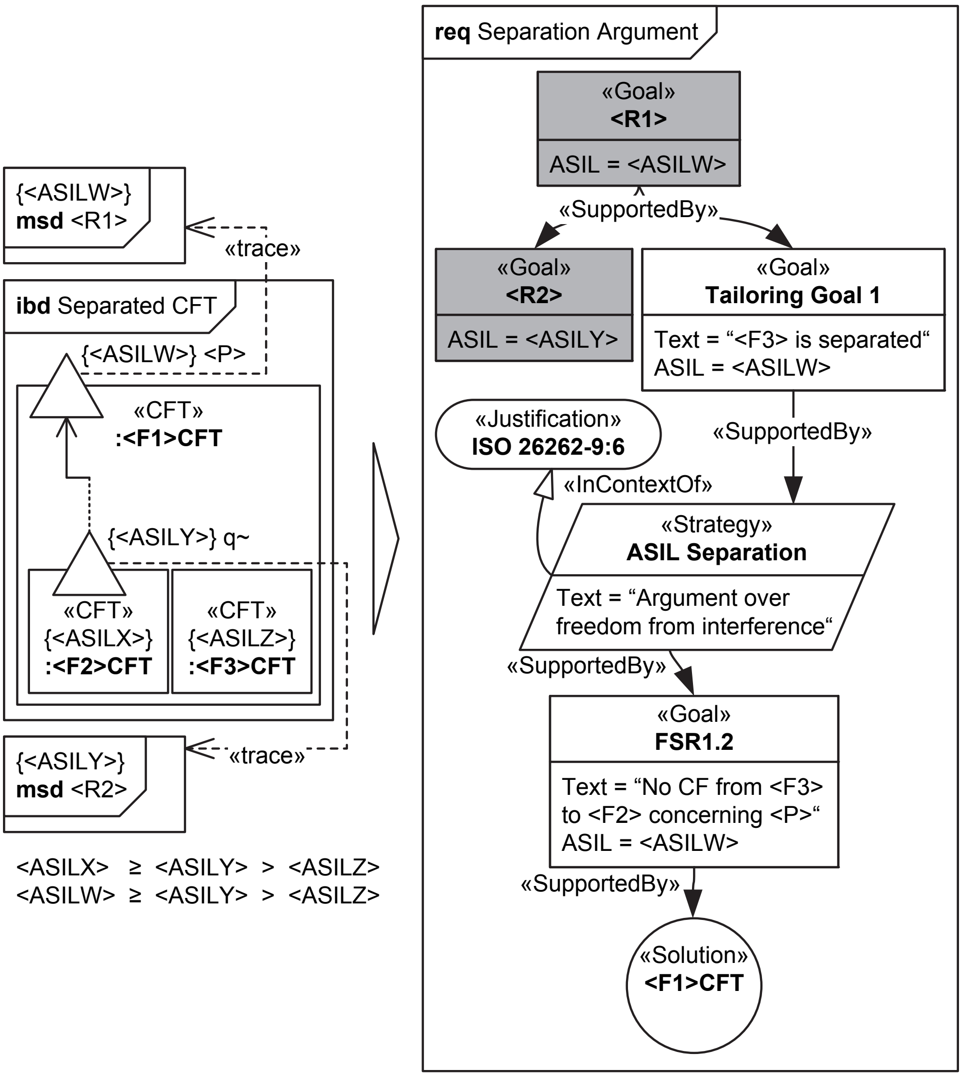


Figure 6.7: GSN generation for ASIL separation

The right of Figure 6.7 shows the safety argument derived from the elements on the left. The goals <R1> and <R2> represent the requirement MSDs with the same name and are generated by the default requirement hierarchy rule (cf. Figure 6.6). All other elements are generated to argue valid separation of <F3> from <F2> concerning the sub-requirements of <R1>. The Tailoring Goal 1 claims that <F3> is separated from <F2> (i.e., <R2>). The strategy ASIL Separation specifies the application of ASIL tailoring in accordance with ISO 26262-9, Clause 6 (cf. Justification ISO 26262-9:6). The supporting goal FSR1.2 specifies the required claim that there are no cascading failures from the function <F3> to <F2> concerning the failure mode <P>. This claim is supported by the solution <F1>CFT referring to the CFT that contains the two CFTs <F2>CFT and <F3>CFT.

Algorithm 6.4 GSN generation for ASIL separation

```
function GenerateSeparationArgument(GSNmodel $G$, OutputFailureMode $p$)
    Let $c_{f_1}$ be the CFT containing $p$
    Let $C$ be the set of all direct child CFTs of $c_{f_1}$
    Let $D \subset C$ be the set of CFTs with FailureModes on a path to $p$
    Let $s_w$ be the ASIL of $p$
    for all CFTs $c_{f_3} \in C \setminus D$ with an ASIL lower than $s_w$ do
        Let $f_3$ be the function traced by $c_{f_3}$
        Add a goal $g_t$ with the text "$f_3$ is separated" and ASIL $s_w$
        for all MSDs $m_{r_1}$ traced from $p$ do
            Connect the existing goal $g_{r_1}$ representing $m_{r_1}$ to $g_t$ via a SupportedBy link
        end for
        Add a strategy $y$ with name "ASIL Separation"
        Connect the goal $g_t$ to the strategy $y$ via a SupportedBy link
        Add a justification $j$ with name "ISO 26262-9:6"
        Connect the strategy $y$ to the justification $j$ via an InContextOf link
        Add a solution $e$ referencing the CFT $c_{f_1}$
        for all CFTs $c_{f_2} \in D$ with an ASIL higher than $c_{f_3}$ do
            Let $f_2$ be the function traced by $c_{f_2}$
            Add a goal $g_2$ with ASIL $s_w$
            Set $g_2$'s text to "No CF from $f_3$ to $f_2$ concerning $p$"
            Connect the strategy $y$ to the goal $g_2$ via a SupportedBy link
            Connect the goal $g_2$ to the solution $e$ via a SupportedBy link
        end for
    end for
end function
```

Figure 6.8 sketches the generation rule for ASIL Decomposition arguments specified in Algorithm 6.5. If a CFT <F2>CFT contains an AND gate and at least one output failure mode s$\sim$ of a directly preceding CFT <F3>CFT on a path to the AND gate has a lower ASIL than the output failure mode <P> of <F2>CFT on a path from the AND gate, ASIL decomposition was applied (cf. Definition 5.7 in Section 5.3.1.1). In the figure, the CFT <F2>CFT contains an AND gate on a failure propagation path to an output failure mode <P> with ASIL <ASILX>. In addition, two input failure modes s and t are on a path to the AND gate. As s has a lower ASIL <ASILV> than <P>, ASIL decomposition concering <P> has been identified as possible during ASIL allocation. The preceding output failure mode s$\sim$ must have an ASIL <ASILY> that is greater or equal to the ASIL of s because of the default ASIL allocation constraint (cf. Figure 5.14 a) on page 118). If ASIL decomposition concerning <P> has been applied, <ASILY> is lower than the ASIL <ASILX> of <P>.

The requirements <R3> and <R4> are the reason for the AND gate in the CFT <F2>CFT. Each of them leads to the output failure mode <P> but caused by a different input failure mode. The requirements <R1> and <R2> lead to those input failure modes.

The right of Figure 6.8 shows the safety argument derived from the elements on the left. The goals <R1> to <R4> represent the requirement MSDs with the same name and are generated by the default requirement hierarchy rule (cf. Figure 6.6). All other elements

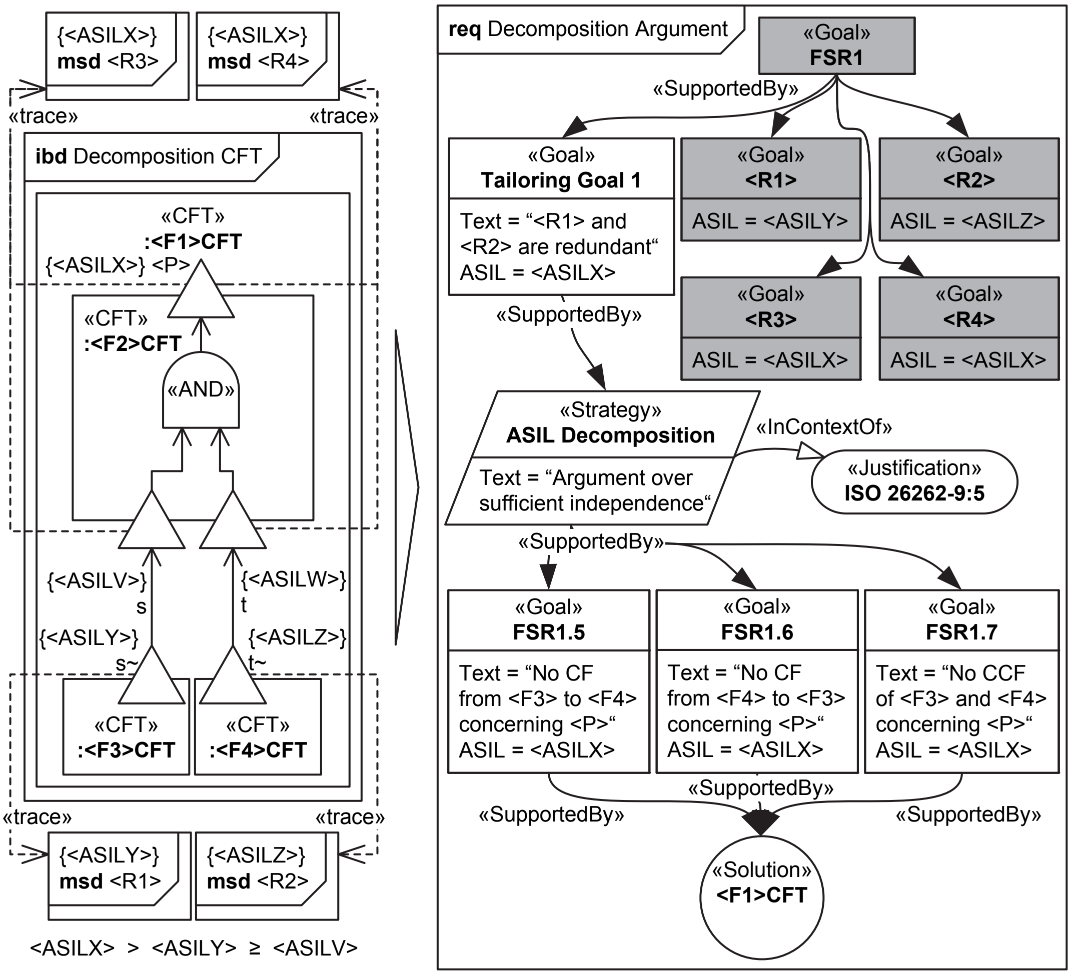


Figure 6.8: GSN generation for ASIL decomposition

are generated to argue valid decomposition concerning the output failure mode <P>. The Tailoring Goal 1 claims that <R1> and <R2> are redundant. The strategy ASIL Decomposition specifies the application of ASIL tailoring in accordance with ISO 26262-9, Clause 5 (cf. Justification ISO 26262-9:5). The supporting goals FSR1.5 and FSR1.6 specify the required claims that there are no cascading failures from the function <F3> to <F4> concerning the failure mode <P>, and vice versa. The supporting goal FSR1.7 specifies the required claim that there are no common cause failures of the functions <F3> and <F4> concerning the failure mode <P>. These claims are supported by the solution <F1>CFT referring to the CFT that contains the two CFTs <F3>CFT and <F4>CFT.

6.3 Assumptions & Limitations

The ASIL tailoring process presented in this thesis focuses on functional safety requirements. Hence, safety argument claims (i.e., GSN goals) only refer to safety goals and functional safety requirements, not to later technical, software, or hardware safety requirements (cf.

Algorithm 6.5 GSN generation for ASIL decomposition

```
function GenerateDecompositionArgument(GSNmodel $G$, OutputFM $p$)
    Let $s_x$ be the ASIL of $p$
    for all AndGates $a$ of $p$'s CFT $c_{f_2}$ on a path to $p$ do
        Let $I$ be the set of all InputFailureModes of $c_{f_2}$ on a path to $a$
        Let $O$ be the set of the first OutputFailureModes on paths to $i \in I$
        if $\exists\ o_s \in O$ on a path to $i_s \in I$ both with lower ASILs than $s_x$ then
            Let $R$ be the set of all MSDs traced from any $o \in O$
            Let $C$ be the set of all CFTs containing any $o \in O$
            Let $F$ be the set of functions represented by the CFTs in $C$
            Let $G$ be the set of existing goals supported by goals representing all $r \in R$
            Add a goal $g_t$ with ASIL $s_x$
            Set $g_t$'s text to "<List of all $r \in R$> are redundant"
            Connect all goals in $G$ to the goal $g_t$ via a SupportedBy link
            Add a strategy $y$ with name "ASIL Decomposition"
            Connect the goal $g_t$ to the strategy $y$ via a SupportedBy link
            Add a justification $j$ with name "ISO 26262-9:5"
            Connect the strategy $y$ to the justification $j$ via an InContextOf link
            Add a goal $g_7$ with ASIL $s_x$
            Set $g_7$'s text to "No CCF of <List of all $f \in F$> concerning $p$"
            Connect the strategy $y$ to the goal $g_7$ via a SupportedBy link
            Add a solution $e$ referencing the CFT containing all $c \in C$
            Connect the goal $g_7$ to the solution $e$ via a SupportedBy link
            for all combinations $f_1, f_2 \in F \times F$ with $f_1 \neq f_2$ do
                Add a goal $g_5$ with ASIL $s_x$
                Set $g_5$'s text to "No CF from $f_1$ to $f_2$ concerning $p$"
                Connect the strategy $y$ to the goal $g_5$ via a SupportedBy link
                Connect the goal $g_5$ to the solution $e$ via a SupportedBy link
            end for
        end if
    end for
end function
```

Section 2.2). In line with that, safety evidence (i.e., GSN solutions) only refer to safety analysis results (i.e., CFTs) and not to evidence from later development phases like test results. We assume that GSN goals referring to functional safety requirements that are not further refined/supported by additional GSN elements (their attribute isUndeveloped is true), will be extended by GSN goals referring to technical safety requirements in later development phases.

6.4 Related Work

We divide the work related to our method of documenting safety arguments into two categories: work related to the used safety argument representation (Section 6.4.1), and to generating safety arguments (Section 6.4.2).

6.4.1 Safety Argument Notations

The EAST-ADL is an architecture description language for automotive electronic systems [EAS13]. It is comprised of a variety of modeling aspects. In its dependability part, it includes a safety argument notation of claims, warrants, and evidence that is also available as UML profile [EAS10]. In contrast to the UML profile used in this thesis, it does not contain any constraints to restrict the language's static semantics. Although the latests EAST-ADL version defines the mentioned own safety argument notation, an earlier version used GSN (cf. [CJL$^+$08]) as we do in this thesis.

Similarly to our approach, Beckers et al. define a UML profile for GSN including OCL constraints for static semantics [BCF$^+$14]. In contrast to our profile, theirs does not extend the SyML, and thus, does not rely on the SysML requirement stereotype but instead defines its own requirement stereotype. Moreover, their profile extends GSN to distinguish different types of goals. For instance, they specialize the stereotype "Goal" into "Safety Goal" and "Functional Safety Requirement". We see this as a mixture of concerns between requirements engineering and safety assurance. In this thesis, we use MSDs for specifying safety requirements and GSN for specifying safety assurance arguments. The goals in safety arguments only link to MSD requirements for traceability. Furthermore, their profile is incomplete because it does not contain GSN solutions. These are essential to specify safety argument evidence (e.g., safety analysis or test results).

6.4.2 Generating Safety Arguments

Kelly and McDermid extend the GSN language to be able to specify patterns of safety arguments [KM97]. This language extension contains placeholders for variable element texts and multiplicities for edges. Kelly and McDermid provide a set of example patterns using the GSN extension but they do not define SIL tailoring argument patterns. The semantics of the extension are not formally defined. For instance, if a Supported-By link with multiplicity n is connected to a goal containing a placeholder, it is unclear whether the placeholder is instantiated n times with the same value or whether it represents a list of n values. Hence, this GSN extension cannot be used to automatically generate safety arguments.

Denney and Pai present a formalization of Kelly and McDermid's GSN extension for safety argument patterns [DP13]. Based on the formalization they provide an algorithm to automatically instantiate patterns. This algorithm requires as input a table of value populations for a pattern's placeholders to instantiate the pattern. The algorithm does not use existing work products, like requirements specifications or safety analysis results, to derive the placeholder values. Consequently, SIL tailoring arguments are not automatically derived from other work products as we do in this thesis.

In [SPB16], Sorokos et al. generate SIL tailoring arguments for the aerospace domain based on SIL allocations calculated via HiP-HOPS [APW+14]. However, they neither present generation rules nor an algorithm for automatic derivation of the SIL tailoring arguments, they only show example results. In addition, they only generate arguments for SIL tailoring, and do not include the hierarchical safety argument structure following the safety requirement hierarchy from the hazards down to applied SIL tailorings. Furthermore, they do not consider traceability from safety arguments to related work products for safety impact analysis. Eventually, their approach is not applied in early requirements engineering but works on system components instead of functions or functional safety requirements.

Retouniotis et al. extend the approach of Sorokos et al. by traceability to design models and related fault trees [RPS+17]. This approach still is only presented by example, sketching a generation rule and the applying algorithm in a figure. They present a custom meta model for GSN arguments and trace links to system components and fault trees. This meta model is an extension of the meta model used by HiP-HOPS. Hence, their approach is not relying on a standard modeling language like UML, and thus, is not easily integrated in a standard modeling environment. On the contrary, the approach presented in this thesis solely extends the UML and SysML by profiles, and thus, can be adopted in any UML modeling environment. Similarly to Sorokos et al., Retouniotis et al. still do not include the hierarchical safety argument structure following the safety requirement hierarchy from the hazards down to applied SIL tailorings. Furthermore, the traceability is limited to system design components and system level fault trees. They do not trace to functions, corresponding failure propagation models, nor requirements. Hence, their approach is not applicable in early requirements engineering, which is the focus of the approach presented in this thesis.

6.5 Conclusion

This chapter describes the details about automated derivation of safety arguments from work products of the ASIL tailoring process presented in Chapter 3. Section 6.2.1 describes how we apply the model-based safety argument notation GSN, and its integration with the other ASIL tailoring process work products. Moreover, Section 6.2.2 describes the automatic generation of GSN safety arguments from ASIL allocated failure propagation models (cf. Chapter 5) and corresponding functions and functional safety requirements (cf. Chapter 4).

The use of GSN for documenting safety arguments and its integration with the related UML-based work products fosters model-based safety assessment in early development phases. The use of a common base language enables integration in a single modeling environment and improves safety case impact analysis and maintenance. The automatic generation of safety arguments saves time in safety case construction and reduces manually introduced errors. Especially, the automatic generation also considers applied ASIL tailorings and derives validity arguments conforming to standards like ISO 26262. All in all, this approach supports time-saving construction of safety cases right from the beginning of development (cf. Challenge C3 in Section 1.2).

7

EVALUATION

This chapter presents the evaluation of the ASIL tailoring process. Section 7.1 gives an overview of the prototype implementation of the automated process steps. In Section 7.2, the applied case study is explained.

7.1 Prototype Implementation

To support and evaluate the concepts from Chapters 3 to 6, we implemented a software prototype. Figure 7.1 shows the components and dependencies of that implementation. Components that were implemented as part of this thesis are depicted with white background, and preexisting components that they depend on, have a gray background.

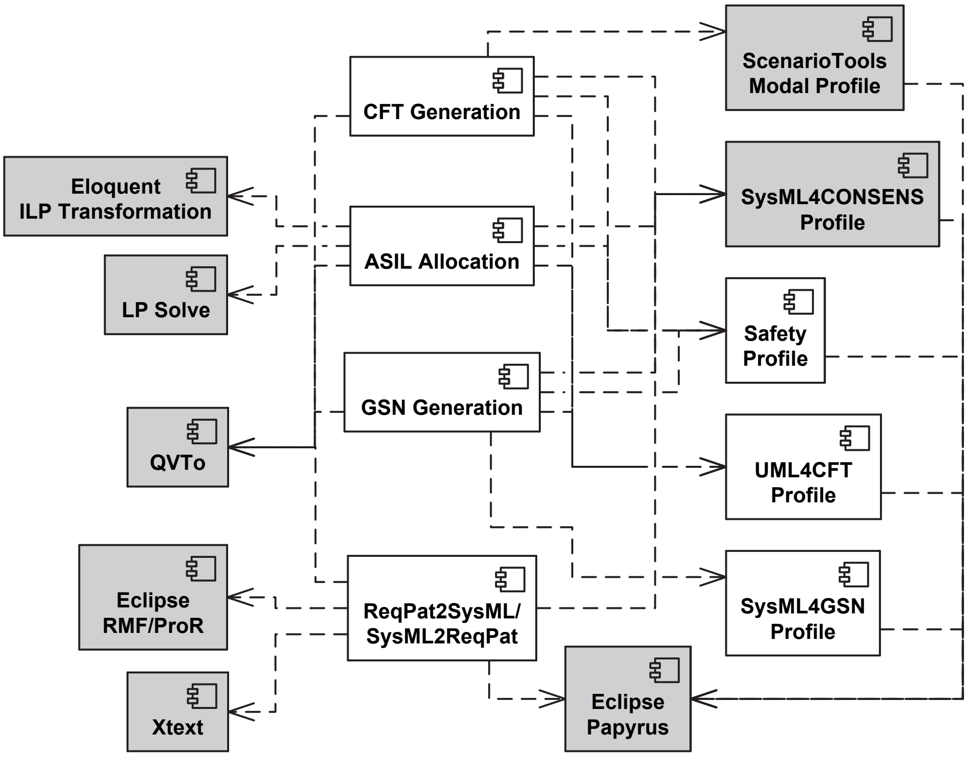


Figure 7.1: Prototype components (white) and dependencies (gray)

All components were implemented as plug-ins for the IDE Eclipse[1]. All models are specified using the UML/SysML editor Eclipse Papyrus[2]. Hence, all required and developed UML profiles are dependent on Papyrus. We use the Modal Profile provided by ScenarioTools[3] to specify MSDs and the SysML4CONSENS Profile for CONSENS model elements (i.e., environment, function hierarchy, and system architecture; cf. Section 2.6). We defined the Safety Profile to specify hazards and safety classified elements (cf. Figure 3.5 on page 45). The UML4CFT Profile is used to model CFTs and Hazard CFTs (cf. Figure 5.5 on page 100). To specify GSN arguments, we defined the SysML4GSN Profile (cf. Figure 6.2 on page 135). The use of the single modeling environment Papyrus allows to have all work products in one SysML model with a single traceability concept from GSN safety arguments to CFT failure propagation and MSD requirements (cf. Figure 3.18 on page 59).

The CFT Generation described in Section 5.2.3 is realized as model-to-model transformation using QVTo[4]. Its implementation is based on student work from members of the project group Aramid [BBB$^+$16]. The ASIL Allocation as explained in Section 5.3 also uses QVTo. The CFT model is transformed into an ILP model based on a meta model provided by Eloquent[5]. The Eloquent ILP Transformation transforms that ILP into input for the ILP solver LP Solve[6]. Eloquent provides an interface to several ILP solvers and also meta heuristics. Hence, the usage of LP Solve is only an example here. We can also switch to other solvers. The ILP solution provided by Eloquent is used to allocate ASILs to the CFT elements, functions, and MSDs.

The GSN Generation explained in Section 6.2.2 uses QVTo to derive safety argument models from hazards, CFTs, and MSDs. Its implementation is based on the bachelor's thesis of Roman Trentinaglia [Tre18].

To be able to also specify textual requirements on functions as described in Section 4.5 (e.g., requirements on absence of cascading and common cause failures), we integrate the function hierarchy model with textual requirements using our plug-in ReqPat [FH15]. It is integrated into the requirements editor Eclipse RMF/ProR[7] and uses Xtext[8] for the specification of textual requirement patterns. Our component ReqPat2SysML/SysML4ReqPat is an adaptation of the transformation described in [FH14] to use QVTo and the CONSENS function hierarchy.

Figure 7.2 shows the implemented ASIL tailoring transformation chain with input and output models. In Step 1 of the ASIL tailoring process, the CONSENS Environment and Hazards are specified using Eclipse Papyrus, the SysML4CONSENS profile, and our safety profile. In Step 2.1, the CONSENS Function Hierarchy is specified with the SysML4CONSENS profile. In Step 2.2, the MSD Requirements are specified using the Modal profile provided by ScenarioTools. In Step 2.3.1, the Hazard CFTs are specified using our UML4CFT profile.

[1] www.eclipse.org

[2] www.eclipse.org/papyrus

[3] www.scenariotools.org

[4] www.eclipse.org/mmt/qvto

[5] www.github.com/upohl/eloquent

[6] lpsolve.sourceforge.net

[7] www.eclipse.org/rmf

[8] www.eclipse.org/xtext

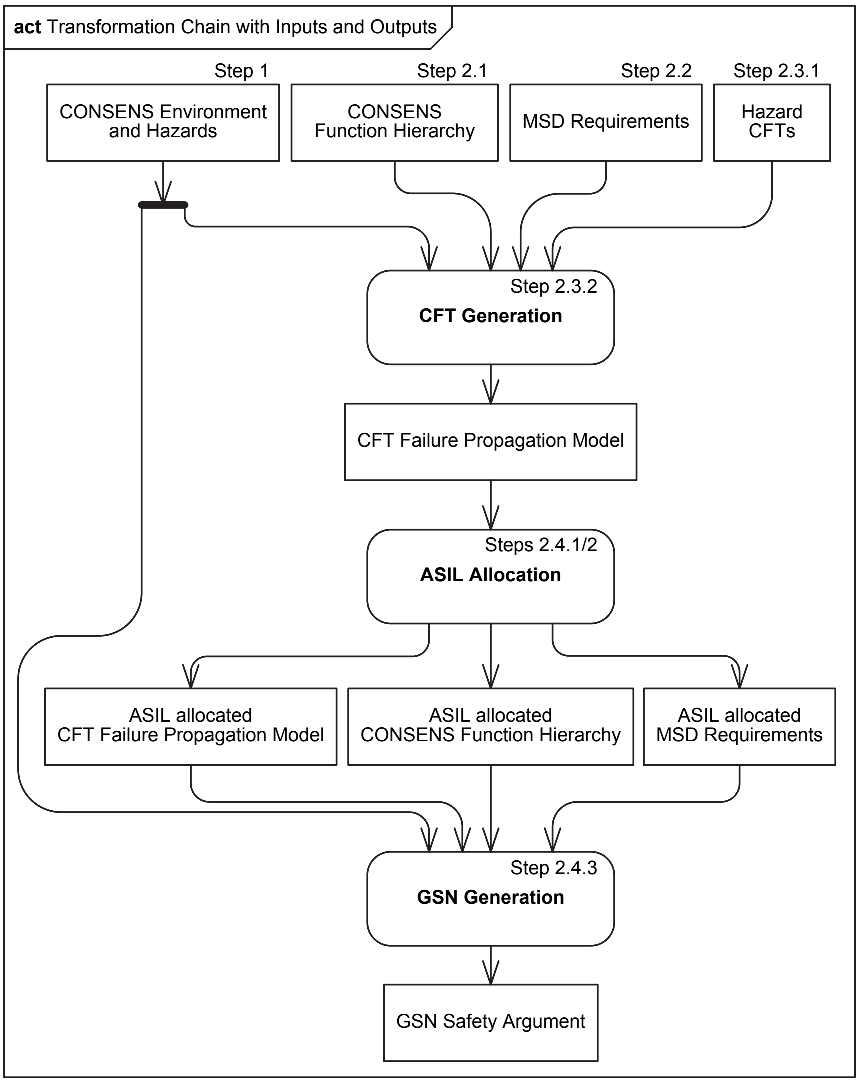

act Transformation Chain with Inputs and Outputs
Step 1
CONSENS Environment and Hazards
Step 2.1
CONSENS Function Hierarchy
Step 2.2
MSD Requirements
Step 2.3.1
Hazard CFTs
Step 2.3.2
CFT Generation
CFT Failure Propagation Model
Steps 2.4.1/2
ASIL Allocation
ASIL allocated CFT Failure Propagation Model
ASIL allocated CONSENS Function Hierarchy
ASIL allocated MSD Requirements
Step 2.4.3
GSN Generation
GSN Safety Argument

Figure 7.2: Chain of transformations with input and output models

These four models are input to the QVTo transformation CFT Generation that generates the CFT Failure Propagation Model using the UML4CFT profile in Step 2.3.2. In addition, the CFT Generation creates traceability links from generated CFT elements to the functions and MSDs that they originate from. The CFT model is used by the QVTo transformation ASIL Allocation in Steps 2.4.1 and 2.4.2 to calculate and allocate ASILs using the Safety Classified Element stereotype of our safety profile. The result is an ASIL allocated CFT Failure Propagation Model, an ASIL allocated CONSENS Function Hierarchy, and ASIL allocated MSD Requirements. Finally, in Step 2.4.3, these ASIL allocated models and the hazards from the CONSENS Environment and Hazards are used by the QVTo transformation GSN Generation to derive the GSN Safety Argument using our SysML4GSN profile. Additionally, it connects safety argument elements via traceability links to hazards, functions, MSD requirements, and CFTs.

7.2 Case Study

To evaluate the applicability and effectiveness of the ASIL tailoring process presented in this thesis, we perform a case study. In that study, the manual and automated process steps are applied on two cases. The automated steps are executed using the prototype implementation explained in Section 7.1.

We perform the case study on the basis of the guidelines defined by Kitchenham et al. [KPP95] and Runeson and Höst [RH08]. Section 7.2.1 describes the posed evaluation questions and the two cases. In Section 7.2.2, hypotheses on the evaluation results and criteria of fulfillment are specified. In Section 7.2.3, the preparations of the case study are described. Section 7.2.4 contains the procedure of collecting and analyzing result data. Section 7.2.5 concludes with the interpretation of the results, and Section 7.2.6 discusses threats to validity.

7.2.1 Context and Cases

The goal of this case study is to evaluate the applicability and effectiveness of the ASIL tailoring process presented in this thesis. Therefore, we evaluate the evaluation questions listed in Section 7.2.1.1. We perform the case study on two cases: a rear door system that automatically opens/closes and locks/unlocks a vehicle's rear door (cf. Section 7.2.1.2), and the EBEAS assistance system for crash avoidance introduced in Section 1.1 (cf. Section 7.2.1.3)

7.2.1.1 Evaluation Questions

To evaluate the applicability and effectiveness of the ASIL tailoring process, we pose the following evaluation questions:

Q1: Does the ASIL tailoring process work for realistic examples?
Q2: Do the automated process steps produce correct results?
Q3: Does the ASIL tailoring process reduce manual effort?

7.2.1.2 Case 1: Rear Door System

Case 1 is a rear door system (RDS) of a car. It automatically opens/closes and unlocks/locks the car's rear door. These actions can be triggered by the car's remote key or by other ECUs of the car.

The rear door system is a realistic example because such systems are already part of series production vehicles. In fact, the case is based on a textual requirements specification document of an automotive OEM that we also used in [Foc16].

The rear door system is a safety-critical ASIL B system because the rear door shall not open while the car is moving, and the door shall be unlocked in case of an accident.

7.2.1.3 Case 2: EBEAS

Case 2 is the Emergency Braking and Evasion Assistance System (EBEAS) as introduced in Section 1.1 and used as running example throughout this thesis. It automatically decides to brake or evade in emergency situations based on coordination with the surrounding vehicles via Vehicle-to-Vehicle communication.

We consider the EBEAS a realistic example because of the following reasons. First, Autonomous Emergency Braking systems are part of series production vehicles already and considered by the Euro NCAP test for vehicle safety since 2014 [Eur14]. Second, examples of Autonomous Emergency Steering systems have been presented by automotive suppliers like Continental [Tut10] and ZF [ZF 16]. Though, they have not yet made it into series production. Third, the combination of autonomous emergency braking and steering has been prototyped in a research project [ISS08]. Fourth, the theoretic feasibility of combining autonomous emergency braking and steering with Vehicle-to-Vehicle communication for decision making has been investigated in [HFK$^+$16]. In conclusion, we argue that systems like the EBEAS are the consequent next step of advanced driver assistance systems towards complex cyber-physical systems.

The EBEAS is highly safety-critical (ASIL D), as it interferes with brakes and steering. If drivers rely on the system, it may not fail to brake or evade when necessary.

7.2.2 Hypotheses

We set the following evaluation hypotheses for this case study. The hypotheses H1 and H2 refer to evaluation question Q1, hypotheses H3 to H5 refer to evaluation question Q2, and hypotheses H6 to H9 refer to evaluation question Q3.

H1: The functional (safety) requirements for the two cases can be specified using MSDs as described in Chapter 4.
We rate this hypothesis as fulfilled if the requirements can be specified using solely the MSD patterns defined in Table 4.4 on page 85.

H2: The CFT Generation supports all MSD patterns defined in Chapter 4.
We rate this hypothesis as fulfilled if each specified MSD pattern results in a CFT obeying the CFT generation rules defined in Section 5.2.3.

H3: The CFT Generation produces correct failure propagation models for the two cases.
We rate this hypothesis as fulfilled if the generated CFTs obey the syntax and semantics of CFTs as defined in Figure 5.3 on page 98, and the model content is as we expected.

H4: The ASIL Allocation produces valid results for the two cases.
We rate this hypothesis as fulfilled if the calculated ASILs obey the ASIL tailoring rules defined in Section 5.3.1.1.

H5: The GSN Generation produces correct safety arguments for the two cases.
We rate this hypothesis as fulfilled if the generated GSN models obey the syntax and semantics of GSN as defined in Figure 6.2 on page 135, and the model content is as we expected.

H6: The performance of the CFT Generation for the two cases is reasonable.
We rate this hypothesis as fulfilled if the execution time of the CFT Generation is below an hour for both cases.

H7: The performance of the ASIL Allocation for the two cases is reasonable.
We rate this hypothesis as fulfilled if the execution time of the ASIL Allocation is below an hour for both cases.

H8: The performance of the GSN Generation for the two cases is reasonable.
We rate this hypothesis as fulfilled if the execution time of the GSN Generation is below an hour for both cases.

H9: The effort saved by the automated process steps outweighs the effort required for specifying MSD requirements.
We rate this hypothesis as fulfilled if the number of generated model elements is greater than the number of specified MSD model elements.

7.2.3 Preparation of the Data Collection

In preparation for the case study, we set up an Eclipse installation that includes the prototype implementation and its dependencies (cf. Section 7.1). In that installation, we prepare a separate project with an empty Papyrus SysML model for both cases. For the performance measurements we use a business laptop with an Intel Core i7-4800MQ (2.7 GHz) with 8 GB RAM. It runs 64 bit Windows 7 SP 1 with 32 bit Java 1.8. We use Eclipse Neon 3.

7.2.4 Data Collection Procedure

We perform the evaluation by going through the steps of the ASIL tailoring process as described in Chapter 3 for both cases.

In Step 1, the environment of the case is specified as defined in the CONSENS systems engineering method. Afterward, the model is extended by hazards with ASIL values as identified in HARA. The hazards are caused by failures later specified in the CFT failure propagation model. The resulting diagram shall look as sketched in Figure 3.4 on page 44.

In Step 2.1, the function hierarchy is specified as defined in the CONSENS method. Each function is extended by ports and interfaces that are required for the message exchange in MSD requirements. The resulting diagram shall look as sketched in Figure 3.8 on page 47.

In Step 2.2, the functional (safety) requirements of the case are specified using the MSD patterns from Section 4.4.2. For each informal requirement from the case's source document, a fitting MSD pattern is chosen and instantiated in the requirements model. We document the chosen patterns and the number of requirements for which no fitting pattern is found to evaluate Hypothesis H1.

In Step 2.3, the CFT failure propagation model is automatically generated from the previous work products using the prototype's CFT Generation (cf. Figure 7.2). First, for each hazard from Step 1, a Hazard CFT is manually specified that defines what failures of the top-level function lead to the occurrence of the hazard. Afterward, the internal failure propagation of the function hierarchy is automatically generated and linked to the Hazard CFTs.

We check each generated CFT model for correctness (cf. Hypothesis H3) using the model validation of Papyrus. It allows to automatically check OCL constraints contained in UML profiles. To be able to use this feature, we translated the informal constrains shown in Figure 5.3 on page 98 into OCL constraints included in our UML4CFT profile (cf. Figure 5.5

on page 100). In addition, we manually review each generated CFT model to check whether the CFT generation rules (cf. Figures 5.9 and 5.10 on page 105 ff.) were applied correctly. For completeness, separately from the two cases, we specify an instance of each MSD pattern that is not used in the two cases, generate a CFT, and apply the same correctness checks as on the two cases (to evaluate Hypothesis H2). Furthermore, for each case, we measure the execution time of three runs of the CFT Generation and document the mean value (to evaluate Hypothesis H6).

In Steps 2.4.1 and 2.4.2, the ASILs of failure modes, functions, and MSDs are automatically calculated based on the CFT model using the prototype's ASIL Allocation (cf. Figure 7.2). First, the ASILs of failure modes and CFTs are calculated, afterward, the ASILs of corresponding functions and MSDs are derived.

We check each calculated ASIL allocation for correctness (cf. Hypothesis H4) by reviewing whether the result obeys the ASIL tailoring rules defined on the CFT meta model (cf. Section 5.3.1.1) and in Figure 5.17 on page 123. Furthermore, for each case, we measure the execution time of three runs of the ASIL Allocation and document the mean value (to evaluate Hypothesis H7).

In Step 2.4.3, GSN safety arguments that assure the validity of applied ASIL tailorings, are automatically generated from the previous work products using the prototype's GSN Generation (cf. Figure 7.2). For each hazard specified in Step 1, a separate GSN argument is generated.

We check each generated GSN argument for correctness (cf. Hypothesis H5) using the model validation of Papyrus. To be able to use this feature, we translated the informal constrains shown in the SysML4GSN profile in Figure 6.2 on page 135 into OCL constraints. In addition, we manually review each generated GSN argument to check its semantic correctness concerning the corresponding MSD requirements and the ASIL allocated CFT model. Furthermore, for each case, we measure the execution time of three runs of the GSN Generation and document the mean value (to evaluate Hypothesis H8).

In Step 3, the system architecture of the case is specified as defined in the CONSENS systems engineering method. This includes the allocation of the leaf functions of the function hierarchy to elements of the architecture. The resulting diagram would look as sketched in Figure 3.17 on page 56. As this is a default step of the systems engineering method, and it is not enhanced by the ASIL tailoring process, we do not apply this step in the case study.

In all steps of the ASIL tailoring process, we count the number of manually specified model elements that are not part of the default CONSENS method. Additionally, we count the number of model elements that are automatically generated by the prototype. This information is used to evaluate Hypothesis H9.

7.2.5 Interpreting the Results

For the analysis of results we rate each hypothesis individually and afterward draw conclusions for the referred evaluation questions.

H1: The functional (safety) requirements for the two cases can be specified using MSDs as described in Chapter 4.

The result of our case study shows that, in both cases, during specification of MSD requirements no MSD pattern was missing. Thus, all requirements could be specified using one of the defined MSD patterns. Therefore, we rate our first evaluation hypothesis H1 as fulfilled.

H2: The CFT Generation supports all MSD patterns defined in Chapter 4.

In addition to the MSD patterns used in the two cases, we generated CFTs for each of the other MSD patterns. The review of these generated CFTs showed no violation of any CFT generation rule from Section 5.2.3. Hence, we also rate our evaluation hypothesis H2 as fulfilled.

H3: The CFT Generation produces correct failure propagation models for the two cases.

For both cases, we checked correctness of the generated CFT model via OCL constraint checking and manual review. The manual review compared the model with the CFT generation rules. In addition, the model was checked for consistency with the hazards specified in the environment model, the function hierarchy, and the MSD requirements. In both cases, no OCL constraint was violated and the manual review showed no deficiencies. Therefore, we rate the evaluation hypothesis H3 as fulfilled.

H4: The ASIL Allocation produces valid results for the two cases.

For both cases, we checked correctness of the ASIL allocation result by manual review of the CFT model that was annotated with ASIL values by the ASIL Allocation. The manual review compared the model with the ASIL tailoring rules. In addition, the model was checked for consistency with the hazard ASILs. In both cases, the manual reviews showed no deficiencies. Therefore, we rate the evaluation hypothesis H4 as fulfilled.

H5: The GSN Generation produces correct safety arguments for the two cases.

For both cases, we checked correctness of the generated GSN model via OCL constraint checking and manual review. The manual review checked for consistency with the hazards specified in the environment model, the function hierarchy, the MSD requirements, and the ASIL allocation. In both cases, no OCL constraint was violated and the manual review showed no deficiencies. Therefore, we rate the evaluation hypothesis H5 as fulfilled.

H6/H7/H8: The performance of the CFT Generation/ASIL Allocation/GSN Generation for the two cases is reasonable.

To evaluate the performance of the automated process steps, we measured the execution times. The results are shown in the bottom of Table 7.1. The execution times of all three automated steps (CFT Generation, ASIL Allocation, and GSN Generation) are below one minute for both cases. Hence, all three together can be executed numerous times a day. Hence, we rate our evaluation hypotheses H6, H7, and H8 as fulfilled.

H9: The effort saved by the automated process steps outweighs the effort required for specifying MSD requirements.

During evaluation, we counted the number of manually specified elements that are solely specified to enable the automated steps of the ASIL tailoring process. This are

- hazards,
- function ports, their interfaces, and their operations,

- MSDs, their lifelines, messages, and conditions, and
- Hazard CFTs, their gates and CFT links.

Additionally, we counted the number of automatically generated elements. This are

- CFT elements (i.e., CFTs, CFT gates, and CFT links),
- ASIL annotations (Safety Classified Element stereotypes),
- GSN elements, and
- trace links between elements of the different models for traceability.

The resulting numbers are shown in Table 7.1.

For successful application of the ASIL tailoring process, in sum 124 model elements have to be specified manually for Case 1 (323 for Case 2). If these are specified, 345 (797) model elements are automatically generated. This is a ratio of 2.8 for Case 1 and 2.5 for Case 2. Therefore, we argue that the number of generated elements outweighs the number of manually specified elements and evaluation hypothesis H9 is fulfilled.

Table 7.1: Evaluation results

Metric	**Case 1 (RDS)**	**Case 2 (EBEAS)**
No. of specified hazards	2	4
No. of specified function elements	25	54
No. of specified MSD elements	78	249
No. of specified Hazard CFT elements	19	16
No. of manually specified elements	124	323
No. of generated CFT elements	110	263
No. of generated ASIL annotations	91	186
No. of generated GSN elements	64	136
No. of generated traceability links	80	212
Sum of generated elements	345	797
Mean execution time of CFT Generation	3s	7s
Mean execution time of ASIL Allocation	3s	14s
Mean execution time of GSN Generation	3s	27s
Sum of mean execution times	9s	48s

Evaluation Questions

Evaluation question Q1 asks whether the ASIL tailoring process works for realistic examples. The two related hypotheses H1 and H2 are fulfilled. The requirements of the two cases could be specified using solely our MSD patterns. As the MSD patterns are derived from a set of pattern catalogs that were developed based on industry requirements specifications, we argue that our MSD patterns also work for more industry cases. We showed that from our MSD patterns, correct CFT failure propagation models can automatically be generated. Hence, all other process steps that rely on the CFT model can be executed. In conclusion, we see our ASIL tailoring process as ready for realistic projects and rate the evaluation question Q1 as fulfilled.

Evaluation question Q2 targets correctness of automated process step results. We evaluated the correctness of the CFT Generation, the ASIL Allocation, and the GSN Generation by the hypotheses H3, H4, and H5. The results show that they all are fulfilled. Hence, we rate evaluation question Q2 as fulfilled.

Evaluation question Q3 asks whether the ASIL tailoring process reduces manual effort for safety engineering and ASIL tailoring on requirements level. To answer this question, we evaluated if the automated process steps can be used by the safety manager in his daily work. Hypotheses H6, H7, and H8 target reasonable execution times of the automated steps. In his daily work, the safety manager shall be able to execute all automated steps regularly. The results show that they can be executed numerous times, as their execution takes less than a minute for the two cases. In addition to execution time, we compared the number of additional work for the safety manager (i.e., model elements he/she has to specify for the ASIL tailoring process) with the number of omitted work (i.e., model elements that are automatically generated). The results show that the number of automatically generated model elements outweighs the number of model elements that need to be specified manually. For the two cases, more than twice as much elements are generated as manually specified. In general, the number of generated CFT elements is strongly dependent on the number of function ports and their interface operations. For each operation of a function, a requirement (MSD) is specified that can result in failure modes of different types for the respective operation. The number of generated GSN elements is strongly dependent on the number of hazards. For each hazard, a separate safety assurance argument is generated. Hence, our approach is especially valuable for highly safety-critical systems with many hazards and complex functionality (many interactions between functions). Independent from safety engineering, the usage of MSDs for requirements engineering is beneficial for complex functionality, as they support automated requirements validation and verification (cf. Section 2.7). If MSDs are already specified for that purpose, the additional effort of our ASIL tailoring process is even lower than described above. In conclusion, we rate the evaluation question Q3 as fulfilled.

7.2.6 Threats to Validity

Case studies have limitations and their results rely to a large extent on the research design. Thus, we see the following threats to validity for this case study.

7.2.6.1 Construct Validity

The case study was designed and conducted by the same researcher that developed the approach. Therefore, a threat is that the construction of the case study by the same person has a bias towards the developed approach. To mitigate this, the case study design and the research questions have been discussed with other researchers.

In addition, case study results have not been evaluated by safety engineering experts from industry. As a mitigation strategy, Case 1 was based on a requirements specification from industry that already contained ASIL allocations for requirements and functions (including applied ASIL tailorings). Hence, the evaluation results of Case 1 were compared to expert results from industry.

The selected cases might be too few and small to viably measure scalability of transformation execution times. However, the measured execution times allow numerous executions per day, and we consider even one execution per day as less effort than manually applying the steps that are automated by our approach. Similarly, the selected cases might be too few and small to viably measure scalability of the ratio of generated model elements vs. manually specified elements. To mitigate this threat, we chose a small case (Case 1) and a larger case (Case 2), such that a possible dependency of the ratio on the case size is more visible than with equally sized cases.

7.2.6.2 External Validity

Concerning the generalizability of the case study results, a threat is that the execution time of the transformations might differ on other platforms. As a mitigation strategy, we used a platform that we consider a standard business environment that would also be found in industry, and documented the used platform in Section 7.2.3.

Furthermore, the number of used cases might be too small to draw generalizable conclusions. To mitigate this threat, we included hypothesis H2 that is checked independently from the cases. For that hypothesis, we check whether the CFT Generation works with a whole catalog of MSD patterns that were derived from industry requirements specifications (cf. Section 4.4.2).

Additionally, the selected cases might not be representative. This threat is mitigated by using a case representing an existing industry system (Case 1) and an advanced case (Case 2) that we consider a realistic, complex cyber-physical system of the near future (cf. Section 7.2.1.3). Both cases stem from the automotive industry but we showed in Section 5.3.3 that safety standards from other domains have similar SILs that can be translated to integer numbers for ILP solving.

7.2.6.3 Reliability

Concerning the reproducibility of our case study results, a threat is that the execution times might differ if the used platform is no longer available. To mitigate this threat, we described the used platform in Section 7.2.3. Therefore, an equal platform can be set up, or if that is not possible, execution times on a different platform (e.g., more powerful) can be set in relation.

Another threat to reproducibility is that the input requirements specification of Case 1 is not publicly available. The input of Case 2 is available as technical report [HFK$^+$16]. To mitigate this threat and ease reproduction, we provide model screenshots in Appendix A.

A further threat is that the used prototype implementation might not be available in the future. To mitigate this, the implemented conceptual algorithms are defined in Chapters 5 and 6 and can be reimplemented.

Finally, all manual review results are dependent on the reviewer's expertise. To minimize the extent and complexity of manual reviews, we defined OCL rules that can be automatically checked, and translated the informal ASIL tailoring rules (cf. Section 2.5) to the CFT language (cf. Section 5.3.1.1) to ease manual verifiability.

8

CONCLUSION

In this thesis, we counter the challenge of early safety effort planning with a SIL tailoring process that is integrated in the development phases of requirements engineering. It supports the safety manager and requirements engineer by automated steps to generate and analyze failure propagation models and to derive safety assurance arguments.

This chapter summarizes the challenges and contributions of this thesis in Section 8.1, and points to directions for future work in Section 8.2.

8.1 Summary

The high degree of innovation in mechatronic systems domains leads to so-called cyber-physical systems that are characterized by their complex functionality and communication with other systems and surroundings. Exemplary for this development is the increasing complexity of advanced driver assistance systems (ADAS) that are realized by connecting ECUs which previously had separate functionality. These ADAS make use of huge amounts of sensory data and Vehicle-to-X communication to automatically take decisions and control brakes or steering. Such systems are not only complex but also highly safety-critical. If an ADAS unintentionally steers or fails to brake when necessary, people are in danger. Central ECUs in those systems realize functions with different levels of safety-criticality and, by that, form so-called mixed-criticality systems. For example, in an emergency braking system, the function activating the brakes is more safety-critical than the function that warns the following traffic by activating the hazard lights. A function's safety-criticality stems from the hazards that are caused by its possible failures. Hazard safety-criticality is categorized by so-called safety integrity levels (SIL) that are defined by safety standards like ISO 26262.

The safety-criticality (i.e., the determined SIL) not only describes the risk of potential harm to people. It also dictates the required degree of rigor to be applied in the development of a system to prevent the hazards or mitigate their consequences. A SIL that is determined for a hazard, propagates through all development process phases and work products. Thus, a high SIL requires the application of safety measures with a high degree of rigor in all phases of development. A high SIL accordingly implies a high safety effort. Additionally, if system failures that would cause a hazard, are detected late in the process, this causes expensive iterations to repeat safety measures or apply additional ones.

In mixed-criticality systems, less critical parts should not require the same high safety effort. If parts of the safety requirements (and the subsystems satisfying them) can be assigned with a lower SIL, this can reduce safety effort because less rigorous safety measures are required. To reduce the number of high SIL requirements and subsystems, the safety manager can apply so-called SIL tailoring. For example, if the non-safety-critical infotainment system of a vehicle is separated from safety-critical systems, it may keep a lower SIL. For the automotive domain, SIL tailoring rules are explicitly defined in ISO 26262. The underlying principles are transferable to other mechatronic systems domains as described in Section 5.3.3. To reduce safety effort (e.g., the number and complexity of high-SIL system parts), the safety manager's goal is to find possible failures and plan safety measures as early as possible. Due to the complexity and mixed-criticality of ADAS and cyber-physical systems in general, reaching this goal is challenging.

The contribution of this thesis is a systematic, tool-supported SIL tailoring process applied in safety requirements engineering (cf. Chapter 3) that copes with the three challenges identified in Section 1.2 as follows:

C1: Complexity of Mixed-Criticality Systems

The complexity and mixed-criticality of systems put a challenge on applying early SIL tailoring because the validity of each tailoring has to be assured by safety analysis. This requires knowledge about possible failures and their propagation paths through the system and its subsystems. The system complexity infers complex failure propagation paths that have to be derived from complex requirements which are typically only specified in informal language. Informal language is prone to ambiguity, incompleteness, and inconsistency. This leads to error-prone and incomplete safety analysis results.

To cope with Challenge C1, the SIL tailoring process is integrated into the model-based systems engineering method CONSENS that supports the interdisciplinary development of complex cyber-physical systems. CONSENS uses a functional abstraction of the system under development prior to specifying the system architecture. We extend and use this function hierarchy to structure and decompose functional requirements and functional safety requirements in form of Modal Sequence Diagrams (MSD) (cf. Chapter 4). This functional decomposition is suggested by the requirements engineering standard ISO 29148 [Int11i]. We use MSDs as a graphical, formal, scenario-based modeling language to cope with the complexity of different application scenarios. MSDs can be simulated for requirements validation and verified for requirements consistency. To further support the specification of correct, formal MSD requirements, we set up an MSD requirement pattern catalog comprised of patterns concerning the occurrence and order of information exchange, including real-time and safety. In addition, we combine the function hierarchy with safety analysis models in form of Component Fault Trees (CFT), such that failure propagation paths can be specified on the function hierarchy to analyze its mixed-criticality (i.e., the required SIL of each function and requirement).

C2: Safety Requirements Engineering Dilemma

To apply valid SIL tailoring, possible failures and their propagation paths through the system need to be analyzed. Unfortunately, there is a problem known as the safety requirements engineering dilemma [Ber98]. It states that a system's possible failures and resulting hazards can best be found late in the development process, but are ideally already known during requirements engineering. In the later phases of the development process, most system details are known, so it is easier to identify possible failures. However, changing the system to prevent or mitigate a failure, then causes expensive development iterations. Because of the missing knowledge about possible failures and their propagation paths, safety analyses on the requirements level are prone to incompleteness. Whenever new information about failures is gained, the safety analyses have to be repeated. As this produces manual effort, in practice, safety analyses are currently not applied on the requirements level, and existing research approaches apply SIL tailoring on technical architectures only.

To cope with Challenge C2, we contribute a concept and prototypical implementation that automatically generates CFT failure propagation models for a function hierarchy based on functional (safety) requirements specified as MSDs (cf. Section 5.2). Furthermore, we contribute a concept and prototypical implementation that uses the failure propagation models to automatically calculate a SIL allocation to functions and requirements, incorporating applied SIL tailorings (cf. Section 5.3). These automated process steps minimize the manual effort to a review of the generated models and SIL allocation. Additionally, these automated steps can be repeated whenever changes to the function hierarchy or requirements are made.

C3: Safety Case Construction and Maintenance

Safety standards like the ISO 26262 require a so-called safety case, which provides the "argument that the safety requirements for an item are complete and satisfied by evidence compiled from work products of the safety activities during development" [Int11a]. The safety manager has to argue the validity of each applied SIL tailoring in the safety case. As safety analyses are time-consuming and error-prone in early development stages (cf. previous challenges), SIL tailoring is applied on technical architectures rather than functional models of the requirements analysis phase. Consequently, safety cases are not built in that phase either. Building a safety case with arguments for valid early SIL tailoring requires traceability throughout the whole development process (from the initial hazards to the safety requirements, to introduced safety mechanisms, to safety analysis results). Reverse engineering that information to build the safety case in the end of development, is a tedious and error-prone task that may identify missing safety measures and, thus, cause expensive process iterations. Existing research approaches describe ways to document model-based safety cases and safety case patterns. However, there is no approach that automatically constructs and maintains the safety case according to applied SIL tailorings on the requirements level. Safety case automation in general was identified as open challenge [HWK+14].

To cope with Challenge C3, we contribute a concept and prototypical implementation that automatically derives model-based safety cases using the Goal Structuring Notation (GSN) from the SIL allocated functions, MSD requirements, and CFT failure propagation model (cf. Chapter 6). The generated safety cases include assurance arguments for the validity of applied SIL tailorings. This automated derivation of safety arguments minimizes manual effort for safety case maintenance, as it can be repeated to update the safety case whenever new hazards are identified or changes to the safety requirements are made.

Altogether, the presented SIL tailoring process integrates concepts of model-based safety engineering with model-based requirements and systems engineering to support the safety manager in planning safety effort on early functional requirements. The evaluation (cf. Chapter 7) showed that the automated process steps for safety analysis, SIL allocation, and safety argumentation save time and avoid manually introduced errors.

8.2 Future Work

The results of this thesis give rise to possibilities for future research that we highlight in the following. Firstly, future work may enhance the contributions of this thesis by overcoming the limitations and relaxing the assumptions that we described in the corresponding Sections 3.7, 4.6, 5.4, and 6.3. In addition, the presented SIL tailoring process should be evaluated in further projects and mechatronic systems domains. In the following paragraphs, we discuss further directions for future research.

For generating CFT failure propagation models, we assume that requirements are specified solely using our catalog of MSD requirement patterns. In future work, it could be evaluated what has to be changed, such that the CFT generation works for arbitrary MSDs that do not necessarily conform to any of the patterns. Although the catalog is helpful to specify requirements correctly to one's intents, this would increase the applicability and robustness of our approach.

For requirements that cannot be specified as MSDs (e.g., quality requirements concerning the failure rate of a system), we provide a bidirectional transformation of the function hierarchy to textual requirements using patterns in a controlled natural language (cf. Section 4.5). To support requirements engineers that do not understand the MSD language, and to support requirements exchange with purely textual requirements management tools, it would be beneficial to also synchronize MSD requirements with a textual representation. This is a complex task, as a single MSD can be comprised of several atomic requirements and a single requirement can be formalized using multiple MSDs. In addition, one has to find the right balance between ambiguity and formality to not reinvent languages like Z [Int02] or CTL [EC82].

A further direction for future research is to adapt our approach to also work for following development phases. Functional safety requirements are refined into technical safety requirements that have to be satisfied by the system architecture. For instance, if technical safety requirements can be specified using MSDs on the structural basis of the CONSENS active structure, CFT failure propagation models could be generated from those MSDs. Then, the SIL allocation and GSN generation could be reused. The challenge of this research is to extend MSDs to work on the technical level or to find a different formal language that can be used for CFT generation. Especially requirements on continuous information flow like feedback loops or flow of hydraulic fluids cannot be specified using the MSD language definition used in this thesis [HFK+16].

The approach presented in this thesis supports the safety manager in analyzing and arguing safety of functions and their requirements including the validity of applied SIL tailorings. As innovations and complexity of cyber-physical systems will increase, so will their requirements specifications. Hence, it will get more difficult to find valid SIL tailoring possibilities. Therefore, it would support the safety manager, if there was an approach that makes suggestions how to change or evolve the function hierarchy and its requirements to enable or improve SIL tailorings. This is a huge challenge because the suggestion mechanism will have to understand the semantics of requirements and incorporate knowledge about functionality cohesion and cost. Otherwise it will suggest many unreasonable solutions.

BIBLIOGRAPHY

The bibliography is structured into three parts: my own publications, the theses I supervised, and foreign literature. Details about my contributions to the papers listed under my own publications are given in Appendix B.

Own Publications

[FHK+18] FOCKEL, MARKUS; HOLTMANN, JÖRG; KOCH, THORSTEN; SCHMELTER, DAVID: “Formal, Model- and Scenario-based Requirement Patterns”. In: *6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*. Funchal, Portugal, Jan. 2018.

[FHK+17] FOCKEL, MARKUS; HOLTMANN, JÖRG; KOCH, THORSTEN; SCHMELTER, DAVID: *Model-based Requirement Patterns*. Tech. rep. tr-ri-17-354. Software Engineering Department, Fraunhofer IEM / Software Engineering Group, Heinz Nixdorf Institute, Oct. 2017.

[Foc16] FOCKEL, MARKUS: “ASIL Tailoring on Functional Safety Requirements”. In: *5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR)*. Ed. by SKAVHAUG, AMUND; GUIOCHET, JÉRÉMIE; SCHOITSCH, ERWIN; BITSCH, FRIEDEMANN. Vol. 9923. LNCS. Co-located with SAFECOMP 2016. Trondheim, Norway: Springer International Publishing, Sept. 2016. DOI: `10.1007/978-3-319-45480-1_24`.

[HFK+16] HOLTMANN, JÖRG; FOCKEL, MARKUS; KOCH, THORSTEN; SCHMELTER, DAVID; BRENNER, CHRISTIAN; BERNIJAZOV, RUSLAN; SANDER, MARCEL: *The MechatronicUML Requirements Engineering Method: Process and Language*. Tech. rep. tr-ri-16-351. Software Engineering Department, Fraunhofer IEM / Software Engineering Group, Heinz Nixdorf Institute, Dec. 2016.

[FH15] FOCKEL, MARKUS; HOLTMANN, JÖRG: “ReqPat: Efficient Documentation of High-quality Requirements using Controlled Natural Language”. In: *Proc. of the 23rd International Requirements Engineering Conference 2015 (RE 2015)*. Ottawa, Canada: IEEE, Aug. 2015, pp. 280–281. DOI: `10.1109/RE.2015.7320438`.

[MFH15] MEYER, JAN; FOCKEL, MARKUS; HOLTMANN, JÖRG: “Systementwurf unter Einbeziehung funktionaler Sicherheit bei automobilen Steuergeräten”. In: *Tag des Systems Engineering 2015 (TdSE 2015).* (in German). Ulm, Germany, Nov. 2015.

[FH14] FOCKEL, MARKUS; HOLTMANN, JÖRG: “A Requirements Engineering Methodology Combining Models and Controlled Natural Language”. In: *4th International Model-Driven Requirements Engineering Workshop (MoDRE).* Co-located with RE 2014. Karlskrona, Sweden: IEEE, Aug. 2014, pp. 67–76. DOI: 10.1109/MoDRE.2014.6890827.

[FHM14] FOCKEL, MARKUS; HOLTMANN, JÖRG; MEYER, MATTHIAS: “Mit Satzmustern hochwertige Anforderungsdokumente effizient erstellen”. In: *OBJEKTspektrum* RE/2014 (June 2014). (Online Themenspecial Requirements Engineering) (in German).

[DFH+13] DAUN, MARIAN; FOCKEL, MARKUS; HOLTMANN, JÖRG; TENBERGEN, BASTIAN: *Goal-Scenario-Oriented Requirements Engineering for Functional Decomposition with Bidirectional Transformation to Controlled Natural Language. Case Study "Body Control Module".* Tech. rep. ICB-Research Report No. 55. Universität Duisburg-Essen, May 2013.

[FHH+12] FOCKEL, M.; HEIDL, P.; HOLTMANN, J.; HORN, W.; HÖFFLINGER, J.; HÖNNINGER, H.; MEYER, J.; MEYER, M.; SCHÄUFFELE, J.: “Application and Evaluation in the Automotive Domain”. In: *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology.* Ed. by POHL, KLAUS; HÖNNINGER, HARALD; ACHATZ, REINHOLD E.; BROY, MANFRED. Springer, 2012. Chap. 12, pp. 157–175. ISBN: 978-3-642-34614-9 (Online), 978-3-642-34613-2 (Print).

[FHM12] FOCKEL, MARKUS; HOLTMANN, JÖRG; MEYER, JAN: “Semi-automatic Establishment and Maintenance of Valid Traceability in Automotive Development Processes”. In: *2nd International Workshop on Software Engineering for Embedded Systems (SEES).* Co-located with ICSE 2012. Zürich, Switzerland, June 2012, pp. 37–43. ISBN: 978-1-4673-1853-2.

Supervised Theses

[BBB+16] BASAK, AINDRILA; BERNIJAZOV, RUSLAN; BÖRDING, PAUL; EIKERLING, HENDRIK; ENSTE, PATRICK; FLOHRE, ANDREAS; NEUMANN, CONRAD; STOLTE, FLORIAN: “Project Group Aramid”. Final Documentation. Paderborn: Paderborn University, 2016.

[Tre18] TRENTINAGLIA, ROMAN: “Deriving Pattern-based Safety Arguments”. Bachelor’s Thesis. Paderborn: Paderborn University, May 2018.

Foreign Publications

[Aca11] ACATECH, ed.: *Cyber-Physical Systems – Innovationsmotor für Mobilität, Gesundheit, Energie und Produktion*. acatech POSITION. Berlin/Heidelberg: Springer, 2011.

[ACD93] ALUR, R.; COURCOUBETIS, C.; DILL, D.: “Model-Checking in Dense Real-Time”. In: *Information and Computation* 104.1 (1993), pp. 2–34. DOI: `http://dx.doi.org/10.1006/inco.1993.1024`.

[Ass18] ASSURANCE CASE WORKING GROUP (ACWG): *Goal Structuring Notation Community Standard*. Version 2. Jan. 2018.

[AHK$^+$08] ATIR, YORAM; HAREL, DAVID; KLEINBORT, ASAF; MAOZ, SHAHAR: “Object Composition in Scenario-Based Programming”. In: *Fundamental Approaches to Software Engineering: 11th International Conference (FASE 2008)*. Ed. by FIADEIRO, JOSÉ LUIZ; INVERARDI, PAOLA. Budapest, Hungary: Springer Berlin Heidelberg, Mar. 2008, pp. 301–316. DOI: `10.1007/978-3-540-78743-3_23`.

[AGL$^+$15] AUTILI, M.; GRUNSKE, L.; LUMPE, M.; PELLICCIONE, P.; TANG, A.: “Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar”. In: *IEEE Transactions on Software Engineering* 41.7 (July 2015), pp. 620–638. DOI: `10.1109/TSE.2015.2398877`.

[AIP07] AUTILI, M.; INVERARDI, P.; PELLICCIONE, P.: “Graphical scenarios for specifying temporal properties: an automated approach”. In: *Automated Software Engineering* 14.3 (2007), pp. 293–340. DOI: `10.1007/s10515-007-0012-6`.

[Aut10] AUTOMOTIVE SPECIAL INTEREST GROUP (SIG): *Automotive SPICE: Process Reference Model, v4.5*. 2010.

[ALR$^+$04] AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C.: “Basic concepts and taxonomy of dependable and secure computing”. In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (Jan. 2004), pp. 11–33. DOI: `10.1109/TDSC.2004.2`.

[APW$^+$14] AZEVEDO, L. S.; PARKER, D.; WALKER, M.; PAPADOPOULOS, Y.; ARAUJO, R. E.: “Assisted Assignment of Automotive Safety Requirements”. In: *IEEE Software* 31.1 (Jan. 2014), pp. 62–68. ISSN: 0740-7459. DOI: `10.1109/MS.2013.118`.

[APW$^+$13] AZEVEDO, LUIS SILVA; PARKER, DAVID; WALKER, MARTIN; PAPADOPOULOS, YIANNIS; ARAUJO, RUI ESTEVES: “Automatic Decomposition of Safety Integrity Levels: Optimization by Tabu Search”. In: *2nd Workshop on Critical Automotive applications: Robustness & Safety, CARS 2013*. 2013.

[Bah14] BAHR, NICHOLAS J.: *System Safety Engineering and Risk Assessment*. 2nd edition. CRC Press, 2014. ISBN: 9781466551619.

[Bai15] BAILEY, BRIAN: *The Wild West of Automotive*. Apr. 9, 2015. URL: `http://semiengineering.com/the-wild-west-of-automotive/`.

[BCF+14] BECKERS, KRISTIAN; CÔTÉ, ISABELLE; FRESE, THOMAS; HATEBUR, DENIS; HEISEL, MARITTA: "Systematic Derivation of Functional Safety Requirements for Automotive Systems". In: *Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2014)*. Ed. by BONDAVALLI, ANDREA; DI GIANDOMENICO, FELICITA. Florence, Italy: Springer International Publishing, 2014, pp. 65–80. DOI: `10.1007/978-3-319-10506-2_5`.

[Ber98] BERRY, D. M.: "The safety requirements engineering dilemma". In: *Procs. of the ninth Int. Workshop on Software Specification and Design*. Apr. 1998, pp. 147–149. DOI: `10.1109/IWSSD.1998.667930`.

[Bit00] BITSCH, FRIEDEMANN: "Classification of Safety Requirements for Formal Verification of Software Models of Industrial Automation Systems". In: *Procs. of the 13th Int. Conf. on Software and Systems Engineering and their Applications (ICSSEA)*. Paris, France, 2000.

[Bit01] BITSCH, FRIEDEMANN: "Safety Patterns – The Key to Formal Specification of Safety Requirements". In: *Proceedings of the 20th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2001)*. Ed. by VOGES, UDO. Budapest, Hungary: Springer Berlin Heidelberg, Sept. 2001, pp. 176–189. DOI: `10.1007/3-540-45416-0_18`.

[BAB+12] BLANQUART, JEAN-PAUL; ASTRUC, JEAN-MARC; BAUFRETON, PHILIPPE; BOULANGER, JEAN-LOUIS; DELSENY, HERVÉ; GASSINO, JEAN; LADIER, GÉRARD; LEDINOT, EMMANUEL; LEEMAN, MICHEL; MACHROUH, JOSEPH; QUÉRÉ, PHILIPPE; RICQUE, BERTARND: "Criticality categories across safety standards in different domains". In: *Embedded Real Time Software and Systems (ERTS 2012)*. Toulouse, France, Feb. 2012.

[BLH+13] BLOM, HANS; LÖNN, HENRIK; HAGL, FRANK; PAPADOPOULOS, YIANNIS; REISER, MARK-OLIVER; SJÖSTEDT, CARL-JOHAN; CHEN, DE-JIU; KOLAGARI, RAMIN TAVAKOLI: *EAST-ADL – An Architecture Description Language for Automotive Software-Intensive Systems*. White Paper. Version 2.1.12. 2013.

[BGH+14] BRENNER, CHRISTIAN; GREENYER, JOEL; HOLTMANN, JÖRG; LIEBEL, GRISCHA; STIEGLBAUER, GERALD; TICHY, MATTHIAS: "ScenarioTools Real-Time Play-Out for Test Sequence Validation in an Automotive Case Study". In: *13th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2014)*. 2014.

[BGP13] BRENNER, CHRISTIAN; GREENYER, JOEL; PANZICA LA MANNA, VALERIO: "The ScenarioTools Play-Out of Modal Sequence Diagram Specifications with Environment Assumptions". In: *12th Int. Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2013)*. 2013.

[BDH+12] BROY, M.; DAMM, W.; HENKLER, S.; POHL, K.; VOGELSANG, A.; WEYER, T.: “Introduction to the SPES Modeling Framework”. In: *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Ed. by POHL, KLAUS; HÖNNINGER, HARALD; ACHATZ, REINHOLD; BROY, MANFRED. Springer, 2012. Chap. 3, pp. 31–49. ISBN: 978-3-642-34614-9 (Online), 978-3-642-34613-2 (Print).

[CEN11] CENELEC: *EN 50128:2011: Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems*. 2011.

[Cha09] CHARETTE, ROBERT: “This Car Runs on Code”. In: *IEEE Spectrum* (Feb. 2009).

[CJL+08] CHEN, DEJIU; JOHANSSON, ROLF; LÖNN, HENRIK; PAPADOPOULOS, YIANNIS; SANDBERG, ANDERS; TÖRNER, FREDRIK; TÖRNGREN, MARTIN: “Modelling Support for Design of Safety-Critical Automotive Embedded Systems”. In: *27th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2008)*. Ed. by HARRISON, MICHAEL D.; SUJAN, MARK-ALEXANDER. Newcastle upon Tyne, UK: Springer Berlin Heidelberg, Sept. 2008, pp. 72–85. DOI: `10.1007/978-3-540-87698-4_9`.

[Cot13] COTNER, JOHN: *Functional Safety and ISO 26262 Compliance - APF-AUT-T0503*. presentation. Sept. 2013.

[Cro15] CROLLA, DAVID: *Encyclopedia of Automotive Engineering*. John Wiley & Sons, 2015. ISBN: 9780470974025.

[DP13] DENNEY, EWEN; PAI, GANESH: “A Formal Basis for Safety Case Patterns”. In: *Proc. of the 32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2013)*. Ed. by BITSCH, F.; GUIOCHET, J.; KAANICHE, M. Vol. 8153. LNCS. Toulouse, France: Springer, Sept. 2013, pp. 21–32. DOI: `10.1007/978-3-642-40793-2_3`.

[Dep12] DEPARTMENT OF DEFENSE, USA: *MIL-STD-882E: Department of Defense Standard Practice – System Safety*. 2012.

[DPS+14] DHOUIBI, MOHAMED SLIM; PERQUIS, JEAN-MARC; SAINTIS, LAURENT; BARREAU, MIHAELA: “Automatic Decomposition and Allocation of Safety Integrity Level Using System of Linear Equations”. In: *Proc. of the 4th International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO 2014)*. 2014. ISBN: 978-1-61208-321-6.

[Dor14] DOROCIAK, RAFAL: “Systematik zur frühzeitigen Absicherung der Sicherheit und Zuverlässigkeit fortschrittlicher mechatronischer Systeme”. PhD thesis. Paderborn, Germany: Paderborn University, 2014.

[DAC99] DWYER, M. B.; AVRUNIN, G. S.; CORBETT, J. C.: “Patterns in property specifications for finite-state verification”. In: *Proceedings of the 1999 International Conference on Software Engineering (ICSE 99)*. Los Angeles, USA, May 1999, pp. 411–420. DOI: `10.1145/302405.302672`.

[EAS10] EAST-ADL ASSOCIATION: *UML Profile of EAST-ADL Domain Model Specification, v2.1.0*. 2010.

[EAS13] EAST-ADL ASSOCIATION: *EAST-ADL Domain Model Specification, v2.1.12*. 2013.

[EJ09] EBERT, C.; JONES, C.: “Embedded Software: Facts, Figures, and Future”. In: *IEEE Computer* 42.4 (Apr. 2009), pp. 42–52. ISSN: 0018-9162. DOI: 10.1109/MC.2009.118.

[EC82] EMERSON, E. ALLEN; CLARKE, EDMUND M.: “Using branching time temporal logic to synthesize synchronization skeletons”. In: *Science of Computer Programming* 2.3 (1982), pp. 241–266. DOI: http://dx.doi.org/10.1016/0167-6423(83)90017-5.

[Eri05] ERICSON, CLIFTON A.: “Event Tree Analysis”. In: *Hazard Analysis Techniques for System Safety*. John Wiley & Sons, Inc., 2005, pp. 223–234. ISBN: 9780471739425.

[Eur14] EUROPEAN NEW CAR ASSESSMENT PROGRAMME (EURO NCAP): *Test protocol – AEB systems v1.0*. Jan. 2014. URL: http://euroncap.blob.core.windows.net/media/1569/aeb-test-protocol-v-10.pdf.

[GRS14] GAUSEMEIER, J.; RAMMIG, F.-J.; SCHÄFER, W.: “Design Methodology for Intelligent Technical Systems”. In: *Lecture Notes in Mechanical Engineering*. Springer, 2014.

[GPD+09] GAUSEMEIER, JÜRGEN; PÖSCHL, MARTIN; DEYTER, SEBASTIAN; KAISER, LYDIA: “Modeling and analyzing fault-tolerant mechatronic systems”. In: *Proceedings of the 17th International Conference on Engineering Design (ICED 09)*. Stanford, USA, Aug. 2009.

[GTS04] GIESE, HOLGER; TICHY, MATTHIAS; SCHILLING, DANIELA: “Compositional Hazard Analysis of UML Component and Deployment Models”. In: *SAFECOMP 2004*. Potsdam, Germany: Springer, 2004. DOI: 10.1007/978-3-540-30138-7_15.

[GH09] GORDON, MICHAL; HAREL, DAVID: “Generating Executable Scenarios from Natural Language”. In: *Computational Linguistics and Intelligent Text Processing*. Ed. by GELBUKH, ALEXANDER. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 456–467. DOI: 10.1007/978-3-642-00382-0_37.

[Gre11] GREENYER, JOEL: “Scenario-based Design of Mechatronic Systems”. PhD thesis. Paderborn, Germany: Paderborn University, 2011.

[GBC+13] GREENYER, JOEL; BRENNER, CHRISTIAN; CORDY, MAXIME; HEYMANS, PATRICK; GRESSI, ERIKA: “Incrementally Synthesizing Controllers from Scenario-based Product Line Specifications”. In: *9th Joint Meeting of the ESEC/FSE*. New York, NY, USA: ACM, 2013, pp. 433–443. DOI: 10.1145/2491411.2491445.

[GGK⁺16] Greenyer, Joel; Gritzner, Daniel; Katz, Guy; Marron, Assaf: "Scenario-Based Modeling and Synthesis for Reactive Systems with Dynamic System Structure in ScenarioTools". In: *MoDELS 2016 Demo and Poster Sessions*. Vol. 1725. 2016, pp. 16–32.

[GHM⁺15] Greenyer, Joel; Haase, Max; Marhenke, Jörg; Bellmer, Rene: "Evaluating a Formal Scenario-based Method for the Requirements Analysis in Automotive Software Engineering". In: *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ESEC/FSE 2015. Bergamo, Italy: ACM, 2015, pp. 1002–1005. doi: 10.1145/2786805.2804432.

[HM08] Harel, David; Maoz, Shahar: "Assert and negate revisited: Modal semantics for UML sequence diagrams". In: *Software & Systems Modeling* 7.2 (2008), pp. 237–252.

[HWK⁺14] Hatcliff, John; Wassyng, Alan; Kelly, Tim; Comar, Cyrille; Jones, Paul: "Certifiably Safe Software-dependent Systems: Challenges and Directions". In: *Future of Software Engineering*. FOSE 2014. Hyderabad, India: ACM, 2014, pp. 182–200. doi: 10.1145/2593882.2593895.

[HTZ⁺12] Höfig, Kai; Trapp, Mario; Zimmer, Bastian; Liggesmeyer, Peter: "Modeling Quality Aspects: Safety". In: *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Ed. by Pohl, Klaus; Hönninger, Harald; Achatz, Reinhold; Broy, Manfred. Springer-Verlag, 2012, pp. 107–118. isbn: 978-3-642-34613-2.

[HBM⁺16] Holtmann, Jörg; Bernijazov, Ruslan; Meyer, Matthias; Schmelter, David; Tschirner, Christian: "Integrated and iterative systems engineering and software requirements engineering for technical systems". In: *Journal of Software Evolution and Process* (May 2016).

[HM13] Holtmann, Jörg; Meyer, Matthias: "Play-out for Hierarchical Component Architectures". In: *11th Workshop Automotive Software Engineering*. Vol. P-220. GI-Edition - Lecture Notes in Informatics (LNI). Bonner Köllen Verlag, 2013.

[Ins12] Institute of Electrical and Electronics Engineers (IEEE): *IEEE 802.11-2012: IEEE Standard for Information technology – Telecommunications and information exchange between systems, Local and metropolitan area networks – Specific requirements Part 11*. 2012.

[Int15] International Council on Systems Engineering (INCOSE): *Systems Engineering Handbook: A guide for system life cycle processes and activities*. 4th edition. Wiley, 2015. isbn: 9781118999400.

[Int01] International Electrotechnical Commission (IEC): *IEC 61882:2001: Hazard and operability studies (HAZOP studies) – Application guide*. 2001.

[Int03] International Electrotechnical Commission (IEC): *IEC 60300-3-1:2003: Dependability management – Part 3-1: Application guide – Analysis techniques for dependability – Guide on methodology*. 2003.

[Int05] International Electrotechnical Commission (IEC): *IEC 62061:2005: Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems*. 2005.

[Int06a] International Electrotechnical Commission (IEC): *IEC 60812:2006: Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*. 2006.

[Int06b] International Electrotechnical Commission (IEC): *IEC 61025:2006: Fault tree analysis (FTA)*. 2006.

[Int06c] International Electrotechnical Commission (IEC): *IEC 61165:2006: Application of Markov techniques*. 2006.

[Int10a] International Electrotechnical Commission (IEC): *IEC 61508:2010: Functional safety of electrical/electronic/programmable electronic safety-related systems*. 2010.

[Int02] International Organization for Standardization (ISO): *ISO 13568:2002: Information technology – Z formal specification notation – Syntax, type system and semantics*. 2002.

[Int08] International Organization for Standardization (ISO): *ISO 15288:2008(E): Systems and software engineering – System life cycle processes*. 2008.

[Int10b] International Organization for Standardization (ISO): *ISO 25119:2010: Tractors and machinery for agriculture and forestry – Safety-related parts of control systems*. 2010.

[Int11a] International Organization for Standardization (ISO): *ISO 26262-1:2011(E): Road vehicles – Functional safety. Part 1: Vocabulary*. 2011.

[Int11b] International Organization for Standardization (ISO): *ISO 26262:2011(E): Road vehicles – Functional safety*. 2011.

[Int11c] International Organization for Standardization (ISO): *ISO 26262-2:2011(E): Road vehicles – Functional safety. Part 2: Management of functional safety*. 2011.

[Int11d] International Organization for Standardization (ISO): *ISO 26262-3:2011(E): Road vehicles – Functional safety. Part 3: Concept phase*. 2011.

[Int11e] International Organization for Standardization (ISO): *ISO 26262-4:2011(E): Road vehicles – Functional safety. Part 4: Product development at the system level*. 2011.

[Int11f] International Organization for Standardization (ISO): *ISO 26262-6:2011(E): Road vehicles – Functional safety. Part 6: Product development at the software level*. 2011.

[Int11g] International Organization for Standardization (ISO): *ISO 26262-8:2011(E): Road vehicles – Functional safety. Part 8: Supporting processes*. 2011.

[Int11h] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 26262-9:2011(E): Road vehicles – Functional safety. Part 9: ASIL-oriented and safety-oriented analyses*. 2011.

[Int11i] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 29148:2011(E): Systems and software engineering – Life cycle processes – Requirements engineering*. 2011.

[Int12a] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 15504-5:2012: Information technology – Process assessment – Part 5: An exemplar software life cycle process assessment model*. 2012.

[Int12b] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 26262-10:2012(E): Road vehicles – Functional safety. Part 10: Guideline on ISO 26262*. 2012.

[IG14] INTERNATIONAL REQUIREMENTS ENGINEERING BOARD IREB E.V.; GLINZ, MARTIN: *A Glossary of Requirements Engineering Terminology – Standard Glossary for the Certified Professional for Requirements Engineering (CPRE) Studies and Exam*. Version 1.6. May 2014.

[Inv14] INVENSITY GMBH: *Automobilindustrie: 90 Prozent der Innovationen finden bei Elektronik und Software statt*. press release. Apr. 29, 2014. URL: http://blog.invensity.com/2014/04/29/automobilindustrie-90-prozent-der-innovationen-finden-bei-elektronik-und-software-statt/.

[ISS08] ISERMANN, R.; SCHORN, M.; STÄHLIN, U.: “Anticollision system PRORETA with automatic braking and steering”. In: *Vehicle System Dynamics* 46.sup1 (2008), pp. 683–694. DOI: 10.1080/00423110802036968.

[IKD+13] IWANEK, PETER; KAISER, LYDIA; DUMITRESCU, ROMAN; NYSSEN, ALEXANDER: “Fachdisziplinübergreifende Systemmodellierung mechatronischer Systeme mit SysML und CONSENS”. In: *Tag des Systems Engineering 2013*. 2013.

[KLM03] KAISER, BERNHARD; LIGGESMEYER, PETER; MÄCKEL, OLIVER: “A New Component Concept for Fault Trees”. In: *8th Australian workshop on Safety critical systems and software*. 2003.

[KDH+13] KAISER, LYDIA; DUMITRESCU, ROMAN; HOLTMANN, JÖRG; MEYER, MATTHIAS: “Automatic Verification of Modeling Rules in Systems Engineering for Mechatronic Systems”. In: *Proceedings of the ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. ASME, July 2013.

[Kel98] KELLY, T. P.: “Arguing Safety - A Systematic Approach to Safety Case Management”. PhD thesis. York, UK: University of York, Sept. 1998.

[KM97] KELLY, T. P.; MCDERMID, J. A.: “Safety Case Construction and Reuse Using Patterns”. In: *Proceedings of the 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP 97).* Ed. by DANIEL, PETER. York, UK, Sept. 1997, pp. 55–69. DOI: 10.1007/978-1-4471-0997-6_5.

[KPP95] KITCHENHAM, B.; PICKARD, L.; PFLEEGER, S. L.: “Case studies for method and tool evaluation”. In: *IEEE Software* 12.4 (July 1995), pp. 52–62. DOI: 10.1109/52.391832.

[KC05] KONRAD, SASCHA; CHENG, BETTY H. C.: “Real-time specification patterns”. In: *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005).* 2005, p. 372. DOI: 10.1145/1062455.1062526.

[KHP+05] KUGLER, HILLEL; HAREL, DAVID; PNUELI, AMIR; LU, YUAN; BONTEMPS, YVES: “Temporal Logic for Scenario-Based Specifications”. In: *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005).* Ed. by HALBWACHS, NICOLAS; ZUCK, LENORE D. Edinburgh, UK: Springer Berlin Heidelberg, Apr. 2005, pp. 445–460. DOI: 10.1007/978-3-540-31980-1_29.

[Kus16] KUSS, INGO: *Audi zFAS – Enorme Datenmengen bewältigen.* July 1, 2016. URL: http://www.elektroniknet.de/automotive/assistenzsysteme/artikel/131797/.

[LF09] LAMI, GIUSEPPE; FALCINI, FABIO: “Is ISO/IEC 15504 Applicable to Agile Methods?” In: *Proceedings of the 10th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2009).* Ed. by ABRAHAMSSON, PEKKA; MARCHESI, MICHELE; MAURER, FRANK. Pula, Italy: Springer Berlin Heidelberg, May 2009, pp. 130–135. DOI: 10.1007/978-3-642-01853-4_16.

[LW10] LAMM, J. G.; WEILKIENS, T.: “Funktionale Architekturen in SysML”. In: *Tag des Systems Engineering 2010.* Ed. by MAURER, M.; SCHULZE, S.-O. Munich, Germany: Carl Hanser Verlag, 2010.

[LW14] LAMM, JESKO G.; WEILKIENS, TIM: “Method for Deriving Functional Architectures from Use Cases”. In: *Syst. Eng.* 17.2 (June 2014), pp. 225–236. DOI: 10.1002/sys.21265.

[Lam77] LAMPORT, LESLIE: “Proving the Correctness of Multiprocess Programs”. In: *IEEE Transactions on Software Engineering* SE-3.2 (Mar. 1977), pp. 125–143. DOI: 10.1109/TSE.1977.229904.

[LLN+09] LARSEN, KIM G.; LI, SHUHAO; NIELSEN, BRIAN; PUSINSKAS, SAULIUS: “Verifying Real-Time Systems against Scenario-Based Requirements”. In: *Formal Methods: Second World Congress (FM 2009). Proceedings.* Ed. by CAVALCANTI, ANA; DAMS, DENNIS R. Eindhoven, The Netherlands: Springer Berlin Heidelberg, Nov. 2009, pp. 676–691. DOI: 10.1007/978-3-642-05089-3_43.

[LGP11] Lauer, Christoph; German, Reinhard; Pollmer, Jens: “Fault Tree Synthesis from UML Models for Reliability Analysis at Early Design Stages”. In: *SIGSOFT Softw. Eng. Notes* 36.1 (Jan. 2011), pp. 1–8. doi: `10.1145/1921532.1921558`.

[LYZ+11] Li, W.; Yang, Z.; Zhang, P.; Wang, Z.: “Model Checking WS-BPEL with Universal Modal Sequence Diagrams”. In: *10th IEEE/ACIS International Conference on Computer and Information Science*. May 2011, pp. 328–333. doi: `10.1109/ICIS.2011.58`.

[MAL+11] Mader, Roland; Armengaud, Eric; Leitner, Andrea; Kreiner, Christian; Bourrouilh, Quentin; Griessnig, Gerhard; Steger, Christian; Weiss, Reinhold: “Computer-Aided PHA, FTA and FMEA for Automotive Embedded Systems”. In: *Computer Safety, Reliability, and Security (SAFECOMP 2011)*. Ed. by Flammini, Francesco; Bologna, Sandro; Vittorini, Valeria. Springer Berlin Heidelberg, 2011, pp. 113–127. doi: `10.1007/978-3-642-24270-0_9`.

[MAL+12] Mader, Roland; Armengaud, Eric; Leitner, Andrea; Steger, Christian: “Automatic and optimal allocation of safety integrity levels”. In: *Proc. of the Annual Reliability and Maintainability Symposium, RAMS 2012*. IEEE, 2012, pp. 1–6. isbn: 978-1-4577-1849-6. doi: `10.1109/RAMS.2012.6175431`.

[MEP+05] McKelvin Jr., Mark L.; Eirea, Gabriel; Pinello, Claudio; Kanajan, Sri; Sangiovanni-Vincentelli, Alberto L.: “A Formal Approach to Fault Tree Synthesis for the Analysis of Distributed Fault Tolerant Systems”. In: *Proceedings of the 5th ACM International Conference on Embedded Software*. EMSOFT ’05. Jersey City, NJ, USA: ACM, 2005, pp. 237–246. doi: `10.1145/1086228.1086272`.

[MZH+16] Möhrle, Felix; Zeller, Marc; Höfig, Kai; Rothfelder, Martin; Liggesmeyer, Peter: “Automating Compositional Safety Analysis Using a Failure Type Taxonomy for Component Fault Trees”. In: *26th European Safety and Reliability Conference (ESREL 2016)*. Glasgow, UK, Sept. 2016.

[NT09] Nicolás, J.; Toval, A.: “On the Generation of Requirements Specifications from Software Engineering Models: A Systematic Literature Review”. In: *Information and Software Technology* 51.9 (2009), pp. 1291–1307. doi: `10.1016/j.infsof.2009.04.001`.

[Obj15a] Object Management Group (OMG): *OMG Systems Modeling Language (OMG SysML)*. Version 1.4. 2015.

[Obj15b] Object Management Group (OMG): *OMG Unified Modeling Language (OMG UML)*. Version 2.5. 2015.

[PBF+06] Pahl, G.; Beitz, W.; Feldhusen, J.; Grote, K.-H.: *Engineering Design: A Systematic Approach*. Springer, 2006. isbn: 978-1846283185.

[PLB+10] Papadopoulos, Y.; Lonn, H.; Berntsson, L.; Johansson, Rolf; Tagliabo, F.; Torchiaro, S.; Sandberg, Anders; Walker, M.; Reiser, M.-O; Weber, M.; Chen, D.; Törngren, M.; Servat, David; Abele, A.; Stappert, F.: "Automatic allocation of safety integrity levels". In: *1st Workshop on Critical Automotive applications: Robustness & Safety, CARS*. 2010. doi: 10.1145/1772643.1772646.

[PM99] Papadopoulos, Yiannis; McDermid, John A.: "Hierarchically Performed Hazard Origin and Propagation Studies". In: *Proc. of the 18th Int. Conf. on Computer safety, reliability, and security, SAFECOMP 99*. Ed. by Felici, M.; Kanoun, K.; Pasquini, A. Vol. 1698. LNCS. Springer, 1999. doi: 10.1007/3-540-48249-0_13.

[PHS13] Priesterjahn, Claudia; Heinzemann, Christian; Schäfer, Wilhelm: "From Timed Automata to Timed Failure Propagation Graphs". In: *Proceedings of the Fourth IEEE Workshop on Self-Organizing Real-time Systems (SORT 2013)*. IEEE, 2013.

[RPS+17] Retouniotis, Athanasios; Papadopoulos, Yiannis; Sorokos, Ioannis; Parker, David; Matragkas, Nicholas; Sharvia, Septavera: "Model-Connected Safety Cases". In: *Model-Based Safety and Assessment*. Ed. by Bozzano, Marco; Papadopoulos, Yiannis. Springer International Publishing, 2017, pp. 50–63. doi: 10.1007/978-3-319-64119-5_4.

[RH08] Runeson, Per; Höst, Martin: "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (Dec. 19, 2008). doi: 10.1007/s10664-008-9102-8.

[SAE10] SAE International: *ARP4754A:2010: Guidelines for Development of Civil Aircraft and Systems*. 2010.

[Saf17] Safar, Mona: "ASIL decomposition using SMT". In: *2017 Forum on Specification and Design Languages (FDL)*. Sept. 2017. doi: 10.1109/FDL.2017.8303902.

[SCL+10] Sandberg, Anders; Chen, DeJiu; Lönn, Henrik; Johansson, Rolf; Feng, Lei; Törngren, Martin; Torchiaro, Sandra; Tavakoli-Kolagari, Ramin; Abele, Andreas: "Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2". In: *Proceedings of the 29th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2010)*. Ed. by Schoitsch, Erwin. Vienna, Austria: Springer Berlin Heidelberg, Sept. 2010, pp. 332–346. doi: 10.1007/978-3-642-15651-9_25.

[SSK14] Scharl, Adam; Stottlar, Kevin; Kady, Rani: "Functional Hazard Analysis (FHA) Methodology Tutorial". In: *International System Safety Training Symposium 2014*. St. Louis, USA, Aug. 2014.

[Sch97] SCHWABER, KEN: "SCRUM Development Process". In: *Business Object Design and Implementation: OOPSLA '95 Workshop Proceedings (16th October 1995)*. Ed. by SUTHERLAND, JEFF; CASANAVE, CORY; MILLER, JOAQUIN; PATEL, PHILIP; HOLLOWELL, GLENN. Austin, USA: Springer London, 1997, pp. 117–134. DOI: `10.1007/978-1-4471-0947-1_11`.

[She96] SHEARD, SARAH A.: "Twelve Systems Engineering Roles". In: *Proceedings of the INCOSE Sixth Annual International Symposium*. Boston, USA, 1996.

[SPB16] SOROKOS, IOANNIS; PAPADOPOULOS, YIANNIS; BOTTACI, LEONARDO: "Maintaining Safety Arguments via Automatic Allocation of Safety Requirements". In: *Proc. of the 3rd IFAC Workshop on Advanced Maintenance Engineering, Services and Technology (AMEST 2016)*. 2016. DOI: `https://doi.org/10.1016/j.ifacol.2016.11.005`.

[Tes16] TESLA MOTORS: *A Tragic Loss*. press release. June 30, 2016. URL: `https://www.tesla.com/blog/tragic-loss`.

[Tut10] TUTU, ANDREI: *Continental Emergency Steer Assist Explained*. June 24, 2010. URL: `https://www.autoevolution.com/news/continental-emergency-steer-assist-explained-21731.html`.

[VDI04] VDI: *VDI 2206: Design methodology for mechatronic systems*. Verein Deutscher Ingenieure, 2004.

[VEF+12] VOGELSANG, A.; EDER, S.; FEILKAS, M.; RATIU, D.: "Functional Viewpoint". In: *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Ed. by POHL, KLAUS; HÖNNINGER, HARALD; ACHATZ, REINHOLD; BROY, MANFRED. Springer, 2012. Chap. 5, pp. 69–83. ISBN: 978-3-642-34614-9 (Online), 978-3-642-34613-2 (Print).

[WC12] WARD, D. D.; CROZIER, S. E.: "The uses and abuses of ASIL decomposition in ISO 26262". In: *7th IET Int. Conf. on System Safety, incorporating the Cyber Security Conference 2012*. Oct. 2012, pp. 1–6. DOI: `10.1049/cp.2012.1523`.

[Wei14] WEILKIENS, T.: *Systems Engineering mit SysML/UML: Anforderungen, Analyse, Architektur.* dpunkt.verlag, 2014. ISBN: 9783864915444.

[WHL+16] WINNER, H.; HAKULI, S.; LOTZ, F.; SINGER, C., eds.: *Handbook of Driver Assistance Systems*. Springer, 2016. ISBN: 978-3-319-12351-6. DOI: `10.1007/978-3-319-12352-3`.

[YBL11] YUE, T.; BRIAND, L.; LABICHE, Y.: "A systematic review of transformation approaches between user requirements and analysis models". In: *Requirements Engineering* 16.2 (2011), pp. 75–99. DOI: `10.1007/s00766-010-0111-y`.

[ZRH16] ZELLER, MARC; RATIU, DANIEL; HÖFIG, KAI: "Towards the Adoption of Model-Based Engineering for the Development of Safety-Critical Systems in Industrial Practice". In: *5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR)*. Ed. by SKAVHAUG, AMUND; GUIOCHET, JÉRÉMIE; SCHOITSCH, ERWIN; BITSCH, FRIEDEMANN. Vol. 9923. LNCS. Springer International Publishing, Sept. 2016, pp. 322–333. DOI: `10.1007/978-3-319-45480-1\_26`.

[ZF 16] ZF FRIEDRICHSHAFEN AG: *ZF Mitigates Rear-End Collisions with New Electronic Safety Assistant for Trucks*. press release. June 29, 2016. URL: `https://press.zf.com/site/press/en_de/microsites/press/list/release/release_23367.html`.

[ZLG10] ZHANG, PENGCHENG; LI, BIXIN; GRUNSKE, LARS: "Timed Property Sequence Chart". In: *Journal of Systems and Software* 83.3 (2010), pp. 371–390. DOI: `http://dx.doi.org/10.1016/j.jss.2009.09.013`.

LIST OF ABBREVIATIONS

ADAS	Advanced Driver Assistance System, page 1
AgPL	Agriculture Performance Level, page 22
ASIL	Automotive Safety Integrity Level, page 1
bdd	SysML block definition diagram, page 29
CCF	Common Cause Failure, page 23
CF	Cascading Failure, page 22
CFT	Component Fault Tree, page 24
CNL	Controlled natural language, page 86
CONSENS	CONceptual design Specification technique for the ENgineering of complex Systems, page 28
CPS	Cyber-Physical System, page 1
DAL	Development Assurance Level, page 22
E/E	Electrics/Electronics, page 2
EBEAS	Emergency Braking & Evasion Assistance System, page 3
ECU	Electronic Control Unit, page 1
FMEA	Failure Mode and Effects Analysis, page 2
FSR	Functional Safety Requirement, page 19
FTA	Fault Tree Analysis, page 23
GSN	Goal Structuring Notation, page 37
HARA	Hazard Analysis & Risk Assessment, page 19
HW	Hardware, page 7
ibd	SysML internal block diagram, page 29

ILP	Integer Linear Program, page 116
MBRE	Model-based Requirements Engineering, page 85
MSD	Modal Sequence Diagram, page 34
NLRE	Natural Language Requirements Engineering, page 85
RDS	Rear Door System, page 152
SEooC	Safety Element out of Context, page 57
SIL	Safety Integrity Level, page 1
SUD	System under development, page 9
SW	Software, page 2
SyRS	System Requirements Specification, page 62
SysML	Systems Modeling Language, page 29
TBD	To be determined, page 45
TSR	Technical Safety Requirement, page 19
UML	Unified Modeling Language, page 9
V2V	Vehicle-to-Vehicle (communication), page 3

List of Definitions

LIST OF FIGURES

LIST OF TABLES

A

Case Study Models

This appendix contains screenshots of the models used for the case study in Chapter 7.

Case 1: Rear Door System (RDS)

Figure A.1 shows the manually specified function hierarchy of the RDS with the interfaces used for its ports.
Figure A.2 shows the Hazard CFTs that were manually specified for the RDS. The event event_RearDoorOpeningCommission represents a corresponding hazard with ASIL B. The event event_RearDoorUnlockingOmission represents a corresponding hazard with ASIL A.

Case 2: Emergency Braking & Evasion Assistance System (EBEAS)

Figure A.3 shows the manually specified function hierarchy of the EBEAS.
Figure A.4 shows the interfaces used for the function hierarchy ports.
Figure A.5 shows the Hazard CFTs that were manually specified for the EBEAS. The event event_HardBrakingOmission represents a corresponding hazard with ASIL D. The event event_SteeringOmission represents a corresponding hazard with ASIL D. The event event_EmcyBrakeWarningOmission represents a corresponding hazard with ASIL B. The event event_EvadeWarningOmission represents a corresponding hazard with ASIL B.

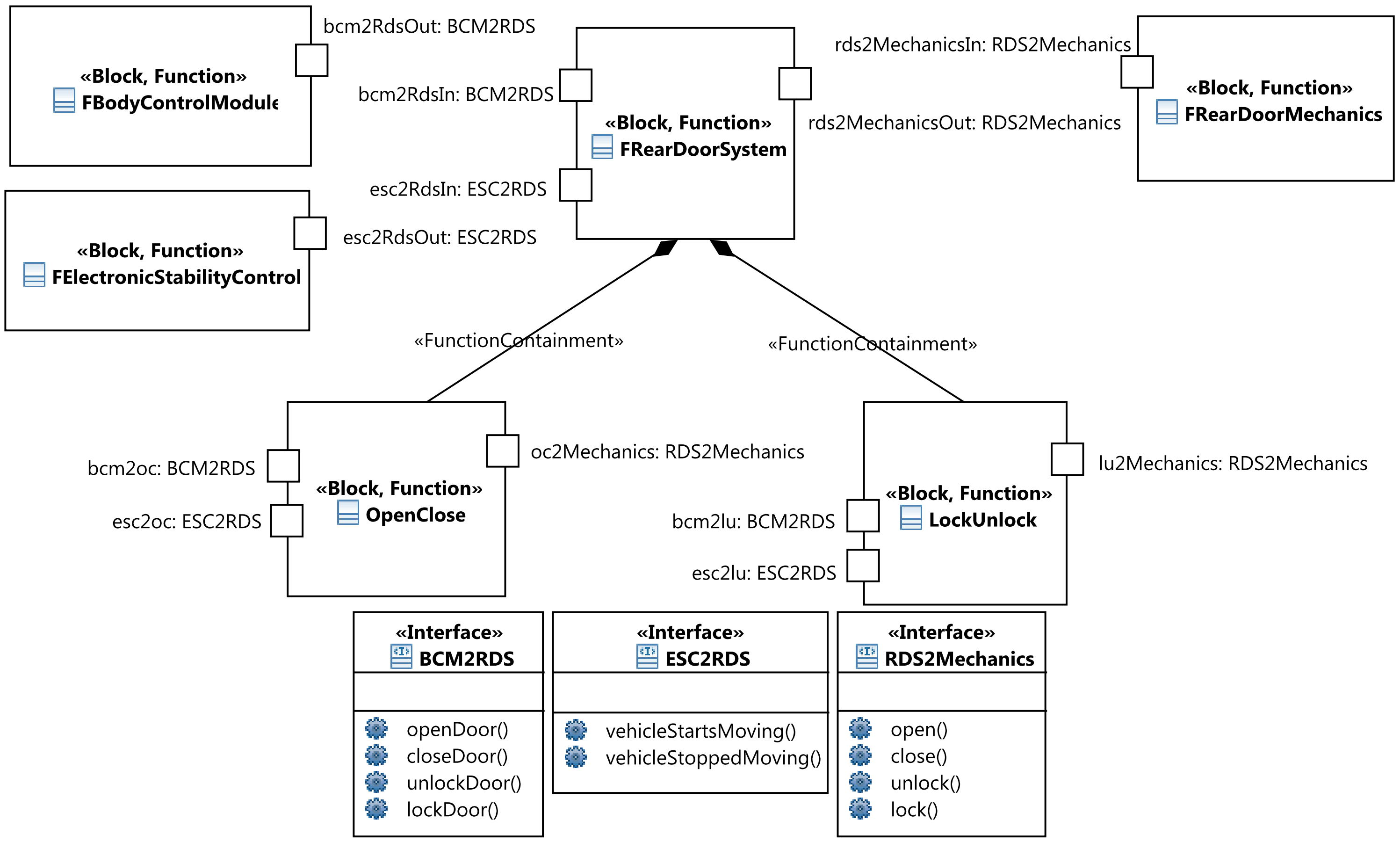


Figure A.1: RDS - Function Hierarchy

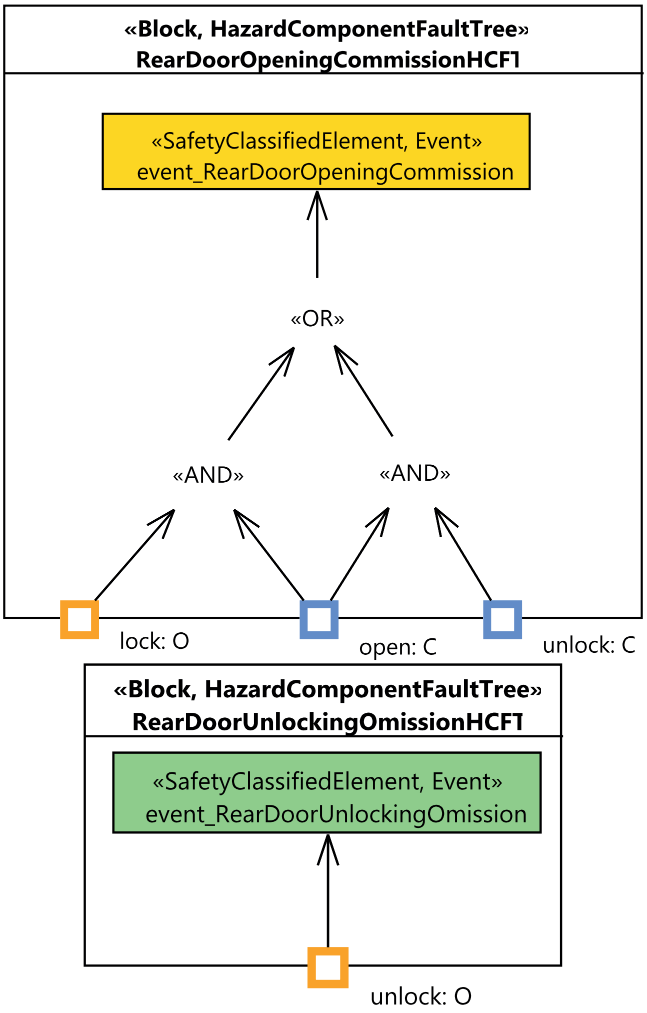

«Block, HazardComponentFaultTree»
RearDoorOpeningCommissionHCFT
«SafetyClassifiedElement, Event»
event_RearDoorOpeningCommission
«OR»
«AND»
«AND»
lock: O
open: C
unlock: C
«Block, HazardComponentFaultTree»
RearDoorUnlockingOmissionHCFT
«SafetyClassifiedElement, Event»
event_RearDoorUnlockingOmission
unlock: O

Figure A.2: RDS - Hazard CFTs

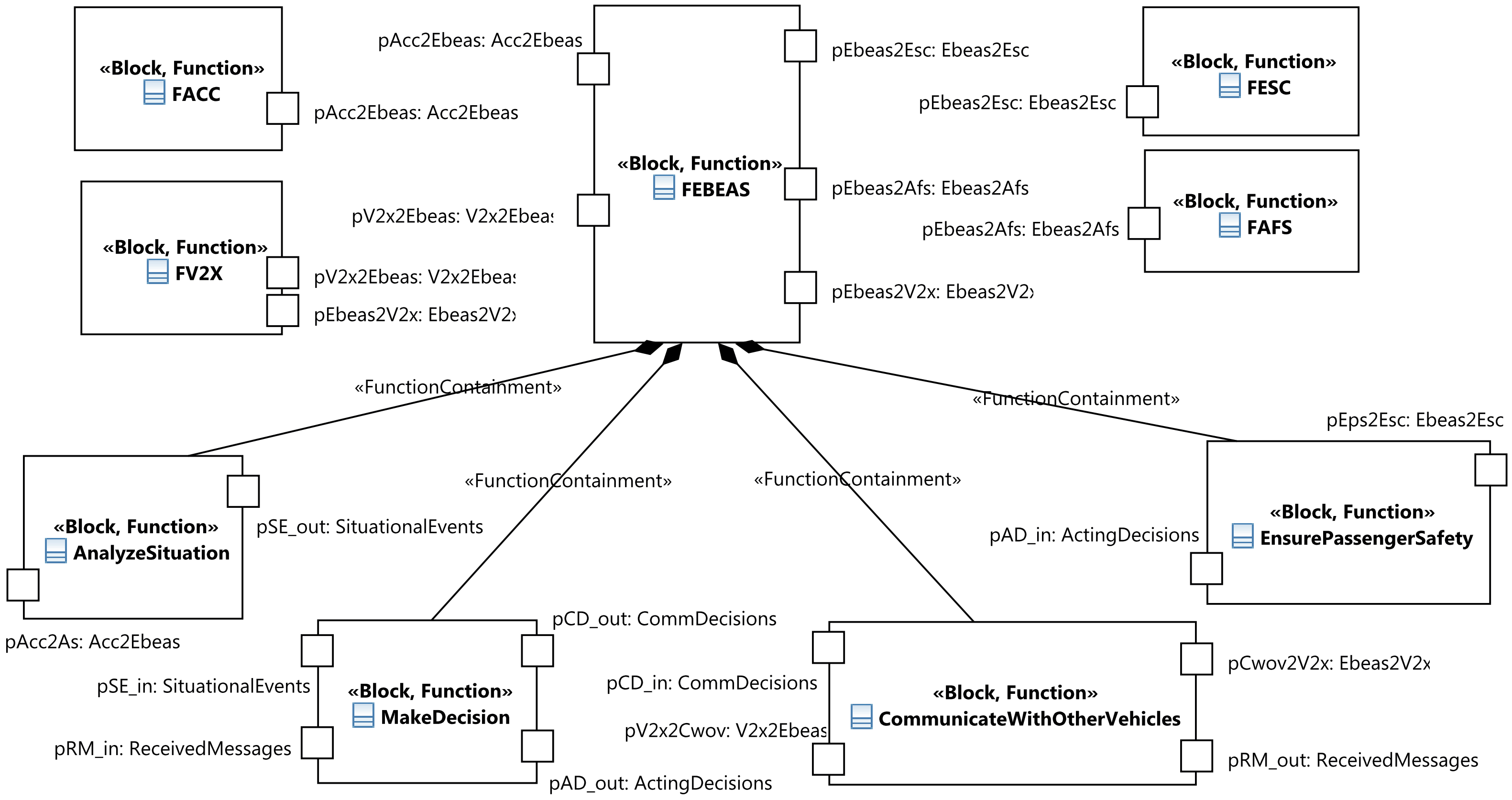


Figure A.3: EBEAS - Function Hierarchy

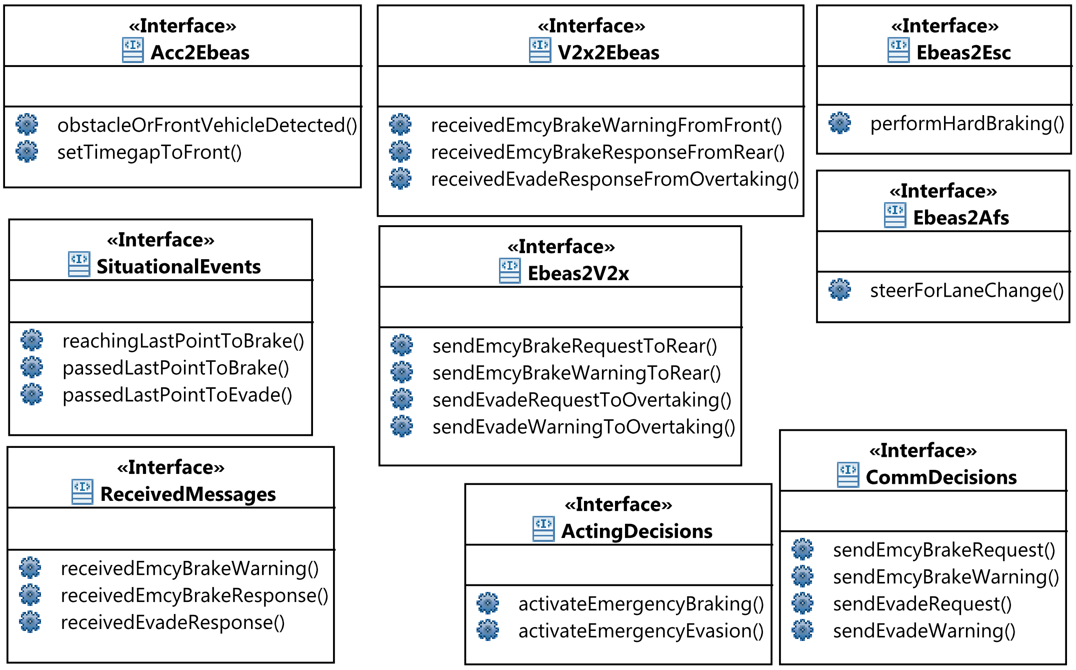


Figure A.4: EBEAS - Function interfaces

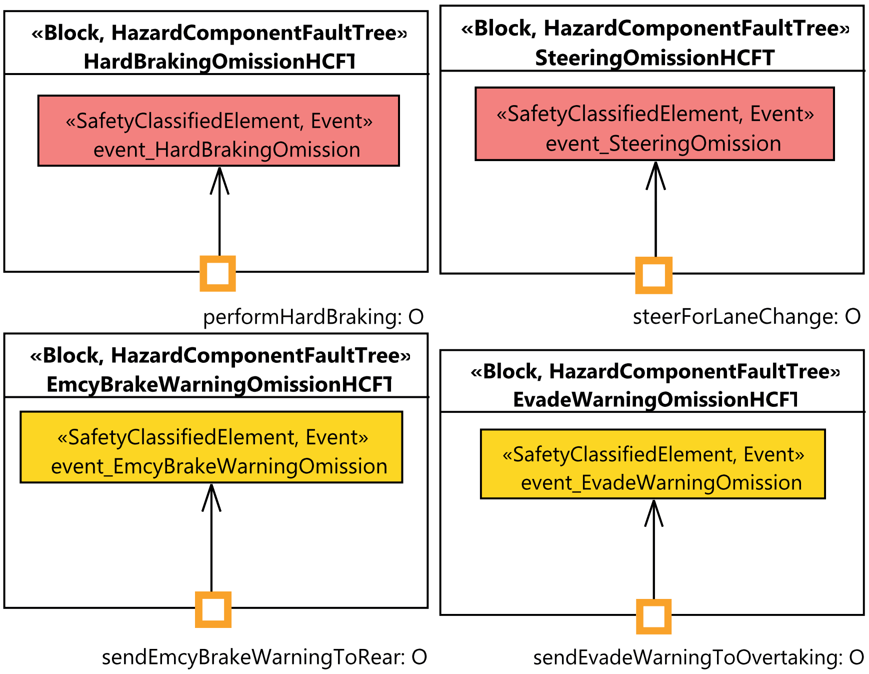

«Block, HazardComponentFaultTree»
HardBrakingOmissionHCFT
«SafetyClassifiedElement, Event»
event_HardBrakingOmission
performHardBraking: O
«Block, HazardComponentFaultTree»
SteeringOmissionHCFT
«SafetyClassifiedElement, Event»
event_SteeringOmission
steerForLaneChange: O
«Block, HazardComponentFaultTree»
EmcyBrakeWarningOmissionHCFT
«SafetyClassifiedElement, Event»
event_EmcyBrakeWarningOmission
sendEmcyBrakeWarningToRear: O
«Block, HazardComponentFaultTree»
EvadeWarningOmissionHCFT
«SafetyClassifiedElement, Event»
event_EvadeWarningOmission
sendEvadeWarningToOvertaking: O

Figure A.5: EBEAS - Hazard CFTs

B

Paper Contributions

This appendix describes my contributions to the papers of my own publications where I am not the only author.

[FHK^{+}18] and [FHK^{+}17]
I was the main author of this paper and associated technical report. I contributed to all presented patterns and to all document sections, especially the main content.

[HFK^{+}16]
This technical report consolidates the MSD syntax and semantics that this thesis works with, and introduces the EBEAS. I was one of the main authors of the EBEAS and a section that describes the usage of MSDs by example requirements on the EBEAS. In addition, I reviewed other sections of the report.

[FH15], [FH14], and [FHM14]
Jörg Holtmann and I were the main authors of these papers and jointly developed the underlying concepts of the controlled natural language and its synchronization with model-based development.

[MFH15]
As a co-author of this paper I contributed the section on related work and reviewed others.

[DFH^{+}13]
This technical report describes a requirements engineering approach integrating two existing approaches, and the application of that integrated approach to a case study. I am one of the main authors of this report, and contributed to the sections on the integrated approach and its application.

[FHH^{+}12]
This book chapter reports results of a national research project. I participated in the project and contributed to the concepts presented in Section 12.3.2. As a co-author of that section, I contributed text and reviewed several revisions.

[FHM12]
Jörg Holtmann and I were the main authors of this paper. We jointly developed the underlying concepts for semi-automatic traceability between requirements engineering and systems engineering using model-to-model transformations and constraint checks.