An approach to co-design and analysis of safety and security for three-layered system modeling : models, formalisms, and tool support

Megha Quamara

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 - Paul Sabatier (UT3 - Paul Sabatier)*

Présentée et soutenue le *14 Novembre 2022* **par :**

Megha QUAMARA

An Approach to Co-design and Analysis of Safety and Security for Three-layered System Modeling: Models, Formalisms, and Tool Support

---

JURY

M. PHILIPPE PALANQUE	Professeur d'Université	Président du Jury
MME MIREILLE BLAY-FORNARINO	Professeur d'Université	Rapporteur
M. BRUNO MONSUEZ	Professeur	Rapporteur
MME PASCALE LE GALL	Professeur d'Université	Examinateur
M. JAIR GONZALEZ	Ingénieur Chercheur	Examinateur
M. BRAHIM HAMID	Professeur d'Université	Directeur de thèse
M. GABRIEL PEDROZA	Ingénieur Chercheur	Co-directeur de thèse

École doctorale et spécialité :
MITT : Informatique et Télécommunications

Unité de Recherche :
IRIT - Institut de Recherche en Informatique de Toulouse (UMR 5505)

Directeur(s) de Thèse :
Brahim HAMID et *Gabriel PEDROZA*

Rapporteurs :
Mireille BLAY-FORNARINO et *Bruno MONSUEZ*

Dedicated to my Supervisors, Beloved Family, and Almighty God

Abstract

The paradigm shift from traditional Industrial Control Systems (ICSs) to software-based systems with intertwined physical elements, like Cyber-Physical Systems (CPSs), has substantially raised the degree of automation, inter-connectivity, and rigor of the requirements. An increasing number of features and functionality driven by interaction among the system entities leads to higher complexity by design. Deploying such systems in critical domains like automotive entails integrating safety and security concerns during the System Engineering (SE) process in light of their importance and mutual influence. Specifically, the associated risks must be anticipated and treated in unison in the early system conception stages, e.g., architectural design, to reduce the likelihood of harm, additional costs, and complexities encountered in identifying dependencies like conflicts and their treatment as an afterthought.

Despite the vast literature addressing the incorporation of safety and security concerns during the SE process, the present system development landscape often exhibits a standalone viewpoint of the two disciplines, owing to the fundamental differences, like the origin of risks. There is often a lack of explicit modeling techniques that can encourage the formal reasoning of inter-dependencies between safety and security in the early design stages. An amalgamation of the benefits of both worlds seems to be difficult due to the complexity associated with understanding and executing formal-based techniques. Moreover, existing works are limited in terms of guidance for non-savvy engineers to facilitate integration and verification of stringent safety constraints and security exigencies. Methodological support in this regard is lacking, consolidating tools and techniques belonging to the system, safety, and security disciplines.

Considering the problematics above, we propose a joint design and analysis approach and tooled framework to better support and ease system safety and security co-engineering. Accordingly, the specific contributions of this work are many-fold: 1) a method to build a safety and security modeling framework, supporting the system safety and security co-engineering process, 2) a three-layered system model for capturing the aspects pertaining to the different stages of system development, 3) a modeling language and design framework, supporting the system architecture and safety and security properties modeling in integration in the context of the three-layered system architecture, 4) a formal-based rigorous specification of the system and properties to be verified, and 5) an operational tool-chain support for facilitating different phases of the approach.

The approach follows an iterative process covering several stages of the system development, and that relies upon existing design and analysis techniques for incorporating a positive vision of safety and security concerns, particularly objectives encoded as properties. It incorporates Model-Driven Engineering (MDE) and formal-based techniques to construct a set of Domain-Specific Modeling Languages (DSMLs) and their corresponding formalisms for specifying and analyzing the system architecture and properties models, capturing and allowing the reuse of the respective domain expertise

and safety and security interplay. The outcomes comprise artefacts corresponding to the design and analysis activities. This broadly includes the following steps: 1) modeling of the system aspects and safety and security properties, 2) formalization of both system and properties models via interpretation into a tooled-formal language, and 3) verification of properties and interplay. Furthermore, the approach strives for a tool-chain leveraging the existing MDE platforms and formal-based techniques to support hierarchical modeling, precise specification, formal interpretation, and verification of safety and security properties in unison regarding different granular representations of the target System Under Design (SUD).

For validation purposes of our work, we consider a three-layered system model, targeting high-level mission, functional, and detailed component-based representations of the target SUD. Moreover, we consider a set of representative properties belonging to the Availability, Integrity, Freshness, and Controlled Accessibility categories that correspond to the safety and security concerns associated with the system model. The system and properties DSMLs are implemented using meta-models for the abstract syntax and profiles for the concrete syntax to provide a standardized modeling environment, relying upon the existing Unified Modeling Language (UML). Furthermore, a logical specification of the system, safety, and security properties is proposed using an abstract system model (i.e., a technology-independent specification language) relying upon First-Order Logic (FOL) and Modal Logic, followed by a more concrete specification of the system model and the properties relying upon a suitable language with automated-tool support, namely Event-B. As part of the assistance for developing safe and secure system architectures, we propose a tool-chain comprising Eclipse Papyrus as a modeling framework and Rodin as a formal-based tool for verification. The approach illustration and assessment are conducted via a Connected-Driving Vehicles (CDVs) use case in the automotive domain, targeting a safety- and security-critical converging road plan scenario.

Keywords: Safety, Security, Co-engineering, Design and Analysis, Model-based Development, Formal Techniques

Résumé

Le changement de paradigme des systèmes de contrôle industriels (SCI) traditionnels vers des systèmes basés sur des logiciels avec des éléments physiques entrelacés, comme les systèmes cyber-physiques (CPS), a considérablement augmenté le degré d'automatisation, d'interconnexion et de rigueur des exigences. Un nombre croissant de caractéristiques et de fonctionnalités découlant de l'interaction entre les entités du système entraîne une complexité accrue par conception. Le déploiement de tels systèmes dans des domaines critiques tels que l'automobile implique l'intégration des questions de sûreté et de sécurité au cours du processus d'ingénierie des systèmes (SE), compte tenu de leur importance et de leur influence mutuelle. Plus précisément, les risques associés doivent être anticipés et traités à l'unisson dès les premières étapes de la conception du système, par exemple la conception architecturale, afin de réduire la probabilité de dommages, les coûts supplémentaires et les complexités rencontrées dans l'identification des dépendances comme les conflits et leur traitement après coup.

Malgré la vaste littérature traitant de l'incorporation des préoccupations de sûreté et de sécurité au cours du processus de SE, le paysage actuel du développement de systèmes présente souvent un point de vue autonome des deux disciplines, en raison des différences fondamentales, comme l'origine des risques. Il y a souvent un manque de techniques de modélisation explicites qui peuvent encourager le raisonnement formel des interdépendances entre la sûreté et la sécurité dans les premières étapes de la conception. Il semble difficile d'amalgamer les avantages des deux mondes en raison de la complexité associée à la compréhension et à l'exécution des techniques formelles. En outre, les travaux existants sont limités en termes d'orientation des ingénieurs non avertis pour faciliter l'intégration et la vérification des contraintes de sécurité strictes et des exigences de sécurité. Il manque un support méthodologique à cet égard, consolidant les outils et les techniques appartenant aux disciplines des systèmes, de la sûreté et de la sécurité.

Compte tenu de la problématique ci-dessus, nous proposons une approche de conception et d'analyse conjointe ainsi qu'un cadre outillé pour mieux soutenir et faciliter la co-ingénierie de la sécurité et de la sûreté des systèmes. En conséquence, les contributions spécifiques de ce travail sont multiples : 1) une méthode pour construire un cadre de modélisation de la sécurité et de la sûreté, soutenant le processus de co-ingénierie de la sécurité et de la sûreté des systèmes, 2) un modèle de système à trois couches pour capturer les aspects relatifs aux différentes étapes du développement du système, 3) un langage de modélisation et un cadre de conception, soutenant l'architecture du système et la modélisation des propriétés de sécurité et de sûreté en intégration dans le contexte de l'architecture du système à trois couches, 4) une spécification rigoureuse basée sur le formalisme du système et des propriétés à vérifier, et 5) une chaîne d'outils opérationnels pour faciliter les différentes phases de l'approche.

L'approche suit un processus itératif couvrant plusieurs étapes du développement du système, et qui

s'appuie sur les techniques de conception et d'analyse existantes pour incorporer une vision positive des préoccupations de sûreté et de sécurité, en particulier les objectifs codés en tant que propriétés. Il intègre l'ingénierie dirigée par les modèles (MDE) et des techniques formelles pour construire un ensemble de langages de modélisation spécifiques au domaine (DSML) et leurs formalismes correspondants pour spécifier et analyser l'architecture du système et les modèles de propriétés, en capturant et en permettant la réutilisation de l'expertise respective du domaine et de l'interaction entre la sécurité et la sûreté. Les résultats comprennent les artefacts correspondant aux activités de conception et d'analyse. Ces activités comprennent en général les étapes suivantes 1) la modélisation des aspects du système et des propriétés de sûreté et de sécurité, 2) la formalisation des modèles de système et de propriétés via l'interprétation dans un langage formel outillé, et 3) la vérification des propriétés et de l'interaction. En outre, l'approche vise à mettre en place une chaîne d'outils exploitant les plates-formes MDE existantes et les techniques formelles pour prendre en charge la modélisation hiérarchique, la spécification précise, l'interprétation formelle et la vérification des propriétés de sûreté et de sécurité à l'unisson en ce qui concerne les différentes représentations granulaires du système cible en cours de conception (SUD).

Aux fins de validation de notre travail, nous considérons un modèle de système à trois couches, ciblant les représentations de haut niveau de la mission, de la fonction et des composants détaillés du SUD cible. De plus, nous considérons un ensemble de propriétés représentatives appartenant aux catégories Disponibilité, Intégrité, Fraîcheur et Accessibilité contrôlée qui correspondent aux préoccupations de sûreté et de sécurité associées au modèle de système. Les DSML du système et des propriétés sont mis en œuvre à l'aide de méta-modèles pour la syntaxe abstraite et de profils pour la syntaxe concrète afin de fournir un environnement de modélisation standardisé, en s'appuyant sur le langage de modélisation unifié (UML) existant. En outre, une spécification logique des propriétés du système, de la sûreté et de la sécurité est proposée à l'aide d'un modèle de système abstrait (c'est-à-dire un langage de spécification indépendant de la technologie) reposant sur la logique du premier ordre (FOL) et la logique modale, suivie d'une spécification plus concrète du modèle de système et des propriétés reposant sur un langage approprié avec un support d'outils automatisés, à savoir Event-B. Dans le cadre de l'assistance au développement d'architectures de systèmes sûrs et sécurisés, nous proposons une chaîne d'outils comprenant Eclipse Papyrus comme cadre de modélisation et Rodin comme outil de vérification basé sur la formalisation. L'illustration et l'évaluation de l'approche sont réalisées par le biais d'un cas d'utilisation de véhicules à conduite connectée (CDV) dans le domaine automobile, ciblant un scénario de plan routier convergent critique pour la sécurité et la sûreté.

Mots clés: Sûreté, Sécurité, Co-ingénierie, Conception et Analyse, Ingénierie des Modèles, Techniques Formelles

Acknowledgements

Completing my doctorate was a rewarding journey—one that nurtured me over the last few years to be more focused, resilient, and systematic. With these lines, I would like to acknowledge all who became part of this journey. Their impact is far more significant than a purely academic one.

First of all, I would like to express my immense gratitude to my thesis director, Professor Brahim Hamid, for giving me an opportunity to pursue this work under his guidance. His enormous support and encouragement have been invaluable throughout this journey. His overall insights have made this an inspiring experience for me and laid the basis for my scientific eagerness that I will carry along in the future. This endeavor would not have been possible without the supervision of my thesis co-director, Dr. Gabriel Pedroza. I cannot express my gratitude towards him in words. I genuinely treasure every discussion I had with him for grooming me into a better person. I will never forget his humility and politeness while he was inculcating thought-provoking suggestions by putting in "my two cents."

I am also much obliged to all the members of the jury for their time and participation in my defense. I thank Professor Mireille Blay-Fornarino and Professor Bruno Monsuez for accepting to be the reviewers for this thesis. The feedback they offered has been vital for its improvement. I also thank Professor Philippe Palanque for agreeing to chair the jury for my defense. I also thank Professor Pascale Le Gall and Dr. Jair Gonzalez for accepting to be the examiners. The questions and comments during the defense process turned out to be a chance for several perspectives.

I am also grateful to Dr. Stéphane Salmons (Head of LECS) and Dalila Guessoum (Former Head of LECS) for their support and encouragement in different phases of my Ph.D. journey. Many thanks to Frédérique and Ewa for their time and help throughout this project. I would like to extend my sincere thanks to Dr. Kunal Suri, for his invaluable advice and unwavering personal support and belief in me. A special thanks to fellow researchers in both labs for creating a fun and lively environment.

Last but not least, my heartfelt thanks go to my family. I will always be grateful to my mother, Chander Kanta Quamara, for all her contributions and sacrifices for me to lead my choices. I thank my father, Jitendra Kumar Quamara, for his unconditional support and for sharing his research experiences to encourage me. I hold gratitude to my brother, Sidharth Quamara, for his kindness and belief in me. I also appreciate my aunts, Vimal and Neelam, and my uncle, Adarsh Dhamija, for their motivation, love, and emotional support throughout this journey. Thanks to my friends for being all ears to all my problems.

Merci beaucoup à tous!

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Contents

1.1 Research Context

The paradigm shift from traditional Industrial Control Systems (ICSs) to software-based systems with intertwined physical elements, such as embedded systems and Cyber-Physical Systems (CPSs), has substantially raised the degree of automation, inter-connectivity, and rigor of the requirements. Consequently, an increasing number of features and functionality driven by the seamless interaction among the system entities lead to higher complexity by design. Typically, these systems demand greater security due to the risks posed by the underlying technologies like communication and control and their implication on the information and physical assets. Moreover, the systems with design flaws are not merely inadequate in terms of their operational capabilities but also unsafe and vulnerable to security attacks. Indeed, the impact of deploying such systems in domains like automotive [1], aeronautics [2], and public infrastructure [3] can be critical regarding the business, economic, and safety criteria, and, in the end, can potentially endanger the human lives. This entails the integration of safety and security concerns during the *System Engineering (SE)* process.

In literature, there exist approaches facilitating the incorporation of safety [4] and security concerns [5] during the SE process. However, they often exhibit a standalone viewpoint of

the two disciplines without focusing on their mutual influence. This is evident for systems deployed in scenarios that are purely safety-critical, such as healthcare, or security-critical, such as e-voting and finance. Nevertheless, with software-based systems exhibiting automation and communication capabilities, safety and security cannot be seen in isolation. For instance, in the automotive domain, the remote access feature offered by the connected cars provides attack surfaces like communication channels and onboard components for potential manipulation of the safety-critical functionality, e.g., jamming of brakes [6]. Specifically, the associated risks must be anticipated and identified in unison in the early stages, like system architectural design, to reduce the likelihood of harm, additional costs, and complexities encountered in identifying inter-dependencies, e.g., conflicts, and their treatment in the later stages [7].

1.1.1 Multi-layered System Model

To make the complexity of engineered systems potentially addressable, model layers can be developed according to the different stages of the SE process. The resulting multi-layered system model allows comprehending, expressing, and explaining different viewpoints [8], e.g., high-level mission or teleological, functional, and detailed component architecture, wherein the involved stakeholders like system domain experts and architects can make design choices and decisions relying upon the layered representation of the target System Under Design (SUD).

The *mission* is primarily a problem-oriented, and often, context-oriented concept. It is multi-paradigmatic, encompassing goal, operation, object, and agent views that allow for the amalgamation of the system functionality with the architectural design process [9]. When defined diligently, it facilitates the engineering process of complex systems and comprehends the interactions among the underlying entities. This, in turn, is useful for creating the design and software implementation of the system behavior and delivering utility to the end-users [8]. Discrete operational requirements form the basis for representing system actions and corresponding responses at the mission layer representation of the system. These requirements are collected in the form of operational scenarios having the following related aspects [10]: 1) pre-conditions and post-conditions associated with the system operations, 2) system's interaction with the environment, and 3) interests and concerns of the stakeholders regarding the system operations.

The *functional* representation allows decoupling the system's functionality from the way of its implementation, i.e., operations [11]. The corresponding system model captures system functions and their interactions based upon the mission objectives for their fulfillment. In general, more than one functional architecture can satisfy the underlying requirements towards achieving the desired goal of the system [12]. At the design stage, it determines what the system

does and, thus, assists in identifying the appropriate and optimal component-based architecture for the underlying system. A function can be represented as a schema that contains a reference to the behavior for the accomplishment of the function and demonstrates the conditions that allow the behavior to achieve the function [13].

The *component*-based representation captures the detailed structure of the system, typically the system components and communication channels, to meet the desired functions and deliver the expected behavior. A component can be described as a self-contained entity that may further have sub-components and constitute the structural part of the system [14]. The engineered systems incorporate complex and seamless interaction between the components, which can be physical or logical, comprising networking, software, and computational elements [15]. This interaction is highly context-dependent as the system structure may change with time, where components and connections between them may be added, removed, or updated.

Developing safety- and security-related systems requires integrating safety and security analysis techniques in the SE—System Engineering process. In the safety context, for instance, the safety expert takes as input the missions and goals of the system, along with the specific context in which the system will operate, and provides a Preliminary Hazard Analysis (PHA) [16] as an outcome, mainly guided by a list of feared events. Herein, the feared events are defined considering the undesirable circumstances the system may face, including operational conditions and system failures, determined via Hazard and Operability Analysis (HAZOP) [17]—for instance. Likewise, a safety expert can conduct a Functional Hazard Analysis (FHA) [18] or Failure Mode and Effects Analysis (FMEA) [19] regarding the different sub-systems having dedicated functionalities, independently of the PHA or after. In the latter case, the FHA/FMEA inherits the already-defined list of feared events. Last yet not least, the safety expert can conduct a fault tree analysis [19] at the component level, which FHA/FMEA may follow. It inherits the impact of the feared events defined in PHA or FHA/FMEA. Likewise, in the security context, a consistent analysis, e.g., attack tree analysis or identification of misuse cases of the related concerns [20], often demands a detailed view of the system architecture, involving vulnerable technologies, insecure ports, and unprotected channels—for instance. The associated feared events may propagate to the sub-system or system level and impact its trustworthiness.

Nevertheless, to the best of our knowledge, the state-of-the-art approaches, e.g., [4, 8], have not addressed incorporating the above safety and security analysis techniques in the SE process regarding the multi-layered system representation (see Figure 1.1). The role of the techniques can be emphasized at different system representation levels, depending upon the design details they offer. However, this often leaves a gap whereby the inter-dependencies between safety and security concerns cannot be analyzed at the same level. Moreover, it is evident from the existing approaches that the system development process from the safety perspective is relatively

mature, owing to the extensive range of standards that provide the basis for the respective concerns' analysis. However, this is not the case with security, which demands consolidating our understanding of the underlying aspects.

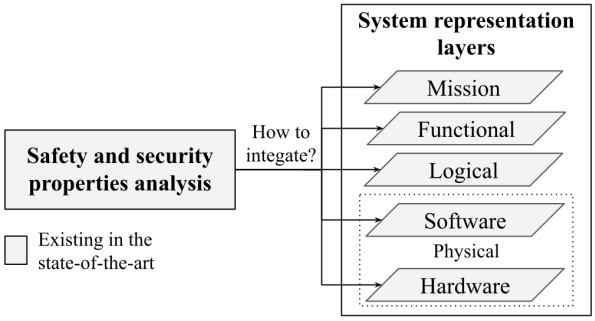


Figure 1.1: Integration of Safety and Security Properties Analysis in the System Development.

The multi-layered system model can facilitate the integration of the safety and security analysis into the system development stages, whereby validation and verification of corresponding concerns can be conducted based on the granularity adapted to the analysis. In addition, relationships can be derived from the impact of concerns at a given layer or across layers, within or between the safety and security domains. The state-of-the-art analysis in this thesis will mainly cover the approaches regarding safety and security properties, in particular, to be incorporated across different modeled representations of the system without explicit associations.

1.1.2 Safety and Security Properties

Safety and security concepts are defined differently and are sometimes used interchangeably [21]. Safety emphasizes the protection from accidental flaws or mistakes that harm the acquired value. On the other hand, security is defined as protecting acquired value from intentional actions taken by the human elements. Acquired value, also known as the object of protection, includes system-related aspects that we value, e.g., system, software or hardware components, transmission channels, data, human lives, etc. The engineering process of safe and secure systems calls for a deep understanding of the underlying requirements to avoid the potential impact of related feared events, e.g., faults, failures, hazards, threats, or attacks. These requirements are defined on top of the functional ones and typically reflect the non-functional *properties* of the system [22, 23]. The set of properties may vary depending on the nature and complexity of the target of protection. In addition, the elementary nature of properties can be qualitative or quantitative [24], temporal or spatial [25], and these may be system or

environment-specific [26]. These can also be internal to the system or associated with the communication interface [27].

In literature, safety is viewed as an attribute of dependability [28]. While security is a composite notion comprising the Confidentiality, Integrity, and Availability (CIA) triad and considered at the same level as dependability [28, 29]. In our context, we are not concerned with such classification. We consider safety and security as the ultimate goals to be achieved by the target SUD as a whole, and these goals can be built upon or influenced by a range of properties. Specifically, in this work, we consider the following list of properties as the representative ones:

- *Availability:* Capability of a product (e.g., system, information, component) to provide a stated function if demanded, under given conditions over its defined lifetime [30].
- *Integrity:* Protection of accuracy and completeness of information or data [31].
- *Freshness:* Non-duplication and temporal validity (i.e., recentness) of the received data or information [32, 33].
- *Controlled Accessibility:* Ability to limit, control, and determine the level of access that entities have to a system, function, information, and component [34, 32].

Figure 1.2 shows an excerpt for a better understanding of how the above properties will be used in our work regarding several system-related aspects that can correspond to the different layers of the multi-layered system representation. Herein, it requires a complex engineering process to ensure consistent passage of the semantics concerning the system and the properties across the layered representations and allow properties' preservation.

Layered representations	**Safety properties**	**Security properties**
Mission layer	Operational availability	System availability
Functional layer	Functional path availability	Functional availability
	Functional integrity	Functional integrity
	Functional data freshness	Function accessibility
Component layer	Component availability	Data integrity
	Connector availability	Message availability
	Data integrity	Message freshness
	Message freshness	Component accessibility

Figure 1.2: Safety and Security Properties: A Correspondence with the Layered System Representations.

1.1.3 Model-based System Development

Model-based system development is an approach to address the growing complexity of modern technical systems based upon time-proven engineering principles, such as the use of abstraction, separation of concerns, step-wise refinement and iteration, and the use of formal methods [35]. It involves computing-based automation techniques to generate the document artefacts to models and ensure consistency. Models are used to denote the abstract representation of the actual system to be engineered. They provide input and output at all the stages of the system development, e.g., design and implementation, until the final system is built.

Model-Driven Engineering (MDE). Model-Driven Engineering (MDE) typically captures the software development approaches that aim to reduce the gaps between problem specification and software implementation using technologies for transforming abstract definitions to executable code [36]. It can traverse knowledge across diverse technical domains via the systematic use of models as first-class elements. Two pre-eminent aspects of MDE are *meta-modeling* and *model transformation*. These aspects play a crucial role, especially when different levels of knowledge formalization are involved [37]. Meta-models are extensively used in defining Domain-Specific Modeling Languages (DSMLs) to describe models for a domain. *Meta-modeling* endows a language with essential concepts and their relationships through abstract notations, thus segregating the abstract syntax from the language's concrete syntax [38]. This assists the system designers in alleviating the complexity of effectively capturing the domain concepts, e.g., safety and security, of the target SUD in separation from the rest of the system-related aspects. *Model transformation* involves the conversion of a source model into a target model using a model transformation language under the influence of some well-defined transformation rules. The transformation is highly dependent on the nature of the source and target models, along with the levels of abstraction.

The last few years have witnessed significant use of MDE for software development in safety [39] and security-related [40] systems. Unified Modeling Language (UML) supports the MDE approach and allows system architects and developers to specify, visualize, construct, and document system software [41]. Being a general-purpose modeling language, it can be adapted using the notion of UML profiles to specify specific domains, e.g., safety and security, to develop large and complex system software. Automated tool support enables checking the model regarding syntactic consistency and completeness. Among all, Papyrus [42] is an Eclipse-based open-source tool that uses the notion of UML profiles for graphical modeling of system requirements.

Formal Techniques. Formal methods involve using techniques for mathematically analyzing the description of a system, including its properties, e.g., safety and security, via the application of formal specification languages. Formal specification involves using precise and unambiguous mathematical concepts, like logic and algebra, with added semantic rules that foster reasoning and verification of properties [43]. For example, First-Order Logic (FOL) [44] describes the system's state in the form of pre-conditions and post-conditions, and Modal logic [45] considers the interpretation of formula within a defined context using modal operators, e.g., possibility and necessity. Likewise, Event-B [46] is a formal language based upon set theory notions for specifying and analyzing systems by modeling them as abstract state machines. Formal-based tools, e.g., Rodin [47] and Alloy analyzer [48], are particularly useful in designing and verifying high-end industrial systems. They broadly cover 1) automated model checkers that rely upon algorithms for exhaustively verifying the desired properties of the system, given a model's state space, and 2) theorem provers (also proof assistants) that involve the use of human expertise for guiding the proof of correctness [49].

1.2 Problematics

Within the above-mentioned context, our overall research problem is: *How to address the interplay between safety and security during the design stages of the SE—System Engineering process?*

Specifically, we aim to address the following methodological issues in this regard:

- *Problematic* **P1:** Despite the existing methodologies, techniques, and tools in the system development realm, a lack of methodological support exists for a joint analysis reconciling safety and security expertise during the early design stages of the SE process.
- *Problematic* **P2:** In the design phase, the requirements are broken down from the high-level teleological representation to the detailed technical architecture of the target SUD—System Under Design. However, this process is often complex and lacks consistent semantic transfer across different granular representations of the system, along with the incorporation of safety and security properties.
- *Problematic* **P3:** There exists a lack of modeling language and semantics, consolidating diverse knowledge from the discipline of system, safety, and security engineering, and to effectively express the domain concepts in an integrated fashion.
- *Problematic* **P4:** Conducting design-level safety and security analysis to increase system design trustworthiness can be error-prone due to ambiguous properties specifications

or biases introduced by non-savvy engineer's interpretation. This incorporation may also lead to potential inter-dependencies like conflicts between the properties due to the absence of precise semantics.

- *Problematic* **P5:** The present landscape of safety and security development from a standalone perspective is quite complex. Existing works are limited in terms of guidance for non-savvy engineers like system architects and analysts involved in the SE process, lacking the expertise to conduct integration and verification of stringent safety constraints and security exigencies. In this regard, there is a lack of automated tool support for integrated analysis of the system safety and security properties.

1.3 Thesis Contributions

To address the problematics above, this work strives for a joint design and analysis approach and tooled framework to better support and ease system safety and security co-engineering. Accordingly, the specific contributions of this thesis are five-fold:

- *Contribution* **C1:** A method to build a safety and security modeling framework, supporting the system safety and security co-engineering process. This involves developing a reusable formal model library of safety and security properties signatures to be later incorporated into the SE process for verifying conformity with the underlying safety and security requirements.

- *Contribution* **C2:** A three-layered system model for capturing the aspects pertaining to different stages of system development, viz. *Mission*, *Functional*, and *Component* architecture, along with the passage of semantics across the layers.

- *Contribution* **C3:** A modeling language and design framework, supporting the system architecture and safety and security properties modeling in integration. To this end, we propose a set of DSMLs to model the target SUD at different stages of system development in the context of the three-layered system model and incorporate safety and security properties across each layer. Herein, we use UML to create system and properties meta-models and profiles.

- *Contribution* **C4:** The formal-based rigorous specification of the modeling language and design framework, supporting the specification of the system and safety and security properties to be verified. To this end, we define logical specification and semantics of the modeled system and properties to facilitate model interpretation and delegate properties

analysis to tooled-formal languages and frameworks. Herein, we use set theory, FOL, and Modal Logic as technology-independent formalisms to ensure the well-formedness of the system models and for the specification of dependencies like conflicts between the properties. Also, we use the Event-B tooled-formal method in early modeling phases to obtain a concrete specification relying upon the formalisms and ensure that once deployed, the system's operation conforms to the properties.

- *Contribution* **C5:** An operational tool-chain support integrating MDE and formal-based techniques for facilitating different phases of the proposed approach, including hierarchical modeling, precise specification, formal interpretation, and verification of safety and security properties in unison. Herein, we use Eclipse Papyrus as a modeling framework and Rodin as a formal-based verification tool.

Thesis Approach Synopsis. Initially, we investigate the state-of-the-art approaches for incorporating safety and security analysis in the SE process. Accordingly, features are defined to review these approaches from both standalone and co-engineering viewpoints. This is followed by examining the approaches for specifying and verifying the safety and security properties of the target SUD to conduct design and analysis. The lack and need for methodological support to an end-user, e.g., system designer, architect, and analyst, are thus identified. Accordingly, an approach is proposed in the context of the problematics defined in Section 1.2, relying upon the existing MDE and formal-based techniques for safety and security integrated design and analysis. The specific contributions are made in the scope of the proposed approach. This is accompanied by tool-chain support that endows an environment for verifying properties and addressing the engineers' needs. Moreover, the approach's applicability is demonstrated with a use case from the automotive domain. Some of the outcomes of this work are published in [50, 51, 52, 53].

1.4 Publications

A concise summary of the publications related to the research contributions presented in this manuscript is provided as follows:

- **Megha Quamara**, Gabriel Pedroza, and Brahim Hamid. Introducing a multi-layered model-based design approach towards safety-security co-engineering. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 1163-1164. IEEE, 2021.

DOI: 10.1109/QRS-C55045.2021.00175

Summary: In this paper, we introduced a multi-layered model-based integrated design and analysis approach for safety and security co-engineering. The approach seeks to leverage existing techniques like MDE and formal methods to facilitate specification and verification of safety and security properties in unison that can be further specialized across different representations, i.e., mission, functional, and component, of a target SUD.

- **Megha Quamara**, Gabriel Pedroza, and Brahim Hamid. Multi-layered model-based design approach towards system safety and security co-engineering. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 274-283. IEEE, 2021.

  DOI: 10.1109/MODELS-C53483.2021.00048

  Summary: In this paper, we provided an instantiation of the proposed approach for a mission-centric system representation. We offered tool-chain support integrating MDE techniques, namely Eclipse Papyrus, and a formal-based tool, namely Rodin, to conduct verification, spot inconsistencies, and ensure design conformity concerning the safety and security properties. The overall approach is validated based upon a Connected-Driving Vehicles (CDVs) use case.

- **Megha Quamara**, Gabriel Pedroza, and Brahim Hamid. Facilitating safety and security co-design and formal analysis in multi-layered system modeling. In *20th IEEE International Conference on Dependable, Autonomic & Secure Computing (DASC)*, pages 1-8. IEEE, 2022.

  DOI: 10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9927773

  Summary: In this paper, we emphasized the modeling aspects of the proposed approach, pursuing joint system safety and security design. To this end, we proposed a set of DSMLs for the three-layered system modeling and for safety and security properties modeling to incorporate them across different system representation layers. Herein, we used UML and its profiles. Moreover, a model interpretation is defined to map system and property models into Event-B specifications to conduct automated verification of safety and security objectives' signatures and conflict solving. For illustration purposes, we considered the use case of CDVs.

- **Megha Quamara**, Gabriel Pedroza, and Brahim Hamid. Formal analysis approach for multi-layered system safety and security co-engineering. In *European Dependable*

Computing Conference (EDCC) Workshop: SERENE, pages 18-31. Springer, Cham, 2022.

DOI: 10.1007/978-3-031-16245-9_2

Summary: In this paper, we emphasized the formalization aspects of the proposed approach, facilitating joint analysis of safety and security objectives, specialize-able across different system views. As a prerequisite, we defined interpretation rules for mapping the modeling concepts to their formal-based counterparts relying upon mathematical logic, namely FOL and Modal Logic. Moreover, we used Event-B, to obtain a more concrete specification of the system and properties conceptual model and the accompanying formal-based tool, namely Rodin, to mechanize properties verification and spot inconsistencies in early modeling phases. The approach is illustrated via the CDVs use case.

1.5 Thesis Outline

The rest of this manuscript is organized into the following chapters:

In **Chapter 2**, we present a systematic review of the state-of-the-art approaches to safety and security co-engineering concerning the problematics addressed in this research work. Specifically, we focus on existing contributions along two principal axes: 1) incorporating safety and security in the SE process from standalone and combined perspectives and 2) designing and analyzing safety and security via specification and verification methodologies. Based on the extraction of the features they exhibit, a lack of methodological support for an integrated design and analysis of safety and security is observed, which provides a rationale for the contributions of this work.

In **Chapter 3**, we provide an overview of our approach that aims to tackle the identified lack towards effective safety and security co-engineering. One of the main contributions consists in the provisioning of a method to build a safety and security modeling framework, supporting the co-engineering process. This also includes a tool-chain prototype architecture relying upon existing MDE and formal-based techniques to support the different phases of the approach, followed by an introduction to the CDVs—Connected-Driving Vehicles use case for illustrating the applicability of the approach.

In **Chapters 4** and **5**, we present the approach phases as the main contributions to integrated system safety and security design and analysis via modeling and formalization activities, respectively. This involves exemplifying the system-related notions relying upon the CDVs use case, along with enlisting the requirements and the details on the accompanying tool support.

In **Chapter 6**, we illustrate and assess the applicability of our approach and proposed tool-chain support prototype via the CDVs use case. An instance of a scenario, which is both safety- and security-critical, is targeted in this regard.

Finally, in **Chapter 7**, we conclude this manuscript by summing up the contributions and discussing the potential future work and perspectives.

Chapter 2

Related Work

Contents

2.1 Introduction

Taking into account the context and problematics stated in Chapter 1, we establish a systematic state-of-the-art in literature and industry practices. In particular, we are interested in understanding how current approaches undertake the integration of safety and security concerns, specifically properties, into the system development process, with existing research gaps and artefacts that can be used to support the methodological aspects to address safety and security interplay. By "approach", we mean a way or strategy covering entangled or correlated aspects like frameworks, methods, or implementation tools to address the mentioned problematics. It is recalled from Section 1.1.2 that safety and security are considered the ultimate goals to be achieved by the target System Under Design (SUD), and that can be built upon or influenced by a range of properties. To this end, we highlight the existing works across two domains of research: 1) approaches to incorporating safety and security in System Engineering (SE) and 2) approaches to safety and security design and analysis. We primarily focus on the system architectural design stages. However, we do not aim to exclude other approaches, e.g., those dealing with life-cycle stages involving dynamic adaptation or different life-cycle models. We relied upon the basic principles from [54] for carrying out a Systematic Literature Review (SLR) to explore, analyze, and interpret various contributions from the literature relevant to

these domains. Instead of individually reporting the works, we characterize them based on multi-attribute taxonomies relying upon the system, safety, and security domain concepts. At the beginning of each major section, viz. Sections 2.2 and 2.3, we position our context and requirements to the state-of-the-art. Likewise, at the end of each major section, an attempt is made to identify and assess concepts and methods as key takeaways to be used in the context of model-based approaches for the design and analysis of safety and security.

Overall, the chapter is organized as follows: In Section 2.2, we detail the existing works on different approaches in safety and security engineering and how to integrate safety and security concerns into the SE process, including some standalone and co-engineering approaches, as well as holistic approaches. In this regard, the features exhibited by these approaches are highlighted, considering the major phases, including requirements, architectural design, and analysis, of the SE process. Furthermore, in Section 2.3, we examine the existing contributions based upon the different constructs with varying degrees of formality adopted for the safety and security design and analysis via specification and verification. In this regard, the capabilities and feasibility of these constructs are precised. Finally, we conclude the chapter in Section 2.4.

2.2 Approaches to Incorporating Safety and Security in System Engineering

Both safety and security domains flourished independently in practice, particularly in terms of elementary requirements, constraints, manifestation of risks, and impact assessment [55]. Nevertheless, considering their mutual impact, it is paramount to incorporate the duo of safety and security concerns during the SE process. Significant research efforts are being invested in realizing the vision of safety and security co-engineering. However, this idea is still lagging for several reasons, including a lack of combined standardization, methodological, and implementation practices during system development [56]. To better understand the status quo in practice, a systematic and comprehensive articulation of the existing contributions incorporating safety and security in the SE process is imperative. In this section, we investigate the state-of-the-art of such approaches from both standalone and combined perspectives. This exploration will contribute to positioning the proposed approach for safety and security co-engineering in the forthcoming Chapter 3.

2.2.1 Materials and Method

To be precise with the scope of our investigation, we mainly focused on the approaches concerning different stages of the SE process. As mentioned before, this work is undertaken by extensively following the guidelines proposed in [54] for conducting an SLR. In this case, the aim is to rely on a commonly accepted template or methodology for identifying and interpreting the existing literature relevant to safety and security interplay while maintaining the transparency of the results. The methodology that this work endorsed broadly includes the following four phases: 1) research context-oriented formulation of the problematics defining the scope of this study, which is already covered in previous Chapter 1, 2) exploration of the existing literature in the direction of the formulated research problems, 3) multi-attribute analysis of the identified literature in the context of the problematics, and 4) discussion on the key findings based on the literature analysis, comprising existing lacks to be addressed. We sought to iteratively go through these different phases for this study to enable its thorough evaluation. A simplified pictorial representation of the entire four-phase process, with specific steps and outcomes per phase, is depicted in Figure 2.1. Besides, the individual steps are documented in detail in the following paragraphs.

Literature Exploration and Organization. To bring to light the relevant contributions in the existing literature, we followed a two-stage literature exploration process. We began with conducting an automated search on existing scientific databases like Web of Science, ACM Digital Library, DBLP, and IEEE Xplore, using a set of search keywords constituting the following Boolean string:

"Safety" **AND** "Security" **AND** ("Engineering" **OR** "Interplay" **OR** "Co-engineering" **OR** "Integration") **AND** ("Requirement", **OR** "Design" **OR** "Analysis" **OR** "Risk assessment") **AND** ("Critical systems" **OR** "Cyber-Physical Systems")

Afterwards, we manually refined the obtained search results using a set of inclusion and exclusion criteria. This mainly involved selecting contributions related to incorporating safety and security concerns in isolation or integration in the SE process. The duplicate contributions appearing across multiple platforms were discarded.

Furthermore, we developed and elucidated a multi-attribute taxonomy to analyze the obtained research contributions, comprising some novel and already used attributes in the existing literature. To accomplish this, we used the concepts across system, safety, and security engineering domains as a basis to characterize the selected research contributions, as detailed in sub-section 2.2.2.

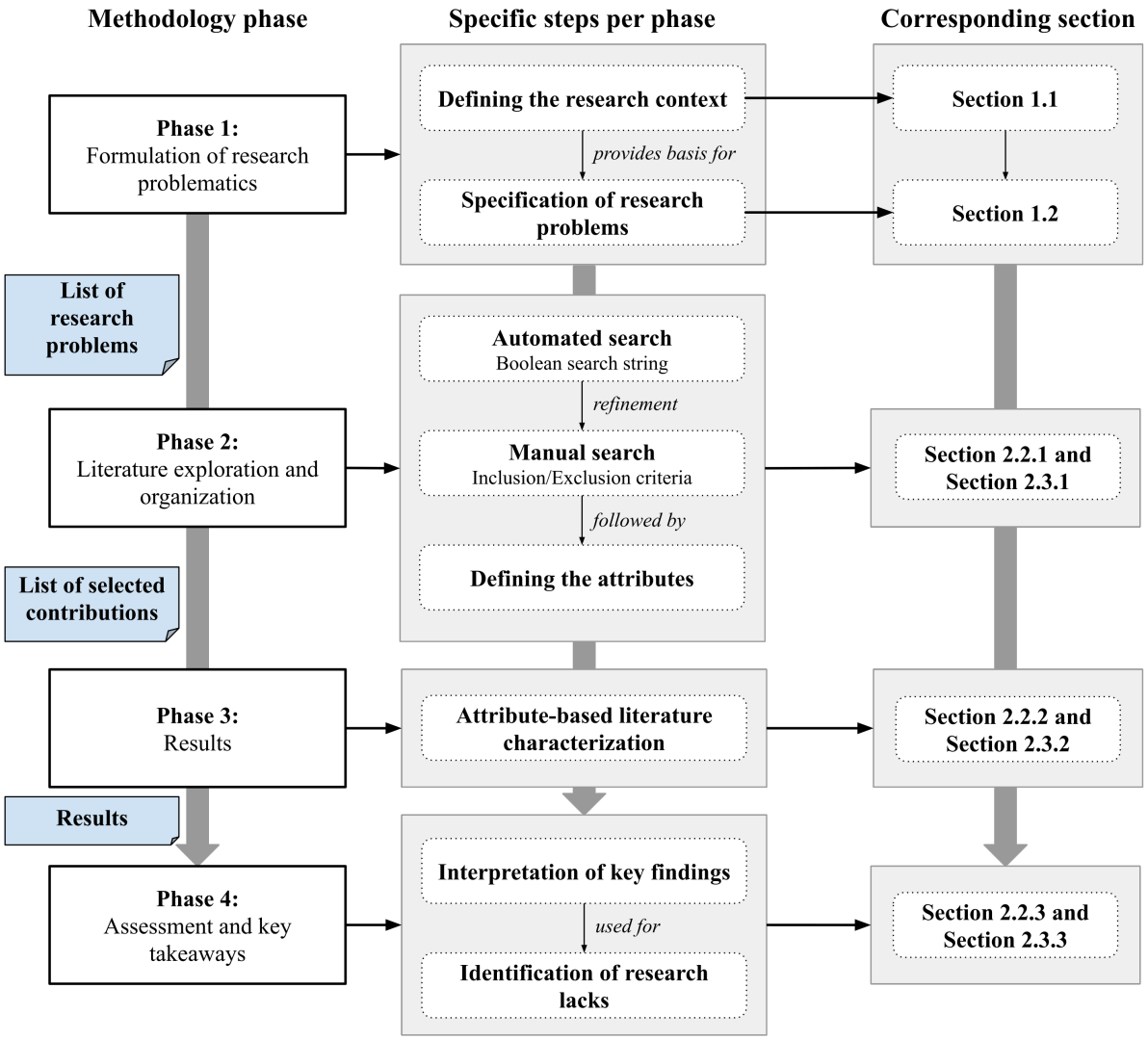


Figure 2.1: Flowchart depicting Systematic Literature Review (SLR) Methodology.

To address the *problematics* **P1**-**P5**, we used the following list of attributes in order to characterize the selected *standalone* approaches for the incorporation of safety and security in system development for prospecting the capabilities offered by them in the domain:

- *Life-cycle stage:* Represents the target phase of the SE process to which the proposed solution is applicable. The values that we considered include: *Requirement* (R), *Design* (D1), *Development* (D2), *Risk Analysis* (RA), or *Generic* (G) for all the phases, inspired by [57, 58].

- *System specification layer:* Represents the layer at which the target system is described. This assists in pinpointing the scope of the proposed solution based on the level of

granularity with which the system is represented. The values that we considered include: *Mission* (M), *Functional* (F), or *Component* (C). From the typical definitions and concepts associated with these layers found in the literature, we considered a set of keywords to characterize the selected contributions based on this attribute. For instance, to group a contribution into the category of mission-centric system representation, we chose a list of keywords that includes, but is not limited to, *Mission*, *Goal*, *Operation*, and alike, and subsequently, analyzed the proposed solution in the context of these keywords. Similarly, for the functional layer, we chose: *Function*, *Functional path*, *Information flow*, and alike. And finally, for the component layer, we chose: *Component*, *Port*, *Connector*, *Object*, *Entity*, and alike.

- *Stakeholders:* Represents the stakeholders that provide the domain knowledge for various activities associated with the applicability of the proposed solution. The values that we considered include: *Safety/Security Expert* (SS), *Analyst* (A), *Designer* (D1), *Developer* (D2), or *Generic* (G) for all the stakeholders [59].

- *Propagation of safety and/or security semantics:* This attribute denotes whether the translation or mapping aspects among different layers of system representation are discussed or addressed. This will assist in identifying the features that the proposed solution exhibits to facilitate or foster communication among several stakeholders with varying levels of expertise in terms of knowledge about the system representation. Possible values are: *Yes* (Y) or *No* (N) [**NB:** Not addressing safety and/or security propagation is not necessarily a drawback in the respective contribution, especially when it is not the goal of the work. Even so, it is highlighted in the present manuscript for the sake of a clear view of the state-of-the-art].

Along with the attributes listed above, we used the following additional ones to characterize and analyze the capabilities of the selected approaches to safety and security *co-engineering*:

- *Safety and security interaction:* Represents the type of interaction between safety and security concerns. This may further assist in envisaging the relationships, e.g., conflicts, that may arise between them. Possible values are: *Safety-informed Security* (SS1), *Security-informed Safety* (SS2), or *Combined* (Co) [60]. Here, *SS1* indicates that security fulfillment depends upon safety, *SS2* indicates that safety fulfillment depends upon security, and *Co* indicates that safety fulfillment depends upon security and vice versa.

- *Conflict resolution:* Whether the proposed solution takes into account or facilitates the identification and/or analysis of the potential conflicts between safety and security concerns. Possible values are: *Yes* (Y) or *No* (N) [59].

2.2.2 Results

Herein, we summarize the results of our study on the approaches from standalone and co-engineering perspectives.

2.2.2.1 Standalone Approaches

Tables 2.1 and 2.2 respectively provide a concrete summary of some of the selected approaches from the literature for incorporating safety and security in the SE—System Engineering process in a standalone fashion, based on the attributes mentioned in the previous sub-section 2.2.1.

Table 2.1: Summary of Selected Approaches to Incorporating Safety in System Engineering (SE) Process.

Ref.	Contribution	Life-cycle Stage	System Specification Layer	Stakeholders	Propagation of Safety Semantics
[61]	• Multi-stage design process to address conflicting safety requirements • Algorithmic approach towards response time and reliability enhancement	D1	F	D1	N
[39]	Modeling framework for integrating functional safety analysis in critical system design process	D1	M, F, C	A, D1	N
[62]	• Model-based holistic approach for analyzing functional safety requirements • Supports technical and social requirements specification through refinement	R, RA	F, C	SS, D1	Y
[63]	Approach for verification and validation of system properties	D1	F	A	N
[64]	Framework for assuring run-time safety of component-based Cyber-Physical Systems (CPSs)	D1	M, F, C	SS, D1	N
[65]	• Component-based design for high-level system description and real-time behavior analysis • Use of Dependable Emergent Ensembles of Components (DEECo) for modeling component-level interaction	D1, D2	C	D1	N
[66]	• Functional-level co-design methodology for safety-critical CPSs • Modeling technique for explicit partitioning of physical and control functions	D1	F, C	SS, D1	N
[67]	• Development approach based on model-based, multi-layered software architecture for robot safety • Interconnected repositories for storage and reuse of the safety case models and software components	D1, D2	M, F, C	SS, D1	N
[4]	• Integration of safety analysis with model-driven software development • Conceptual modeling to identify safety concerns complementary to the software architecture	RA, D1	M, C	A, D1	N

Table 2.2: Summary of Selected Approaches to Incorporating Security in System Engineering (SE) Process.

Ref.	Contribution	Life-cycle Stage	System Specification Layer	Stakeholders	Propagation of Security Semantics
[68]	Approach for requirement specification, verification, and treatment	R, D1	C	SS, D1	N
[69]	• Analytical framework for capturing mission-centric viewpoint of critical Cyber-Physical Systems (CPSs) • Modeling technique for visualizing the system behavior as per mission specification	R, RA	M	G	N
[70]	Model-Driven Security (MDS) approach for guiding and automatizing pattern application in system development	D1	M, F, C	D1	Y
[71]	Security-aware modeling methodology for robustness validation	D1	F	D1	N
[72]	Roadmap for incorporating requirements in system life-cycle process	R	M, F, C	SS	N
[27]	Security characterization framework for exposing security profiles of software components	D2, RA	C	D2	N

2.2.2.2 Co-engineering Approaches

Furthermore, Table 2.3 provides a concrete summary of the selected research contributions from the literature for safety and security co-engineering based on the listed attributes.

2.2.3 Assessment and Key Takeaways

Herein, we discuss the obtained results for the key findings upon which we build our research and the lack in the existing contributions from a methodological perspective yet to be addressed.

Key Findings. Based on the characterization summary of the approaches presented in the previous sub-section 2.2.2, we offer a concrete view of our findings regarding the selected attributes:

- *Life-cycle stage:* Most of the approaches are inclined to incorporate safety [61, 39, 65] and security [70, 71] concerns in the early design stages of system development. The earlier we identify and address these domain-specific concerns, the more adequate will be the design of the system under development with a lesser number of repetitions. This reduces the overall cost and effort of developing safety and security-critical systems.

- *System specification layer:* Regarding the system specification layer, as we move from the

Table 2.3: Summary of Selected Approaches to Safety and Security Co-engineering.

Ref.	Contribution	Life-cycle Stage	System Specification Layer	Stakeholders	Propagation of Safety and Security Semantics	Safety and Security Interaction	Conflict Resolution
[73]	Co-analysis study	G	F	A, D1	N	Co	N
[74]	System-theoretic co-analysis approach	R, D1	M, F, C	SS, D1	N	Co	Y
[75]	Joint deployment approach	RA	M	SS, A	N	Co	N
[76]	Co-engineering approach	R	M, C	SS, D1, D2	N	Co	N
[77]	Security risks analysis method and framework	RA	F, C	SS, A	N	SS1	Y
[78]	Co-engineering process	G	F	D2	N	Co	N
[79]	Co-certification approach	G	M, F, C	SS	N	Co	N
[80]	Co-analysis approach	RA	C	SS, A	N	Co	N
[81]	Co-engineering approach	D1	F	D1	N	SS1	N
[82]	Integrated analysis approach	RA	F	SS, A	N	Co	N
[83]	Attack injection framework	D1, D2	C	D1, D2	N	SS2	N
[84]	Framework for integrated analysis	D1	M, F, C	D2	N	Co	N
[85]	Co-engineering approach	G	C	G	N	Co	Y
[86]	Co-engineering method	RA	F, C	A, D2	N	Co	N

higher levels of granularity to the lower ones, more details about the underlying system and its entities are available for a refined safety and security analysis. Notably, safety analysis can still be conducted at the mission layer [69] with high-level system-related information. However, security concerns are often dealt with at the information and communication level [68] due to the availability of the necessary implementation-related details.

- *Stakeholders:* Safety and security experts contribute to the knowledge for the assessment of safety [62, 67] and security solutions [68, 72] and how the system can go wrong within the considered operational environment. Similarly, analysts use their practical experience to elicit the potential hazards [63, 4] or threats [69] concerning the system functionality.

- *Propagation of safety and security semantics:* Only a few contributions discuss the layer-wise refinement of the system specification semantics in the safety [62] and security domains [70]. The results show that how to maintain consistency and transformation of safety and security concerns between different layers of a system representation is an open question.

- *Safety and security interaction:* A range of contributions, e.g., [73, 74], aims towards an interplay of safety and security concerns from a combined perspective. This can be attributed to the fact that a combined approach often assists in bringing the fragmented requirements and standalone practices from both domains under a realm via identifying potential conflicts—for instance, without losing the domain-specific traits. In such cases, safety and security engineers can mutually benefit each other towards a systematic and rigorous safety and security analysis of the system under consideration.

- *Conflict resolution:* Fewer contributions [74, 77] discuss resolving conflicts between safety and security concerns. And none of them are related to safety informed by security, even though security issues at the component layer can propagate functional failure, risking the availability of safety-related functions. Moreover, some contributions [77] discuss or address conflict resolution in the early design stages, thereby focusing on a preventive vision towards the occurrence of these conflicts during implementation.

Based on the domain-specific knowledge from the existing literature, we aim to build our research upon the following aspects:

- *Architectural design and analysis:* The system architecture development stage allows flexibility regarding the design decisions, where safety and security concerns can be incorporated and analyzed to obtain a safe and secure architectural model of the target application.

- *Three-layered model:* To address safety and security concerns across different stages of system development, i.e., mission, functional, and component, offering varying levels of details for analysis, we aim to consider a three-layered model in the context of the multi-layered system representation discussed in Section 1.1.1. This would provide better coverage for specifying and verifying safety and security properties.

- *Multi-stakeholder environment:* The safety and security activities are conceived from the perspective of different stakeholders, e.g., safety and security experts, analysts, and architects, involved in the system development. This requires support for their collaboration and cross-fertilization of their knowledge to yield a global safety and security system development process.

Lacks in Existing Research. Despite a range of contributions in the domain, our analysis reveals the following challenges pertinent for additional research efforts to realize safety and security interplay:

- *Conflicting results:* Joint safety and security analysis may lead to conflicting results, also termed antagonism [87]. In such cases, there is a need for approaches to identify and support the analysis of potential conflicts.

- *Lack of consistent semantic transfer:* Most of the contributions are not concerned with the layer-wise translation or mapping of the underlying semantics. As mentioned before, safety analysis can be conducted regarding the high-level goal-oriented description of the system up to the detailed architectural view. However, the security analysis is

highly information-centric and often requires the component-level details of the system. Cascading down the safety-specific information to the level where an integrated analysis with security can be conducted requires consistent semantic transfer of knowledge among the stakeholders involved at each layer [88]. Nevertheless, there is still a gap in a uniform language to transmit information between different layers without losing details.

- *Lack of application domain-agnostic knowledge:* Literature study in the field shows the inclination of the existing approaches for safety and security interplay towards specific application domains to harmonize the respective concerns via methodological support. Moreover, clear applicability of the solutions in the literature concerning the claimed application domains is not demonstrated in many cases (e.g., in [89, 90]).

- *Complex standalone development and lack of automation:* The present landscape of safety and security development from a standalone perspective (often considered as "silos") is quite complex and expensive [91]. Moreover, it relies on human intervention to conduct the necessary tasks, with a lack of automated tool support to ease and speed up the safety and security co-engineering process, which is still in its infancy. The theoretical and practical exploration of the inter-relations between both domains towards consolidating best practices remains limited.

2.3 Approaches to Safety and Security Design and Analysis

Specification and verification of safety and security concerns, e.g., properties as introduced in Section 1.1.2, is crucial in the design and analysis of safety- and security-critical systems. This typically demands elicitation of the underlying requirements and assurance of consistency, accuracy, completeness, and fulfillment of the corresponding properties through specification and assessment. Identifying appropriate means to support these aspects becomes essential to incorporate concerns in the SE—System Engineering process, obtaining effective system functionality regarding safety and security. Several languages, tools, and methods exist in the literature to achieve this, which can be categorized based on the degree of formality of the constructs used to represent and validate the system under consideration [92, 93].

In this section, we are particularly interested in investigating the existing methodologies targeting safety and security design and analysis from the viewpoint of specification and verification. The exploration shall shed light on the methodological support for integrated safety and security design and analysis. This will contribute to positioning the specific contributions in the forthcoming Chapters 4 and 5.

2.3.1 Materials and Method

According to the degree of formality, safety and security concerns can be structured and refined differently. Based upon this, various levels of *formalization* exist in the literature for specifying, modeling, and verifying these concerns [94]. These comprise 1) language constructs that offer syntax, semantics, and modeling rules, 2) tools to automatize the modeling and verification process, and 3) techniques to address the concerns by synthesizing or analyzing the system during design. These can be used in a standalone or integrated manner to specify the safety and security properties, capture the whole set of underlying requirements, and represent design rationale. Accordingly, relying upon these aspects, we pursued the methodology depicted in Figure 2.1 to conduct an SLR study on the approaches to safety and security design and analysis, which is detailed in the following paragraphs.

Literature Exploration and Organization. We sought to identify the building blocks and associated literature to address the specification and verification of safety and security concerns. We adopted both automated and manual search processes for exploring the literature concerning conceptual and empirical contributions made by the research community relevant to this survey. The automated search involved using the following Boolean string on existing scientific databases:

"Safety" **AND** "Security" **AND** ("Properties" **OR** "Requirements") **AND** ("Design" **OR** "Analysis") **AND** ("Model-Driven Engineering" **OR** "Formal methods" **OR** "Standards")

Afterwards, we conducted a structured classification of the obtained research contributions following two aspects: 1) the hypotheses or foundations for capturing safety and security properties and 2) how research practitioners have used these fundamentals towards conducting a scientific investigation or analysis through experimentation.

Furthermore, we developed and elucidated a multi-attribute taxonomy to analyze the obtained research contributions. To accomplish this, we used the aspects across design and analysis activities as a basis to characterize the selected research contributions, as detailed in sub-section 2.3.2.

To address the *problematics* **P1**-**P5**, we used the following list of attributes in order to characterize the selected approaches based on the means and techniques to support safety and security design and analysis:

- *Modeling aspects:* Represents the aspects of the target system that are modeled.
- *Property:* Outlines the set of safety and security properties addressed in the contribution.

- *Relationship between properties:* Outlines inter-related properties (if any).
- *Language:* Outlines the modeling languages (semi-formal or formal) used to model or specify system aspects and safety and security concerns. The language constructs determine the extent to which accurate, consistent, and unambiguous construction and analysis of these specifications can be conducted. A brief summary of the formal building blocks and their classification in the literature is provided in Appendix A.
- *Tool:* Outlines the automated tool (if any) used to specify and analyze system properties, including safety and security. Tools may be accompanied by methodologies for modeling and verifying properties.
- *Evaluation:*
  - *Use case/Case study:* Highlights the use case or case study (if any) for demonstrating the proposed solution and feasibility analysis.
  - *Assessment parameters:* Outlines the means provided to interpret and assess the results of the analysis conducted on the system under consideration and to use them as feedback to improve it.

In addition, we consider the following set of attributes to characterise the context of the selected contributions:

- *Target system:* Represents the system under consideration and the way it is represented during design and/or analysis.
- *Application domain:* Determines the target application domain for study or evaluation of results.
- *Standard/Regulation/Guideline (S/R/G):* Highlights the standards, regulations, or guidelines (if any) that provide the basis for the common knowledge and legal directives for safety and security properties and their specifications. A brief summary of the list of safety and security standards across several industrial application domains is provided in Appendix B.
- *Key assumptions:* Highlights the key assumptions made (if any) regarding the target system. The interpretation of requirements and scope of study may differ depending on these assumptions in certain cases.

2.3.2 Results

In this section, we summarize the results of our study on the approaches to safety and security design and analysis based on the attributes mentioned in the previous sub-section 2.3.1.

2.3.2.1 Characterization based on the Means and Techniques To Support Design and Analysis

Tables 2.4, 2.5, and 2.6 provide a concrete summary of the selected approaches from the literature based upon the means and techniques used by them to realize the safety and security design and analysis activities.

Table 2.4: Summary of Selected Approaches based on the Means and Techniques to Support Safety Design and Analysis.

Ref.	Modeling Aspects	Property	Relationship between Properties	Language	Tool	Evaluation	
						Use Case/Case Study	Assessment Parameters
[61]	System functions, non-functional requirements	• Response time • Reliability	• Severity • Exposure • Controllability	Set theory	-	-	• Response time • Reliability • Energy consumption
[25]	Spatial and temporal modeling	• Periodicity • Boundness	-	Metric Temporal-Spatial Logic (MTSL)	-	Train control system	Non-functional aspects
[95]	Network	Availability	-	Assertion language	ALFHA, VITE	Electric car	System functionality and time
[96]	Data monitoring units	• Integrity • Data freshness	-	Event-B	Rodin	Temperature Monitoring System (TMS)	Refinement results
[97]	Functions, components, control system	Sequential, temporal, combinatory	-	SysML, FOL, Temporal logic	UPPAAL	Mechanical press	-
[98]	Functions, components	• Response time • Robustness	-	-	-	Maneuver Assistance System (MAS)	-
[39]	System, functional, hardware, software-level	-	-	Meta-model	XText	-	Traceability
[99]	Component modeling	Temporal (e.g., time-out)	-	Textual, Meta-model	Artop	Production software	• Communication • Execution • Dynamic storage
[4]	Development modeling	Availability	-	UML	Objecteering 5.3, Rational Software Architect (RSA)	Consult System Flight Plan (SFPL)	• Consistency • Model integration
[63]	Behavioral constraints	Functional and non-functional	-	Simulink, Embedded Matlab (EML)	SDV, Uppaal-SMC	Autonomous traffic sign recognition vehicle	Energy consumption
[100]	Computation	Safety properties in general	-	Word automata	-	-	Reachability
[101]	Functional modeling	Behavioral	-	Petri-nets, B method	AtelierB	ERTMS/ETCS 3 train automation	Structural and behavioral concerns

2.3.2.2 Context-oriented Characterization

Furthermore, Tables 2.7, 2.8, and 2.9 provide a concrete summary of the selected approaches to safety and security design and analysis based on their scope.

Table 2.5: Summary of Selected Approaches based on the Means and Techniques to Support Security Design and Analysis.

Ref.	Modeling Aspects	Property	Relationship between Properties	Language	Tool	Evaluation	
						Use Case/Case Study	Assessment Parameters
[68]	Component-Port-Connector (CPC) architecture modeling	• Confidentiality • Integrity • Availability	-	FOL, Modal Logic	Alloy	Smart meter gateway demonstrator	Security properties
[40]	• Static structural modeling • Dynamic behavioral modeling	• Confidentiality • Integrity	-	UML, Z	Z/EVES	Bicycle parking embedded software	Domain and security checking
[102]	Intruder modeling	Confidentiality	-	Markov Decision Processes	PRISM	-	• Attack probability • Degree of confidentiality
[103]	• Cloud computing system modeling • Modeling of state-machine diagrams	Confidentiality	-	UML, Set theory	Proverif	ConfiChair	• Secrecy • Unlinkability
[104]	• Low-level modeling for security protocols • High-level modeling for the whole networked system	• Privacy • Authentication • Integrity • Freshness • Non-repudiation	-	Applied pi calculus (a-pi)	ProVerif	Fieldbus security system	Scalability (Limiting the growth of state space)
[105]	Trust chain modeling	• Static integrity • Dynamic integrity	• Trust • Access control	Set theory	-	Backdoor and code injection attacks	Program credibility
[106]	Communication model in distributed system	• Confidentiality • Integrity • Authenticity • Non-repudiation	• Integrity with Confidentiality • Authenticity with Integrity	First-order epistemic and modal logics	Alloy	-	• Eavesdropping • Corruption • Spoofing • Deniable reception • Deniable sending
[107]	Control loops	Availability	Resilience	Petrinets	-	Laboratory plant with water tanks	• MTTF • Availability
[108]	Threat modeling	• Users privacy • Information integrity • Authorized access • Accountability of actions	Safety	-	-	-	-
[109]	Data storage	Integrity	-	-	-	-	-
[110]	Modeling of system and security properties	• Confidentiality • Authenticity	-	SysML-Sec	ProVerif, TTool	Key distribution protocol	Attack resistance

Table 2.6: Summary of Selected Approaches based on the Means and Techniques to Support Safety and Security Integrated Design and Analysis.

Ref.	Modeling Aspects	Property	Relationship between Properties	Language	Tool	Evaluation	
						Use Case/Case Study	Assessment Parameters
[111]	Function, architecture, and task mapping	• Safety: Reachability and Liveness • Security: Confidentiality and Authenticity	-	SysML-Sec	TTool	-	-
[112]	Unified modeling of life-cycle phases of safety and security	-	-	FACT Graph	-	Stuxnet	-
[113]	Domain modeling of safety and security	-	-	Maude	-	Building automation	
[114]	System architecture	Structural and behavioral	-	First-Order Logic (FOL)	Alloy	Thales avionic system	Protection against failures and attacks
[115]	Interactions	-	Mutual reinforcement, Antagonism, Conditional dependency	BDMP formalism	FigSeq	Automatic door shutting system	Undesired event probability
[116]	Communication architecture	• Confidentiality • Authenticity	-	Domain-specific	UPPAAL, ProVerif	Keying protocol	-
[117]	Risk scenarios	-	Reinforcement, Antagonism, Conditional dependency	BDMP formalism	KB3 quantification	Pipeline	Event probability
[118]	Functional aspects	Integrity	Security	-	-	Communication systems	-

2.3.3 Assessment and Key Takeaways

In this section, we discuss the results obtained from the previous sub-section for the key findings upon which we build our research and the lacks in existing contributions that are yet to be

Table 2.7: Context-oriented Characterization Summary of the Approaches for Safety Design and Analysis.

Ref.	Target System	Application Domain	S/R/G	Key Assumptions
[61]	Cyber-Physical Systems (CPSs)	Automotive	ISO 26262	Design methodology-specific
[25]	Mission-critical CPSs	Transportation	-	No spatial change
[95]	Distributed time-triggered CPSs	-	-	Fault-tolerant functionality
[96]	Data Monitoring Systems (DMSs)	-	-	• Fault-tolerant functionality • Timing constraints
[97]	Control system	Machinery	IEC 61508, IEC 62061, EN 954-1, EN 574	Properties composition
[98]	CPSs	Automotive	ISO 26262	Intended driver's actions
[39]	-	Automotive	ISO 26262	-
[99]	System software	Automotive	ISO 26262, AUTOSAR	Requirement specification
[4]	Safety-critical	-	-	Software failure
[63]	CPSs	Automotive	-	-
[100]	Critical systems	-	-	Automaton conventions-specific
[101]	Critical systems	Railway	-	System model-specific

addressed.

Key Findings. Based on the characterization summary of the approaches concerning means and techniques to support safety and security design and analysis presented in the previous sub-section 2.3.2, we offer a concrete view of our findings regarding the selected attributes:

- *Modeling aspects:* The approaches capturing the high-level mission-centric aspects [61, 63] or functionalities [97, 98] of the target system to be modeled are dominant in the safety context. Likewise, most of the approaches capturing the detailed technical architecture of the target system can be found in the security context [68, 99]. This can be attributed to the fact that as we move towards the lower levels of granularity concerning the representation of the system-related aspects, more details about the underlying system and its entities are available for better security analysis. It is recalled from Section 2.2.3 that safety analysis can still be conducted at the mission layer with limited information. However, security aspects are often dealt with at the information and communication level, where the required details are available.

- *Property:* In the safety context, response time [61], boundness [25], and periodicity [25] are considered as the key properties considering the quantitative aspects concerning the system operations that are characterized by spatial and time attributes. Likewise, in the security context, confidentiality [40], integrity [68], and availability [107] are considered the key properties associated with the communication systems. As already mentioned in

Table 2.8: Context-oriented Characterization Summary of the Approaches for Security Design and Analysis.

Ref.	Target System	Application Domain	S/R/G	Key Assumptions
[68]	Distributed Component-Port-Connector (CPC) software system	Information and Communications Technology (ICT) systems	ISO/IEC 27000	Model-specific
[40]	Embedded software	Embedded Systems	-	-
[102]	Data dispersal algorithms	Cloud storage systems	-	Passive and probabilistic intruders
[103]	Cloud computing systems	Cloud computing systems	-	Secrecy, Unlinkability
[104]	State transition systems comprising nodes defined by a set of attributes	Fieldbus systems	-	• All fieldbuses, gateways, and system users belong to the same domain • Dolev-Yao adversary
[105]	Program under execution	Trusted computing systems	-	Separation of data and code storage
[106]	System comprising finite number of processes/components and communication connectors	-	WS-Security	• Reliable broadcast messaging semantics • Any process can hear any message sent by any other process
[107]	Cyber-Physical Systems (CPSs) equipped with Intrusion Detection System (IDS)	-	-	• Signature and anomaly-based IDS • System is in secure state initially
[108]	Implantable Medical Devices (IMD)	Healthcare	WMTS, MICS, MedRadio, NFC	Adversary models
[109]	Cloud computing-based storage	-	-	-
[110]	Embedded systems	Automotive	-	• No limits on loop • Attribute secrecy

Table 2.9: Context-oriented Characterization Summary of the Approaches for Integrated Safety and Security Design and Analysis.

Ref.	Target System	Application Domain	S/R/G	Key Assumptions
[111]	Embedded systems	-	-	Method iterations
[112]	Cyber-Physical Systems (CPSs)	-	ISA 84, ISA 99	-
[113]	CPSs	-	-	Safety/security world-specific
[114]	Embedded systems	Avionic	Visual Flight Rules (VFR), Instrument Flight Rules (IFR)	-
[115]	Industrial control systems	-	IEC 64443	System-related
[116]	Critical embedded systems	Automotive	-	Dolev-Yao attacker
[117]	Critical systems	-	-	Quantitative parameters-specific
[118]	Building Automation and Control Systems (BACS)	-	IEC 61508, ISO/IEC 15408	Safety Integrity Level (SIL)-specific

Section 1.1.2, the list of safety and security properties addressed by the contributions may vary depending on the target of protection. In addition, there is an influence of

the application domain-specific assumptions on the selection of properties in the safety context based on the applicability of the system. However, the set of security properties remains uniform due to the commonalities of the underlying communication platforms across different application domains.

- *Relationship between properties:* Only a handful of contributions discuss the relationships between properties within the safety [61] or security domains [106, 105]. Regarding safety and security interplay, some approaches [115, 117, 118] discuss the relationships, including mutual reinforcement, antagonism, conditional dependencies, etc., between the safety and security domain-specific properties. It can be envisioned that security issues at the component architecture layer can propagate functional failure, risking the reliability of safety-critical functions. Moreover, some of the contributions, e.g., [117], discuss or address early design stage conflict resolution.

- *Language:* Based on the degree of formality of the constructs, most of the contributions in the safety context used formal representations relying upon temporal logic [97], B method [101], etc., to express or analyze the underlying quantitative aspects of their solutions. On the other hand, security-related approaches, e.g., [103], relied upon semi-formal constructs with a certain level of analysis they offer. Although formal constructs offer reasoning capabilities; however, due to the higher level of complexity of understanding and implementing them, a compromise is often made concerning the analysis by relying on semi-formal notions that offer a better representation.

- *Tool:* To comprehend the integration of safety and security disciplines, industries now realize the importance of conquering the approaches that can better address their underlying concerns. Essentially, in some cases, most of these concerns coincide with common system installations and toolsets [117]. In such cases, to model safety and security inter-dependencies in an integrated framework, tools that can be extended from safety to the security domain and vice versa can be nominated as potential candidates. Based on the diverse set of activities performed during the SE—System Engineering process, these tools may range from requirement specification and system modeling tools, e.g., Xtext and Papyrus, to risk assessment and testing ones, e.g., Rodin and UPPAAL.

- *Evaluation:* It can be observed that a majority of the approaches, e.g., [25, 107], for safety and security design and analysis rely upon the use cases or case studies driven by the empirical evidence supporting the underlying hypotheses. Diversity can be observed in this regard for the choice of systems and constituents to demonstrate the approach or validate the results.

Furthermore, our findings regarding the characterization summary of the approaches concerning their context are presented as follows:

- *Target system:* Existing approaches address the safety and security design and analysis considering the specification and/or verification of underlying properties for different target systems. However, the development of integrated safety and security solutions for specific types of emerging systems (e.g., Cyber-Physical Systems (CPSs), communication, and control systems) and embedded systems is still a challenging aspect due to the difficulty in adhering to the standards belonging to the broad spectrum of domains and lack of well-defined design and implementation practices.

- *Application domain:* With regards to the application domain, most of the reviewed contributions, e.g., [61, 98, 99], targeted automotive systems for design and analysis in the context of safety. This can be attributed to the fact that in the automotive domain, there lies a potential for failure of the underlying system leading to accidents and causing injury or death. Likewise, the approaches, e.g., [103, 102], in the security domain are mostly inclined towards Cloud-based systems due to the conventional or emerging security risks associated with the underlying communication.

- *Standard/Regulation/Guideline (S/R/G):* Standards provide a conceptual basis for capturing the engineered systems' safety and security concerns. Several industrial standards have been published in which safety and security properties are defined, considering the specificities of the target application domain. These definitions serve to measure the completeness of the safety and security properties' specifications in the works found in the literature [61, 97, 68]. Safety and security domain-specific standards can be coupled with a holistic system-level process definition to capture the global behavior of large and complex systems, where safety and security-critical aspects can be conceptualized even without the availability of detailed architectural knowledge of the system [75].

- *Key assumptions:* In the safety context, some of the approaches address the assumptions related to the fault-tolerant functionalities of the target system [95]. In some cases [96], the assumptions are associated with the timing constraints capturing the temporal aspects of the system behavior. In the security context, most of the approaches, e.g., [104, 108], adhere to assumptions related to the underlying adversarial models that capture the capabilities of the attackers in terms of their access to the system and communication model.

Lacks in Existing Research. The choice of appropriate constructs or notions for specification and verification of safety and security properties may depend upon a plethora of aspects, including 1) the level of abstraction of the system model, 2) technical applicability, size, and complexity of the system, and 3) experience of the associated stakeholders, e.g., developers, designers, and users. As a must, coverage and traceability of safety and security requirements make the choice of approach more complex. Safety and security co-engineering is a relatively new domain and requires a critical understanding of the needs, gaps, state-of-the-art, and practices [119]. Our study of the existing literature brings forward the following associated gaps structured across the modeling, analysis, and evaluation activities, along with the support for their realization that must be fulfilled to address the idea of safety and security co-engineering:

- *Modeling:* The use of Domain-Specific Modeling Languages (DSMLs) implemented as Unified Modeling Language (UML) or Systems Modeling Language (SysML) meta-models and profiles is widespread for modeling software-based systems incorporating desired safety [101] and security concerns [110] and inconsistencies' detection, often in a standalone manner. Some approaches, e.g., [116], incorporate both safety and security properties. However, they adopt a fixed modeling view, not covering different layered representations nor design steps, like allocation. In addition, existing approaches often provide a unified viewpoint in terms of the modeling language to capture the safety and security domain-specific concerns. However, comprehending safety and security interplay requires maintaining the individual set of concerns from both domains and analyzing them for potential conflicts—for instance [115].

- *Analysis:* Literature review in the field shows that a lack of concrete formalization still exists in handling and analyzing inter-dependencies between safety and security properties, along with their quantitative aspects. These can be captured from different perspectives; no single technique can cover both of them. While standards provide ease of comprehension regarding the fundamental domain-specific notions for properties specification, formal constructs offer more precision in conducting the verification. Consequently, the need arises to integrate these notions vertically across different system representations (refer to Sections 1.1.1 and 1.1.2), as they require safety and security properties compliance. Safety analysis approaches are often quantitative in nature and typically based on formal-based techniques, such as Boolean-Driven Markov process (BDMP), Failure Mode, Effects, and Criticality Analysis (FMECA), and software reliability growth [117]. On the contrary, the security analysis is primarily qualitative and relies on graphical techniques, such as misuse cases and sequence diagrams [20]. It is evident from the review of primary studies that there is still a lack of explicit modeling

techniques that can encourage the formal reasoning of inter-dependencies between safety and security aspects in the early stages of the SE process [85]. An amalgamation of the benefits of both worlds seems to be difficult due to the complexity associated with understanding and executing formal techniques. In some efforts, formal logic, like First-Order Logic (FOL) and temporal logic, are used for rigorous specification of safety and security properties [25, 97, 68, 114]. Some of these works (e.g., [25, 97, 68]) are nonetheless less oriented to cover the integrated specification of high-level safety and security properties. As a drawback, works in this category are often constrained either by the underlying formal language (inability to represent required properties) or by lack of guidance (or even impossibility) to apply them at different layers, thus preventing the reusability of the outcomes.

- *Tool support:* The main concern demanding tool support is the passage from semi-formal structured models to their formal specifications. Tooled-formal methods, when used in the engineering process of safety- and security-critical systems, increase confidence in the aspects (e.g., properties) defined by the respective standards [120, 31]. Several works demonstrate the use of the Event-B formal method for rigorous analysis of safety [96, 121] and security concerns [122]. However, these works are constrained by the granularity level and concepts chosen for modeling what imposes requirements to be specified at the same level. If another layer is needed for conceptual modeling, further guidance is still required to interpret the concepts from the semi-formal structured models to their formal counterparts for specification and reasoning (including verification).

- *Evaluation:* Several contributions rely upon a specific set of parameters to assess the proposed solutions. Nonetheless, this requires at least a minimum level of maturity to understand the solution and extract a generic set of features that can be extrapolated and migrated across diverse application areas.

2.4 Conclusion

In this chapter, we surveyed several existing approaches for 1) incorporating safety and security in the engineering process of safety and security-critical systems and 2) conducting safety and security design and analysis via specification and verification of the underlying concerns, e.g., properties. We outlined and analyzed their features and capabilities regarding our research's key objective: provisioning methodological support for non-savvy engineers to conduct integrated design and analysis of safety and security in the SE—System Engineering process. First, we provided a basis to analyze the existing methodologies allowing incorporation of safety and

security concerns in the system development process in Section 2.2. To facilitate this survey, we categorized the identified approaches into standalone and co-engineering. We defined a multi-attribute taxonomy relying on previous surveys and knowledge in the domain to analyze the selected contributions. By doing that, we prepared a characterization summary of these approaches. This was followed by exploring and analyzing approaches to facilitate integrated design and analysis of safety and security in Section 2.3. To this end, we targeted different techniques in the literature for safety and security properties specification and verification and their interplay.

According to the findings in Sections 2.2.3 and 2.3.3, a new methodology is needed that should consider the existing approaches and fulfill the identified methodological lacks for incorporating safety and security concerns in the system development in unison. The approach should be holistic, adopting a global view of the SE process, targeting different stages, e.g., mission, functional, and component, of system development. Despite existing approaches addressing these stages, mainly in a segregated fashion, further work is needed to integrate these approaches during the system development. Besides, the conceived methodology should be benefited from the foundational notions of the safety and security standards and exploit the capabilities of the existing techniques, including Model-Driven Engineering (MDE) platforms, and precision and rigorousness of formal methods to facilitate integrated safety and security design and analysis. Accordingly, the approach must improve the feasibility of the existing modeling frameworks regarding the complexity associated with understanding and integrating formal techniques into the engineering process to conduct a sound safety and security properties analysis.

Chapter 3

Approach

Contents

3.1 Introduction

In this chapter, we present a general overview of our proposed joint design and analysis approach to ease safety and security co-engineering concerning the overall research problem stated in Section 1.2, which is—*How to address the interplay between safety and security during the design stages of the System Engineering (SE) process?* The aim is to articulate a methodological view for the overall development process comprising activities and building blocks belonging to two complementary aspects, viz. Model-Driven Engineering (MDE) and formal-based techniques used in the rest of this work to incorporate safety and security into the system architectural design. The proposal starts by adopting a stakeholders' viewpoint in the context of the engineering process, pursuing support and guidance during the integration and verification of system safety and security concerns. Subsequently, we outline the key aspects upon which the approach is constructed and applied. This includes the development of a safety and security modeling framework to support the safety and security co-engineering process (addressing the *problematic* **P1**) and three-layered system modeling (addressing the *problematic*

P2). In addition, we discuss the technological support in terms of language constructs, tools, and techniques for assisting the framework in the defined context, resulting in a prototype tool-chain support architecture. Recalling the context provided in Section 1.1, the methodological support involves leveraging the capabilities of MDE and formal-based tools and techniques to facilitate safety and security design and analysis in unison.

Overall, the chapter is organized as follows: In Section 3.2, we present an instance of the stakeholders' interaction in a typical SE process to derive the need for safety and security co-design and analysis, the key aspects to support the development of the proposed approach, and the method to build the safety and security modeling framework. This is followed by the description of the prototype tool-chain support architecture proposed in Section 3.3 in the context of this work. Subsequently, we provide an introduction to a use case in Section 3.4 that will be useful to illustrate and assess the approach and specific contributions as part of it. The research methodology is detailed in Section 3.5. Finally, we conclude by summing up the contribution of the chapter in Section 3.6.

3.2 Proposed Approach

The overall approach and its related aspects are detailed in the following sub-sections.

3.2.1 Need for Safety and Security Co-design and Analysis – The Stakeholder's Perspective

Developing safety- and security-critical systems require a dedicated engineering process incorporating safety and security concerns, which can rely on high-level modeling to define the system architecture. In a model-driven design, analysis by verification of the safety properties, e.g., availability, can be conducted over the system, sub-system, or components, according to the design granularity and details present. Likewise, risk analysis to address high-level security concerns, e.g., unauthorized access, is often information-centric, which demands cascading down to a detailed system view, particularly that involving components and transmission channels. These aspects, in turn, call for 1) a consistent passage of system-related knowledge across different-granularity system views; 2) the integration of properties and their consistency, preservation, and traceability; and 3) the coordination and harmonization of collaborative work among dispersed stakeholders and their tasks in the engineering process.

However, the aspects above lead to a complex enrichment process that often lacks methodological guidance, modeling language including semantics, and automated tool support. This is particularly true for non-savvy engineers during the integration and verification of

safety and security properties. For example, we consider the scenario depicted in Figure 3.1 as an instance to capture the interaction among different stakeholders in a development process driven by models. This interaction typically occurs in the following order, with the numbers in parentheses corresponding to the numbers in the figure: 1) the different experts, namely *Distributed System Expert*, *Safety Expert*, and *Security Expert*, elaborate a variety of models encapsulating their respective knowledge or interests, 2) the *Analyst* interprets the models and conducts analysis, 3) the *Architect* considers the models and harmonize the domain expertise and outcomes to build the system architecture, 4) the *Architect* performs analysis on the system architecture model by specifying and verifying safety and security properties based upon the related architecture analysis artefacts. This, in turn, forms the basis to facilitate the *System Developer's* tasks. Notably, these stakeholders must reach a consensus regarding an optimal system design. Nevertheless, such collaboration makes the job of the *Analyst* complex, particularly when perceiving security models at different system representation views and jointly analyzing safety- and security-related feared events to produce artefacts or solutions with the expected safety and security features at the architecture level.

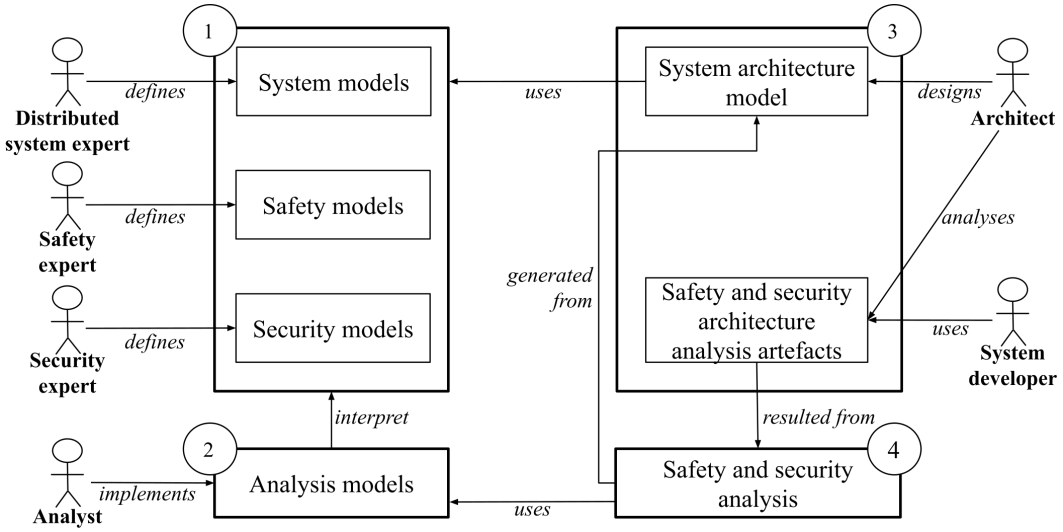


Figure 3.1: System Engineering (SE) Process for Safety and Security Co-engineering: An Instance of the involved Stakeholders.

By adopting the stakeholders' perspective and to address the previously described issues, we present an approach to support the safety and security co-engineering process. The approach primarily leverages the idea of capturing of safety and security concerns with the aim of co-engineering to facilitate the complexity of the design that decouples 1) system-related aspects from safety and security properties and 2) safety and security needs from solution architecture design, as detailed in the following sub-sections.

3.2.2 Key Aspects Supporting the Approach Development

Herein, we outline the important aspects regarding the construction of the proposed approach. We also refer to where in the manuscript we introduce them.

- *Three-layered system model:* We are targeting systems that are either three-layered specified or three-layered modeled. Although other systems that do not satisfy this aspect could also be addressed with this approach, they are nonetheless not discussed in this work. Refer to Chapter 4 for details.
- *Preliminary hazard and threat analysis:* We follow a preliminary hazard and threat analysis approach based upon which the safety and security concerns, e.g., properties, are specified. Refer to Section 6.2.3 for its applicability to the use case.
- *Modeling and software language engineering:* We assume that meta-modeling and profiling are means that can be leveraged in the SE process to address the modeling of system and safety and security properties and to capture the domain expertise. Refer to Chapter 4 and Section 6.2.4 for details.
- *Analysis:* We assume that formal-based techniques are means that can be leveraged in the SE process to address the verification of safety and security properties and improve system safety and security. Refer to Chapter 5 and Section 6.2.5 for details.

The overall approach is depicted in Figure 3.2. It facilitates 1) modeling of a target System Under Design (SUD) with varying granularity levels, thus defining the layers corresponding to the different stages of system development, and 2) incorporation of safety and security properties via separation of modeling purposes and languages, as opposed to incorporation into a flat single model or view. Accordingly, the system-related aspects can be rendered at the following three layers:

- **Layer 1**, *Mission layer*, concentrates on the formulation of high-level strategic aspects, called missions, of a complex engineered system, thereby offering a teleological view to capture its overall purpose.
- **Layer 2**, *Functional layer*, represents a classical functional decomposition of the system, reflecting the design objectives correlated with its functionality.
- **Layer 3**, *Component layer*, focuses on the low-level detailed technical specification of the target system wherein it is decomposed into a set of components, representing self-contained computational/communication elements or physical entities and their channels.

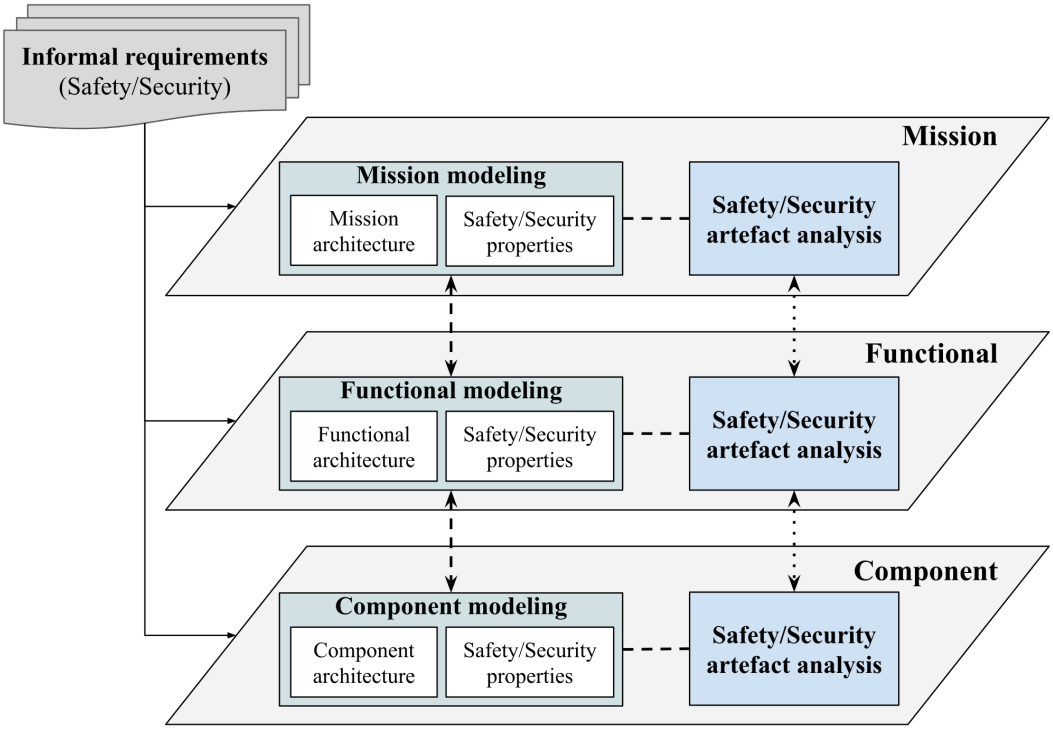


Figure 3.2: Approach for System Safety and Security Co-engineering.

The idea behind the aforementioned layered representation is driven by the aim to provide design choices and cover system-related aspects allowing safety and security properties analysis. Herein, the allied conceptual models are the cornerstone that 1) encompasses fundamental notions, their attributes, and potential relationships for capturing the structural and behavioral aspects of the target SUD at different levels of granularity, with *Mission* at the highest and 2) endows models with semantics to allow verification of properties. To address the layers' intertwined semantics, the framework is amenable for both top-down and bottom-up design strategies, whereby models' formalization shall allow progressive detailing of the SUD. Specifically, the top-down strategy would allow the incorporation of greater details during the system modeling, as described below:

- Mission captures *what* is needed to be achieved by the system.
- Function captures *how* to achieve *what* is needed to be achieved by the system.
- Component architecture captures *which* elements can finally realize the *what* and *how*.

On the other hand, the bottom-up strategy would allow for analysis of the propagation of malfunctioning at the lowest layer upwards up to the highest one. For instance, from a safety perspective, a faulty component like a sensor may eventually trigger a hazardous situation for

the whole system, e.g., an autonomous driving vehicle, by transmitting erroneous information. The structural and semantic linking among these layers concentrates on ensuring the consistency between corresponding representations and the preservation of properties via traceability.

3.2.3 Method to Build the Safety and Security Modeling Framework

With the three-layered model as its foundation, we present a method that encompasses the "safety- and security-by-design" principle, disambiguation in properties' specification, and early detection of potential conflicts between properties in the SE process, the three later as contributions for an effective co-engineering. This is accomplished via the following main steps that are depicted on the left side in Figure 3.3:

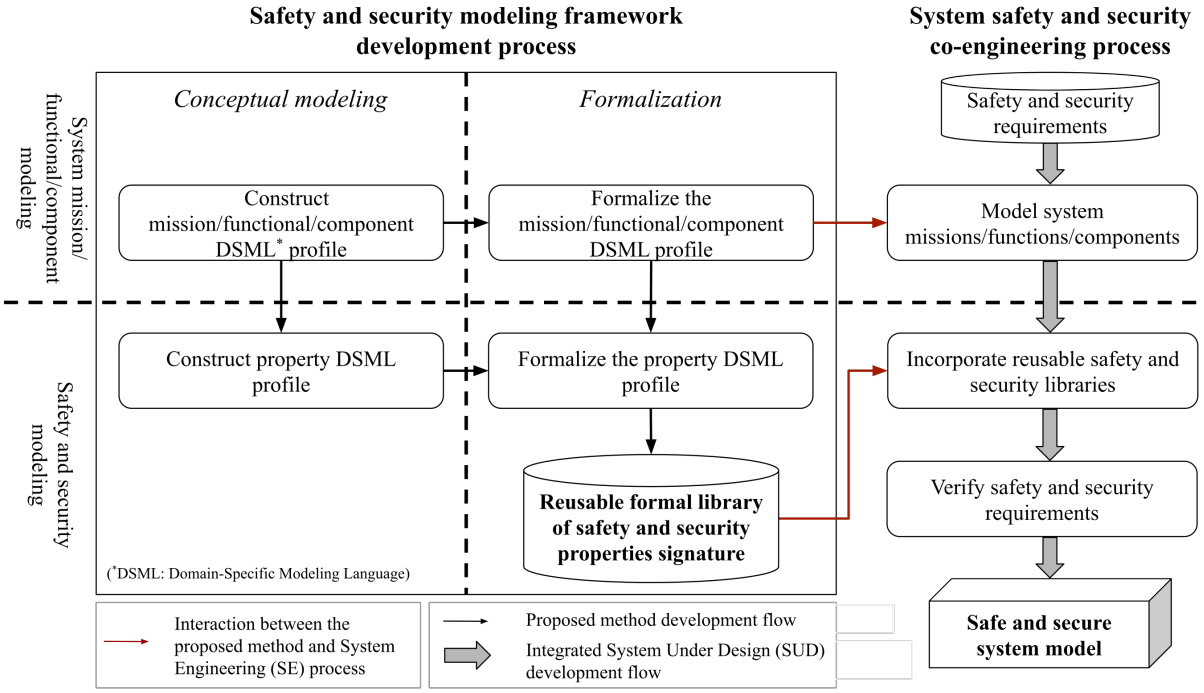


Figure 3.3: Method to Build a Framework to Support the Co-engineering Process of Safety and Security.

- **Step 1** *Conceptual modeling*: We begin with developing a conceptual model of the system aspects and safety and security properties concerning the three-layered model. This would provide a common understanding of the notions and their relationships in modeling the system and properties at the layers associated with different stages of system development. The corresponding Domain-Specific Modeling Languages (DSMLs) would allow the creation of the system architecture model from the conceptual model and identified safety and security concerns in the form of properties of the modeled system.

- **Step 2** *Formalization:* This involves the formalization of both system aspects and properties via logical specification of the corresponding DSMLs. In addition, a set of reusable formal model libraries for safety and security properties is also provided that are verified via interpretation into a delegated formal tool.

Once formalized, the properties' specifications instantiated at each layer remain generic enough to be accommodated as reusable libraries to assess the system model for the properties' violations. This, in turn, can be used to verify the fulfillment of the safety and security requirements. This is depicted on the right side in Figure 3.3.

3.3 Proposed Tool-chain Support Architecture

Herein, we propose a tool-chain prototype integrating the MDE paradigm and formal-based tools, supporting the modeling framework, DSMLs (meta-models and profiles), and formal specifications to assist the developers of safe and secure systems. The underlying tools must be able to support or illustrate the different phases and aspects of the proposed approach that will be detailed in the forthcoming Chapters 4 and 5. This would include the creation of system and properties DSML meta-models and profiles and a library of safety and security properties involved during the modeling and analysis activities in the proposed approach.

Figure 3.4 depicts the prototype tool-chain support architecture used in our work, comprising the following two primary blocks: 1) *Safety and security modeling framework development* block and 2) *System safety and security co-engineering* block. These are detailed in the following sub-sections, with their related activities numbered in parentheses.

3.3.1 Block 1: Safety and Security Modeling Framework Development

This block supports the following five activities:

- **(A1.1)** Creation of DSML meta-models and profiles for the three-layered system and safety and security properties.
- **(A1.2)** Defining the mapping for the DSML profiles to their corresponding logical specification, along with the logic-based properties' specification.
- **(A1.3)** Interpretation of the profiles to a concrete formal specification, using a formal method with accompanying tool support to capture the structural and behavioral aspects of the three-layered system and properties. This constitutes the formal model library corresponding to the system architecture and properties.

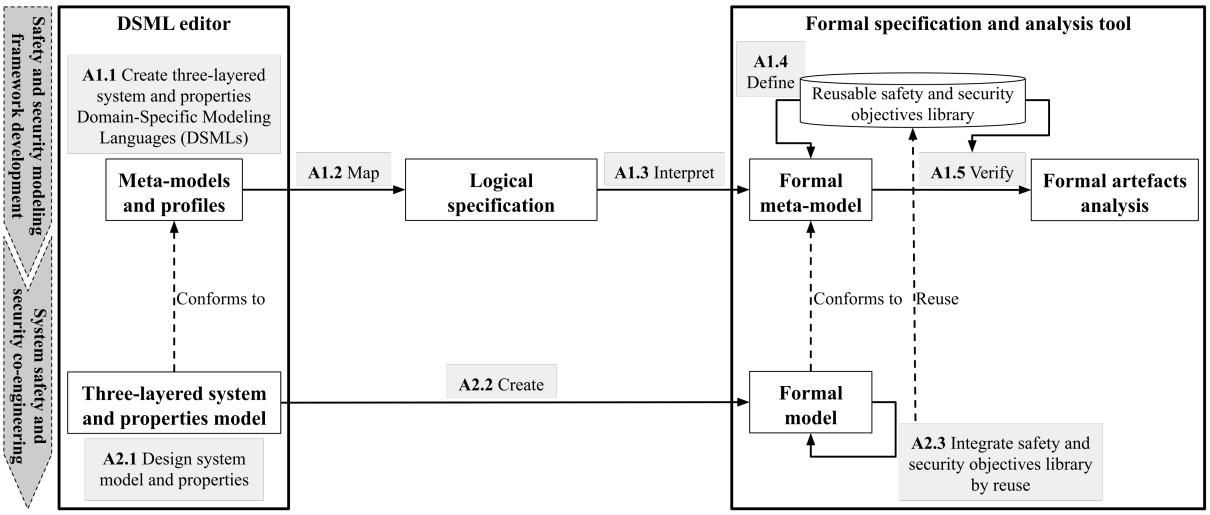


Figure 3.4: Overview of the Tool Architecture developed to Support the Co-engineering of Safety and Security: Integrated Design and Formal Analysis.

- **(A1.4)** Defining the reusable libraries of safety and security objectives as signatures in the context of the formal meta-model.
- **(A1.5)** Verifying the system design regarding the objectives signatures using the tool support.

3.3.2 Block 2: System Safety and Security Co-engineering

This block includes user-oriented features and supports the following three activities:

- **(A2.1)** Modeling a three-layered system and properties conforming to the DSML profiles created in (A1.1).
- **(A2.2)** Enabling the generation of a three-layered system formal model via refinement or interpretation of the already developed meta-models in (A1.3).
- **(A2.3)** Integration of safety and security objectives' specifications by reusing and adapting the libraries defined in (A1.4).

The details about the level of automation of the above-listed activities in the two blocks will be provided in Chapter 6 for the entire chain of approach instantiation. This will include activities broadly typed as manual, partially automated, or automated, depending upon the level of human intervention required.

3.4 Illustrative Use Case: Autonomous Connected-Driving Vehicles (CDVs)

In this section, we introduce the use case of autonomous Connected-Driving Vehicles (CDVs) as an automotive domain-specific application for the illustration purposes of the proposed approach. To address the current lack of literature related to integrated design and analysis of safety and security in the SE—System Engineering process, as discussed in Chapter 2, we demonstrate the applicability of our approach on a converging road plan scenario concerning CDVs, which will be detailed in Chapter 6. We rely on this use case to exemplify the basic notions involved in the different stages, viz. modeling and analysis, of the approach in forthcoming Chapters 4 and 5, respectively. In addition, the complete approach will be instantiated with the proposed tool-chain support prototype in Chapter 6 in the scope of this use case.

3.4.1 System Description

As the name implies, CDVs are enabled by inter-connectivity, allowing communication among the vehicles, Road-Side Infrastructure (RSI), and other remote entities like Cloud servers and smartphones. Communication encompasses technologies, services, devices, and applications, within or outside the vehicle, which are designed to facilitate transport, ease traffic, and mitigate accidents [123]. The transmission of information among these systems is often related to road congestion, vehicle's present speed, location, and so forth [6]. Unlike traditional vehicles, autonomous CDVs rely upon software to actualize features like self-driving, context adaption, and automated decision-making.

To briefly understand the underlying architecture of CDV systems, we focus on the logical decomposition of the system's motion-control module. It aims to ensure certain high-level functional goals of the system, such as *ensure driving*, and non-functional ones, such as *avoid collision with obstacles* and *ensure braking when needed*, as depicted in Figure 3.5. Accordingly, this module comprises various sub-systems, components, and sub-components for achieving these goals. The components can be classified into the following categories: 1) *Perception*, 2) *Processing and decision-making*, 3) *Actuation*, and 4) *External communication*, based on the dedicated functions they perform, as depicted in Figure 3.6.

Perception can be described as the interpretation and semantic understanding of the data about the vehicle and its surrounding environment [124]. This involves functions related to sensing the vehicle's state, for example, vehicle dynamics comprising speed,

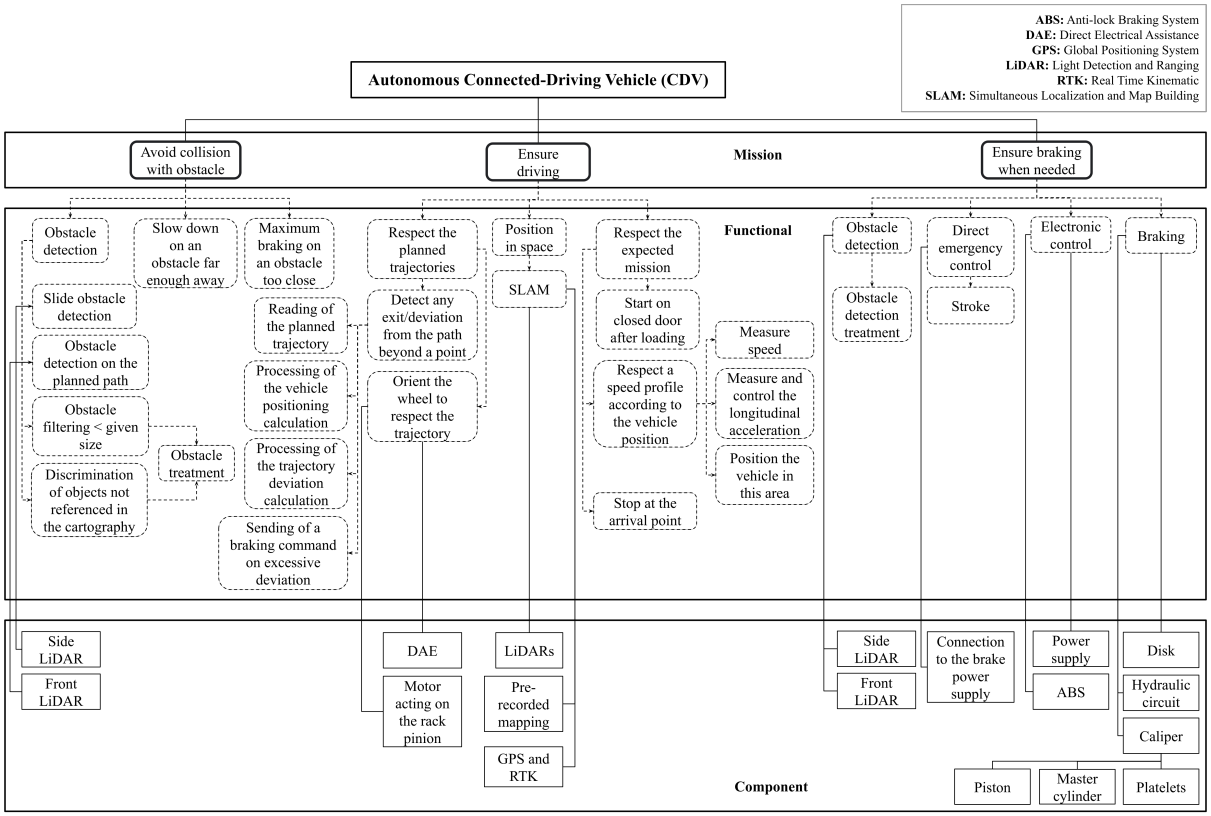


Figure 3.5: Layered Decomposition of the Motion Control Ecosystem in Connected-Driving Vehicles (CDVs).

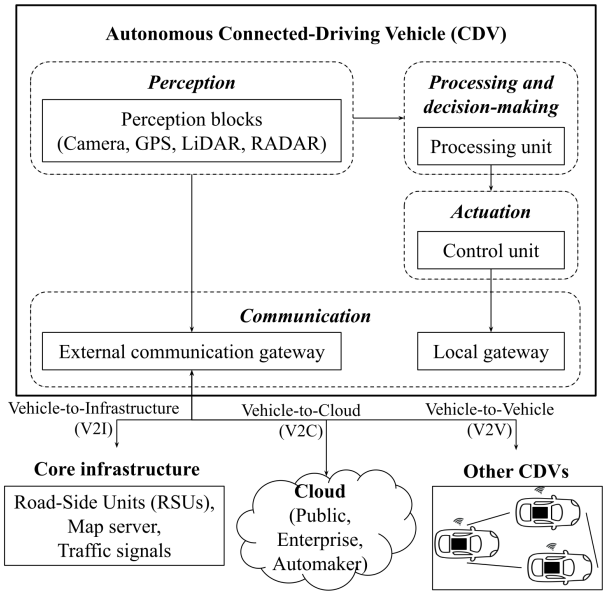


Figure 3.6: A Typical Connected-Driving Vehicle (CDV) System Architecture.

acceleration, angular momentum, etc., and also, the state of the environment in which the vehicle operates, for example, detection of obstacles, traffic, pedestrians, driving lanes, lane crossings, etc. In this regard, the vehicle relies on perception blocks like Light Detection and Ranging (LiDAR), Global Positioning System (GPS), surveillance cameras, inertial measurement sensors, etc., that translate the data captured by physical variables into signals for measurement. Specific algorithms like the ones for data fusion and localization aim to enhance vehicle localization accuracy and decision-making by coupling heterogeneous data sources [125].

Processing and decision-making deals with processing perception data and applying algorithms to determine a driving strategy in the context of the operational environment of the vehicle. It aims to ensure driving behaviors subject to constraints to generate human-safe driving. This involves functions related to perception data processing, vehicle positioning, trajectory planning, speed profile generation, navigation, etc., usually conducted by the On-Board Units (OBUs) [124].

Actuation deals with directing the vehicle or its components to operate physically. This involves functions related to controlling the vehicle's dynamics, for example, transmission, propulsion, steering, braking, etc. This is achieved via actuator devices like the steering wheel, electric and hydraulic brakes, main engine, etc., which transform the input signals to motion [125].

External communication involves the data exchange between the vehicle and the external environment to enable surrounding data collection, configuration updates, etc. [125]. Vehicle-to-Vehicle (V2V) communication allows data exchange among the vehicles, relying upon different protocols and antennas, e.g., Dedicated Short-Range Communications (DSRC), to access speed or location-related information. Vehicle-to-Infrastructure (V2I) enables exchanging information about traffic congestion, weather warnings, etc., between the vehicle and the external infrastructure. Other types of communications include Vehicle-to-Remote Cloud (V2C), Vehicle-to-Grid (V2G), etc.

3.4.2 Relevance of the CDV Systems regarding Safety and Security

Despite the theoretical and engineering-related technological advances achieved in the domain so far [126, 127], CDV systems are not yet completely free from the underlying design flaws from safety and security perspectives. The consequence of this is demonstrated by several instances in the past, some of which resulted in fatal [128].

Safety. From a safety perspective, there are driving situations in which malfunctioning of an underlying sub-system can induce hazard-causing behavior of the vehicle. Considering the motion-control module as the specific case, failure of the braking sub-system while the vehicle is approaching a junction—for instance, may lead to a collision with other vehicles and injury or death of the persons involved [129]. In such cases, faulty onboard components, e.g., sensors embedded in CDV system, can severely impact the entire vehicle's safety. The violation of timing constraints hinders the system's response to a safety-related event, e.g., the presence of an obstacle. This, in turn, can be associated with the failure of the sub-system or the components responsible for the in-time delivery of status updates to ensure safety-critical functionality. For example, on the appearance of an obstacle, the control signals from the multi-function control unit to the brake actuators should be sent in time to ensure the application of the brakes by the CDV. The timing constraints' violation, thus, can cause a delay in the actuator's action, affecting the in-time availability of the braking functionality [130].

Security. From a security perspective, increasing inter-connectivity offers potential attack surfaces, making CDV systems prone to intentional harm and exploiting their functionality. Communication is often susceptible to intentional but unwanted manipulations within a highly networked environment. An attacker can cause deliberate delays in transmitting messages, leading to the unavailability of the necessary functionality. Vulnerabilities associated with the ports may be exploited [131] that may cause the entire communication to be accessible for undesired manipulation. It is essential to ensure that the information or messages must not change during transmission without being detected by the intended receiver. Likewise, the system must restrict any malicious action performed by the components or entities whose behavior has been modified [132].

Need for Safety and Security Interplay. Due to the inherent design inter-dependencies, security-related concerns can impact the vehicle's safety and vice versa. For example, a vulnerability in the system can lead to a threat that can be exploited to interrupt the transmission of control signals for braking. Likewise, in case of an emergency call to avoid an accident, the concerned information is often unencrypted to prevent delays, and hence, prone to leakage by a security attack. The safety- and security-critical nature of CDV systems demands the correct implementation and verification of the mentioned concerns in the early stages of the SE process. Specifically, in the design phase, the aim is to prevent inconsistencies in the safety and security attributes of the system. For stakeholders involved in the system development process, e.g., those already mentioned in Section 3.2.1, documents remain the most suitable artefact. For example, hierarchical charts like the one depicted in Figure 3.5 are a first-hand description of the safety and

security concerns of the system. Given an operational context, a common understanding needs to be established regarding the design assumptions for multi-concern analysis and assurance [133]. Concerning the complexity associated with designing safe and secure CDV systems, methods and tools are required to support the passage of design-stage semantics among the involved stakeholders and ensure consistency of the document artefacts.

3.5 Research Methodology

This section accounts for the methodology we followed to construct the proposed safety and security co-engineering method and address the problematics in this Ph.D. With a two-fold purpose, this involved engineering of DSMLs and building the formalisms relying on the fundamental notions presented in Section 1.1 to respectively facilitate integrated system and safety and security design and analysis, as depicted in Figure 3.7. To develop the modeling and analysis framework, we conducted our research using the Design Science Research (DSR) method [134]. This method primarily focuses on designing and implementing the artefacts, including procedures, process models, and algorithms, to improve their applicability. Accordingly, we followed an iterative approach in this work, comprising three main steps related to the artefacts for design, analysis, and tool support and their application regarding the use case, with iterations concerning the different layers of the three-layered model.

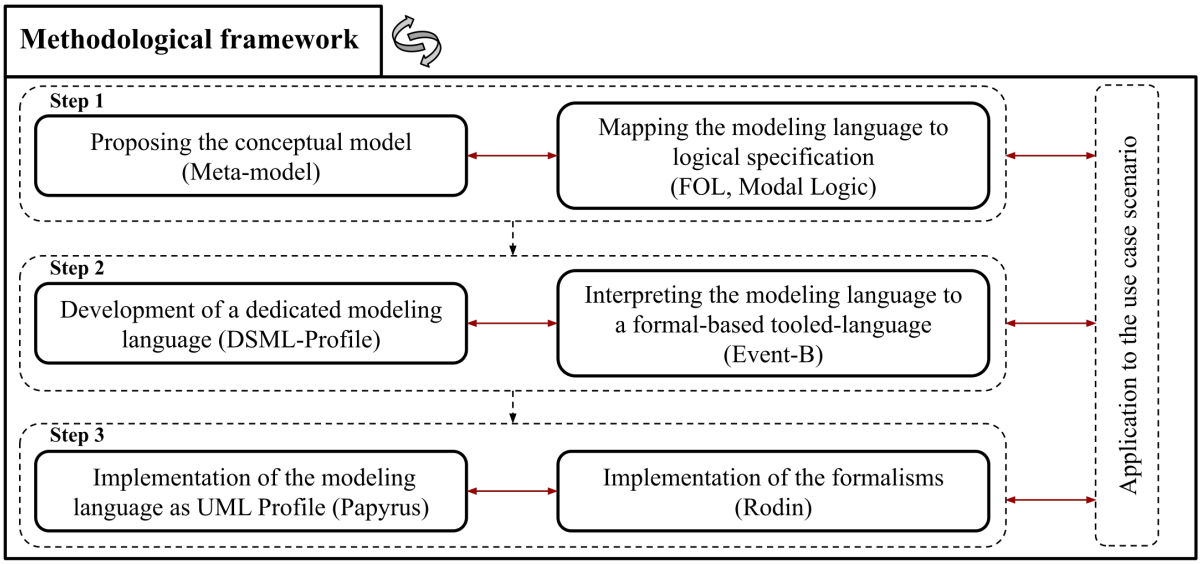


Figure 3.7: Methodological Framework.

- The *first step* involved the identification of the domain concepts to create a conceptual

model of the target SUD—System Under Design and properties. Furthermore, we worked on defining the formalisms corresponding to these fundamental notions and specifying properties using logic-based notions as extensions in the form of signatures that can be instantiated considering the target system model.

- The *second step* involved the development of a dedicated modeling language to facilitate the system and properties modeling and interpreting the modeling language to a formal-based tooled language.

- The *third and final step* involved the implementation of the modeling language and formalisms to facilitate the system and properties modeling for a target application and conducting proofs to analyze the fulfillment of properties by the target SUD.

The proposed approach uses the philosophy as in [135]: A logical specification of the system, safety, and security properties is proposed using an abstract system model (i.e., a technology-independent specification language, such as FOL—First-Order Logic and Modal Logic) followed by a more concrete specification of the system model and the properties (i.e., a suitable language with automated-tool support, such as Event-B).

As mentioned previously, in each step, we applied the design and analysis artefacts to the CDV use case to evaluate the respective contributions, viz. modeling languages and formalisms. For example, the *first step* involved conceptual modeling of the fundamental notions captured by the meta-models and the formalisms relying upon the use case scenario. We used FOL [44] and Modal Logic [45] to create technology-independent formalisms. Likewise, the *second step* involved designing the CDV system architectural model using the profiles and its formal modeling using the formal-based tooled language, namely Event-B [136]. Finally, the *third step* involved implementing the resulting modeling languages and formalisms in tool support comprising Papyrus [42] and Rodin [47] for respectively mechanizing the design and analysis of the CDV system architecture candidates and safety and security properties. In addition, applying the approach to the use case involved iteratively illustrating and analyzing the system aspects and properties in each step regarding the layers of the three-layered model. For example, the *first step* was further detailed with conceptual modeling of the CDV system regarding the notions defined for the *mission*, *functional*, and *component* layers, and safety and security *properties* in unison. The iterative process allowed for receiving feedback based on the results obtained from the use case-based application of the approach and improving the related artefacts before progressing to the subsequent iterations.

3.6 Conclusion

In this chapter, we presented a global view of the proposed approach for safety and security co-engineering. The approach relies upon MDE and formal-based techniques as the primary means for addressing integrated system safety and security design and analysis. The proposed method allows the creation of the design and analysis framework to assist the safety and security co-engineering process. Moreover, the prototype tool-chain support architecture aims to facilitate different phases of the approach via application and evaluation in the context of the CDV use case. This will provide methodological support to the system engineer via capturing the safety and security expertise and conducting a joint analysis in the early design stages of the system development.

In the forthcoming chapters, we apply our research methodology to address the *problematics* **P3**-**P5** stated in Chapter 1 via a detailed description of the following:

1. Three-layered system and safety and security properties modeling for creating a design integrating safety and security in Chapter 4.

2. Formalization of the system and properties models to facilitate an integrated safety and security analysis in Chapter 5.

3. Application of the approach phases and proposed tool-chain support prototype in the context of the CDV use case in Chapter 6.

Chapter 4

Modeling

Contents

4.1 Introduction

This chapter is dedicated to describing one of the main constituents of our proposed approach that is related to system safety and security co-engineering based upon Model-Driven Engineering (MDE) techniques. This part comprises a three-layered system modeling and safety and security properties modeling for addressing the *problematic* **P3**, which is creating a system design integrating safety and security, as mentioned in Section 1.2. To this end, we propose a set of Domain-Specific Modeling Languages (DSMLs), relying upon existing Unified Modeling Language (UML) [137] to define concepts and semantics corresponding to each layer and across layers of the three-layered system model and safety and security properties model. The idea behind different DSMLs is attributed to the fact that even if a single-layered view can express the desired vocabulary, it may become complex to use and confusing for the stakeholders or experts involved in the System Engineering (SE) process. Although by splitting the model, design complexity may increase given the different layers introduced; however, the relationships via links between the layered elements will simplify the passage of semantics across them.

Furthermore, the DSMLs are implemented using meta-models for the abstract syntax and profiles for the concrete syntax to provide a standardized modeling environment. To illustrate the notions provided herein, we rely on the use case scenario of autonomous Connected-Driving Vehicles (CDVs) introduced in Section 3.4, which indeed will be detailed in Chapter 6.

Overall, the chapter is organized as follows: In Section 4.2, we describe the method to engineer the three-layered system and properties modeling languages to benefit from their use and address safety and security interplay. In Section 4.3, we describe the features that seem fundamental for the three-layered DSMLs and the languages and techniques used to support them. This establishes the ground for our contribution towards integrated safety and security design. In Section 4.4, we introduce the three-layered system modeling, wherein the DSMLs are implemented as meta-models and profiles, capturing the fundamental notions of the System Under Design (SUD) and related concerns that can be mapped to the various stages of the system development, including mission, functional, and component architecture. Subsequently, in Section 4.5, we present DSMLs to support the modeling of safety and security in the context of the three-layered system model. Essentially, this includes features to tackle the interplay between the underlying properties, with the support for identifying conflicts. In Section 4.6, we present the details of the tool support used in this work to facilitate system and properties modeling. Finally, we conclude by summing up the contribution of the chapter in Section 4.7.

4.2 Method to Modeling Languages Engineering

This section describes the method to engineer the system and safety and security properties modeling languages. Recalling Figure 3.7, this involved several iterations for modeling system aspects and properties, as depicted in Figure 4.1. The constituent meta-models were evolved based on the consolidation of the feedback from an extensive literature review conducted during this process, as presented in Chapter 2. Herein, we choose the top-down design flow, although it is recalled from Section 3.2.2 that the method is also amenable to supporting the bottom-up flow.

The iterations are detailed as follows:

- **Iteration 1:** This iteration is related to the modeling of the target SUD at the mission layer. It begins with identifying the core concepts based upon existing mission specifications as input. Accordingly, we reuse and adopt the domain concepts to define the language elements and their relationships for modeling the system's missions.
- **Iteration 2:** In this iteration, we determine the elements to define the language for modeling the system's functionality. Accordingly, relationships are defined between the

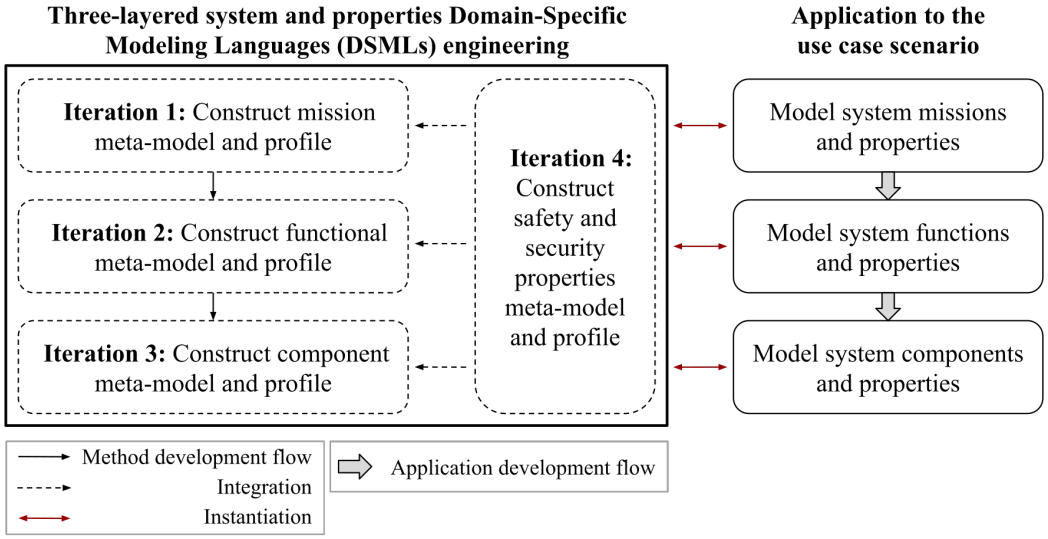


Figure 4.1: Method for Domain-Specific Modeling Languages (DSMLs) Engineering.

elements belonging to the mission and functional layers to ensure the consistency of the passage of knowledge across the corresponding representations.

- **Iteration 3:** In this iteration, we determine the concepts to design the system's detailed component-based architecture. Accordingly, links are defined to establish a relationship between the functional and component layer elements and ensure consistency in semantic transfer.

- **Iteration 4:** In this iteration, we define the language elements to model the safety and security properties of the target SUD and their interplay concerning the three-layered system model. This involves defining reusable model libraries of corresponding objectives that can be incorporated with the language elements defined for each layer. If a property cannot be verified at a given layer, it is delegated to the following layer to be corroborated.

These iterations are accompanied by an illustration involving a primary instance of the CDV use case borrowed from the state-of-the-art (refer to Section 3.4) to model and analyze a safety- and security-critical scenario relying upon the meta-model notions. Thus, prior to the iterations, we consider the overall system's description (refer to Section 3.4.1) as input, targeting the set of safety and security missions, underlying functions, and components involved, which will be detailed in Chapter 6.

4.3 Desired Features for the Modeling Languages

Globally, the aim is to allow harmonizing knowledge derived from the stakeholders involved in the SE process and to capture basic concepts in a generic way, facilitating safety and security interplay. To this end, the desired features of the modeling languages are described as follows:

- *Allow three-layered system modeling.* The three-layered DSMLs shall allow capturing the concepts and their relationships pertaining to the different granular representations of the target SUD. In particular, they must reflect notions that can be mapped to the various stages of the system development, including high-level purpose, functional features, and software and hardware architecture. This should assist in modeling the instances of a three-layered system representation as hierarchies of these notions. Moreover, the DSMLs should be accompanied by means of grouping the related modeling elements in a structured manner, like a package, for model comprehension and reusability.

- *Allow integrated safety and security modeling.* The languages shall serve the integration of the safety and security concerns associated with the system's representations. To this end, they should provide the elements necessary to model properties, e.g., via a library or catalogue of properties, and conduct a consistent interplay between them when necessary, e.g., to identify conflicts and support further analysis.

- *Ensure semantic transfer consistency and preservation.* Given that some properties may not be verified at a given layer. In such a case, it is recalled that to delegate the properties' verification to the lower layers, the DSMLs should provide means to link the notions and properties across layers. This is to ensure semantic transfer consistency between the corresponding representations and properties preservation via traceability.

The system and properties models are indeed decomposed in conformity with the different phases a system is developed in an MDE context. As mentioned in Chapter 1, UML supports the MDE approach and allows system architects and developers to specify, visualize, construct, and document system software. Being a general-purpose modeling language, it can be adapted using the notion of UML profiles to specify specific domains, e.g., safety and security, to develop large and complex system software. Accordingly, we rely upon semi-formal constructs offered by UML for the modeling aspects involved in this work. We use UML *Class* diagrams to create meta-models, describing concepts involved in the three-layered system and safety and security properties modeling, whereby concepts are represented by *classes*, concept attributes are represented as class *attributes*, and relationships among concepts within or across layers are represented by *links*, e.g., association, generalization, etc. Subsequently, we define UML *profiles*

that allow reusing and extending the existing UML concepts for customizing the models with domain-specific details. The profiles facilitate instantiating the meta-models, including system aspects and safety and security properties.

4.4 Three-layered System Modeling

In this section, we describe the DSMLs to facilitate the modeling of a three-layered system. The language elements allow capturing the target SUD at different levels of granularity, viz. *Mission*, *Functional*, and *Component*, with mission at the highest level. For this, we propose meta-models at each layer to define the associated concepts, their attributes, and their relationships. Afterwards, the relationships between the layered meta-models are introduced based on their underlying semantics, allowing them to be used in standalone and coupled modes. Furthermore, we propose UML profiles for the implementation of the proposed meta-models.

One of the key concepts of the models is the use of views [138]. A *view* in the present context can be typically described as a collection of all the relevant entities, their attributes, and relationships among the entities, corresponding to the phenomenon of interest concerning the mission, functional, and component layers. The introduced views stand for keeping a separation between the modeling purposes, including structural/behavioral aspects and safety and security properties of the target SUD, in the aim to facilitate incorporation and treatment of properties, rather than keeping them in a single encapsulated description. These models form the basis for the proposed design and analysis approach that can be aligned with different system development stages for safety and security co-engineering.

4.4.1 Meta-models for the Three-layered System

The meta-models corresponding to each layer of system representation are described as follows.

4.4.1.1 Meta-model for the System Mission Layer

The model for the system mission layer concentrates on the high-level strategic aspects, i.e., missions, of the target system without any low-level technical details. This involves analyzing the mission specifications as input and designing a mission model to achieve the underlying goals.

The associated meta-model captures concepts related to mission engineering. Herein, some of the representative elements constituting the “mission view” offered by this meta-model are semantically inherited from the state-of-the-art artefacts proposed for rigorously defining the

missions, as discussed in Chapter 2. The principal concepts of this meta-model are illustrated in Figure 4.2 and detailed as follows [**NB:** Concepts are represented in **Bold** and their attributes are represented in *Italics*]:

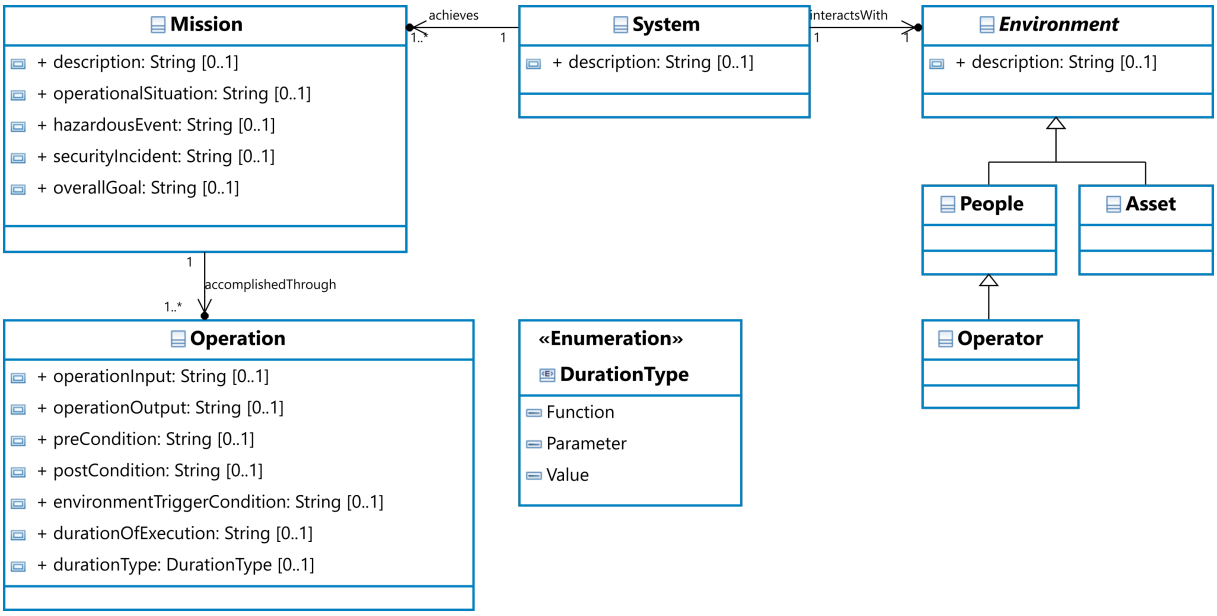


Figure 4.2: Mission-layer System Meta-model.

- **System:** It represents the target system, e.g., CDV, as an atomic unit, which is intended to achieve a set of missions. The system shares a physical and logical border to interact with its environment.
  - *description:* It provides a high-level textual description of the system and its characteristics, comprising details like the type of connectivity and mode of autonomy.
- **Mission:** It represents the system's overall purpose or an atomic finality related to some behavior, constraint, or property. For example, collision avoidance and grant only authorized actions in the CDVs use case. Herein, we consider only the high-level atomic missions that cannot be further decomposed into sub-missions. This is a choice to simplify the configuration and explanations without considering any dependency that may arise by introducing the sub-missions.
  - *description:* It describes the mission in the form of a statement. The mission statement reflects an obligation comprising the modal verbs, like *must*, *will*, and *should*, to indicate the necessity and certainty of something to happen.

- *operationalSituation:* It captures the operation-specific scenario that can take place during the system's life [30], like in the converging road map scenario.
- *hazardousEvent:* It represents the combination of a hazard and an operational situation [30]. Thus, it captures a situation that can lead to a hazard, e.g., unintended acceleration.
- *securityIncident:* Extending the notion of an information security incident from [31], a security incident in our context represents the combination of a threat and an operational situation. Thus, it captures a situation that can lead to a threat, e.g., unauthorized operations.
- *overallGoal:* It captures the desired operational situation or state of the system derived from the hazardous event or security incident.

- **Operation:** It represents the system's strategy or implementation steps for the accomplishment of the missions. For example, braking and access provisioning operations in the CDVs use case. Application of the operation leads to an elementary state transition concerning the system that may or may not be aligned with the behavior, constraint, or property prescribed by the mission statement. Two or more operations can be realized sequentially or in parallel. We extend some of the concepts provided in the operation model in [9], e.g., input/output, pre-/post-/trigger conditions, to define the following attributes:
  - *operationInput:* It represents the operation-specific input in the form of a stimulus, crossing the physical and logical border between the system and its environment. It impacts the system as a whole irrespective of its current state.
  - *operationOutput:* It represents the operation-specific output in the form of a response, crossing the physical and logical border between the system and its environment. It restricts the scope of the operation applicability, as only those system attributes variables will be affected by the application of the operation that are declared in the operation output clause.
  - *preCondition and postCondition:* It captures the descriptive pre-conditions and post-conditions associated with the application of the operation. These conditions are irrespective of any particular domain and typically capture the system's state from the operational perspective.
  - *environmentTriggerCondition:* It captures the prescriptive obligation to trigger an operation from the environmental perspective. Thus, it represents an additional

strengthening that assures accomplishing the system missions via the realization of the operations. For example, the decreasing distance between the CDV and its obstacle beyond a certain threshold should trigger the braking operation. An important concern is the inconsistencies resulting from multiple operations getting triggered with the same environment trigger condition. However, we assume that a trigger condition can only affect one operation at a time.

  - *durationOfExecution:* It defines the interval between the point at which an operation gets triggered and the point at which it produces an output.
  - *durationType:* It denotes the type of the operation duration, typed as **DurationType**. The possible values include: *Function*, *Parameter*, or *Value*. The operation duration can be a function of the input received from other system operations, a parameter describing the overall duration of execution for the operation, or a value indicating the maximum execution time of an operation.

- **Environment:** It represents the physical environment with which the system interacts. Inspired by [24], the notion of the environment in which the system operates may comprise: 1) **People** having cooperative or even malicious intent, e.g., system operators, maintainers, users, or observers, and 2) **Asset** that may be either tangible or intangible and exist outside the system, e.g., commercial, personal, or civic, including money, services, or physical property. We consider other systems similar to the target system, a part of the environment, specifically assets.

  - *description:* It provides a textual description of the environmental context in the form of a statement, comprising details like persons interacting with the system, infrastructure around the system, and environmental conditions.

4.4.1.2 Meta-model for the System Functional Layer

The model for the system functional layer reflects the design correlated with the system's functionality. This typically involves defining functions driven by the operations corresponding to the high-level mission specifications of the target system to accomplish its overall purpose.

The associated meta-model includes concepts related to functional engineering. To model the system functions at the detailed level, some representative elements are borrowed from the literature, as discussed in Chapter 2, to define the "functional view" offered by this meta-model. The principal concepts are illustrated in Figure 4.3 and detailed as follows:

- **System:** It represents the target system decomposed into a set of functions wherein it can be observed in the form of collaborative sub-systems performing a set of cohesive

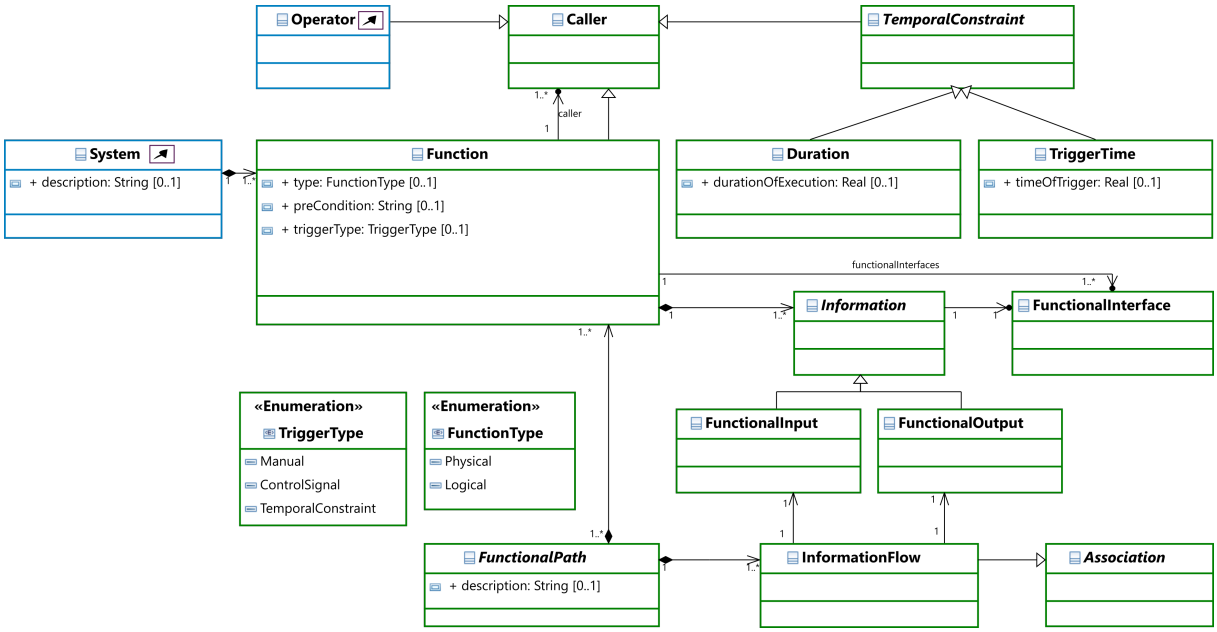


Figure 4.3: Functional-layer System Meta-model.

functions [**NB:** This element is inherited from the mission-layer system meta-model and extended for the purposes of functional design or engineering].

- **Function:** It represents one or more elementary behaviors, like perception- or actuation-related, which are performed or provided by the target system, commonly expressed in the active verb form.
  - *type:* It represents the overall type of the function, typed as **FunctionType**. The possible values include: *Physical* or *Logical*. The physical functions are representative of the physical processing elements of the system, whereas logical functions reflect the information processing tasks.
  - *preCondition:* It represents the mandatory conditions to be fulfilled and/or states to be reached for the invocation or execution of the function [139]. For example, pre-conditions associated with the signature of the function input.
  - *triggerType:* It represents the type of trigger as an external stimulus activating a function, typed as **TriggerType**. The possible values include: *Manual*, *ControlSignal*, or *TemporalConstraint*. The manual trigger lies in relation to the physical representation of the functions. It is associated with the invocation of the functions via physical actions that the system's operator can induce. A control signal is a signal received by a function from another function, thus triggering

the execution of the former. The temporal constraints represent the requirements regarding the temporal notions as a threshold for restricting the function's trigger time and execution duration. The precise semantics of the temporal constraints will be specified and detailed in the forthcoming Chapter 5, where a logical specification and semantics will be added to the languages herein defined.

- **Caller:** A caller is responsible for invoking a function. In correspondence with the above-mentioned trigger types, the function caller can be either a system **Operator** that initiates a manual trigger, an external **Function** that sends control signals, or **TemporalConstraint** that may be related to the recursive calls made by the function for self-invocation based on a temporal constraint [**NB:** The element **Operator** is inherited from the mission layer system meta-model].

- **Information:** It represents the information associated with the function that can be either of the following, inspired by the notion of input and output in [139] to describe a function:

  - **FunctionalInput:** It corresponds to the information received by the function that invokes or enables the function, which is further processed or transformed by the function.
  - **FunctionalOutput:** It corresponds to the information generated by the function as a result of the transformation of input, for invoking other functions or providing the final outcome.

- **InformationFlow:** It represents the link or flow of information from the functional output of one function to the functional input of another function, thus forming a sequence of functions.

- **FunctionalInterface:** It represents the point of interaction between the functions to allow their invocation via the exchange of information. The notion of the interface is relative based on its applicability. In the present context, it determines the semantics related to the function signatures, comprising input and output for the passage of information between the functions.

- **FunctionalPath:** It represents a sequence of functions, like obstacle detection → position inference → deceleration, to realize the system's operations with information flows, e.g., related to the dynamics of the obstacle. In other terms, it captures the flow of information between different functions, constituting a functional path. The first and last function in the sequence of functions constituting a path represent the boundary functions involving the system's interaction with its environment.

– *description:* It describes the ordering of the functions constituting the functional path in textual form, based on the flow of information among them. The constituent functions correspond to the operations defined at the mission layer for mission accomplishment.

4.4.1.3 Meta-model for the System Component Layer

The model for the system component layer concentrates on the low-level technical details of the system. It defines the system as a composition of component sub-sets representing self-contained computational/communication elements or physical entities.

The associated meta-model incorporates concepts related to the engineering of system components. We reuse the Component-Port-Connector (CPC) model presented in [68] with message passing-based communication primitives for the present work. The aim is to target both components and communication channels involved in the system interactions. It provides a recognized way of visualizing the system's structural and behavioral aspects, constituting the "component view" of this meta-model, as depicted in Figure 4.4. The structural elements associated with this meta-model are defined as follows [**NB:** Other communication policies or channels can be added according to the target system and modeling needs]:

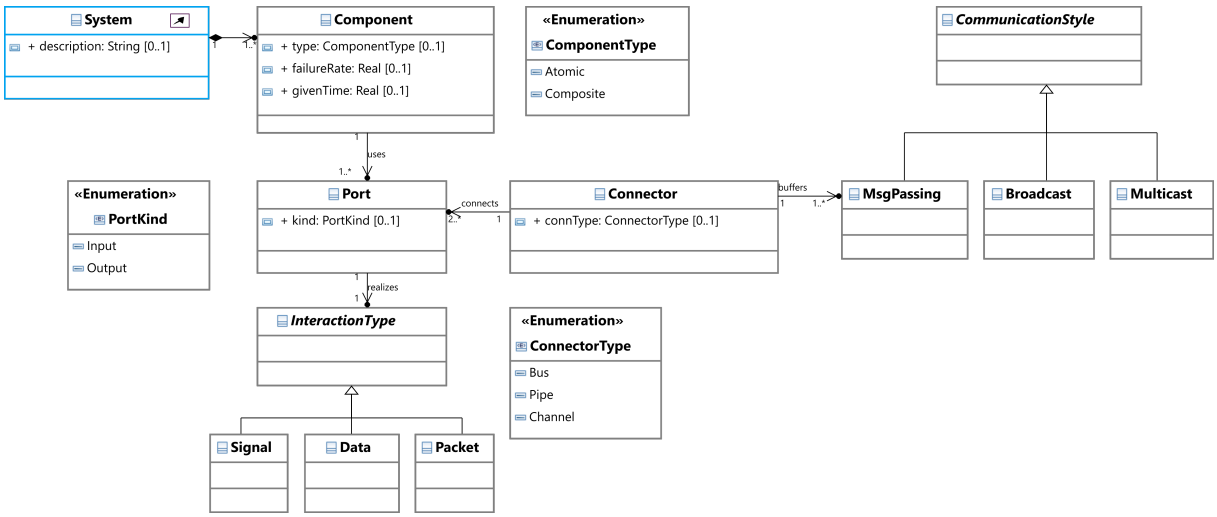


Figure 4.4: Component-layer System Meta-model.

- **System:** It represents the system composed of different components, including physical and logical [**NB:** This element is inherited from the mission-layer system meta-model and extended by adding details to make it suitable for the component layer].

- **Component:** It represents the self-contained computation elements or physical entities, like the camera, processing unit, and brake actuators, constituting the system, i.e., the CDV.
  - *type:* It represents the type of the component, typed as **ComponentType**. The possible values include: *Atomic* or *Composite*. The former are characterized by the basic functionalities they provide independent of the other system components. On the other hand, the latter rely on other components for their functionalities.
  - *failureRate:* It represents the rate of failure [30] of a component, in case the element represents a physical component.
  - *givenTime:* It represents the current time under observation.
- **Port:** Each component uses different ports that provide an interface for the exchange of data or alike between the components. A port is an interaction point to establish communication between a component and its environment comprising other components.
  - *kind:* It represents the kind of the port, typed as **PortKind**. The possible values include: *Input* or *Output*. An input port is capable of receiving the data or alike. On the other hand, an output port can send the data or alike.
- **Connector:** A connector establishes a connection between two or more components via ports for communication.
  - *connType:* It represents the type of the connector, typed as **ConnectorType**. The possible values include: *Bus*, *Pipe*, or *Channel*. A bus represents the wired communication link between the physical components. A pipe is a connector that serves the data stream being transformed by the components acting as filters. A channel represents the logical communication link between components.
- **InteractionType:** It determines the deployment characteristics of ports and connectors within a system to represent the type or semantics of interaction between the components. The type of ports and connectors constrain the semantics of interaction between the components of the system. An interaction may involve the transmission of signals (**Signal**), data (**Data**), or packets (**Packet**).

The behavioral elements associated with this meta-model are defined as follows:

- **CommunicationStyle:** It represents a specific communication behavior between the components that may be senders or receivers. We consider message passing-based

communication primitives (**MsgPassing**) as the communication style. Other possible communication styles may include: broadcast (**Broadcast**) or multi-cast (**Multicast**). The language can be enriched with the elements and semantics to allow the integration of different communication styles and more technology-dependent models at the level of component architecture design.

- **MsgPassing:** It represents the transmission of messages from the sender component to the receiver component via sending and receiving actions, respectively.

4.4.2 Relationships between the Layered Meta-models

The first by-construction linking between the layers is the set of inherited elements, namely **System** and **Operator** between the mission and the functional layers, and **System** between the mission and the component layers. Referred inherited elements ensure a direct and correct structuring relying upon standard UML extension mechanisms. We additionally define a structural and semantic linking via association between the layered elements to ensure consistency during the passage of knowledge between the corresponding layered representations. Accordingly, Figure 4.5 depicts the resulting three-layered system meta-model. This is a choice that simplifies the structuring and configuration of layers. These links are described as follows:

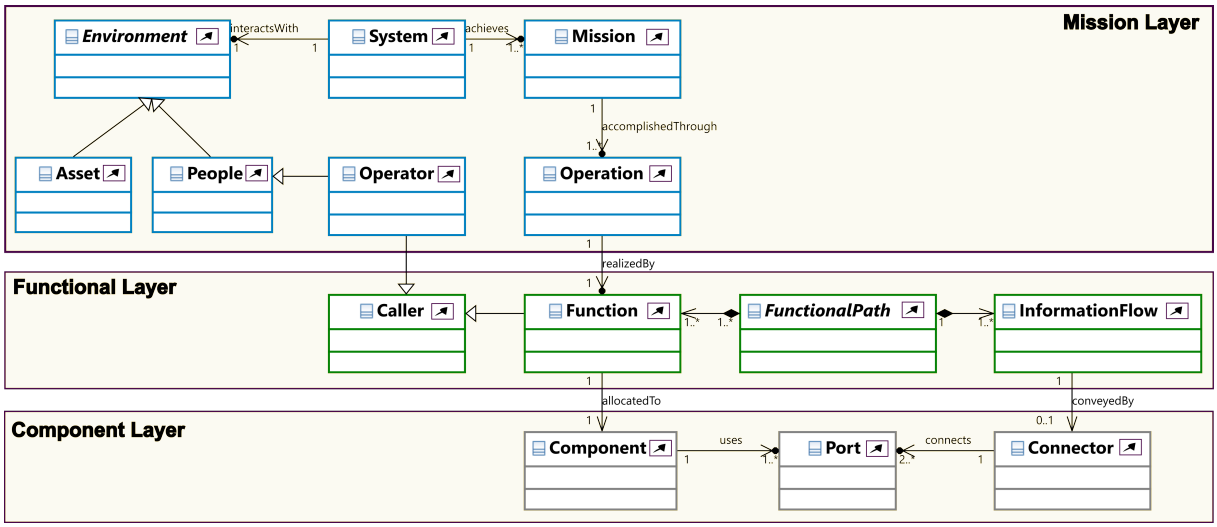


Figure 4.5: Three-layered System Meta-model.

- *realizedBy:* A system operation, like decision-making, at the mission layer, is realized by a function, e.g., position inference, at the functional layer.

- *caller-operator generalization:* A function caller at the functional layer can be an operator of the system at the mission layer. For example, the deceleration function can be manually triggered by a person sitting inside the CDV.
- *allocatedTo:* A function, like position inference, at the functional layer, is allocated to a component, e.g., processing unit, at the component layer.
- *conveyedBy:* An information flow between functions at the functional layer may be conveyed by the connector between the corresponding components at the component layer. According to this allocation, the inputs and outputs associated with the functional interfaces are implicitly mapped to their respective ports.

4.4.3 UML Profiles for the Three-layered System

To support the modeling of a three-layered system, we develop and implement UML profiles from the previously defined concepts in the respective layered meta-models. The profiles reuse the existing concepts from the UML 2.5 [140] and extend them according to the modeling needs. They import both UML Standard Profile and Primitive Types of the Model Library. Herein, we use Eclipse Papyrus [42] that allows the creation of the profiles by providing a UML modeling environment.

4.4.3.1 UML Profile for the System Mission Layer

The UML profile for the system mission layer defines all the necessary stereotypes to model the system missions and the related notions presented in the meta-model for the system mission layer (refer to Section 4.4.1.1). Figure 4.6 depicts the complete profile, which is detailed in the following paragraph.

As mentioned previously, a target SUD—System Under Design aims to achieve a set of missions that are, in turn, accomplished through a set of operations. Therefore, we introduce a set of stereotypes, like *«System»*, *«Mission»*, *«Operation»*, extending the UML meta-class *Class*. Likewise, the stereotype *«Environment»* extends the meta-class *Class* to capture the people and the assets. The attributes like *description*, *operationalSituation*, *hazardousEvent*, and *preCondition* are considered as *String*, as their values are captured in the textual form. The *durationType* attribute of an operation is represented by the tagged value *durationType*, whose values are provided by the enumeration *«DurationType»*.

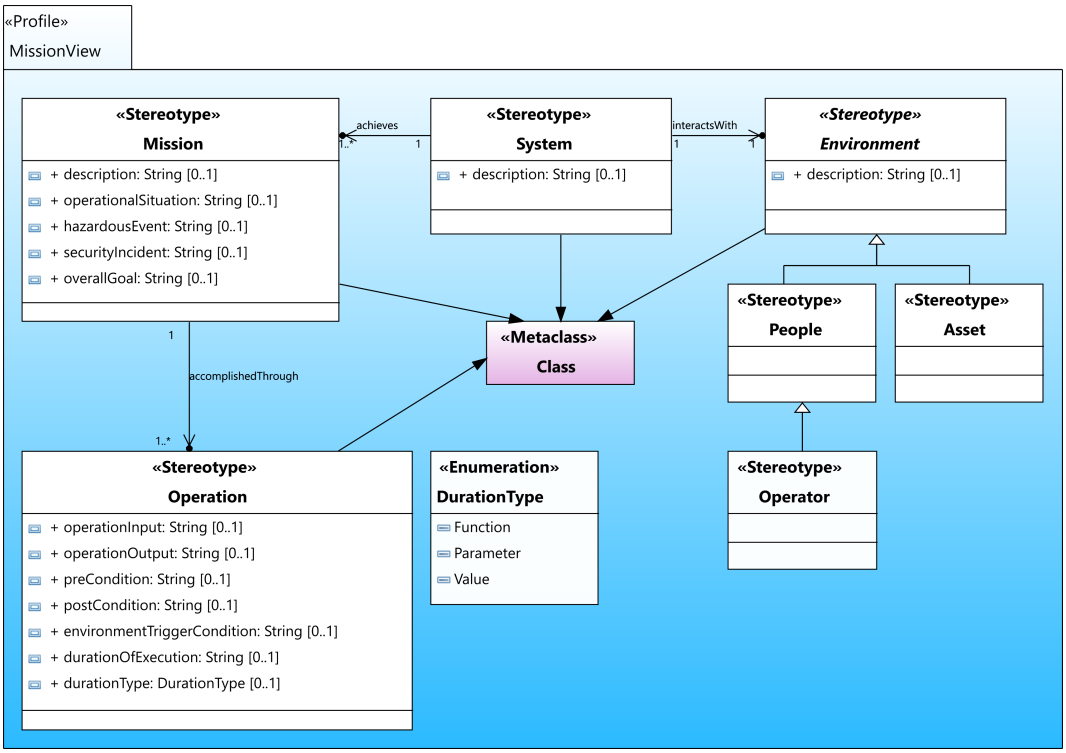


Figure 4.6: Mission-layer System UML Profile.

4.4.3.2 UML Profile for the System Functional Layer

The UML profile for the system functional layer incorporates all the necessary stereotypes to model the system functionality and the related notions presented in the meta-model for the system functional layer (refer to Section 4.4.1.2). Figure 4.7 depicts the complete profile, which is detailed in the following paragraph.

The tasks performed by the system are modeled by the *«Function»* stereotype, extending the UML meta-class *Action* that captures both caller (via a UML Call Behavior Action) and callee (via a UML Opaque Action) functions. The attributes *type* and *triggerType* of a function are defined as tagged value enumerations to respectively define the type of the function and the trigger associated with it. The type of these attributes is *«FunctionType»* and *«TriggerType»*, respectively. The attribute *preCondition* representing the conditions necessary for the function's invocation in textual form is typed as *String*. The functional interfaces are modeled with the stereotype *«FunctionalInterface»*, which extends both *Parameter* and *Interface* meta-classes. Likewise, the function caller is typed via the stereotype *«Caller»*, extending the meta-class *Class*. To model the input and output information respectively accepted and produced by the function, we define the stereotypes *«FunctionalInput»* and *«FunctionalOutput»*, respectively extending the *InputPin* and *OutputPin* meta-classes. To model the link between functions

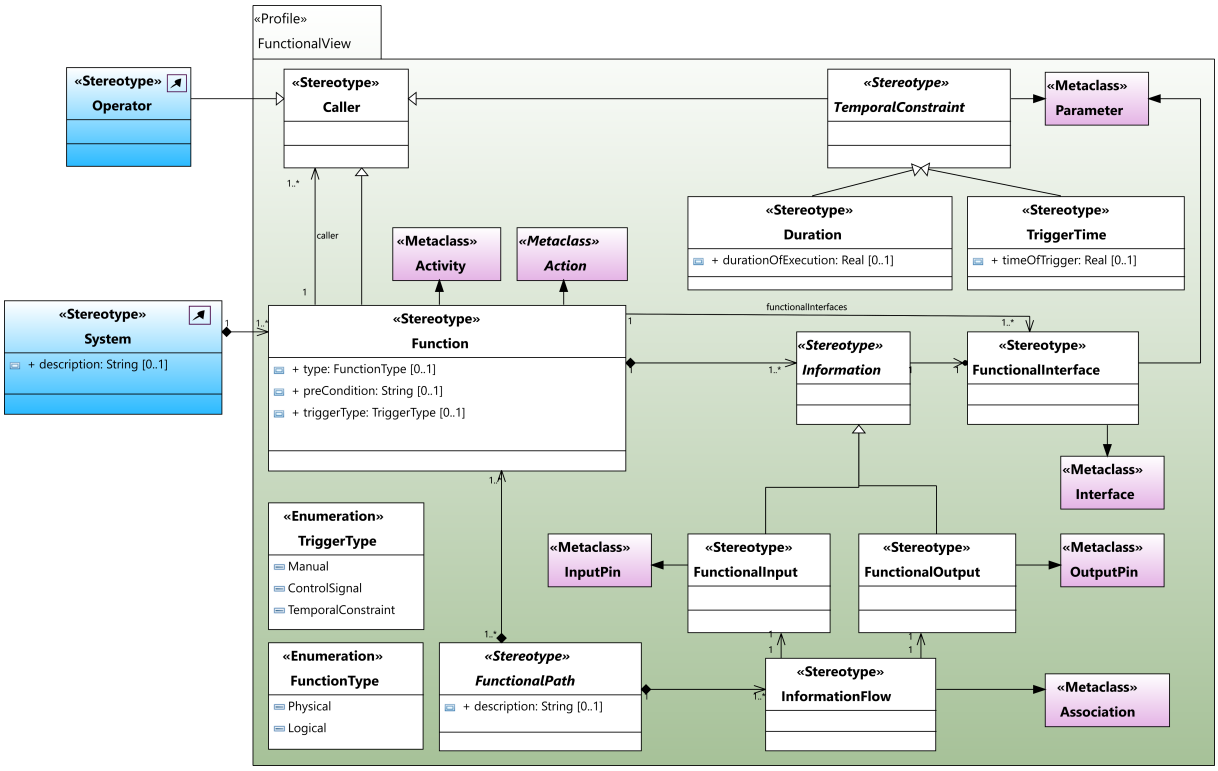


Figure 4.7: Functional-layer System UML Profile.

corresponding to the flow of information across them, we define *«InformationFlow»* stereotype, extending the UML meta-class *Association*. Finally, the temporal constraints representing the function's trigger are captured using the stereotype *«TemporalConstraint»*, extending the meta-class *Parameter*. The values for *durationOfExecution* and *timeOfTrigger* attributes are typed as *Real*.

4.4.3.3 UML Profile for the System Component Layer

The UML profile for the system component layer defines all the necessary stereotypes to model the CPC-based system architecture and the related notions presented in the meta-model for the system component layer (refer to Section 4.4.1.3). Figure 4.8 depicts the complete profile, which is detailed in the following paragraph.

The *«Component»*, *«Port»*, and *«Connector»* stereotypes, respectively extend the UML meta-classes *Component*, *Port*, and *Connector*. Their respective types are captured using the enumeration tagged values, stereotyped as *«ComponentType»*, *«PortKind»*, and *«ConnectorType»*. The possible interaction types, viz. signal, data, and packet, between the components, are stereotyped using *«InteractionType»*, extending the meta-class *Class*.

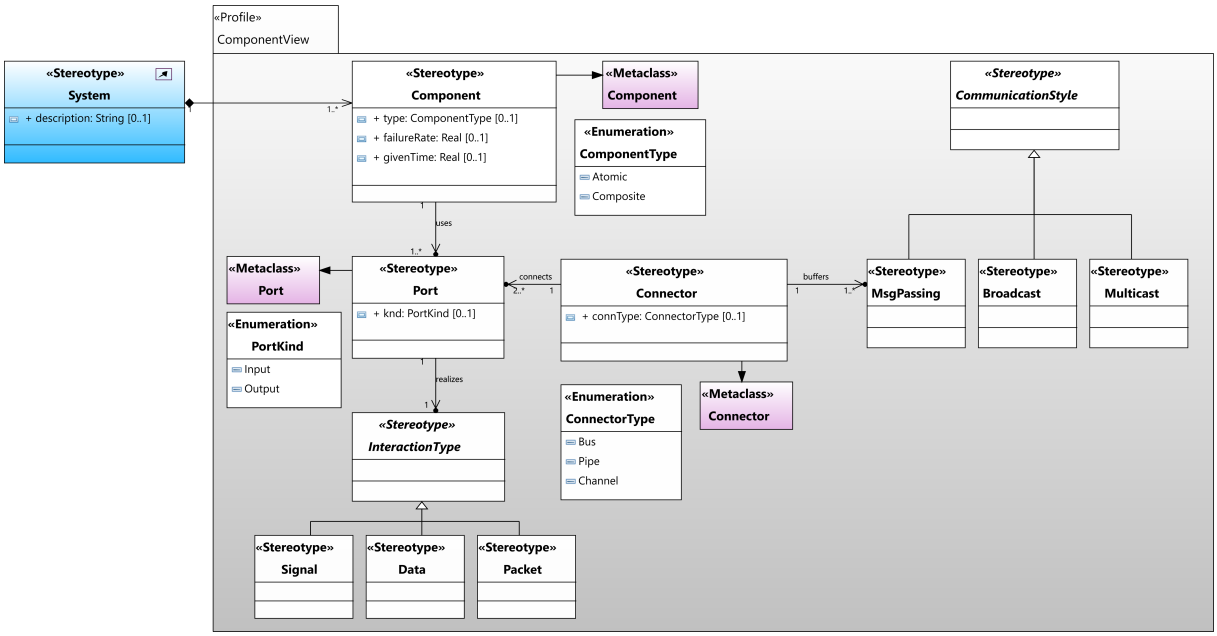


Figure 4.8: Component-layer System UML Profile.

Likewise, the communication styles defining the behavior of communication between the components are captured by *«CommunicationStyle»*, extending the meta-class *Class*.

4.4.3.4 UML Profile for the Layered Meta-models Relationships

Based upon the structural and semantic linking defined in Section 4.4.2, the resulting UML profile targeting the relationships between the layered meta-models is depicted in Figure 4.9. Herein, the links are defined via associations between the respective elements. Since there is a one-to-one correspondence between the semantic meaning of instantiated associations in the profile and their respective counterparts in the meta-model in Figure 4.5, no further description is provided.

4.5 Safety and Security Properties Modeling

In this section, we describe the DSML to facilitate the modeling of safety and security properties and their interplay concerning the three-layered system architecture. For this, we propose meta-models to define the concepts, their attributes, and relationships pertaining to the properties at different layers of system representation. Furthermore, we propose the UML profiles to implement the proposed meta-models.

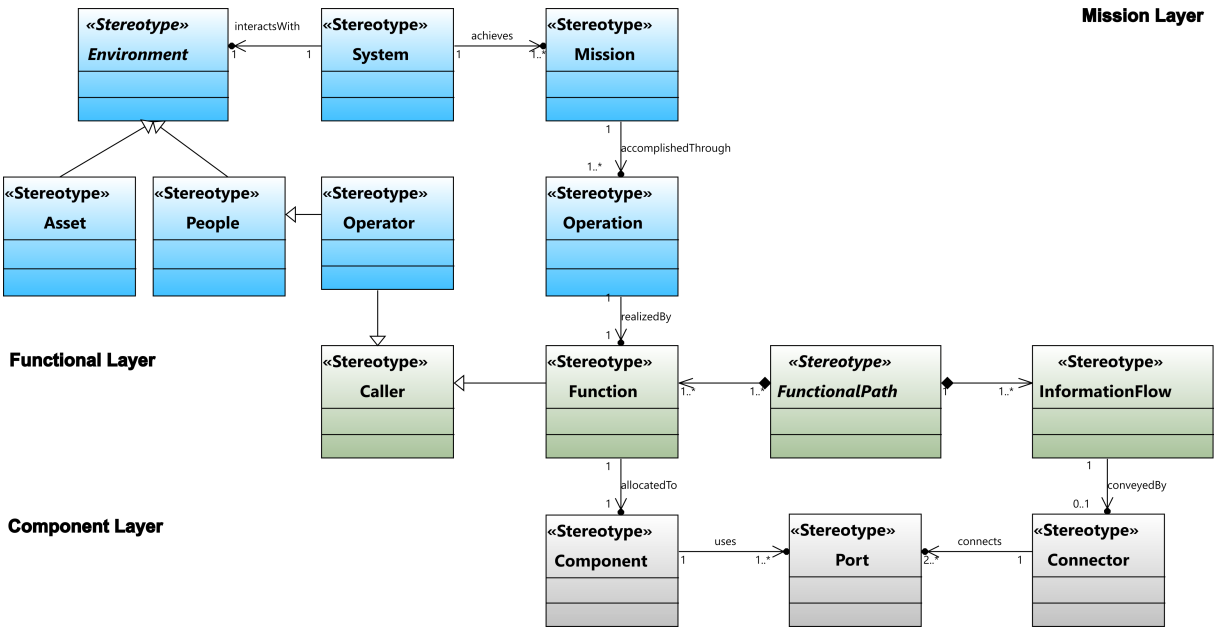


Figure 4.9: Three-layered System UML Profile.

4.5.1 Meta-models for Safety and Security Properties

The meta-models corresponding to safety and security properties at each layer of system representation are described in the following sub-sections. The set of safety and security properties characterize our understanding of those concerns for the SE process. This is indeed the result of a selection conducted after a thorough review of the literature in Chapter 2, which finally produced distinctive properties often referred to address and cope with safety and security events. The properties have been adapted to the syntax and semantics captured in the different meta-models and profiles at each layer.

4.5.1.1 Meta-model for the Properties at Mission Layer

The meta-model for the properties at the mission layer incorporates concepts related to safety and security properties of the target SUD at this layer. The "property view" offered by this meta-model extends the "mission view" presented in Figure 4.2 and is depicted in Figure 4.10 (see top part: *Mission Layer*). **PropertyCategoryLibrary** represents the reusable model libraries, defining high-level properties of the system.

Furthermore, the "property view" is extended by the "property category view", as depicted in Figure 4.11 (see top part: *Mission Layer*). **PropertyCategory** represents the classification of safety and security objectives. An *objective* represents the desired or positive feature a system should have. For example, in the safety context, **OperationalAvailability** is defined as an

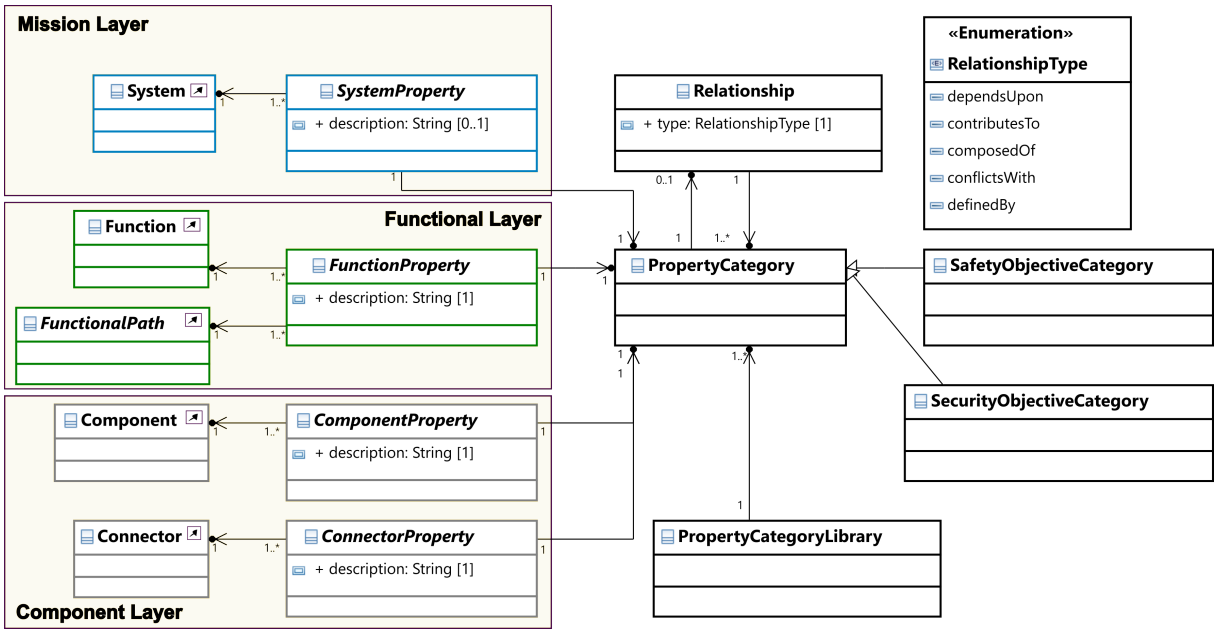


Figure 4.10: Properties Meta-model: PropertyView.

objective within the **Availability** category. Extended the notion of availability from the standard ISO 26262 [30], as presented in Chapter 1, the operational availability objective is defined as follows:

Operational Availability (Prop1). A system in a certain operational situation should allow for the realization of safety-critical operation(s), whenever a hazardous event is detected.

Herein, the readiness of the system in terms of completion of the operations assigned to it, is of significant importance to guarantee distinct missions in safety-critical systems, e.g., obstacle detection in autonomous CDVs to avoid a crash. Likewise, in the security context, **SystemAccessibility** is defined as an objective within the **ControlledAccessibility** category. According to this,

System Accessibility (Prop2). A system must allow and limit the access of safety-critical operations to only authorized entities.

The semantics behind these objectives will be detailed in Chapter 5. The set of objectives considered in this work represents the pragmatical choices inspired by the literature, as introduced in Section 1.1.2, concerning system safety and security. The libraries are subsequently used as external models for capturing the objectives as signatures.

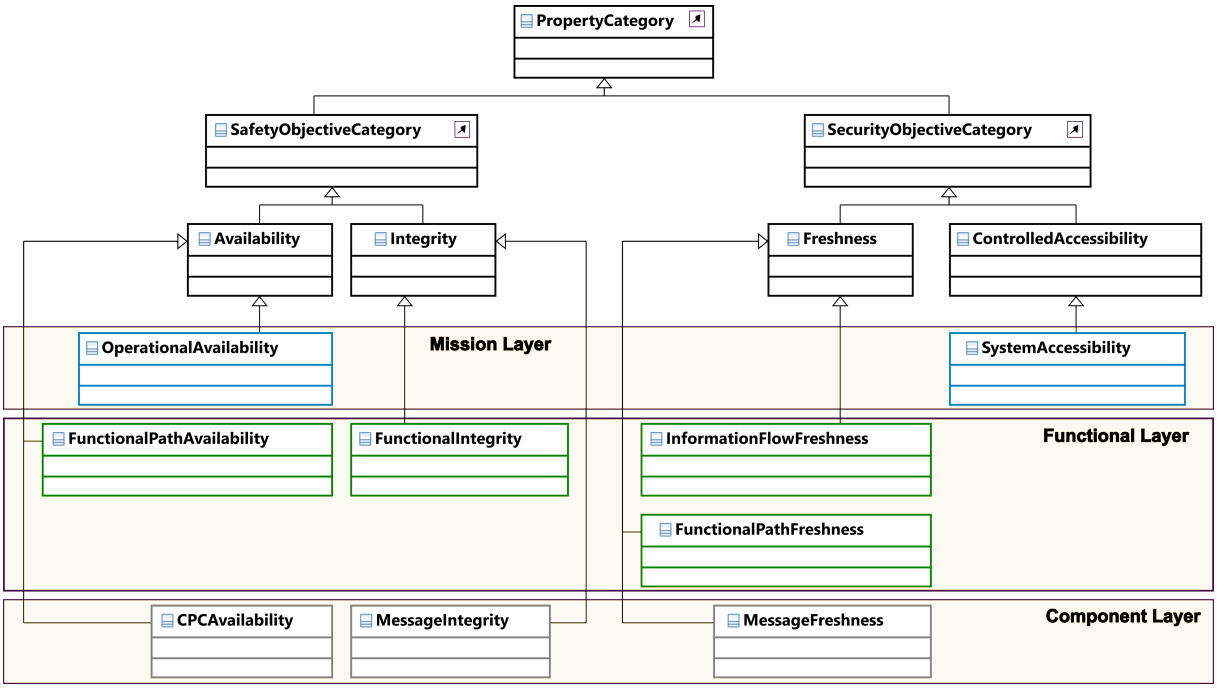


Figure 4.11: Properties Meta-model: PropertyCategoryView.

4.5.1.2 Meta-model for the Properties at Functional Layer

The meta-model for the properties at the functional layer incorporates concepts related to safety and security properties of the target SUD at this layer. The "property view" offered by this meta-model extends the "functional view" presented in Figure 4.3 and is depicted in Figure 4.10 (see middle part: *Functional Layer*). It enriches the **PropertyCategoryLibrary** by defining high-level properties essentially concerning the system functions and functional path.

Furthermore, the "property view" is extended by the "property category view", as depicted in Figure 4.11 (see middle part: *Functional Layer*). In the safety context, we consider two objectives, namely **FunctionalIntegrity** and **FunctionalPathAvailability**, belonging to the **Integrity** and **Availability** categories, respectively.

Functional Integrity (Prop3). It ensures the preservation of the information flow between orderly execution of functions, e.g., obstacle detection, position inference, and deceleration, constituting a functional path.

Functional Path Availability (Prop4). It ensures that a sequence of functions constituting a functional path should globally complete their execution within a required temporal bound.

Likewise, in the security context, **FunctionalPathFreshness** and

InformationFlowFreshness are considered as two objectives within the **Freshness** category.

Functional Path Freshness (Prop5). It ensures the temporal consistency of the execution of functions across a functional path.

Information Flow Freshness (Prop6). It ensures the temporal consistency of the information flow across a functional path, as a specific case of functional path freshness.

The semantics behind these properties will be detailed in Chapter 5.

4.5.1.3 Meta-model for the Properties at Component Layer

The meta-model for the properties at the component layer incorporates concepts related to safety and security properties of the target SUD at this layer. The "property view" offered by this meta-model extends the "component view" presented in Figure 4.4 and is depicted in Figure 4.10 (see bottom part: *Component Layer*). It enriches the **PropertyCategoryLibrary** by defining high-level properties of the system components and connectors.

Furthermore, the "property view" is extended by the "property category view", as depicted in Figure 4.11 (see bottom part: *Component Layer*). In the safety context, we consider two objectives, namely **CPCAvailability** and **MessageIntegrity**, belonging to the **Availability** and **Integrity** categories, respectively.

CPC Availability (Prop7). It ensures that the components, like the sensor and processing unit, must be able to engage in communication for the transmission of critical messages, comprising the position of the obstacle—for instance.

Message Integrity (Prop8). It ensures that whenever a component sends some message, the same message will eventually be received by the intended component.

Likewise, in the security context, **MessageFreshness** is considered as an objective within the **Freshness** category.

Message Freshness (Prop9). It ensures the temporal validity of the message in transmission across the system components.

For safety-critical CDV systems, it may act as an indicator of timely status updates. The semantics behind these properties will be detailed in Chapter 5.

4.5.2 Properties Interplay Modeling

Potential relationships between the objectives belonging to the safety domain (intra-relationships), security domain (intra-relationships), and both safety and security domains (inter-relationships) may arise within a layer or across layers. Thus, we aim to establish an alignment via links between them to support further analysis, including safety and security interplay. We define associations, like *dependsUpon*, *definedBy*, and *conflictsWith*, between the objectives (refer to Figure 4.10). These are similar to the ones used by the pattern community to define pattern relationships, like in [70], and are specified in **RelationshipType**, with the description provided as follows:

- *dependsUpon:* An objective $\mathcal{A}$ depends upon another objective $\mathcal{B}$ implies $\mathcal{B}$ needs to be satisfied for the fulfillment of $\mathcal{A}$.
- *contributesTo:* An objective $\mathcal{A}$ contributes to another objective $\mathcal{B}$ implies fulfillment of $\mathcal{A}$ strengthens or complements the fulfillment of $\mathcal{B}$.
- *composedOf:* An objective $\mathcal{A}$ is composed of another objective $\mathcal{B}$ if $\mathcal{B}$ constitutes $\mathcal{A}$.
- *conflictsWith:* An objective $\mathcal{A}$ conflicts with another objective $\mathcal{B}$ if their joint consideration leads to conflicting situations or inconsistencies.
- *definedby:* An objective $\mathcal{A}$ is defined by another objective $\mathcal{B}$ if $\mathcal{A}$ semantically depends upon $\mathcal{B}$.

In essence, the links among the objectives are specified based upon the structural and semantic linking in the three-layered system model, thus enriching our approach. Moreover, they provide the means to model dependencies and conflicts between objectives. To clarify the referred interplay, some instances are explained in the following paragraphs.

For example, consider the link *accomplishedThrough* in Figure 4.5 that associates one mission to more than one operation. This link is followed by the link *realizedBy* that associates one operation to one function. Given the notion of availability, whenever the property is demanded at the mission layer for a set of operations to accomplish a mission, the property also needs to be satisfied at the functional layer for a sequence of functions constituting a functional path. The fact that the property is verified at the mission layer does not necessarily imply its fulfillment at the functional layer, as additional details, like time and semantics, are incorporated for the function execution that need to be verified. Thus, we create a link *dependsUpon* between the **OperationalAvailability** and **FunctionalPathAvailability** objectives. Accordingly, if a mission is accomplished through a sequence of operations that are, in turn, realized by a set

of functions constituting a functional path, the fulfillment of the availability of the operations finally depends upon the availability of the functional path.

Similarly, in the same Figure 4.5, consider the link *allocatedTo* that associates one function to one component, followed by the link *conveyedBy* that associates an information flow to at most one connector, and the implicit association between the functional interface and port. Given the notion of availability, whenever the property is demanded at the functional layer for a functional path, the property also needs to be satisfied at the component layer for components, ports, and connectors involved. The fact that the property is verified at the functional layer does not necessarily imply its fulfillment at the component layer due to constraints, like the memory register's size, bus speed, and computational throughput, introduced by the underlying hardware that executes the function and demands further analysis. Therefore, we create a link *dependsUpon* between **FunctionalPathAvailability** and **CPCAvailability** objectives. Accordingly, if a sequence of functions and information flows between them constituting a functional path are allocated to a set of components and conveyed by the connectors, respectively, the fulfillment of the functional path availability objective depends upon the fulfillment of the CPC availability objective.

Likewise, the **SystemAccessibility** objective at the mission layer *conflicts with* the **OperationalAvailability** objective at the same layer in certain cases. Table 4.1 captures the pair of objectives and their relationships identified for the safety and security interplay. Among others, this table is the first summary that helps characterize the interplay between properties in two axes: properties at the same layer (intra-layer) and properties between layers (inter-layer).

Table 4.1: Relationships between Objectives within or across the Layers.

Layers	Objective 1	Relationship	Objective 2
Mission	SystemAccessibility	*conflictsWith*	OperationalAvailability
Mission-Functional	OperationalAvailability	*dependsUpon*	FunctionalPathAvailability
Functional	InformationFlowFreshness	*definedBy*	FunctionalIntegrity
	FunctionalPathFreshness	*definedBy*	FunctionalIntegrity
	FunctionalPathAvailability	*definedBy*	FunctionalIntegrity
Functional-Component	InformationFlowFreshness	*dependsUpon*	MessageFreshness
	FunctionalPathFreshness	*dependsUpon*	MessageFreshness
	FunctionalPathAvailability	*dependsUpon*	CPCAvailability
	FunctionalIntegrity	*dependsUpon*	MessageIntegrity
Component	MessageFreshness	*definedBy*	MessageIntegrity

4.5.3 UML Profiles for Safety and Security Properties

To facilitate the modeling of the safety and security properties at the three layers presented in Section 4.4, we consider all the notions therein defined. To do so, we develop and implement UML profiles from the concepts defined in the properties meta-models at the respective layers. Specifically, we introduce stereotypes like *«SystemProperty»*, *«FunctionProperty»*, *«ComponentProperty»*, extending the UML meta-class *Class*. The stereotype *«PropertyCategoryLibrary»* models the library of safety and security properties. The corresponding objectives are captured using stereotypes like *«OperationalAvailability»* and *«MessageFreshness»*, which extend the stereotypes *«SafetyObjectiveCategory»* and *«SecurityObjectiveCategory»*.

Likewise, to model safety and security interplay, the relationship among the properties is specified by the *«Relationship»* stereotype, extending the UML meta-class *Class*. The actual values of relationship type are defined in the enumeration *«RelationshipType»* as a tagged value. The complete profile description is illustrated in Figures 4.12 and 4.13.

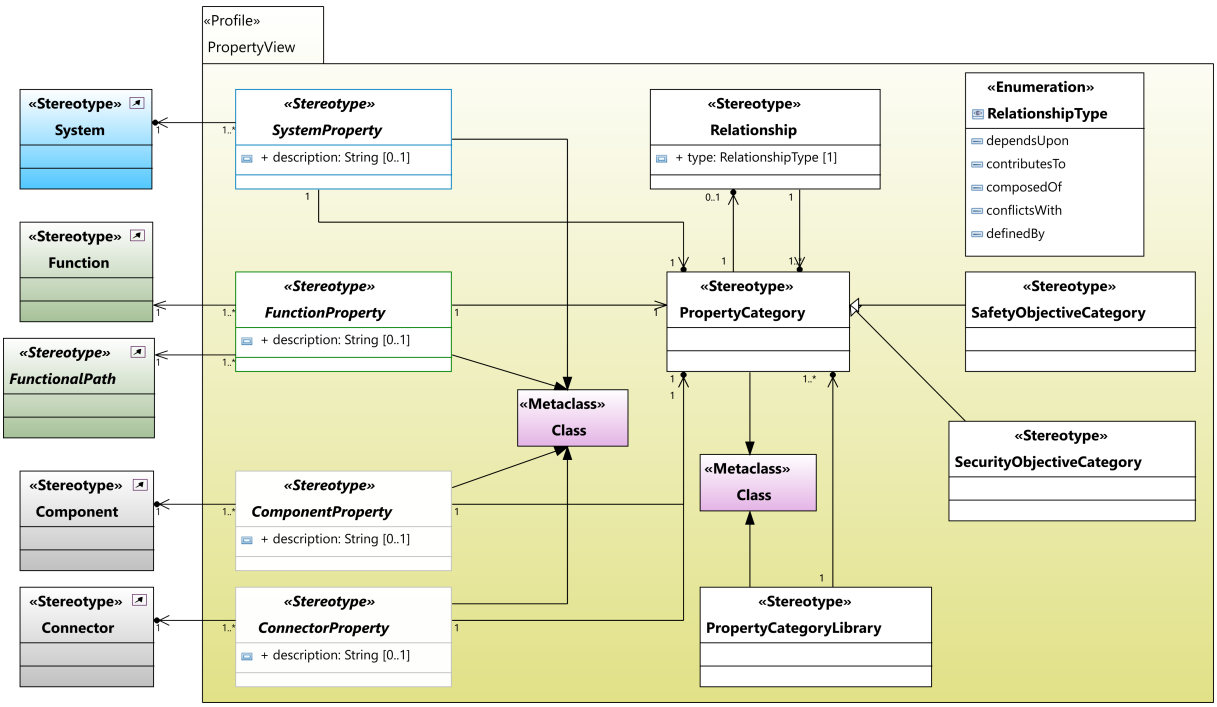


Figure 4.12: Properties UML Profile: PropertyView.

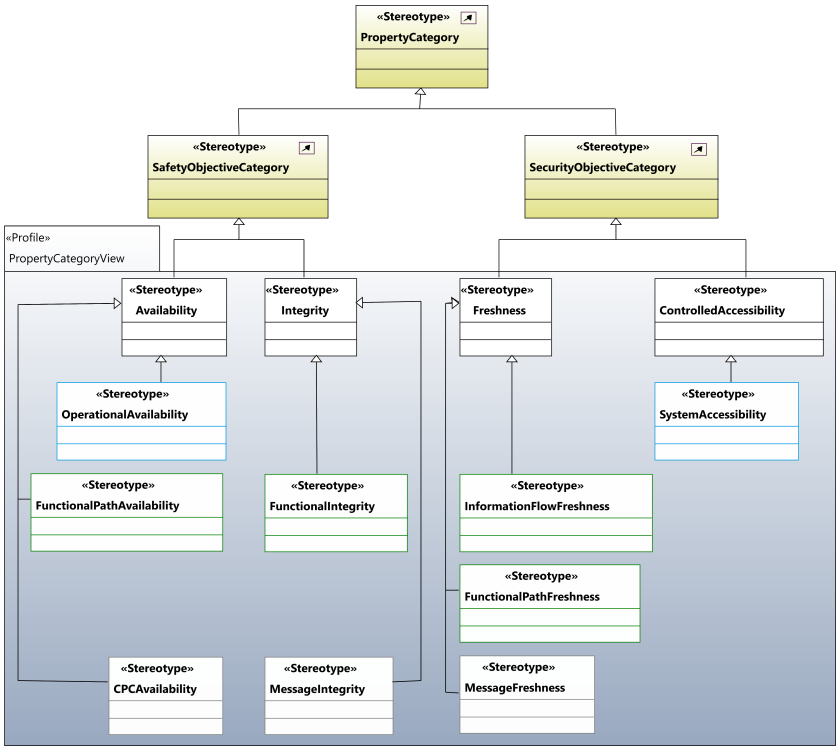


Figure 4.13: Properties UML Profile: PropertyCategoryView.

4.6 Tool Support for Modeling

In this section, we propose tool support relying upon MDE paradigm, supporting the DSMLs presented in the previous Sections 4.4 and 4.5 to assist the architects and designers of safe and secure systems. We begin by outlining the essential requirements that the underlying tools must fulfill to support the modeling aspects of the proposed approach, followed by the one used in our work.

4.6.1 Tool Support Requirements

The tool support for modeling aspects in our approach is designed to primarily facilitate the DSML meta-modeling and profiling tasks. As mentioned in Section 4.3, we rely upon semi-formal constructs offered by UML in this regard. Accordingly, the underlying tools must fulfill the following key requirements:

- Enable the creation of the UML meta-models for describing the three-layered system and safety and security properties.
- Enable the creation of customized UML profiles corresponding to the three-layered system

and safety and security properties meta-models.

- Enable the creation of the three-layered system model, describing the target SUD according to the meta-model.
- Support the reuse of the safety and security properties model libraries while creating the target system model.

Any tool that can fulfill these requirements can be used to conduct the system and properties modeling tasks. In our case, we used Eclipse Papyrus [42] amongst the existing alternatives; the key features of which leveraged in this work are described in the following sub-section.

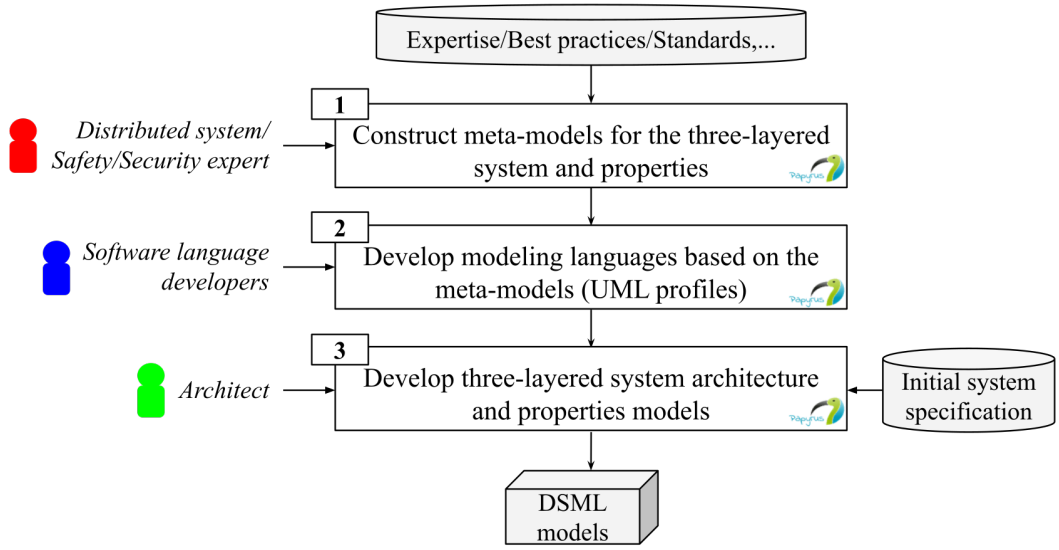


Figure 4.14: Tool Support for Modeling.

4.6.2 Modeling Environment: Eclipse Papyrus

For conceptual meta-modeling, profiling, and modeling of the three-layered system and safety and security properties involved in our approach, we rely upon Papyrus, a UML and Systems Modeling Language (SysML) modeler developed as an open-source Eclipse project [42]. Salient features it offers include full UML 2.5 support, model validation, and code generation. The consistency between the layered DSMLs is ensured by the extension mechanisms inherited from UML. Models' correctness can be checked via extensions of Papyrus validation features.

Specifically, the system and properties modeling aspects in the proposed approach incorporate the following steps concerning the tool support (see Figure 4.14): 1) Construction of the meta-models for the three-layered system and properties by the respective domain experts,

2) Based on the resulting meta-models, development of the DSML profiles by the software language developers, and 3) Development of the three-layered system architecture and properties models by the system architect in conformity to the profiles.

The first two steps (i.e., Steps 1 and 2) are performed once for a set of domains. The inputs of these steps are expertise, standards, and best practices from the system, safety, and security experts. Step 3 is performed once per application domain. Performing Step 2 requires knowledge of the system and software language engineering, whereas Step 3 requires knowledge of the system development process for a specific application domain. The input for Step 3 is the initial system specification (e.g., requirements).

4.7 Conclusion

Summary. In this chapter, we addressed the *problematic* **P3** related to the integration of system safety and security at the design phase with the aim of achieving an approach for co-engineering. To this end, we focused on the modeling aspects of the system and properties via a three-layered representation, comprising high-level mission, functional, and detailed component views. The design is conducted at the semi-formal language level, relying upon generic, technology-agnostic, and standardized UML constructs to create system and properties DSMLs. The DSMLs are first defined as meta-models, then implemented as profiles. The former allows capturing the domain-oriented concepts and their relationships, enabling system architects to understand the language and detect potential inconsistencies. At the same time, the latter is built by stereotyping the domain modeling concepts and mapping the same to UML meta-classes. The mapping from meta-models to profiles is mainly conducted by a review of the UML standard, a selection of the meta-classes semantically aligned with the notions in the meta-models, and, finally, the extension of the selected meta-classes to accordingly incorporate the specificities of the system model, at the three layers, and the fundamental attributes capturing the safety and security properties at stake for the analysis foreseen.

Usefulness. So far, model-driven constructs allow addressing the incorporation of safety and security concerns in two manners and respective axes. First, MDE is used to decouple the model of a target system into different layers according to the notions and purposes pertaining to the phases of the engineering process, namely for the mission, functional, and component design. Secondly, a set of properties that characterize safety and security concerns, mostly borrowed from the literature, are specialized and integrated into each layer so that properties belonging to the same category consistently cascade down across the three-layered model and are referred to concerning specific elements. All these constructs provide a basis upon which

properties analysis can be conducted, e.g., for verification, conflict solving, etc. The DSMLs ensure separation of concerns as the system and properties are decoupled, thereby simplifying the modeled reality. Engineers can use the same in familiar environments to model safety and security interplay.

Concluding Remarks. It is noteworthy that an objective from a category, e.g., integrity or availability, in a given context may belong to either the safety or security objective categories. The notion of property categories thus does not aim at a strict classification but to assist the designers in foreseeing the aspects that can influence the modeling of safety- and security-related systems. The DSMLs will further encourage the formal reasoning and analysis of the inter-dependencies between safety and security in the early design stages of the SE process, as pursued in the forthcoming Chapter 5.

Chapter 5

Formalization for Specification and Verification

Contents

5.1 Introduction

This chapter presents another main contribution constituting our proposed approach, which is motivated to address the *problematic* **P4** related to error-prone design-level properties' verification due to ambiguous specifications, as mentioned in Section 1.2. The Domain-Specific Modeling Language (DSML) profiles presented in the previous Chapter 4 offer semi-formal semantics, thus supporting certain analyses. However, as mentioned in Chapter 1, the absence of precise semantics may lead system developers to infer the aspects in different ways that may prevent them from bringing consensus in the system design and coherent modeling regarding safety and security properties. To this end, we present the formalization of the DSML profiles for rigorous specification and analysis of the system and safety and security properties in unison. The aim is to consider the complexities associated with using formal-based languages and

techniques in the System Engineering (SE) process and improve the approach usability by automatizing properties verification. Accordingly, we begin with the logical specification for the DSMLs belonging to the three-layered system and properties model, relying upon set theory notions [141]. This is followed by the formal logic-based specification of the safety and security properties and their interplay. For this, we use First-Order Logic (FOL) [44] and Modal Logic [45] to create technology-independent formalisms. The former allows describing the system's state via attribute variables, while the latter captures the necessity and possibility related to the change of the state. Finally, using these formalisms, we provide an interpretation of the models to the language, namely Event-B [46], of a formal-based tool, namely Rodin [47], for model checking and formal verification. For illustration purposes, we rely on the use case scenario of Connected-Driving Vehicles (CDVs) introduced in Section 3.4, which will be detailed in Chapter 6.

Overall, the chapter is organized as follows: In Section 5.2, we describe the method to define the three-layered system and properties formalisms to leverage them and address safety and security interplay. In Section 5.3, we describe the desired features of the formalisms and the languages and techniques used to support them. This establishes the ground for our contribution towards integrated safety and security analysis. In Section 5.4, we introduce set theory-based logical specification for the three-layered system and properties DSMLs. Subsequently, we present the formal logic-based specification of the safety and security properties and their interplay concerning the three-layered system. Furthermore, using the formalisms as interpretation rules, in Section 5.5, we provide an interpretation of the DSMLs to the Event-B language to obtain a more concrete specification. In Section 5.6, we present the convention to build a concrete architecture of the target system in Event-B, relying upon the abstract system model. In Section 5.7, we present the details of the tool support used in this work to mechanize model checking and formal verification. Finally, we conclude by summing up the contribution of the chapter in Section 5.8.

5.2 Method to Defining the Formalisms

This section describes the method for defining the system and safety and security properties formalisms. Recalling Figure 3.7, this involved several iterations for analyzing the system aspects and properties concerning the three-layered model, as depicted in Figure 5.1. To this end, the fundamental notions were extracted from an extensive state-of-the-art study presented in Chapter 2. The candidate formal modeling languages were selected based on the desired features listed in Section 5.3 to build the formalisms. These formalisms evolved by consolidating the state-of-the-art notions and feedback received to debug the DSMLs during this research work.

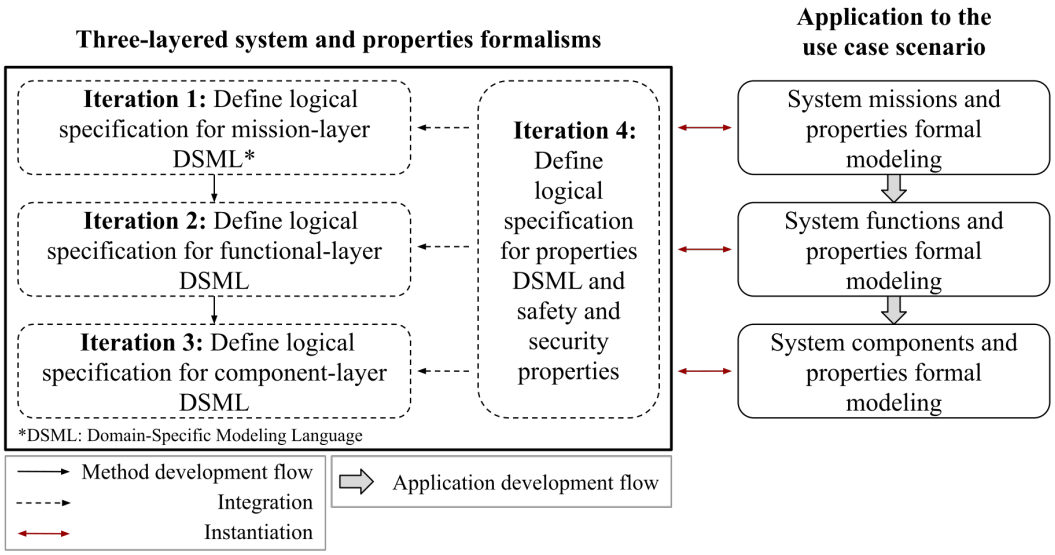


Figure 5.1: Method for Defining the Formalisms.

The iterations are detailed as follows:

- **Iteration 1:** This iteration is related to defining the formalism for the mission-layer system DSML. It begins with mapping the mission-layer modeling language to a set theory-based logical specification corresponding to the notions described therein. This is followed by interpreting the respective profile using the logical specification in Event-B.
- **Iteration 2:** In this iteration, we conduct formalization concerning the system design at the functional layer. Accordingly, we define formalism corresponding to the functional-layer system DSML, which includes mapping to the set theory-based logical specification and interpreting to the Event-B language.
- **Iteration 3:** In this iteration, we conduct formalization concerning the detailed technical design of the system at the component layer, comprising components and communication channels. Accordingly, we define formalism corresponding to the component-layer system DSML, which includes mapping to the set theory-based logical specification and interpreting to the Event-B language.
- **Iteration 4:** In this iteration, we define the set theory-based logical specification corresponding to the properties DSML. This is followed by specifying safety and security properties using logic-based notions. The properties signatures are instantiated considering the target system model at different layers.

The iterations are accompanied by an illustration involving a primary and modeled instance

of the CDV use case borrowed from the state-of-the-art (refer to Section 3.4) to analyze a safety- and security-critical scenario relying upon the formal notions. Thus, prior to the iterations, we consider the three-layered system and properties DSMLs from previous Chapter 4 as input, targeting the set of safety and security properties considered therein and analyze the system model concerning those properties by conducting proofs, which will be detailed in Chapter 6.

5.3 Desired Features for the Formalisms

The overall aim is to allow unambiguous and precise specification of the modeled system and properties, facilitating integrated safety and security analysis of the target system. To this end, the desired features of the formalisms are described as follows:

- *Expressiveness.* The formalisms should be generic enough to capture the different viewpoints of the target system model. In our context, we materialize it as per the different stages of system development, viz, *Mission*, *Functional*, and *Component* constituting the three-layered system and properties model.

- *Allow integrated safety and security analysis.* The formalisms should provide means to specialize and integrate the safety and security properties across the three-layered model, irrespective of the system design flow, i.e., top-down or bottom-up, for a joint analysis of properties.

- *Technology-independent.* The formalisms should be independent of the technology to provide flexibility regarding the choice of the formal specification language and Validation and Verification (V&V) tool support.

- *Consistency-checking and verifiability.* The formalisms shall support a three-layered system design amenable to the interpretation to an existing tooled-language in a consistent manner. In addition, they must support the translation of the properties specifications to a formal-based framework to conduct sound proofs and automated verification in order to spot inconsistencies and check for design conformity. Essentially, this shall involve constructs and semantics as means to support properties interplay analysis via conflict identification—for instance.

To formally capture the three-layered system model and properties model libraries in a generic way, we use basic *set theory* notions. Moreover, to specify properties in a technology-independent way, we use FOL. As mentioned in Chapter 1, FOL provides enough expressiveness, sound proof theory, and capability to define pre-conditions and post-conditions

associated with the system behavior on abstract types. The generic specification would serve as a foundation to capture the safety and security properties of diverse systems under consideration and their interplay. Similarly, the technology-independent way would prevent technology-specific assumptions while analyzing different use cases. To capture the temporal and ordering aspects associated with the properties covered in this work, we use Modal Logic. Moreover, the Event-B language and the proposed approach share the scope of modeling and reasoning the system as a whole. In order to carry out safety and security analysis on the system model, we rely on theorem proving supported by Event-B via axioms and derivation rules. Indeed this process is iterative and compatible with the refinement process that can be used to introduce more details in the system design pertaining to different modeling layers and verify properties, and allows model instantiation. Nevertheless, any formal language that can support the desired features mentioned above can be used to conduct the formal modeling and analysis of the system and properties.

5.4 Logical Specification of the Three-layered System and Properties

In this section, we propose a logical specification for the three-layered system and properties modeling languages presented in the previous Chapter 4. To this end, we rely upon the basic set theory notions that allow capturing the user-defined types corresponding to the Unified-Modeling Language (UML) profile elements. The logical specification for each layer preserves the associations and types in the respective profiles. In addition, we present the formal logic-based precise specification of safety and security properties and their interplay pertaining to different layers of the three-layered system model. This involves defining a set of elementary properties that serve as the building block to draw up the safety and security properties belonging to different categories in our work. This is followed by FOL and Modal Logic-based formal specification of safety and security properties, with associated preliminaries and semantics for each layer of the three-layered system model. Furthermore, we provide examples demonstrating the interplay between these properties, within or across the layers.

The logical specification being technology-independent would facilitate comprehension and future extensions of the modeling languages towards different formal frameworks and tool support. In addition, as a side effect, instead of directly mapping the profile elements to their Event-B counterparts, by introducing the logical specification, the origin and target elements in the interpretation are better identified and structured in the proofs facilitated by the automated tool support Rodin for properties verification, which will be detailed in Section 5.5.

5.4.1 Interpreting the DSMLs to Set Theory

Herein, we describe the set theory-based logical specification of the three-layered system and properties DSMLs. Once the system aspects, like missions, functions, and components, along with the safety and security properties, are modeled in the language presented in Chapter 4, their corresponding logical specification can be obtained by following the mapping presented in the following sub-sections. For readability purposes, the concept attributes in the respective DSMLs are represented in *Italics*.

5.4.1.1 Mission Layer

Herein, a system S is a tuple comprising a set of missions M_i, $i \in \{1 \dots \mathbb{N}\}$ and a variable set named Environment, the domain of which is the set $\{People, Asset\}$. A mission M_i is a quintuple, comprising a set of operations Op_j, $j \in \{1 \dots \mathbb{N}\}$ and variables that correspond to the attributes of its respective class in the mission-layer system DSML illustrated in Figure 4.2. Likewise, an operation Op_j is a tuple, comprising its respective attribute variables, like *operationInput*, *operationOutput*, etc. Herein, the domain of *durationType* is the set $\{Function, Parameter, Value\}$ that captures operation duration as a function of the input received from other system operations, a parameter describing the overall execution duration for the operation, or a value indicating the maximum execution time of an operation. The complete mapping is described in Table 5.1.

Table 5.1: Mapping: Mission-layer System DSML $\mapsto$ Logical Specification.

DSML Element	Logical Specification
System	S := ($\{M_i\}$, Environment), $i \in \{1 \dots \mathbb{N}\}$
Mission	M_i := ($\{Op_j\}$, *operationalSituation*, *hazardousEvent*, *securityIncident overallGoal*), $i, j \in \{1 \dots \mathbb{N}\}$
Operation	Op_j := (*operationInput*, *operationOutput*, *preCondition*, *postCondition*, *environmentTriggerCondition*, *durationOfExecution*, *durationType*), *durationType* $\in$ {*Function*, *Parameter*, *Value*}, $j \in \{1 \dots \mathbb{N}\}$
Environment	Environment :$\in$ {People, Asset}

5.4.1.2 Functional Layer

The semantics associated with the mapping of the key elements in the functional-layer system DSML depicted in Figure 4.3 to their logical counterparts is described as follows:

- **Information flow:** For a set $\mathcal{F}$ of functions F_k, $k \in \{1 \dots \mathbb{N}\}$, the information flow between two functions F_i and F_j is denoted by (F_i, F_j). An information flow (F_i, F_j) determines an exchange of information between the output O_i of function F_i and the input

I_j of function F_j. Thus, the information flow (F_i, F_j) induces indeed the flow (O_i, I_j). When there is no confusion and to simplify the notation, (F_i, F_j) and (O_i, I_j) can both be used interchangeably to denote the information flow.

- **Functional path:** Given the set of functions $\mathcal{F}$, a functional path denoted by FP = (F_1, …, F_k) satisfies the following:
  - $F_k \in \mathcal{F}, \forall k \in \{1 \dots \mathbb{N}\}$ and
  - (F_k, F_{k+1}) is an information flow, $\forall k \in \{1 \dots \mathbb{N}\text{-}1\}$.

  The constituent functions realize the set of operations Op_j defined at the mission layer via information flows for mission accomplishment.

The complete mapping is described in Table 5.2.

Table 5.2: Mapping: Functional-layer System DSML ↦ Logical Specification.

DSML Element	Logical Specification
System	S := ({F_k}), $k \in \{1 \dots \mathbb{N}\}$
Function	F_k := (*type*, *preCondition*, *triggerType*, *caller*, *functionalInterfaces*, Information), *type* ∈ {*Physical*, *Logical*}, *triggerType* ∈ {*Manual*, *ControlSignal*, *TemporalConstraint*}, TemporalConstraint ∈ {Duration, TriggerTime}, *caller* ∈ {Operator, F_i, TemporalConstraint}, $k, l \in \{1 \dots \mathbb{N}\}$
Information	Information :∈ {FunctionalInput (or I_k), FunctionalOutput (or O_k)}, $k \in \{1 \dots \mathbb{N}\}$
FunctionalInterface	FI | FI*R*I_k or FI*R*O_k, $k \in \{1 \dots \mathbb{N}\}$ (*R*: "*is related with*", where type of *R* is determined based on the link type in the UML profile)
InformationFlow	InformationFlow := ({(O_i, I_j) $\stackrel{\Delta}{=}$ (F_i, F_j)} | FunctionalOutput[F_i] = O_i, FunctionalInput[F_j] = I_j), $i, j \in \{1 \dots \mathbb{N}\}$
FunctionalPath	FP := (F_1,… , F_k) | $\forall$(F_k, F_{k+1}), $\exists$ (O_k, I_{k+1}) ∈ {(O_i, I_j)}, $i, j, k \in \{1 \dots \mathbb{N}\}$

5.4.1.3 Component Layer

Herein, S denotes a system consisting of a set of components C_m, $m \in \{1 \dots \mathbb{N}\}$. Each component C_m is characterized by a set $\mathcal{P}$ of ports P_n, $n \in \{1 \dots \mathbb{N}\}$ and the attribute variables from the component-layer system DSML illustrated in Figure 4.4. A connector Conn between two ports P_i and P_j is denoted by an ordered pair (P_i, P_j). It establishes a connection between the output/input port P_i of component C_i and the input/output port P_j of component C_j. The domain of possible interaction types denoted by IntType is the set {Signal, Data, Packet}. Herein, Data corresponds to the information being transmitted, Signal refers to the carrier of data over a communication channel, and Packet is a segment of data. Likewise, the domain of the communication styles defining the behavior of interaction between the components, denoted by CommunicationStyle, is the set {MsgPassing, Broadcast, Multicast}. Herein, MsgPassing involves the transmission of messages from one component to other over a communication

channel, Broadcast involves simultaneous message transmission to all the recipient components, and Multicast involves simultaneous message transmission to a group of components. It is recalled that other communication styles and more technology-dependent models can be included. The complete mapping is described in Table 5.3.

Table 5.3: Mapping: Component-layer System DSML ↦ Logical Specification.

DSML Element	Logical Specification
System	$S := (\{C_m\}), m \in \{1 \dots \mathbb{N}\}$
Component	$C_m := (\{P_n\}, type, failureRate, givenTime), type \in \{Atomic, Composite\}, m, n \in \{1 \dots \mathbb{N}\}$
Port	$P_n := (kind, \text{IntType}), kind \in \{Input, Output\}, n \in \{1 \dots \mathbb{N}\}$
Connector	$\text{Conn} := \begin{matrix}(\{(P_i, P_j)\}, connType, \text{CommunicationStyle}), P_i.kind = \text{Output/Input}, P_j.kind = \text{Input/Output},\\ connType \in \{Bus, Pipe, Channel\}, i, j \in \{1 \dots \mathbb{N}\}\end{matrix}$
InteractionType	IntType :∈ {Signal, Data, Packet}
CommunicationStyle	CommunicationStyle :∈ {MsgPassing, Broadcast, Multicast}

5.4.1.4 Properties

Logical specification of the properties DSML presented in Section 4.5 considers the logical specification of a set of elements from the three-layered system model to which the properties are associated. Herein, PropertyCategory is a set comprising safety and security objective categories, e.g., Availability and Freshness. It is recalled from Section 1.1.2 that these categories and their respective objectives are considered as representative ones. The relationship between the properties is characterized by an attribute variable *type*, the value of which spans across the set {*dependsUpon*, *contributesTo*, *composedOf*, *conflictsWith*, *definedBy*}. The mapping herein provides the basis to incorporate the safety and security properties specifications and their interplay in the modeled target system to be furtherly analyzed. The complete mapping is described in Table 5.4.

5.4.2 Basic Properties

A subset of safety and security properties belonging to certain categories in this work, e.g., *Integrity* and *Freshness*, are characterized by ordering and timing-based notions defining the system behavior. For example, the freshness property incorporates the notion of timeliness to determine the validity of the messages based on when they were exchanged between the components. This, in turn, is of concern to prevent any outdated messages from being involved in control decisions for the safety-critical functionality of the system [142]. Therefore, it requires expressing and reasoning these notions to prove that the system satisfies the required safety and security properties.

Table 5.4: Mapping: Properties DSML ↦ Logic Specification.

DSML Element	Logical Specification
System	S := ({M_i}, Environment), $i \in \{1 \dots \mathbb{N}\}$
Mission	M_i, $i \in \{1 \dots \mathbb{N}\}$
Operation	Op_j, $j \in \{1 \dots \mathbb{N}\}$
Function	F_k, $k \in \{1 \dots \mathbb{N}\}$
FunctionalPath	FP := (F_1, …, F_k), $k \in \{1 \dots \mathbb{N}\}$
Information	Information :∈ {I_k, O_k}, $k \in \{1 \dots \mathbb{N}\}$
Component	C_m, $m \in \{1 \dots \mathbb{N}\}$
Message	*msg*
PropertyCategory	PropertyCategory :∈ {SafetyObjectiveCategory, SecurityObjectiveCategory}
Relationship	Relationship := (*type*), *type* ∈ {*dependsUpon*, *contributesTo*, *composedOf*, *conflictsWith*, *definedBy*}
SafetyObjectiveCategory	SafetyObjectiveCategory :∈ {Availability, Integrity}
SecurityObjectiveCategory	SecurityObjectiveCategory :∈ {ControlledAccessibility, Freshness}

Hence, we present a set of properties to describe the notions mentioned above via generic specifications. We call these properties the basic ones since they play an elementary role in defining, formally specifying, and reasoning the safety and security properties used by the end user in system modeling. The specific safety and security objectives associated with the target system model, as depicted in Figure 4.11, are instantiated at the different layers of the three-layered model relying upon the basic properties. Some of these properties are based upon well-accepted notions from the state-of-the-art; however, we extend them by adapting to our three-layered design approach and tooled framework. Accordingly, these properties can be instantiated and formally specified at different modeling layers, viz. *Mission*, *Functional*, and *Component*, using logic-based languages (e.g., FOL and Modal Logic). However, not each property can be instantiated at all the layers of the three-layered system model. In essence, it depends upon the availability of the essential design details a layer offers to analyze specific safety and security properties. Moreover, the instantiation of the basic property at a layer is also subjected to the semantics of the property itself. For example, a time-related property like *Freshness* cannot be verified on an element *Mission* that is static and does not change across time.

The basic properties used in this work are described as follows:

- **Precedence:** In literature, precedence properties are commonly used in the specification of concurrent systems for defining the ordering relationship between a pair of states or events, where the occurrence of one is a necessary pre-condition for the other [143]. Extending this by adapting it to the notions and syntax of the DSMLs, the precedence property in our context imposes a partial ordering among the DSML elements, where two or more elements are realized sequentially. In simpler terms, it is associated with the orderly or relative consideration of the elements within a defined scope. Herein, the

scope captures the level of granularity with which the target system is modeled at the three layers. Accordingly, the term realization can be related to the execution of functions at the functional layer, enabling components at the component layer, etc.

- **Equivalence:** Equivalence checking is commonly used in literature for verifying the symbolic, logical, or functional equivalence in real-time systems [144]. In communication models, it is often related to the indistinguishability properties [145]. In our work, equivalence between two elements is related to the equivalence of information flows between origin and destination elements, like in sending and receiving events. Both structural and behavioral aspects can be considered concerning these elements for equivalence verification.

- **Timeliness:** In literature, the notion of timeliness is often observed as a performance measure for defining an optimal time for task completion in real-time systems or communication scheduling algorithms [146]. Inspired by this, the timeliness property in our work imposes time constraints for the realization of elements. For instance, it may be concerned with the specification of time bounds to constrain the execution of system functions.

5.4.3 FOL and Modal Logic-based Properties Specification

We use standard FOL operators, including $\wedge$ (*conjunction*), $\vee$ (*disjunction*), $\neg$ (*negation*), $\rightarrow$ (*implication*), and $\leftrightarrow$ (*equivalence*) for logical specification of safety and security properties. The FOL-based properties formalism is extended using a range of modalities, including $\circ$ (*next*), $\Diamond$ (*eventually*), $\Diamond_{\leq\Theta}$ (*bounded eventually*, where Θ denotes the threshold or bounded gap between two events, occurrences, actions, etc.), and $\Box$ (*always*), for capturing the notion of future [147]. Likewise, $\bullet$, $\blacklozenge$, $\blacklozenge_{\leq\Theta}$, and $\blacksquare$, respectively, are used as their past counterparts. These modalities are briefly described in Table 5.5. For example, predicates of the form $P \Rightarrow Q$ in FOL can be refined to $\Box(P \rightarrow \Diamond Q)$ using Modal Logic. Similarly, predicates of the form $P \Leftrightarrow Q$ can be refined to $\Box(P \leftrightarrow Q)$.

Table 5.5: Meaning of the Modalities used in this Work.

Notation	Meaning	Past-Counterpart	Meaning
$\circ$P	P will be true in the next state	$\bullet$P	P was true in the previous state
$\Diamond$P	P will be true sometime in the future	$\blacklozenge$P	P was true sometime in the past
$\Diamond_{\leq\Theta}$P	P will be true sometime in the future, no later than Θ	$\blacklozenge_{\leq\Theta}$P	P was true sometime in the past, at most Θ ago
$\Box$P	P will be true always in the future	$\blacksquare$P	P was always true in the past

P: Predicate, Θ: Threshold or bounded gap

Here, $\Rightarrow$ means *strongly implies* and $\Leftrightarrow$ means *strongly equivalent*. These refinements may vary based on the informal definition of the specific objectives [**NB:** It is recalled that objectives herein represent the desired properties or features, capturing the positive vision of safety and security]. Depending on the priority of the system behavior defined using the notions, e.g., operation, function, etc., concerning safety and security, $\circ$ (highest priority), $\Diamond_{\leq\Theta}$ (medium priority), or $\Diamond$ (lowest priority) can be used in the formal interpretations.

The formal logic defining the safety and security objective signatures across the layered models is introduced in the following sub-sections.

5.4.3.1 Mission Layer

At the mission layer, we consider one safety objective category (i.e., *Availability*) and one security objective category (i.e., *Controlled Accessibility*). Following the definitions introduced in Section 4.5.1.1, the FOL-based formalism of the specific objectives belonging to these categories is presented as follows:

- **Operational Availability:** Given a mission M_i, $i \in \mathbb{N}$ and an operation Op_j, $j \in \mathbb{N}$, the operational availability objective denoted by ***OperationalAvailability***(M_i, Op_j) holds iff (see definition **Prop1**):

$$(operationalSituation[\mathrm{M}_i] \wedge hazardousEvent[\mathrm{M}_i]) \Rightarrow accomplishedThrough[\mathrm{M}_i, \mathrm{Op}_j]$$

  To satisfy ***OperationalAvailability***(M_i, Op_j), the following specification in Modal Logic must be respected:

$$\Box((operationalSituation[\mathrm{M}_i] \wedge hazardousEvent[\mathrm{M}_i]) \rightarrow \Diamond(accomplishedThrough[\mathrm{M}_i, \mathrm{Op}_j]))$$

  i.e., it will always be the case that on the detection of a hazardous event, a system in a certain operational situation will eventually perform an operation Op_j for the accomplishment of the safety-critical mission M_i. For example, on the detection of an obstacle, a CDV moving towards a junction will eventually undergo braking to avoid a collision.

- **System Accessibility:** Given a system S and an operation Op_j, $j \in \mathbb{N}$, the system accessibility objective denoted by ***SystemAccessibility***(S, Op_j, People) holds iff (see definition **Prop2**):

$$privilege[\text{People, S, Op}_j] \Rightarrow access[\text{People, Op}_j]$$

To satisfy ***SystemAccessibility***(S, Op_j, People), the following specification in Modal Logic must be respected:

$$\Box(privilege[\text{People, S, Op}_j] \rightarrow \Diamond(access[\text{People, Op}_j]))$$

i.e., it will always be the case that on acquiring the required privileges, environmental entities, like a person, can eventually access the system S (i.e., CDV) to perform an operation Op_j, e.g., controlling the navigation.

Table 5.6 summarizes the aforementioned logical specifications of safety and security objectives at the mission layer.

Table 5.6: Logical Specification of Safety and Security Objectives at the Mission Layer.

Objective	Notation	Logical Specification
Operational availability	***OperationalAvailability***(M_i, Op_j)	$operationalSituation[\text{M}_i] \wedge hazardousEvent[\text{M}_i] \Rightarrow accomplishedThrough[\text{M}_i, \text{Op}_j]$
System accessibility	***SystemAccessibility***(S, Op_j, People)	$privilege[\text{People, S, Op}_j] \Rightarrow access[\text{People, Op}_j]$

5.4.3.2 Functional Layer

To specify safety and security objectives at the functional layer, we present semantics associated with their underlying temporal notions as follows:

- **Global time scale:** We elicit the notion of a global time scale from [148] as a timeline representing past and future concerning the events occurring in the system.
- **Instant:** An instant represents a single point in the timeline.
- **Occurrence:** To capture the evolution in the system state, we define the notion of occurrence that represents something has happened at an instant, leading to state transition regarding the system. Formally, an occurrence with respect to the execution of a function F_k can be defined as a quadruple $\mathcal{O}cc(\text{F}_{k-1}, \text{F}_k, t_i, \text{X}_k)$, $i, k \in \{1 \ldots \mathbb{N}\}$, $t_i \in \mathbb{R}^+$, where
  - F_{k-1} corresponds to the caller function,
  - F_k corresponds to the callee function,
  - t_i denotes the time of occurrence, and

  - X_k denotes either an input I_k or output O_k associated with the function F_k.

- **Gap between occurrences:** Given two occurrences $\mathcal{Occ}(F_{i-1}, F_i, t_i, X_i)$ and $\mathcal{Occ}(F_{j-1}, F_j, t_j, X_j)$, the gap between occurrences is given by

$$d = |t_j - t_i|$$

- **Duration of execution of a function:** It represents the duration d_k it takes to execute a function F_k by the system. It is defined with two occurrences—*begin*, i.e., $\mathcal{Occ}(F_{k-1}, F_k, t_{2k-1}, I_k)$ and *end*, i.e., $\mathcal{Occ}(F_{k-1}, F_k, t_{2k}, O_k)$—corresponding to the triggering and termination of function execution, respectively. Formally, duration is given by

$$d_k = |t_{2k} - t_{2k-1}|$$

[**NB:** The sub-indices in this and other definitions are meant to simplify notation, assuming a functional path with functions sequentially indexed over 1, …, $\mathbb{N}$.]

Moreover, the specification of objectives at the functional layer adheres to the following meta-properties:

- **MP1:** Every function F_i in the functional path FP(F_1, …, F_k) is defined as F_i: $I_i \mapsto O_i$, $i \in \{1 \dots k\}$.

- **MP2:** For every function F_i in the functional path FP(F_1, …, F_k), the notation for sub-indices is (F_{i-1}, F_i, t_{2i-1}, I_i) and (F_{i-1}, F_i, t_{2i}, O_i), $\forall i \in \{2 \dots k\}$.

- **MP3:** Consider that the function F_1 for any functional path FP is self-invoked. Hence, the notation for sub-indices is (F_1, F_1, t_1, I_1) and (F_1, F_1, t_2, O_1).

- **MP4:** For the initial function F_1 in every FP, the value of threshold Θ_1 corresponding to the input and output occurrences of information flow cannot be considered, as there is no function previous to F_1 for interaction and it is self-invoked.

- **MP5:** Information flows correspond to payloads exchanged during the function calls.

Basic Properties. Prior to formally specifying the safety and security properties at the functional layer, we instantiate the basic properties presented in Section 5.4.2 as follows:

- **Functional Precedence:** The functional precedence property imposes a partial ordering on the execution of the system functions, where two or more functions are executed sequentially. For example, the function to infer the position of the obstacle is preceded by

the execution of the obstacle detection function that determines first whether the obstacle is present.

Formally, we instantiate the notion of precedence presented in Section 5.4.2 for two functions F_i and F_j. Thus, the functional precedence denoted by ***FunctionalPrecedence***(F_i, F_j) holds iff:

$$F_j \Rightarrow F_i$$

To satisfy ***FunctionalPrecedence***(F_i, F_j), the following specification in Modal Logic must be respected:

$$\blacksquare(F_j \rightarrow \blacklozenge(F_i))$$

i.e., if F_j is ever invoked, it was preceded by the execution of F_i. To keep time consistency in the occurrences corresponding to the functions, the following strict partial ordering constraint should be satisfied:

$$timeOfTrigger(F_i) + d_i < timeOfTrigger(F_j)$$

In other words, F_i precedes F_j provided F_i is completed within a duration d_i before the trigger time of F_j, namely F_i occurs strictly ($<$) before the occurrence of F_j. Herein, duration d_i is defined by the occurrences corresponding to the triggering and termination of F_i's execution. Additional constraints ($<_\Theta$) can be added to specify the gap between occurrences corresponding to F_i's termination and F_j's beginning.

- **Information Equivalence:** The information equivalence property ensures the preservation of the information exchanged between two or more functions, particularly those constituting a functional path.

  Formally, we instantiate the notion of equivalence presented in Section 5.4.2 for a given information flow (O_i, I_j) between two functions F_i and F_j. Thus, the information equivalence denoted by ***InformationEquivalence***(O_i, I_j) holds iff:

$$I_j \Leftrightarrow O_i$$

  To satisfy ***InformationEquivalence***(O_i, I_j), the following specification in Modal Logic must be respected:

$$\square(I_j \leftrightarrow O_i)$$

i.e., it will always be the case that the functional input I_j of F_j is equivalent to the functional output O_i of F_i. For instance, $\leftrightarrow$, in this case, may denote the equivalence relation induced by partitioning the possible information space into classes containing equivalent information [149].

- **Execution Timeliness:** Execution of a functional path involves the execution of the functions constituting the path. Accordingly, the execution timeliness property ensures that whenever there is an occurrence corresponding to the acceptance of input by the first function in the functional path, the occurrence corresponding to the last function generating the final output should appear not beyond the pre-defined threshold time gap.

  Formally, we instantiate the notion of timeliness presented in Section 5.4.2 for a functional path FP and a given threshold Θ, where $\Theta \in \mathbb{R}^+$. Thus, the execution timeliness denoted by ***ExecutionTimeliness***(FP, Θ) holds iff:

$$\mathcal{Occ}(F_1, F_1, t_1, I_1) \Rightarrow_\Theta \mathcal{Occ}(F_{k-1}, F_k, t_{2k}, O_k)$$

  Herein, the timeliness property is instantiated explicitly regarding the occurrences corresponding to the ordered execution of the functions that constitute the functional path.

  To satisfy ***ExecutionTimeliness***(FP, Θ), the following specification in Modal Logic must be respected:

$$\Box(\mathcal{Occ}(F_1, F_1, t_1, I_1) \rightarrow_\Theta \Diamond(\mathcal{Occ}(F_{k-1}, F_k, t_{2k}, O_k)))$$

  that can be verified iff:

$$|t_{2k} - t_1| \leq \Theta$$

  i.e., it will always be the case that whenever there is an occurrence corresponding to the acceptance of input by the first function F_1 in the functional path, the occurrence corresponding to the last function F_k generating the final output should appear not beyond the pre-defined threshold time gap Θ.

- **Overall Timeliness:** The overall timeliness property ensures that whenever there is an occurrence corresponding to the generation of the output by the last function in the functional path, the occurrence corresponding to the acceptance of input by the first function in the functional path should not have happened beyond the pre-defined threshold time gap in the past.

Formally, we instantiate the notion of timeliness presented in Section 5.4.2 for a functional path FP and a given threshold Θ, where $\Theta \in \mathbb{R}^+$. Thus, the overall timeliness denoted by ***OverallTimeliness***(FP, Θ) holds iff:

$$\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k}, \mathrm{O}_k) \Rightarrow_{\Theta} \mathcal{Occ}(\mathrm{F}_1, \mathrm{F}_1, t_1, \mathrm{I}_1)$$

Herein, the timeliness property is instantiated explicitly regarding the occurrences corresponding to the execution of the functions that constitute the functional path.

To satisfy ***OverallTimeliness***(FP, Θ), the following specification in Modal Logic must be respected:

$$\blacksquare(\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k}, \mathrm{O}_k) \rightarrow_{\Theta} \blacklozenge(\mathcal{Occ}(\mathrm{F}_1, \mathrm{F}_1, t_1, \mathrm{I}_1)))$$

that can be verified iff:

$$|t_{2k} - t_1| \leq \Theta$$

[**NB:** ***OverallTimeliness***(FP, Θ) can be considered as the past counterpart of ***ExecutionTimeliness***(FP, Θ).]

- **Information Flow Timeliness:** The information flow timeliness property ensures that whenever there is an occurrence corresponding to the acceptance of the input by a function in the functional path, the occurrence corresponding to the generation of output by the previous function in the functional path should not have happened beyond the pre-defined threshold in the past.

  Formally, we instantiate the notion of timeliness presented in Section 5.4.2 for a functional path FP, information flow (O_{k-1}, I_k), and a given threshold Θ_k, where $\Theta_k \in \mathbb{R}^+$. Thus, the information flow timeliness denoted by ***InformationFlowTimeliness***(FP, $\{(\mathrm{O}_{k-1}, \mathrm{I}_k)\}$, Θ_k) holds iff:

$$\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k-1}, \mathrm{I}_k) \Rightarrow_{\Theta_k} \mathcal{Occ}(\mathrm{F}_{k-2}, \mathrm{F}_{k-1}, t_{2(k-1)}, \mathrm{O}_{k-1}), \forall k \geq 2$$

  Herein, the timeliness property is instantiated explicitly regarding the occurrences corresponding to the execution of the functions that constitute the functional path.

  To satisfy ***InformationFlowTimeliness***(FP, $\{(\mathrm{O}_{k-1}, \mathrm{I}_k)\}$, Θ_k), the following specification in Modal Logic must be respected:

$$\blacksquare(\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k-1}, \mathrm{I}_k) \rightarrow_{\Theta_k} \blacklozenge(\mathcal{Occ}(\mathrm{F}_{k-2}, \mathrm{F}_{k-1}, t_{2(k-1)}, \mathrm{O}_{k-1}))), \forall k \geq 2$$

that can be verified iff:

$$|t_{2k-1} - t_{2(k-1)}| \leq \Theta_k$$

Table 5.7 summarizes the aforementioned logical specifications of basic properties at the functional layer.

Table 5.7: Logical Specification of Basic Properties at the Functional Layer.

Property	Notation	Logical Specification
Functional precedence	***FunctionalPrecedence***(F_i, F_j)	$F_j \Rightarrow F_i$
Information equivalence	***InformationEquivalence***(O_i, I_j)	$I_j \Leftrightarrow O_i$
Execution timeliness	***ExecutionTimeliness***(FP, Θ)	$\mathcal{Occ}(F_1, F_1, t_1, I_1) \Rightarrow_{\Theta} \mathcal{Occ}(F_{k-1}, F_k, t_{2k}, O_k)$
Overall timeliness	***OverallTimeliness***(FP, Θ)	$\mathcal{Occ}(F_{k-1}, F_k, t_{2k}, O_k) \Rightarrow_{\Theta} \mathcal{Occ}(F_1, F_1, t_1, I_1)$
Information flow timeliness	***InformationFlowTimeliness*** (FP, $\{(O_{k-1}, I_k)\}$, Θ_k)	$\mathcal{Occ}(F_{k-1}, F_k, t_{2k-1}, I_k) \Rightarrow_{\Theta_k} \mathcal{Occ}(F_{k-2}, F_{k-1}, t_{2(k-1)}, O_{k-1})$, $\forall k \geq 2$

Safety and Security Properties. At the functional layer, we consider two safety objective categories (i.e., *Integrity* and *Availability*) and one security objective category (i.e., *Freshness*). Following the definitions introduced in Section 4.5.1.2, the FOL-based formalism of the specific objectives belonging to these categories is presented as follows:

- **Functional Integrity:** A functional path FP := (F_1, …, F_k), $k \in \mathbb{N}$ satisfies the functional integrity objective denoted by ***FunctionalIntegrity***(FP) iff (see definition **Prop3**):
  - There exists a partial ordering on the execution of functions constituting the functional path, i.e., ***FunctionalPrecedence***(F_k, F_{k+1}) holds, $\forall k \in \{1 \ldots \mathbb{N}\text{-}1\}$ and
  - The information flow is preserved across the functions constituting the functional path, i.e., ***InformationEquivalence***(O_k, I_{k+1}) holds, $\forall k \in \{1 \ldots \mathbb{N}\text{-}1\}$, for at least one information flow (O_k, I_{k+1}) pertaining to (F_k, F_{k+1}).
- **Functional Path Availability:** A functional path FP := (F_1, …, F_k), $k \in \mathbb{N}$ satisfies the functional path availability objective denoted by ***FunctionalPathAvailability***(FP) iff (see definition **Prop4**):
  - There is a preservation of the information flow between orderly execution of functions constituting the functional path, i.e., ***FunctionalIntegrity***(FP) holds and
  - The timeliness of the execution of the functional path is ensured, i.e., ***ExecutionTimeliness***(FP, Θ) holds.

[**NB:** It is recalled that MP1, MP2, MP3 mentioned in the beginning of sub-section 5.4.3.2 hold in this case.]

- **Functional Path Freshness:** A functional path FP := (F_1, …, F_k), $k \in \mathbb{N}$ satisfies the functional path freshness objective denoted by ***Freshness***(FP) iff (see definition **Prop5**):
  - There is a preservation of the information flow between orderly execution of functions constituting the functional path, i.e., ***FunctionalIntegrity***(FP) holds and
  - The timeliness between the final output and initial input occurrences of the functional path is ensured, i.e., ***OverallTimeliness***(FP, Θ) holds.

  [**NB:** It is recalled that MP1, MP2, MP3 mentioned in the beginning of sub-section 5.4.3.2 hold in this case.]

- **Information Flow Freshness:** A functional path FP := (F_1, …, F_k), $k \in \mathbb{N}$ satisfies the information flow freshness objective denoted by ***Freshness***(FP, {(O_{k-1}, I_k)}) iff (see definition **Prop6**):
  - There is a preservation of the information flow between orderly execution of functions constituting the functional path, i.e., ***FunctionalIntegrity***(FP) holds and
  - The timeliness of the information flows between consecutive functions across the functional path are ensured, i.e., ***InformationFlowTimeliness***(FP, {(O_{k-1}, I_k)}, Θ_k) holds.

  [**NB:** It is recalled that MP1, MP4, MP5 mentioned in the beginning of sub-section 5.4.3.2 hold in this case.]

Table 5.8 summarizes the aforementioned logical specifications of safety and security objectives at the functional layer.

Table 5.8: Logical Specification of Safety and Security Objectives at the Functional Layer.

Objective	Notation	Logical Specification
Functional integrity	***FunctionalIntegrity***(FP)	***FunctionalPrecedence***(F_k, F_{k+1}) ∧ ***InformationEquivalence***(O_k, I_{k+1})
Functional path availability	***FunctionalPathAvailability***(FP)	***FunctionalIntegrity***(FP) ∧ ***ExecutionTimeliness***(FP, Θ)
Functional path freshness	***Freshness***(FP)	***FunctionalIntegrity***(FP) ∧ ***OverallTimeliness***(FP, Θ)
Information flow freshness	***Freshness***(FP, {(O_{k-1}, I_k)})	***FunctionalIntegrity***(FP) ∧ ***InformationFlowTimeliness***(FP, {(O_{k-1}, I_k)}, Θ_k)

5.4.3.3 Component Layer

Basic Properties. Prior to formally specifying the safety and security properties at the component layer, we instantiate the basic properties presented in Section 5.4.2 as follows:

- **Component and Connector Availability:** In literature, availability analysis of system components often includes reliability in conjunction with their replacement/recovery from faults [150]. For communication protocols, the availability of the components is associated with their ability to engage in communication [151]. Inspired by these notions, the component (or connector) availability property in our context represents the probability that a component (or a connector) is operational at a given time to take part in communication.

  We use the typical definition of reliability [152] to measure availability. Formally, let R(t) denotes the probability that a component C_i with failure rate λ_i (or a connector Conn with failure rate λ_{conn}) is operational for a certain time interval (0, t], $t \in \mathbb{R}^+$. Then, the component (or connector) availability property denoted by ***ComponentAvailability***(C_i, λ_i, t) (or ***ConnectorAvailability***(Conn, λ_{conn}, t)) holds iff for a time to failure T:

  $$R(t) = 1 - CDF(t) = P(T > t)$$

  To satisfy ***ComponentAvailability***(C_i, λ_i, t) (or ***ConnectorAvailability***(Conn, λ_{conn}, t)), the following specification in Modal Logic must be respected:

  $$R(t) = P\{\Box^{(0,t]}\mathcal{OC}\}$$

  where, $\mathcal{OC}$ denotes operational constraints that the component (or connector) must fulfill within the interval (0, t].

  [**NB:** The following notions are adopted from the literature, where probabilistic analysis [153] is used as one of the tools for determining the availability of the system components:

  - To model time to failure, an exponential random variable X(t), $t \in \mathbb{R}^+$ is used [150].
  - For an interval $[t_1, t_2]$, $t_1, t_2 \in \mathbb{R}^+$, the probability of occurrence of an event E, i.e., P(E)$[t_1, t_2]$ is directly proportional ($\propto$) to $|t_2 - t_1|$. If $t_1 = 0$ and $t_2 = t$, then P(E)[0, t] $\propto |t - 0| = t$.
  - Let $\lambda \in \mathbb{R}^+$ denotes the failure rate of the component (or connector). Then,

  $$ProbabilityDensityFunction : PDF(t) = \begin{cases} \lambda e^{-\lambda t} & \text{*for } t > 0,\ \lambda > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

$$CumulativeDensityFunction : CDF(t) = \int_0^{t'} \lambda e^{-\lambda t'} dt' \tag{5.2}$$

*Models exponential failure distribution]

- **Message Delivery:** In protocol design for safety-critical systems based on a message passing-based communication model, message delivery typically ensures the successful transmission of safety-critical messages (e.g., control signals, packets, etc.). It may have the following two variants:

  – **Eventual Message Delivery:** This property ensures that following the sending of some message by a component, the intended receiving component will eventually receive the message.

  Formally, given two components C_i and C_j, where $i, j \in \mathbb{N}$ and a message msg, the eventual message delivery property denoted by ***EventualMessageDelivery***(C_i, C_j, msg) holds iff:

  $$send(C_i, msg) \Rightarrow receive(C_j, msg)$$

  To satisfy ***EventualMessageDelivery***(C_i, C_j, msg), the following specification in Modal Logic must be respected:

  $$\Box(send(C_i, msg) \rightarrow \Diamond(receive(C_j, msg)))$$

  i.e., it will always be the case that if a component C_i sends a message msg, it will be eventually received by the intended component C_j.

  – **Bounded Message Delivery:** This property imposes time constraints for the timely delivery of safety-critical messages to the intended receiver.

  Formally, as an instantiation of the notion of timeliness presented in Section 5.4.2 for two components C_i and C_j, where $i, j \in \mathbb{N}$, a message msg, and a pre-defined threshold $\Theta \in \mathbb{R}^+$, the bounded message delivery property denoted by ***BoundedMessageDelivery***(C_i, C_j, msg, Θ) holds iff:

  $$send(C_i, msg) \Rightarrow_\Theta receive(C_j, msg)$$

  Herein, the timeliness property is instantiated explicitly regarding the sending and receiving actions performed by the system components.

  To satisfy ***BoundedMessageDelivery***(C_i, C_j, msg, Θ), the following specification in Modal Logic must be respected:

  $$\Box(send(C_i, msg) \rightarrow_\Theta \Diamond(receive(C_j, msg)))$$

i.e., it will always be the case that if a component C_i sends a message msg, it will be received by the intended receiving component C_j within Θ time interval.

- **Logical Conformity:** Logical conformity validates the component architecture layer against the functional layer. In other words, it validates the configuration of the system components in association with the specifications stemming from the functional model.

  Formally, considering a model instantiation depicted in Figure 5.2, logical conformity of components with function F_3 is elaborated as follows:

$$F_3\colon I_X \mapsto O_3$$
$$\text{where, } I_X \stackrel{\Delta}{=} I_{3,1} \wedge I_{3,2};\ I_X \stackrel{\Delta}{=} I_{3,1} \vee I_{3,2}$$
$$I_3 = O_1 \wedge O_2;\ I_3 = O_1 \vee O_2$$
$$O_1 \wedge O_2 \mapsto O_3;\ O_1 \vee O_2 \mapsto O_3$$
$$P(C_1 = 1, t) \wedge P(C_2 = 1, t) = P(C_3 = 1, t);\ P(C_1 = 1, t) \vee P(C_2 = 1, t) = P(C_3 = 1, t)$$

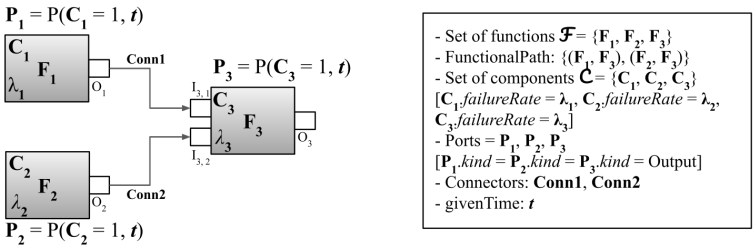


Figure 5.2: Model Instantiation to depict Logical Conformity between Functional and Component Architecture.

- **Component Precedence:** The component precedence property imposes an ordering on the enabling of the components via interfaces, i.e., ports.

  Formally, instantiating the notion of precedence presented in Section 5.4.2, the precedence between two components C_i and C_j denoted by ***ComponentPrecedence***(C_i, C_j) holds iff:

$$C_j \Rightarrow C_i$$

  To satisfy ***ComponentPrecedence***(C_i, C_j), the following specification in Modal Logic must be respected:

$$\blacksquare(C_j \rightarrow \blacklozenge(C_i))$$

i.e., if C_j is ever enabled, it was preceded by the enabling of C_i. To preserve the order in which the components are enabled, the following constraint regarding the kind of ports forming the connecting link between these components must be satisfied:

$$\forall C_i, C_j, \exists P_i, P_j | (P_i.kind = \text{Output}) \wedge (P_j.kind = \text{Input})$$

- **Recentness:** The recentness property ensures that whenever a component receives some message, the same was sent by another component at most some pre-defined time gap ago.

  Formally, we instantiate the notion of timeliness presented in Section 5.4.2 for two components C_i and C_j, where $i, j \in \mathbb{N}$, a message msg, and a pre-defined threshold Θ. Thus, the recentness property denoted by ***Recentness***(C_i, C_j, msg, Θ) holds iff:

  $$receive(C_j, msg) \Rightarrow_{\Theta} send(C_i, msg)$$

  Herein, the timeliness property is instantiated explicitly regarding the sending and receiving actions performed by the system components.

  To satisfy ***Recentness***(C_i, C_j, msg, Θ), the following specification in Modal Logic must be respected:

  $$\blacksquare(receive(C_j, msg) \rightarrow_{\Theta} \blacklozenge(send(C_i, msg)))$$

  i.e., if C_j ever received a message msg, it must not have been sent before Θ time gap ago by C_i in the past.

- **Non-Duplication:** The non-duplication property ensures that a system component has not received the same message previously. In simpler terms, it ensures the uniqueness of the messages delivered to a component.

  Formally, given a component C_i and a message msg, the non-duplication property denoted by ***NonDuplication***(C_i, msg) holds iff:

  $$receive(C_i, msg) \Rightarrow \neg(receive(C_i, msg))$$

  To satisfy ***NonDuplication***(C_i, msg), the following specification in Modal Logic must be respected:

  $$\blacksquare(receive(C_i, msg) \rightarrow \neg(\blacklozenge(receive(C_i, msg))))$$

  i.e., if C_i ever received a message msg, it should not have received the same message msg in the past.

Table 5.9 summarizes the aforementioned logical specifications of basic properties at the component layer.

Table 5.9: Logical Specification of Basic Properties at the Component Layer.

Property	Notation	Logical Specification
Component or connector availability	***ComponentAvailability***(C_i, λ_i, t) or ***ConnectorAvailability***(Conn, λ_{conn}, t)	R(t) = 1 - CDF(t) = P(T > t)
Eventual message delivery	***EventualMessageDelivery***(C_i, C_j, msg)	$send(C_i, msg) \Rightarrow receive(C_j, msg)$
Bounded message delivery	***BoundedMessageDelivery***(C_i, C_j, msg, Θ)	$send(C_i, msg) \Rightarrow_{\Theta} receive(C_j, msg)$
Component precedence	***ComponentPrecedence***(C_i, C_j)	$C_j \Rightarrow C_i$
Recentness	***Recentness***(C_i, C_j, msg, Θ)	$receive(C_j, msg) \Rightarrow_{\Theta} send(C_i, msg)$
Non-duplication	***NonDuplication***(C_i, msg)	$receive(C_i, msg) \Rightarrow \neg(receive(C_i, msg))$

Safety and Security Properties. At the component layer, we consider two safety objective categories (i.e., *Availability* and *Integrity*) and one security objective category (i.e., *Freshness*). Following the definitions introduced in Section 4.5.1.3, the FOL-based formalism of the specific objectives belonging to these categories is presented as follows:

- **Component-Port-Connector (CPC) Availability:** Given two components C_i having a failure rate λ_i and C_j having a failure rate λ_j, where $i, j \in \mathbb{N}$, a connector $Conn$ having a failure rate λ_{Conn} connecting C_i and C_j, a message msg, and a pre-defined threshold $\Theta \in \mathbb{R}^+$, the CPC availability objective denoted by ***CPCAvailability*** is satisfied iff (see definition **Prop7**):
  - Availability of components is ensured, i.e., ***ComponentAvailability***(C_i, λ_i, t) holds,
  - Availability of connector is ensured, i.e., ***ConnectorAvailability***(Conn, λ_{Conn}, t) holds,
  - Delivery of critical messages between the components is ensured, i.e., ***EventualMessageDelivery***(C_i, C_j, msg) (or ***BoundedMessageDelivery***(C_i, C_j, msg, Θ)) holds, and
  - There exists logical conformity of the component architecture with functional architecture verified at the respective component and functional layers.
- **Message Integrity:** Given two components C_i and C_j, where $i, j \in \mathbb{N}$, and a message msg, the message integrity objective denoted by ***MessageIntegrity***(msg) is satisfied iff (see definition **Prop8**):
  - The message once sent by a component, is eventually received by the intended component in the same form, i.e., ***EventualMessageDelivery***(C_i, C_j, msg) holds.
- **Message Freshness:** Given two components C_i and C_j, where $i, j \in \mathbb{N}$, a message msg,

and a pre-defined threshold $\Theta \in \mathbb{R}^+$, the freshness objective regarding msg denoted by ***MessageFreshness***(msg) holds iff (see definition **Prop9**):

- Precedence between the interacting components is ensured, i.e., ***ComponentPrecedence***(C_i, C_j) holds,
- The message remains unaltered during its transmission, i.e., ***MessageIntegrity***(msg) holds,
- The message received by a component from another component is recent regarding a pre-defined time gap between its corresponding sending and receiving actions, i.e., ***Recentness***(C_i, C_j, msg, Θ) holds, and
- There is no duplicate reception of the same message by the receiving component, i.e., ***NonDuplication***(C_j, msg) holds.

Table 5.10 summarizes the aforementioned logical specifications of safety and security objectives at the component layer.

Table 5.10: Logical Specification of Safety and Security Objectives at the Component Layer.

Objective	Notation	Logical Specification
Component-Port-Connector (CPC) availability	***CPCAvailability***	***ComponentAvailability***(C_i, λ_i, t) $\wedge$ ***ConnectorAvailability***(Conn, λ_{Conn}, t) $\wedge$ ***EventualMessageDelivery***(C_i, C_j, msg) (or ***BoundedMessageDelivery***(C_i, C_j, msg, Θ)) $\wedge$ Logical conformity holds
Message integrity	***MessageIntegrity***(msg)	***EventualMessageDelivery***(C_i, C_j, msg)
Message freshness	***MessageFreshness***(msg)	***ComponentPrecedence***(C_i, C_j) $\wedge$ ***MessageIntegrity***(msg) $\wedge$ ***Recentness***(C_i, C_j, msg, Θ) $\wedge$ ***NonDuplication***(C_j, msg)

5.4.4 Safety and Security Interplay: Conflicts Identification

Following the notions presented in Section 4.5.2, we describe examples of the interplay between safety and security properties within the same layer or across layers of the three-layered system model.

Preliminaries for Properties Analysis. Concerning the system model at the mission layer, the process of operationalization is as follows. An important assumption made concerning

the operations performed by the system is that these operations are derived from the mission descriptions themselves. More specifically, inference rules can be applied to identify the correct set of pre/post-conditions and environment trigger conditions corresponding to the operations to accomplish a mission. In this process, we rely upon both *generative* and *pruning* semantics [9]. Generative semantics disallow all the behavioral changes except those explicitly required by the mission description. In this case, the operations performed by the system can only change the relevant attributes of the system captured by the operation output attribute, thereby limiting the scope of operation execution. However, generative semantics do not support incremental reasoning concerning the partial models, where if new operations are added, there may arise inconsistencies among the operation specifications. To avoid this problem, we also use the pruning semantics that allow all the behavioral changes except for the explicitly disallowed ones. In this case, the operations are observed as restrictive executions on the state transitions associated with the system mission. A mission allocation pattern can be defined for the allocation of the independent missions M_i, $i \in \{1 \ldots \mathbb{N}\}$ to one or more independent or dependent operations Op_j, $j \in \{1 \ldots \mathbb{N}\}$ for the accomplishment of M_i, as follows:

$$((operationalSituation[\mathrm{M}_i] \wedge hazardousEvent[\mathrm{M}_i]) \Leftrightarrow (preCondition[\mathrm{Op}_j] \wedge environmentTriggerCondition[\mathrm{Op}_j])) \Rightarrow accomplishedThrough[\mathrm{M}_i, \mathrm{Op}_j]$$

Upon execution of the operation Op_j to which the mission M_i is allocated, the *overallGoal* of M_i becomes equivalent to the *postCondition* of Op_j. Formally,

$$accomplishedThrough[\mathrm{M}_i, \mathrm{Op}_j] \Rightarrow (overallGoal[\mathrm{M}_i] \Leftrightarrow postCondition[\mathrm{Op}_j])$$

Likewise, the execution of a function at the functional layer depends on the presence of the trigger and the fulfillment of pre-conditions. This ensures consistency concerning the application of the operations and execution of functions, and traceability between the functional path, sequence of operations, and their underlying missions.

Finally, at the component layer, each component C_m can perform a particular action, comprising the behavioral aspects of the system, including sending ($send(\mathrm{C}_m, msg)$) or receiving ($receive(\mathrm{C}_m, msg)$) a message msg to or from other components. These actions depend on the *kind* of the port, whose behavior is defined by the simple sending or receiving semantics. In addition, the underlying system introduces communication features like First-In-First-Out (FIFO) delivery of messages, eventual delivery of the messages, and no false message creation, which define the communication semantics. FIFO would preserve the order in which messages are sent and received and is used for illustration purposes.

Interplay within Mission Layer. Consider the use case presented in Section 3.4, where collision avoidance is the high-level mission of the CDV. The manual triggering of brakes to

avoid the collision situation may hinder the denial of manual operation access to the operator in cases where safety is prioritized over security.

In such cases, conflicts between safety (namely *operational availability*) and security (namely *system accessibility*) objectives can be identified by analyzing the predicates, more concretely, for any states in which all post-conditions are not satisfied simultaneously. For example, consider the following predicate, at the mission layer formal specification, where Op_1 := braking and Op_2 := access provisioning, respectively denote the safety- and security-critical operations performed by the CDV for the accomplishment of the mission M_1 := collision avoidance:

$$accomplishedThrough[\mathrm{M}_1, \mathrm{Op}_1, \mathrm{Op}_2] \Rightarrow (overallGoal[\mathrm{M}_1] \Leftrightarrow (\neg postCondition[\mathrm{Op}_1] \vee \neg postCondition[\mathrm{Op}_2]))$$

Using the semantics defined in the previous Section 5.4.3, the above predicate is specified as:

$$\Box(accomplishedThrough[\mathrm{M}_1, \mathrm{Op}_1, \mathrm{Op}_2] \rightarrow \Diamond(overallGoal[\mathrm{M}_1] \leftrightarrow (\neg postCondition[\mathrm{Op}_1] \vee \neg postCondition[\mathrm{Op}_2])))$$

Thus, no simultaneous fulfillment of post-conditions belonging to Op_1 and Op_2 after mission accomplishment shall indicate, in particular, a conflict between the safety and security objectives of the target system model.

Interplay between Mission and Functional Layers. Consider a situation when the trigger condition of an operation becomes true within the interval of another operations' execution. For example, the premature or unintended deployment [154] of the vehicle's airbag during braking, which is still in progress without effective crash condition. Herein, the conflict may occur at the level of high-level mission specification since the two missions' overall goals may have conflicting requirements.

In such cases, conflicts between objectives can be identified during the operations' realization (i.e., Op_j) to accomplish different missions. Hence, for the overall system model at the mission layer, the following predicate is violated as the triggering of the airbag deployment conflicts with the pre-condition for the braking operation, which is related to the lack of crash condition:

$$environmentTriggerCondition[\mathrm{Op}_j] \Rightarrow preCondition[\mathrm{Op}_j]$$

Using the semantics defined in the previous Section 5.4.3, the above predicate is specified as:

$$\Box(environmentTriggerCondition[\mathrm{Op}_j] \rightarrow \blacklozenge(preCondition[\mathrm{Op}_j]))$$

To illustrate the interplay with the functional layer, we consider the link *realizedBy* between the operations and the functions constituting the functional path (refer to Figure 5.3) and the introduction of time details. Herein, the assurance of *functional integrity* for the functional path realizing a sequence of operations is assumed. However, the delay introduced by any constituent function, i.e., deceleration, during execution may violate *functional path freshness*, influencing the timely realization of the operations, i.e., airbag deployment. Hence, the above mission-layer predicate can be analyzed with the details offered by the functional layer using the following predicate:

$$(triggerTime[\mathrm{F}_1] > 0 \wedge trigger[\mathrm{F}_2]) \Rightarrow (triggerTime[\mathrm{F}_1] + duration < triggerTime[\mathrm{F}_2])$$

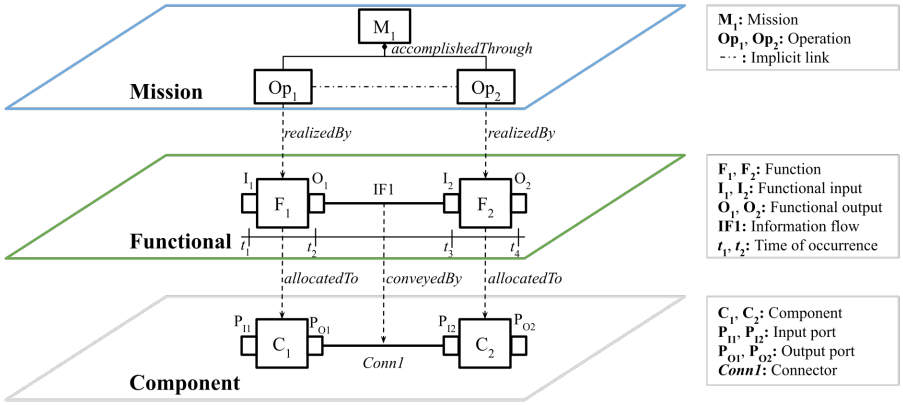


Figure 5.3: Model Instantiation to depict Relationships across Layers.

Using the semantics defined in the previous Section 5.4.3, the above predicate is specified as:

$$\Box((triggerTime[\mathrm{F}_1] > 0 \wedge trigger[\mathrm{F}_2]) \rightarrow (triggerTime[\mathrm{F}_1] + duration < triggerTime[\mathrm{F}_2]))$$

Thus, for a sequence of functions F_1 and F_2, if the time of trigger of F_1 has some positive value and the trigger for F_2 is true, then the time of trigger of F_1 summed up with its duration of execution should be less than the time of trigger of F_2.

Interplay within Functional Layer. Consider a scenario comprising three system functions, viz. F_1 := obstacle detection, F_2 := position inference, and F_3 := deceleration. For a full chain of information transmission across a function path (i.e., $\mathrm{F}_1 \rightarrow \mathrm{F}_2 \rightarrow \mathrm{F}_3$), the satisfaction of the *execution timeliness* does not necessarily guarantee *information flow timeliness*. This can be attributed to the delays introduced during the transmission of information between consecutive functions (thereby violating *information flow freshness*), which in aggregation, still satisfy the constraints for the satisfaction of *functional path availability*.

Hence, to ensure that both functional path availability and information flow freshness objectives hold, the following predicate needs to be satisfied (Refer to Figure 5.3):

$$\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k}, \mathrm{O}_k) \Rightarrow_{\Theta} \mathcal{Occ}(\mathrm{F}_{k-2}, \mathrm{F}_{k-1}, t_{2(k-1)}, \mathrm{O}_{k-1}), \forall k \geq 2$$

[**NB:** It is recalled that the sub-indices here correspond to occurrences of function output generation.]

Using the semantics defined in the previous Section 5.4.3, the above predicate is specified as:

$$\blacksquare(\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k}, \mathrm{O}_k) \rightarrow_{\Theta} \blacklozenge(\mathcal{Occ}(\mathrm{F}_{k-2}, \mathrm{F}_{k-1}, t_{2(k-1)}, \mathrm{O}_{k-1})))$$

that can be verified iff:

$$\sum_{k>=2}^{\mathbb{N}} |t_{2k} - t_{2(k-1)}| + |t_2 - t_1| \leq \Theta$$

Thus, the non-fulfillment of the predicate above will indicate the potential delays in the transmission of information between the consecutive functions, provided the individual functions respect the time-bound for execution, thereby violating the information flow freshness objective.

Interplay between Functional and Component Layers. Consider a situation comprising a sequence of functions, viz. F_1 := obstacle detection, F_1 := position inference, and F_3 := deceleration. For the transmission between consecutive functions, the fulfillment of *information flow timeliness* objective does not necessarily guarantee *execution timeliness* across the entire chain of functions. This can be attributed to the fact that an attacker can introduce delays in the packets for an underlying protocol for time synchronization, leading to failure in the measurement of end-to-end delays by the communicating nodes, thereby violating the *functional path availability*. However, with the assumption that each function execution and the communication between the consecutive functions respect the time-bound, *information flow freshness* still holds.

Hence, to ensure that both functional path availability and information flow freshness objectives hold, the following predicate needs to be satisfied:

$$\mathcal{Occ}(\mathrm{F}_1, \mathrm{F}_1, t_1, \mathrm{I}_1) \Rightarrow_{\Theta} \mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k-1}, \mathrm{I}_k)$$

Using the semantics defined in the previous Section 5.4.3, the above predicate is specified as:

$$\square(\mathcal{Occ}(\mathrm{F}_1, \mathrm{F}_1, t_1, \mathrm{I}_1) \rightarrow_{\Theta} \Diamond(\mathcal{Occ}(\mathrm{F}_{k-1}, \mathrm{F}_k, t_{2k-1}, \mathrm{I}_k)))$$

that can be verified iff:

$$\sum_{k=1}^{\mathbb{N}} |t_{2k-1} - t_1| + |t_{2k} - t_{2k-1}| \leq \Theta$$

To illustrate the interplay at the component layer, we consider the use case scenario comprising two components, viz. C_1 := processing unit and C_2 := multi-function control unit. Herein, non-fulfillment of the *component availability* objective due to failure of the component C_1 will, in turn, affect the delivery of the messages to the component C_2, influencing the *eventual message delivery* and *message integrity*. As a result, the *message freshness* objective is not met, which is crucial to prevent the replaying of the control signals. In this case, the above functional-layer predicate can be analyzed with the details offered by the component layer using the following predicate:

$$P(T > t) \wedge send(C_i, msg) \Rightarrow_t receive(C_j, msg)$$

Thus, the time to failure T of the component C_1 must appear after the time interval $[0, t]$ to ensure the fulfillment of the message integrity objective.

Interplay within Component Layer. Consider a scenario comprising three system components, viz. C_1 := processing unit, C_2 := multi-function control unit, and C_3 := brake actuator, engaged in transmitting a message msg := braking command, from C_1 to C_3 via C_2, i.e., $C_1 \longrightarrow_{msg} C_2$ and $C_2 \longrightarrow_{msg} C_3$. Herein, we consider *eventual message delivery*, influencing *CPC Availability*, and *non-duplication*, influencing *message freshness*, as the safety and security objectives to be respectively satisfied. We assume that all these components are legitimate. However, if C_2 misbehaves and becomes faulty, it may tamper with msg to msg', leading to the following flow: $C_1 \longrightarrow_{msg} C_2$ and $C_2 \longrightarrow_{msg'} C_3$.

Thus, even after tampering, the non-duplication objective is satisfied as none of the recipients (viz. C_2 and C_3) undergo the repeated transmission of the message. However, the eventual message delivery for msg is not satisfied as C_3, the intended recipient, does not receive the original message msg. Hence, in such scenarios, the safety and security analysis should not be conducted standalone but integrated to ensure corresponding objectives, especially across the entire chain of transmission.

Hence, for the overall system model targeting the component-layer details, the aspect above can be analyzed via the following predicates:

$$send(C_2, msg) \Rightarrow receive(C_3, msg) \wedge \neg(receive(C_2, msg))$$

$$send(C_1, msg) \Rightarrow receive(C_3, msg)$$

Using the semantics defined in the previous Section 5.4.3, the above predicate is specified as:

$$\Box(send(\mathrm{C}_2, msg') \rightarrow \Diamond(receive(\mathrm{C}_3, msg')) \wedge \neg(\blacklozenge(receive(\mathrm{C}_2, msg'))))$$

$$\Box(send(\mathrm{C}_1, msg) \rightarrow \Diamond(receive(\mathrm{C}_3, msg)) \wedge \neg(\blacklozenge(receive(\mathrm{C}_3, msg))))$$

The first predicate is not expected to be satisfied; otherwise, the message sent by C_2 to C_3 is not the same as the one sent by C_1 to C_2. The second predicate should be satisfied whenever the uniqueness of the message is expected to occur.

As a starting point, such predicates can assist engineers in conducting sound analysis, as presented in the next section, by verifying the system's safety and security trade-offs.

5.5 Formal Modeling and Verification in Event-B

In this section, we describe the interpretation of the structural and behavioral aspects of the target system model at different layers and safety and security properties in Event-B, using the formalisms and logical specifications presented in Section 5.4. Event-B was selected as the target language given that it fulfills the desired features for the formalisms and properties analysis, as mentioned in Section 5.3.

5.5.1 Introduction to Event-B

Event-B is a formal method for system-level modeling and analysis, wherein the model allows to capture the structural and behavioral aspects of a discrete state transition system [46]. It facilitates the use of set theory as a modeling notation, refinement to represent the system at different levels of abstraction, and mathematical proof to verify the consistency between the refinement levels. An Event-B model comprises: 1) *Context* to model the static structural aspects of the system via *sets*, *constants*, *axioms*, and *theorems*, as depicted in Figure 5.4 and 2) *Machine* to capture the dynamic behavioral aspects of the system via *variables*, *invariants*, *variants*, and *events*, as depicted in Figure 5.5. Variables v define the state of the machine and are constrained by the invariants I(v). An event e comprises event parameters p, guards G(p, v) representing its enabling condition, and action A(p, v) describing the evolution of the state variables on the execution of the event, as depicted in Figure 5.6. Accordingly, the event e has the following form:

any p **when** G(p, v) **then** A(p, v) **end**

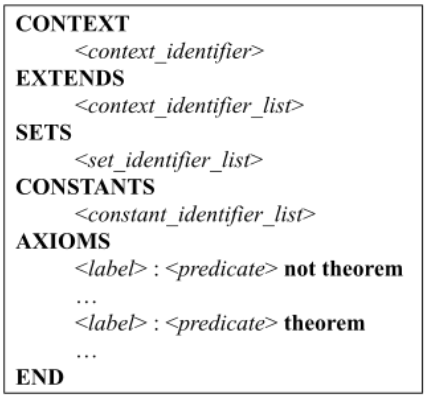


Figure 5.4: Structure of an Event-B Context.

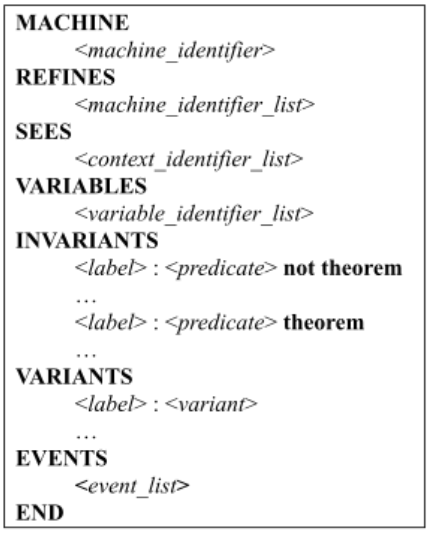


Figure 5.5: Structure of an Event-B Machine.

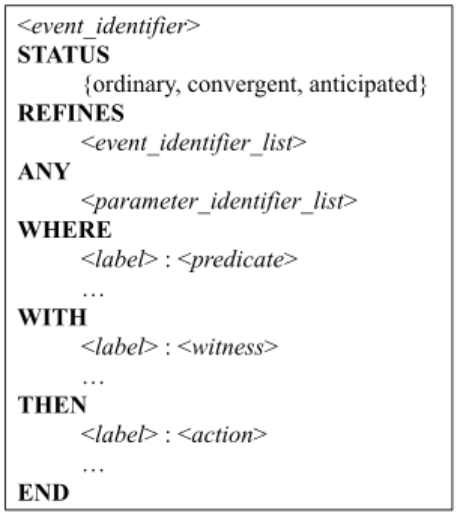


Figure 5.6: Structure of an Event-B Event.

We rely upon mathematical proofs comprising a set of rules to reason and verify the invariants. These rules are based upon the convention of Proof-Obligations (POs). A PO is generated for every invariant that can be affected by an event, i.e., the invariant contains variables that an event can change. In this case, we rely upon PO rules for invariant preservation that take the form:

event_name/invariant_label/INV

The notion of *extends*, *sees*, and *refines* facilitate defining relationships between the contexts, machines, and events that would serve for instantiating the system model for a target application design and properties analysis. We use the open-source tool Rodin to support the Event-B method. In Event-B, the specification and the proofs are separated, and Rodin is designed to support these two parts.

In the following sub-sections, we leverage the aforementioned constructs and features of the Event-B framework to provide a concrete specification of the three-layered system and properties models, facilitating integrated safety and security analysis. For more details regarding Event-B constructs, the reader is referred to Appendix C.

5.5.2 Three-layered System Formal Modeling

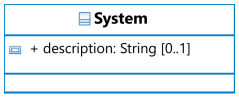
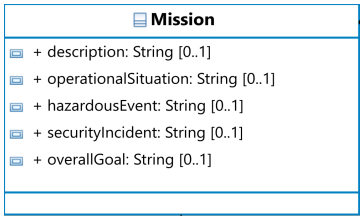
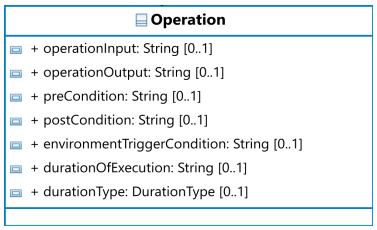
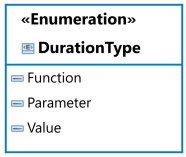
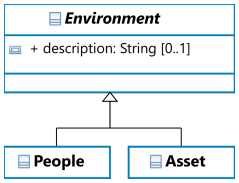
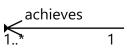
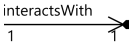
The formal interpretation of the three-layered system DSMLs presented in Section 4.4 into Event-B relies upon the corresponding logical specifications presented in Section 5.4.1 and is detailed in the following paragraphs. It is recalled that the logical specifications are the prerequisites, defining rules for facilitating the interpretation of the modeling concepts to their formal counterparts. Event-B is a target language and tool selected to instantiate and show such interpretation.

5.5.2.1 Mission Layer

The interpretation of the mission-layer system DSML to Event-B considers the meta-model depicted in Figure 4.2 and the mapping described in Table 5.1. The interpretation is described in Table 5.11, along with the rationale and the elements and relationships in the DSML.

Structural Elements. To provide a formal declaration of the structural elements of the mission-layer system DSML, we define a context `C0MissionView`. Herein, the key elements, including **System**, **Mission**, **Operation**, **Environment**, and the enumeration **DurationType** are defined as Event-B sets to model their respective classes or user-defined types. For instance, the `Mission` set represents the collection of pre-defined system mission instances that can be either

Table 5.11: Interpretation: Mission-layer System DSML ↦ Event-B.

DSML Element	Event-B Element	Rationale
	System (SET) *description**	*Required only for informal specification
	Mission (SET) *description** operationalSituation, hazardousEvent, securityIncident, overallGoal (VARIABLES)** operationalSituation ∈ Mission → BOOL (INVARIANTS) hazardousEvent ∈ Mission → BOOL securityIncident ∈ Mission → BOOL overallGoal ∈ Mission → BOOL	*Required only for informal specification **Split the mission description as per the form: *PreCondition* ⇒ *PostCondition*, for formal analysis
	Operation (SET) *operationInput*, *operationOutput** preCondition, postCondition, environmentTriggerCondition, counter** (VARIABLES) preCondition ∈ Operation → BOOL (INVARIANTS) postCondition ∈ Operation → BOOL environmentTriggerCondition ∈ Operation → BOOL counter ∈ ℤ ∧ (0 ≤ counter) ∧ (counter ≤ *n****)	*Required only for informal specification **Represents *durationOfExecution* ****n* = number of events
	DurationType (SET) Function, Parameter, Value (CONSTANTS) Function ∈ ℙ (DurationType) (AXIOMS) Parameter ∈ ℙ (DurationType) Value ∈ ℙ (DurationType) partition(DurationType, Function, Parameter, Value)*	*Function, Parameter, and Value partition the DurationType; i.e., (DurationType = Function ∪ Parameter ∪ Value) ∧ (Function ∩ Parameter ∩ Value = ϕ)
	Environment (SET) People, Asset (CONSTANTS) People ∈ ℙ (Environment) (AXIOMS) Asset ∈ ℙ (Environment) partition(Environment, People, Asset)*	*People and Asset partition the Environment; i.e., (Environment = People ∪ Asset) ∧ (People ∩ Asset = ϕ)
	achieves (VARIABLE) achieves ∈ System → Mission (INVARIANTS) achieves∼∈ Mission ↠ System	A system can achieve one or more missions
	interactsWith (VARIABLE) interactsWith ∈ System → Environment (INVARIANT)	A system interacts with its environment
	accomplishedThrough (VARIABLE) accomplishedThrough ∈ Mission → Operation (INVARIANTS) accomplishedThrough∼∈ Operation ↠ Mission	A mission can be accomplished through one or more operations

operational, safety, or security-related. This is a choice to represent the domain notions that are, in essence, static regarding the proposed three-layered model. Likewise, the child classes, e.g., **People**, **Asset**, and enumeration values, e.g., *Parameter*, *Value*, are represented as constants. Herein, `Parameter` and `Value` capture the sub-sets of the objects of related types. To model the relation between a carrier set $\mathbb{S}$ (e.g., `Environment`) and the constants s_1, s_2, ..., s_n (i.e., `People`, `Asset`), the *partition* operator is used; i.e., *partition*($\mathbb{S}$, s_1, s_2, ..., s_n). Herein, `People` and `Asset` represent the mutually disjoint sub-sets of the set `Environment`. An excerpt of the context `C0MissionView` is depicted in Listing 5.1.

```
// Mission layer structural aspects
CONTEXT C0MissionView
SETS
    Mission , Operation , Environment , DurationType
CONSTANTS
    People , Asset , Parameter , Value
AXIOMS
    People ∈ ℙ (Environment)
    Asset ∈ ℙ (Environment)
    partition(Environment , People , Asset)
END
```

Listing 5.1: Excerpt of Event-B context at the mission layer.

To model the instances for undefined sets[1], we define a context `C2MissionUtility` that provides declaration of the utility constants[2], e.g., `s1`, `m1`, `o1`, `p1`, and `a1` for a generic representation of system, mission, operation, people, and asset, respectively. Herein, the *extends* keyword is used for inheriting the elements, including axioms, from `C0MissionView` to `C2MissionUtility`. An excerpt of this context is depicted in Listing 5.2.

```
// Mission layer utility constants
CONTEXT C2MissionUtility
EXTENDS C0MissionView
CONSTANTS
    m1
AXIOMS
    m1 ∈ Mission ∧ Mission = {m1} // m1 is of type Mission and is the only
    // element in the Mission set
END
```

Listing 5.2: Excerpt of Event-B context defining utility constants at the mission layer.

[1] Sets with no pre-defined concrete values. The end-users can introduce the values that must respect the axioms defined in the context.

[2] Constants used for generic representation of set instances.

Behavioral Elements. We apply the following refinement strategy at the mission layer. The initial abstract specification `M0MissionView` formally captures the desired behavioral aspects of the system at this layer. Moreover, it introduces the safety and security objectives to be verified, which will be discussed in the forthcoming Section 5.5.3. Subsequently, the first refinement, `M1MissionLayerInstance`, helps to illustrate how the former model is instantiated. The refinement is also based upon the CDV use case, which is further detailed in Chapter 6.

The machine `M0MissionView` is described as follows. It uses the *sees* keyword to inherit the axioms of the context `C2MissionUtility`, and in turn `C0MissionView`, in conjunction as hypotheses in the mathematical proofs. Herein, the attributes like *hazardousEvent* and *preCondition*, corresponding to *Mission* and *Operation* in the respective DSML are defined as variables. To type these variables, we define invariants that capture their relationships in the DSML, which can be verified either in the same machine or later refinements. In addition, we define constraints presented in Section 5.4.4 for properties analysis as model invariants to restrict the model's behavior. The events are defined to capture the state transitions associated with the system missions and operations. For instance, the `INITIALIZATION` event assigns an initial value to the variables. The guards in the `when` section, corresponding to the events, restrict the values of the variables as enabling conditions for the events. The `INITIALIZATION` event has no guards, implying that it is always possible. An excerpt of the machine `M0MissionView` is depicted in Listing 5.3.

```
// Mission layer behavioral aspects
MACHINE M0MissionView
SEES C2MissionUtility
VARIABLES operationalSituation , hazardousEvent , environmentTriggerCondition ,
counter , achieves , accomplishedThrough , interactsWith , privilege , access
INVARIANTS
    operationalSituation ∈ Mission → BOOL
    counter ∈ ℤ ∧ (0 ≤ counter) ∧ (counter ≤ n) //n = no. of events
    privilege ∈ ℙ(People x System x Operation)
    access ∈ (People x Operation) → BOOL
EVENTS
    INITIALIZATION
    STATUS
        ordinary
    BEGIN
        operationalSituation := {m1 ↦ TRUE}
        ...
    END
    EVENT1
    STATUS
```

```
        ordinary
    WHERE
        operationalSituation = {m1 ↦ TRUE}
        hazardousEvent = {m1 ↦ FALSE}
        environmentTriggerCondition = {o1 ↦ FALSE}
        achieves = {s1 ↦ m1}
        counter > 0
    THEN
        hazardousEvent ≔ {m1 ↦ TRUE}
        environmentTriggerCondition ≔ {o1 ↦ TRUE}
        accomplishedThrough ≔ {m1 ↦ o1}
        interactsWith ≔ {s1 ↦ a1}
        counter ≔ counter − 1
    END
END
```

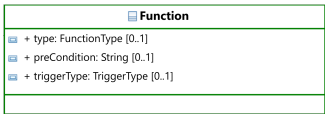
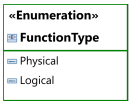
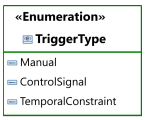
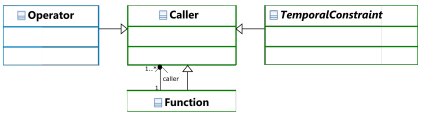
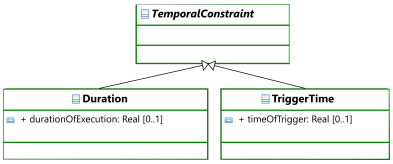
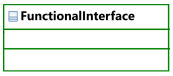
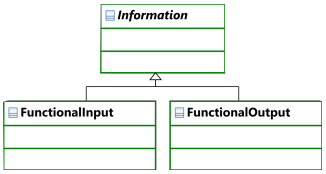
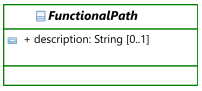
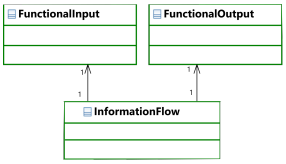
Listing 5.3: Excerpt of Event-B machine at the mission layer.

5.5.2.2 Functional Layer

The interpretation of the functional-layer system DSML to Event-B considers the meta-model depicted in Figure 4.3 and the mapping described in Table 5.2. The interpretation is described in Table 5.12, along with the rationale and the elements and relationships in the DSML.

Structural Elements. To provide a formal declaration of the structural elements in the function-layer system DSML, we define a context `C5FunctionView`. Herein, the key elements, including **Function**, **Information**, **Caller**, and the enumeration **FunctionType** and **TriggerType** are defined as Event-B sets to model their respective classes. For instance, the `Function` set represents the collection of pre-defined system functions. Likewise, the child classes, e.g., **FunctionalInput**, **FunctionalOutput**, and enumeration values, e.g., *Physical*, *Logical*, are represented as constants. The relation between these constants and their corresponding carrier sets is represented using the *partition* operation, e.g., `partition(Information, FunctionalInput, FunctionalOutput)`. To model a one-to-one relation between `InformationFlow` and `FunctionalInput`—for instance, we use the bijections denoted by ⤖. Bijections will allow specifying that an information flow is associated with a functional input and one functional input constitutes at least one information flow. This is a choice to maintain distinctiveness and facilitate traceability of the state transitions when information passes across a functional path. Similarly, a functional path defined via information flows between the functions is modeled using the total surjective relation ⇼, where

Table 5.12: Interpretation: Functional-layer System DSML ↦ Event-B.

DSML Element	Event-B Element	Rationale
	Function (SET) type, triggerType (CONSTANTS) *preCondition** type ∈ Function ⇸ FunctionType (AXIOMS) triggerType ∈ Function ⇸ TriggerType	*Required only for information specification
	FunctionType (SET) Physical, Logical (CONSTANTS) Physical ∈ ℙ (FunctionType) (AXIOMS) Logical ∈ ℙ (FunctionType) partition(FunctionType, Physical, Logical)	Function can be of type physical or logical
	TriggerType (SET) Manual, ControlSignal, TemporalConstraint (CONSTANTS) Manual ∈ ℙ (TriggerType) (AXIOMS) ControlSignal ∈ ℙ (TriggerType) TemporalConstraint ∈ ℙ (TriggerType) partition(TriggerType, Manual, ControlSignal, TemporalConstraint)	Trigger type can be manual, control signal, or temporal constraint
	Caller (SET) Operator, ExternalFunction, caller (CONSTANTS) Operator ∈ ℙ (Caller) (AXIOMS) ExternalFunction ∈ ℙ (Caller) partition(Caller, Operator, ExternalFunction)* caller ∈ Function → Caller caller∼∈ Caller ⇸ Function	*Caller can be an operator, function or temporal constraint; Herein, function is represented as ExternalFunction, operation is inherited from mission model and temporal constraint is captured as a behavioral element
	Duration, TriggerTime (CONSTANTS) Duration ∈ ℕ (TemporalConstraint) (AXIOMS) TriggerTime ∈ ℙ (TemporalConstraint) partition(TemporalConstraint, Duration, TriggerTime) triggerTime ∈ Function → ℕ (INVARIANTS) time* ∈ ℕ	*time represents the current system time on the global time scale, traversing delays from the beginning
	FunctionalInterface (SET)	Semantics are defined via passage of information
	Information (SET) FunctionalInput, FunctionalOutput (CONSTANTS) FunctionalInput ∈ ℙ (Information) (AXIOMS) FunctionalOutput ∈ ℙ (Information) partition(Information, FunctionalInput, FunctionalOutput)	Abstract class Information partitioned into FunctionalInput and FunctionalOutput
	FunctionalPath (CONSTANT) FunctionalPath ∈ Function → Function (AXIOM)	Functional path as a sequence of objects of type Function
	sysFunc (CONSTANT) sysFunc ∈ System ⇸ Function (AXIOMS) sysFunc∼∈ Function ⇸ System	A system may execute one or more functions
	funcInfo (CONSTANT) funcInfo ∈ Function ⇸ Information (AXIOM) funcInfo∼∈ Information ⇸ Function (AXIOM)	One or more information elements may be associated with a function
	infoFI (CONSTANT) infoFI ∈ Information ⇸ FunctionalInterface (AXIOM)	A functional interface will allow passage of one information element at a time
	functionalInterfaces (CONSTANT) functionalInterfaces ∈ Function → FunctionalInterface functionalInterfaces∼∈ FunctionalInterface ⇸ Function (AXIOMS)	A function may have one or more functional interfaces
	InformationFlow (SET)* flowFI, flowFO (CONSTANTS) flowFI ∈ InformationFlow ⇸ FunctionalInput (AXIOMS) flowFO ∈ InformationFlow ⇸ FunctionalOutput	*Dynamically represented as interacting events; An information flow is associated with one functional input or one functional output
	pathFlow (CONSTANT) pathFlow ∈ FunctionalPath ⇸ InformationFlow (AXIOMS) pathFlow∼∈ InformationFlow ⇸ FunctionalPath	A functional path links functions via one or more information flows

every information flow has at least one functional path through which it passes, and a partial function ⇸ as an additional constraint to derive the injective relation. An excerpt of the context `C5FunctionView` is depicted in Listing 5.4.

```
// Functional layer structural aspects
CONTEXT C5FunctionView
SETS
    Function , Information , Caller , InformationFlow
CONSTANTS
    FunctionalInput , FunctionalOutput , Physical , Logical , flowFI , flowFO
AXIOMS
    FunctionalInput ∈ ℙ (Information)
    FunctionalOutput ∈ ℙ (Information)
    partition(Information , FunctionalInput , FunctionalOutput)
    flowFI ∈ InformationFlow ⇸ FunctionalInput
    flowFO ∈ InformationFlow ⇸ FunctionalOutput
END
```

Listing 5.4: Excerpt of Event-B context at the functional layer.

To model the instances for undefined sets, we define a context `C7FunctionUtility` that provides a declaration of the utility constants, e.g., `f1`, `f2`, and `info` for a generic representation of functions and information, etc. Herein, the *extends* keyword is used for inheriting the elements, including axioms, from `C5FunctionView` to `C7FunctionUtility`. An excerpt of this context is depicted in Listing 5.5.

```
// Functional layer utility constants
CONTEXT C7FunctionUtility
EXTENDS C5FunctionView
CONSTANTS
    f1 , f2 , info
AXIOMS
    f1 ∈ Function ∧ f2 ∈ Function ∧ Function = {f1 , f2} // f1 and f2 are of
    // type Function
    info ∈ Information
END
```

Listing 5.5: Excerpt of Event-B context defining utility constants at the functional layer.

Behavioral Elements. We apply the following refinement strategy at the functional layer:

- The initial abstract specification `M2FunctionView` formally captures the desired behavioral aspects of the system at this layer, including the abstract function definitions.

The machine uses the *sees* keyword to inherit the axioms of the context `C7FunctionUtility`, and in turn `C5FunctionView`, in conjunction as hypotheses in the mathematical proofs. Herein, we define the following seven variables to capture the state transitions associated with function execution:

- `trigger` is a flag corresponding to a function that encodes the assumptions related to the ordering of functions constituting a functional path in the form of a Boolean variable. It denotes whether a function execution has taken place or not in the abstract model.
- `triggerTime` represents the occurrence time of a function call and is typed as a natural to capture the passage of time in the form of discrete ticks.
- `duration` denotes a natural variable denoting the discrete gap between the occurrence of events corresponding to the function execution, i.e., function call and function completion.
- `time` represents the current system time on the global time scale.
- `functionalInput` and `functionalOutput` represent the input and output, respectively, associated with a function.
- Finally, `var` represents an information variable used for the passage of information between two functions.

To type these variables, we define invariants that capture the relationships between the elements represented by the context `C5FunctionView`. To ensure the consistency of the execution of functions with the trigger time, we define an invariant `consistencyOfOccurrence`. Likewise, to denote the minimal gap delta between the execution of functions, we define an invariant `occuranceGap`. Moreover, we define events to capture the state transitions associated with the system functions. The state transitions involve functional call, function in progress, and function termination. An excerpt of the machine `M2FunctionView` is depicted in Listing 5.6.

```
// Functional layer behavioral aspects (abstract)
MACHINE M2FunctionView
SEES C7FunctionUtility
VARIABLES trigger, triggerTime, duration, time, functionalInput,
functionalOutput, var
INVARIANTS
    trigger ∈ Function → BOOL
    triggerTime ∈ Function → ℕ
    duration ∈ ℕ
```

```
        ∀fk.fk ∈ Function ∧ triggerTime(fk) > 0 ⇔ trigger[{fk}] =
        {TRUE} // consistencyOfOccurrence
    EVENTS
        INITIALIZATION
        STATUS
            ordinary
        BEGIN
            trigger ≔ {f1 ↦ FALSE}
            triggerTime ≔ {f1 ↦ 0}
            ...
        END
        FUNCTION1
        STATUS
            ordinary
        ANY
            tick
            o1 // functional output local to this event
        WHERE
            trigger = {f1 ↦ FALSE}
            functionalInput = {f1 ↦ i1}
            o1 ∈ FunctionalOutput
            tick ∈ ℕ1
        THEN
            trigger ≔ {f1 ↦ FALSE}
            triggerTime(f1) ≔ time + tick
            duration ≔ duration + tick
            functionalOutput ≔ {f1 ↦ o1}
            var ≔ o1
        END
    END
```

Listing 5.6: Excerpt of Event-B abstract machine at the functional layer.

- Subsequently, the first refinement `M3FunctionViewRefinement`, considers the invocation and termination of the functions in a concrete way. Moreover, it introduces the safety and security objectives to be verified, which will be discussed in the forthcoming Section 5.5.3.

  This machine uses the *refines* keyword to concretize the machine `M2FunctionView`. Some of the variables defined herein include the following:

  - `beginFunc` and `endFunc` are flags in the form of Boolean variables corresponding to the occurrences that respectively mark the beginning and ending events of a function

execution, as refinement of the `trigger` from the abstract machine.

- Likewise, `beginTimeFunc` and `endTimeFunc` represent the occurrence time corresponding to these events and are typed as natural to capture the passage of time in the form of discrete ticks.
- `deadlineFunc` denotes the instant when the results must be produced by a function, and is typed as a natural.

Apart from the invariants to type these variables and restrict the system's behavior, we define gluing invariants to establish the relationship between the variables of the abstract and concrete models. For example, `endFunc` holds the same value regarding the variable `trigger`. The events defined to capture the state transitions associated with the function execution, concretize the events presented in the abstract machine using the *refines* keyword. An excerpt of the machine `M3FunctionViewRefinement` is depicted in Listing 5.7.

```
// Functional layer behavioral aspects (concrete)
MACHINE M3FunctionViewRefinement
REFINES M2FunctionView
SEES C7FunctionUtility
VARIABLES beginFunc, endFunc, beginTimeFunc, endTimeFunc,
deadlineFunc
INVARIANTS
    beginFunc ∈ Function → BOOL
    endFunc ∈ Function → BOOL
    beginTimeFunc ∈ Function → ℕ
    endTimeFunc ∈ Function → ℕ
    deadlineFunc ∈ Function → ℕ1
    ∀fk.fk ∈ Function ∧ endFunc[{fk}] = {TRUE} ⇔ trigger[{fk}]
    = {TRUE} // gluing invariant
    ∀fk.fk ∈ Function ∧ endTimeFunc[{fk}] = triggerTime[{fk}]
    ∀fk.fk ∈ Function ∧ beginTimeFunc(fk) > 0 ⇔ beginFunc[{fk}]
    = {TRUE} //consistencyOfOccurrence
    ∀fk.fk ∈ Function ∧ endTimeFunc(fk) > 0 ⇔ endTimeFunc[{fk}]
    = {TRUE}
EVENTS
    INITIALIZATION
    STATUS
        ordinary
    BEGIN
        beginFunc ≔ {f1 ↦ FALSE}
        beginTimeFunc ≔ {f1 ↦ 0}
```

```
        ...
    END
    FUNC1_BEGIN
    STATUS
        ordinary
    ANY
        tick
    WHERE
        beginFunc = {f1 ↦ FALSE}
        tick ∈ ℕ1
    THEN
        duration ≔ duration + tick
        beginTimeFunc(f1) ≔ time + duration
        beginFunc ≔ {f1 ↦ TRUE}
        functionalInput ≔ {f1 ↦ i1}
    END
    FUNC1_END
    STATUS
        ordinary
    REFINES
        FUNCTION1
    ANY
        tick
        o
    WHERE
        beginFunc = {f1 ↦ TRUE}
        endFunc = {f1 ↦ FALSE}
        beginFunc = {f2 ↦ FALSE}
        tick ∈ ℕ1
        duration = duration + tick
        time ≤ beginTimeFunc(f1) + deadlineFunc(f1) + duration
        o ∈ FunctionalOutput
    WITH
        o = o1
    THEN
        endFunc ≔ {f1 ↦ TRUE}
        endTimeFunc(f1) ≔ time
        functionalOutput ≔ {f1 ↦ o}
        var ≔ o
    END
END
```

Listing 5.7: Excerpt of Event-B concrete machine at the functional layer.

- Finally, the last refinement, `M4FunctionalLayerInstance` will instantiate the proposed model for the CDV use case, which will be detailed in Chapter 6.

5.5.2.3 Component Layer

The interpretation of the component-layer system DSML to Event-B considers the meta-model depicted in Figure 4.4 and the mapping described in Table 5.3. The interpretation is described in Table 5.13, along with the rationale and the elements and relationships in the DSML.

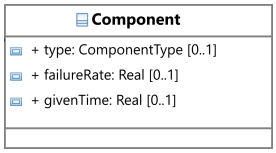
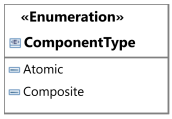
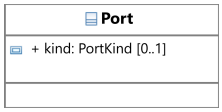
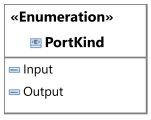
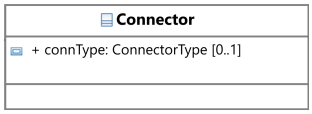
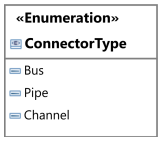
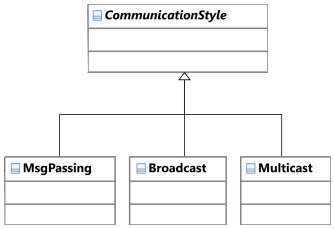
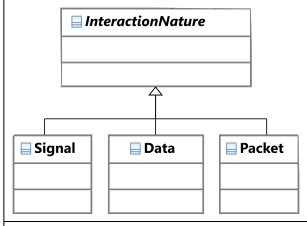
Structural Elements. To provide formal declaration of the structural elements of the component-layer system DSML, we define a context `C10ComponentView`. Herein, the key elements, including **Component**, **Port**, and the enumerations **ComponentType**, **PortKind**, and **ConnectorType** are defined as Event-B sets to model their respective classes. For instance, the `Component` set represents the collection of system components. Likewise, **CommunicationStyle** and **InteractionType** are captured as pre-defined sets, with their child classes, like **MsgPassing**, **Data**, and enumeration values, e.g., *Atomic*, *Input*, *Bus*, as constants. The relation between these constants and the corresponding carrier sets is represented using the *partition* operation, e.g., `partition(CommunicationStyle, MsgPassing, Broadcast, Multicast)`. Other relationships are accordingly modeled. An excerpt of the context `C10ComponentView` is depicted in Listing 5.8.

```
// Component layer structural aspects
CONTEXT C10ComponentView
SETS
    Component, Port, Connector, CommunicationStyle
CONSTANTS
    compType, Atomic, Composite
AXIOMS
    compType ∈ Component ↠ ComponentType
    Atomic ∈ ℙ (ComponentType)
    Composite ∈ ℙ (ComponentType)
    partition(ComponentType, Atomic, Composite)
END
```

Listing 5.8: Excerpt of Event-B context at the component layer.

Behavioral Elements. To model the system behavior, we consider discrete-time as a parameter to express the instants of the occurrence of actions. Execution of a system is a sequence of steps (instants), where a step is determined by two successive time points. We

Table 5.13: Interpretation: Component-layer System DSML ↦ Event-B.

DSML Element	Event-B Element	Rationale
	Component (SET) compType, failureRate, givenTime (CONSTANTS) compType ∈ Component ⇸ ComponentType (AXIOMS) failureRate ∈ ℕ1* givenTime ∈ ℕ1	*Positive natural number
	ComponentType (SET) Atomic, Composite (CONSTANTS) Atomic ∈ ℙ (ComponentType) (AXIOMS) Composite ∈ ℙ (ComponentType) partition(FunctionType, Atomic, Composite)	Component can be of type atomic or composite
	Port (SET) kind (CONSTANT) kind ∈ Port ⇻ PortKind (AXIOM)	Each port has a kind
	PortKind (SET) Input, Output (CONSTANTS) Input ∈ ℙ (PortKind) (AXIOMS) Output ∈ ℙ (PortKind) partition(PortKind, Input, Output)	Port can be of kind input or output
	Connector (SET) connType (CONSTANT) connType ∈ Connector ⇻ ConnectorType (AXIOM)	Each connector has a type
	ConnectorType (SET) Bus, Pipe, Channel (CONSTANTS) Bus ∈ ℙ (ConnectorType) (AXIOMS) Pipe ∈ ℙ (ConnectorType) Channel ∈ ℙ (ConnectorType) partition(ConnectorType, Bus, Pipe, Channel)	Connector can be of type bus, pipe, or channel
	CommunicationStyle (SET) MsgPassing, Broadcast, Multicast (CONSTANTS) MsgPassing ∈ ℙ (CommunicationStyle) (AXIOMS) Broadcast ∈ ℙ (CommunicationStyle) Multicast ∈ ℙ (CommunicationStyle) partition(CommunicationStyle, MsgPassing, Broadcast, Multicast)	Communication style can be message passing, broadcast, or multi-cast
	InteractionType (SET) Signal, Data, Packet (CONSTANTS) Signal ∈ ℙ (InteractionType) (AXIOMS) Data ∈ ℙ (InteractionType) Packet ∈ ℙ (InteractionType) partition(InteractionType, Signal, Data, Packet)	Interaction type can be signal, data, or packet
	sysComp (CONSTANT) sysComp ∈ System ⇻ Component (AXIOMS) sysComp∼∈ Component ⇸ System	A system may comprise one or more components
	uses (CONSTANT) uses ∈ Component → Port (AXIOMS) uses∼∈ Port ⇸ Component	A component uses one or more ports
	connects (CONSTANT) connects ∈ Connector → Port (AXIOMS) connects∼∈ Port ⇸ Connector	A connector connects ports

define a context C13ComponentMessaging to express time in a discrete sense, referred as Tick, where time is explicitly modeled as a set of discrete ordered Tick instants, as depicted in Listing 5.9.

```
// Component layer messaging module
CONTEXT C13ComponentMessaging
EXTENDS C10ComponentView
SETS
    Msg
CONSTANTS
    from , to , sent , received , Tick
AXIOMS
    from ∈ Msg ↠ Component
    to ∈ Msg ↠ Component
    Tick ⊆ ℕ
    sent ∈ Msg → Tick
    received ∈ Msg → Tick
END
```

Listing 5.9: Excerpt of messaging module at the component layer.

Constraints are defined to restrict the interactions between the components in the form of invariants. For example, a port must belong to at most one component, and each port has a kind from the domain PortKind. An excerpt for such invariants is depicted in Listing 5.10.

```
INVARIANTS
    failureRate ∈ ℕ // failure rate of a physical component
    ∀c1.∀c2.∀p.c1 ∈ Component ∧ c2 ∈ Component ∧ p ∈ Port ∧ uses[{c1}] = {p}
    ⇒ ¬ uses[{c2}] = {p} // port not shared
    ∀c.c ∈ Component ∧ finite(uses[{c}]) // finite ports
    ∀con.con ∈ Connector ∧ card(connects[{con}]) ≥ 2 // Connector connects
    //more than two ports
```

Listing 5.10: Excerpt of invariants at the component layer to declare components' interaction constraints.

5.5.3 Safety and Security Properties and Interplay Formal Modeling

In this section, we describe the formal modeling of safety and security properties and their interplay in Event-B.

Safety and Security Properties. The interpretation of the properties DSML to Event-B considers the meta-models depicted in Figures 4.10 and 4.11 and the mapping presented in Table

5.4. The interpretation is described in Table 5.14, along with the rationale and the elements and relationships in the DSML.

We define the safety and security objectives presented in Section 5.4.3 in Event-B in the form of model *invariants* to verify that the events satisfy the safety and security properties at different layers. Herein, the invariants are represented as the combination of the following:

1. Utility constants that are represented as lowercase alphanumeric characters (e.g., m, f, $c1$, $c2$) and set-theoretic symbols, like $\in$ that provide a generic representation and instantiation of the notions for the system model at the three layers.

2. Quantifiers, like $\forall$ and $\exists$, to define the extent to which the invariant is true concerning the utility constants.

3. Attribute variables to define the state of the notions for the system model at the three layers. The domain of attribute variables can be a boolean set (BOOL), a set of integers ($\mathbb{Z}$), a set of natural numbers ($\mathbb{N}$), or a set of positive natural numbers ($\mathbb{N}_1$).

4. Logic symbols, like $\Rightarrow$ and $\Leftrightarrow$, to represent the FOL operators between the attribute variables.

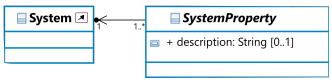
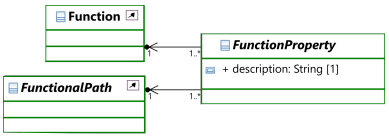
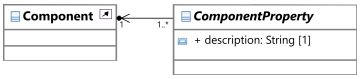
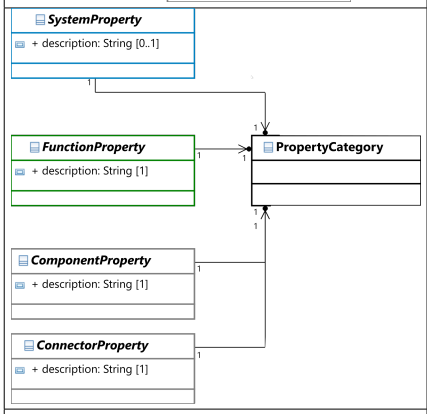
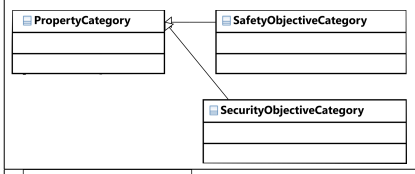
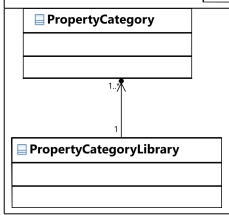
5. Relations, like $\mapsto$, to expand the multi-argument attributes by expressing the relationship between the arguments.

6. Arithmetic operators, like $+$, and relational operators, like $<$ and $\leq$ to express the relationship between numeric attribute variables.

In addition, the modal operators are translated to Event-B using the notion of guards. In this case, an event is allowed to occur if the guard is true. Consider two events, E_1 and E_2, with guards g_1 and g_2, respectively, as part of the system model. To model $\circ E_1$, according to which E_1 must be the event to be executed next, we define an additional guard, i.e., $\neg g_2$ as a constraint to prohibit the simultaneous or prior execution of E_2. Thus, the overall guard on E_1 becomes $g_1 \wedge \neg g_2$. A similar notion is presented in [155] to model obligations in Event-B.

Likewise, to model $\Diamond_{\leq \Theta} E_1$, according to which E_1 must be executed not beyond the pre-defined threshold gap Θ, we define an integer variable n corresponding to this threshold that represents the number of events defined in the system model and an integer counter $counter_{E_1}$. Whenever trigger g_1 is TRUE, $counter_{E_1}$ is set to n and is decremented on every execution of other events. If $counter_{E_1}$ becomes zero and g_1 is still true, other events will be disabled and E_1 will be executed. Otherwise, if g_1 becomes false with $counter_{E_1}$ still active and non-zero, it is reset to n. The unbounded case $\Diamond E_1$, according to which E_1 will be executed eventually, is

Table 5.14: Interpretation: Properties DSML ↦ Event-B.

DSML Element	Event-B Element	Rationale
	SystemProperty (SET) *description** systemProperty (CONSTANT) systemProperty ∈ SystemProperty ⇸ System (AXIOM)	*Required only for informal specification
	FunctionProperty (SET) *description** functionProperty, functionalPathProperty (CONSTANTS) functionProperty ∈ FunctionProperty ⇸ Function (AXIOMS) functionPathProperty ∈ FunctionProperty ⇸ FunctionalPath	*Required only for informal specification
	ComponentProperty (SET) *description** componentProperty (CONSTANT) componentProperty ∈ ComponentProperty ⇸ Component (AXIOM)	*Required only for informal specification
	ConnectorProperty (SET) *description** connectorProperty (CONSTANT) connectorProperty ∈ ConnectorProperty ⇸ Connector (AXIOM)	*Required only for informal specification
	propertyCategory (CONSTANT) propertyCategory ∈ SystemProperty ⇸ PropertyCategory (AXIOMS) propertyCategory ∈ FunctionProperty ⇸ PropertyCategory propertyCategory ∈ ComponentProperty ⇸ PropertyCategory propertyCategory ∈ ConnectorProperty ⇸ PropertyCategory	A property is associated with a property category
	PropertyCategory (SET) SafetyObjectiveCategory, SecurityObjectiveCategory (CONSTANTS) SafetyObjectiveCategory ∈ ℙ (PropertyCategory) (AXIOMS) SecurityObjectiveCategory ∈ ℙ (PropertyCategory) partition(PropertyCategory, SafetyObjectiveCategory, SecurityObjectiveCategory)	A property category can be either safety or security objective category
	PropertyCategoryLibrary (SET) propertyCategoryLibrary (CONSTANT) propertyCategoryLibrary ∈ PropertyCategoryLibrary ⇸ PropertyCategory (AXIOMS) propertyCategoryLibrary∼∈ propertyCategory ⇸ PropertyCategoryLibrary	A property category library comprise one or more property categories

modeled by considering a non-deterministic unbounded n. Herein, n is chosen once the guard g_1 becomes true.

Listing 5.11 depicts the Event-B interpretation for the *operational availability* objective at the mission layer in the safety context.

```
// Operational availability
INVARIANTS
∀m.∃o.m ∈ Mission ∧ o ∈ Operation ∧ (operationalSituation[{m}] = {TRUE} ∧
hazardousEvent[{m}] = {TRUE}) ⇒ accomplishedThrough[{m}] = {o}
```

Listing 5.11: Operational availability objective as Event-B invariant.

Likewise, in the security context, the Event-B invariant corresponding to *System Accessibility* objective is depicted in Listing 5.12.

```
// System accessibility
INVARIANTS
∀p.∀s.∀o.p ∈ People ∧ s ∈ System ∧ o ∈ Operation ∧ p ↦ s ↦ o ∉ privilege
⇒ access[{p ↦ o}] = {FALSE}
```

Listing 5.12: System accessibility objective as Event-B invariant.

Of particular interest here, objectives, like *functional integrity* at the functional layer, are defined as extensions of basic properties like *functional precedence* and *information equivalence*. Then, they are captured in Event-B via invariant decomposition, where each basic property is associated individually with an invariant, as depicted in Listing 5.13.

```
INVARIANTS
// Functional precedence
∀fk1.∀fk2.fk1 ∈ Function ∧ fk2 ∈ Function ∧ beginFunc[{fk2}] = {TRUE} ⇒
endFunc[{fk1}] = {TRUE}
∀fk1.∀fk2.fk1 ∈ Function ∧ fk2 ∈ Function ∧ beginFunc[{fk1}] = {FALSE} ⇒
beginFunc[{fk2}] = {FALSE}
∀fk1.∀fk2.fk1 ∈ Function ∧ fk2 ∈ Function ∧ endFunc[{fk1}] = {TRUE} ∧
beginFunc[{fk2}] = {TRUE} ⇒ beginTimeFunc(fk1) + duration <
beginTimeFunc(fk2)
// Information equivalence
∀fk1.∀fk2.fk1 ∈ Function ∧ fk2 ∈ Function ∧ functionalOutput[{fk1}] =
functionalInput[{fk2}]
```

Listing 5.13: Functional integrity objective as Event-B invariant.

Interplay. The Event-B model also supports the specification of predicates corresponding to the properties conflict identification presented in Section 5.4.4 to analyze safety and security

interplay. We define these predicates as Event-B invariants to demonstrate that the resulting system model satisfies the desired safety and security properties in unison.

For instance, the Event-B invariant for the conflict identification predicate at the mission layer is depicted in Listing 5.14.

```
// Conflict identification at the mission layer
INVARIANTS
∀m.∃o1.∃o2.m ∈ Mission ∧ o1 ∈ Operation ∧ o2 ∈ Operation ∧
accomplishedThrough[{m}] = {o1, o2} ⇒ (overallGoal[{m}] = {TRUE} ∧
(¬postCondition[{o1}] = {TRUE} ∨ ¬postCondition[{o2}] = {TRUE}))
```

Listing 5.14: Mission layer properties conflict identification predicate as Event-B invariant.

These invariants are proven during the three-layered system model verification process.

5.5.4 Safety and Security Properties and Interplay Analysis

As mentioned in Section 5.5.1, a PO—Proof-Obligation is generated for every invariant that can be affected by an event. Figure 5.7 depicts an excerpt of the Event-B proving process [46]. In the CDV use case, the operational availability objective concerning the braking operation in the event of obstacle detection is analyzed by relying upon the following PO, among others:

Obstacle_Detection/BrakingAvailability/INV

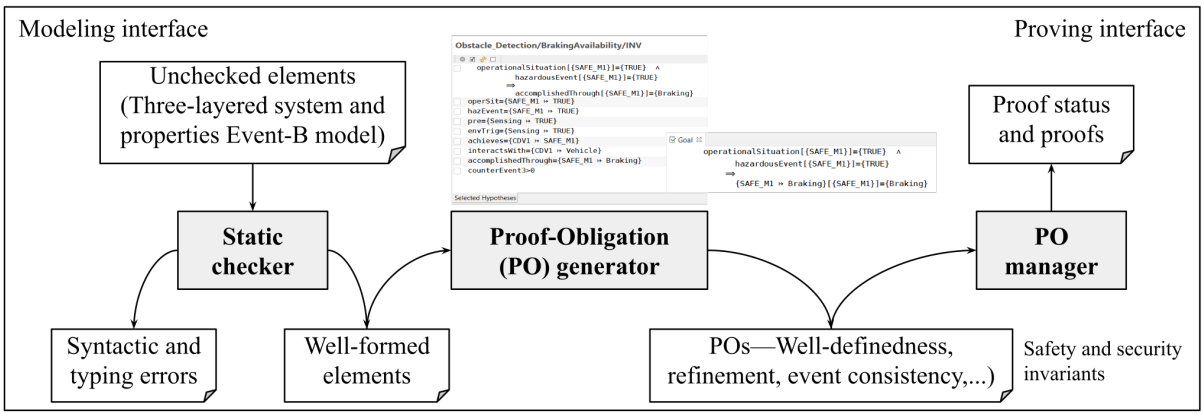


Figure 5.7: Event-B Proving Process.

The hypothesis for these POs relies upon the satisfaction of all the invariants (including behavioral, safety and security objectives, and gluing) and the validity of the guards restricting the values of the variables before the triggering of events in every reachable state of the system.

To illustrate the analysis, we present extraction of the PO for the satisfaction of the operational availability objective as an invariant, relying upon two invariants, INV1 and INV2, and a theorem THM, as follows:

1. **MissionAllocation (INV1):** According to this invariant, whenever there is an equivalence between the operational situation and hazardous event of an operation to the precondition and environment trigger condition of an operation, the mission is allocated to that operation for its accomplishment. Formally,

   $$\forall m \in \text{Mission}, \exists o \in \text{Operation} \mid ((\text{operationalSituation}[m] \wedge \text{hazardousEvent}[m]) \Leftrightarrow (\text{preCondition}[o] \wedge \text{environmentTriggerCondition}[o])) \Rightarrow \text{accomplishedThrough}[m] = o$$

   holds for all the events in the defined system model.

2. **MissionConsistency (INV2):** According to this invariant, given an operational situation, if the system does not encounter a hazardous event, the overall goal adhering to the input mission statement will not be satisfied. It signifies that there is no need to realize a safety-critical operation. Formally,

   $$\forall m \in \text{Mission} \mid (\text{operationalSituation}[m] \wedge \neg \text{hazardousEvent}[m]) \Rightarrow \neg \text{overallGoal}[m]$$

   holds for all the events in the defined system model.

3. **MissionAccomplishment (THM):** According to this theorem, on the realization of an operation, to which the safety-critical mission is allocated, the overall goal of the mission becomes equivalent to the post condition of the operation, denoting mission accomplishment. Formally,

   $$\forall m \in \text{Mission}, o \in \text{Operation} \mid (\text{accomplishedThrough}[m] = o) \Rightarrow (\text{postCondition}[o] \Leftrightarrow \text{overallGoal}[m]);$$

   which is a derived axiom that relies upon INV1 and INV2.

For the safety and security missions collision avoidance and grant only authorized actions, respectively, in the use case scenario, the PO corresponding to the invariant in Listing 5.14 is discharged, depicting a potential conflict between braking and access provisioning operations. The reason can be attributed to the lack of privilege to the operator to realize the braking operation manually, i.e., Operator $\mapsto$ CDV1 $\mapsto$ Braking $\notin$ *privilege*.

5.6 Building a Concrete Architecture

In order to formally model and verify the three-layered system and safety and security properties for a target application, we instantiate the proposed abstract Event-B models as follows:

- **System aspects via context and machine:** The model for a specific application would comprise a set of contexts and machines to capture the structural and behavioral aspects. The sets in the abstract model's context are populated with values captured as constants in the concrete model's context. Additionally, the axioms are defined to relate the constants with the parent sets. These aspects are supported by the extension mechanism (i.e., *extends* keyword) in Event-B. Likewise, the variables in the abstract model's machine are initialized with concrete values in the concrete model's machine. Herein, gluing invariants are defined to establish a link between the variables in both machines. Accordingly, the events in the concrete machine target the variable values, incorporating witnesses to relate with the variables in the abstract machine. This is done via using the refinement mechanism (i.e., *refines* keyword) in Event-B.

- **Safety and security properties and interplay via invariants:** The invariants corresponding to the safety and security properties and interplay in the abstract model are reused via instantiation (i.e., concrete values), based on the variables corresponding to the concrete model's machine.

- **Analysis via proofs:** To conduct safety and security properties verification and interplay analysis, we rely on the POs corresponding to the abstract system model. Accordingly, the abstract system model and the invariants can be reused to adapt to the domain concepts of the target application.

A concrete illustrative example is provided in Sections 6.2.5 and 6.3.1 in Chapter 6.

5.7 Tool Support for Formalization

In this section, we propose tool support relying upon formal-based techniques, supporting the formalisms presented in the previous Sections 5.4-5.5 to facilitate integrated analysis of safety and security properties. We begin by outlining the essential requirements that the underlying tools must fulfill to support the formalization aspects of the proposed approach, followed by the one used in our work.

5.7.1 Tool Support Requirements

The tool support for formalization aspects in our approach is designed to primarily support the model checking and properties verification tasks. As mentioned in Section 5.3, we rely upon formal constructs offered by the set theory, FOL, and Modal Logic in this regard. Accordingly, the underlying tools must fulfill the following essential requirements:

- Support the formal specification of the three-layered system model according to the DSMLs.
- Support the specification and automated verification of safety and security properties of the system as formal model libraries.
- Support the reuse of the resulting formal model libraries during the creation of the target system model.

Any tool that can fulfill these requirements can be used to conduct the properties analysis. In our case, we used Rodin [47] amongst the existing alternatives; the key features of which leveraged in this work are described in the following sub-section.

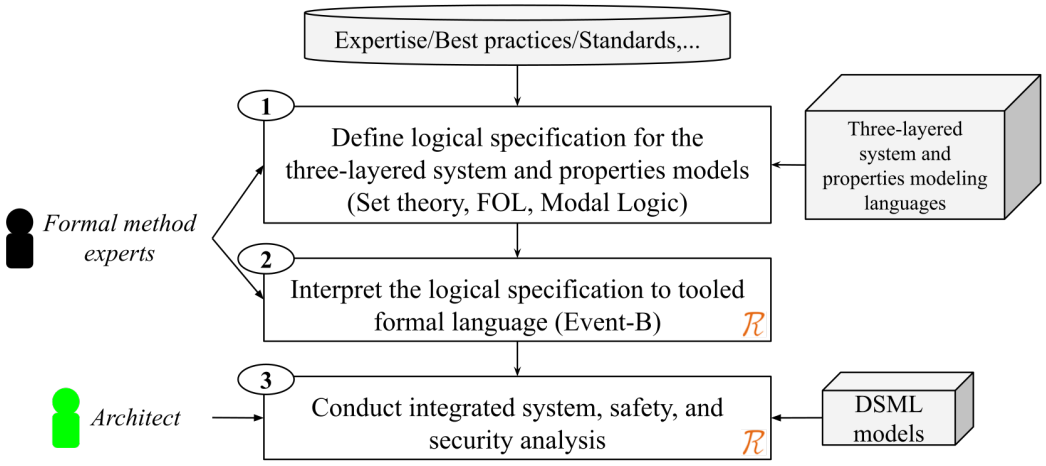


Figure 5.8: Tool Support for Analysis.

5.7.2 Formal Verification Tool: Rodin

Rodin is an Eclipse-based Integrated Development Environment (IDE) and open tool supporting the Event-B method. It facilitates mathematical proofs and refinement on discrete state transition-based system models [47]. The consistency between the different levels of refinement

is ensured using mathematical proofs. The modeling database in Rodin comprises support of various plug-ins, including requirement handlers, model checkers, proof obligation generator, automatic and interactive provers, and UML transformers [46]. In order to carry out safety and security analysis on the system model, we rely on theorem proving supported by Event-B via axioms and derivation rules. Indeed this process is iterative and compatible with the refinement process that allows model instantiation.

Specifically, the system and properties analysis aspects in the proposed approach incorporate the following steps concerning the tool support (see Figure 5.8): 1) Defining the logical specification for the three-layered system and properties models by the formal method experts, 2) Interpretation of the logical specification to a tooled-formal language by the formal method experts, and 3) Conducting integrated safety and security analysis by the system architect.

The first two steps (i.e., Steps 1 and 2) are performed once for a set of domains. The input of Step 1 is the expertise, standards, and best practices from the system, safety, and security domain, along with the modeling languages corresponding to the three-layered system and properties. Step 3 is performed once per application domain. Performing Steps 1 and 2 requires knowledge of the formal-based techniques (e.g., formal methods), whereas Step 3 requires knowledge of the system development process for a specific application domain. The input for this step is the DSML models resulting from the modeling as discussed in Chapter 4 (see Figure 4.14).

5.8 Conclusion

Summary. In this chapter, we addressed the *problematic* **P4** related to the integrated analysis of system safety and security properties during the system design phase. To this end, we focused on the formalization aspects of the three-layered system and properties DSMLs presented in the previous Chapter 4. Accordingly, the design is conducted at the formal language level, relying upon formal syntaxes, semantics, and rules for sound specification, reasoning, and verification of system and properties. Firstly, the logical specifications were defined for the three-layered system and the properties DSMLs, using basic notions from the set theory. This was followed by the formal logic-based specification of the safety and security properties and their interplay using FOL and Modal Logic. A set of basic properties was first incorporated concerning the three modeled layers to later define and specify the specific safety and security objectives. The resulting formalisms were then used to interpret the system and properties models into Event-B to obtain a more concrete specification and exploit the accompanying tool Rodin to conduct automated verification of safety and security objectives' signatures in unison.

Usefulness. It is evident from the review of the primary studies in Chapter 2 that there is still a lack of explicit modeling techniques that can encourage the formal reasoning of inter-dependencies between safety and security concerns in the early stages of the SE process. An amalgamation of the benefits of both worlds seems to be difficult, particularly due to the complexity associated with understanding and integrating formal techniques into the engineering process. Our approach addressed this concern by facilitating an integrated system safety and security analysis with respect to the typical stages of system development, viz. high-level missions, functionalities, and detailed technical and component-based architecture. This work provides a link between the semi-formal UML-based representations and formal constructs required to conduct proofs. The formalized safety and security objectives' signatures can be accommodated as reusable libraries in the development process of different safety- and security-critical systems at design time. This shall relieve the system designers and software architects of the complexity of dealing with the formal languages and proofs since these details are hidden by the modeling at the front end based upon DSMLs.

Concluding Remarks. Event-B has been used as formal language and technique as it was able to comply with the desired features, thus providing sound semantics and proof capabilities. The concept of refinement allowed the gradual introduction of the details in the system model pertaining to different modeling layers and specialize properties, and model instantiation for a target application that will be detailed in the forthcoming Chapter 6. Nevertheless, additional work is demanded to address the analysis of the system and properties in the case of the introduction of feared events (e.g., hazards, threats) via other complementary approaches like *model checking* and *animation*. The formalisms being generic enough, would serve for future extensions to other tooled-languages and formal frameworks via consistent interpretation.

Chapter 6

Assessment of the Contributions

Contents

6.1 Introduction

This chapter targets to illustrate our approach for safety and security co-engineering through a domain-specific application. Recalling the overall research problem stated in Chapter 1, the aim is to address the following question: *How to facilitate incorporation of safety and security concerns and their joint analysis in the design stages of the System Engineering (SE) process?* To this end, we apply and assess the contributions presented in previous Chapters 3-5 for the design and analysis of an autonomous Connected-Driving Vehicle (CDV) system from the transportation domain. The selection of the CDVs application is driven by the safety- and security-critical nature of these systems [6]. Their malfunctioning behavior concerning the design intent[1] is hazardous to the system, environmental assets, and humans involved [156, 157]. Moreover, their high-networked architecture offers exposure of the functionalities to potentially hostile players [158, 159]. Notwithstanding the extensive ongoing engineering effort in the field, further work is still needed regarding methodological guidance for integrated safety and security verification to increase CDVs design trustworthiness. Accordingly, we consider the design phase of system development conforming to International Organization for Standardization (ISO)

[1] Captures the design decisions made by the designer

26262 [30] and ISO 15288 [160] standards focusing on the integrated design and analysis of safety and security concerns in the automotive domain. Moreover, we consider the concept phase of system development as input to the overall approach and provide a preliminary definition of the System Under Design (SUD). Despite the complete approach proposed in Chapter 3 will be applied to treat the use case scenario in this chapter, the use case will also be used to illustrate the contents of the approach relying upon the fragments provided, in particular, in Chapters 4 and 5. Applying the approach to the use case scenario using the proposed tool-chain support (addressing the *problematic* **P5**) would enable us to determine to which extent the approach facilitates expressing, analyzing, and verifying the relationships between safety and security properties within a layer or across the layers in the three-layered system model.

Overall, the chapter is organized as follows: In Section 6.2, we introduce a realistic scenario inspired by the literature for illustration purposes of the approach concerning CDVs and untangle the associated safety and security concerns and their mutual impact. We also present the CDV system pertaining to high-level mission, functional, and detailed component-based specifications. Based upon these specifications as input, we present the modeling of the CDV system, followed by the formal-based joint analysis of safety and security concerns. This involves using the proposed tool-chain support prototype and describing the results obtained for each contribution. In Section 6.3, we discuss the approach's feasibility by assessing these results and highlight its potential for extensibility. Finally, we conclude by summing up the contribution of the chapter in Section 6.4.

6.2 Use Case: Connected-Driving Vehicles (CDVs)

In this section, we detail the CDV use case scenario from the system, safety, and security perspectives and demonstrate the application of the proposed approach.

6.2.1 Scenario Description: Converging Road Plan

We assume the applicability of CDVs in limited areas, such as shuttle buses in airport terminals, zoos, parks, and playgrounds [161] to simplify the illustration. Furthermore, we consider a realistic scenario of a converging road plan inspired by [162], as shown in Figure 6.1. In this scenario, two vehicles, denoted by CDV1 and CDV2, are both in driving mode and approaching each other on different but converging roads with non-negligible speeds. Another third vehicle, CDV3, is in the same lane and direction as CDV1 but behind it. To stay on the road lane, these vehicles follow virtual guides placed over the road.

We consider the motion-control module of the CDV system regarding this scenario,

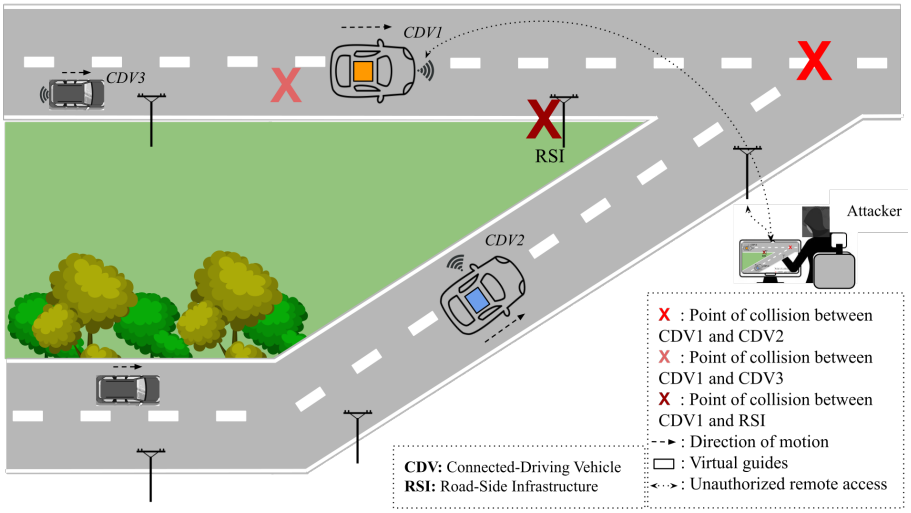


Figure 6.1: Use Case Scenario: Converging Road Plan.

following the description provided in Section 3.4.1. For simplicity, the aspects related to road guides' detection and communication at the system level are disentangled. We also assume that CDVs in this scenario are in full automation or self-driving mode, with no possibility of any operational takeover by the operator or passengers [163]. The operational scenario considered in this work is a choice relevant to both safety and security; however, we are aware of the possibility of several other cases that are nonetheless not in the present scope.

6.2.2 Safety and Security Concerns

Herein, we introduce the safety and security concerns and their interplay in the context of the scenario. We aim to analyze a subset of safety and security objectives[2] and their relationships, as depicted in Figure 6.2, from the catalogue presented in Chapter 4 across the three-layered model of the CDV system.

Safety Concerns. Since the future motion path of CDV1 and CDV2 may intersect at the junction, the unavailability of the braking operation while CDV1 is approaching the intersection may lead to a collision with CDV2. To avoid this, CDV1, as the target SUD, should decelerate upon identifying and locating CDV2. However, in this case, the faulty sensing components may fail to perceive the environment and can cause a severe impact on the safety of the entire vehicle [164]. This is captured under the safety column in Figure 6.2, wherein the availability of the system operations depends upon the availability of its underlying functions and components. In

[2]It is recalled that objectives herein represent the desired properties or features, capturing the positive vision of safety and security.

addition, the violation of the timing constraints for in-time delivery of status updates may hinder the vehicle's response to the presence of an obstacle [165]. For example, on locating CDV2 too close, the control signals from the multi-function control unit to the brake actuators should be sent in time to ensure the application of brakes by CDV1.

Security Concerns. The connected and autonomous technology typically relies upon the data or information made accessible to the components of the vehicular system or exchanged with other vehicles or remote entities for the system to be operational. This opens up opportunities to compromise the system's behavior from a security perspective. Any direct attack or unauthorized access to the system, e.g., CDV1, may cause the vehicle to misbehave. This can also cause progressive misbehavior of the surrounding vehicles, e.g., CDV2, as they exchange information with the vehicle under attack. The messages in transmission by the vehicle are subject to tampering or delays upon remotely intercepting the ongoing communication by an attacker [166]. Moreover, manipulation and replay attacks can influence the accurate delivery of signals, leading to the actuation sub-system receiving a tampered with or duplicate command [167]. This is captured under the security column in Figure 6.2, wherein the freshness of the information flows within a functional sub-system depends upon the freshness of the messages in transmission among the components.

Safety and Security Interplay Concerns. The presence of safety-related concerns may have an adverse effect on security and vice versa. For example, the reception of messages related to an emergency stop by CDV1 at the junction point can fall under the consequence of a replay attack and, in turn, influence the availability of brake-on-demand functionality, being a time-critical module. Thus, ensuring the freshness of the messages contributes to the in-time availability of safety-related functions (see Figure 6.2). The Light Detection and Ranging (LiDAR) component of CDV1, for instance, can be deceived into seeing a non-existing obstacle or phantom objects [168], tricking the vehicle into triggering abrupt braking, possibly leading to a collision with CDV3. In addition, by accessing the onboard control units remotely, an attacker can cause the vehicle to accelerate. Consequently, it is insufficient to evaluate the properties belonging to safety and security domains in segregation. During the design phase, it is essential to understand the interaction between these properties and keep them free from any inconsistencies based upon the underlying assumptions made within an operational context of the CDV system.

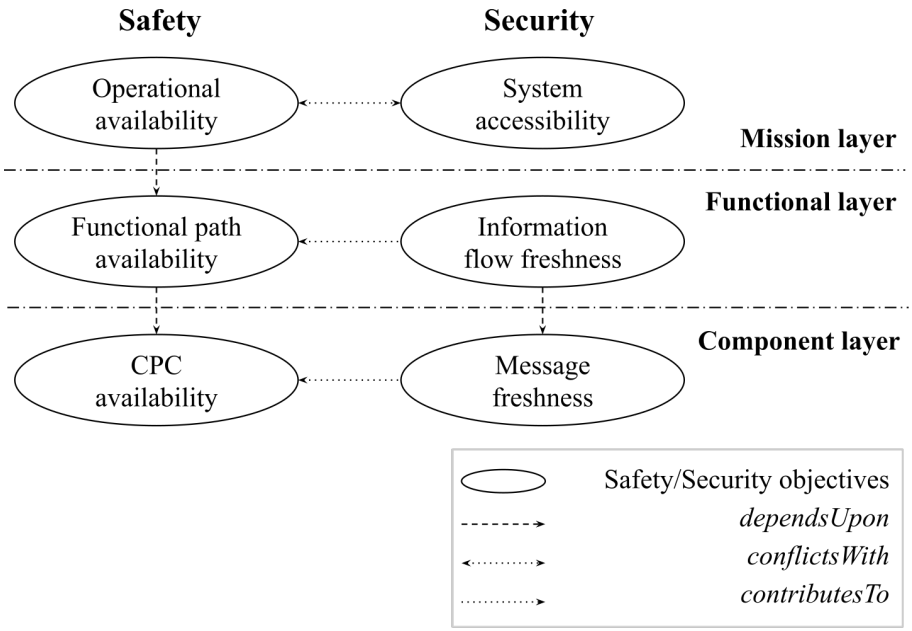


Figure 6.2: Safety and Security Objectives to be Verified.

6.2.3 CDV System Specification

In this sub-section, we present an initial specification of the CDV system concerning the different stages of system development, viz. mission, functional, and component. This is similar to the system development process targeting the concept phase as described in ISO 26262 [30] and ISO 15288 [160] standards and serves as an input to the proposed approach.

6.2.3.1 Mission Specification

Herein, we consider a CDV as an atomic system, with no focus on the lower-level functional or component architectural details. Regarding the converging road plan scenario depicted in Figure 6.1, we derive a set of operational, safety, and security missions, which the CDV system aims to achieve, in the following paragraphs.

Specification of Operational Missions. Initially, we begin with defining high-level operational missions of the CDV system that capture its underlying purpose. The specification of the operational missions follows the generic language syntax proposed in Section 4.4.1.1. In the converging road plan scenario, an example operational mission OPER_M is specified as follows:

[OPER_M] *A CDV should allow transportation of the passengers from one place to another.*

Identification of Safety Hazards and Security Threats. There can be several potentially hazardous and threat situations encompassing incompatible states of the CDV and its environment comprising other vehicles concerning OPER_M in the converging road plan scenario. These situations may be of varying complexity and can lead to severe safety-related accidents and security attacks, thereby causing harm to the environment, humans, or the CDV itself.

Safety Hazards. We follow a preliminary approach based on ISO 26262 Hazard Analysis and Risks Assessment (HARA) [169] to identify the safety hazards concerning OPER_M. The result of our analysis as an excerpt is summarized in Table 6.1. For each identified hazard (SH1-SH3), we describe the hazardous event and the consequence of the hazard in the form of an accident.

Table 6.1: Result of Preliminary Safety Hazard Analysis and Risk Assessment (HARA).

ID	Safety Hazard	Hazardous Event	Accident	ASIL
SH1	Unintended acceleration	Two CDVs approaching each other on converging roads with non-negligible speed	A CDV is side-collided, leading to rollover	D (E4, C3, S3)
SH2	Unintended stopping	A moving CDV stops in the middle of the road while another vehicle is coming from behind, with certain speed	Rear-end collision of the CDV with the following vehicle	C (E4, C3, S2)
SH3	No detection	A moving CDV is unable to detect the virtual guides on the road	CDV leaving the road lane, colliding with the surrounding infrastructure	A (E2, C3, S2)

Moreover, we determine the Automotive Safety Integrity Levels (ASIL) [170] for each hazard by evaluating the following risk impact factors:

- **Exposure:** It denotes the classes (E0-E4) of the probability of exposure concerning the operational situation, where E4 indicates the highest probability of exposure. We evaluate the exposure factor of SH1 and SH2 to E4 since CDV can frequently encounter operational scenarios involving acceleration and stopping during OPER_M. However, for SH3 involving deviation from the desired path, we evaluate the exposure factor to E2 based on the frequency of its occurrence.

- **Controllability:** It denotes the classes (C0-C3) of controllability of the hazardous event by an operator or other persons involved in the risky situation, where C3 indicates the lowest level of controllability. We evaluate the controllability factor concerning all the identified safety hazards to C3, considering the fully autonomous mode of the CDV with no possibility of a human taking over the control.

- **Severity:** It denotes the classes (S0-S3) of the severity of potential harm based on a defined rationale for each hazardous event, where S3 indicates the highest severity level. We

evaluate the severity factor concerning SH1 to S2 due to the vehicle rollover resulting from a side collision. Likewise, the severity factor for the other hazards SH2 and SH3 is S2 because of the potential harm caused by the rear-end collision and path deviation, respectively.

Security Threats. We follow a Preliminary Threat Assessment (PTA) approach to identify the security threats concerning OPER_M. The result of our analysis as an excerpt is summarized in Table 6.2. For each identified threat (ST1-ST3), we describe the security incident and the consequence of the threat in the form of an attack. Moreover, we type each identified threat based on the STRIDE [5] threat model.

Table 6.2: Result of Preliminary Threat Assessment (PTA).

ID	Security Threat	Security Incident	Attack	STRIDE Category
ST1	Unauthorized operations	Hostile party performing unauthorized actions within on-board CDV architecture	Introduction of malware in CDV's computing facilities	Elevation of privilege
ST2	Physical proximity	A CDV is in physical proximity of a person who is not an authorized user	Physical damage to the CDV	Denial of Service (DoS)
ST3	Remote control	A person controlling the CDV remotely	Misleading navigation of the CDV	Tampering

The hazards and threats identified above are considered as input for specifying the safety and security missions of the system, as detailed in the following paragraph.

Specification of Safety and Security Missions. To limit the risks associated with the aforementioned safety hazards and security threats, we respectively derive safety and security missions from them. We elicit one safety-related mission per hazard mentioned in Table 6.1 and one security-related mission per threat mentioned in Table 6.2. The list though not exhaustive, captures the essence of the potentially risky scenarios.

Safety Missions. The safety mission SAFE_M1 specific to the safety hazard SH1—for instance, is specified as follows:

[SAFE_M1] *A moving CDV should stop whenever an obstacle is detected.*

Herein, we assume the operational behavior of the CDV system as a binary—either moving or stopped—due to the lack of obstacle's distance-related details for the vehicle to allow deceleration without causing the engine to stop [**NB:** These details are not necessary and might not be available at this stage].

Security Missions. The security mission SEC_M1 specific to the security threat ST1—for instance, is specified as follows:

[SEC_M1] *An operational CDV should only allow authorized actions being performed by its operator.*

6.2.3.2 Functional Specification

Functions are the primary elements while modeling the functional architecture of the system. Recalling the scenario depicted in Figure 6.1 in the context of the functional layer, we consider CDV system comprising different sub-systems to execute its intended functionality. Specifically, we focus on the motion-related part of the functional representation of CDVs. This roughly includes 1) perception-related functions concerning the external environment in which the CDV operates, 2) decision-making functions based on the perceived external environment, and 3) actuation-related functions based upon the control decisions. In this case, a typical functional path can be:

[FP] *obstacle detection* → *position inference* → *deceleration*

The above functions respectively support perception, processing, and motion control. The designer proposes these functions constituting the functional path to perform the *sensing*, *decision-making*, and *braking* operations, which should accomplish the system missions, e.g., SAFE_M1, modeled at the mission layer.

Accordingly, from a safety perspective, the key requirement is to ensure that the safety-critical functions are always available. Specifically, regarding FP:

[SAFE_FP1] *Whenever a collision situation is detected, an in-time availability of the braking function response must be ensured.*

Likewise, from a security perspective, to prevent any undesirable triggering of functions, the following security requirement must be ensured:

[SEC_FP1] *The freshness of the information generated by the sequence of functions constituting the functional path FP must be ensured to prevent any delayed triggering of functions.*

6.2.3.3 Component Specification

The functional blocks mentioned in the previous sub-section, in turn, can be executed by cross-application generalized components, like cameras, and application-specific components, like Global Positioning System (GPS) and sensors that are responsible for collecting the data, e.g., obstacle-related, as input from the surroundings and sending the same to the processing unit for producing information relevant for decision-making (refer to Figure 3.6). This may also involve the use of data fusion algorithms to integrate data from multiple sensors. The controlling software uses this information to generate and issue signals via the local gateway to other vehicle components, e.g., brake actuators.

From the safety perspective, any of these components, in the case of the introduction of faults, may hinder the transmission of data or control signals for effective braking in a collision situation. Hence, some of the essential safety requirements regarding availability are:

- **[SAFE_C1]** *The availability of sensors must be ensured to periodically detect the presence of obstacles in the CDV's surroundings.*
- **[SAFE_C2]** *The availability of the processing unit must be ensured whenever the sensors transmit obstacle-related data to infer its position.*
- **[SAFE_C3]** *The availability of brake actuators must be ensured for deceleration in case of an obstacle too close to the CDV.*
- **[SAFE_C4]** *Whenever obstacle-related data or control signals are being transmitted between the sensors, processing unit, and brake actuators, the availability of connectors, e.g., data buses and communication channels, between these components must be ensured.*

Likewise, in the case of security, to prevent any delays introduced to the messages in transmission, the following security requirement regarding freshness must be ensured:

- **[SEC_C1]** *The freshness of the messages comprising any environment-related data collected by the sensors must be ensured.*
- **[SEC_C2]** *The freshness of the control signal sent by the controlling software to the brake actuator must be ensured.*

6.2.4 Modeling the CDV System Design

In this section, based upon the initial system specification provided previously as input, we present the modeling of the CDV system regarding different stages of system development,

viz. mission, functional, and component, using our proposed framework and tool support. The models incorporate the desired safety and security properties as extensions, using the Domain-Specific Modeling Language (DSML) profiles presented in Chapter 4. These models provide the candidates for the system architecture that are further evaluated regarding the formalized safety and security properties specifications in Section 6.2.5. The idea is to provide a tooled-method for facilitating the system, safety, and security *modeling* at the design phase as described in ISO 26262 [30] and ISO 15288 [160] standards.

System and Properties Modeling at the Mission Layer. To model the CDV system at the mission layer, we consider the initial mission specifications in Section 6.2.3.1 as input to instantiate the mission layer DSML profile presented in Section 4.4.3.1. Figures 6.3 and 6.4, respectively depict an excerpt of the mission model for safety mission SAFE_M1 and security mission SEC_M1. Herein, each element is represented as a Unified Modeling Language (UML) class. The interactions between the elements are captured by the class attributes. For example, the association *achieves* between the system CDV1 and the mission SAFE_M1 appears as an attribute in the CDV1 class. The *sensing*, *decision making*, and *braking* operations respectively influence the obstacle detection, position inference, and speed attribute variables of CDV1 [**NB:** The operator herein is responsible for monitoring and maintaining the system that controls the CDV and reporting any issues to the backend team].

The safety and security objectives to be satisfied are incorporated by instantiating the profile elements from Section 4.5.3 concerning the system CDV1. This integration is achieved in an extended version of the Papyrus modeling environment to ensure: 1) SAFE_M1 via the availability of the safety-related operations, viz. sensing, decision making, and braking, and 2) SEC_M1 via the controlled accessibility of the system operations by its operator. The additional details regarding these objectives are provided in pragmas as customized text-based comments.

System and Properties Modeling at the Functional Layer. To model the CDV system functionality regarding the use case scenario, we consider the functional path FP in the initial functional specification in Section 6.2.3.2 to instantiate the functional layer DSML profile presented in Section 4.4.3.2. Each function is represented as an opaque action, as depicted in Figure 6.5, i.e., the details of the function are, at this stage, unknown, and hence, it is considered as a black box. An object flow captures the flow of information between each pair of functions. Functional input and output are mapped to input and output pins, respectively. The *obstacle detection* function calls itself periodically to realize the sensing operation. Likewise, the *position inference* function obtains the information related to the obstacle and determines whether or not the CDV is at a safe distance from the obstacle. After inferring the position, it sends the related

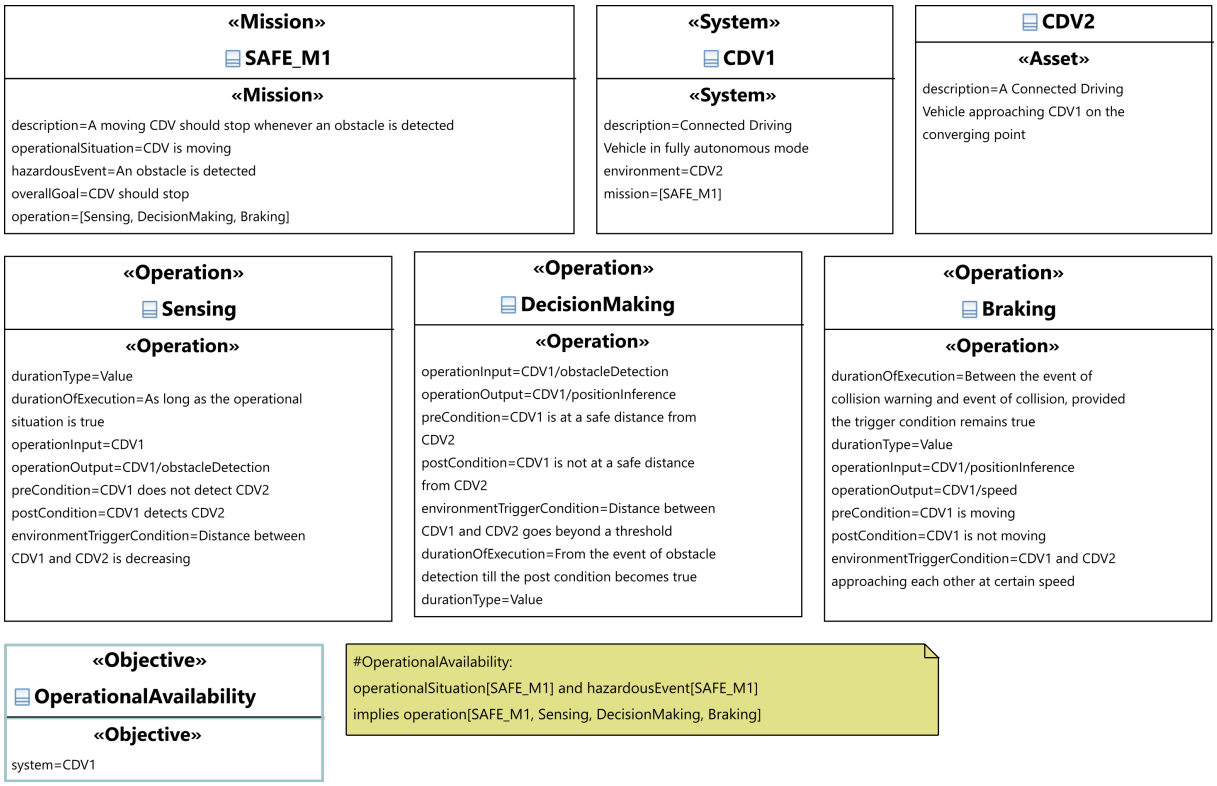


Figure 6.3: Mission-layer System UML Profile Instantiation for Safety Mission SAFE_M1.

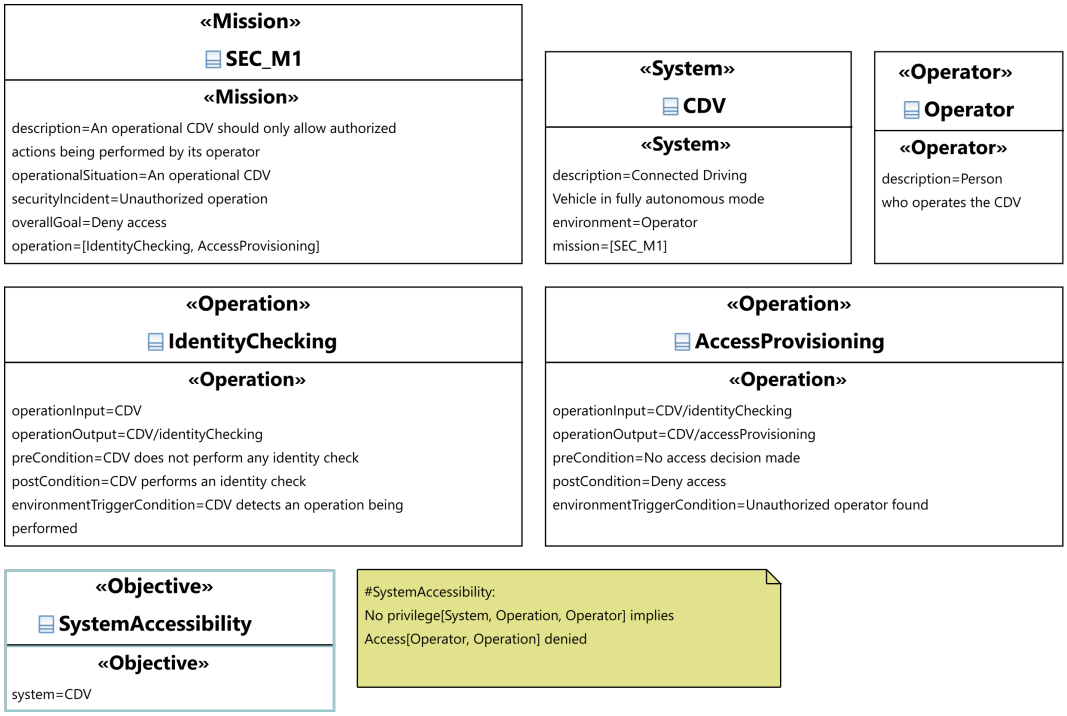


Figure 6.4: Mission-layer System UML Profile Instantiation for Security Mission SEC_M1.

information to the *deceleration* function that manipulates the speed of the CDV, in turn, related to braking.

The safety and security objectives to be satisfied are incorporated by considering the profile elements from Section 4.5.3 in the form of pragmas as customized text-based comments. In this case, to ensure the in-time braking response (SAFE_FP1) in the safety context, an order must exist in executing these functions and preserving the information flow across them. In addition, the execution of the functions must respect the period during which their availability is required. Thus, the functional path availability objective must be verified. Likewise, in the security context, the integration aims at ensuring SEC_FP1 via freshness of the information flows between consecutive functions across the functional path FP. This would also contribute to timeliness regarding the execution of the functions to ensure safe braking.

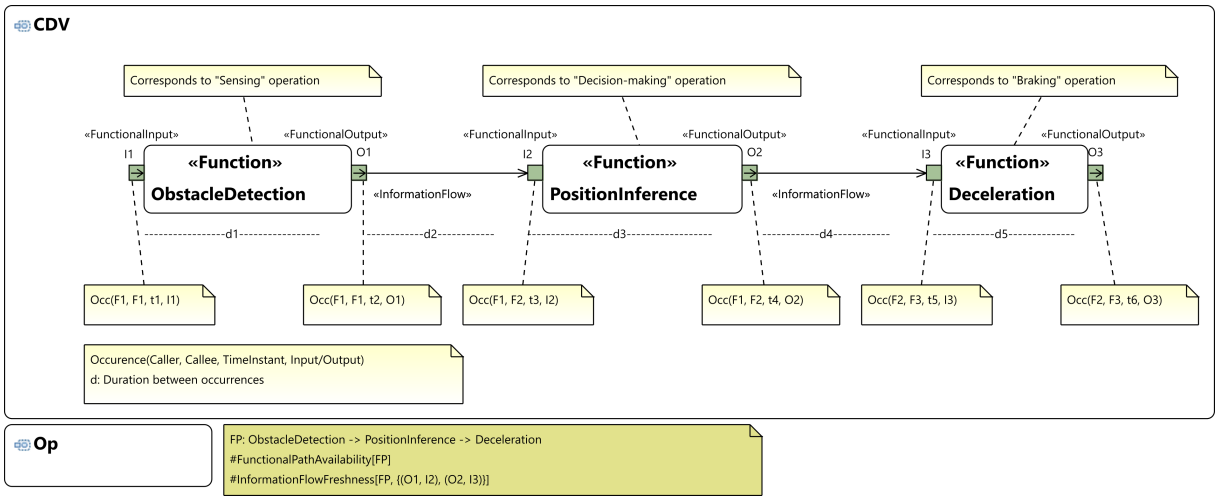


Figure 6.5: Functional-layer System UML Profile Instantiation for Functional Path FP.

System and Properties Modeling at the Component Layer. To model the CDV system component-based architecture, we consider the set of components in the initial component specification in Section 6.2.3.3 to instantiate the component layer DSML profile presented in Section 4.4.3.3. An excerpt of the component diagram for the CDV use case scenario is depicted in Figure 6.6. It comprises three system components, viz. *sensor*, *processing unit*, and *brake actuator*. Semantically, the functions considered in the functional path FP are respectively allocated to these components in a one-to-one corresponding. The sensor communicates the obstacle-related reading Read_Obstacle with the processing unit via the S_PU connector. Subsequently, relying upon this reading, the processing unit transmits the control decision Sig_Brake to the brake actuator via the PU_BA connector. In addition, the respective ports are added that provide an interface for exchanging messages between these components.

The safety and security objectives to be satisfied are incorporated by considering the profile elements from Section 4.5.3 in the form of pragmas as customized text-based comments. This integration aims to ensure: 1) the requirements SAFE_C1-SAFE_C4 via the verification of the CPC availability objective, 2) SEC_C1 and SEC_C2 via the verification of the message freshness objective.

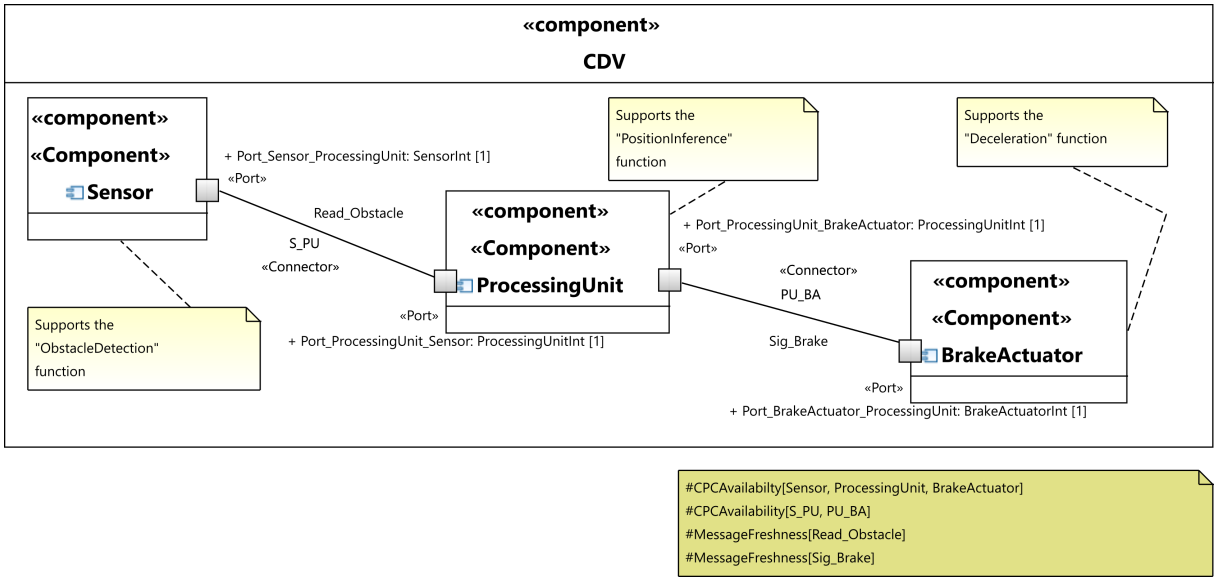


Figure 6.6: Component-layer System UML Profile Instantiation for the Use Case Scenario.

6.2.5 Formal-based Joint Analysis of CDV Safety and Security

Since the Event-B interpretation of the three-layered system and properties model presented in Section 5.6 is generic, it is instantiated to incorporate the specific details of a concrete system according to the application domain. This facilitates the integration of the properties in the system development process of the target application using pre-defined library signatures. The approach also allows for analyzing system safety and security properties in both a standalone and joint fashion. The analysis involves the safety and security objectives—in **Bold**, using the basic properties (if any)—in *Italics*, for defining these objectives, as depicted in Figure 6.7. The figure is an excerpt to illustrate the dependencies among the properties across the three-layered model. It is noteworthy that these dependencies may not hold in all the cases and are considered specifically in our context for illustration purposes. Accordingly, we consider the system architecture models presented in the previous sub-section 6.2.4 for formal-based analysis of safety and security in an integrated manner, as detailed in the following paragraphs. The idea is to provide support for facilitating the system safety and security *analysis* at the design phase

as described in ISO 26262 [30] and ISO 15288 [160] standards.

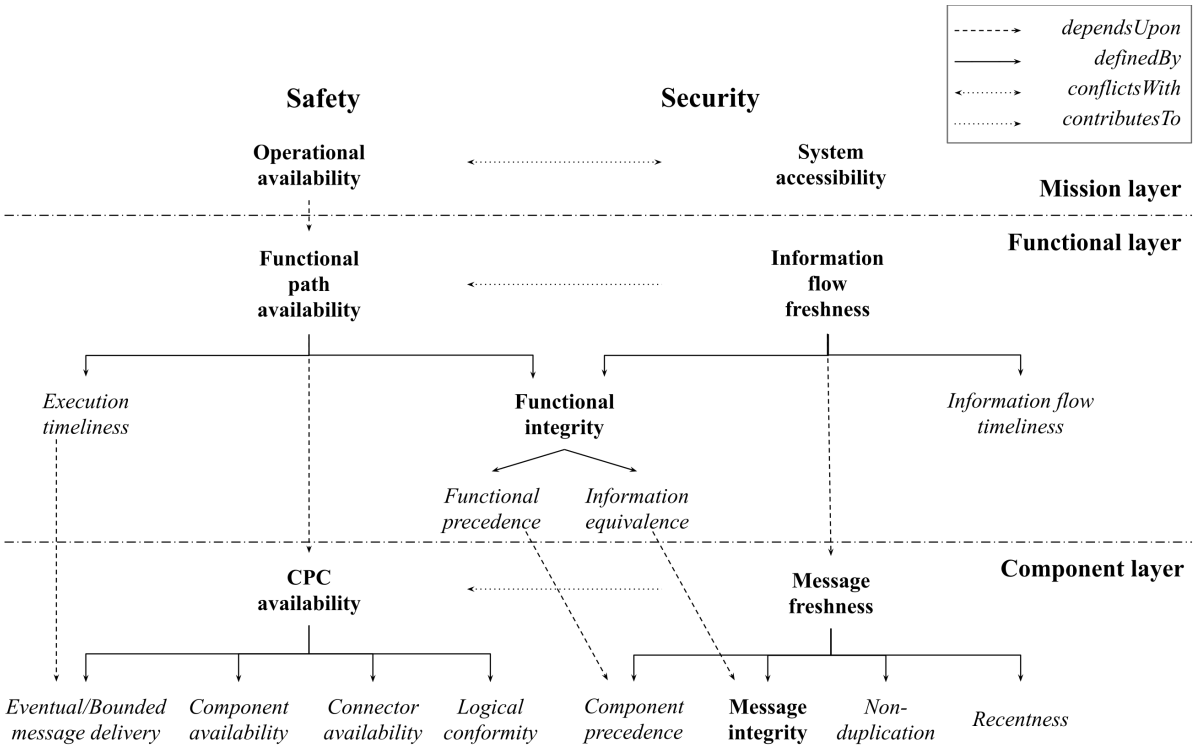


Figure 6.7: Unfolding the Safety and Security Objectives using Basic Properties.

System and Properties Analysis at the Mission Layer. In the light of the use case, the mission-layer system and property views presented in Section 5.6 are instantiated as 2 Event-B contexts, viz. `C3MissionViewInstance` and `C4MissionPropertyViewInstance`, respectively, using the *extends* keyword. An excerpt of the context `C3MissionViewInstance` is depicted in Listing 6.1. Herein, based on the mission models depicted in Figures 6.3 and 6.4, the CDV system is interpreted as a set of constants, e.g., `CDV1`, `SAFE_M1`, `Braking`, along with axioms to define their relationships with the carrier sets `System`, `Mission`, `Operation`, of the context `C0MissionView`.

```
// Structural aspects of CDV system: Mission layer
CONTEXT C3MissionViewInstance
EXTENDS C0MissionView
CONSTANTS
    CDV1, SAFE_M1, SEC_M1, Person, Vehicle, Braking, AccessProvisioning
AXIOMS
    CDV1 ∈ System
    SAFE_M1 ∈ Mission ∧ SEC_M1 ∈ Mission ∧ Mission = {SAFE_M1, SEC_M1}
```

```
    Person ∈ People
    Vehicle ∈ Asset
    Braking ∈ Operation ∧ AccessProvisioning ∈ Operation ∧ Operation =
    {Braking, AccessProvisioning}
END
```

Listing 6.1: Instantiation excerpt of C0MissionView context at the mission layer capturing CDV structural aspects.

A concrete machine `M1MissionViewInstance` refines the abstract machine `M0MissionView` in the form of an instantiation specific to the use case scenario. In this machine, variables, e.g., `operSit`, `goal`, `pre`, `post`, and `envTrig`, define the state corresponding to the CDV system in operation [**NB:** The variables are not explicitly listed herein and follow the declarations from Listing 5.3]. We specify gluing invariants[3], e.g., in Listing 6.2, for maintaining the consistency between the variables belonging to the abstract system model in Section 5.5.2.1 and concrete CDV system model.

```
INVARIANTS
    (operSit = {SAFE_M1 ↦ FALSE} ⇔ operationalSituation = {m1 ↦ FALSE}) ∧
    (operSit = {SAFE_M1 ↦ TRUE} ⇔ operationalSituation = {m1 ↦ TRUE})
```

Listing 6.2: An example of gluing invariant to link abstract and concrete CDV system at the mission layer.

In addition, we recall the variables corresponding to the relationships between the mission view elements defined in the abstract machine `M0MissionView`. The CDV system state is defined in the collision scenario, comprising CDV in motion, obstacle detection, and assurance of braking. We do not add new events and reuse the existing ones via *refinement* considering the concrete machine's variables. More specifically, events like `Moving_CDV`, `Obstacle_Detection`, and `Ensure_Braking` of the concrete machine refine the events `Event1`, `Event2`, and `Event3` of the abstract machine, respectively, following the syntax in Listing 5.3.

The safety and security objectives are instantiated as invariants in the use case context, as shown in Listing 6.3.

```
INVARIANTS
// Braking availability
(operationalSituation[{SAFE_M1}] = {TRUE} ∧ hazardousEvent[{SAFE_M1}]
= {TRUE}) ⇒ accomplishedThrough[{SAFE_M1}] = {Braking}
// CDV accessibility
Person ↦ CDV1 ↦ Braking ∉ privilege ⇒ access[{Person ↦ Braking}] = {FALSE}
```

[3] A gluing invariant glues the space of the abstract and concrete machines [136].

Listing 6.3: CDV safety and security objectives at the mission layer.

The Proof-Obligations (POs) corresponding to the guard strengthening and validation of the gluing invariants help to prove the correctness of the instantiated model, along with the verification of the safety (SAFE_M1) and security (SEC_M1) objectives in segregation. However, when modeled together, the PO corresponding to the invariant in Listing 6.4 is discharged, depicting a potential conflict between braking availability and CDV accessibility. This is due to the lack of privilege for the operator to realize the braking operation on CDV1 manually, i.e., (Operator $\mapsto$ CDV1 $\mapsto$ Braking) $\notin$ *privilege*.

```
INVARIANTS
(accomplishedThrough[{SAFE_M1}] = {Braking} ∧ accomplishedThrough[{SEC_M1}]
= {AccessProvisioning}) ⇒ (overallGoal[{SAFE_M1}] = {TRUE} ∧
(¬postCondition[{Braking}] = {TRUE} ∨ ¬postCondition[{AccessProvisioning}]
= {TRUE}))
```

Listing 6.4: Mission layer conflict identification predicate as Event-B invariant in the CDV scenario.

System and Properties Analysis at the Functional Layer. The instantiation of the functional-layer system and property views presented in Section 5.6 concerning the use case involves 2 Event-B contexts, viz. `C8FunctionViewInstance` and `C9FunctionPropertyViewInstance`. An excerpt of the former is depicted in Listing 6.5, based on the functional model depicted in Figure 6.5.

```
// Structural aspects of CDV system: Functional layer
CONTEXT C8FunctionViewInstance
EXTENDS C5FunctionView
CONSTANTS
    ObstacleDetection, PositionInference, Deceleration
AXIOMS
    // Functions
    ObstacleDetection ∈ Function ∧ PositionInference ∈ Function ∧
    Deceleration ∈ Function ∧ Function = {ObstacleDetection,
    PositionInference, Deceleration}
END
```

Listing 6.5: Instantiation excerpt of C5FunctionView context at the functional layer capturing CDV functions.

Likewise, the machines, viz. M4FunctionViewInstance and M4FunctionViewRefinementInstance model the behavioral aspects of the CDV system. An excerpt of the latter specific to the use case scenario is depicted in Listing 6.6. In this machine, variables like funcInput and funcOutput, capture the state conditions corresponding to the system functions. Furthermore, events like OBSTACLEDETECTION, POSITIONINFERENCE, and DECELERATION of the concrete machine refine the events FUNCTION1, FUNCTION2, and FUNCTION2, respectively, of the abstract machine.

```
// Behavioral aspects of CDV system: Functional layer
MACHINE M5FunctionViewRefinementInstance
REFINES M3FunctionViewRefinement
SEES C8FunctionViewInstance, C9FunctionPropertyViewInstance
VARIABLES bFunc, eFunc, bTimeFunc, eTimeFunc, dFunc, d, t, funcInput,
funcOutput, v
EVENTS
    DECELERATION_BEGIN ≙
    STATUS
        ordinary
    REFINES
        FUNC3_BEGIN
    ANY
        tick
    WHERE
        bFunc = {Deceleration ↦ FALSE}
        eFunc = {PositionInference ↦ TRUE}
        tick ∈ ℕ1
        d = d + tick
        t ≥ eTimeFunc(f1) + d
    THEN
        bFunc ≔ {Deceleration ↦ TRUE}
        bTimeFunc(Deceleration) ≔ t
        funcInput ≔ {Deceleration ↦ v}
    END
END
```

Listing 6.6: Instantiation excerpt of Event-B machine at the functional layer: Event capturing beginning of deceleration.

The safety and security objectives are instantiated as invariants in the use case context. For instance, for the pair of functions Deceleration and PositionInference, both *Functional path availability* and *Information flow freshness* objectives are defined by instantiating the *Functional integrity* objective defined in Listing 5.13, along with *Execution Timeliness* and

Information Flow Timeliness properties. The functional integrity is analyzed via the conjunction of invariants corresponding to functional precedence and information equivalence, as shown in Listing 6.7.

```
INVARIANTS
// Functional precedence
beginFunc[{Deceleration}] = {TRUE} ⇒ endFunc[{PositionInference}] = {TRUE}
beginFunc[{PositionInference}] = {FALSE} ⇒ beginFunc[{Deceleration}] = {FALSE}
endFunc[{PositionInference}] = {TRUE} ∧ beginFunc[{Deceleration}] = {TRUE} ⇒
beginTimeFunc(PositionInference) + duration < beginTimeFunc(Deceleration)
// Information equivalence
functionalOutput[{PositionInference}] = functionalInput[{Deceleration}]
```

Listing 6.7: Instantiation of functional integrity objective at the functional layer.

Herein, generation of the output by the function `PositionInference` upon detection of the obstacle (i.e., CDV2) by `ObstacleDetection`, and `Deceleration` upon determining the position of CDV2 by `PositionInference`, before their respective deadlines captured by `dFunc`, ensures the functional path availability objective (SAFE_FP1). Essentially, `dFunc` for the function `Deceleration` lies within the interval comprising the event of collision warning and the event of the actual collision. In addition, the use of POs corresponding to the passage of information between these consecutive functions fulfills the information flow timeliness objective, resulting in the fulfillment of the information flow freshness objective (SEC_FP1). However, this holds as long as the interval comprising *tick* of the ending and beginning of two consecutive functions across the functional path respects the duration d.

System and Properties Analysis at the Component Layer. The instantiation of the component-layer system and property views presented in Section 5.6 concerning the use case involves 2 Event-B contexts, viz. `C14ComponentViewInstance` and `C15ComponentPropertyViewInstance`. An excerpt of the former is shown in Listing 6.8, based on the component model depicted in Figure 6.6.

```
// Structural aspects of CDV system: Component layer
CONTEXT C14ComponentViewInstance
EXTENDS C10ComponentView
CONSTANTS
    Sensor, ProcessingUnit, BrakeActuator, S_Out, PU_In, PU_Out, BA_In,
    S_PU, PU_BA
AXIOMS
    // Components
    Sensor ∈ Component ∧ ProcessingUnit ∈ Component ∧ BrakeActuator ∈
    Component ∧ Component = {Sensor, ProcessingUnit, BrakeActuator}
```

```
    // Ports
    S_Out ∈ Port ∧ PU_In ∈ Port ∧ PU_Out ∈ Port ∧ BA_In ∈ Port ∧
    Port = {S_Out, PU_In, PU_Out, BA_In}
    // Connectors
    S_PU ∈ Connector ∧ PU_BA ∈ Connector ∧ Connector = {S_PU, PU_BA}
END
```

Listing 6.8: Instantiation excerpt of C10ComponentView context at the component layer capturing CDV components-ports-connectors.

Likewise, the machine `M9ComponentMPSTickInstance` models the behavioral aspects of the CDV system. An excerpt of this machine specific to the event of transmission of a braking command from the processing unit to the brake actuator in the use case scenario is depicted in Listing 6.9.

```
// Behavioral aspects of CDV system: Component layer
MACHINE M9ComponentMPSTickInstance
REFINES M8ComponentMPSTick
SEES C13ComponentMessaging, C14ComponentViewInstance,
C15ComponentPropertyViewInstance
VARIABLES available, sentMsg, visible, receivedMsg, timeProgress, sendTick
visibleTick, receiveTick
INVARIANTS
    // Message integrity
    ∀m.∀mps.∀c1.∀c2.m ∈ Msg ∧ mps ∈ MsgMPS ∧ sender[{mps}] = {c1} ∧
    sent[{m}] = {sendTick} ⇒ receiver[{mps}] = {c2} ∧ received[{m}] =
    {receiveTick}
    // Non-duplication
    ∀c1.∀m.∀mps.∀t1.∀t2.m ∈ Msg ∧ mps ∈ MsgMPS ∧ receiver[{mps}] = {c1}
    ∧ received[{m}] = {t1} ∧ t2 < t1 ⇒ ¬received[{m}] = {t2}
EVENTS
    BrakeCommand_Transmission ≙
    STATUS
        ordinary
    REFINES
        TRANSMISSIONMPS
    ANY
        Brk_CMD
        ProcessingUnit
        BrakeActuator
    WHERE
        Brk_CMD ∈ sentMsg
        ProcessingUnit ∈ from[{Brk_CMD}] ∧ BrakeActuator ∈ to[{Brk_CMD}]
        visibleTick ∈ timeProgress
```

```
    WITH
        Brk_CMD = m
        ProcessingUnit = c1
        BrakeActuator = c2
    THEN
        visible := visible ∪ {Brk_CMD}
        sentMsg := sentMsg \ {Brk_CMD}
        timeProgress := timeProgress \ {visibleTick}
    END
END
```

Listing 6.9: Instantiation excerpt of M8ComponentMPSTick machine at the component layer: Event capturing braking command transmission.

The CPC availability objective corresponding to SAFE_C1-SAFE_C4 is proven by verifying the availability of the components and connectors in the given duration, along with the delivery of the obstacle-related data `Obt_DATA` and braking command `Brk_CMD` to the processing unit and brake actuator, respectively, in logical conformance to the position inference and deceleration functions. Likewise, the message freshness objective to ensure SEC_C1 and SEC_C2 is proven by verifying that the messages in transmission fulfill the message integrity, recentness in message delivery, and non-duplication of messages for each pair of components, i.e., (`Sensor`, `ProcessingUnit`) and (`ProcessingUnit`, `BrakeActuator`), interacting with each other. In case bounded delivery of messages is required, fulfilling the recentness objective contributes to the CPC availability objective. Since these verifications are conducted at the component layer, it is recalled that the model is supposed to contain enough details of the architecture, including time.

6.3 Assessment of the Approach

In this section, we provide a light-weight evaluation of the proposed approach regarding its applicability to the CDV use case, along with a discussion on its key features and the potential for its generalization.

6.3.1 Evaluating the Approach on the Use Case

The outcomes of applying the approach to the CDV use case correspond to its two main phases: 1) system and properties *modeling* and 2) properties *verification*.

6.3.1.1 Modeling Distinctive Features

The modeling of the CDV architecture at the three layers involved a set of profiles with their respective stereotypes corresponding to the fundamental notions presented in Section 4.4. The number of stereotypes is summarized in Table 6.3 that contribute to apprehending and comprehending the elements involved in this work and, in turn, the understandability of the models.

Table 6.3: Number of Stereotypes in the Three-layered System Profile for CDV Use Case.

Profile	# Stereotypes
Mission-layer system profile	7
Functional-layer system profile	11
Component-layer system profile	11

Herein, functional- and component-layer profiles inherited 2 (viz., **System**, **Operator**) and 1 (viz., **System**) stereotypes, respectively, from the mission-layer profile. Likewise, the number of safety and security objectives covered in the properties catalogue for modeling the system properties and their interplay at the three layers are summarized in Table 6.4. The number of relationships addressed across layers includes 1 between mission and functional layers, and 4 between functional and component layers.

Table 6.4: Coverage of Safety and Security Objectives and their Interplay for CDV Use Case.

Perspective	# Objectives		
	Mission Layer	Functional Layer	Component Layer
Safety	1	2	2
Security	1	2	1
Interplay	1	3	1

It is recalled that safety and security objectives were associated with specific stereotypes corresponding to the three-layered system modeling. For example, the *Braking Availability* objective at the mission layer applies to *Mission* and *Operation*. Likewise, the *CDV Accessibility* objectives applies to *People*, *System*, and *Operation*. Table 6.5 summarizes the elements to which the properties in Figure 6.7 are associated.

6.3.1.2 Formalization and Verification

The formalization of the CDV system architecture and properties at the three layers involved defining 6 contexts—2 for each layer—to capture the structural aspects and properties of the system. Likewise, the system behavior is captured via machines; 1 at the mission layer, 2 at the

Table 6.5: Safety and Security Objectives and Basic Properties with associated Stereotypes in the Three-layered System Model.

Layer	Objectives/Properties	Elements
Mission	Operational Availability	Mission, Operation
	System Accessibility	System, Operation, People
Functional	Functional Path Availability	Functional Path
	Information Flow Freshness	Functional Path, Functional Input, Functional Output
	Functional Integrity	Functional Path
	Execution Timeliness	Functional Path
	Functional Precedence	Function
	Information Equivalence	Functional Input, Functional Output
	Information Flow Timeliness	Functional Path, Functional Input, Functional Output
Component	CPC Availability	Component, Connector, Message
	Message Freshness	Message
	Eventual Message Delivery	Component, Message
	Bounded Message Delivery	Component, Message
	Component Availability	Component
	Connector Availability	Connector
	Logical Conformity	Component, Port
	Component Precedence	Component
	Message Integrity	Component, Message
	Non-duplication	Component, Message
	Recentness	Component, Message

functional layer, and 1 at the component layer, to capture the state transitions targeting CDV operations, functions, and component-based message transmission. The Event-B specification of the CDV system model adheres to the interpretation rules defined in Chapter 5. Afterwards, the results were obtained at the front end regarding proof of safety and security objectives and their interplay. An excerpt of how hypotheses are selected to conduct interactive proving in Rodin is depicted in Figure 6.8. The selected hypotheses are used to prove that the goal depicted in Figure 6.9—for instance, corresponding to an invariant is satisfied.

Being correct-by-construction, some of the POs corresponding to the safety and security objectives and their interplay for the instantiated CDV model were automatically discharged. These POs comprise the hypotheses as a set of predicates. A summary of the number of POs for objectives at each layer is provided in Table 6.6. In the failure of proof attempts, the following 2 cases are possible: 1) the abstract specification in Section 5.6 was fixed regarding the design assumptions for the well-definedness of the predicates (see Figure 6.10) and 2) the set of selected hypothesis that contribute to the sequent rules can be modified. Herein, the end-user, e.g., the

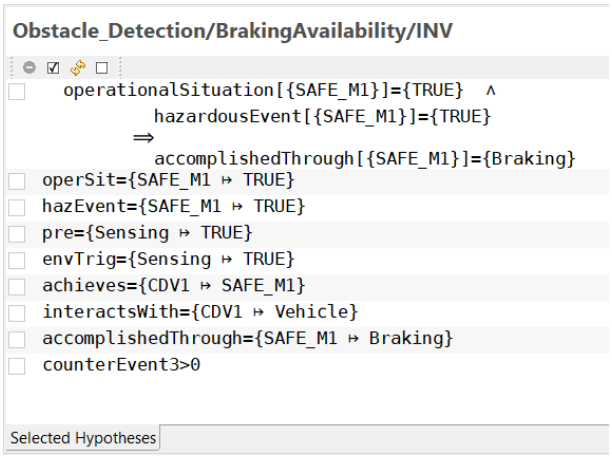


Figure 6.8: Selected Hypotheses for conducting Proofs in Rodin: An Excerpt concerning Proof-Obligation (PO) for Operational Availability Objective.

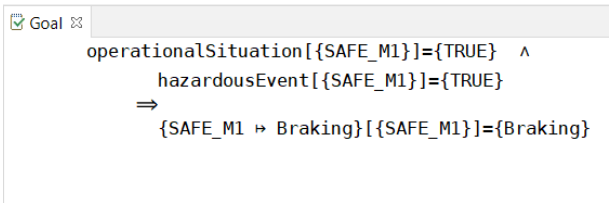


Figure 6.9: Excerpt of a Goal corresponding to the Operational Availability Objective.

designer, does not need to be aware of the theory for the proving task in the model instantiation as one can rely on the POs presented in the abstract model. This way, the model can be reused to adapt to the domain concepts, as several POs are shared among different kinds of systems [171].

Table 6.6: Number of Proof-Obligations (POs) in the Properties Model for CDV Use Case.

Proof-Obligation (PO) @	# Safety	# Security	# Safety and Security Interplay
Mission layer	8	3	4
Functional layer (Abstract)	8	8	-
Functional layer (Concrete)	35	42	7
Component layer	12	16	8

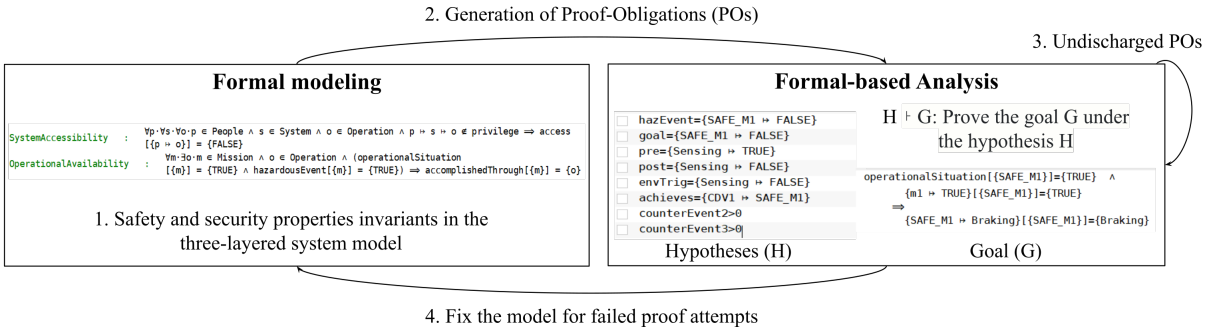


Figure 6.10: Formal Modeling and Analysis in Event-B.

6.3.2 Key Features of the Approach

The proposed approach exhibits several features to assist non-savvy engineers, e.g., system architects, in building safe and secure architectures via capturing and integrating safety and security expertise in the SE—System Engineering process. In line with these aspects, the contributions are made in the scope of the approach to model and analyze safety and security properties in unison. The distinctive features of the approach herein proposed lay in the following main axes:

- The approach relied upon existing Model-Driven Engineering (MDE) and formal-based techniques. The former allowed capturing the system, safety, and security domain expertise in an integrated fashion. The latter allowed discerning the resulting models, and jointly and consistently analyzing safety and security objectives, thus introducing architecture artefacts that ensured suitable safety and security levels. The approach

leverages the best of both worlds by systematically reducing the gap between the semi-formal and formal semantics.

- We adopted a global view of the SE process. This included a three-layered system, targeting high-level mission, functional, and component-based system development stages. The approach is amenable to covering these different phases irrespective of the design flow (top-down or bottom-up).
- Along with ensuring consistent design flow, the approach allows engineers to select a single layer adequate to the modeling granularity and analysis needs or the three layers together, thus resulting flexible enough. Since each layer has its respective formal interpretation, it can be used in standalone or coupled mode with other layers.

The above distinctive features are encompassed with the capabilities to specify and model pre-defined safety and security properties and verify them. This, in turn, supports the early identification and systematic analysis of their inter-dependencies and conflicts. The approach also contributes to the reusability of safety and security objectives' signatures across different design projects. For now, the feedback or results from the formal tool have not been fully integrated into the MDE modeler. Further research work is necessary to accomplish this objective, which will be discussed in the forthcoming Chapter 7. Furthermore, there is a possibility of other configuration choices involving, for instance, more missions, functions, functional paths, and functions belonging to more than one functional path that will be a subject of future work. A concrete summary of our work regarding the existing state-of-the-art discussed in Chapter 2 based on the characterization attributes used therein, is provided in Tables 6.7, 6.8, and 6.9.

Table 6.7: Our Results: Safety and Security Co-engineering.

Ref.	Contribution	Life-cycle Stage	System Specification Layer	Stakeholders	Propagation of Safety and Security Semantics	Safety and Security Interaction	Conflict Resolution
[50]	Co-engineering approach	D1	M, F, C	G	Y	Co	Y
[51]	Integrated design approach to safety and security	D1	M	D1, A	N	Co	Y

D1: Design | M: Mission, F: Functional, C: Component | G: Generic, A: Analyst, D1: Designer | Y: Yes, N: No | Co: Combined

6.3.3 Genericity for Applicability and Extensibility of the Approach

The instantiation of the modeling languages and formalisms in the context of the CDVs use case pointed out several important aspects that the approach must meet to address the integrated safety and security design and analysis. Based upon this, we discuss the genericity for the applicability

Table 6.8: Our Results: Means and Techniques to Support Safety and Security Design and Analysis.

Ref.	Modeling Aspects	Property	Relationship between Properties	Language	Tool	Evaluation	
						Use Case/ Case Study	Assessment Parameters
[53]	Three-layered system—Mission, Functional, and Component-Port-Connector (CPC)	Figure 4.11	Table 4.1	Unified Modeling Language (UML), First-Order Logic (FOL), Modal Logic, Event-B	Eclipse Papyrus, Rodin	-	Properties verification
[52]	Three-layered system—Mission, Functional, and CPC	Figure 4.11	Table 4.1	UML, Event-B	Eclipse Papyrus, Rodin	Converging road plan scenario	Properties verification

Table 6.9: Our Results: Context-oriented Characterization of the Approach for Safety and Security Design and Analysis.

Ref.	Target System	Application Domain	S/R/G	Key Assumptions
[53, 52]	Safety and security-critical systems	Automotive	ISO 26262, ISO 27001, ISO 15288	Layered representations and design methodology-specific

and extensibility of the proposed approach across two dimensions, i.e., languages and tools for implementation, and incorporating additional safety and security properties.

Languages and Tools for Implementation. We can highlight a salient feature of our approach that relies upon two orthogonal but complementary aspects: *generality* and *amenability* of the languages for specialization and refinement. Regarding generality, it is often considered a pre-condition to capture high-level and complex notions. The choice of First-Order Logic (FOL) as an entry language for safety and security objectives specification is aligned with this consideration. On the other hand, regarding the support for specialization and refinement, Event-B is a formal method that allows system-level modeling and analysis, along with the definition of extensible signatures in the form of invariants. This feature particularly facilitates the alignment of the incremental system, safety, and security engineering processes.

In addition, the three-layered system DSML offers a generalized way to abstract and capture complex use cases. We reused the three-layered system models, along with the safety and security objectives for specifying the static and dynamic aspects of the system, describing the use case of CDVs in Event-B. The set theory and FOL-based Rodin-aligned formal syntax will provide the system architects that are non-experts a way to transform models into formal specifications for verification. However, it is recalled from Chapter 5 that we can envision the applicability of the proposed approach with other formal specification languages, e.g., temporal logic [147], and model-checking tools, e.g., UPPAAL [172] or Alloy analyzer [48]. This would

require mapping the fundamental notions proposed in this work via interpretation to the target formal specification language, accompanied by the automated tool support. This will allow the use of the proposed approach among practitioners familiar with diverse MDE and formal-based tools and environments.

Incorporating Additional Safety and Security Properties. Applicability regarding the use case shows that the proposed approach fosters the *reuse* of formal specification of safety and security objectives in the form of invariants across different system designs, given that they rely explicitly on the respective layered models considered in this work. An engineer can instantiate the reusable set of safety and security objectives libraries by extending the signature definitions and corresponding invariants. Ultimately, it assists system designers or architects in integrated modeling and analyzing the interplay between safety and security properties. Automated suggestions can be provided in this regard as part of future work to address the violation of the objectives, which will be discussed in Chapter 7.

In addition, it is recalled that the set of safety and security objectives considered in this work are representative ones (refer to Section 1.1.2), and we established a way to identify relationships between them. However, the approach offers the capability to *extend* and incorporate additional desired properties, e.g., data confidentiality and message authenticity, or other time-based properties. This may require, in some cases, e.g., time-based properties, to enrich the DSML notions and DSML-to-EventB interpretations to handle those notions.

6.4 Conclusion

Summary. In this chapter, we showed the applicability of the approach for safety and security co-engineering proposed in Chapter 3 through a domain-specific application. In particular, the contributions presented in Chapters 4-5 were assessed via the design and analysis of an autonomous CDV system for a converging road plan scenario that demands both safety and security. As part of the concept phase, the initial system specification involved a preliminary analysis of the feared events, e.g., hazards and threats, regarding the three-layered system representation. This specification served as input to the overall approach in textual form. Accordingly, we instantiated the DSML profiles presented in Chapter 4 to model the CDV system architecture at the three layers, incorporating desired safety and security properties. The resulting system architectures were mapped to the Event-B specification by instantiating the formalisms presented in Chapter 5, which was further analyzed against the safety and security objectives and their interplay by reusing the formalized objectives signatures. It is noteworthy that the Event-B specification of the CDV system model involved hybrid means, wherein the

system elements were interpreted manually to Event-B language, and the relationships with the abstract specification were defined using the language elements and facilities, thus yielding a partially automated process. Moreover, the genericity of the approach was evaluated based on the outcomes obtained from each contribution.

Usefulness. Recalling the question mentioned in Section 6.1—*How to facilitate incorporation of safety and security concerns and their joint analysis in the design stages of the SE—System Engineering process?*—the approach facilitates the integration of the safety and security properties in the engineering process of the target SUD—System Under Design by construction and definitions of the notions defined in the DSML profiles. This is achieved by selecting a layer from the three-layered model, which corresponds to different design stages, and incorporating the properties in the model referring to the layered elements in the respective profiles. The formal-based analysis of the system architecture candidates against safety and security objectives allowed us 1) to verify whether the modeled system is consistent and conforms to the corresponding requirement specifications and 2) to analyze the potential inter-dependencies between them and their impact on the modeled system.

Concluding Remarks. The properties belonging to categories like availability and integrity are analyzed from the safety perspective in the use case. However, when considered in another context, they may influence the system's security. For example, integrity property may be applied to detect any unauthorized alterations of the messages in transmission. **It is recalled from Chapter 4 that the purpose of the notion of property categories is not to emphasize a strict classification of the properties in the safety and security domain but to facilitate analysis in a given context.**

Following the MDE paradigm, the proposed approach considers a loop involving modeling at the front end with three-layered system and safety and security properties models as the outcome, followed by the formal-based interpretation of the models in a tooled-formal language, namely Event-B. This interpretation is currently made using manual means, relying upon hand-made tables in Section 5.6, and further work is needed to automatize this step. The Event-B specification constitutes the back end that results in the generation of the safety and security analysis artefacts regarding the system architectural model for verification. The results of the verification process could then be used for generating feedback—for instance, related to the detection of properties violation or potential conflicts, which again demands further work.

Chapter 7

Conclusion and Future Work

Contents

7.1 Summary and Contributions

In this thesis, we addressed the problematics concerning a lack of methodological support for an integrated system, safety, and security analysis, reconciling respective domain expertise during the early design stages of the System Engineering (SE) process. The inherent complexities of standalone safety and security engineering practices and semantic gaps pose difficulties in conducting a consistent analysis considering their mutual influence. To this end, we proposed a model-based joint design and analysis approach and tooled-framework to better support and ease safety and security co-engineering. The approach has been instantiated in the context of a three-layered—*mission*, *functional*, and *component*-based—system representation, targeting a positive vision of safety and security, particularly *objectives*, encoded as *properties*. It relied upon 1) the Model-Driven Engineering (MDE) paradigm allowing the use of models for conducting the system design incorporating safety and security properties, and 2) the rigorousness of formal-based techniques facilitating safety and security properties analysis in unison. Accordingly, the specific contributions of this work are five-fold, as shown in Table 7.1 and summarized in the following sub-sections.

Table 7.1: Summary of the Thesis Contributions.

Problematic	Contribution	Language/Tool/Technique Used
P1: Lack of tooled-approach to facilitate incorporation of integrated safety and security analysis in the System Engineering (SE) process	**C1:** A method to build safety and security modeling framework, supporting the safety and security co-engineering process (Chapter 3)	Model-Driven Engineering (MDE) and formal-based techniques
P2: Lack of consistent semantic transfer across the different granular system representations	**C2:** A three-layered system representation/model for capturing the details corresponding to different stages of system development (Chapter 3)	Conceptualization via different-granularity system viewpoints
P3: Lack of modeling language, consolidating system, safety, and security domain concepts specification in unison	**C3:** A set of Domain-Specific Modeling Languages (DSMLs) for three-layered system and safety and security properties modeling (Chapter 4)	Meta-modeling and profiling (Unified Modeling Language (UML))
P4: Error-prone design-stage safety and security analysis due to the ambiguities in properties specifications and interplay	**C4:** Technology-independent semantics for the modeling languages and interpretations to tooled-language (Chapter 5)	Set theory, First-Order Logic (FOL), Modal Logic, Event-B
P5: Lack of automated tool support for integrated safety and security design and analysis	**C5:** An operational tool support integrating MDE and formal-based techniques (Chapters 4, 5, 6)	Eclipse Papyrus, Rodin

7.1.1 Method for Safety and Security Co-design and Analysis

The proposed approach comprised a method to build a safety and security modeling framework and support their co-engineering process regarding *problematic* **P1**, as presented in Chapter 3. This involved conceptual modeling, developing the modeling languages, and their interpretations to tooled-formal language to facilitate integrated design and analysis of the system and safety and security properties. The safety and security co-engineering process for a target application involved the model-based system architecture design and reusing the existing formal model libraries for the properties to verify the fulfillment of underlying safety and security requirements—refer to Section 3.2.3. The methodological support allows an engineer to conduct system and properties modeling and verification targeting the system architectural design stages, thereby facilitating the integration of the proposed approach in the SE process. The iterative methodology assists the system designer with the passage of knowledge from textual requirement specifications to system and properties models that are verifiable regarding conformity to those requirements.

7.1.2 Three-layered System and Properties Model

The three-layered system and properties model provided the basis for the proposed method to support the safety and security co-engineering process, as presented in Chapter 3. It allowed capturing the target System Under Design (SUD) regarding different stages of system development with varying granularities, viz. *Mission* for high-level teleological representations to capture and understand the overall purpose of the complex engineered systems, *Functional* to

capture the system's functionality, and *Component* for the detailed technical architecture of the system—refer to Section 3.2.2. In addition, it allowed defining safety and security properties concerning the system-related aspects at each layer. In this regard, relationships were derived from the impact of properties at a given layer or across layers, within or between the safety and security domains, regarding *problematic* **P2**. This allows an engineer to select a single layer or multiple layers adequate to the modeling granularity and analysis needs, thus resulting flexible enough.

7.1.3 Modeling Language and Integrated Design Framework

In Chapter 4, to address the *problematic* **P3**, we proposed a set of Domain-Specific Modeling Languages (DSMLs) that capture the target SUD and associated safety and security properties regarding different stages of system development represented by the three-layered model. The DSMLs were first defined as *meta-models* to capture the domain concepts and their relationships, thereby allowing system architects to understand the language and detect potential inconsistencies—refer to Sections 4.4 and 4.5. Subsequently, they were implemented as *profiles* built by stereotyping the domain modeling concepts and mapping the same to Unified-Modeling Language (UML) meta-classes, relying upon the UML standard. The selected meta-classes are semantically aligned with the notions in the meta-models and extended to incorporate the specificities of the system model at the three layers and the fundamental attributes capturing the desired safety and security properties. The DSMLs ensure separation of modeling purposes via decoupling of the system and properties, thereby simplifying the modeled reality. This contribution provides a basis to analyze safety and security properties and their interplay by supporting the system architecture and properties modeling in unison.

7.1.4 Formal-based Rigorous Specification

In Chapter 5, to address the *problematic* **P4**, we proposed the formalization of the DSML profiles for rigorous specification and verification of the modeled system and safety and security properties. Accordingly, we defined *logical specification* for the system and properties DSMLs at the three layers based upon set theory notions. This was followed by the formal specification of the safety and security properties and their interplay, relying upon First-Order Logic (FOL) and Modal Logic. The formalisms allowed the interpretation of the models to the Event-B language to obtain a concrete specification. This contribution supports formal reasoning of the inter-dependencies between the safety and security concerns in the design stages of the system development. The Event-B specification of the three-layered system being generic, can

be instantiated via a refinement facility to obtain a formal-based specification of the target SUD and conduct analysis. This indeed involves refining the events in the abstract machine to capture the target system behavior, along with the generation of existential Proof-Obligations (POs) for witnesses. Accordingly, the safety and security properties specified as invariants in the abstract machine are also incorporated.

7.1.5 Tool-chain Support

Furthermore, to address the *problematic* **P5**, we used the following set of tools that support and automatize the different phases of the proposed approach:

- *Papyrus* for conceptual meta-modeling, profiling, and modeling of the system aspects at the three layers and safety and security properties, as detailed in Section 4.6.
- *Rodin* for model checking and properties verification via theorem proving, as detailed in Section 5.7.

To address the approach's applicability regarding integrated modeling and verification of safety and security system architecture, we presented a tool-chain support prototype architecture in Section 3.3. Based upon that, an example scenario involving the use of the tools mentioned above is depicted in Figure 7.1, comprising the following two blocks:

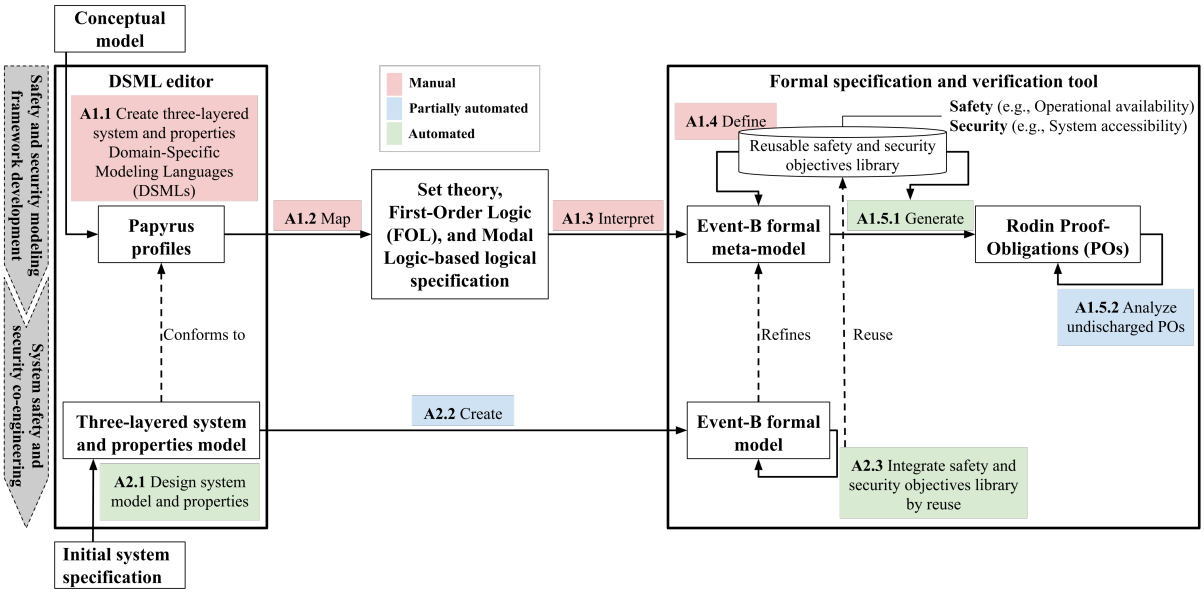


Figure 7.1: Example Scenario for Tool Support Architecture and Related Approach Artefacts.

- *Safety and security modeling framework development*, which primarily involved the creation of the DSML meta-models and profiles using Papyrus (A1.1), defining the mapping for the fundamental notions captured by the profiles to a logical specification based upon set theory, along with FOL and Modal Logic-based properties formal specification (A1.2), and their formal-based interpretation to Event-B (A1.3). This was followed by defining a library of reusable safety and security objectives signatures (A1.4). Finally, an integrated verification of safety and security objectives was conducted, which involved the automatic generation of POs in Rodin (A1.5.1), followed by checking the undischarged POs (A1.5.2) for identifying and fixing the issues related to the well-definedness of the predicates and selection of hypothesis in the back end. Automated interpretation of the models to their formal-based specification and provisioning of the feedback that could be exhibited to the end user at the modeling stage constituting the front end, if needed, will be part of the perspective work, as detailed in Section 7.3.

- *System safety and security co-engineering*, which involved the modeling of system architecture and safety and security properties for a target application by an engineer, based on the already defined profiles (A2.1), followed by the creation of a formal model in Event-B using semi-automated means (A2.2), relying upon the language constructs to instantiate the formal meta-model in (A1.3), and incorporation of formal-based safety and security objectives signatures using pre-defined libraries (A2.3).

Notably, the approach is agnostic of the underlying tool-chain technology, which is only developed for proof-of-concept purposes. Notwithstanding this choice, the tool-chain integrates mechanisms that facilitate approach extension and specialization. Being generic and mostly automated, it aims to enable model interpretation towards other formal frameworks, thus alleviating the complexity of manually integrating formal analysis of safety and security concerns into the design phase.

7.1.6 Approach Illustration and Assessment

We illustrated the applicability of the proposed approach and tooled-framework by relying upon the use case of autonomous Connected-Driving Vehicles (CDVs) from the automotive domain. Specifically, we considered a converging road plan scenario relevant to both safety and security. This involved exemplifying the notions involved in the *contributions* **C3** and **C4**, in Chapters 4 and 5, respectively, to contribute to the safety and security modeling framework development block in Figure 7.1. The complete instantiation of the approach and proposed tool-chain support prototype for the *contribution* **C5** is detailed in Chapter 6, which contributes to the system safety

and security co-engineering block. We also discussed the evaluation of the approach based on the results obtained for each contribution—refer to Section 6.3. The selection of the use case scenario has been considered an important aspect in this work to assess the approach regarding integrated safety and security properties modeling and verification. In essence, the framework and the resulting tool-chain support shed light on the key aspects regarding the use of MDE and formal-based techniques for the design and analysis of system safety and security in unison.

7.2 Shortcomings and Future Work

The proposed approach and tooled-framework aim to provide methodological support to assist non-savvy engineers in safe and secure system architecture design in a coordinated fashion. Despite the features they offer, we identified certain lacks and, accordingly, define the following future objectives to consolidate our work:

1. **Analyzing complex configurations:** The vertical relationships between the layered DSMLs are illustrated through some instances, mainly covering the target SUD—refer to Section 4.4.2, and safety and security objectives—refer to Section 4.5.2. Nevertheless, there is a possibility of even more complex configurations involving, for instance, 1) composite missions that can be further decomposed to sub-missions, 2) realization of an operation by more than one function, i.e., a one-to-many relationship, 3) functions belonging to more than one functional path, 4) allocation of a function to more than one component, i.e., a one-to-many relationship, and 5) incorporation of more missions, functions, or functional paths. Moreover, the use case in Chapter 6 may call upon a large set of additional objectives, e.g., those within the confidentiality and authenticity categories. Thus, we plan to explore and analyze them to assess the approach's coverage and confidence. It is worth noting that an objective, e.g., within the integrity and availability categories, in a given context may affect the overall safety of the system, while in the other, it may influence the security-related requirements. Therefore, we also aim to consider such possibilities.

2. **Classification of conflicts:** The approach paves the way to analyze the interplay between safety and security objectives across the three-layered system model, which was illustrated through some instances—refer to Section 5.4.4. However, the proposal is limited from the point of view of conflict-handling strategies. This necessitates determining the potential causes of conflicts that can be due to 1) the mechanisms to ensure the respective objectives, e.g., encryption in case of message confidentiality and time-out in case of message freshness, 2) no simultaneous fulfillment of the desired post-conditions, e.g., in case

of operational availability and system accessibility, and 3) the influence of the details offered by the layers, e.g., time-based notions in case of functional path availability at the functional layer and message freshness at the component layer. All these cases can provide dimensions for identifying and analyzing conflict mitigation mechanisms based on the features they can offer. Therefore, we aim to classify the conflicts based on these dimensions, considering the elementary horizontal and vertical links in the three-layered model.

3. **Negative safety and security perspectives:** A suitable coverage of the feared events potentially impacting the fulfillment of the safety and security objectives demands the integration of unwanted properties constituting a negative vision of safety and security. For example, if sub-systems of a system are not operational anymore, it can lead to the system's failure [30]. Likewise, if the functionality of the components is not available, it might be related to the faults [30]. In many cases, the positive and negative visions are dual to each other [173]. Accordingly, *fault*, *failure*, *hazard*, and *threat* models can be introduced to serve as evidence for the system design during various stages of system development to meet the defined goals and safety and security objectives in the presence of feared events. Thus, we aim to formulate domain-specific language as a reference to the negative vision and establish a common understanding of the stakeholders involved in safety and security.

4. **Formal analysis in Event-B:** The analysis of system safety and security properties can be carried out in Event-B using three complementary approaches: *theorem proving*, *model checking*, and *animation*. In this work, we relied upon theorem proving as mentioned in Section 5.3. However, in case of the introduction of feared events, the system needs to be deadlock-free, for which it should be guaranteed that it can transit from a dangerous state to a safe state. Model checking identifies the unreachable states and automatically verifies whether the system is deadlock-free. Moreover, animation can identify the potential weak links in the formal specifications and harmonize the same with the informal specification of the system and properties by visually observing the system behavior. Thus, we seek to explore these capabilities. In addition, we aim to improve the usage of Event-B using more elaborated facilities, like injective functions and refinements, to formalize the vertical relationships across layers.

5. **Safety and security interplay for systems in practice:** To demonstrate the technical applicability of the proposed approach, we relied upon the use case of CDVs from the automotive domain presented in Chapter 6. Another significant future work in this

direction is to determine whether the approach would suffice to address the safety and security properties of such industrial systems in practice. To this end, we aim to collect feedback from industry practitioners and incorporate the same to assess and address the improvements. We also seek opportunities to address other industrial domains for the approach applicability.

7.3 Perspectives

Apart from the short-term goals discussed above, the work presented in this Ph.D. thesis opens important orientations to be investigated in the middle and long term. This mainly includes enhancement of the proposed approach concerning the following aspects:

1. **Tool support:** Regarding the tool-chain support prototype architecture described in Section 3.3, we seek to study the integration of the proposed approach with other modeling frameworks and tools based upon MDE and formal-based techniques to support the following:

   (a) *Automation:* We would like to automatize the integration of safety and security objectives libraries in the system architectural model when the designer selects the respective elements at the three modeling layers. The aim is to simplify the system safety and security modeling task conforming to the DSML meta-models. This can be achieved by developing a textual or graphical editor using Xtext [174]—for instance.

   (b) *Model transformation:* We plan to incorporate transformation engines in the functionality of the tool-chain support architecture for 1) automatic or incremental [175] generation of the Event-B formal specification from the DSMLs, addressing the tables presented in Sections 5.5.2 and 5.5.3 and 2) reporting mechanism to the end-user regarding any violation of safety and security objectives, or inter-dependencies like conflicts. To this end, we seek to explore features provided by Xtend [174]—for instance.

2. **Design patterns for safety and security co-engineering:** We aim to incorporate the outcomes of the approach to a more global process within the pattern life-cycle. This will involve exploring sophisticated techniques to build a pattern system [176], providing the creation, selection, and integration of design patterns for safety and security properties and their relationships, to the overall SE—System Engineering process.

3. **Solution space for safety and security interplay:** Besides the conflicts appearing during problem specification, the solutions, e.g., mechanisms and policies, provided at the architectural levels, may also suffer from the same issue. For instance, implementing ciphering techniques as a security mechanism for the confidentiality of data transmission may have a non-negligible impact on the system's overall performance due to the introduction of delays, potentially violating the safety constraints [91]. In such cases, the approach needs to be extended to resolve the conflicts without prioritizing safety or security solutions.

4. **Implementation phase:** Our approach focused on the architectural design phase of the SE process. However, safety and security properties violations and inter-dependencies like conflicts may also arise once the system is implemented [177], which is an even more complex undertaking. In such cases, testing practices can reveal some of them. This requires the extension of the approach with code generation facilities, complying with the domain standards to move to the implementation phase. Accordingly, we need to inspect a way to define the specified behavior of the system being implemented. Moreover, further work is needed to address annotations in the system and properties models to ensure that the properties verified at the model level inherit at the code level.

5. **Standardization and certification:** In this thesis, we relied upon the system, safety, and security domain-specific standards to use the respective fundamental notions as input to the overall approach. However, there is a need for standards and certification practices regarding the integration of safety and security concerns during system development. Besides, current certification practices contemplate safety and security in an isolated context towards compliance with the stringent domain-specific standards. For instance, the conventional safety standards do not consider inter-connectivity and interactions among systems, which is one of the primary reasons for raising security issues and propagation of failures among systems [56]. Cross-disciplinary knowledge from the system's application context can be leveraged in this direction to identify the intersections between both domains. Existing industrial safety (e.g., ISO 26262) and security (e.g., ISO 27000) standards can be revisited in terms of their role in reducing the ambiguity of a joint safety and security engineering process.

Appendices

Appendix A

Formal Methods

A.1 Classification of Formal Building Blocks in Literature

In literature, formal specification languages and tools have been collectively described as formal methods, with a classification based on specific characteristics, including foundational theory, the target of application, and research or user-oriented aspects [178]. Formal constructs have been widely accepted for specification in agent-based systems within highly distributed and concurrent environments. In light of this, a discussion on the underlying aspects of formal modeling and detailed classification of formal methods, along with associated research issues, can be found in [179].

The specification style using formal constructs adopted during the system design phase may rely on the formal semantics of the programming languages and general specification of the system [43]. Some of the formal languages that exist in the literature for specifying system properties include Z notation [180], Vienna Development Method (VDM) [181], Petri nets [182], and process algebra [183]. Logic-based notions are accompanied by different levels of expressiveness and broadly include propositional logic [184], First-Order Logic (FOL) [185], temporal logic [186], Burrows–Abadi–Needham (BAN) Logic [187], and knowledge logic [188]. A plethora of open-source tools based on formal notions are available for assisting formal specification, modeling, reasoning, and analysis during system design [49]. Tools based on formal languages include Alloy [48], Games, Omega-Automata, and Logics (GOAL) [189], Electrum [190], and UPPAAL [172].

Table A.1 summarizes the classifications of the formal building blocks in the literature, highlighting the basis of classification and corresponding examples.

Table A.1: State-of-the-art on Classification of Formal Building Blocks.

Formal Notions	Classification Basis	Examples
Formal specification [43]	Formal semantics of programming languages	Axiomatic, Denotational, Operational
	General system specification	Model-based, Algebraic
Formal languages [43]	Predicate logic-based	Z, VDM, B
	Temporal logic-based	CTL, TLA
	Process algebras	CSP, CSS, CIRCAL
	Others	RSL, Action semantics, ASM
Formal tools [49]	Abstract model checking	SPIN, UPPAAL, SMV, NuSMV, FDR, Alloy, Simulink design verifier
	Hardware description language checking	Questa formal, Solidify, JasperGold, Incisive
	Software correctness checking	Frame-C, BLAST, Java Pathfinder, Spark ADA, Malpas
	Provably correct design creation	VDM, Z, B, Event-B, Rodin
Formal methods [178][179]	Model-oriented approaches	ASM, B-Method, RAISE, VDM, Z notation, FSMs, Statecharts, Petri Nets, XM
	Property-oriented approaches	Axioms, Algebra
	Process algebras	CSP, CCS, Pi-calculus, IOA
	Logics	Temporal logic, RTL, BDI logics, KARO logic
	Other approaches	Artificial physics, SCR, Mathematical analysis, Game theory, OCL, Hybrid approaches
	Foundational theory	Set theory, Transition systems, Calculus, Universal algebra
	Target application	Protocols, Real-time systems, Data processing

Appendix B

Safety and Security Standardization Efforts

Herein, we outline the published standards that provide a conceptual basis for capturing the engineered systems' safety and security concerns. Specifically, we focus on how safety and security properties are defined in these documents considering the specificities of the target application domain.

B.1 Safety Standards

In the safety context, Table B.1 outlines some of the key application domain-specific standards, regulations, and guidelines that reflect the industrial perception of the safety properties [26]. International Electro-technical Commission (IEC) 61508 is a functional safety standard that applies to industries dealing with electrical, electronic, or programmable electronic safety-critical systems [191]. According to this umbrella standard, integrity is "the probability with which a safety-related system satisfactorily performs the required safety functions, under all the pre-defined conditions for a stated period".

Various industry-specific variants of IEC 61508 have been defined by standards-making bodies and adopted by the research and development communities. International Standard Organization (ISO) 26262 captures the functional safety concerns, like severity, exposure, and controllability, of electrical and electronic systems in automobiles [120]. According to this standard, availability is "the capability with which a product provides an already-stated function on demand, under all the pre-defined conditions for a stated period". International Atomic Energy Agency (IAEA) published safety standards for nuclear safety to capture the essential requirements, principles, and recommendations associated with the domain [213]. According

Table B.1: Application Domain-specific Safety Standards, Regulations, and Guidelines.

Application Domain	Standards (S)/ Regulations (R)/ Guidelines (G)
Generic	IEC 61508 [191] (S)
Automotive	ISO 26262 [120] (S), EN 50495 [192] (S), ISO/PAS 21448 [193] (S)
Aviation	DO 178 [194] (G), DO 254 [194] (G)
Chemical process	IEC 61511 [195] (S)
Power grid	NERC reliability standards [196] (S)
Railway	EN 50128 [197] (S), IEC 62278 [198] (S)
Machine	IEC 62061 [199] (S), ISO 13849 [200] (S), IEC 61131 [201] (S)
Nuclear power plants	IEC 61513 [202] (S), IAEA safety standards [203] (S), IEC 60880 [204] (S)
Medical	IEC 62304 [205] (S), IEC 60601 [206] (S), ISO 14971 [207] (S), EU medical device regulation [208] (R), ISO 13485 [209] (S), FDA [210] (R)
Household	IEC 60335 [211] (S), IEC 60730 [212] (S)

to this standard, safety is “an attribute of dependability that describes the trustworthiness of the system as a whole”. This standard describes availability as “the fraction of time for which the system is capable of fulfilling its planned purpose”. The latest definition of availability provided by this standard follows the one presented by ISO 26262, under the provision of essential external resources.

B.2 Security Standards

In the security context, experts and researchers have considered some properties, including Confidentiality, Integrity, Availability, and Authenticity, abbreviated as CIAA, as a benchmark for secure computing and communication systems. These are based on the security standards, regulations, and guidelines for different application domains and target aspects of security, as provided in Tables B.2 and B.3, respectively. Notably, some of the standards highlighted in Table B.3 address security in a holistic way, thus covering different aspects, subjects, views, etc. According to ISO/IEC 27000 [214], 1) confidentiality is “the property of non-disclosure or unavailability of information to unauthorized entities, processes, or individuals”, 2) Integrity is “the property of completeness or accuracy”, 3) Availability is “the property of usability and accessibility when demanded by an authorized entity”, and 4) Authenticity is “the property that an entity is what it claims to be”. Moreover, this standard defines other security properties, including non-repudiation, accountability, and reliability. National Institute of Standards and Technology (NIST) Special Publication (SP) 800-12 Rev. 1 [215] defines confidentiality as “preserving authorized restrictions on access to information and its disclosure, including ways to protect proprietary information and personal privacy”. Likewise, it defines integrity as “the ability to guard against improper information modification or destruction, ensuring information

authenticity and non-repudiation".

Table B.2: Application Domain-specific Security Standards, Regulations, and Guidelines.

Application Domain	Standards (S)/Regulations (R)/Guidelines (G)
Automotive	ISO/SAE DIS 21434 [216] (S)
Aviation	ATA Spec 42 [217] (S), ED-201 [218] (G), ED-202A [219] (G), ED-203A [220] (G), ED-204 [221] (G), ARINC 664 [222] (S), ARINC 811 [223] (S)
Chemical process industry	CFATS [224] (S)
Power grid	IEC 62351 [225] (S), NERC CIP standards [226] (S)
Railway	AS 7770 [227] (S), VDE 0831-102 [228] (S), VDE 0831-104 [229] (S)
Machine	EU machinery directive 2006/42/EC [230] (G)
Nuclear power plants	IEC 62645 [231] (S), IEEE 692-2010 [232] (S), NRC RG 5.71 [233] (R)
Medical	FDA [210] (R), AAMI TIR 57 [234] (S), DTSec [235] (S), DTMoSt [236] (G)
Household	ETSI [237] (S)

Table B.3: Target Aspect-specific Security Standards, Regulations, and Guidelines.

Target Aspect of Security	Standards (S)/Regulations (R)/Guidelines (G)
Secure information management	ISO 27001 [238] (S), ISO 27032 [239] (S)
Software and hardware	ISO 15408 [240] (S)
Industrial Automation and Control Systems (IACS)	ANSI/ISA 62443 [241] (S)
Critical infrastructure	NIST [242] (S)
Information and Communications Technology (ICT)	DHS [243] (S/G)

CHAPTER B. SAFETY AND SECURITY STANDARDIZATION EFFORTS

Appendix C

Underlying Formal Platform

C.1 Event-B Formal Method

Context. A context describes static properties of the model that do not change over time. Each context has a distinct name with regard to the other components, viz. contexts and machines, within the same model. It comprises the following:

- **Extends:** Lists the contexts that are extended explicitly. This allows using the sets, constants, and axioms of the extended contexts as references.
- **Sets:** Also called carrier sets, it is used to declare the user-defined types. These are implicitly assumed to be non-empty.
- **Constants:** Enlists the constants introduced in the context. These have distinct identifiers regarding the constants and sets in the extended contexts or the context itself. The type of constants is declared in the axioms section.
- **Axioms:** Contains the list of predicates, called axioms, which define rules that are obeyed by the constants. Axioms can be marked as "theorems". A theorem can be proved by using the predicates listed before it in the context. The order of axioms is concerned with avoiding circular reasoning.

A general structure of a context is depicted in Figure C.1.

Machine. A machine describes the dynamic behavior of the model by means of variables whose values change by the events as the system evolves. Each machine has a different name from the other components, viz. contexts and machines, within the same model. It comprises the following clauses that are not all mandatory, and some can be left empty:

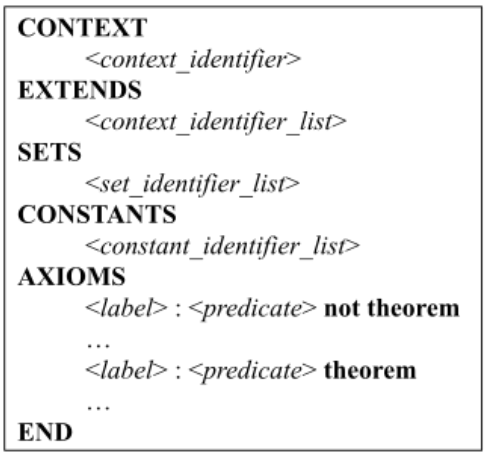


Figure C.1: Structure of an Event-B Context.

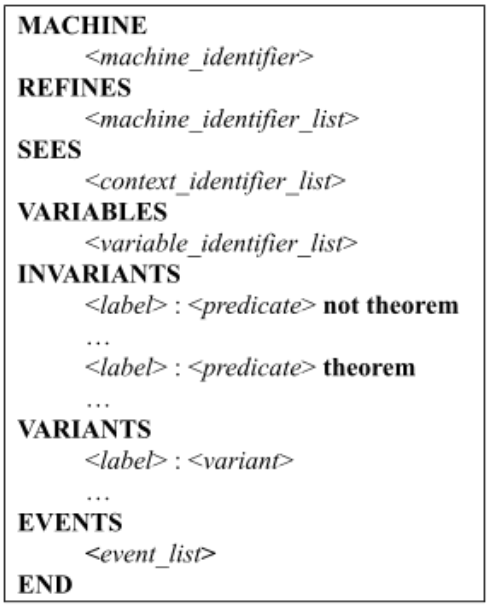


Figure C.2: Structure of an Event-B Machine.

- **Refines:** Contains the machine (if any) that is refined by this machine.
- **Sees:** Contains the list of contexts that are referred to explicitly by this machine. Accordingly, the machine can use the sets and constants defined therein or seen explicitly. Also, axioms defined in the seen context are used in proofs in the machine as hypotheses.
- **Variables:** Lists the variables introduced in the machine, having distinct identifiers. However, some variables can be the same as some in the abstract machine. The type of variables is declared in the invariants section.
- **Invariants:** Lists the predicates that should be true for every reachable state during the model execution. Invariants can be marked as "theorem". A theorem can be proved by using the predicates in the seen contexts and abstract machines and local invariants and theorems written before it. An invariant may also comprise variables of the refined machine and is known as gluing invariant, as it glues the space of the refined machine and the present machine.
- **Variants:** These are defined in a machine comprising some "convergent" events. The "Initialization" event determines the values of the variables that may change by the events.
- **Events:** Lists the events defined in the machine.

A general structure of a machine is depicted in Figure C.2.

Events. An event describes the transitions in a model and represents a relation between two successive states. It comprises the following clauses that are not all mandatory, and some can be left empty:

- **Status:** It can be any one of "ordinary", "convergent" (where the event has to decrease the variant), or "anticipated" (where the event must not increase the variant).
- **Refines:** Lists the abstract events refined by this event.
- **Any:** Lists the parameters of the event.
- **Where:** Lists the guards of the event. A guard specifies when an event is allowed to occur. In other words, these represent the necessary conditions for the event to be enabled. An event executes iff the values of the machine's variables and parameters match the values specified in the guard.
- **With:** Lists the witnesses of the abstract event refined by this event.
- **Then:** Lists the actions of an event. An action denotes the changes that will be applied to the variables when the event gets enabled.

A general structure of an event is depicted in Figure C.3.

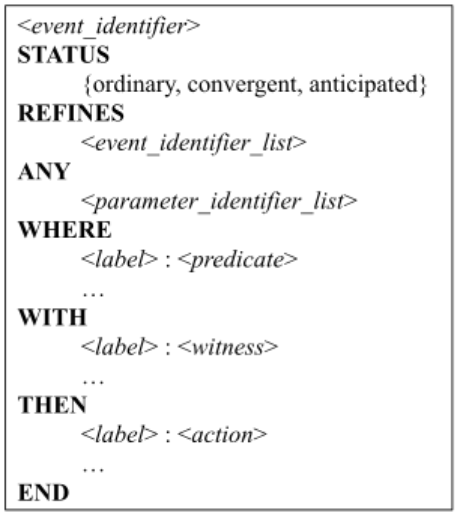


Figure C.3: Structure of an Event-B Event.

Bibliography

[1] Jason Borenstein, Joseph Herkert, and Keith Miller. Self-driving cars: Ethical responsibilities of design engineers. *IEEE Technology and Society Magazine*, 36(2):67–75, 2017.

[2] Gregory Travis. How the boeing 737 max disaster looks to a software developer. *IEEE Spectrum*, 18, 2019.

[3] Thomas M Chen and Saeed Abu-Nimeh. Lessons from stuxnet. *Computer*, 44(4):91–93, 2011.

[4] Miguel A De Miguel, Javier Fernández Briones, Juan Pedro Silva, and Alejandro Alonso. Integration of safety analysis in model-driven software development. *IET software*, 2(3):260–280, 2008.

[5] Anupam Chattopadhyay, Kwok-Yan Lam, and Yaswanth Tavva. Autonomous vehicle: Security by design. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[6] Derrick Dominic, Sumeet Chhawri, Ryan M Eustice, Di Ma, and André Weimerskirch. Risk assessment for cooperative automated driving. In *Proceedings of the 2nd ACM workshop on cyber-physical systems security and privacy*, pages 47–58, 2016.

[7] Marilyn Wolf and Dimitrios Serpanos. Safety and security in cyber-physical systems and internet-of-things systems. *Proceedings of the IEEE*, 106(1):9–20, 2017.

[8] Imane Cherfa, Nicolas Belloir, Salah Sadou, Régis Fleurquin, and Djamal Bennouar. Systems of systems: From mission definition to architecture description. *Systems Engineering*, 22(6):437–454, 2019.

[9] Emmanuel Letier and Axel Van Lamsweerde. Deriving operational software specifications from system goals. *ACM SIGSOFT Software Engineering Notes*, 27(6):119–128, 2002.

[10] P Oliveira, A Pascoal, V Silva, and C Silvestre. Design, development, and testing of a mission control system for the marius auv. In *Proceedings of the 3rd Workshop on Mobile Robots for Subsea Environments, Toulon, France*. Citeseer, 1996.

[11] Arquimedes Canedo, Eric Schwarzenbach, and Mohammad Abdullah Ai Faruque. Context-sensitive synthesis of executable functional models of cyber-physical systems. In *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 99–108. IEEE, 2013.

[12] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Functional model-based design methodology for automotive cyber-physical systems. *IEEE Systems Journal*, 11(4):2028–2039, 2015.

[13] Ashok Goel, Spencer Rugaber, and Swaroop Vattam. Structure, behavior & function of complex systems: The sbf modeling language. *International Journal of AI in Engineering Design, Analysis and Manufacturing*, 23(1):23–35, 2009.

[14] Shourong Lu, Wolfgang A Halang, and Janusz Zalewski. Component-based hazop and fault tree analysis in developing embedded real-time systems with uml. *WSEAS Transactions on Computers*, 4(12):1852–1857, 2005.

[15] Kaiyu Wan, KL Man, and D Hughes. Specification, analyzing challenges and approaches for cyber-physical systems (cps). *Engineering Letters*, 18(3), 2010.

[16] Lawrence C Barr, Richard Newman, Ersin Ancel, Christine M Belcastro, John V Foster, Joni Evans, and David H Klyde. Preliminary risk assessment for small unmanned aircraft systems. In *17th AIAA Aviation Technology, Integration, and Operations Conference*, page 3272, 2017.

[17] Jordi Dunjó, Vasilis Fthenakis, Juan A Vílchez, and Josep Arnaldos. Hazard and operability (hazop) analysis. a literature review. *Journal of hazardous materials*, 173(1-3):19–32, 2010.

[18] PJ Wilkinson and TP Kelly. Functional hazard analysis for highly integrated aerospace systems. In *IEE Certification of Ground/Air Systems Seminar (Ref. No. 1998/255)*, pages 4–1. IET, 1998.

[19] Thomas Maier. Fmea and fta to support safe design of embedded software in safety-critical systems. In *Safety and reliability of software based systems*, pages 351–367. Springer, 1997.

[20] Andreas L Opdahl and Guttorm Sindre. Experimental comparison of attack trees and misuse cases for security threat identification. *Information and Software Technology*, 51(5):916–932, 2009.

[21] Ludovic Piètre-Cambacédès and Marc Bouissou. Cross-fertilization between safety and security engineering. *Reliability Engineering & System Safety*, 110:110–126, 2013.

[22] Nary Subramanian and Janusz Zalewski. Assessment of safety and security of system architectures for cyberphysical systems. In *2013 IEEE International Systems Conference (SysCon)*, pages 634–641. IEEE, 2013.

[23] Saad Zafar and R Geoff Dromey. Integrating safety and security requirements into design of an embedded system. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8–pp. IEEE, 2005.

[24] Donald G Firesmith. Common concepts underlying safety security and survivability engineering. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2003.

[25] Haiying Sun, Jing Liu, Xiaohong Chen, and Dehui Du. Specifying cyber physical system safety properties with metric temporal spatial logic. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 254–260. IEEE, 2015.

[26] Ludovic Piètre-Cambacédès and Claude Chaudet. The sema referential framework: Avoiding ambiguities in the terms "security" and "safety". *International Journal of Critical Infrastructure Protection*, 3(2):55–66, 2010.

[27] Khaled M Khan and Jun Han. Composing security-aware software. *IEEE software*, 19(1):34–41, 2002.

[28] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of dependability. *Department of Computing Science Technical Report Series*, 2001.

[29] William Stallings, Lawrie Brown, Michael D Bauer, and Michael Howard. *Computer security: principles and practice*, volume 2. Pearson Upper Saddle River, 2012.

[30] Iso 26262-1:2011 road vehicles — functional safety — part 1: Vocabulary. `https://www.iso.org/standard/43464.html`, 2019. Accessed: September 2019.

[31] Iso/iec 27000:2018 information technology — security techniques — information security management systems — overview and vocabulary. `https://www.iso.org/standard/73906.html`, 2018. Accessed: September 2019.

[32] Alastair Ruddle, David Ward, Benjamin Weyl, Sabir Idrees, Yves Roudier, Michael Friedewald, Timo Leimbach, Andreas Fuchs, Sigrid Gürgens, Olaf Henniger, et al. Deliverable d2. 3: Security requirements for automotive on-board networks based on dark-side scenarios. *EVITA project*, 2009.

[33] Guohui Li, Jianjun Li, Bing Guo, et al. Maintaining data freshness in distributed cyber-physical systems. *IEEE Transactions on Computers*, 68(7):1077–1090, 2019.

[34] Mohamed Al-Kuwaiti, Nicholas Kyriakopoulos, and Sayed Hussein. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications Surveys & Tutorials*, 11(2):106–124, 2009.

[35] Manfred Broy, Martin Feilkas, Markus Herrmannsdoerfer, Stefano Merenda, and Daniel Ratiu. Seamless model-based development: From isolated tools to integrated model engineering environments. *Proceedings of the IEEE*, 98(4):526–545, 2010.

[36] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE'07)*, pages 37–54. IEEE, 2007.

[37] Deniz Cetinkaya and Alexander Verbraeck. Metamodeling and model transformations in modeling and simulation. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 3043–3053. IEEE, 2011.

[38] Douglas C Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[39] Bart Meyers, Klaas Gadeyne, Bentley Oakes, Matthias Bernaerts, Hans Vangheluwe, and Joachim Denil. A model-driven engineering framework to support the functional safety process. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 619–623. IEEE, 2019.

[40] Xinwen Hu, Yi Zhuang, Zining Cao, Tong Ye, and Mi Li. Modeling and validation for embedded software confidentiality and integrity. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6. IEEE, 2017.

[41] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Elsevier, 2011.

[42] CEA. Eclipse Papyrus Modeling Environment, 2021. https://www.eclipse.org/papyrus/.

[43] Nikolaos S Voros, Wolfgang Mueller, and Colin Snook. An introduction to formal methods. In *UML-B Specification for Proven Embedded Systems Design*, pages 1–20. Springer, 2004.

[44] Raymond M Smullyan. *First-order logic*. Courier Corporation, 1995.

[45] RA Bull and K Segerberg. Basic modal logic. handbook of philosophical logic, v. ii, 1984.

[46] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in event-b. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.

[47] Rodin. Rodin Platform, 2021. https://wiki.event-b.org/.

[48] Alloy. http://alloy.lcs.mit.edu/alloy/, 2019. Accessed: September-2019.

[49] Robert C Armstrong, Ratish J Punnoose, Matthew H Wong, and Jackson R Mayo. Survey of existing tools for formal verification. *SANDIA REPORT SAND2014-20533*, 2014.

[50] Megha Quamara, Gabriel Pedroza, and Brahim Hamid. Introducing a multi-layered model-based design approach towards safety-security co-engineering. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 1163–1164. IEEE, 2021.

[51] Megha Quamara, Gabriel Pedroza, and Brahim Hamid. Multi-layered model-based design approach towards system safety and security co-engineering. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 274–283. IEEE, 2021.

[52] Megha Quamara, Gabriel Pedroza, and Brahim Hamid. Facilitating safety and security co-design and formal analysis in multi-layered system modeling. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1–8. IEEE, 2022.

[53] Megha Quamara, Gabriel Pedroza, and Brahim Hamid. Formal analysis approach for multi-layered system safety and security co-engineering. In *European Dependable Computing Conference*, pages 18–31. Springer, 2022.

[54] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[55] Alan Burns, John McDermid, and J Dobson. On the meaning of safety and security. *The Computer Journal*, 35(1):3–15, 1992.

[56] Erwin Schoitsch, Christoph Schmittner, Zhendong Ma, and Thomas Gruber. The need for safety and cyber-security co-engineering and standardization for highly automated automotive vehicles. In *Advanced Microsystems for Automotive Applications 2015*, pages 251–261. Springer, 2016.

[57] Shirley Radack. The system development life cycle (sdlc). Technical report, National Institute of Standards and Technology, 2009.

[58] M Ann Garrison Darrin and William S Devereux. The agile manifesto, design thinking and systems engineering. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–5. IEEE, 2017.

[59] Georgios Kavallieratos, Sokratis Katsikas, and Vasileios Gkioulos. Cybersecurity and safety co-engineering of cyberphysical systems—a comprehensive survey. *Future Internet*, 12(4):65, 2020.

[60] Elena Lisova, Irfan Šljivo, and Aida Čaušević. Safety and security co-analyses: A systematic literature review. *IEEE Systems Journal*, 13(3):2189–2200, 2018.

[61] Guoqi Xie, Hao Peng, Jing Huang, Renfa Li, and Keqin Li. Energy-efficient functional safety design methodology using asil decomposition for automotive cyber-physical systems. *IEEE Transactions on Reliability*, 2019.

[62] Mohamad Gharib, Paolo Lollini, Andrea Ceccarelli, and Andrea Bondavalli. Engineering functional safety requirements for automotive systems: A cyber-physical-social approach. In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, pages 74–81. IEEE, 2019.

[63] Eun-Young Kang, Dongrui Mu, Li Huang, and Qianqing Lan. Verification and validation of a cyber-physical system in the automotive domain. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 326–333. IEEE, 2017.

[64] Dung Phan, Junxing Yang, Matthew Clark, Radu Grosu, John Schierman, Scott Smolka, and Scott Stoller. A component-based simplex architecture for high-assurance cyber-physical systems. In *2017 17th International Conference on Application of Concurrency to System Design (ACSD)*, pages 49–58. IEEE, 2017.

[65] Alejandro Masrur, Michał Kit, Vladimír Matěna, Tomáš Bureš, and Wolfram Hardt. Component-based design of cyber-physical applications with safety-critical requirements. *Microprocessors and Microsystems*, 42:70–86, 2016.

[66] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Cyber–physical codesign at the functional level for multidomain automotive systems. *IEEE Systems Journal*, 11(4):2949–2959, 2015.

[67] Vladislav Gribov and Holger Voos. A multilayer software architecture for safe autonomous robots. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014.

[68] Quentin Rouland, Brahim Hamid, Jean-Paul Bodeveix, and Mamoun Filali. A formal methods approach to security requirements specification and verification. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 236–241. IEEE, 2019.

[69] Bryan T Carter, Georgios Bakirtzis, Carl R Elks, and Cody H Fleming. A systems approach for eliciting mission-centric security requirements. In *2018 Annual IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE, 2018.

[70] Phu H Nguyen, Koen Yskout, Thomas Heyman, Jacques Klein, Riccardo Scandariato, and Yves Le Traon. Sospa: A system of security design patterns for systematically engineering secure systems. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 246–255. IEEE, 2015.

[71] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Security-aware functional modeling of cyber-physical systems. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4. IEEE, 2015.

[72] N Mead, Venkatesh Viswanathan, and Justin Zhan. Incorporating security requirements engineering into standard lifecycle processes. *International Journal of Security and Its Applications*, 2(4):67–80, 2008.

[73] Nelson H Carreras Guzman, Jin Zhang, Jing Xie, and Jon Arne Glomsrud. A comparative study of stpa-extension and the ufoi-e method for safety and security co-analysis. *Reliability Engineering & System Safety*, 211:107633, 2021.

[74] Xiang-Yu Zhou, Zheng-Jiang Liu, Feng-Wu Wang, and Zhao-Lin Wu. A system-theoretic approach to safety and security co-analysis of autonomous ships. *Ocean Engineering*, 222:108569, 2021.

[75] Christophe Ponsard, Jeremy Grandclaudon, and Philippe Massonet. A goal-driven approach for the joint deployment of safety and security standards for operators of essential services. *Journal of Software: Evolution and Process*, page e2338, 2021.

[76] Christophe Ponsard, Jean-Christophe Deprez, and Robert Darimont. Formalizing security and safety requirements by mapping attack-fault trees on obstacle models with constraint programming semantics. In *2020 IEEE Workshop on Formal Requirements (FORMREQ)*, pages 8–13. IEEE, 2020.

[77] Gabriel Pedroza and Guillaume Mockly. Method and framework for security risks analysis guided by safety criteria. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–8, 2020.

[78] Robert Bramberger, Helmut Martin, Barbara Gallina, and Christoph Schmittner. Co-engineering of safety and security life cycles for engineering of automotive systems. *ACM SIGAda Ada Letters*, 39(2):41–48, 2020.

[79] Morgagni Andrea, Massonet Philippe, Dupont Sbastien, and Grandclaudon Jeremy. Towards incremental safety and security requirements co-certification. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 79–84. IEEE, 2020.

[80] Erik Nilsen Torkildson, Jingyue Li, and Stig Ole Johnsen. Improving security and safety co-analysis of stpa. In *Proceedings of the 29th European Safety and Reliability Conference (ESREL)*, pages 22–26, 2019.

[81] Dominik Püllen, Nikolaos Athanasios Anagnostopoulos, Tolga Arul, and Stefan Katzenbeisser. Security and safety co-engineering of the flexray bus in vehicular networks. In *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, pages 31–37, 2019.

[82] Xinyu Sun. A safety-security integrated analysis approach. In *2018 International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2018)*, pages 575–586. Atlantis Press, 2018.

[83] Behrooz Sangchoolie, Peter Folkesson, and Jonny Vinter. A study of the interplay between safety and security using model-implemented fault injection. In *2018 14th European Dependable Computing Conference (EDCC)*, pages 41–48. IEEE, 2018.

[84] Dajiang Suo, Joshua E Siegel, and Sanjay E Sarma. Merging safety and cybersecurity analysis in product design. *IET Intelligent Transport Systems*, 12(9):1103–1109, 2018.

[85] Inna Vistbakka, Elena Troubitsyna, Tuomas Kuismin, and Timo Latvala. Co-engineering safety and security in industrial control systems: a formal outlook. In *International Workshop on Software Engineering for Resilient Systems*, pages 96–114. Springer, 2017.

[86] William G Temple, Yue Wu, Binbin Chen, and Zbigniew Kalbarczyk. Systems-theoretic likelihood and severity analysis for safety and security co-engineering. In *International Conference on Reliability, Safety and Security of Railway Systems*, pages 51–67. Springer, 2017.

[87] Siwar Kriaa, Ludovic Pietre-Cambacedes, Marc Bouissou, and Yoran Halgand. A survey of approaches combining safety and security for industrial control systems. *Reliability engineering & system safety*, 139:156–178, 2015.

[88] Marc Sango, Jean Godot, Antonio Gonzalez, and Ricardo Ruiz Nolasco. Model-based system, safety and security co-engineering method and toolchain for medical devices design. In *Frontiers in Biomedical Devices*, volume 41037, page V001T07A001. American Society of Mechanical Engineers, 2019.

[89] Alejandra Ruiz, Javier Puelles, Jabier Martinez, Thomas Gruber, Martin Matschnig, and Bernhard Fischer. Preliminary safety and security co-engineering process in the industrial automation sector. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.

[90] Siddhartha Verma, Thomas Gruber, Christoph Schmittner, and P Puschner. Combined approach for safety and security. In *International Conference on Computer Safety, Reliability, and Security*, pages 87–101. Springer, 2019.

[91] Gabriel Pedroza. Towards safety and security co-engineering. In *Security and Safety Interplay of Intelligent Software Systems*, pages 3–16. Springer, 2018.

[92] Reiner Hähnle, Kristofer Johannisson, and Aarne Ranta. An authoring tool for informal and formal requirements specifications. In *International Conference on Fundamental Approaches to Software Engineering*, pages 233–248. Springer, 2002.

[93] Raymond Richards, David Greve, Matthew Wilding, and W Mark Vanfleet. The common criteria, formal methods and acl2. In *ACL2 Workshop*, 2004.

[94] Matthias Jarke, Klaus Pohl, Stephan Jacobs, Janis Bubenko, Petia Assenova, Peter Holm, Benkt Wangler, Colette Rolland, Veronique Plihon, Jean-Roch Schmitt, et al. Requirements engineering: an integrated view of representation, process, and domain. In *European Software Engineering Conference*, pages 100–114. Springer, 1993.

[95] Christoph Stückjürgen. Testing safety properties of cyber-physical systems with non-intrusive fault injection–an industrial case study. In *Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings*, volume 9923, page 105. Springer, 2016.

[96] Yuliya Prokhorova, Elena Troubitsyna, Linas Laibinis, Dubravka Ilić, and Timo Latvala. Formalisation of an industrial approach to monitoring critical data. In *International Conference on Computer Safety, Reliability, and Security*, pages 57–69. Springer, 2013.

[97] Jean-François Pétin, Dominique Evrot, Gérard Morel, and Pascal Lamy. Combining sysml and formal methods for safety requirements verification. In *22nd International Conference on Software & Systems Engineering and their Applications*, page CDROM, 2010.

[98] Mohamad Gharib, Paolo Lollini, Andrea Ceccarelli, and Andrea Bondavalli. Dealing with functional safety requirements for automotive systems: A cyber-physical-social approach. In *International Conference on Critical Information Infrastructures Security*, pages 194–206. Springer, 2017.

[99] Raphael Fonte Boa Trindade, Lukas Bulwahn, and Christoph Ainhauser. Automatically generated safety mechanisms from semi-formal software safety requirements. In *International Conference on Computer Safety, Reliability, and Security*, pages 278–293. Springer, 2014.

[100] Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal methods in system design*, 19(3):291–314, 2001.

[101] Akram Idani, Yves Ledru, Abderrahim Ait Wakrime, Rahma Ben Ayed, and Simon Collart-Dutilleul. Incremental development of a safety critical system combining formal methods and dsmls. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 93–109. Springer, 2019.

[102] Marco Baldi, Alessandro Cucchiarelli, Linda Senigagliesi, Luca Spalazzi, and Francesco Spegni. Parametric and probabilistic model checking of confidentiality in data dispersal algorithms. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pages 476–483. IEEE, 2016.

[103] Kunding Fang, Xiaohong Li, Jianye Hao, and Zhiyong Feng. Formal modeling and verification of security protocols on cloud computing systems based on uml 2.3. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 852–859. IEEE, 2016.

[104] Manuel Cheminod, Alfredo Pironti, and Riccardo Sisto. Formal vulnerability analysis of a security system for remote fieldbus access. *IEEE Transactions on Industrial Informatics*, 7(1):30–40, 2011.

[105] Xiang Ling, Li Wan, and Guoqing Wu. Protection of program integrity based on trusted computing. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 2, pages 278–281. IEEE, 2010.

[106] Paul S Grisham, Charles L Chen, Sarfraz Khurshid, and Dewayne E Perry. Validation of a security model with the alloy analyzer, 2006.

[107] Hamed Orojloo and Mohammad Abdollahi Azgomi. Modelling and evaluation of the security of cyber-physical systems using stochastic petri nets. *IET Cyber-Physical Systems: Theory & Applications*, 4(1):50–57, 2019.

[108] Riham AlTawy and Amr M Youssef. Security tradeoffs in cyber physical systems: A case study survey on implantable medical devices. *IEEE Access*, 4:959–979, 2016.

[109] Preeti Sirohi and Amit Agarwal. Cloud computing data storage security framework relating to data integrity, privacy and trust. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, pages 115–118. IEEE, 2015.

[110] Florian Lugou, Letitia W Li, Ludovic Apvrille, and Rabéa Ameur-Boulifa. Sysml models and model transformation for security. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 331–338. IEEE, 2016.

[111] Ludovic Apvrille, Letitia Li, and Yves Roudier. Model-driven engineering for designing safe and secure embedded systems. In *2016 Architecture-Centric Virtual Integration (ACVI)*, pages 4–7. IEEE, 2016.

[112] Giedre Sabaliauskaite and Aditya P Mathur. Aligning cyber-physical system safety and security. In *Complex Systems Design & Management Asia*, pages 41–53. Springer, 2015.

[113] Mu Sun, Sibin Mohan, Lui Sha, and Carl Gunter. Addressing safety and security contradictions in cyber-physical systems. In *Proceedings of the 1st Workshop on Future Directions in Cyber-Physical Systems Security (CPSSW'09)*. Citeseer, 2009.

[114] Julien Brunel, Laurent Rioux, Stéphane Paul, Anthony Faucogney, and Frédérique Vallée. Formal safety and security assessment of an avionic architecture with alloy. *arXiv preprint arXiv:1405.1113*, 2014.

[115] Siwar Kriaa, Marc Bouissou, Frederic Colin, Yoran Halgand, and Ludovic Pietre-Cambacedes. Safety and security interactions modeling using the bdmp formalism: case study of a pipeline. In *International Conference on Computer Safety, Reliability, and Security*, pages 326–341. Springer, 2014.

[116] Gabriel Pedroza, Ludovic Apvrille, and Daniel Knorreck. Avatar: A sysml environment for the formal verification of safety and security properties. In *2011 11th Annual International Conference on New Technologies of Distributed Systems*, pages 1–10. IEEE, 2011.

[117] Ludovic Piètre-Cambacédès and Marc Bouissou. Modeling safety and security interdependencies with bdmp (boolean logic driven markov processes). In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 2852–2861. IEEE, 2010.

[118] Thomas Novak, Albert Treytl, and Peter Palensky. Common approach to functional safety and system security in building automation and control systems. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1141–1148. IEEE, 2007.

[119] Christophe Ponsard, Gautier Dallons, and Philippe Massonet. Goal-oriented co-engineering of security and safety requirements in cyber-physical systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 334–345. Springer, 2016.

[120] Iso26262. https://www.iso.org/standard/68383.html, 2018. Accessed: September 2020.

[121] Zhang Hong and Xu Lili. Application of software safety analysis using event-b. In *2013 IEEE Seventh International Conference on Software Security and Reliability Companion*, pages 137–144. IEEE, 2013.

[122] Inna Vistbakka and Elena Troubitsyna. Towards a formal approach to analysing security of safety-critical systems. In *2018 14th European Dependable Computing Conference (EDCC)*, pages 182–189. IEEE, 2018.

[123] Elisabeth Uhlemann. Introducing connected vehicles [connected vehicles]. *IEEE vehicular technology magazine*, 10(1):23–31, 2015.

[124] Sagar Behere and Martin Törngren. A functional reference architecture for autonomous driving. *Information and Software Technology*, 73:136–150, 2016.

[125] Morayo Adedjouma, Gabriel Pedroza, and Boutheina Bannour. Representative safety assessment of autonomous vehicle for public transportation. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 124–129. IEEE, 2018.

[126] David Elliott, Walter Keen, and Lei Miao. Recent advances in connected and automated vehicles. *journal of traffic and transportation engineering (English edition)*, 6(2):109–131, 2019.

[127] Ching-Yao Chan. Advancements, prospects, and impacts of automated driving systems. *International journal of transportation science and technology*, 6(3):208–216, 2017.

[128] Puneet Kohli and Anjali Chadha. Enabling pedestrian safety using computer vision techniques: A case study of the 2018 uber inc. self-driving car crash. In *Future of Information and Communication Conference*, pages 261–279. Springer, 2019.

[129] Victoria A Banks, Katherine L Plant, and Neville A Stanton. Driver error or designer error: Using the perceptual cycle model to explore the circumstances surrounding the fatal tesla crash on 7th may 2016. *Safety science*, 108:278–285, 2018.

[130] Bowen Zheng, Hengyi Liang, Qi Zhu, Huafeng Yu, and Chung-Wei Lin. Next generation automotive architecture modeling and exploration for autonomous driving. In *2016 IEEE computer society annual symposium on VLSI (ISVLSI)*, pages 53–58. IEEE, 2016.

[131] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[132] Sen Nie, Ling Liu, and Yuefeng Du. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25:1–16, 2017.

[133] Martin Skoglund, Fredrik Warg, Hans Hansson, and Sasikumar Punnekkat. Synchronisation of an automotive multi-concern development process. In *International Conference on Computer Safety, Reliability, and Security*, pages 63–75. Springer, 2021.

[134] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[135] Quentin Rouland. *Rigorous development of secure architecture within the negative and positive statements: properties, models, analysis and tool support*. PhD thesis, Université Paul Sabatier-Toulouse III, 2021.

[136] Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[137] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. An introduction to uml profiles. *UML and Model Engineering*, 2(6-13):72, 2004.

[138] David Sanderson, Jack C Chaplin, and Svetan Ratchev. A function-behaviour-structure design methodology for adaptive production systems. *The International Journal of Advanced Manufacturing Technology*, 105(9):3731–3742, 2019.

[139] Hollnagel Erik. *FRAM: the functional resonance analysis method: modelling complex socio-technical systems*. CRC Press, 2017.

[140] OMG. Unified Modeling Language 2.5, 2021. https://www.omg.org/spec/UML/2.5/About-UML/.

[141] Felix Hausdorff. *Set theory*, volume 119. American Mathematical Soc., 2005.

[142] Jean-Philippe Monteuuis, Jonathan Petit, Jun Zhang, Houda Labiod, Stefano Mafrica, and Alain Servel. Attacker model for connected and automated vehicles. In *ACM Computer Science in Car Symposium*, 2018.

[143] Matthew B Dwyer, George S Avrunin, and James C Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, pages 411–420, 1999.

[144] Kushal Babel, Vincent Cheval, and Steve Kremer. On the semantics of communications when verifying equivalence properties. *Journal of Computer Security*, 28(1):71–127, 2020.

[145] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computational Logic (TOCL)*, 17(4):1–32, 2016.

[146] Ken Chen. A study on the timeliness property in real-time systems. *Real-Time Systems*, 3(3):247–273, 1991.

[147] E Allen Emerson. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[148] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[149] Christian Cachin. *Entropy measures and unconditional security in cryptography*. PhD thesis, ETH Zurich, 1997.

[150] Clifford W Marshall. Component availability. *IEEE Transactions on Reliability*, 31(2):216–218, 1982.

[151] Hugo A López, Flemming Nielson, and Hanne Riis Nielson. Enforcing availability in failure-aware communicating systems. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 195–211. Springer, 2016.

[152] Marvin Rausand and Arnljot Hoyland. *System reliability theory: models, statistical methods, and applications*, volume 396. John Wiley & Sons, 2003.

[153] Anton Tarasyuk, Elena Troubitsyna, and Linas Laibinis. Towards probabilistic modelling in event-b. In *International Conference on Integrated Formal Methods*, pages 275–289. Springer, 2010.

[154] Husain Aljazzar, Manuel Fischer, Lars Grunske, Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. Safety analysis of an airbag system using probabilistic fmea

and probabilistic counterexamples. In *2009 Sixth International Conference on the Quantitative Evaluation of Systems*, pages 299–308. IEEE, 2009.

[155] Juan Bicarregui, Alvaro Arenas, Benjamin Aziz, Philippe Massonet, and Christophe Ponsard. Towards modelling obligations in event-b. In *International Conference on Abstract State Machines, B and Z*, pages 181–194. Springer, 2008.

[156] Keith Naughton. Ubers fatal crash revealed a self-driving blind spot. *Night Vision, Bloomberg*, 2018.

[157] Jack Stewart. Tesla's autopilot was involved in another deadly car crash. *Wired*, 3:30, 2018.

[158] Dave Gershgorn and D Gershgorn. Instead of hacking self-driving cars, researchers are trying to hack the world they see. *Quartz. Accessed July*, 21, 2017.

[159] A Greenberg. After jeep hack, chrysler recalls 1.4 m vehicles for bug fix. wired, 2015.

[160] ISO15288. Iso/iec/ieee 15288:2015 systems and software engineering — system life cycle processes. `https://www.iso.org/standard/63711.html`, 2015. Accessed: December 2019.

[161] Song Zhiwei, Huang Weiwei, Wu Ning, Wu Xiaojun, Wong Chern Yuen Anthony, Vincensius Billy Saputra, Benjamin Chia Hon Quan, Chen Jian Simon, Zhang Qun, Yao Susu, et al. Map free lane following based on low-cost laser scanner for near future autonomous service vehicle. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 706–711. IEEE, 2015.

[162] Donald Firesmith. Engineering safety-and security-related requirements for software-intensive systems. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2007.

[163] BW Smith. Summary of levels of driving automation for on-road vehicles. *Center for Internet and Society, Stanford Law School*, 1, 2013.

[164] Riccardo Mariani. An overview of autonomous vehicles safety. In *2018 IEEE International Reliability Physics Symposium (IRPS)*, pages 6A–1. IEEE, 2018.

[165] Mark S Young and Neville A Stanton. Back to the future: Brake reaction times for manual and automated vehicles. *Ergonomics*, 50(1):46–58, 2007.

[166] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91), 2015.

[167] Lucian Constantin. Researchers hack tesla model s with remote attack, 2016.

[168] Security challenges for connected and autonomous vehicles. `https://www.baesystems.com/sites/Satellite?blobcol=urldata&blobkey=id&blobtable=MungoBlobs&blobwhere=1578932891373&ssbinary=true`, 2021. Accessed: September 2021.

[169] ISO26262-3Part3. Iso 26262-3:2011 road vehicles — functional safety — part 3: Concept phase. `https://www.iso.org/standard/51358.html`, 2011. Accessed: September 2019.

[170] ISO26262-3Part1. Iso 26262-3:2011 road vehicles — functional safety — part 1: Concept phase. `https://www.iso.org/standard/68385.html`, 2011. Accessed: September 2019.

[171] Stefan Hallerstede. On the purpose of event-b proof obligations. *Formal Aspects of Computing*, 23(1):133–150, 2011.

[172] Uppaal. http://www.uppaal.org/, 2019. Accessed: November-2019.

[173] Sonila Dobi, Mario Gleirscher, Maria Spichkova, and Peter Struss. Model-based hazard and impact analysis. *arXiv preprint arXiv:1512.02759*, 2015.

[174] Lorenzo Bettini. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[175] Thomas Buchmann. Bxtend-a framework for (bidirectional) incremental model transformations. In *MODELSWARD*, pages 336–345, 2018.

[176] Diomidis Spinellis. Notable design patterns for domain-specific languages. *Journal of systems and software*, 56(1):91–99, 2001.

[177] Christoph Schmittner, Zhendong Ma, and Erwin Schoitsch. Combined safety and security development lifecylce. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1408–1415. IEEE, 2015.

[178] Jean-Francois Monin. *Understanding formal methods*. Springer Science & Business Media, 2012.

[179] Michael Hinchey, Jonathan P Bowen, and Christopher A Rouff. Introduction to formal methods. In *Agent Technology from a Formal Perspective*, pages 25–64. Springer, 2006.

[180] Jonathan P Bowen. Z: A formal specification notation. In *Software specification methods*, pages 3–19. Springer, 2001.

[181] Georg Droschl, Walter Kuhn, Gerald Sonneck, and Michael Thuswald. A formal methods case study: Using light-weight vdm for the development of a security system module. In *International Conference on Computer Safety, Reliability, and Security*, pages 187–197. Springer, 2000.

[182] Thomas M Chen, Juan Carlos Sanchez-Aarnoutse, and John Buford. Petri net modeling of cyber-physical attacks on smart grid. *IEEE Transactions on smart grid*, 2(4):741–749, 2011.

[183] Annalisa Bossi, Carla Piazza, and Sabina Rossi. Action refinement in process algebra and security issues. In *International Symposium on Logic-Based Program Synthesis and Transformation*, pages 201–217. Springer, 2007.

[184] Shiu-Kai Chin and Susan Beth Older. *Access control, security, and trust: a logical approach*. Chapman and Hall/CRC, 2010.

[185] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *International Conference on Automated Deduction*, pages 314–328. Springer, 1999.

[186] Musard Balliu, Mads Dam, and Gurvan Le Guernic. Epistemic temporal logic for information flow security. In *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*, pages 1–12, 2011.

[187] Colin Boyd and Wenbo Mao. On a limitation of ban logic. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 240–247. Springer, 1993.

[188] Louise E Moser. A logic of knowledge and belief for reasoning about computer security. In *Proceedings of the Computer Security Foundations Workshop II,*, pages 57–63. IEEE, 1989.

[189] Goal. http://goal.im.ntu.edu.tw/wiki/doku.php, 2019. Accessed: October-2019.

[190] Electrum. http://haslab.github.io/Electrum/, 2019. Accessed: October-2019.

[191] Ron Bell. Introduction to iec 61508. In *ACM International Conference Proceeding Series*, volume 162, pages 3–12, 2006.

[192] Eric Fae, Martial Patra, Stéphane Spohr, and Jérôme Almin. Safety devices in ex applications. are you complying with ex regulations? In *5. PCIC Middle East Conference*, pages 41–47, 2017.

[193] https://www.iso.org/standard/70939.html, 2019. Accessed: September 2020.

[194] Vance Hilderman and Tony Baghi. *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)*. Avionics Communications, 2007.

[195] https://webstore.iec.ch/publication/5527, 1992. Accessed: September 2020.

[196] https://www.nerc.com/pa/Stand/Pages/default.aspx, 2020. Accessed: September 2020.

[197] https://www.perforce.com/resources/qac/how-achieve-en-50128-compliance, 2002. Accessed: September 2019.

[198] https://webstore.iec.ch/publication/6747, 2002. Accessed: September 2019.

[199] https://webstore.iec.ch/publication/6426, 2005. Accessed: September 2019.

[200] https://www.iso.org/standard/69883.html, 2015. Accessed: September 2019.

[201] https://webstore.iec.ch/publication/62427, 2020. Accessed: September 2020.

[202] https://webstore.iec.ch/publication/5532, 2011. Accessed: September 2020.

[203] https://www.iaea.org/resources/safety-standards/search, 2020. Accessed: September 2020.

[204] https://webstore.iec.ch/publication/3795, 2006. Accessed: September 2020.

[205] https://www.iso.org/standard/38421.html, 2006. Accessed: September 2020.

[206] https://webstore.iec.ch/publication/2603, 2020. Accessed: September 2020.

[207] https://www.iso.org/standard/72704.html, 2019. Accessed: September 2020.

[208] https://www.iso.org/standard/72704.html, 2020. Accessed: September 2020.

[209] https://www.iso.org/standard/59752.html, 2016. Accessed: September 2019.

[210] Fda. https://www.fda.gov/medical-devices/medical-device-safety, 1982. Accessed: September 2019.

[211] https://www.iecee.org/dyn/www/f?p=106:49:0::::FSP_STD_ID:1499, 2010. Accessed: September 2019.

[212] https://www.iecee.org/dyn/www/f?p=106:49:0::::FSP_STD_ID:17482, 2010. Accessed: September 2019.

[213] International atomic energy agency (iaea). https://www.iso.org/standard/43464.html, 2019. Accessed: September 2019.

[214] Iso. https://www.iso.org/standard/73906.html, 2019. Accessed: September 2019.

[215] Nist. https://csrc.nist.gov/publications/detail/sp/800-12/rev-1/final, 2019. Accessed: September 2019.

[216] Iso/sae 21434 road vehicles — cybersecurity engineering. https://www.iso.org/standard/70918.html, 2019. Accessed: September 2019.

[217] Spec 42: Aviation industry standards for digital information security. https://publications.airlines.org/CommerceProductDetail.aspx?Product=294, 2019. Accessed: September 2019.

[218] Ed-201 - aeronautical information system security (aiss) framework guidance. https://eshop.eurocae.net/eurocae-documents-and-reports/ed-201/#non-member, 2019. Accessed: September 2019.

[219] Ed-202a - airworthiness security process specification. https://www.sae.org/learn/content/c1949/, 2019. Accessed: September 2019.

[220] Ed-203a - airworthiness security methods and considerations. https://www.eurocae.net/news/posts/2018/june/ed-203a-airworthiness-security-methods-and-considerations/, 2019. Accessed: September 2019.

[221] Ed-204 - information security guidance for continuing airworthiness. https://eshop.eurocae.net/eurocae-documents-and-reports/ed-204/, 2019. Accessed: September 2019.

[222] Arinc 664 - avionics full-duplex switched ethernet (afdx). https://www.nexeya-testandintegration.com/datasheet/A-664_datasheet.pdf, 2019. Accessed: September 2019.

[223] Arinc 811 - commercial aircraft information security concepts of operation and process framework. https://standards.globalspec.com/std/320101/ARINC%20811, 2019. Accessed: September 2019.

[224] Chemical facility anti-terrorism standards (cfats). https://www.cisa.gov/chemical-facility-anti-terrorism-standards, 2019. Accessed: September 2019.

[225] Iec 62351 – cyber security series for the smart grid. https://syc-se.iec.ch/deliveries/cybersecurity-guidelines/security-standards-and-best-practices/iec-62351/, 2019. Accessed: September 2019.

[226] Nerc cip – critical infrastructure protection. https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx, 2019. Accessed: September 2019.

[227] As 7770 – rail cyber security. https://www.rissb.com.au/products/as-7770-rail-cyber-security/, 2019. Accessed: September 2019.

[228] Vde 0831-102 – electric signalling systems for railways, part 102: Protection profile for technical functions in railway signalling. https://www.vde-verlag.de/standards/0800085/din-vde-v-0831-102-vde-v-0831-102-2013-12.html, 2019. Accessed: September 2019.

[229] Vde 0831-104 – electric signalling systems for railways, part 104: It security guideline based on iec 62443. https://www.vde-verlag.de/standards/0800264/din-vde-v-0831-104-vde-v-0831-104-2015-10.html, 2019. Accessed: September 2019.

[230] Directive 2006/42/ec of the european parliament and of the council of 17 may 2006 on machinery, and amending directive 95/16/ec. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32006L0042, 2019. Accessed: September 2019.

[231] Iec 62645: 2019 nuclear power plants - instrumentation, control and electrical power systems - cybersecurity requirements. https://webstore.iec.ch/publication/32904, 2019. Accessed: September 2019.

[232] Ieee 692-2010 ieee standard criteria for security systems for nuclear power generating stations. https://standards.ieee.org/ieee/692/3844/, 2019. Accessed: September 2019.

[233] Nrc rg 5.71 cyber security programs for nuclear facilities. https://www.nrc.gov/docs/ML0903/ML090340159.pdf, 2019. Accessed: September 2019.

[234] Aami tir57: 2016 principles for medical device security - risk management. https://webstore.ansi.org/Standards/AAMI/aamitir572016, 2019. Accessed: September 2019.

[235] Dtsec standard for wireless diabetes device security (dtsec). https://www.diabetestechnology.org/dtsec/DTSec%20Standard.pdf, 2019. Accessed: September 2019.

[236] Guidance for use of mobile devices in diabetes control contexts. https://www.diabetestechnology.org/dtmost/DTMoSt%20Guidance.pdf, 2019. Accessed: September 2019.

[237] European telecommunications standards institute. https://www.etsi.org/, 2019. Accessed: September 2019.

[238] Iso/iec 27001 information security management. https://www.iso.org/isoiec-27001-information-security.html, 2019. Accessed: September 2019.

[239] Iso/iec 27032:2012 information technology — security techniques — guidelines for cybersecurity. https://www.iso.org/standard/44375.html, 2019. Accessed: September 2019.

[240] Iso/iec 15408-1:2009 information technology — security techniques — evaluation criteria for it security — part 1: Introduction and general model. https://www.iso.org/standard/50341.html, 2019. Accessed: September 2019.

[241] Ansi/isa-62443-4-1-2018, security for industrial automation and control systems - part 4-1: Secure product development lifecycle requirements. https://www.isa.org/products/ansi-isa-62443-4-1-2018-security-for-industrial-au, 2019. Accessed: September 2019.

[242] Nist critical infrastructure resources. https://www.nist.gov/cyberframework/critical-infrastructure-resources, 2019. Accessed: September 2019.

[243] Dhs guidelines for completing an accessibility conformance report. https://www.dhs.gov/accessibility-requirements, 2019. Accessed: September 2019.

Glossary

AAMI

Association for the Advancement of Medical Instrumentation

ANSI

American National Standards Institute

ARINC

Aeronautical Radio INC

AS

Australian Standard

ASIL

Automotive Safety Integrity Level

ATA

Air Transport Association

BDMP

Boolean-logic Driven Markov Processes

CDV

Connected-Driving Vehicle

CFATS

Chemical Facility Anti-Terrorism Standards

CIA

Confidentiality, Integrity, Availability

CIAA

Confidentiality, Integrity, Availability, Authenticity

CIP

Critical Infrastructure Protection

CPC

Component-Port-Connector

CPS

Cyber-Physical System

DAG

Directed Acyclic Graph

DHS

Department of Homeland Security

DMS

Data Monitoring System

Domain-Specific Modeling Language (DSML)

A language to create models that are specific to a certain domain.

DSML

see: Domain-Specific Modeling Language (DSML)

DSR

Design Science Research

DSRC

Dedicated Short-Range Communications

DTMoSt

Diabetes Technology Society Mobile Platform Controlling a Diabetes Device Security and Safety Standard

DTSec

Standard for Wireless Diabetes Device Security

EN

European Standard

ETSI

European Telecommunications Standards Institute

EU

European Union

FACT

Failure-Attack-Countermeasure

FDA

Food and Drug Administration

FHA

Functional Hazard Analysis

FIFO

First-In-First-Out

FMEA

Failure Mode and Effects Analysis

FOL

First-Order Logic

Formalization

The process of creating formalized structure.

GPS

Global Positioning System

HARA

Hazards Analysis and Risks Assessment

HAZOP

Hazard and Operability Analysis

IAEA

International Atomic Energy Agency

ICS

Industrial Control System

IEC

International Electro-technical Commission

IEEE

Institute of Electrical and Electronics Engineers

IMD

Implantable Medical Devices

ISA

International Society of Automation

ISO

International Organization for Standardization

LiDAR

Light Detection and Ranging

MDE

see: Model-Driven Engineering (MDE)

MICS

Medical Implant Communication System

Model-Driven Engineering (MDE)

A software development methodology that focuses on creating and exploiting models for abstract representations of the knowledge governing a particular domain.

MTSL

Metric Temporal-Spatial Logic

MTTF

Mean Time To Failures

NERC

North American Electric Reliability Corporation

NIST

National Institute of Standards and Technology

NRC

Nuclear Regulatory Commission

OBU

On-Board Unit

PAS

Publicly Available Specification

PHA

Preliminary Hazard Analysis

PO

see: Proof-Obligation (PO)

Proof-Obligation (PO)

A theorem stating that a certain property must hold in order for a formal specification to be internally consistent.

Properties

Fundamental well-defined notions that are the building blocks upon which high-level requirements can be decomposed and characterized.

PTA

Preliminary Threat Assessment

RG

Regulatory Guides

RSI

Road-Side Infrastructure

SAE

Society of Automotive Engineers

SE

see: System Engineering (SE)

SLR

Systematic Literature Review

SP

Special Publication

SUD

System Under Design

SysML

Systems Modeling Language

System Engineering (SE)

A transdisciplinary approach to enable the successful realization, use, and retirement of engineered systems.

TIR

Technical Information Report

UML

see: Unified Modeling Language (UML)

Unified Modeling Language (UML)

A general-purpose modeling language in the field of software engineering that is intended to provide a standardized way to visualize system design.

V2C

Vehicle-to-Remote Cloud

V2G

Vehicle-to-Grid

V2I

Vehicle-to-Infrastructure

V2V

Vehicle-to-Vehicle

WMTS

Wireless Medical Telemetry Service