Facilitating safety and security co-design and formal analysis in multi-layered system modeling

Megha Quamara, Gabriel Pedroza, Brahim Hamid

Facilitating Safety and Security Co-design and Formal Analysis in Multi-layered System Modeling

Megha Quamara*†, Gabriel Pedroza*, Brahim Hamid†
*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
Email: {megha.quamara, gabriel.pedroza}@cea.fr
†IRIT, Université de Toulouse, CNRS, UT2, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France
Email: brahim.hamid@irit.fr

***Abstract*—The engineering process of systems deployed in critical domains (e.g., automotive) advocates for early-stage integrated analysis of safety and security concerns, given their mutual influence. Specifically, in the design phase, safety and security requirements undergo a transition to the system architectural design across different granular and conceptual representations. However, such an enrichment process is often complex and lacks preliminary guidance to consistently break down high-level system specifications and requirements into intricate architecture and deployment. In particular, engineers require further support to interpret diverse system, safety, and security expertise and facilitate the consistent passage of knowledge pertaining to these disciplines for automated analysis. To this end, we propose an approach to facilitate the joint design and formal analysis of system safety and security concerns. Notably, the approach aims for a three-layered system modeling, integrating mission, functional and component views, and also, reusable libraries of pre-defined safety and security properties, specialize-able across them. We couple the Model-Driven Engineering (MDE) paradigm and Formal Methods (FM) for the hierarchical-precise modeling, formal interpretation, and verification of model views w.r.t. the desired properties. The accompanying tool-chain support for approach instantiation builds upon Papyrus as a modeling framework and Rodin as a formal-based tool for verification. The proposed approach is illustrated via a Connected-Driving Vehicles (CDVs) use case.**

***Index Terms*—safety, security, co-engineering, design, analysis, model-driven engineering, formal methods**

I. Introduction

The increasing automation level and inter-connectivity in modern engineered systems deployed over critical domains and public infrastructure, makes it essential to address both safety and security-related concerns during the System Engineering (SE) process, in light of their mutual influence. For instance, in the automotive domain, the remote access feature offered by the connected cars also exposes attack surfaces to impair safety-critical functionality, e.g., via brakes jamming [1]. Thus, the associated risks must be jointly anticipated and conflicts must be treated during early design stages (e.g., architectural design) to reduce the likelihood/harm and expenses of modifying the overall architecture once implemented [2].

A joint analysis reconciling safety and security expertise is technically challenging in practice due to the model's diversity, resulting from the stakeholders' collaboration, heterogeneous requirements, frameworks/languages involved, etc. Although several engineering approaches for safety [3] and security [4] analyses have been proposed, they often rely on the standalone viewpoint of safety and security disciplines. The mutual impact of the intertwined safety constraints and security exigencies is often not addressed, e.g., messages' latency vs. encryption mechanisms' overhead, as evident in several safety-critical domains like automotive [5] (*issue* **P1**). Particularly, in the design phase, the requirements are broken down from the high-level teleological representations to the detailed technical architecture of the System Under Design (SUD). However, this enrichment process is often complex and lacks guidance for consistent and integrated semantic transfer across different representations, which is particularly true for safety and security (*issue* **P2**). Besides, conducting design-level properties'[1] verification to increase design trustworthiness [6] can be error-prone due to their ambiguous specifications or biases introduced by non-savvy engineer's interpretation (*issue* **P3**). In addition, we observe a lack of methodological tool support for integrated analysis of the system, safety, and security properties; the disciplines of software architecture, security, and safety engineering are yet to be consolidated regarding their methods and tools (*issue* **P4**).

To address the problematics above (P1-P4), we propose a joint design and analysis approach for safety and security co-engineering in the context of a multi-layered system modeling, similar to [7], with the aim to make the complexity of engineered systems potentially addressable. The initial formulation of the approach presented in this paper has been previously published in [8]. Indeed, this work extends those ideas for a three-layered mission-functional-component system representation, mainly focusing on (1) the modeling aspects for system design, (2) the formal analysis of safety and security properties, and (3) the implementation of a tool-chain support prototype. We propose a set of Domain-Specific Modeling Languages (DSMLs) for the three-layered system modeling (*contribution* **C2**) and for safety and security properties' modeling (*contribution* **C1**) to incorporate them across different system layers. Herein, we use the widely-known Unified Modeling Language (UML) and its profiles [9]. Moreover, we

[1]Fundamental well-defined notions that are the building blocks upon which high-level requirements can be decomposed and characterized.

978-1-6654-6297-6/22/$31.00 ©2022 IEEE

use Formal Methods (FMs), namely Event-B [10], to ensure that once deployed, the system's operation conforms to the properties (*contribution* **C3**). As a prerequisite, we define the mapping from the DSML notions to their namesake types in Event-B. Besides, we use formal logic to ensure the well-formedness of the system models and properties' conflicts identification, which is not fully covered in this paper due to lack of space. We also develop a tool-chain support integrating Model-Driven Engineering (MDE) techniques, namely Eclipse Papyrus [11], and a formal-based tool, namely Rodin [12], to conduct verification and spot inconsistencies (*contribution* **C4**). The entire approach is illustrated via a Connected Driving Vehicles (CDVs) use case.

The remaining paper is organized as follows. Section II provides an overview of our proposed approach. Sections III and IV respectively present the modeling of the three-layered system, and safety and security properties. Section V presents the formalization of the modeling languages for properties' verification. Subsequently, Section VI describes the implementation of the tool-chain support. Section VII reviews related works and positions our contribution (C1-C4). Finally, Section VIII concludes this paper including future work.

II. Global Approach

A. Need for Safety and Security Co-design

The development of safety- and security-critical systems requires a dedicated engineering process, which can rely upon high-level modeling to define the system architecture. In a model-driven design, analysis by verification of safety properties (e.g., availability) can be conducted over the system, subsystems, 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 (*i*) a consistent passage of system-related knowledge across different-granularity system views; (*ii*) the integration of properties and their consistency, preservation, and traceability; and (*iii*) coordination and harmonization of collaborative work among different stakeholders and their tasks (see Fig. 1).

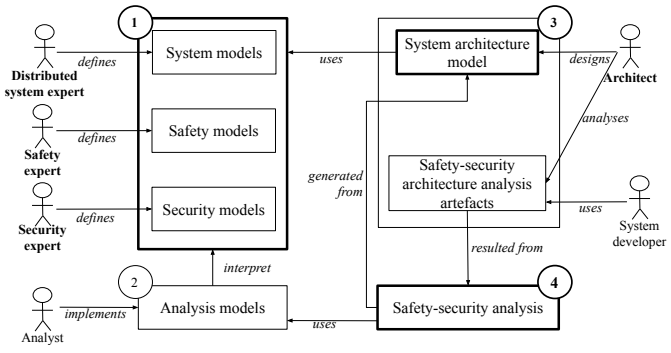

Fig. 1. Our approach in an user-oriented case involving the stakeholders.

However, the aspects above lead to a complex enrichment process that often lacks methodological guidance, modeling language (including semantics), and automated tool support for non-savvy engineers during the integration and verification of safety and security properties. For example, in Fig. 1, different experts, namely *Distributed System Expert*, *Safety Expert*, and *Security Expert*, elaborate a variety of models, and the *Architect* should consider and harmonize their expertise/outcomes to build the software architecture, which is the basis for the *System Developer's* tasks. These stakeholders must reach a consensus regarding an optimal system design, but the collaboration makes the job of the *Analyst* more complex, in particular when perceiving security models at different layers, and jointly analyzing safety- and security-related feared events so as to produce artefacts/solutions with the expected safety and security features at the architecture level.

B. Proposed Approach

To handle the complexity of safety and security co-design and analysis, we propose to model the system and its properties in the context of three-layered system modeling similar to [7]. Accordingly, the system-related aspects can be structured as depicted in Fig. 2. Herein, **Layer 1** - *Mission architecture*, concentrates on the formulation of high-level strategic concerns, i.e., missions of the system, thereby offering a teleological view to capture its overall purpose, **Layer 2** - *Functional architecture*, represents a classical functional decomposition of the system, reflecting the design objectives correlated with its functionality, and **Layer 3** - *Component architecture*, focuses on the detailed technical specification of the target system wherein it is decomposed into a set of components.

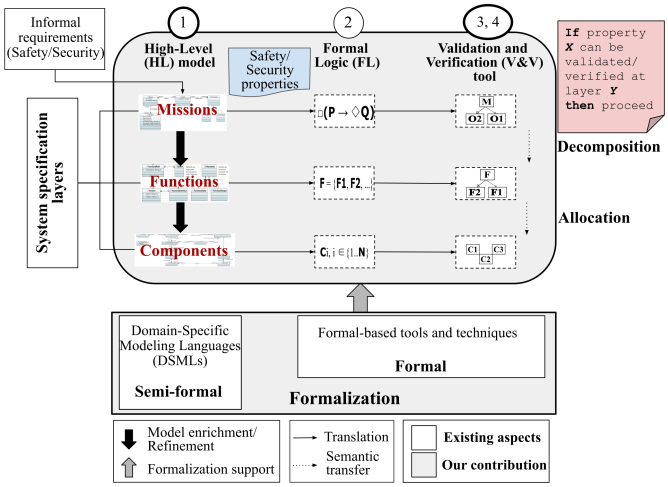

Fig. 2. Overall three-layered modeling approach.

With the three-layered model as its foundation, the approach encompasses the "safety- and security-by-design" principle, disambiguation in properties' specification, and early detection of conflicts between properties in the SE process. This is accomplished via (1) *modeling* of detailed system aspects and safety and security properties, (2) *formalization* of both system

aspects and properties, (3) *integration* of formalized model elements, and (4) *verification* of properties via interpretation into a delegated formal tool. Once formalized, the properties' specifications (instantiated at the three model layers) remain generic enough to be accommodated as reusable libraries.

The approach detailed in the forthcoming sections relies upon MDE and formal techniques as the primary means for addressing the issues P1-P4.

C. Illustrative Use Case

To illustrate the approach, we consider autonomous Connected Driving Vehicles (CDVs) as a safety-security critical application case. We assume CDVs deployed in limited perimeter areas such as shuttle buses in airport terminals. Furthermore, we investigate a realistic converging road plan scenario shown in Fig. 3, and inspired from [13]. In this scenario, two CDVs are in self-driving mode, moving towards the intersection point by following the virtual guides with non-negligible speed. For simplicity, the aspects related to road guides' detection and system-level communication are disentangled.

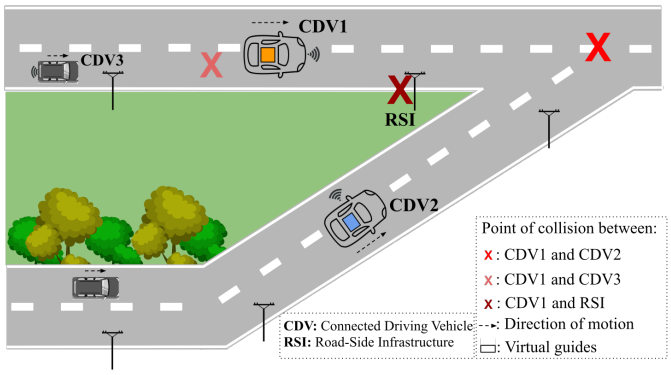

Fig. 3. Use case scenario: Multiple CDVs in a converging road plan.

III. Three-Layered System Modeling

The three-layered system modeling environment comprises one DSML per layer (*contribution* **C2**). This facilitates the separation of concerns across layers, regarding system features and properties, rather than treating them in a single-featured view. The three DSMLs are implemented as UML meta-models and profiles to provide a standardized modeling environment and are detailed in the following sub-sections. An excerpt of the graphical representation of the concepts and relationships is given in Fig. 4.

A. Three-layered System Meta-Model

1) Mission Layer: The *Mission* layer meta-model defines concepts for the engineering of system purposes. Some of the representative elements constituting the *"mission view"* offered by this model are semantically inherited from the state-of-the-art artifacts proposed for rigorously defining the missions [13]–[17].

Recalling the use case scenario in Section II-C, we define a CDV as an atomic **System** that *interacts with* its **Environment**, including **People**, **Assets**, through a physical and logical border. The CDVs are intended to *achieve* a set of critical missions, like collision avoidance [SAFE_M] and grant only authorized actions [SEC_M]. Thus, **Missions** represent the system's high-level purposes or finality related to some behavior, constraint, or feature. Given an operational situation, like in the converging road map in Fig. 3, the mission statement reflects an obligation comprising modal verbs, like *must*, *will*, *should*, and is related to hazardous events, like unintended acceleration, and also security incidents, like unauthorized operation, taken as inputs. **Operations** like braking, access provisioning, and the alike, represent the strategy/implementation steps performed by the CDV for the *accomplishment* of the missions [SAFE_M] and [SEC_M]. *operationInput* and *operationOutput* respectively represent the operation-specific stimuli and response, crossing the border between the system and its environment. *pre/postCondition* and *environmentTriggerCondition* respectively capture the descriptive and prescriptive conditions associated with the operation's application. For example, the decreasing distance between the CDV and its obstacle beyond a certain threshold should trigger the braking operation.

2) Functional Layer: The *Functional* layer meta-model defines concepts related to the engineering of behaviours. To model system functions at detailed-level, some representative elements from the literature [18], [19] are considered to define the *"functional view"* of the model as follows.

Function represents elementary behaviors like perception- or actuation-related, which are performed/provided by the SUD, commonly expressed in the active verb form. *triggerType* represents the type of trigger: *Manual*, *ControlSignal*, or *TemporalConstraint*, i.e., the external stimulus activating a function. Accordingly, a function *caller* can be a system **Operator**, an external **Function**, or a **TemporalConstraint**, respectively. A **FunctionalInterface** represents the interaction point between the functions for their invocation via **Information** exchange. A **FunctionalPath** represents a sequence of functions like obstacle detection → position inference → deceleration, that realize the system's operations via **InformationFlows**, e.g., the dynamics of the obstacle. The first and last functions in the sequence represent the system's boundary functions involving its interaction with the environment.

3) Component Layer: The *Component* layer meta-model defines concepts related to the engineering of components (logical/physical). In the present context, we reuse the Component-Port-Connector (CPC) model presented in [20], with message passing-based communication primitives. It provides a recognized way of visualizing the system's structural and behavioral aspects, constituting the *"component view"* of the model.

A **Component** represents the self-contained computation elements/physical entities like camera, processing unit, brake actuator, constituting the system (i.e., the CDV). Each component *uses* different ports (**Port**) for data exchange. The

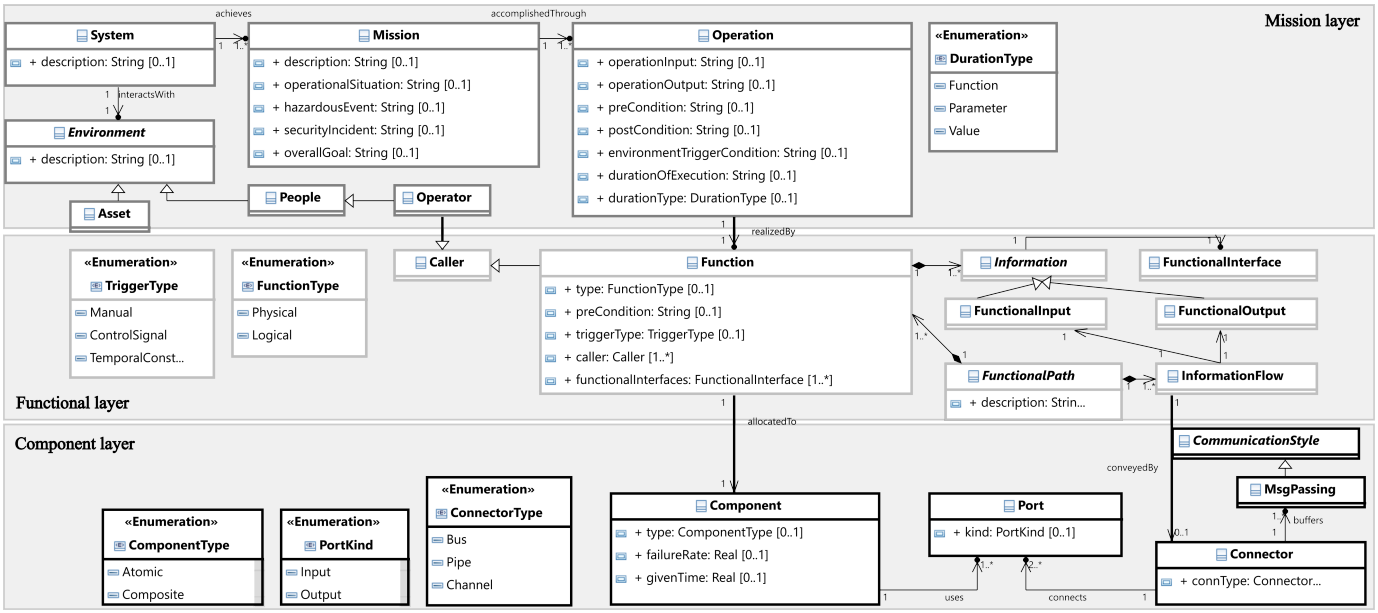

Fig. 4. Three-layered system specification meta-model: Excerpt showing fundamental notions captured.

kind of the port, typed as **PortKind**, constrains the semantics of *Input* and *Output* interactions between the components. The **Connector** establishes a *connection* between two or more components via communication ports. **MsgPassing** represents the **CommunicationStyle** involving message transmission between components via sending/receiving actions. Other possible communication styles defining the communication behavior may include broadcast or multicast.

4) Vertical Integration between the Layered Model: The structural and semantic linking is done by the association between the layered elements, ensuring consistency between the corresponding representations and preserving properties' traceability. For instance, sensing operation at the mission layer is *realized by* obstacle detection function at the functional layer, which in turn, is *allocated to* a sensor at the component layer. Likewise, an information flow between obstacle detection and position inference functions may be *conveyed by* the connector between the sensor and the processing unit components. According to this allocation, inputs/outputs associated with the functional interfaces are implicitly mapped to their respective ports. An operator can manually call upon a function like braking and realize the interaction between the system and its environment. This is a choice that simplifies the structuring and configuration of layers.

B. Three-layered System UML Profiles

The three-layered system UML profiles define all the necessary stereotypes to model the concepts presented in the three DSMLs in a UML environment. For instance, we introduce a set of stereotypes like *«System»*, *«Mission»*, *«Operation»*, *«Environment»*, extending the UML meta-class *Class* to model the mission-layer notions. The *durationType* attribute of an operation is represented by the tagged value *durationType*, whose values are provided by the enumeration *«DurationType»*.

Likewise, the system functionality is modeled by *«Function»* stereotype, extending the UML meta-class *Action* that captures both caller (call behavior action) and callee (opaque actions) functions. The attributes *type* and *triggerType* of a function are defined as a tagged value enumeration to respectively define the type of the function and the trigger associated with it. The type of these attributes is *«TriggerType»* and *«FunctionType»*. The attribute *functionalInterfaces* is modeled with the stereotype *«FunctionalInterface»*, which extends both *Parameter* and *Interface* meta-classes. Likewise, the attribute *caller* is represented via the stereotype *«Caller»*, extending the meta-class *Class*. To model the input and output information 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 via the flow of information across them, we define *«InformationFlow»* stereotype, extending the UML meta-class *Association*.

Finally, 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»*.

IV. Safety and Security Properties

This section presents an overview of the modeling aspects for integrated specification and analysis of safety and security properties (*contribution* **C1**). The full extent of the libraries' contents is not shown due to lack of space.

A. Properties Meta-Model

The three-layered system model includes the *"property view"* and *"property category view"*, shown in Fig. 5, which extend the views presented in Section III-A. They represent the

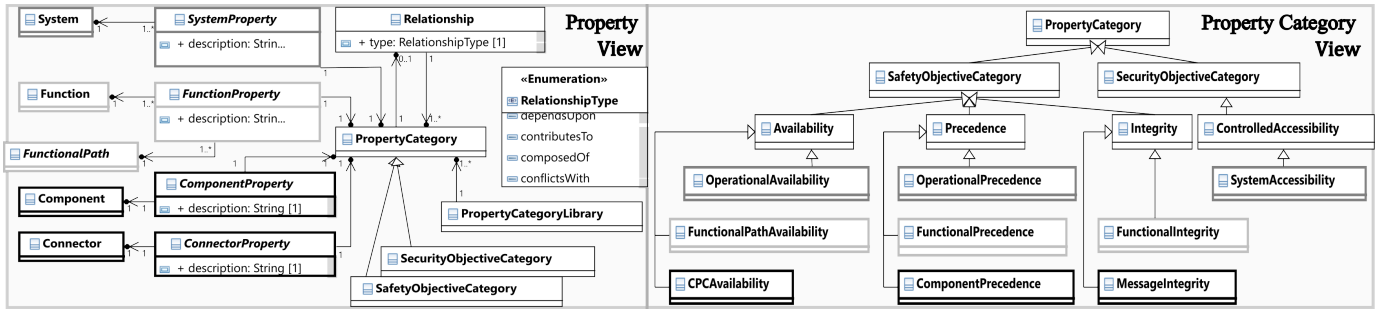

Fig. 5. Safety and security properties' specification meta-model.

reusable model libraries typed as **PropertyCategoryLibrary**, containing high-level properties related to the system, functions/functional path, and components/connectors. **PropertyCategory** represents the classification of safety and security objectives. The libraries are subsequently used as external templates for capturing the objectives as signatures.

1) Safety and Security Properties Specification: We extend some properties from the literature like *Precedence* [21], *Equivalence* [22], by adapting them to the context of our approach. For instance, *Precedence* imposes a partial ordering among the elements, applicable to *functions/functional paths*, *components/connectors*, and whenever two or more elements perform sequentially. Likewise, *Equivalence* between two elements represents an equivalence between their states/attributes. We call these properties *basic ones* since they play an elementary role in defining specific safety and security objectives associated with the target SUD. For instance, *Functional Integrity* ensures the preservation of the information flow between an ordered execution of functions in a functional path.

Regarding safety properties, we extend the notion of *Availability* [16] as *Operational Availability* objective at the mission layer, according to which "a system in a certain operational situation shall allow for the realization of safety-critical operation(s), whenever a hazardous event is detected." Herein, the execution of the operations must be in alignment with the accomplishment of the safety-critical missions, for instance, the realization of sensing operation for obstacle detection in autonomous driving vehicles to avoid crashes. Furthermore, the notion of availability is specialized to *Functional Path Availability* and *Component-Port-Connector (CPC) Availability* objectives at the functional and component layers, respectively. Likewise, regarding security properties, we extend the notion of *Controlled Accessibility* [23] as *System Accessibility* objective at the mission layer, according to which "a system must allow and limit the access of security-critical operation(s) to only authorized entities."

2) Properties Interplay Modeling: Potential inter- or intra-relationships between safety and security objectives may arise within a layer or across layers. Thus, we aim to establish an alignment via links between objectives to support further analysis, including safety and security interplay. We define associations between these objectives like *dependsUpon*, *conflictsWith*, similar to the ones used by the pattern community to define pattern relationships [24]. For example, operational availability, at the mission layer, *depends upon* functional path availability, at the functional layer. System accessibility at the mission layer, *conflicts with* operational availability at the same layer, etc. In essence, the links provide the means to declare dependencies and conflicts between properties.

B. Properties UML Profiles

To model safety and security properties at the three layers ensuring their integration w.r.t. notions therein defined, we introduce stereotypes like *«SystemProperty»*, *«FunctionProperty»*, *«ComponentProperty»*, extending the UML *Class*. The stereotype *«PropertyCategoryLibrary»* models the library of safety and security properties. The specification of corresponding objectives is represented using stereotypes like *«OperationalAvailability»* and *«SystemAccessibility»*, which extend the stereotypes *«SafetyObjectiveCategory»* and *«SecurityObjectiveCategory»*. Finally, the relationship among properties is specified by the *«Relationship»* stereotype, extending the UML *Class*. The actual values of relationship type are defined in the enumeration *«RelationshipType»* as a tagged value.

V. Formalization for Specification and Analysis in Event-B

We consider the interpretation into Event-B of the three-layered system DSMLs shown in Fig. 4, the properties' DSML in Fig. 5, and the properties conflict identification introduced in Section IV-A2 (*contribution* **C3**).

a) System Specification: The three-layered system model is represented in Event-B via a set of *contexts* and *machines*, refined within each layer. The *contexts* are used for formal declaration of DSML structural elements, along with the utility constants for their generic representation. Herein, the key elements like Mission, Function, Component, and enumerations are defined as Event-B *sets*, while their attributes, e.g., caller, type, are represented as *constants*. Likewise, *axioms* capture the relationship between the elements and their attributes, and define the type of the attributes. To model the relation between a set $\mathbb{S}$ (e.g., PortKind) and its sub-sets $s_1, s_2, \ldots, s_n$ represented as constants (e.g., Input, Output), the *partition* operator is used: $partition(\mathbb{S}, s_1, s_2, \ldots, s_n)$.

Based upon this, an excerpt of the instantiation of the three-layered system model concerning the CDV scenario at the

mission layer is presented in Listing 1. Herein, the *extends* keyword is used for inheriting the constants and axioms from *C4MissionUtility* to *C7MissionViewInstance*.

```
CONTEXT C7MissionViewInstance
EXTENDS C4MissionUtility // Context for utility
    constants, e.g., s1, m1, o1
CONSTANTS CDV1, SAFE_M, SEC_M, CDV2, Braking,
    AccessProvisioning
AXIOMS
    CDV1 ∈ System
    SAFE_M ∈ Mission ∧ SEC_M ∈ Mission ∧ Mission = {
        SAFE_M, SEC_M}
    CDV2 ∈ Asset
    Braking ∈ Operation ∧ AccessProvisioning ∈
        Operation ∧ Operation = {Braking,
        AccessProvisioning}
```

Listing 1. Excerpt of Event-B context at mission layer in the CDV scenario.

Furthermore, *machine* specifications formally capture the desired behavioral aspects of the system as model *invariants*. A machine *sees* a context to use its axioms in conjunction as hypotheses in the mathematical proofs. We define *events* in the machines targeting each layer to capture the state transitions associated with the application of operations, function execution, and CPC-based message transmission. The guards in the *when* section, corresponding to the events, restrict the values of the system's attribute variables as enabling conditions for the events. Triggering an event leads to the updating of the variables (e.g., *achieves*, *counter*) via the deterministic assignment ≔. An excerpt of this interpretation for the mission layer concerning the CDV scenario is depicted in Listing 2.

```
MACHINE M1MissionView
SEES C4MissionUtility
VARIABLES operationalSituation, hazardousEvent,
    environmentTriggerCondition, achieves, counter,
    accomplishedThrough, interactsWith
EVENTS HazardousEventPresence
STATUS ordinary
WHEN
    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
```

Listing 2. Excerpt of Event-B machine at mission layer in the CDV scenario.

b) Properties Specification: We interpret the safety and security objectives presented in Section IV-A1 in Event-B as model *invariants* to verify that the defined events satisfy the same during system model verification at different layers.

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

Listing 3. Operational availability objective as an Event-B invariant.

Herein, the *invariants* are represented as the combination of state variables, i.e., the concept attributes, logic symbols like ⇒, ⇔, and quantifiers like ∀, ∃ between propositions. An excerpt of the Event-B interpretation for the *Operational Availability* objective is given in Listing 3.

c) Properties Analysis: 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) supported by the Rodin platform [12]. A PO is generated for every invariant that can be affected by an event. The hypothesis for these POs relies upon the satisfaction of all the invariants, including behavioral, safety and security objectives, and gluing. In addition, the guards restricting the values of the variables are validated before triggering events in every reachable state of the system. The correctness of the instantiated CDV model is dependent on ninety-one POs, which are related to mission accomplishment through the execution of the operation, function, and CPC-centric events (POs' distribution: 32 Variable initialization, 48 System specification, 11 Safety and security invariants).

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

1) **MissionAllocation (INV1):** ∀m ∈ Mission, ∃o ∈ Operation | ((operationalSituation[m] ∧ hazardousEvent[m]) ⇔ (preCondition[o] ∧ environmentTriggerCondition[o])) ⇒ accomplishedThrough[m] = o; holds for all the events.
2) **MissionConsistency (INV2):** ∀m ∈ Mission, (operationalSituation[m] ∧¬ hazardousEvent[m]) ⇒ ¬ overallGoal[m]; holds for all the events.
3) **MissionAccomplishment (THM):** ∀m ∈ Mission, o ∈ Operation | (accomplishedThrough[m] = o) ⇒ (postCondition[o] ⇔ overallGoal[m]); a derived axiom that relies upon INV1 and INV2.

Likewise, the predicate corresponding to the properties' conflict identification, mentioned in Section IV-A2, is shown in the Listing 4 as an Event-B invariant 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 4. Conflict identification invariant for mission layer properties.

For the safety and security missions SAFE_M and SEC_M, respectively, introduced in Section III-A1 for the CDV scenario, the PO corresponding to the invariant in Listing 4 is discharged, depicting a potential conflict between braking and access provisioning operations. This can be due to the lack of privilege for the operator to realize the braking operation manually, *i.e.,* Operator ↦ CDV1 ↦ Braking ∉ *privilege*.

VI. Tool-Chain Support

Fig. 6 depicts the tool-chain support architecture used in our work, comprising the following two primary blocks (*contribution* **C4**):

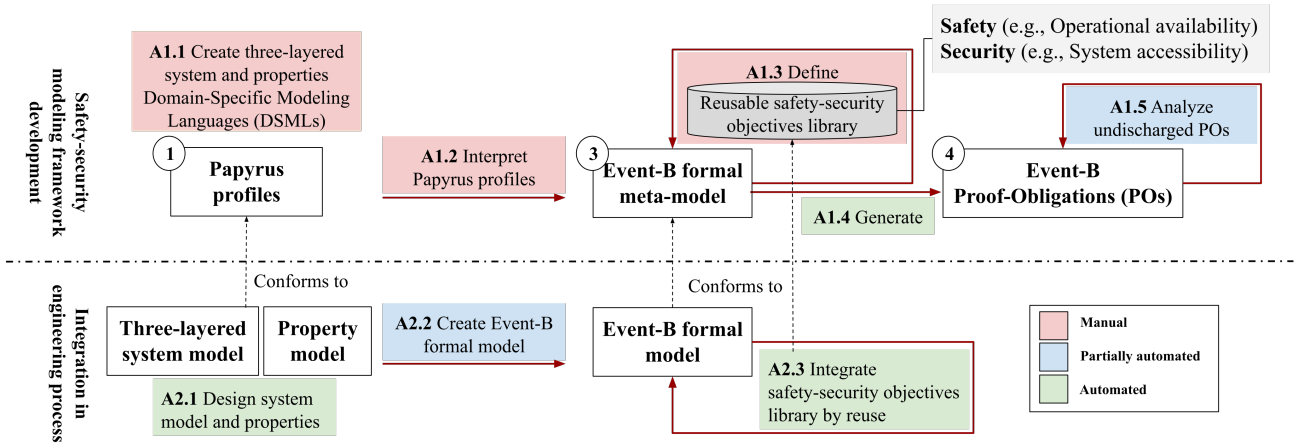

Fig. 6. Overview of the tool architecture developed to support co-engineering of safety and security: Integrated design and formal analysis.

a) Development of Safety and Security Modeling Framework: This block supports the following five activities: (A1.1) involves the creation of the DSML profiles for the three-layered system and safety and security properties in Papyrus [11]. These profiles are further interpreted to their corresponding formal syntaxes, involving the use of the Event-B method [10] and Rodin [12] (A1.2) [**NB:** The work concerning logic-based formalism of the DSMLs as a pivot language facilitating this interpretation, is already conducted; however, not shown in this paper]. Furthermore, reusable libraries of safety (e.g., *Availability*) and security objectives (e.g., *ControlledAccessibility*) are defined as signatures in the context of the formal meta-model (A1.3). Rodin automatically generates POs corresponding to the safety and security objectives invariants (A1.4). Undischarged POs are interactively analyzed to identify and fix the underlying issues (A1.5).

b) Integration into the Engineering Process: This block includes user-oriented features and supports the following three activities: (A2.1) allows the software architect to model a three-layered system and properties conforming to the DSML profiles in (A1.1). Subsequently, (A2.2) enables the generation of Event-B formal models via refinement/interpretation of the already developed meta-models in (A1.2). Finally, safety and security objectives specifications are integrated (A2.3) by reusing and adapting the libraries defined in (A1.3).

VII. Related Work and Positioning

This section dwells on some related works and positions our contributions, emphasizing system modeling frameworks, design and analysis approaches, and formal techniques/tools.

The authors in [15] presented a mission-based process strengthening the links between analysis and architecture design stages, and adapted it to the wave development cycle, thus resulting in less generic than our approach. A functional-layer co-design methodology for automotive systems is proposed in [25] that captures high-level functional specification and explicit partitioning of physics and control using Design Space Exploration (DSE) capabilities. Component-based design techniques and frameworks are presented in the literature concerning safety [26] and security [4] requirements via modeling and analysis of real-time component interactions in the automotive domain. Our work is aligned with these X-by-design approaches and aims, additionally, to provide generic and specializable means to integrate safety and security aspects across different layered representations of the target SUD holistically, irrespective of the design flow (top-down or bottom-up).

The use of DSMLs implemented as UML/SysML meta-models and profiles is widespread for modeling software-based systems incorporating desired safety [27] and security [28] aspects and inconsistencies' detection, often in a standalone manner. Some approaches, e.g., [5], incorporate both safety and security properties, however, adopt a fixed modeling view not covering other layers, nor design steps, like allocation, as in our approach.

Several works demonstrate the applicability of the Event-B method for rigorous analysis of safety [29] and security [30] concerns. However, these works are constrained by the granularity level and concepts chosen for modeling, which imposes requirements specification at the same level. If another modeling layer is needed, further guidance is required to integrate and interpret models into formal specifications.

The multifaceted approach proposed herein strives to reduce the previous gaps across four main axes. First, it is amenable to cover different phases of critical systems' development irrespective of the design flow (top-down or bottom-up) being thus more generic. Second, the properties are defined and integrated, according to the three-layered system model, offering, in addition, sound analysis across different granular representations of the SUD. Third, the layers' independent formal interpretation facilitates engineers to use them in standalone or coupled mode, also adequate for multi-stakeholder usage. Lastly, being technology-agnostic, it provides flexibility for the choice of modeling languages and tool support for verification. These distinctive features are encompassed with the capabilities to specify and model predefined safety and

security properties: 9 safety and security properties are predefined and integrated so far. The approach is extensible and includes guidance to operationalize properties' instantiation and verification. The methodological support for early identification and systematic analysis of their inter-dependencies and conflicts is a key feature.

VIII. Conclusion and Perspectives

We have proposed an approach for co-engineering of safety and security, covering co-design and formal analysis. The work is mainly focused on both system and properties modeling via a three-layered system representation, comprising high-level mission, functional, and detailed component views. The design is conducted at two language levels: 1) semi-formal; relying upon technology-agnostic, generic, and standardized constructs to create system DSMLs, and 2) formal; relying upon formal syntax, semantics, and rules for precise specification of properties and reasoning. A model interpretation was defined to map system and property models into Event-B specifications to conduct automated verification of safety and security objectives' signatures and conflicts solving. The approach is illustrated via a CDV use case from the automotive domain. The approach contributes to the reusability of signatures of safety and security objectives across different design projects. Being generic and mostly automated, it facilitates model interpretation towards other formal frameworks, thus alleviating the complexity of manually integrating formal analysis of safety and security aspects into the design phase.

Among the perspectives to improve the approach, we can mention increasing coverage of safety and security properties libraries, further formal means for conflicts identification and resolution, and formalizing vertical links to propagate verification results across layers. The negative vision of the properties is still to be introduced: fault/failure/hazard and threat models shall corroborate the soundness in safety and security objectives fulfillment versus feared events.

References

[1] D. Dominic, S. Chhawri, R. M. Eustice, D. Ma, and A. Weimerskirch, "Risk assessment for cooperative automated driving," in *Proceedings of the 2nd ACM workshop on cyber-physical systems security and privacy*, 2016, pp. 47–58.

[2] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and internet-of-things systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 9–20, 2017.

[3] M. A. De Miguel, J. F. Briones, J. P. Silva, and A. Alonso, "Integration of safety analysis in model-driven software development," *IET software*, vol. 2, no. 3, pp. 260–280, 2008.

[4] A. Chattopadhyay, K.-Y. Lam, and Y. Tavva, "Autonomous vehicle: Security by design," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[5] G. Pedroza, L. Apvrille, and D. 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*. IEEE, 2011, pp. 1–10.

[6] S. Zafar and R. G. Dromey, "Integrating safety and security requirements into design of an embedded system," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*. IEEE, 2005, pp. 8–pp.

[7] J. J. M. Jiménez and R. Vingerhoeds, "A system engineering approach to predictive maintenance systems: from needs and desires to logical architecture," in *2019 International Symposium on Systems Engineering (ISSE)*. IEEE, 2019, pp. 1–8.

[8] M. Quamara, G. Pedroza, and B. 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)*. IEEE, 2021, pp. 274–283.

[9] L. Fuentes-Fernández and A. Vallecillo-Moreno, "An introduction to UML profiles," *UML and Model Engineering*, vol. 2, no. 6-13, p. 72, 2004.

[10] J.-R. Abrial *et al.*, "Rodin: an open toolset for modelling and reasoning in Event-B," *International journal on software tools for technology transfer*, vol. 12, no. 6, pp. 447–466, 2010.

[11] CEA, "Eclipse Papyrus Modeling Environment," 2021, https://www.eclipse.org/papyrus/.

[12] Rodin, "Rodin Platform," 2021, https://wiki.event-b.org/.

[13] D. Firesmith, "Engineering safety-and security-related requirements for software-intensive systems," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, Tech. Rep., 2007.

[14] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of computer programming*, vol. 20, no. 1-2, pp. 3–50, 1993.

[15] I. Cherfa, N. Belloir, S. Sadou, R. Fleurquin, and D. Bennouar, "Systems of systems: From mission definition to architecture description," *Systems Engineering*, vol. 22, no. 6, pp. 437–454, 2019.

[16] "ISO 26262-1:2018 Road vehicles — Functional safety," https://www.iso.org/standard/43464.html, 2018.

[17] "ISO/IEC 27000:2018 Information technology — Security techniques — Information security management systems," https://www.iso.org/standard/73906.html, 2018.

[18] H. Erik, *FRAM: the functional resonance analysis method: modelling complex socio-technical systems*. CRC Press, 2017.

[19] O. Semeráth, Á. Barta, Á. Horváth, Z. Szatmári, and D. Varró, "Formal validation of domain-specific languages with derived features and well-formedness constraints," *Software & Systems Modeling*, vol. 16, no. 2, pp. 357–392, 2017.

[20] Q. Rouland, B. Hamid, and J. Jaskolka, "Specification, detection, and treatment of STRIDE threats for software components: Modeling, formal methods, and tool support," *Journal of Systems Architecture*, vol. 117, p. 102073, 2021.

[21] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proceedings of the 21st international conference on Software engineering*, 1999, pp. 411–420.

[22] K. Babel, V. Cheval, and S. Kremer, "On the semantics of communications when verifying equivalence properties," *Journal of Computer Security*, vol. 28, no. 1, pp. 71–127, 2020.

[23] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, "A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 106–124, 2009.

[24] P. H. Nguyen, K. Yskout, T. Heyman, J. Klein, R. Scandariato, and Y. 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)*. IEEE, 2015, pp. 246–255.

[25] J. Wan, A. Canedo, and M. A. Al Faruque, "Cyber–physical codesign at the functional level for multidomain automotive systems," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2949–2959, 2015.

[26] A. Masrur, M. Kit, V. Matěna, T. Bureš, and W. Hardt, "Component-based design of cyber-physical applications with safety-critical requirements," *Microprocessors and Microsystems*, vol. 42, pp. 70–86, 2016.

[27] A. Idani, Y. Ledru, A. Ait Wakrime, R. Ben Ayed, and S. Collart-Dutilleul, "Incremental development of a safety critical system combining formal methods and dsmls," in *International Workshop on Formal Methods for Industrial Critical Systems*. Springer, 2019, pp. 93–109.

[28] F. Lugou, L. W. Li, L. Apvrille, and R. Ameur-Boulifa, "Sysml models and model transformation for security," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 2016, pp. 331–338.

[29] Z. Hong and X. Lili, "Application of software safety analysis using event-b," in *2013 IEEE Seventh International Conference on Software Security and Reliability Companion*. IEEE, 2013, pp. 137–144.

[30] I. Vistbakka and E. Troubitsyna, "Towards a formal approach to analysing security of safety-critical systems," in *2018 14th European Dependable Computing Conference (EDCC)*. IEEE, 2018, pp. 182–189.