Model-based Safety and Security Co-Analysis using Component Attack Fault Trees in the Automotive Domain

Victor L. Grechi[1], André L. de Oliveira[2], Barbara Gallina[3], Leonardo Montecchi[4], Rosana T. V. Braga[1]

[1]Inst. de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP), São Carlos – SP – Brasil

[2]Dept. de Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF), Juiz de Fora – MG – Brasil

[3]Mälardalen University (MDU), Västerås – Sweden

[4]Norwegian University of Science and Technology (NTNU), Trondheim – Norway

{victor.grechi,rtvb}@icmc.usp.br, andre.oliveira@ufjf.br, barbara.gallina@mdu.se, leonardo.montecchi@ntnu.no

Abstract. *Cyber-physical systems (CPSs) in critical domains such as self-driven cars and unmanned aerial vehicles involve complex interactions between safety and security concerns. Failures in CPSs such as self-driven cars are caused either by hardware/software component faults or attacks. The identification of hazards, their risks, and root causes is addressed by Safety Engineering, e.g., using Fault Tree Analysis. On the other hand, Security Engineering addresses the identification of asset vulnerabilities, their associated external threats, risks, and causes, using Attack Tree Analysis. Although both disciplines use separate terminology, processes, and tools, they rely on a common system architecture and in the use models such as Component Fault Trees and Attack Trees to support their analyses. In the automotive domain, such analyses should be performed in alignment with guidance defined in assurance standards, e.g., ISO 26262 for functional safety, and ISO 21434 for cybersecurity. However, existing techniques that integrate safety and security models are not fully aligned with the ISO 21434. In this paper, we introduce a novel Component Attack Fault Trees (CAFT) modelling language, built upon Component Fault Trees and ISO 21434 concepts, for integrating safety and security analysis models. Since CAFT was built in alignment with traditional safety and security analysis formalisms and standards, it has the potential to guide engineers in the development of multi-concern analysis model-driven engineering tools. We illustrate the use of our CAFT language to support safety and security co-analysis of an automotive headlamp system.*

1. Introduction

The development of safety-relevant functions in Cyber-Physical System (CPS) domains has been facing an increasing complexity with the introduction of more intelligent and automation features, functions realized by software, and complex interactions between safety and security concerns [Moncada, 2020]. Component-based development, e.g., using UML [OMG, 2017] and SysML [OMG, 2025] modelling languages, has been proven to be effective in handling the inherent complexity of self-driven cars and systems from other safety-critical CPS domains such as unmanned aerial vehicles, by supporting modularity and reuse. Modelling languages like UML and SysML have been

recommended by assurance standards (e.g., ISO 26262 [ISO, 2018] and ISO 21434 [ISO, 2021], and largely adopted in the automotive industry.

The correct operation of CPSs such as self-driven cars is of paramount importance to ensure the safety of passengers and other road users [Kruck *et al*. 2021]. Consequently, a dependable design of such CPSs is required to prevent failures caused by faults as well as cyber-attacks performed by external malicious agents. Random or systematic faults and malicious attacks are considered in the separate disciplines of safety and security, each one with separate skills, teams, terminology, processes, and tools. The increasing interaction between architecture, safety and cybersecurity lifecycles poses challenges to safety/security engineers maintaining consistency between system design and dependability analysis artefacts. Traditionally, this is the case for Fault Tree Analysis (FTA) [Vesely *et al*., 1981], one of the most adopted safety and reliability analysis technique, applied over 50 years and standardized by industry standards e.g., IEC 61508, which are not strongly related to architecture models. With the aim to support modularity into fault trees and connection to architecture models, Component Fault Trees (CFTs) [Kaiser *et al*., 2018; Adler *et al*., 2011; Kaiser *et al*., 2003] and its extension Component Integrated Fault Trees [Domis, 2012] were introduced as a means to associate modular UML/SysML elements such as blocks with CFTs. To achieve a better separation of concerns and supporting the complexity of the analysis of the root causes of multiple hazards, the Hazard-driven realization views for fault tree analysis was introduced into CFTs [Moncada, 2020]. This extension supports the separation of concerns along with hierarchical composition, i.e., subcomponent fault trees, which can be influenced according to reuse possibilities of those sub-CFTs.

The increasing adoption of Information and Communication Technologies (ICT), such as V2X (i.e., vehicle-to-everything – other vehicles, infrastructures, and networks) communication, and Artificial Intelligence in automotive systems raises vehicle cybersecurity concerns, not addressed yet by CFTs. The communication channel could be a vulnerability that attack vectors can exploit to compromise the vehicle system or data confidentiality, integrity, availability, or other cybersecurity properties of interest [Biro *et al*., 2017]. Attackers would necessarily have to be physically close to carry out attacks against vehicles earlier, which is no longer needed with connected cars [Dantas *et al*., 2020]. For instance, the infamous Jeep attack [Wired, 2015] demonstrated that, without appropriate countermeasures, attackers can take control of any vehicle function remotely, such as turning the engine off, steering, and wheel braking. The automotive industry has increasingly become a target of cyber-attacks aimed at compromising the ICT infrastructure of car manufacturers, seeking to steal data or demanding a ransom through *ransomware*, posing significant threats to companies and their customers [Huq, 2024]. The first decade of automotive cybersecurity was marked by an increase in the number of cyber incidents and attacks against OEMs (Original Equipment Manufacturers) and the ecosystem, continuously introducing new attack vectors (e.g., ECUs, APIs, infotainment, telematics, and cloud systems, remote keyless) and methods [Upstream, 2024].

In the automotive domain, the ISO 21434 [ISO, 2021] cybersecurity standard defines the concept of Cybersecurity Assurance Levels, and a detailed workflow for Threat Analysis and Risk Assessment (TARA) without prescribing a specific methodology. The process starts with the ***item*** (*system*) ***definition***, which includes the definition of the item boundary and function, the preliminary architecture, as well as its operational environment, followed by the identification of the assets (Targets Of

Evaluation - TOEs) to be protected and their cybersecurity properties, damage scenarios, threat scenarios, and attack paths. The *threat scenario identification* describes a set of potential actions that may lead to one or more damage scenarios, which specify the adverse consequences (results) of an attack, e.g., using the STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) [Hernan *et al*., 2006] threat analysis method. STRIDE supports engineers to enumerate cybersecurity threats on the asset based on its Data-Flow Diagram (DFD). Then, attack path analysis should be performed to identify the possible attacks that might realize each identified threat scenario. Attack path analysis can be performed using top-down approaches, e.g., attack trees and attack graphs, that deduce attack paths by analysing the different ways a threat scenario could be realized, or bottom-up approaches that build attack paths from the identified vulnerabilities.

Attack trees (ATs) [Schneier, 1999] are similar to fault trees and component fault trees used in safety modelling, facilitating the communication between safety and security engineers. In the same way FTA [Vesely *et al*., 1981] supports engineers in determining how a system can fail, and the likelihood of those failures, ATs allows cyber-security engineers to identify the root causes (threat scenarios and attacks) of potential damage scenarios for an asset (e.g., system, hardware/software component, network), and to define appropriate cybersecurity requirements, which can be specified in the form of Cybersecurity Assurance Levels (CALs) or architectural decisions to reduce security risks. However, the integration of security concerns from attack tree models with safety-related concerns expressed in Component Fault Trees still remains a challenge. In this paper, we introduce the concept of Component Attack Fault Trees (CAFT), built upon CFT formalism and ISO 21434 cybersecurity concepts, for integrating safety and security analysis models. CAFT has the potential to support engineers in the development of multi-concern analysis model-driven engineering tools. We evaluated the feasibility of the proposed CAFT modelling language in a headlamp automotive system extracted from the ISO 21434 cybersecurity standard. This paper is organized as follows. Section 2 provides an overview of the ISO 21434 cybersecurity standard and Component Fault Trees formalism. Section 3 introduces the Component Attack Fault Tree metamodel. Section 4 describes the application of CAFT in an automotive headlamp system. Finally, in Section 5, we present the conclusion and sketches future research directions.

2. Background

In this section, we provide an overview of the ISO 21434 cybersecurity standard (Subsection 2.1) and related work on Component Fault Trees formalism (Subsection 2.2) needed for the reader to understand the contributions of this paper.

2.1 The ISO 21434 Cybersecurity Standard

The ISO 21434 [ISO, 2021] defines the concept of Cybersecurity Assurance Levels and a detailed workflow for Threat Analysis and Risk Assessment (TARA) without prescribing a specific methodology (see Fig. 1). The process starts with the **item** (system) **definition**, which includes the definition of the item boundary and function, the preliminary architecture as well as its operational environment, and relevant assumptions about the item and its environment in an initial data-flow diagram (DFD).

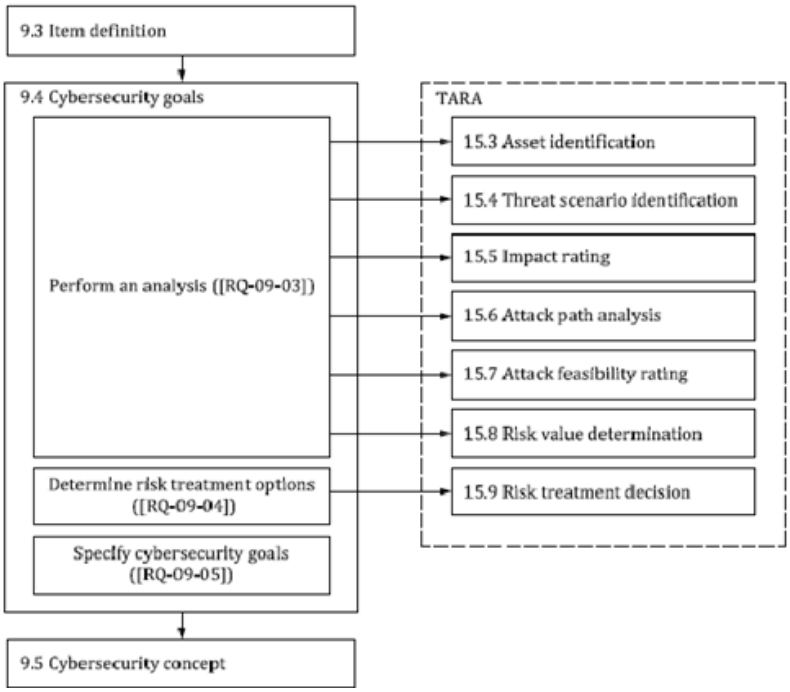

Fig. 1. The ISO 21434 cybersecurity assessment process [ISO 2021].

The DFD includes the most important entities and data-flow of the item and it is the input for asset, damage, and threat scenario identification. After defining the item boundaries and its operational environment, the assets, their associated cybersecurity properties, and damage scenarios are identified. An **asset** is an object that has value, or contributes to a value. An asset could be a system, a component, data, or data communication.

The identified **damage scenarios** are later assessed against their potential adverse consequences for road users in **impact categories** according to their safety, financial, operational, and privacy (S, F, O, P) impact. The **impact** is the degree of magnitude of damage or physical harm from a damage scenario. The **impact rating** of a damage scenario can be classified as severe, major, moderate, or negligible according to the assigned safety, financial, operation, and/or privacy impact rating criteria, derived from the ISO 26262 [ISO, 2018] functional safety standard for road vehicles. The *threat scenario identification* describes a set of potential actions that may lead to one or more damage scenarios, which specify the adverse consequences (results) of an attack, e.g., using the STRIDE threat analysis method. STRIDE supports engineers to enumerate cybersecurity threats on the asset based on the DFD. A threat scenario may lead to the occurrence of multiple damage scenarios. For instance, spoofing of CAN (Controller Area Network) messages for the braking ECU (Electronic Control Unit) of a vehicle leads to the **loss** of **integrity** of the **CAN messages** and to the **loss** of **integrity** of the **braking** function.

ISO 21434 also prescribes performing the *attack path analysis* to identify the possible attacks that might realize each threat scenario. An **attack path** (**attack**) is a set of deliberate actions to realize a threat scenario. Attack path analysis can be performed

using top-down approaches, e.g., attack trees and attack graphs, that deduce attack paths by analysing the different ways a threat scenario could be realized, or bottom-up approaches that build attack paths from the identified vulnerabilities. Attack trees are similar to fault trees used in safety modelling, facilitating the communication between safety and security engineers. Since damage scenarios and threat scenarios have been identified, we'll create an **attack tree** for each damage scenario, which is the root node of the tree. **Threat scenarios** leading to those **damage scenarios** are children of the root node. Depending on the specificity of the threat scenario, it can be split into sub-nodes, or new nodes can be added on a path to an external interface, which may be the entry point for one or more attacks that realize the threat scenario. The total set of identified attack paths are the set of unique paths from the leaf to the root node for all attack trees. Later, each identified attach path should receive an *attack feasibility rating* (high, medium, low, or very low). The attack feasibility rating can be determined based on the **elapsed time**, **specialist expertise**, **knowledge** of the **item** or component, **window of opportunity**, and **equipment** core factors from the attack potential-based approach. The standard also defines an approach to *define cybersecurity goals* for each identified threat scenario in the form of a Cybersecurity Assurance Level (CAL). **Cybersecurity goals** define the applicable risk sharing, avoidance, reduction, or acceptance measures according to the asset **cybersecurity risk**. The ISO 21434 also recommends the production of a cybersecurity case to provide the argument for the cybersecurity of the item or component, supported by work products (evidence).

2.2 Component Fault Trees

A Component Fault Tree (CFT) is a Boolean model associated to system development elements such as components [Hofig *et al*., 2018, Adler *et al*., 2011]. It has the same expressiveness of classic fault trees [Vesely *et al*., 1981]. CFTs are used to model the failure behaviour of safety-critical systems[1]. This failure behaviour, stored into CFTs, is used to document that a system is safe, and can also be used to identify drawbacks in the design of a system. In CFTs, a separate CFT element is related to a component. CFT is a modular approach to system safety analysis that enables hierarchical component-based modelling of failures within a system. CFTs is built upon the principle of cutting out parts of fault trees related to technical components and making them reusable units with input and internal errors and faults, and output failure ports.

CFT relates every output event (failure) to its internally caused basic events (internal faults) and its relevant input faults (i.e., which can be external faults) (see Fig. 2) [Zeller *et al*., 2017]. An external fault can be a failure from another component propagated through input ports of another component, or it could be an attack performed by an external malicious agent aimed at compromising data confidentiality, integrity, or availability. Failures that are visible at the output port of a component are modelled using output failure modes related to a specific output port. Input failure modes are used to model how specific faults propagate from an input port of a compon-

[1] A system in which a failure may lead to catastrophic consequences to people, causing serious injuries or loss of life, damage to the property or to the environment [Knight 2002].

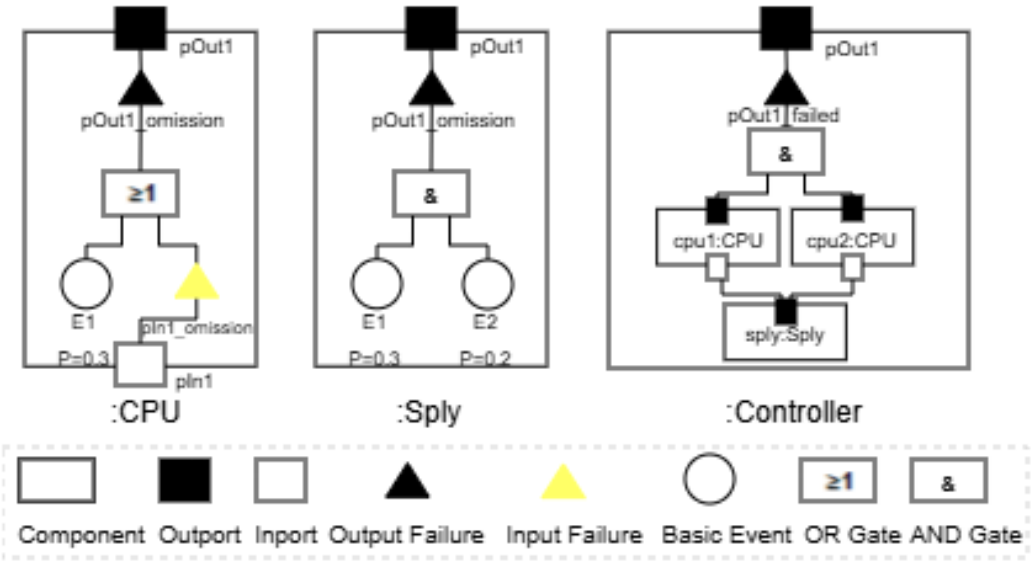

Fig. 2. Example of component fault trees.

ent to its output port. The internal failure behaviour (e.g., errors/faults) that influences the output failure modes is modelled using Boolean gates such as OR (>=1) and AND (&) gates, as well as Basic Events. A basic event is an atomic event that cannot be decomposed, which is the root cause of a top-event, in this case, the output failure mode of a CFT of a component.

CFT is a direct acyclic graph structure defined by edge connections between failures in component output (source) ports and faults in component input (target) ports and mappings from subcomponents to their corresponding CFT models describing their internal fault and error propagations across the system model [Kaiser *et al*., 2018]. This avoids the artificial splitting of common cause errors into multiple repeated events. Instead, it is possible for more than one path to start from the same basic event or sub-graph. Moreover, it is possible to have more than one top event, e.g., an accident when a primary failure coincides with the failure of a countermeasure, but only system unavailability when the same primary failure occurs while the countermeasure is still working. It saves analysts time, since large parts of the failure logic can be shared between two top events.

A CFT can be transformed into a classic fault tree by extracting the input and output failure modes of CFT instances of each system component. CFTs models allow, additionally to the Boolean formula from classic fault trees, to associate specific failure modes to the corresponding ports where those failures can appear. CFT methodology provides the following benefits for the development of safety-critical systems: support for modularity and reusability by encapsulating the failure logic of individual components, reduction of analytical effort by connecting component failures, internal and external faults to top events, facilitating maintainability of the safety analysis model [Hofig et al., 2018]. Similar to classic fault trees, CFT supports qualitative (i.e., minimal cut sets), and quantitative (e.g., probabilistic) analysis of safety-critical systems.

Fig. 2 illustrates an example of CFTs for a controller system, including two redundant CPUs (two instances of the same component type), and one common power supply (*Sply*) component, which would be a repeated event in a traditional fault tree [Kaiser *et al*., 2018]. The Controller is unavailable if both CPUs are in the state ‘failed’.

The inner fault tree of CPU component type illustrates the failure can be caused by some inner basic event 'E1' (the repetition of E1 identifier in several components is not a problem since each component constitutes its own namespace). CPU failure can also be caused by an external failure connected via an input port. Since both causes result in a CPU failure, they are grouped via a 2-input OR gate. The failure logic of power supply is modelled as a separate CFT. An omission of *pOut1* output port from the *power supply* component is caused by simultaneous E1 and E2 internal failures (basic events) (for example, having two redundant batteries) joined by an AND (&) gate. Instead of a single large fault tree, the CFT model comprises small, reusable, and easy-to-review component fault trees.

3 Component Attack Fault Trees

Here, we introduce the concept of Component Attack Fault Trees (CAFT), an extension of Component Integrated Fault Trees [Kaiser *et al*., 2018; Domis, 2012] to support cybersecurity analysis and co-analysis of security concerns on functional safety. CAFT metamodel can be used in the development of tooling to support the integration of fault tree and attack tree analysis in a single model.

The abstract syntax of Component Attack Fault Tree language is defined in its metamodel. CAFT metamodel was built upon Avisienis *et al*. (2004) and ISO 21434 [ISO, 2021] cybersecurity concepts, Fault Tree Analysis [Vesely *et al*., 1981], Open Dependability Exchange metamodel FTA package [Zeller *et al*., 2023], and Attack Tree Analysis [Schneier, 1999] formalisms. Table 1 presents the concepts used in the proposed metamodel and their sources.

Fig. 3 illustrates the proposed Component Attack Fault Tree metamodel in an UML Class Diagram [OMG, 2017]. We used colors to establish a distinction between metamodel classes, highlighted in orange and external classes highlighted in blue. **Component Attack Fault Tree** (**CAFT**) **Component** is a specialized type of **UML4SysML::Component**, which is a set of Boolean functions, each one associated to a component output port, similar to the **CFT** element from the Component Fault Trees [Adler *et al.*, 2011; Kaiser *et al.*, 2003] formalism. Each function maps faults in input ports and/or internal events (e.g., random hardware failures, software errors) to a Boolean gate (e.g., AND, OR) assigned to a failure in an output port. **UML4SysML::Component** may include a set of input and output ports represented by **UML4SysML::Port** element in our metamodel. **UML4SysML::Component** is a subtype of **ISO26262::Item**.

Table 1. Safety, cybersecurity concepts, and their sources.

Concepts	**Source**
Attack, Attack Feasibility, Asset, Cybersecurity Assurance Level, Damage Scenario, Item, Impact, Threat Scenario, Vulnerability, Weakness.	ISO 21434 [ISO, 2021] Attack Tree Analysis [Schneier, 1999]
Dependability Threat, Error, Fault, Failure	[Avzienis et al., 2004]
Component, Port, Connector, ConnectorEnd	Unified Modeling Language (UML) [OMG, 2017] and System Modeling Language (SysML) [OMG, 2025]
Hazard, Assurance Level, Cause, Gate, Probability Distribution, Probability Distribution Parameter	ISO 26262 [ISO, 2018] FTA [Vesely et al., 1981], ODE FTA and Dependability packages [Zeller et al., 2023]

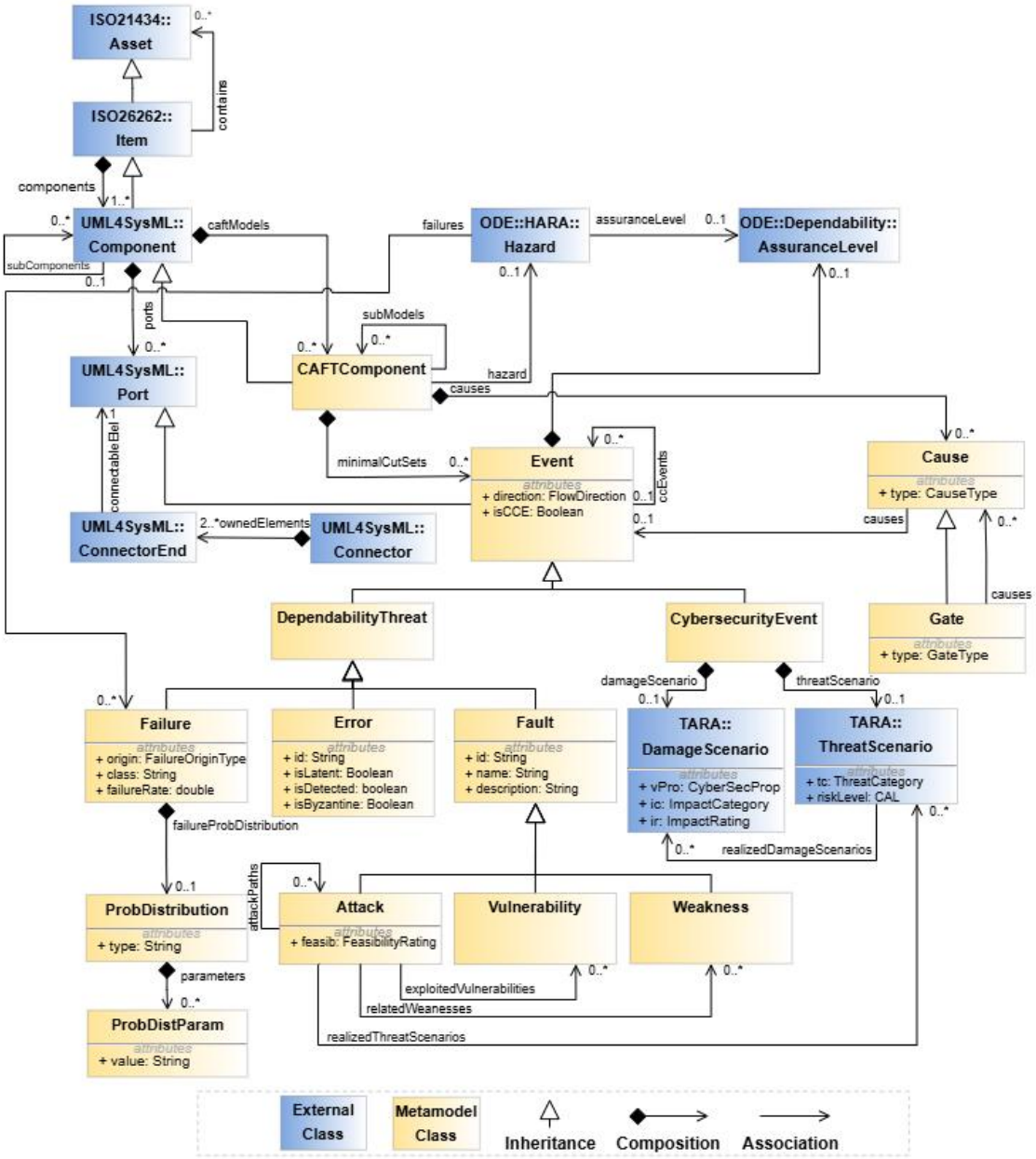

Fig 3. Component Attack Fault Trees metamodel.

An **Item** stands for a **component** or a set of **components** that implement a **function**. An item may contain zero or more assets. An **ISO21434::Asset** is an object that has value or contributes to a value, which is intended to be protected. An **asset** could be a **system**, a **component**, **data**, or **data communication**. A **CAFT Component** consists of: a set of input, internal, and output events (see **Event** subtypes in Fig 3), and a set of **Gates** (**Cause** subtype), connecting output events (**failures**) to their causes (i.e., input and/or internal events); a set of sub-components with their input and output ports and their own CAFT component; and a set of **UML4SysML::Connectors** each one connecting an output port from a **CAFT Component** to an input port from another CAFT component via **UML4SysML::ConnectorEnd**. A component **Connector** can have more than one target. An **Event** in a CAFT Component can be a **DependabilityThreat** or a **Cybersecurity** event. An **Event** has a flow direction (in, out, in-out), related to the port direction it is triggered, and it could be a common cause

leading to the occurrence of a Hazard, denoted by its 'isCCE' Boolean attribute. isCCE is set to true if this event occurs in two or more components of the system at the same time. An Event may also refer to other events (ccEvents) that can be caused by the common cause event.

A **Hazard** is a condition or combination of conditions that is a potential source of harm or damage. The **causes** of a **Hazard** can be one or a combination of component output **Failures** defined by the association relationship between **ODE::HARA::Hazard** and **Failure** UML classes. An **Assurance Level** (e.g., an Automotive Safety Integrity Level – ASIL) can be assigned to a **Hazard** to mitigate its effects on the overall system safety. Safety standards such as the automotive ISO 26262 prescribe a set of safety objectives, guidance to perform verification and validation activities, such as different software testing criteria to be addressed according to the degree of rigor of the assigned assurance level. For instance, achieving a stringent **ASIL D** assigned to a system function **(item**) requires all paths test coverage. On the other hand, code inspection is sufficient to verify an **ASIL A** function.

A **DependabilityThreat** can be: a service **Failure**, which is an event that occurs when the delivered service deviates from correct the correct service; an **Error**, which is a deviation of at least one (or more) external state of the system from the correct service state; or a **Fault**, i.e., the cause of an error, which can be **internal** or **external** to the system. A service can fail because it does not comply with its specification, or because its specification did not adequately describe the system function. The deviations from the correct service may assume different forms, called **service failure modes (Failure** class in the metamodel), and are ranked according to their severity. Service failure modes can be classified, from the failure domain viewpoint, as **content**, **timing**, and **content** & **timing** (**service provision**) failures [Avizienis *et al*., 2004]. **Service provision** failure modes include omission and commission failures. An **omission** or **halt** failure occurs when the service is not delivered at the service interface (i.e., output port). A **commission** failure happens due to a service delivered at the service interface when not required. A **content** or **value** (**too low** or **too high**) failure occurs if the content of the information delivered at the service interface deviates from the implementing system function. Finally, a **timing** failure happens when the time of arrival or the duration of the information delivered at the service interface, i.e., the timing that the service is delivered, deviates from the implementing system function. A **timing** failure may be early or late, depending on whether the service is delivered too **early** or **late** than required. **Omission**, **commission**, **value**, **early**, and **late** failure modes are encapsulated in the **FailureOriginType** enumeration type stated as the 'origin' attribute from **Failure** class in our metamodel. Since we cannot determine the causes of a hardware failure, safety engineers can assign their **failure rate** based on the chosen **Probability Distribution**, e.g., Weibull-Poisson, and its parameters. This is useful to calculate the probability of occurrence of a Hazard in quantitative dependability assessment.

Vulnerability is an internal fault that enables an **external fault** to **harm** the system. Vulnerability is necessary for an external fault, e.g., an **Attack**, to cause an error and possibly subsequent failures in the system. In most cases, a **Fault** first causes an **Error** in the **service state** of a **component** that is a part of the internal state of the system, and the internal state is not immediately affected [Avizienis *et al.*, 2004]. An **Error** is part of the total state of the system that may lead to a subsequent service failure. It is important to highlight that many errors do not reach the system external

state, causing a failure. A **Fault** is **active** when it causes an error, otherwise it is **dormant**. **Vulnerability** is a **dormant fault**. **Vulnerability** is a **weakness** (of the **asset**) that can be exploited as part of an **Attack**. A **Weakness** is a **fault** (subtype) or **characteristic** (from the **asset**) that can lead to undesirable behaviour.

An **Attack** is a set of **deliberate actions** to realize a **Threat Scenario** (i.e., a subtype of **CybersecurityEvent**) aiming to compromise an asset cybersecurity property, e.g., system/data confidentiality, integrity, availability. An **Attack** can be decomposed into a set of **attack steps** (denoted by the self-relationship labelled as 'attack paths' from the **Attack** class), which must be performed to execute the attack. **Attack steps** are a set of actions needed to perform an **attack**. Each identified **Attack** receives a **feasibility rating** (high, medium, low, or very low). The attack feasibility rating can be determined based on Attack Potential, Common Vulnerability Scoring System, Attack Vector, or other approaches. In this paper, we considered the Attack Potential-based approach, defined in the ISO/IEC 18045 [ISO/IEC, 2008] security standard, which determines the **attack feasibility rating** based on **elapsed time**, **specialist expertise**, **knowledge** of the **item** or **component**, **window of opportunity**, and **equipment** parameters. Each **attack potential parameter** can receive one of a set of possible **values**. Attack feasibility ranges from very low, low, high to very high, encapsulated into **FeasibilityRating** enumeration type in our metamodel.

CybersecurityEvent is an **Event** subtype used to refer to **Threat Scenario** and **Damage Scenario** metaclasses from Threat Analysis and Risk Assessment (TARA) in a node of a system **Component Attack Fault Tree**, keeping conformance with Avizienis *et al*. (2004) and ISO 21434 dependability and security concepts. A **Threat Scenario** is a potential cause of violation of **cybersecurity properties** of one or more **assets** that realizes a **Damage Scenario**. STRIDE [Hernan *et al.*, 2006] technique can support cybersecurity engineers to enumerate **cybersecurity threats** on the **asset** based on a DFD. Each **STRIDE** threat category is associated with the violation of a particular **cybersecurity property**. A **spoofing** threat where attackers pretend to be legitimate entities violates **authenticity**. **Tampering** threats, which attackers modify data in transit or in a data store violate **integrity**. A **repudiation** threat where attackers perform actions that cannot be traced back to them violates **non-repudiation**. **Information disclosure** threats related to attackers accessing and publishing data to unauthorized users violate **confidentiality/privacy**. **Denial of service** happens when attackers interrupt a system legitimate operation, violating **availability**. **Elevation of privilege** threats correspond to attackers performing unauthorized actions, violating **authorization**. STRIDE threats are encapsulated in a **Threat Category** enumeration type stated as an attribute from **Threat Scenario**. Each identified threat scenario has an associated cybersecurity risk, denoted by a **Cybersecurity Assurance Level** (**CAL**) enumeration type value stated as the risk level attribute. CAL enumeration type, derived from the ISO 21434 cybersecurity standard, has four possible values, being 1 the less and 4 the most stringent cybersecurity assurance level. The CAL assigned as the risk level of a threat scenario associated with a given asset determine the degree of rigor for the cybersecurity validation process. According to the ISO 21434 cybersecurity standard, addressing the most stringent CAL 4 requires performing penetration testing.

Damage Scenario is a violation of an asset's **cybersecurity property**. The possible violated cybersecurity properties: confidentiality, integrity, availability, authenticity, authorization, non-repudiation, and freshness are grouped in the **CyberSecProp** enumeration type. Each identified **damage scenario** is later assessed

against their potential adverse consequences for road users in **impact categories** according to their **safety**, **financial**, **operational**, and/or **privacy** (S, F, O, P) impact, also grouped in an enumeration type. The **impact rating** of a **damage scenario** can be classified as severe, major, moderate, or negligible values, which are grouped in the **ImpactRating** enumeration type.

The risk posed by a **Threat Scenario** is determined based on the highest **impact** assigned to an associated **damage scenario** and the **highest feasibility rating value** assigned to an associated **attack path** leading to the threat scenario. For instance, consider a threat scenario ***t1*** leading to two different damage scenarios (**d1** and **d2**) with **severe** and **moderate** safety impacts, respectively, and two associated **attack paths** (***a1*** and ***a2***) with low and high feasibility ratings, respectively. To determine the **risk value** of the ***t1*** threat scenario, we consider the **severe** impact assigned to the ***d1*** damage scenario and the **high** feasibility rating assigned to the ***a2*** attack path. The risk value of a threat scenario is determined based on a risk matrix, comprising **damage impact** and **attack feasibility rating**. The ISO 21434 cybersecurity standard specifies five **risk levels** that can be assigned to a **threat scenario** according to its **impact** and **attack feasibility rating**. A risk matrix can support assigning a risk level to a threat scenario. The matrix is almost symmetric except for a lower risk value for negligible impact and a high attack feasibility rating. The definition of what constitutes an acceptable risk (i.e., **As Low As Reasonably Practicable - ALARP**) depends on the project. We can consider risk values between 1 and 2 as acceptable levels in most cases.

A **CAFT Component** has a set of **Causes** and **minimal cut sets**, i.e., a set of minimal combinations of events that may lead a Component/System to fail. **Cause** element is used to refer to various types of **Events** (**Dependability Threats** and **Cybersecurity Events**) represented in a Component Attack Fault Tree. Each **Cause** refers to a single **Event**. **Gate** is a **Cause** subtype, which is a Boolean logic connector (AND, OR) used to *chain hierarchies* of **Causes** in a **CAFT Component**. The root **CAFT Component** may refer to a **Hazard**. This allows deriving a secured-informed system Component Attack Fault Tree model that describes the combinations of root causes (i.e., **basic events** that could be **Faults** and **Attacks**), leading to the occurrence of the **top event**, i.e., a safety-related **Hazard**.

4 Evaluation

In this section, we illustrate the application of concepts from the proposed CAFT metamodel and Component Fault Trees [Kaiser et al., 2018] visual notation to support the co-analysis of safety and cybersecurity for an automotive headlamp system, as outlined in the ISO 21434 [ISO, 2021] standard, in a system Component Attack Fault Tree model. We conducted safety and security analysis using CAFT in compliance with the ISO 26262 functional safety and ISO 21434 cybersecurity standards.

The **headlamp system** (see the SysML Block Diagram from Fig. 4) turns on/off the headlamp in accordance with the switch at the driver's demand. If the headlamp is in high-beam mode, the system switches the headlamp automatically to the low-beam mode when an oncoming vehicle is detected by the camera. It also returns the headlamp automatically to the high-beam mode if the oncoming vehicle is no longer detected. This is a safety-critical system since a failure leading to harm (e.g., an accident) may occur if the lamp is unexpectedly turned off overnight. The **Body Control ECU** sends

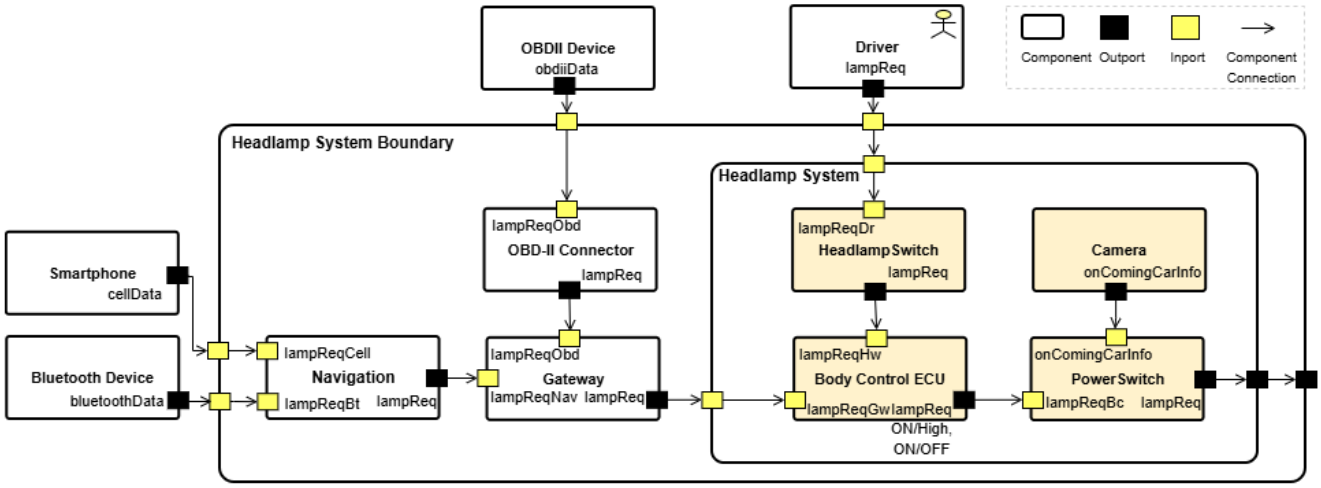

Fig. 4. Headlamp system architecture (adapted from [ISO, 2021]).

signals to the **Power Switch** actuator through the CAN bus. These signals are requests from the driver, coming from the **Headlamp Switch**, to turn the headlamp on or off.

Camera ECU detects oncoming vehicles and sends signals to the **Power Switch** actuator. The signals sent by the **Body Control** and **Camera** ECUs are used to automatically switch the headlamp from high-beam to low-beam mode and vice versa. The left upper sides of Fig. 4 contain external components that may access the headlamp system. **Gateway** controls access from external components such as **Navigation ECU** to the headlamp system. Gateway receives signals from an **OBD-II Connector**. Navigation has two interfaces, namely **Cellular** and **Bluetooth**. Both interfaces may access the **Body Control ECU** to, e.g., carry out software updates. **OBD-II**, **Cellular**, and **Bluetooth** interfaces are located outside the **headlamp system** boundary. **Safety Analysis**: We considered the *unintended turning-off of the headlamp during night driving* (**HZD1**) hazard. We classified the risk posed by **HZD1** as **ASIL C** since its occurrence may lead to life threatening injuries (**S3** severity), it has medium probability of occurrence (**E3** exposure), and it is difficult to control (**C3** controllability).

The following faults and failures may lead to the occurrence of **HZD1**: i) **FM1**: Body Control ECU is faulty leading to not turning the headlamp on upon the driver's request. This may happen if this fault is triggered by a failure of erroneous type (i.e., omission). ii) **FM2**: The logical channel between **Body Control ECU** and **Power Switch** is faulty, leading to not turning the headlamp on upon driver's request. We derived the following safety goal (**SG1**) to mitigate **HZD1** effects: avoiding potential harm after the occurrence of an erroneous failure (**FM1**), and the system shall transition to a safe state after the occurrence of FM2. Addressing this safety goal improves both safety and availability of the headlamp system. Fig. 5a illustrates the **Component Attack Fault Tree** model for the **Headlamp** system, its components (Body Switch ECU, Camera, Power Switch Actuator), and external interfaces. This model describes the root causes for the omission of the lamp request signal leading to the occurrence of **HZD1**. An **omission** of **oncoming car information** from **Camera**, or an **omission** of the **lamp request signal** sent by the **Body Control ECU**, or an **internal fault** of **Headlamp Switch** (see Fig. 5b) are the root causes of the occurrence of *unintended turning-off of the headlamp during night driving*.

Cybersecurity Analysis: We consider the headlamp system an asset once a failure to turn on the headlamp during night driving may lead to an accident. We performed such an analysis in the OBD-II Connector and Navigation ECU external interfaces. Fig. 5c illustrates the Component Attack Fault Tree for the OBD-II Connector. An internal fault in the OBD-II Connector may propagate to an incorrect value of its output, which propagates to the Body Control ECU, changing the lamp request signal. The violation of integrity of the lampReqObd input port is a damage scenario that could be exploited by a malicious external agent. A spoofing of the lamp request signal encapsulated in the lampReqObd input port is a threat scenario that can realize the identified damage scenario. Finally, the occurrence of a CAPEC – 411 Mali-

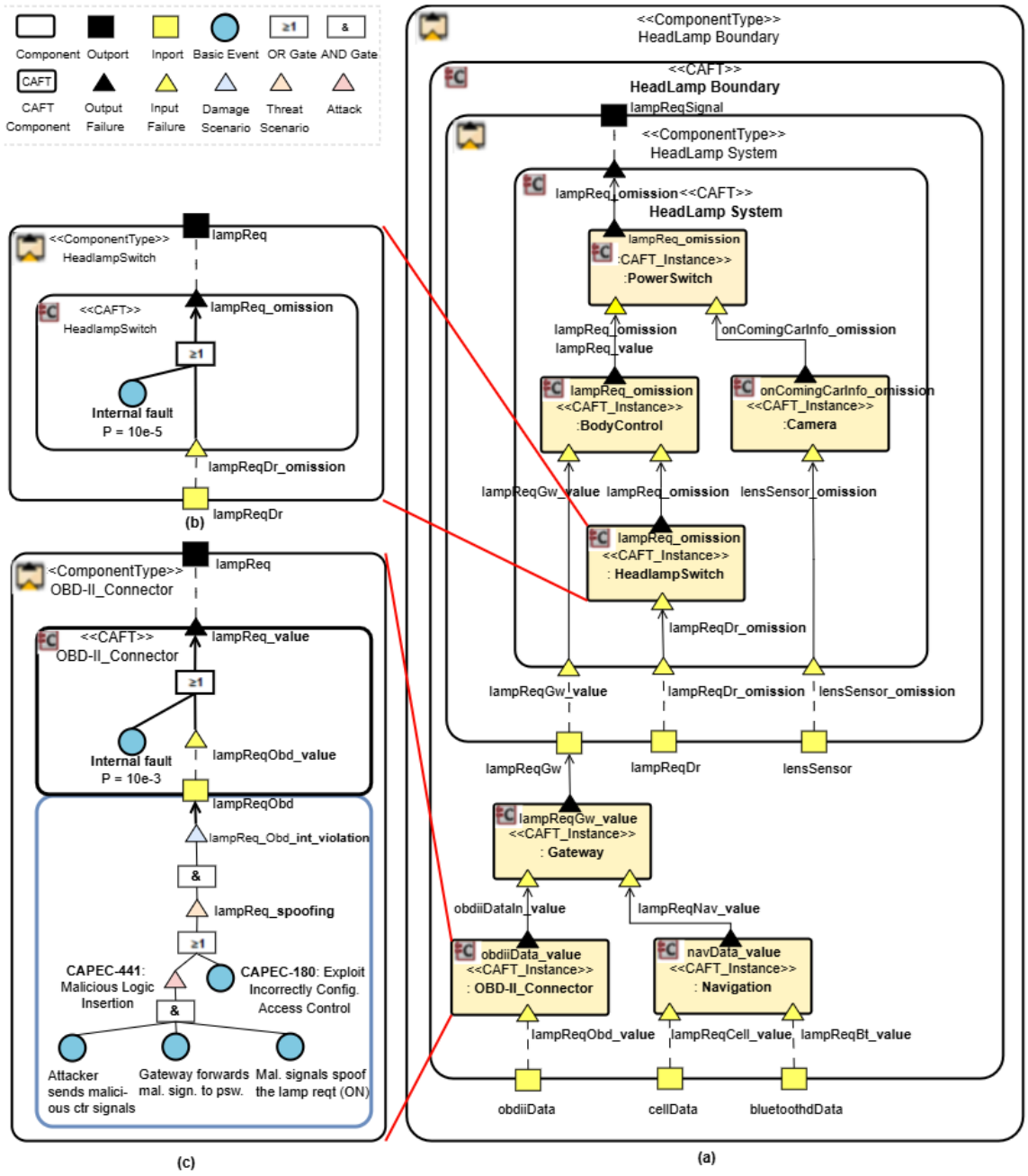

Fig. 5. Headlmap system component attack fault tree analysis. a) Headlamp CAFT model. b) Headlamp Switch CAFT model with failure and its potential causes. c) OBD-II Connector CAFT with failures, internal faults, and cybersecurity events.

cious Logic Insertion [2] or CAPEC 180 - Exploiting incorrect configuration access control attack may lead to the realization of the spoofing threat scenario, violating the integrity of the lamp request signal. Malicious Logic Insertion attack is further decomposed into the following steps: i) sending a malicious turning off lamp request signal, ii) Gateway forwarding the signal to the Body Control ECU, and iii) spoofing the lamp request signal (from ON to OFF). A Firewall between the Gateway ECU and the Headlamp System can be introduced to mitigate malicious incoming data flows from Bluetooth, Cellular, or OBD-II Connector external interfaces.

5 Conclusion

In this paper, we introduced a Component Attack Fault Trees metamodel, an extension of the Component Fault Trees [Kaiser *et al*., 2018; Adler *et al*., 2011; Kaiser *et al*., 2003] formalism to support cybersecurity and safety co-analysis in the automotive domain. The security extensions for CFTs were defined based on the ISO 21434 and Avizienis *et al*. (2004) dependable and secure computing concepts. The CAFT metamodel can be used to guide safety and cybersecurity engineers in conducting an analysis of safety and security concerns within a single system Component Attack Fault Tree model, as it integrates the Component Fault Trees and Attack Trees formalisms.

The CAFT metamodel can be used to guide the development of tooling to enable the co-analysis of safety and security concerns in compliance with the automotive ISO 26262 and ISO 21434 assurance standards. We evaluated the feasibility of our CAFT metamodel in supporting safety and security co-analysis in the automotive Headlamp system extracted from the ISO 21434 cybersecurity standard. As future work, we intend to formally define a visual notation for Component Attack Fault Tree security concepts. We will conduct an experimental study to evaluate the expressiveness of the CAFT metamodel with security experts from both academia and industry. Finally, we intend to develop extensions to the PolarySys CHESS framework to support Component Attack Fault Tree analysis in UML and SysML models.

Acknowledgments

We thank University of São Paulo and CAPES PRO-DEFESA grant number V3084362P for funding this research.

References

Adler, R., Domis, D., Höfig, K., Kemmann, S., Kuhn, T., Schwinn, J.P., Trapp, M.: Integration of component fault trees into the UML. In: Dingel, J., Solberg, A. (eds.) Models in Software Engineering, 312–327p. Springer, New York (2011)

Avizienis, A., Laprie, J. C., Randell, B., Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. **Dependable and Secure Computing, IEEE Transactions on**. 1. 11 - 33. doi: http://doi.org/10.1109/TDSC.2004.2.

Biro, M., Mashkoor, A., Sametinger, J., Seker, R.: Software safety and security risk mitigation in cyber-physical systems. **IEEE Software**. 35 (1), 24–29, 2017.

[2] https://capec.mitre.org/

Dantas, Y. G., Nigam, V., Ruess, H. Security Engineering for ISO 21434. Fortiss GmbH White Paper, Munich, Germany, 2020.

Domis, D. Integrating Fault Tree Analysis and Component-Oriented Model-Based Design of Embedded Systems. Dissertation Technische Universität Kaiserslautern, 2012.

Hernan S., Lambert S., Ostwald T., Shostack A. Threat Modeling - Uncover Security Design Flaws Using the STRIDE Approach, MSDN Mag. 2006.

Hofig, K., Joanni, A., Zeller, M., Montrone, F., Rothfelder, M., Amarnath, R. and Munk, P., and Nordmann, A. Model-Based Reliability and Safety: Reducing the Complexity of Safety Analyses Using Component Fault Trees. 2018 Annual Reliability and Maintainability Symposium (RAMS), 2018, pp. 1-7.

Huq, N. Automotive Cyber Security - Emerging Risks and New Case Study Insights. *ATZ Electron Worldw* **19**, 14–19 (2024). doi: https://doi.org/10.1007/s38314-024-1890-0.

ISO/SAE 21434: Road Vehicles – Cybersecurity Engineering, ISO/TC 22/SC 32, 2021.

ISO 26262. Road Vehicles— Functional Safety for Road Vehicles, ISO/TC 22/SC32, 2018.

ISO/IEC. ISO/IEC 18045:2008 – Information technology – Security techniques – Methodology for IT security evaluation, 2008.

Kaiser, B., Schneider, D., Adler, R., Domis, D., Mohrle, F., Berres, A., Zeller, M., Hofig, K., Rothfelder, M. Advances in Component Fault Trees. **Safety and Reliability – Safe Societies in a Changing World** – Haugen et al. (Eds), 2018, Taylor & Francis Group, London.

Kaiser, B., Liggesmeyer, P., Mackel, O. A New Component Concept for Fault Trees. In Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software, Canberra, 2003.

Knight, J. C. Safety Critical Systems: Challenges and Directions. International Conference on Software Engineering (ICSE), Orlando, Florida, USA, 2002.

Kruck, B., Munk, P., Angermeier, D. 2021. Safe and secure: Mutually supporting safety and security analyses with model-based suggestions. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 172-181). IEEE.

Moncada, D. S. V. Hazard-driven realization view for Component Fault Trees. **Software and Systems Modeling** (2020) 19: 1465-1481p. doi: http://doi.org/10.1007/s10270-020-00792-8

OMG. The Unified Modeling Language specification version 2.5.1. Object Management Group. 2017. Online: https://www.omg.org/spec/UML/.

OMG. The System Modeling Language specification version 2.0. Object Management Group. 2025. Online: https://www.omg.org/spec/SysML.

Schneier, B. Modeling security threats. Dr. Dobb's J. 24 (12), 1999.

Upstream. Global Automotive Cybersecurity Report. Upstream Sec. Ltd., 2024. Online: https://info.upstream.auto/hubfs/Security_Report/Security_Report_2024/Upstream_2024_Global_Automotive_Cybersecurity_Report.pdf.

Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F. Fault Tree Handbook. US Nuclear Regulatory Commission, 1981.

Wired. Hackers remotely kill a Jeep on the highway with me in it, 2015. Online: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

Zeller, M., Sorokos, I., Reich, J., Adler, R., and Schneider, D. (2023). Open Dependability Exchange Metamodel: A Format to Exchange Safety Information. 2023 Annual Reliability and Maintainability Symposium (RAMS), Orlando, FL, USA, 2023, pp. 1-7. doi: http://doi.org/10.1109/RAMS51473.2023.10088190.

Zeller, M., Hofig, K., Schwinn, J. P. ArChes – Automatic generation of component fault trees from continuous function charts. 2017 IEEE 15th International Conference on Industrial Informatics (INDIN) (2017): 577-582p.