embeddedworld2022 Exhibition&Conference ... it's a smarter world

Integrating Security and Safety with Systems Engineering: a Model-Based Approach

Matthias Bergler
Technische Hochschule Nürnberg
Nürnberg, Germany
matthias.bergler@th-nuernberg.de

Juha-Pekka Tolvanen
MetaCase
Jyväskylä, Finland
jpt@metacase.com

Ramin Tavakoli Kolagari
Technische Hochschule Nürnberg
Nürnberg, Germany
ramin.tavakolikolagari@th-nuernberg.de

***Abstract*—Development of reliable systems requires that safety and security concerns are acknowledged during system development. Adding them afterwards is risky as many concerns are missed if not elicited together with the system requirements. Unfortunately, languages for systems engineering, like SysML, typically ignore security and safety forcing development teams to split the work into different formats, languages and tools without easy collaboration, with limited traceability, separate versioning and restricted use of automation that tools can provide. We present a model-based approach targeting automotive that integrates safety and security aspects with other system development practices. This is achieved via a comprehensive domain-specific modeling language that is extendable by language users. We demonstrate this approach with practical examples on how security and safety concerns are recognized along with traditional system design and analysis phases.**

Keywords—model-based development; security, safety, domain-specific language; system engineering; software engineering

I. Introduction

The development of reliable systems requires the consideration of safety aspects during system development. Adding them afterwards is difficult and error-prone, as important stakeholder concerns can be lost. Unfortunately, systems engineering languages such as SysML [12] typically ignore safety and security, forcing development teams to divide the work into distinct subsections: Safety and security are defined in different formalisms and languages, without clear traceability, collaborative work and use of other automations that tools can provide.

We present a model-based approach targeting the automotive domain that integrates safety and security aspects into the rest of system development practices. This is achieved through a modeling language that combines systems engineering concepts with those of safety and security. Thus, modeling is not only about specifying a system with blocks, signal connections, etc., but at the language level, security-relevant aspects such as attacks and vulnerabilities as well as safety-relevant aspects such as hazardous events and feature flaws are considered too. Safety and security are thus directly present in the development language as first-class citizens, just like the other aspects relevant for the specification and modeling of systems.

Our work on integrated language development is based on collaborative work over several years with automotive companies, researchers, and tool providers [2]. We demonstrate the approach with practical examples on how security and safety concerns are recognized with system design — covering traceability and automatically conducting several relevant safety-related methods such as ISO26262 [4], FTA [8] and FMEA [13], and security-related methods such as vulnerability scores [3]. The integrated model-based approach helps to ensure that safety and security is done for the intended system currently being developed and assists engineers with automation that tools can provide, covering editing, tracing, versioning and various analysis and reporting.

We start by presenting our solution by describing how the modeling languages are defined and integrated with existing languages. This is followed by case studies demonstrating the approach with practical examples. First, we describe how an existing automotive language is extended with security modeling: covering both attacks from the social engineering side and from the technical side. This is followed by describing language extensions for safety following strictly the functional safety as defined in ISO26262. For both language extensions we show examples on their use. We conclude by sharing our experiences on defining languages for safety and security.

II. Defining Modeling Languages and Their Integration

In this section, we give a brief overview of the possibilities of describing modeling languages (Section II-A) and the underlying modeling concepts that have been considered in the creation of safety/security aspects in the automotive context in the present language (Section II-B and II-C). The section concludes with a summary of the major benefits of an integrated modeling approach (Section II-D).

A. *Metamodelling: description of the allowed structure and syntax*

All computer-based languages, including programming languages and modeling languages, are defined with some formalism. For model-based development the languages are typically defined by metamodels [6]. A metamodel describes the allowed structure and syntax with which we can create models. Consider Fig. 1 illustrating a very small fragment of the

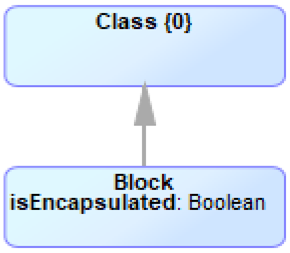


Fig. 1. Small part of SysML metamodel

metamodel taken from SysML: A ‘Block’ has one Boolean property called ‘isEncapsulated’ to define if a block is treated as a black box element [12]. Based on this metamodel, the ‘Block’ does not have any other properties as those that are defined by its supertype ‘Class’— and these ‘Class’-specific properties are omitted from this small metamodel.

Language definition normally also includes constraints to ensure that the created models are syntactically complete, correct and consistent. Some of these constraints are expressed directly in the metamodel whereas others are defined with additional scripts or constraint definitions. An example of the former is that an attribute is mandatory and an example of the latter that an element must have a certain number of connections.

Definition of a general-purpose modeling language, like SysML or UML, contains hundreds of elements like ‘Class’ and ‘Block’ and their properties like ‘isEncapsulated’. Although they can be considered as large languages neither of them covers

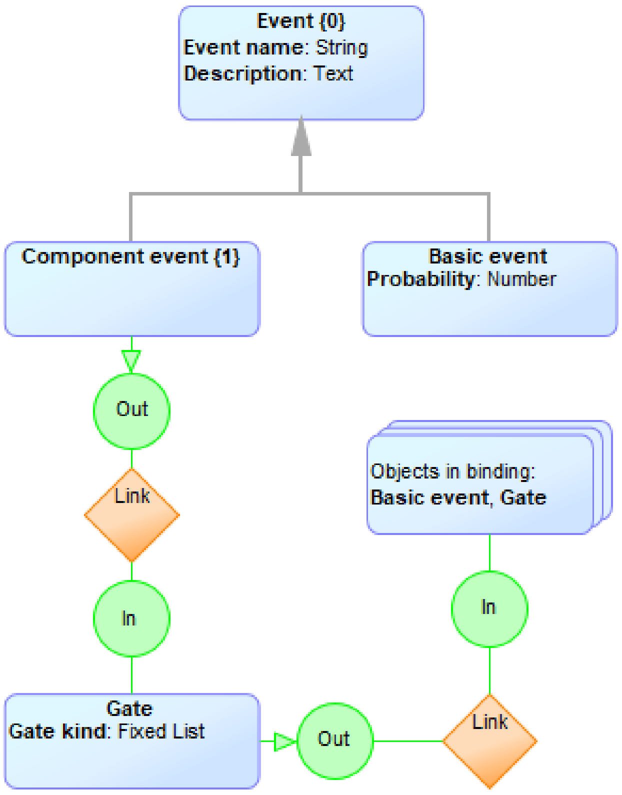


Fig. 2. Complete metamodel of Fault Tree modeling language

aspects relevant to safety or security. For example, a common practice for safety engineering is defining fault trees and performing Fault Tree Analysis (FTA) [8]. Fault trees is also a language having own metamodel that is independent and thus unrelated with SysML. Fig. 2 illustrates a metamodel of the fault tree. This metamodel contains just a few elements but it still defines the complete FTA language: There are two types of events: a ‘Component event’ acting as a root for the fault tree and a number of ‘Basic events’ causing an error with some probability. Both elements have ‘Event name’ and ‘Description’ properties that are inherited from the abstract supertype ‘Event’. In addition, ‘Basic event’ has a ‘Probability’ property to indicate how often the event can occur. Third element is a ‘Gate’ to indicate relationships among these three elements. Gate has a property ‘Gate kind’ with values like AND, OR for Boolean logic. Finally, a ‘Link’ element allows relating a root component event to a ‘Gate’ and those can be again linked with other gates or basic events.

In addition to the definition of a metamodel and related constraints, modeling languages also have notation for humans to create and read the models. Once notational symbols are given, a modern tool can provide the desired modeling support. Fig. 3 shows an example of fault tree modeling in a tool based on the metamodel defined in Fig. 2. This modeling editor, and related functionality, is provided automatically based on the language definition. In the example model ‘flat burns down’ is a component event that is the single root element, and there are two different types of gates with five different basic events. As basic events have failure probabilities, the tool can subsequently calculate the probability of the component failure.

If fault tree modeling should support other aspects, like link to other subtrees of faults the metamodel could be extended. Therefore, tools giving access to the metamodel (as in Fig. 2) enable users to extend the modeling capabilities directly without any lock or waiting for features from the tool vendor. This gives engineers tooling that can fit exactly to the needs and gives control over the ways systems are specified. Engineers using the tool can dictate, not tool providers.

In our example, the metamodel of fault tree and SysML are, however, disconnected which is understandable as capturing risks and safety concerns were not targeted when SysML was defined. However, we can identify connections between these two languages. For example, the root component of the fault tree most likely should refer to at least one of the blocks in SysML so that failure of the system block could be measured. Also, if working in automotive systems and on their functional safety the modeling support should be able to express ‘Item’, ‘Hazard’, ‘HazardousEvents’ that are all defined in the safety standard [4]. In the modeling approach presented in this paper, this desired connection between systems and safety as well as security modeling is fully supported via the language definition.

We present next language extensions that cover security and safety concerns related to automotive systems using EAST-ADL language that is made for developing automotive systems [2]. The principle of language extension and the practices described here are not limited to any particular modeling language though.

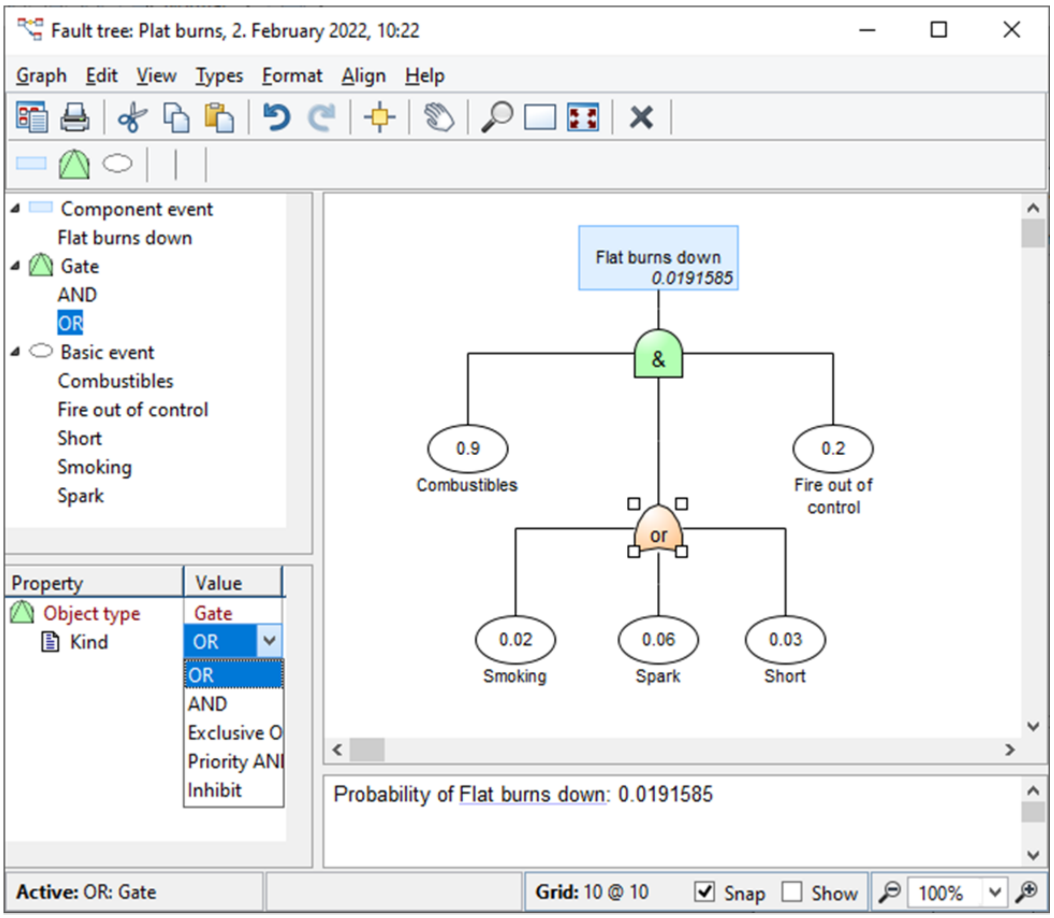


Fig. 3. Fault tree modeling based on the metamodel defined in Fig. 2.

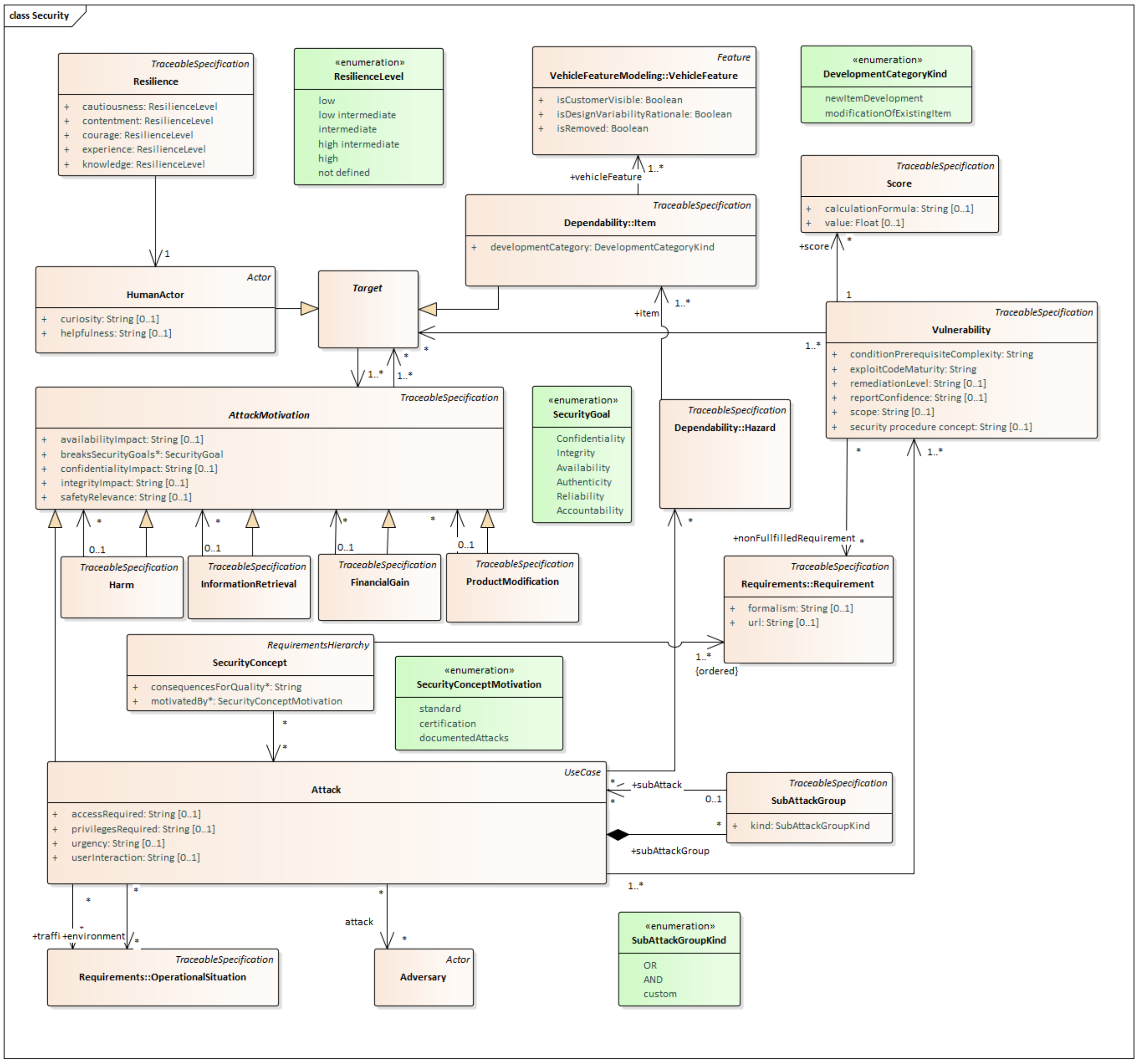
B. *Security related capabilities of this approach*

Security Abstraction Model (SAM) targets representing security-related properties in automotive software systems. This modeling language enables a security analysis of attack vectors in the automotive sector and allows for an in-depth risk analysis. With SAM both potential attacks and countermeasures against these attacks can be specified. This allows the connection of security management and model-based systems engineering on an abstract description level according to the principles of automotive security modeling. SAM was defined based on security requirements from common industrial scenarios. It aims to be a solution for representing attack vectors on vehicles and provide a thorough security modeling for the automotive industry.

SAM has a close link to the system architecture description via the modeling entity 'Item' being part of the architecture model of EAST-ADL, an Architecture Description Language, aligned with the AUTOSAR automotive standard [2]. Item refers to a number of features of an automotive system. SAM tries to present all important criteria of the attack vectors, from the adversary's motivation up to the security breach. This allows a system to be represented from a security perspective in the early software development phase. In addition to the 'Attack motivations', SAM also describes all intrinsic and temporal characteristics of an 'Attack', e.g., effects on the security objectives (confidentiality, availability, integrity, etc.), the complexity of the attack, the affected object and the 'Vulnerability'. The latest version of SAM address also social engineering attacks [1].

Fig. 4 describes the metamodel of SAM. SAM acts as an extension to the EAST-ADL, because the EAST-ADL addresses relevant aspects of automotive systems (being a major requirement for security modeling that is not offered by languages like SysML [12] or AADL [15], which only offers feature modeling); especially the features of a vehicle of any kind. In addition, the EAST-ADL speaks directly about functional safety and ISO 26262 in its Dependability Model. SAM identifies the same 'Item', 'Requirement' and 'Hazard' from architecture and dependability modeling and relates them to 'Attacks' and 'Security Concepts'.

Although SAM is developed as part of the EAST-ADL, it is not necessarily bound to EAST-ADL. SAM as a metamodel is independent of other languages but for connectivity links to 'Item' and 'Requirement' of the EAST-ADL. In addition, SAM can also be used independently of the rest of the system model to provide an overview of safety critical system parts before or at the beginning of the system engineering process. For more information about SAM please see [16][17].



class Security
TraceableSpecification Resilience
+ cautiousness: ResilienceLevel
+ contentment: ResilienceLevel
+ courage: ResilienceLevel
+ experience: ResilienceLevel
+ knowledge: ResilienceLevel
«enumeration» ResilienceLevel
low
low intermediate
intermediate
high intermediate
high
not defined
Feature VehicleFeatureModeling::VehicleFeature
+ isCustomerVisible: Boolean
+ isDesignVariabilityRationale: Boolean
+ isRemoved: Boolean
«enumeration» DevelopmentCategoryKind
newItemDevelopment
modificationOfExistingItem
+vehicleFeature
1..*
TraceableSpecification Dependability::Item
+ developmentCategory: DevelopmentCategoryKind
TraceableSpecification Score
+ calculationFormula: String [0..1]
+ value: Float [0..1]
+score
1
Actor HumanActor
+ curiosity: String [0..1]
+ helpfulness: String [0..1]
Target
+item
1..*
TraceableSpecification Vulnerability
+ conditionPrerequisiteComplexity: String
+ exploitCodeMaturity: String
+ remediationLevel: String [0..1]
+ reportConfidence: String [0..1]
+ scope: String [0..1]
+ security procedure concept: String [0..1]
TraceableSpecification AttackMotivation
+ availabilityImpact: String [0..1]
+ breaksSecurityGoals*: SecurityGoal
+ confidentialityImpact: String [0..1]
+ integrityImpact: String [0..1]
+ safetyRelevance: String [0..1]
«enumeration» SecurityGoal
Confidentiality
Integrity
Availability
Authenticity
Reliability
Accountability
TraceableSpecification Dependability::Hazard
+nonFullfilledRequirement
TraceableSpecification Harm
TraceableSpecification InformationRetrieval
TraceableSpecification FinancialGain
TraceableSpecification ProductModification
0..1
TraceableSpecification Requirements::Requirement
+ formalism: String [0..1]
+ url: String [0..1]
RequirementsHierarchy SecurityConcept
+ consequencesForQuality*: String
+ motivatedBy*: SecurityConceptMotivation
«enumeration» SecurityConceptMotivation
standard
certification
documentedAttacks
1..*
{ordered}
UseCase Attack
+ accessRequired: String [0..1]
+ privilegesRequired: String [0..1]
+ urgency: String [0..1]
+ userInteraction: String [0..1]
+subAttack
TraceableSpecification SubAttackGroup
+ kind: SubAttackGroupKind
+subAttackGroup
+traffi +environment
attack
TraceableSpecification Requirements::OperationalSituation
Actor Adversary
«enumeration» SubAttackGroupKind
OR
AND
custom

Fig. 4. Security metamodel. View Online at https://www.in.th-nuernberg.de/professors/BerglerMa/SAM/

Models created according to SAM permit calculating a vulnerability score based on the Common Vulnerability Scoring System [3]. This scoring system allows a qualitative representation (such as low, medium, high and critical) of the severity of an attack enabling prioritization in the vulnerability management process. First attempts were to implement a generator that transfers the model data to an online tool. However, since this would have required a longer modeling time due to the transfer to the online tool and a permanent internet connection, this idea was rejected. In the current version, the CVSS calculator is integrated directly into the SAM modeling tool MetaEdit+. The advantage of this is that no internet connection is required and the results can be viewed in real time next to the rest models. During the integration, we oriented ourselves to the color scheme of the CVSS. In this way, other analysis tools can also be integrated [1]. We demonstrate the use of the security modeling extension and CVSS calculation with examples in Section III.

C. Safety-related capabilities of this approach

To integrate aspects of functional safety, we followed functional safety standard ISO 26262 applied in automotive [4]. As in ISO 26262 an ‘Item’ is related to ‘Hazards’ and these are related to ‘Hazardous events’, which in turn can be classified according to ‘Severity’, ‘Exposure’ and ‘Controllability’. All these elements are also elements in the metamodel, like modeling objects or their properties. Fig. 5 shows the overview of the complete metamodel for dependability modeling for safety as defined in EAST-ADL.

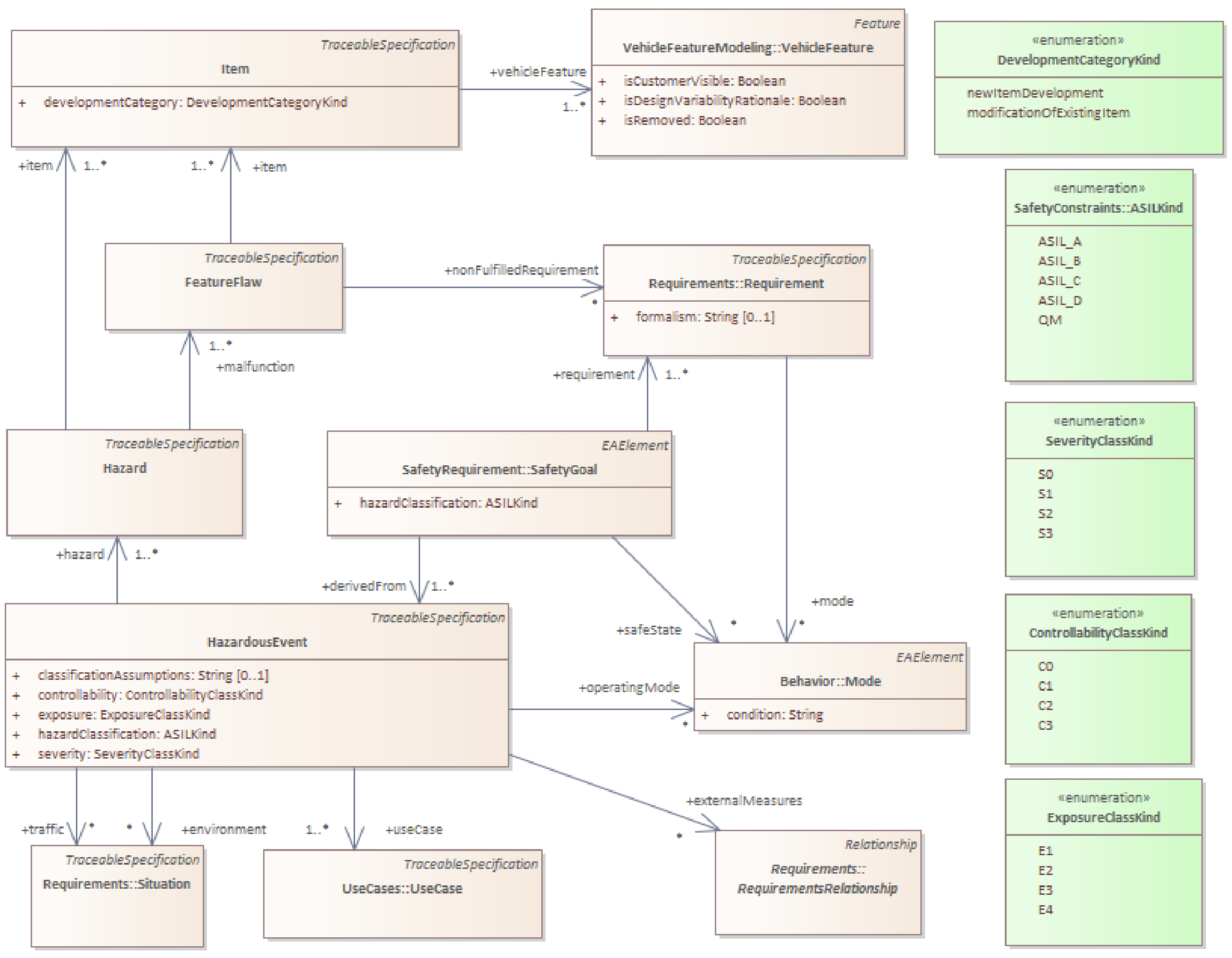


Fig. 5. Dependability package of the metamodel

The dependability package includes support for defining and classifying safety requirements through preliminary Hazard Analysis Risk Assessment (HARA), tracing and categorizing safety requirements according to their role in the safety life-cycle, as well as formalizing safety requirements using safety constraints. The dependability package itself is an extension to the automotive architecture modeling language EAST-ADL which already covers the modeling support for features, functions, hardware and related allocations. This full metamodel is described at http://east-adl.info/Specification.html along with the new version currently under review.

The metamodel shows how the integration with system architecture specification is established: the 'Item' (as defined in ISO 26262) is connected to features of the vehicle. As illustrated in Fig. 5 also constraints for this connection are defined making it mandatory (multiplicity is 1..*). In the metamodel of EAST-ADL, these individual features and their subfeatures are again connected via their context to any element in system architecture, such as to HardwareComponentTypes or FunctionTypes. Another form of connecting dependability modeling for safety is to apply 'Requirement' defined already in EAST-ADL and relate it with 'Safety Goals' and 'Feature Flaws'.

To formalize and assess fault propagation within the system, the dependability package also includes support for error modeling and organizing evidence of safety in a Safety Case. These are defined also via metamodels albeit not presented in Fig. 5. The integration of the system developed, the nominal system, and the error models is defined in the metamodel with trace links. Also, to minimize the modeling effort, automated error model generation based on system models is possible - and described with examples in Section IV.

D. Benefits from integrated languages

Integration of security and safety concerns with the language for system development offers several benefits over using separated languages — and tools and formats:

- Trace and analysis: A change in system engineering models or in safety/security-related models can be traced and analyzed. For example, all elements in system design that the safety-related ‘Item’ and its ‘Hazard’ can be related with can be identified. Similarly, if the system design is changed, e.g., by removing a feature, the related security or safety models can be identified and removed too — as they have become obsolete.
- Collaboration: Access to the system design as well as to security and safety aspects enables collaboration with fast feedback loop. If the modeling tools are not file-based, but apply a shared repository, then collaboration is even possible real-time. It is also up to tool features if access and collaboration is managed in some form, like allowing safety engineers to view system designs but not change them.
- Versioning: All models can be versioned together. There is no need to work with possible different formats, versioning systems, or collect data from different sources to get the complete picture at a particular point of time.
- Once the models share the same metamodel it is possible to run model checking and model transformations, like automatically produce initial safety models for the currently designed system. This way safety engineers don’t need to manually create all safety models and they can better assure that their safety analysis is based on the planned system. With traces they can also follow what changes are made in the planned system too. Also, security aspects can be analyzed in the same way such as calculating vulnerability scores as described with example in Section III.
- Last and most importantly, security and safety design become tightly related to system designs sharing the same model structure.

III. Security Example

To illustrate the practical application, we show two scenarios of malware injections as examples for possible attacks: One via social engineering attacks and one via vehicle-to-vehicle attacks. In the first example an attacker attacks vehicle-to-vehicle communication in autonomous vehicles to install malware. The second attacker uses a social engineering attack to trick the owner into installing malicious software.

Fig. 6. shows the first case on how vehicle-to-vehicle communication is used in autonomous driving vehicles to infiltrate malware into the attacked vehicle via an attacking vehicle and what suitable countermeasures are available. Autonomous vehicles must know themselves and their surroundings very well in order to navigate through traffic as safely as possible. This requires not only many sensors and their evaluated data, but also the ability to communicate with other vehicles (Vehicle 2 Vehicle) or their environment (Vehicle 2 Everything) in an emergency. Many development projects in the field of autonomous driving are already taking place and aim to increase driving safety. But it is precisely the additional communication interfaces here that give rise to the possibility of attacks from outside: the attacker only must pose as a

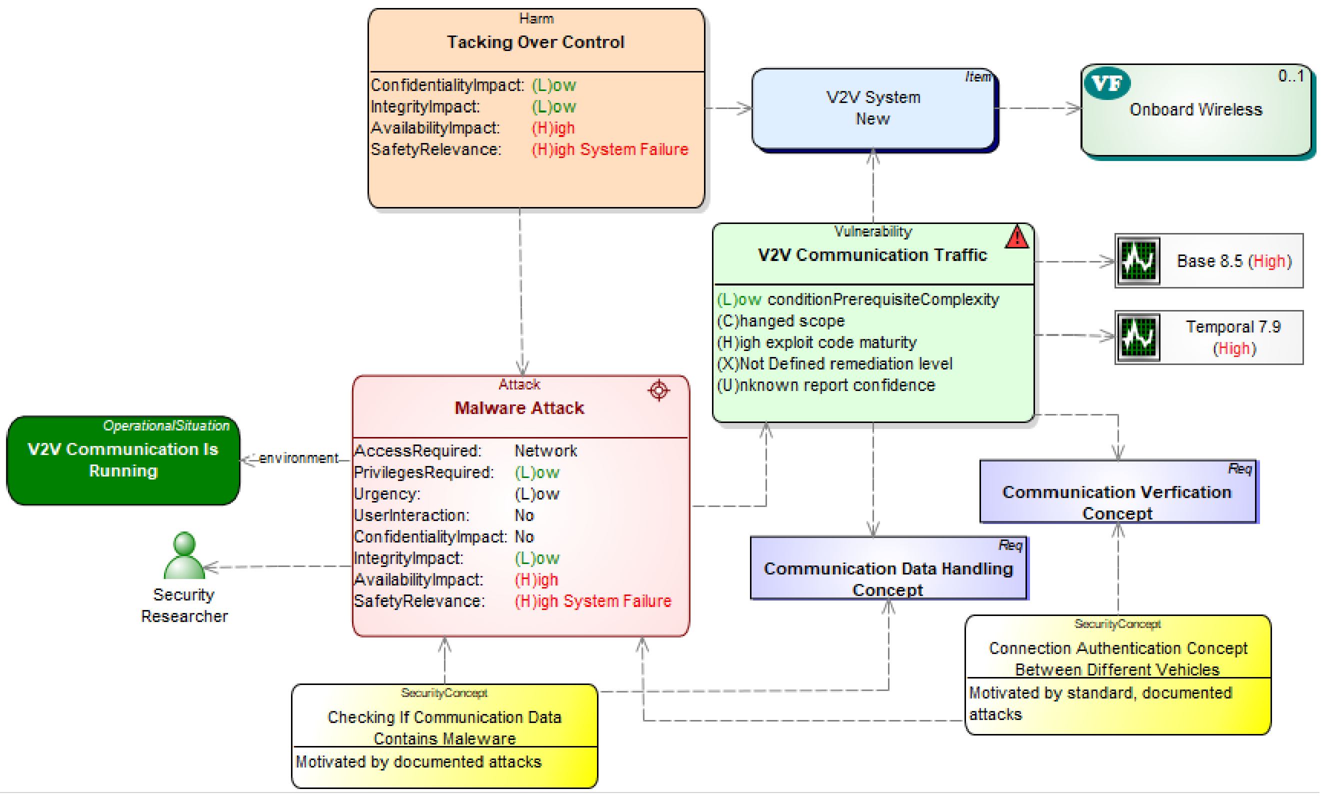

Harm
Tacking Over Control
ConfidentialityImpact: (L)ow
IntegrityImpact: (L)ow
AvailabilityImpact: (H)igh
SafetyRelevance: (H)igh System Failure
Item
V2V System New
VF
0..1
Onboard Wireless
Vulnerability
V2V Communication Traffic
(L)ow conditionPrerequisiteComplexity
(C)hanged scope
(H)igh exploit code maturity
(X)Not Defined remediation level
(U)nknown report confidence
Base 8.5 (High)
Temporal 7.9 (High)
Attack
Malware Attack
AccessRequired: Network
PrivilegesRequired: (L)ow
Urgency: (L)ow
UserInteraction: No
ConfidentialityImpact: No
IntegrityImpact: (L)ow
AvailabilityImpact: (H)igh
SafetyRelevance: (H)igh System Failure
OperationalSituation
V2V Communication Is Running
environment
Security Researcher
Req
Communication Verfication Concept
Req
Communication Data Handling Concept
SecurityConcept
Connection Authentication Concept Between Different Vehicles
Motivated by standard, documented attacks
SecurityConcept
Checking If Communication Data Contains Maleware
Motivated by documented attacks

Fig. 6. Injecting Malware via V2V communication

trustworthy vehicle or environmental object and can thus establish a data connection to the attacked vehicle. The attacker can then use this data connection to install malware or steal sensitive user data. Therefore, it must be ensured that both the communication between the vehicles is sufficiently authenticated beforehand and that the transmitted data is checked for possible attacks such as the direct installation of malware or access to internal program structures through a memory overflow.

In the second example Fig. 7 shows a social engineering attack with the aim of getting the vehicle owner to update the infotainment system with malicious software. Due to the advanced digitization worldwide, more and more things are being done online without meeting the person opposite in person. It is also possible to install updates for infotainment systems simply from a downloaded file on a USB. However, this is precisely where the danger lies that one becomes the victim of a social engineering attack and corrupted software is installed on one's system, which serves as a gateway to further attacks. Fig. 7 shows such an example. The attacker first spies on the vehicle type, license plate number and other details of the intended victim. Contact is then made with information about a required service update for the vehicle's on-board computer, which the owner can easily install himself. If the victim is persuaded, the attacker will send them the software via a fake website, a CD or a USB stick. The victim installs the malicious software and the attacker gains access to the system via a back door and can, for example, listen to calls made by the victim via the hands-free kit or use the navigation system to track the victim's location. Countermeasures for this would be, for example, an update lock, which only official car workshops can bypass with special devices, but also a warning from the infotainment system itself that no new one is necessary and thus warns the victim well before something is installed. To model this, the latest extension of the SAM metamodel for social engineering attacks was used (https://www.in.th-nuernberg.de/professors/BerglerMa/SAM/).

Modeling support for security covers calculating and displaying CVSS score for the attacked components. In Fig 7 the base and temporal scores are both 'high' for the vulnerability with values 8.8 and 8.1 respectively. While the calculations can be omitted, showing them at modeling time helps security engineers easier to identify possible weak points and initiate countermeasures as early as the design phase.

Since SAM supports comprehensive threat modeling that captures very detailed interrelationships between the properties of the attack and the vulnerability as well as their relationships to the architecture, some of which are not known in practice or should not be captured in all details for pragmatic reasons, a step-by-step approach to modeling with SAM makes sense. Therefore, we introduce different levels of modeling details, where Level 1 only models the motivation of the attacker, whereas Levels 2 and 3 are based on more details and information and thus enable a better representation of the threat situation; of course, this more detailed representation is accompanied by a higher development effort. Modeling tool also assist here security engineer to complete the specification by informing what elements can be considered next.

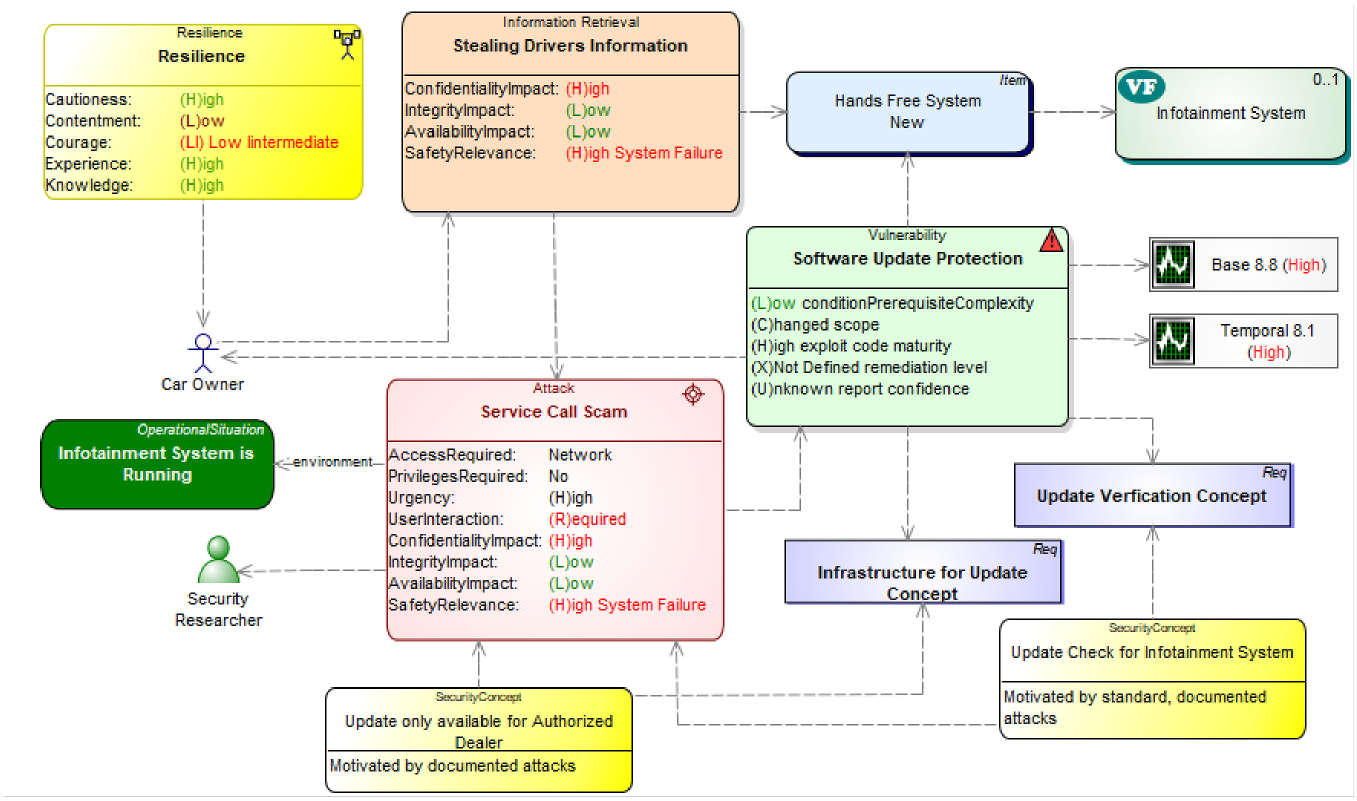

Resilience
Resilience
Cautioness: (H)igh
Contentment: (L)ow
Courage: (Ll) Low Intermediate
Experience: (H)igh
Knowledge: (H)igh
Information Retrieval
Stealing Drivers Information
ConfidentialityImpact: (H)igh
IntegrityImpact: (L)ow
AvailabilityImpact: (L)ow
SafetyRelevance: (H)igh System Failure
Item
Hands Free System
New
0..1
Infotainment System
Vulnerability
Software Update Protection
(L)ow conditionPrerequisiteComplexity
(C)hanged scope
(H)igh exploit code maturity
(X)Not Defined remediation level
(U)nknown report confidence
Base 8.8 (High)
Temporal 8.1 (High)
Car Owner
OperationalSituation
Infotainment System is Running
environment
Attack
Service Call Scam
AccessRequired: Network
PrivilegesRequired: No
Urgency: (H)igh
UserInteraction: (R)equired
ConfidentialityImpact: (H)igh
IntegrityImpact: (L)ow
AvailabilityImpact: (L)ow
SafetyRelevance: (H)igh System Failure
Security Researcher
Req
Update Verfication Concept
Req
Infrastructure for Update Concept
SecurityConcept
Update Check for Infotainment System
Motivated by standard, documented attacks
SecurityConcept
Update only available for Authorized Dealer
Motivated by documented attacks

Fig. 7. Injecting Maleware via Social Engineering Attack

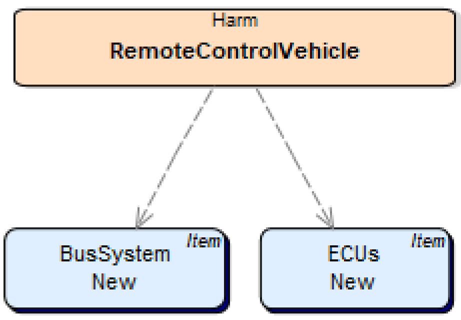


Fig. 8. Level 1 Representation of an attack in SAM

Level 1 is an entry level when working with SAM. It is to be used if only the motivation of the attacker is known about the attack(s) in question. This information already allows a pragmatic overview of the situation, a reference to the architecture of the automotive software system and an assessment of the severity of the attack, which is particularly relevant from a management perspective. As in Fig. 8, in Level 1 only the motivation is specified; specifically, this is one (or more) of the subclasses of ‘AttackMotivation’, i.e. either ‘Harm’, ‘InformationRetrieval’, ‘ProductModification’ or ‘FinancialGain’.

The link between attack motivation and item plays a central role here. It arises from considerations of which item(s) is/are endangered by the attack. In Level 1, no details are known yet, so the item can be an umbrella term. For example, it is known that the car's bus system needs to be attacked for a given attack. However, it is not yet possible to say exactly which items are affected. Therefore, a "bus system" item can be created that includes a group of items. However, the item must be specified because it forms the connection to the rest of the automotive software architecture. Without the item, the threat modeling would be detached and would have no context to the rest of the architecture.

At Level 2, a more detailed threat analysis already takes place. The methodical orientation on Level 2 is recommended as soon as details about the attack are known, especially regarding the attribute ‘breaksSecurityGoal’. The collection of (selected) details about the attack(s) are on the one hand not very time-consuming, but on the other hand offer a significantly better assessment of the threat situation compared to Level 1, since the focus of analysis is now no longer on human experience, but on technical feasibility. The entity Attack inherits the attribute breaksSecurityGoals of AttackMotivation. However, it is important to note that a single motivation for an attack can be divided into several subattacks, which can have different breaksSecurityGoals.

In addition to the breaksSecurityGoals, the reference to the followUpAttacks must also be modeled and the reference to the affected item must be established. The risk associated with the attack can be assessed qualitatively (which SecurityGoals are broken) or quantitatively (how many SecurityGoals are broken); this represents an improved assessment compared to Level 1,

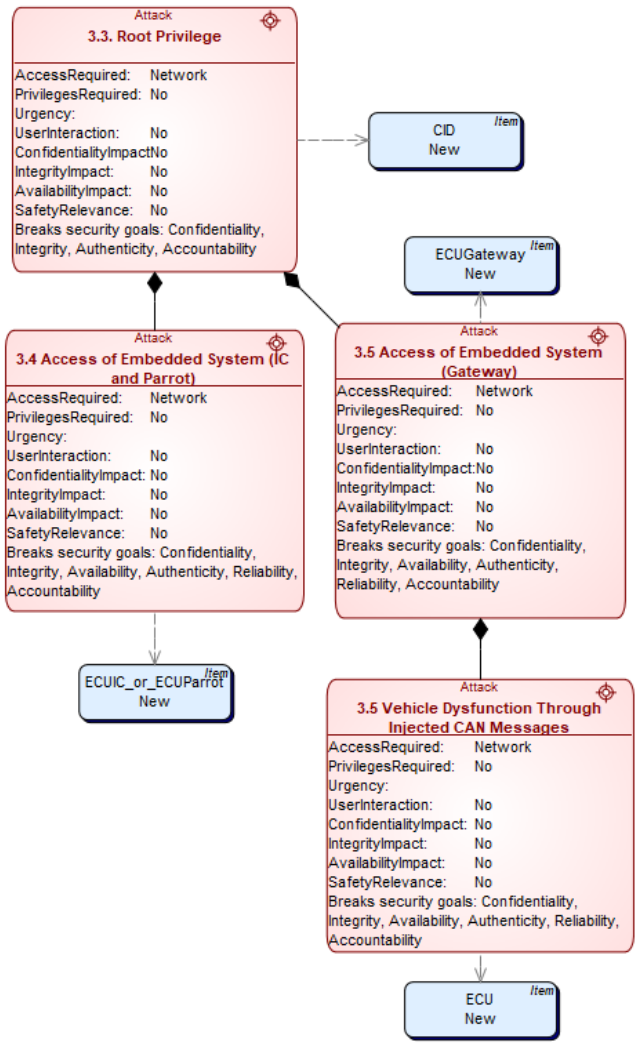


Fig. 9. Level 2 Representation of an attack in SAM

where the analysis is only carried out based on the motivation of the attack.

A better assessment of the severity of an attack results from the calculation of a score, for example analogous to the Common Vulnerability Scoring System (CVSS, see level 3). Level 2 is not detailed enough to calculate this score, but an initial assessment based on the experience of security experts can already be made and documented in the Score entity. To make it clear that this score is experience-based and has not been calculated, the attribute ‘calculationFormula’ is left empty in this case. Score can be omitted if no experience values are available.

ISO/SAE 21434 has been in place since 2021 to ensure a complete risk assessment analysis of cyber-attacks on vehicle systems [5]. This standard was developed because of the need to counteract against cyber-attacks due to the increasing networking of vehicle systems. The standard is related to the UNECE regulation R 155 "Cyber security and cyber security management system". The application of ISO 21434 is considered as a building block to facilitate certification. However, ISO 21434 does not cover all the requirements of R

155. To develop a reporting system for SAM that is understandable for different viewers, it makes sense to do this based on ISO 21434. The main steps in performing an ISO/SAE 21434 compliant threat analysis and risk assessment are (in order of an idealized linear execution):

1. Item Definition (Section 9.3)
2. Asset Identification (Section 15.3)
3. Identification of threat scenarios (Section 15.4)
4. Impact Rating (Section 15.5)
5. Attack Path Analysis (Section 15.6)
6. Attack Feasibility Rating (Section 15.7)
7. Risk Value Determination (Section 15.8)
8. Risk Treatment Decision (Section 15.9)
9. Cyber Security Goals (Section 9.4) [WP-09-03 & RQ-09-07]
10. Cyber Security Claims [WP-09-04 & RQ-09-06]
11. Cyber Security Concept (Section 9.5)

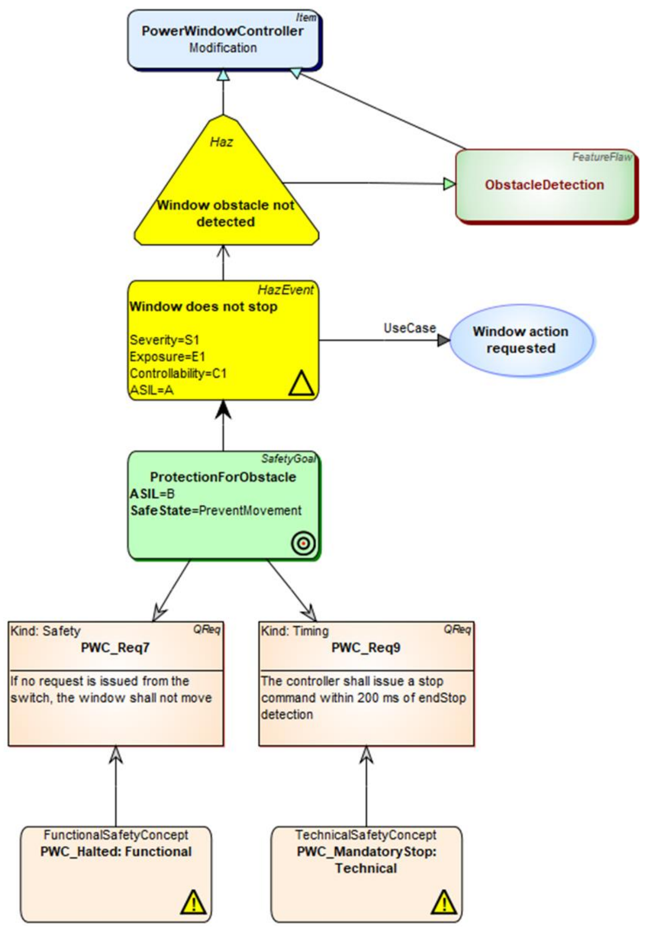
Most of the points listed are already implemented by SAM, including points 1, 2, 3, 4, 9, 10 & 11. Therefore, only the following points 5, 6, 7 & 8 have to be added to SAM to be able to create a reporting system based on ISO 21434. The points 9 to 11 are also fulfilled by the fact that they are applied holistically and not just domain-specifically and thus also cover the aspect of cybersecur1ity.

IV. Safety Example

We focus next on safety and illustrate dependability modeling, error modeling and automated FTA/FMEA analysis that are possible due to metamodel extensions for safety. Fig. 10 shows the dependability model for PowerWindowController. This dependability model closely follows functional safety standard ISO26262 as has been defined in the metamodel (see Fig 5). In top of Fig. 10, PowerWindowController is considered as an item. Next, a hazard 'Window obstacle not detected' is specified and related to HazardousEvent 'Window does not stop'. This Hazardous event is related to the use case that window action is requested. It may also be linked - albeit not described in the Fig. 10 - to other scenarios linked to traffic situation, environment or to operating mode. In other words, the model follows the metamodel as defined based on ISO 26262.

The example also describes the work on hazard analysis in which Severity, Exposure and Controllability and ASIL values are being defined. The dependability model also defines the safety goal with a safe state "PreventMovement". It reduces the ASIL level to an acceptable level (B) with Requirements #7 and #9 that are already defined in the requirements model.

Thanks to the integrated metamodel safety design is not separated from the rest of the systems modeling. The integration is achieved by an 'Item' called PowerWindowController that, according to the metamodel, can refer to one or more features. In our example the item refers to the PowerWindow feature of the vehicle. This feature is defined in the feature model of EAST-ADL and is illustrated in Fig. 11.



If no request is issued from the switch, the window shall not move

The controller shall issue a stop command within 200 ms of endStop detection

Fig. 10. Dependability model of PowerWIndowController

The features of Power Window are defined by systems engineers along with its functional architecture (as illustrated in Fig 12). Features in the feature model map to individual elements in the functional architecture. This enables traceability between system designs and safety designs.

System design as in Fig. 12, however, is not directly suitable for safety design as it does not identify and capture typical safety aspects like faults, failures or failure rates. Design models also include information that is not relevant for safety work adding unnecessary complexity for safety engineers. Consider for example the small system illustrated in Fig. 12. It shows the functional architecture of a PowerWindow controller: its functions, ports and connections among them. Functions are classified, have a more detailed internal hierarchical structure, and their ports have interfaces and data types. Some of these aspects, like classification of functions or data type definitions, are not directly relevant in safety design, but are necessary to generate the implementation (like AUTOSAR ARXML, Simulink models etc.).

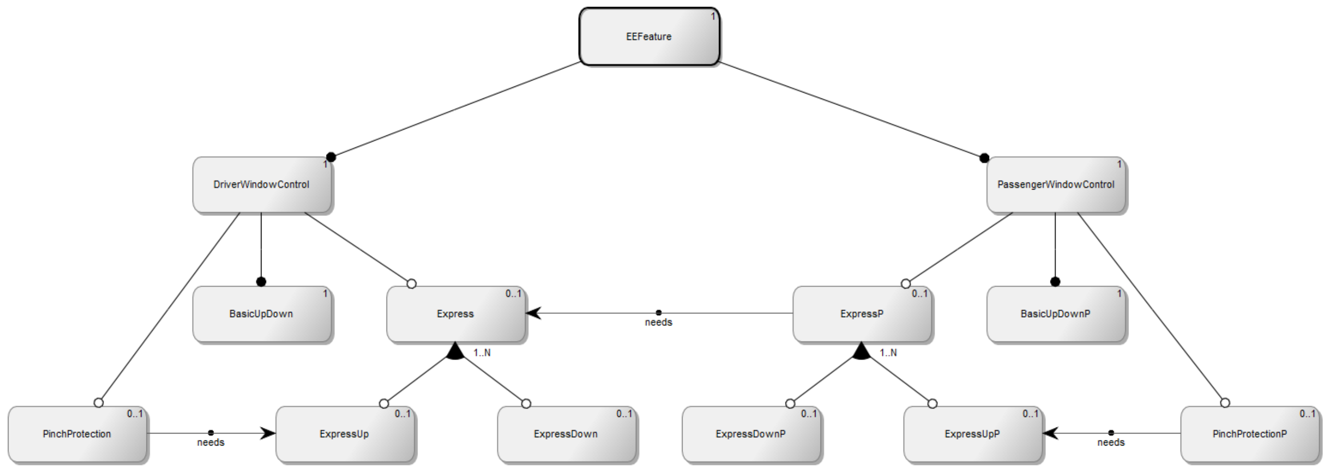


Fig. 11. Features of power window

Adding safety-related information, like faults or failure logic, directly to the design model is often not practical because it would quickly make the model complex with too many details. For safety analysis, unlike design, we also usually need several models covering different analysis scenarios. One solution is to generate the initial safety models from design specifications. For example, Fig. 13 shows the result of such a transformation: an initial error model produced from the system design shown in Fig. 12. The error model is then detailed for safety analysis focusing on failures, faults, and error types. Safety engineers may then add error behavior to this model or add other aspects of safety, like e.g., in Fig. 13 a FailureOut port is added to analyze an error on obstacle detection (top right in the model with blue thick border for propagation).

With error modeling, safety engineers can specify various faults and failure logic without changing the actual system designs. Yet, there is a link from error models back to nominal planned system descriptions. As error models can specify failure logic (Boolean and temporal) the models also serve as the basis for automated Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) [8][13]. This means that rather than creating fault tree diagrams manually (as in Fig. 3 earlier), they can be generated. For this purpose, we implemented model transformation that takes error models and translates them to the formats needed by FTA/FMEA tools. Since the transformations are fully automated the cost and effort to carry FTA and FMEA are greatly reduced. Fig. 14 shows the result of running the transformation from error models to analysis tools for FMEA.

Development of safety-critical functions with the model-based approach starts with hazard analysis and risk assessment in the dependability model, is detailed with error models for FMEA, and ends with verification of safety goals and safety requirements. Since models contain the needed information, it is possible to generate the documents like functional and technical safety concepts as well as verification and validation of safety goals (as done e.g., in [14]).

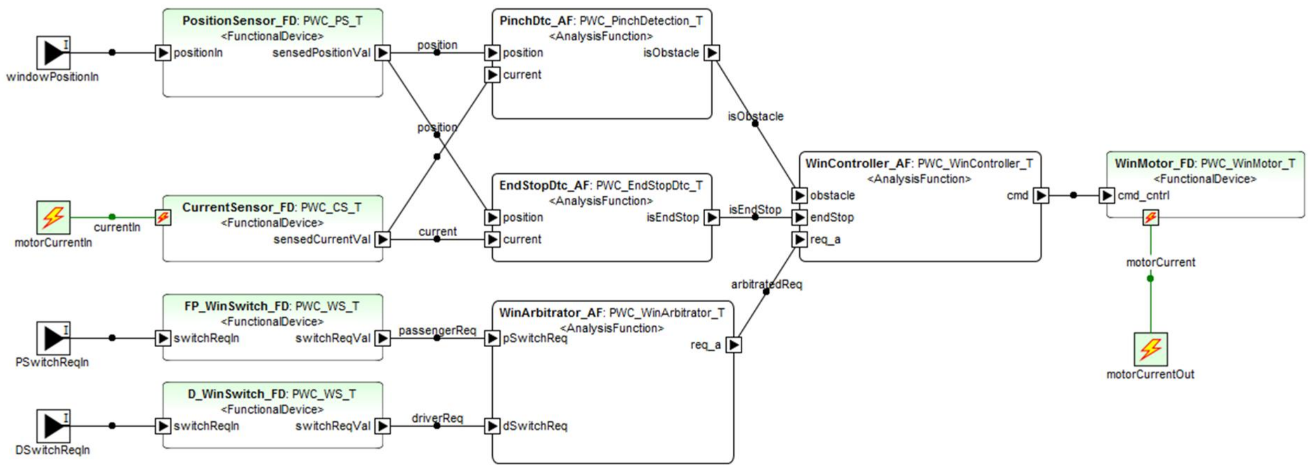
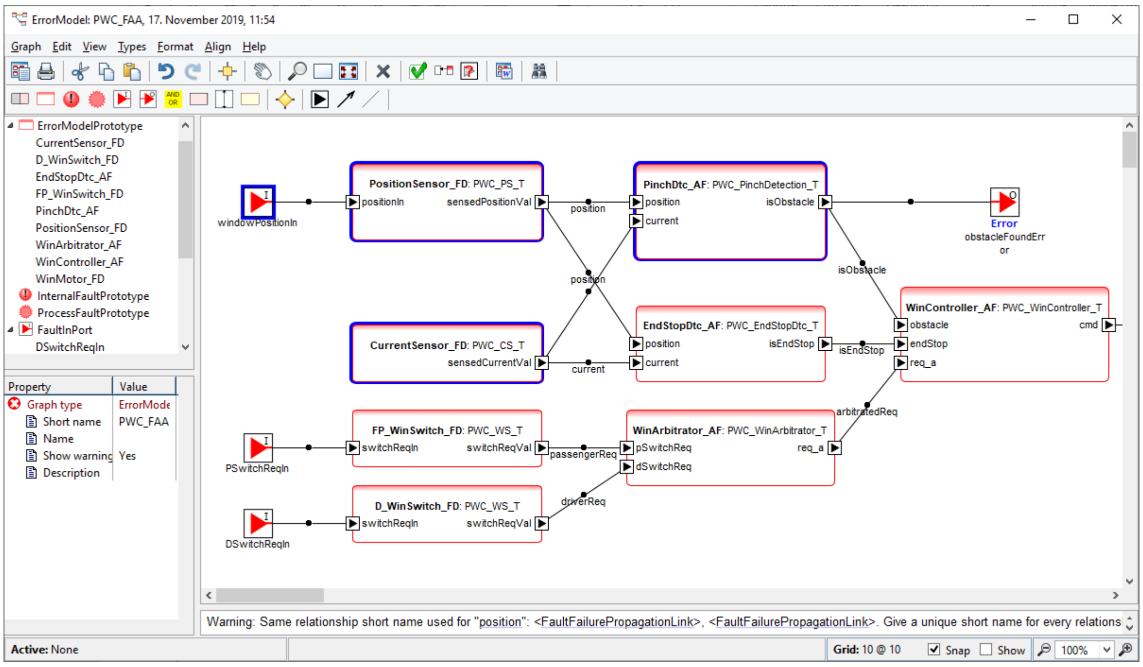


Fig. 12. Functional architecture of PowerWindow controller



Property	Value
Graph type	ErrorMode
Short name	PWC_FAA
Name	
Show warning	Yes
Description	

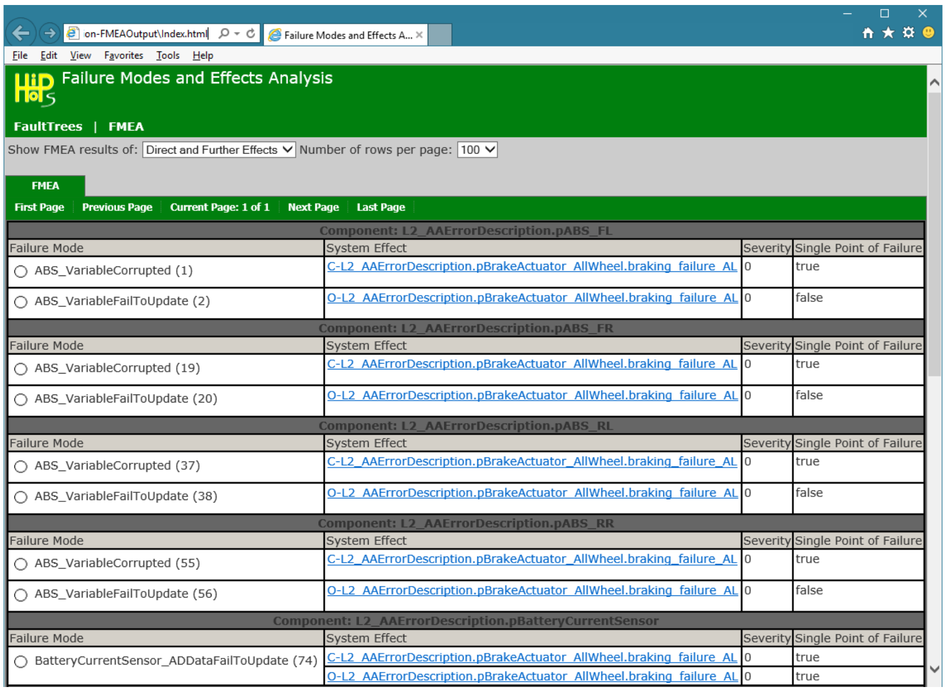
Warning: Same relationship short name used for "position": <FaultFailurePropagationLink>, <FaultFailurePropagationLink>. Give a unique short name for every relations

Fig. 13. Functional architecture of PowerWindow controller

These automations make safety work easier and faster to do as well as reduce manual error-prone routine tasks. Perhaps most importantly, they enable feedback from safety analysis earlier to be acknowledged in the system design.

V. Experiences on the Language Definition

Our efforts on implementing security and safety modeling have been related to existing automotive system development language, namely to EAST-ADL. For this reason, the actual language implementation required us to only specify the extensions (as in Fig. 4 and 5) and link them to the existing metamodel. The effort and process would be largely the same as if these would be added to other languages, like to AADL or SysML - given that these languages would have language constructs (elements in the metamodel) that are suitable for extension and integration with safety and security.



Show FMEA results of: Direct and Further Effects Number of rows per page: 100

Failure Mode	System Effect	Severity	Single Point of Failure
Component: L2_AAErrorDescription.pABS_FL			
ABS_VariableCorrupted (1)	C-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	true
ABS_VariableFailToUpdate (2)	O-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	false
Component: L2_AAErrorDescription.pABS_FR			
ABS_VariableCorrupted (19)	C-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	true
ABS_VariableFailToUpdate (20)	O-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	false
Component: L2_AAErrorDescription.pABS_RL			
ABS_VariableCorrupted (37)	C-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	true
ABS_VariableFailToUpdate (38)	O-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	false
Component: L2_AAErrorDescription.pABS_RR			
ABS_VariableCorrupted (55)	C-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	true
ABS_VariableFailToUpdate (56)	O-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	false
Component: L2_AAErrorDescription.pBatteryCurrentSensor			
BatteryCurrentSensor_ADDataFailToUpdate (74)	C-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	true
	O-L2_AAErrorDescription.pBrakeActuator_AllWheel.braking_failure_AL	0	true

Fig. 14. Results of fault tree analysis and failure modes and effects analysis.

The language definition steps consist of:

1. Defining the metamodel for the extensions and linking them with the existing language definition. Linking enables reuse, references and traces between model elements.
2. Setting constraints to keep specifications consistent and ensure syntactic completeness and correctness.
3. Defining notation that fits or resembles the domain being addressed (e.g., safety, security).
4. Implementing generators for model checking and trace as well as producing various kinds of artifacts like data for FMEA or CVSS.

In this paper we described the first two steps via the metamodel in Section II. Defining the notation in step 3 deals with symbol definitions and the work by Moody [10] can be applied directly here. The actual implementation of notations then varies on tools as some require programming them whereas for others they can be imported as images without much additional work [9].

The last step on generators then brings in more automation possibilities for checking, tracing, reporting and generating code, simulations etc. In our case the generators targeted external tools for performing Failure Mode and Effects Analysis (FMEA) as well as calculating vulnerability scores. In both cases implementing the actual generators were straightforward as for both needs specifications of the formats to be generated were available. The second kind of generators were those reporting the work in trace reports or various other documents. These were performed in the same MetaEdit+ tool.

The effort to create modeling support goes mostly to identifying and testing the suitable level of abstraction. We did not measure the effort on the safety side, but for the security side after having the initial metamodel (as in Fig. 4), its implementation into a modeling tool was done in the period of two calendar weeks by one person. Verification and validation were done by other people using the language for typical modeling cases.

It is important to note that if the tools provide access to the language definitions the modeling support can be extended incrementally based on the needs. For example, both the metamodel definitions addressing security and safety have evolved because of the changes in the modeling requirements and because learning from the language usage. This makes the suggested approach also future proof as we already know that possible new requirements can be addressed. Access to the metamodel and generators also gives possibilities to adapt the support for company specific needs - as described in [14]. We are also aware of a case in developing ADAS systems in which modeling support of EAST-ADL is extended with concepts like Safety Measures.

VI. Conclusions

We have presented a model-based approach for integrating safety and security concerns with the rest of system development. This is achieved via an integrated modeling language that provides modeling support for safety and security at language level similarly we have got used to with traditional system and software modeling languages. We demonstrated our approach been applied with practical examples. The benefits of integration include:

- Collaborative development: Access to the system design as well as to security and safety aspects enables collaboration with fast feedback loop.
- Trace and analysis are possible — even at modeling time — between different aspects of the developed system.
- All work items can be versioned together. There is no need to work with possible different formats, versioning systems, or collect data from different sources to get a complete picture.
- Automated analysis and transformations: combined models can be used as input for checking and model transformations, like automatically producing initial safety models for the currently planned system design.
- Security and safety design becomes tightly related to system designs sharing the same model structure.

With EAST-ADL and its extensions these benefits are already available [2], but the same principles can be applied if other modeling languages would be extended. For example, extend SysML instead of EAST-ADL, or add metamodel of RAAML from [11] instead of the dependability metamodel of EAST-ADL.

References

[1] Bergler, M. et al. Social Engineering Exploits in Automotive Software Security: Modeling Human-targeted Attacks with SAM. Proceedings of the 31st European Safety and Reliability Conference (ESREL 2021), 2021

[2] EAST-ADL, 2021, http://www.east-adl.info/Specification.html [Accessed 21 April 2022].

[3] First, Common Vulnerability Scoring System version 3.1: Specification Document, 2019, https://www.first.org/cvss/specification-document [Accessed 21 April 2022].

[4] ISO Functional Safety, 26262-1, 2018

[5] ISO/SAE 21434:2021 - “Road vehicles - Cybersecurity engineering” https://www.iso.org/standard/70918.html [Accessed 16 May 2022]

[6] Kelly, S., Tolvanen, J.-P., Domain-Specific Modeling: Enabling full code generation, Wiley-IEEE Computer Society Press, 2008

[7] Kritzinger, D., Fault tree analysis, in Aircraft System Safety, Elsevier, 2017

[8] Lee, W. S., Grosh, D. L., Tillman, F. A., Lie C. H., Fault Tree Analysis, Methods, and Applications - A Review. IEEE Transactions on Reliability, Volume: R-34, Issue: 3, Aug. 1985.

[9] MetaCase, MetaEdit+ User’s Guide. [Online]. Available at: https://metacase.com/support/55/manuals/, 2018 [Accessed 21 April 2022].

[10] Moody, D., The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering, IEEE Transactions on Software Engineering, vol. 35, no. 6, 2009.

[11] OMG, Risk Analysis and Assessment Modeling Language (RAAML), https://www.omg.org/spec/RAAML/1.0/Beta1/PDF, 2021 [Accessed 21 Jan 2022].

[12] OMG, System Modeling Language, version 1.6. [online] Available at: https://www.omg.org/spec/SysML/, 2019 [Accessed 21 April 2022]

[13] Reifer, D., Software Failure Modes and Effects Analysis, IEEE Transactions on Reliability, Volume: R-28, Issue: 3, 1979.

[14] Sari, B., Fail-Operational Safety Architecture for ADAS/AD Systems and a Model-driven Approach for Dependent Failure Analysis. Springer, 2020.

[15] SEI, Architecture Analysis and Design Language (AADL), https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439,191439 [Accessed 13 May 2022]

[16] Zoppelt, M. Tavakoli Kolagari, R., SAM: A Security Abstraction Model for Automotive Software Systems, ISSA/CSITS@ESORICS, 2018.

[17] Zoppelt, M., Tavakoli Kolagari, R., UnCle SAM: Modeling Cloud Attacks with the Automotive Security Abstraction Model, 2019.