Engineering Framework for the Generation and Integration of Digital Dependability Identities

White Paper

This document builds upon the general DDI concept and provides methodology, concepts and specifications for the generation and integration of DDI at development time

www.deis-project.eu

Table of Contents

Publishable Executive Summary 6
1 Introduction 7
2 Engineering Framework for the Generation and Integration of DDI 8
2.1 DDI concept evolution 8
2.1.1 Recap of the DDI concept 8
2.1.2 Modifications and Extensions since the initial DDI concept 11
2.2 Semi-automated Engineering of DDIs 13
2.2.1 The challenge 13
2.2.2 Engineering tasks to be supported by DDIs 14
2.2.3 How DDIs enable (semi-)automated engineering based on the ODEv2 18
2.2.4 Interfaces of SACM to define artefact and process semantics 23
2.2.5 Initial ideas for an action language to enable automated reasoning based on DDIs 26
2.2.6 Advances regarding runtime dependability assurance 28
3 The Open Dependability Exchange Meta-model (ODE) v2 29
3.1 The complete ODE meta-model 29
3.2 Structured Assurance Case Meta-Model (SACM) 2.0 31
3.2.1 SACM Assurance Case Component 31
3.2.2 SACM Machine-Readable Design 32
3.2.3 SACM Argumentation Component 32
3.2.4 SACM Artifact Component 34
3.2.5 SACM Terminology Component 34
3.2.6 Summary 36
3.3 Product meta-models 36
3.3.1 Architectural modelling package 36
3.3.2 Hazard and risk analysis (HARA) modelling package 37
3.3.3 Failure logic modelling package 39
3.3.4 Threat and risk analysis (TARA) modelling package 42
3.4 Certification meta-models 43
3.5 Crosscutting Aspects 43

3.5.1 Information abstraction and IP Hiding concept 43
3.5.2 Harmonization of the ODE with common meta-modeling languages 44
4 Utilization of DDIs in concrete engineering activities of DEIS use cases 50
4.1 Utilization of DDIs in an OEM-TIER integration scenario in the context of the ETCS 50
4.1.1 Use case overview 50
4.1.2 Semi-automated ETCS assurance case instantiation 51
4.1.3 Optimized supplier component selection based on DDI 55
4.2 Utilization of DDIs in security and safety co-engineering in the context of the Physiological Health Monitoring Use Case 57
4.2.1 Use case overview 57
4.2.2 Addressed challenge in safety-security co-engineering 58
4.2.3 Long cycle times integration processes at development time 60
4.2.4 Engineering Story 1: Unauthorized Emergency Brake 61
4.2.5 Engineering Story 2: Vehicle unintended behavior 64
4.2.6 ES3: Compromise driver's privacy 65
4.3 Utilization of DDIs in applications involving General Data Protection Regulations (GDPR) 66
4.3.1 Introduction 66
4.3.2 GDPR Requirements 67
4.3.3 DDI Implementation of GDPR 67
5 Summary and Outlook 70
References 71
Appendix A 72

List of Figures

Figure 1 - SACM-based safety argumentation as the backbone of a DDI 9
Figure 2 - Integration scenario where the assurance case models are the backbone of the DDI 10
Figure 3 - ODEv1 high-level DDI structure 11
Figure 4 - ODEv2 high-level DDI structure 12
Figure 5 - Goal of semi-automated DDI engineering 13
Figure 6 - Anticipated design time engineering process 14
Figure 7 - User interface for exporting a component DDI 16
Figure 8 - (Semi-)automated assurance case instantiation with DDIs 17
Figure 9 - An example DDI: Enriching the assurance case with process and product semantics 18
Figure 10 - Automated claim validity assessment activity 20
Figure 11 - The Open Dependability Exchange Meta-Model (ODE) v2 22
Figure 12 – SACM fundamental packages 23
Figure 13 - SACM terminology package example 24
Figure 14 - SACM argument package segment 25
Figure 15 - SACM citation mechanism segment 25
Figure 16 - DDI Instantiation as a DAG 27
Figure 17 - Transitioning from design time to runtime DDI execution 28
Figure 18 - The complete ODE Meta-model v2 30
Figure 19 - SACM::AssuranceCase Component 31
Figure 20 - SACM::Base Component 32
Figure 21 - SACM::Argumentation Component 33
Figure 22 - SACM::Artifact Component 34
Figure 23 - SACM::Terminology Component 35
Figure 24 - ODE::Design Package 36
Figure 25 - ODE::Dependability Package 37
Figure 26 - ODE::Dependability::Domain Package 38
Figure 27 - ODE::Dependability::HARA Package 38
Figure 28 - ODE::Dependability::Requirements Package 39
Figure 29 - ODE::FailureLogic Package 40
Figure 30 - ODE: FailureLogic::FMEA Package 40
Figure 31 - ODE::FailureLogic::FTA Package 41
Figure 32 - ODE::FailureLogic::Markov Package 41
Figure 33 - ODE::TARA Package 42
Figure 34 - SafeML usage concept 45
Figure 35 - ERTMS/ETCS Reference Architecture 50
Figure 36 - Safety goal of the ETCS trackside part as defined in UNISIG Subset-091 52
Figure 37 - Example GSN diagram of the safety argumentation fragment of the ETCS trackside sub-system 52
Figure 38 - Process / product model and mapping model 53
Figure 39 - Process of the semi-automated assurance case instantiation based in the ODE 54
Figure 40 - DDI in form SACM AssuranceCasePackage for the ETCS trackside part 54
Figure 41 - Selection of the suppliers based on DDI 55
Figure 42- DPMS decision-making action paths 58
Figure 43 - DPMS general security model 59

FIGURE 44: ATTACK TREE EXAMPLE 60
FIGURE 45 - INTERNAL CO-ENGINEERING PROCESS 61
FIGURE 46 - UNAUTHORIZED EMERGENCY BRAKE ATTACK TREE 62
FIGURE 47 - LINUX-EMBEDDED-OS ATTACK TREE 63
FIGURE 48 - PRIVILEGED PROCESS ATTACK TREE........ 64
FIGURE 49 - VEHICLE UNATTENDED BEHAVIOR ATTACK TREE 65
FIGURE 50 - COMPROMISE DRIVER'S PRIVACY ATTACK TREE 66
FIGURE 51 - RELATIONSHIP BETWEEN SECURITY AND SAFETY RISKS........ 68

List of Tables

TABLE 1 - SAFEML TO ODE MAPPING........ 45
TABLE 2 - EAST-ADL2 TO HIP-HOPS MAPPING........ 48
TABLE 3 - SYSML (FOR COMPASS ARTISAN STUDIO) TO HIP-HOPS MAPPING........ 48
TABLE 4 - AADL TO HIP-HOPS MAPPING 49
TABLE 5 - HIP-HOPS TO ODE MAPPING........ 49
TABLE 6 - GDPR REQUIREMENTS (REFINED PROJECT REQUIREMENTS FOR SEMI-AUTOMATION)........ 67

Abbreviations

Abbreviation	Long Version
GDPR	General Data Protection Requirements
TARA	Threat Assessment and Remediation Analysis
CPS	Cyber Physical System
DPMS	Dependable Physiological Monitor Sytem

Authors

Dr. Gilbert Regan, Lero/DKIT	**Gilbert.regan@dkit.ie**
Dr. Fergal Mc Caffery, Lero/DKIT	Fergal.mccaffery@dkit.ie
Simone Longo, GM	Simone.longo@gm.com
Jan Reich, Fraunhofer IESE	Jan.reich@iese.fraunhofer.de
Daniel Schneider, Fraunhofer IESE	Daniel.schneider@iese.fraunhofer.de
Ioannis Sorokos, University of Hull	I.Sorokos@hull.ac.uk
Joe Guo, Siemens AG	Joe.guo@siemens.com
Marc Zeller, Siemens AG	Marc.zeller@siemens.com
Ran Wei, University of York	ran.wei@york.ac.uk
Tim Kelly, University of York	tim.kelly@york.ac.uk

Publishable Executive Summary

Cyber-Physical Systems (CPS) provide enormous potential for new types of applications, services and business models in any embedded systems domain, such as automotive, rail, healthcare or home automation. Overall, we anticipate a future of heavily interconnected, distributed, heterogeneous and intelligent systems, which are bound to have a significant economical and societal impact in the years to come.

However, several challenges need to be tackled before the full potential of CPS can be unlocked. One core challenge is to ensure the trustworthiness and dependability of single and composite systems, as established approaches and standards were designed with closed standalone systems in mind, thus building on a complete understanding and analysability of a system and its relevant environment. As this is no longer a given, we urgently require new types of approaches that do not (solely) rely on this basic assumption (now rendered void).

A general solution concept involves shifting parts of the assurance activities into runtime, where unknowns and uncertainties can be resolved dynamically. To this end, it is necessary to equip the constituent systems with dedicated and adequate modularised and formalised dependability information. The key innovation that is the aim of DEIS is the corresponding concept of a Digital Dependability Identity (DDI). A DDI contains all the information that uniquely describes the dependability characteristics of a CPS or CPS component. DDIs are synthesised at development time and are the basis for the (semi-)automated integration of components into systems during development, as well as for the fully automated dynamic integration of systems into systems of systems in the field.

In this document we build upon the initial version of the ODE meta-model and present version 2 of the model, which now includes a new security package. The main goal of this document is to specify the algorithms needed to provide adequate engineering support for the generation and integration of DDIs. With this goal in mind, tool transformations are specified so that DDI's can be generated from information already stored in existing tools. Additionally, this document demonstrates how tools can be used to generate DDI's in a semi-automated way, and how the integration of DDI into existing systems can be achieved through the use of supported tools in a semi-automated way, thus increasing efficiency as well as the confidence in the system's dependability. For the integration of DDIs, the information contained in DDIs must be transformed back into an appropriate (ODE-compliant) format that can be used in the tool chain used by the integrator.

Finally, verification of the utility of DDI is demonstrated through presenting how DDIs support a number of DEIS use cases and engineering stories, and how DDIs can be used in applications that must comply with the new General Data Protection Regulations (GDPR).

DEIS aims at providing comprehensive tool support for DDI, covering the supported/semi-automated synthesis of DDI as well as the (semi-)automated integration at development time. Moreover, it is our aim to support multi-tool scenarios, where DDI are exchanged and evolved among different development teams and tools.

1 Introduction

Cyber-Physical Systems (CPS) provide enormous potential for new types of applications, services and business models in any embedded systems domain, such as automotive, rail, healthcare or home automation. Overall, we anticipate a future of heavily interconnected, distributed, heterogeneous and intelligent systems, which are bound to have a significant economical and societal impact in the years to come.

However, several challenges need to be tackled before the full potential of CPS can be unlocked. One core challenge is to ensure the trustworthiness and dependability of single and composite systems, as established approaches and standards were designed with closed standalone systems in mind, thus building on a complete understanding and analysability of a system and its relevant environment. As this is no longer a given, we urgently require new types of approaches that do not (solely) rely on this basic assumption (now rendered void).

A general solution concept involves shifting parts of the assurance activities into runtime, where unknowns and uncertainties can be resolved dynamically. To this end, it is necessary to equip the constituent systems with dedicated and adequate modularised and formalised dependability information. The key innovation that is the aim of DEIS is the corresponding concept of a Digital Dependability Identity (DDI). A DDI contains all the information that uniquely describes the dependability characteristics of a CPS or CPS component. DDIs are synthesised at development time and are the basis for the (semi-)automated integration of components into systems during development, as well as for the fully automated dynamic integration of systems into systems of systems in the field.

This deliverable builds on the initial version of the ODE meta-model and specifies the algorithms needed to provide adequate engineering support for the generation and integration of DDIs. In order to generate a DDI, means of completing the following tasks need to be developed:

1. For the generation of DDI from existing safety engineering artefacts, tool-transformations shall be specified to translate the information stored in existing tools like ComposR, HiP-HOPS, ACME and safeTbox into an ODE-compliant model.

2. Generation of a DDI in a (semi-)automated way from ODE-compliant models. A DDI is designed to mask as much information as possible and to only provide what is absolutely required for a sound integration. The information contained in the full-fledged ODE-compliant model (i.e. the white box dependability case specification) is to be abstracted, simplified and formalized for the DDI. In order to enable a degree of automation in the construction of a DDI, a system for the synthesis of DDI's is required.

3. With regards to the integration of DDIs, the information contained in DDIs must be transformed back into an appropriate (ODE-compliant) format that can be included in the tool chain used by the integrator. When a component is integrated into a system, or when a system is integrated into another system, corresponding DDIs shall enable (semi-)automated integration into the dependability case of the overall system and thus significantly increase efficiency. A similar case is the exchange or the modification of a component and the respective DDI, when the impact of the change is to be analysed on the level of the overall integrated system.

Section 3 provides a recap of version 1 of the DDI concept, along with the modifications that have been made for the Open Dependability Exchange (ODE) Meta-model version 2. This section is the core of this document as it describes on the one hand, which types engineering tasks are to be (semi-)automatically supported by DDIs and on the other hand the conceptual basis, how we envision semi-automated generation and integration of DDIs to happen technologically.

Section 4.1 provides the meta-model for ODE version 2 which now includes a threat and risk analysis package. Section 4.2 describes the Structured Assurance Case Meta-Model (SACM) and its extension mechanisms, which constitute the backbone of the ODE. Section 4.3 describes the auxiliary modular ODE packages, which provide coverage for architectural modelling, hazard and risk analysis, threat and risk analysis, failure logic modelling and dependability requirements. Section 4.4 provides a certification meta-model which has been developed by the AMASS project. This meta-model has been developed for process and certification modelling and DEIS plans to use part of it in conjunction with ODE and SACM. Section 4.6 highlights the ODE's flexibility by briefly mentioning the potential for interoperation with pre-existing, established modelling languages such as SysML, EAST-ADL and AADL.

Section 5.1 describes the utilization of DDI in the European Train Control System use case, while Section 5.2 describes how the DDI can be used in applications involving General Data Protection Regulations (GDPR).

2 Engineering Framework for the Generation and Integration of DDI

2.1 DDI concept evolution

Section 3.1 provides an outline of the initial DDI concept, in addition to the modifications and extensions that now make up version 2 of the ODE meta-model.

2.1.1 Recap of the DDI concept

The general concept of the ODE meta-model and the DDI was designed on the basis of the diverse previous work of the DEIS consortium and the DEIS objectives and vision. A fundamental outcome of the ODEv1 was the decision to use SACM 2.0, the Structured Assurance Case Meta-Model of the Object Management Group, as a core ingredient for the ODE meta-model. Using the SACM entails several advantages which include:

1. It is already standardised and relatively mature, which might help us get the DDI concept and the ODE meta-model accepted and adopted eventually.
2. SACM is an exchange format for structured assurance cases and provides corresponding means for modularisation.

These properties render SACM a good candidate for being utilised as a backbone for the ODE meta-model.

To illustrate the utilisation of the SACM in the context of the ODE, consider the example depicted in Figure 1. It shows how an argumentation structure can be the front and centre artefact within a DDI. All other relevant artefacts – functional design, hazard and risk analyses or failure logic models – are directly linked to elements of the argumentation.

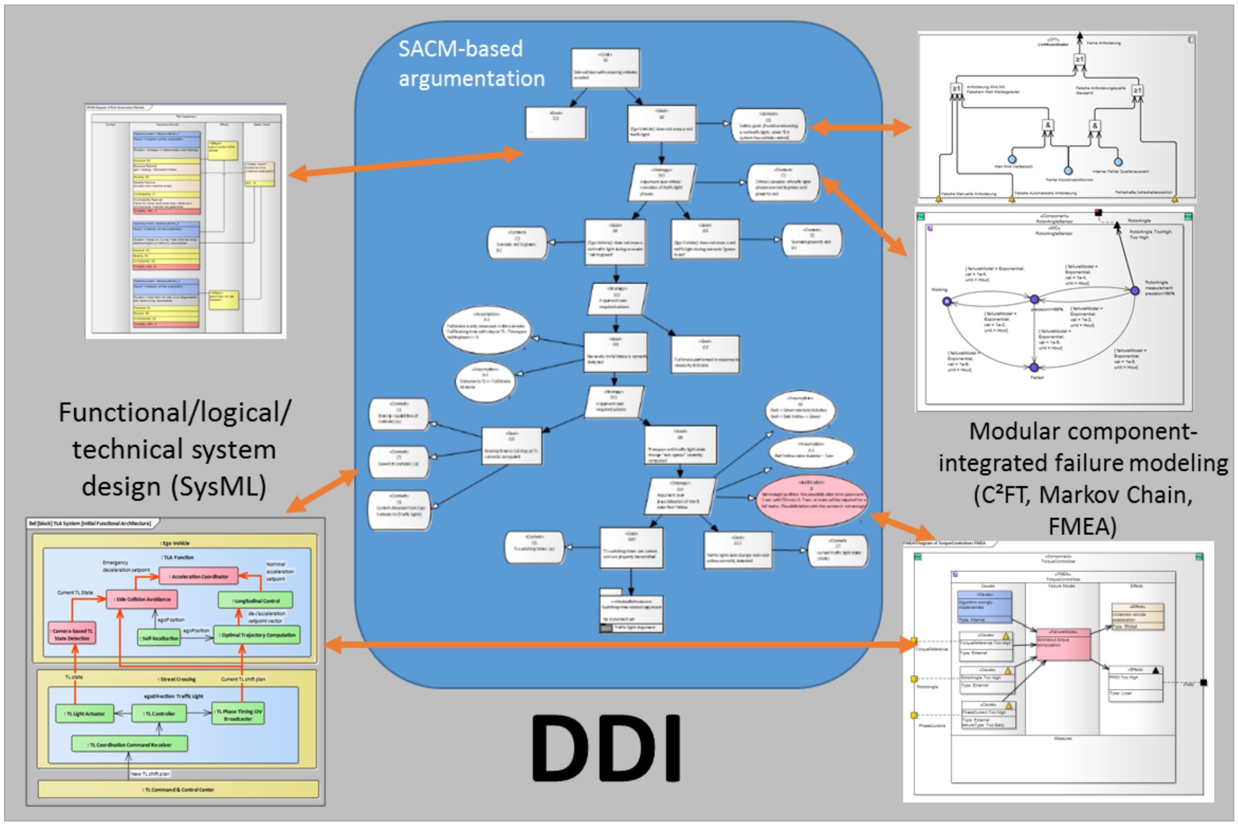

SACM-based argumentation
Functional/logical/ technical system design (SysML)
Modular component-integrated failure modeling (C²FT, Markov Chain, FMEA)
DDI

Figure 1 - SACM-based safety argumentation as the backbone of a DDI

The integration of different systems, each equipped with a DDI, is then also done by linking the modularised assurance case fragments of the systems (cf. Figure 2). This means that in an integration scenario, there is no direct linkage between any other safety engineering artefacts (e.g., failure logic models) beyond the argumentation (at least not initially). Everything is interlinked and interrelated via the argumentation structure of the assurance case, which thus constitutes the backbone of all DDIs (regardless of whether the DDI is a constituent system DDI or an integrated system-of-systems DDI). On this basis, it is also relatively easy to integrate systems where different failure logic modelling techniques such as HiP-HOPS and CFTs have been used.

In order to enable semi-automated (or even fully automated) integration of DDIs, it is necessary to formalise the interfaces of the assurance case fragments sufficiently. Moreover, it is important to enable a certain extent of flexibility because the assured properties given by a constituent system DDI might not fit exactly with what is demanded by the superordinate assurance case of the integrating system. Here ConSerts provide a good starting point, even though the flexibility enabled by ConSerts is still not as good as we would expect for the development time "white-box" DDI integration scenario. Here we would like to achieve deeper integration between the different safety engineering artefacts so that, for instance, a change in the architecture at one point of a constituent system would propagate through different channels (e.g., failure logic models) and the integrator would see the impact for the overall integrated system on the level of its safety guarantees. Alternatively, in the grey-box case, the DDI of a constituent system may provide a bundle of variants that can be switched by the integrator. Even though the details are masked due to IP protection, the different variants exhibit different properties at the assurance case interface. Based thereon, optimisation could be performed to find an overall system configuration for the integrating system (i.e., resolving the variants in the supplier systems) that is optimal with respect to dependability, cost and performance.

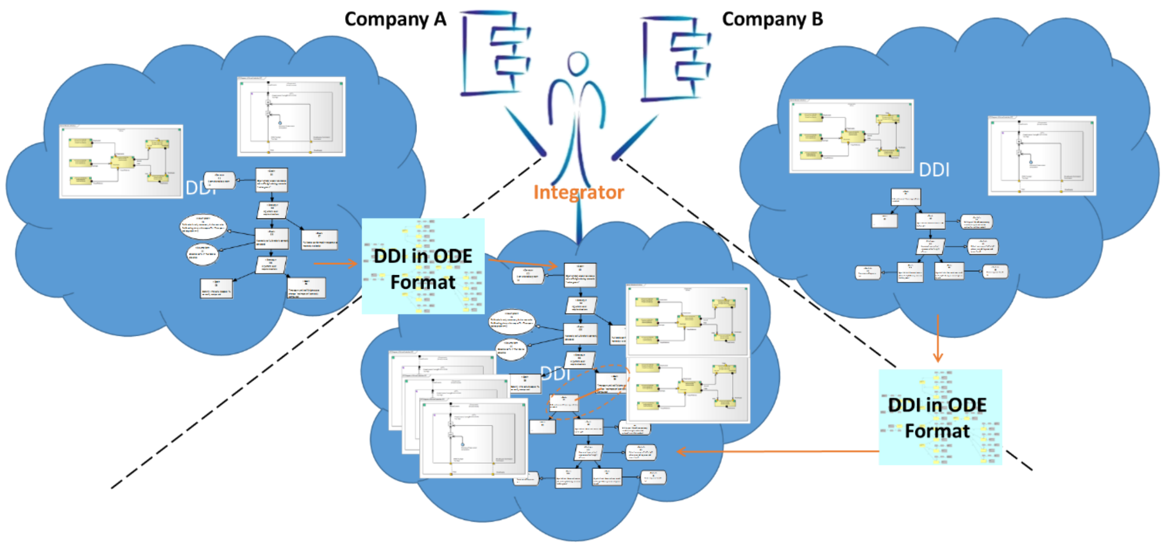


Figure 2 - Integration scenario where the assurance case models are the backbone of the DDI

The examples and elaborations above reveal that an assurance case argumentation is well-suited as a central artefact of a DDI, but that other aspects obviously need to be covered as well. Thus, the ODE cannot just be a slightly adapted version of the SACM, but rather needs to be a set of interlinked meta-models covering all relevant dependability concerns. Still, we chose to use the SACM as the DDI interface language for the reasons mentioned above (standard, acceptance, adoption due to potential widespread tool support). In addition, there is a mechanism within the SACM that could be utilised to this end: The so-called terminology package allows arbitrary information to be referenced in an assurance case (in the form of SACM Expressions/Terms/Categories). In this sense, the SACM is able to link to models (which may contain system information, FME(D)A, FTA, dependability requirement models etc.). One may choose to use either a weak link or a strong link. For weak links, the SACM can simply point to the referred model with text. For strong links, with the help of the facilities provided by the SACM, such information can be retrieved automatically from the referenced models (by using a model querying language such as the Object Constraint Language or the Epsilon Object Language). Thus, the SACM can link (either via weak links using text or via strong links using queries) models which conform to heterogeneous meta-models in order to extract relevant information.

2.1.2 Modifications and Extensions since the initial DDI concept

Since the first version of the documented DDI concept, the concept has been evolved in several different topic directions, which led to modifications and extensions to the ODE meta-model. This section summarizes the different lines of evolution that occurred during the creation of the engineering framework for DDI. The general objective of this activity has been to enable semi-automated engineering support for the synthesis and integration of DDI. Therefore, the ODE needed to be reviewed for issues, which have either not been covered as of ODEv1 or prohibit automation support to be built in.

Refinement of roles of ODE packages and SACM

The ODEv1 treated the assurance case expressed in SACM as a conceptual backbone of the DDI. However, this was not reflected properly in the technical specification of the ODE, where the SACM packages were expressed on the same level as the other dependability aspects. In ODEv2, the roles of SACM and the ODE packages have been revised to reflect the conceptual importance of assurance cases as a root in the DDI structure (see Figure 3 and Figure 4). Details on this aspect can be found in section 2.2.

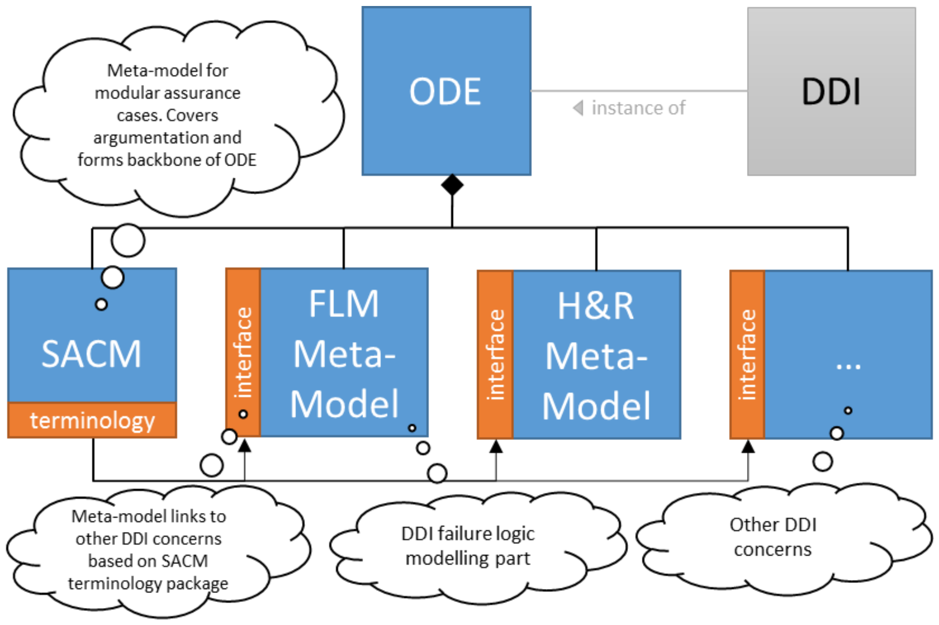

Meta-model for modular assurance cases. Covers argumentation and forms backbone of ODE
ODE
instance of
DDI
SACM
terminology
interface
FLM Meta-Model
interface
H&R Meta-Model
interface
...
Meta-model links to other DDI concerns based on SACM terminology package
DDI failure logic modelling part
Other DDI concerns

Figure 3 - ODEv1 high-level DDI structure

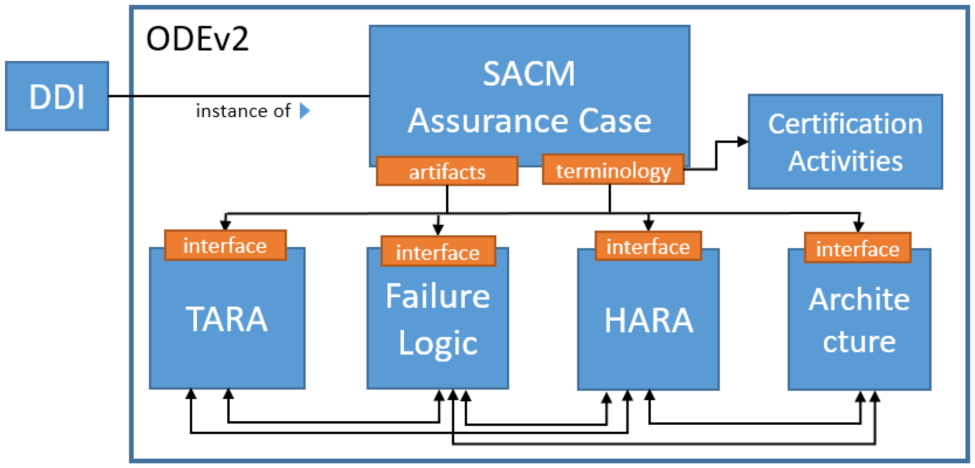


Figure 4 - ODEv2 high-level DDI structure

Identified need to express certification activity models

In ODEv1, the meta-model packages included product-related model types such as architecture, failure logic or hazard models. These describe dependability properties of the system and thus provide a rich basis for supporting dependability assurance claims. However, when it comes to semi- or fully automated processing of these models to synthesize evidence or integrate assurance cases, the following question arises: *how do these models have to be processed to achieve the desired result*? In order to fill in this gap, the ODEv2 contains a package, which allows for the specification of certification activities that should be finally performed by a machine to (semi-)automate the dependability engineering and assessment process. Conceptual details on this aspect can be found in sections 2.2 and 3.4 and its utilization in concrete engineering scenarios is demonstrated in section 4.1 based on the European Train Control System use case (ETCS).

Added package for assessing safety-critical security threats and supporting the satisfaction of security requirements, e.g. the general data protection regulations (GDPR)

Due to the fact that openness and connectivity is a key characteristic of cyber-physical systems, security aspects play an important role for assuring their dependability. Thus, DDIs need to contain models for expressing and assessing the impact of security threats on system safety. In addition, the general data protection regulations (GDPR) make it mandatory to demonstrate classical security properties such as privacy or confidentiality. To account for these issues in the DDI concept, a new ODE package for so-called Threat and Risk Analysis (TARA) has been added. Conceptual details on this aspect can be found in Section 3.3.4 and its utilization in concrete engineering scenarios is demonstrated in Sections 4.2 and 4.3.

Harmonization of ODE package meta-models with commonly used aspect meta-models

An important requirement of the ODE meta-model is its flexibility with respect to adding new packages or replacing certain aspects such as the architecture modelling language, without affecting the core functionality of the DDI. Since there exists a variety of different commonly known meta-model languages for certain aspects covered by the ODE, initial efforts have been carried out to harmonize the ODE package contents with existing standards (such as OMG SysML) or reference meta-models such as EAST-ADL or SafeML. Details on this aspect can be found in Section 3.5.2.

2.2 Semi-automated Engineering of DDIs

2.2.1 The challenge

The focus of the first version of the ODE meta-model was to technically integrate meta-models for dependability aspects such as hazard models, failure logic models and architecture models that were formerly only loosely or not interrelated at all. This set of meta-model packages is in essence data models depicting different dependability aspects of the product (i.e. the system) to be developed. Therefore, these models are referred to as *product models* in the following sections.

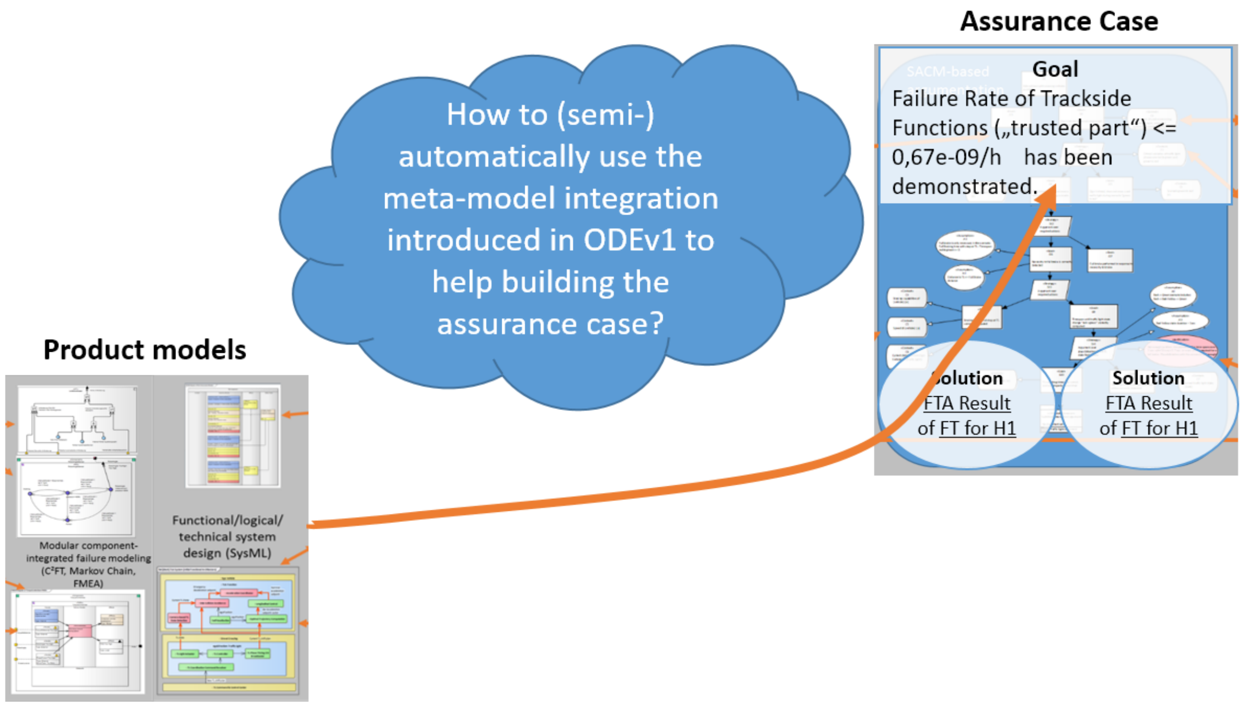

Assurance Case
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
Solution
FTA Result of FT for H1
Solution
FTA Result of FT for H1
How to (semi-) automatically use the meta-model integration introduced in ODEv1 to help building the assurance case?
Product models
Modular component-integrated failure modeling (C²FT, Markov Chain, FMEA)
Functional/logical/ technical system design (SysML)

Figure 5 - Goal of semi-automated DDI engineering

Having these formally integrated product models as a structured source for dependability data, the question arises how to make use of this data to (semi-)automatically generate/modify/integrate the assurance case that is the backbone of the DDI (see Figure 5). Depending on the assurance task to be accomplished, different parts of different dependability aspects might be used to synthesize new information (e.g. evidence) or to automatically instantiate new assurance case structures based on existing dependability models.

The remainder of this section is structured as follows: In Section 2.2.2 we will illustrate different engineering challenges that are enabled by the DDI concept. Afterwards, Section 2.2.3 presents the essential conceptual ideas of the new version of the ODE meta-model that help solve the outlined challenge by enabling the (semi-)automated support for the engineering tasks. Sections 2.2.4 and 2.2.5 provide more detail regarding the technical solution, i.e. what the interface between product models and the assurance case as well as an action language can look like that enables fully automated reasoning based on DDIs. Finally, Section 2.2.6 discusses the differences between DDI usage at design time versus DDI usage at runtime and highlights the specific extension points that will have to be tackled when concepts for runtime dependability certification should be enabled.

2.2.2 Engineering tasks to be supported by DDIs

In order to be able to identify engineering tasks that are supported by partial or full automation, it is required to anticipate a certain development process in which the engineering tasks are embedded. This does not only include a sequential order, in which these tasks are carried out, but should also respect typical supply chains dictating development interfaces being naturally existent through organizational boundaries.

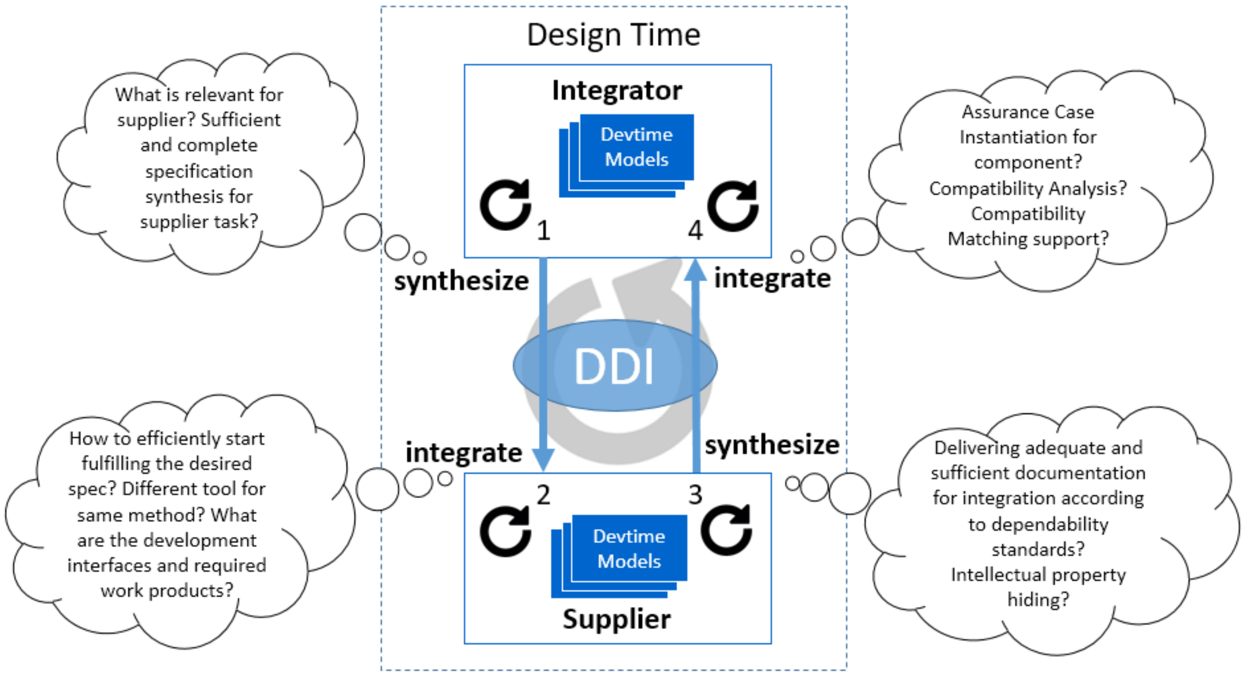

Design Time
Integrator
Devtime Models
1
4
synthesize
integrate
DDI
integrate
synthesize
2
3
Devtime Models
Supplier
What is relevant for supplier? Sufficient and complete specification synthesis for supplier task?
Assurance Case Instantiation for component? Compatibility Analysis? Compatibility Matching support?
How to efficiently start fulfilling the desired spec? Different tool for same method? What are the development interfaces and required work products?
Delivering adequate and sufficient documentation for integration according to dependability standards? Intellectual property hiding?

Figure 6 - Anticipated design time engineering process

Figure 6 shows an abstract development setting that is representative for domains such as railway or automotive. There is an integrator company (referred to as original equipment manufacturer or OEM in the automotive domain) that is building a new system by integrating a set of components that are supplied by supplier companies. This process involves four steps, in which the DDI concept, together with the (semi-)automated engineering support, leads to improvement.

Step 1 – DDI Synthesis @ Integrator

Step 1 involves the synthesis of component specifications that the supplier company must adhere to. One particular challenge of synthesizing this specification is to collect *all relevant information* that is needed by the supplier in order to develop the component in isolation. This is not only necessary for information about required functionality, but also for dependability requirements. In this scenario, DDIs can be seen as a container, where all this information can be captured in an integrated and structured way. Thus, engineering tool support for this step should focus on helping the engineer to collect the relevant information required by the DDI *for the specific tasks* the supplier should carry out. Such specific tasks could be for instance to demonstrate the adequate satisfaction of interface dependability requirements, or to check compatibility of the supplier component interface with the interface definition provided by the integrator.

Step 2 – DDI Integration @ Supplier

Step 2 represents the integration of the specification DDI into the development process of the supplier company. This could mean for example that model stubs are automatically generated based on the development interface extracted from the imported DDI. Such a development interface typically consists of functional or technical data-flow interfaces or dependability requirements allocated to those data-flow interfaces. In advanced scenarios, this could also be placeholders for assurance evidence artefacts that are to be instantiated by the supplier.

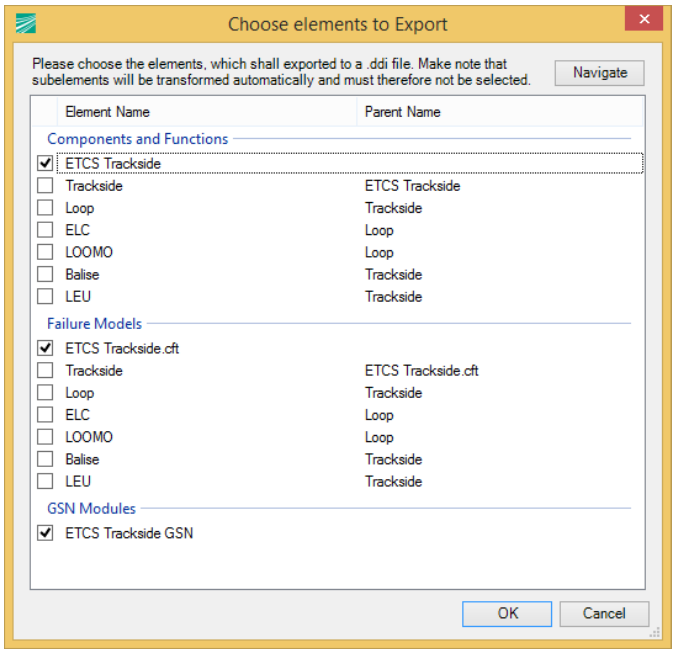
Step 3 – DDI Synthesis @ Supplier

After the DDI has been integrated in a (semi-)automated way, the supplier performs the actual development work until it is time to deliver the component back to the integrator. In this instance delivery means not only the physical component, but also the dependability documentation that is required to build a sound assurance case for the integrated system (typically according to one of the commonly known standards such as ISO 26262 in the automotive domain or CENELEC EN 50129 in the railway domain). Step 3 is concerned with synthesizing a DDI containing *all relevant information* that is needed so that the integrator can properly perform the integration task. From a supplier company perspective, *all relevant* information means explicitly *not all* information and, therefore, engineering support for the supplier should focus on identifying and collecting the minimal set of information to be delivered whilst respecting minimal intellectual property disclosure.

In order to generate a DDI from a model created in a specific tool (e.g. safeTbox by Fraunhofer IESE), the tool-specific aspect models have to be transformed into a DDI. In the current DDI export implementation, the engineer has to manually select the different aspects such as architecture models, failure logic models and safety argument models to be put into the DDI (see Figure 7). In the future, the engineer should only be required to select the component's root element (e.g. Trackside component) and the aspects to be exported (Functional Interface, Abstract failure propagation between interfaces, argument fragments). Afterwards, intelligent automation support collects relevant information and finally produces the DDI with the desired properties.

Automated Evidence Generation

Another supplier engineering task that shall be supported by DDIs is the partially automated generation of evidence for a certain assurance goal. These goals typically come in the shape of interface dependability requirements, such as the one illustrated in Figure 10, where the failure rate of a set of supplied functions (*Trackside Functions*) should be demonstrated to be lower than a target value ($\leq 0{,}67\ e^{-09}/h$). This would be a target value required by the integrator to adequately address system safety goals. The objective of this engineering task would be to employ DDIs for automatic generation of evidence based on a sufficient formal description of 1) What kind of evidence is required and 2) How the evidence is constructed based on aspect models such as failure logic or architecture models.


Choose elements to Export

Please choose the elements, which shall exported to a .ddi file. Make note that subelements will be transformed automatically and must therefore not be selected.

	Element Name	Parent Name
	Components and Functions	
✓	ETCS Trackside	
	Trackside	ETCS Trackside
	Loop	Trackside
	ELC	Loop
	LOOMO	Loop
	Balise	Trackside
	LEU	Trackside
	Failure Models	
✓	ETCS Trackside.cft	
	Trackside	ETCS Trackside.cft
	Loop	Trackside
	ELC	Loop
	LOOMO	Loop
	Balise	Trackside
	LEU	Trackside
	GSN Modules	
✓	ETCS Trackside GSN	

Figure 7 - User interface for exporting a component DDI

Step 4 – DDI Integration @ Integrator

Step 4 is the last step in the development iteration and deals with the integration of a component DDI into the overall system. This engineering task typically involves performing compatibility and structural analyses as well as behavioural compatibility matching with other components. This task needs to be performed for both from both a functional and dependability aspect. Thus, the component dependability documentation has to be integrated into the system assurance case in order to demonstrate confidence in the high-level dependability claims. Engineering support should focus on the (semi-) automated generation and integration of assurance case fragments, i.e. instantiating argument patterns for typical component types or the (semi-)automated assessment of claims in system assurance cases.

Automated Assurance Case Instantiation

The aforementioned (semi-)automated dependability claim assessment is only feasible on an integrated assurance case. Thus, it is necessary beforehand to take the supplied assurance argument fragment, including supporting evidence, and integrate it systematically into a predefined placeholder, which is created before *Step 1* of the development process, where the integrator refined the system specification into compatible component specifications. Such placeholders can be assurance case patterns such as the one shown in Figure 8, where the instantiation strategy is encoded within the pattern. The logic of the instantiation would be in this specific case that for all critical hazards of the system, a goal has to be instantiated for all supplied components which could potentially contribute to those hazards. The integrated dependability data models (ODE-conformant product models in Figure 8) serve as a basis to extract the relevant data for the assurance case instantiation, while the instantiation activity is encoded in the pattern along with a defined logic on how the instantiation should be carried out sequentially.

The goal of this engineering task is the technical integration of an assurance case fragment into a pre-defined placeholder in the integrated assurance case. A fundamental property of this integrated assurance case is the formal relation between assurance claims on the one hand and the product models on the other hand (e.g. architecture, failure logic, hazard models), which served as a source for generating the evidence for building confidence in assurance claims. In this way, traceability is maintained from the assurance case, as a source for required dependability activities, towards those models showing the confidence in a proper activity execution or the quality of its output artefact.

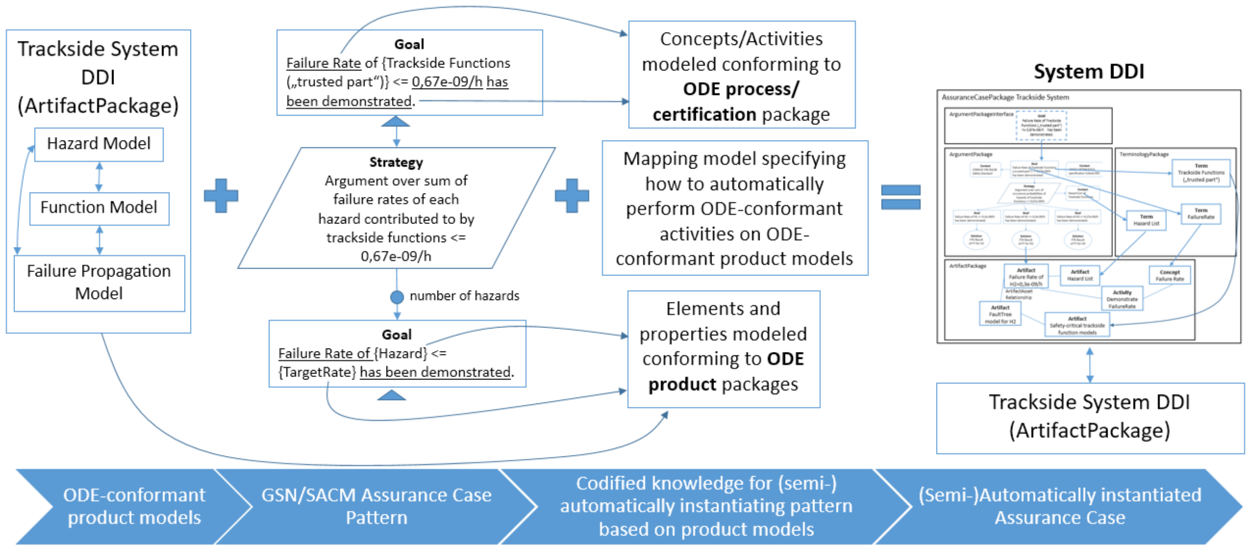

Trackside System DDI (ArtifactPackage)
Hazard Model
Function Model
Failure Propagation Model
Goal
Failure Rate of {Trackside Functions („trusted part")} <= 0,67e-09/h has been demonstrated.
Strategy
Argument over sum of failure rates of each hazard contributed to by trackside functions <= 0,67e-09/h
number of hazards
Goal
Failure Rate of {Hazard} <= {TargetRate} has been demonstrated.
Concepts/Activities modeled conforming to ODE process/ certification package
Mapping model specifying how to automatically perform ODE-conformant activities on ODE-conformant product models
Elements and properties modeled conforming to ODE product packages
System DDI
AssuranceCasePackage Trackside System
Trackside System DDI (ArtifactPackage)
ODE-conformant product models
GSN/SACM Assurance Case Pattern
Codified knowledge for (semi-) automatically instantiating pattern based on product models
(Semi-)Automatically instantiated Assurance Case

Figure 8 - (Semi-)automated assurance case instantiation with DDIs

Automated Claim Assessment

After the integrator has received all DDIs by all component suppliers, a dependability engineering task is to assess whether the provided argumentation and evidence yields enough confidence in the validity of system level assurance claims. One such claim could be for instance to demonstrate that the system safety goals are adequately implemented through the interplay of all supplied components.

In order to assess the adequacy of evidence and argumentation, with partial or full automation support, the engineering support for DDIs shall make use of the aforementioned formal interrelation of assurance case models and product models. It is envisioned that DDI engineering tools can perform analyses to assess how changes in any of the DDI product models propagate up to the assurance case, and indicate to the dependability engineer the impact of the changes on the validity of claims.

Step 5 – Iterative development

As realistic development endeavours do not follow the waterfall process model, but are executed iteratively, steps 1-4 are anticipated to be repeated multiple times, thereby refining DDIs gradually. In consequence, automation support can for instance be given for identifying repeated integration of the same component DDI and offering a summary of changes in order to make impact analyses more efficient.

2.2.3 How DDIs enable (semi-)automated engineering based on the ODEv2

Having illustrated in Section 2.2.2 some design time engineering tasks, which greatly benefit from DDIs, the upcoming subsections of section 2.2 explain the core principles of the ODE meta-model version 2 enabling those benefits.

In order to realize the vision of automated generation and evaluation of a dynamic assurance case, the following four essential ingredients are necessary:

1. A language that is capable of expressing the assurance case (=the argument and the evidence to support the top-level claims) with sufficient level of formalism;
2. Semantics that describe how the product models representing the evidence artifacts formally relate to each other;
3. Semantics that precisely describe the nature of dependability concepts and associated activities that allow for the assessment and generation of argument structures (including evidence), by performing defined operations on the interrelated product models;
4. Means for orchestrating ingredients 1-3 to (semi-)automatically execute the engineering tasks associated to assurance case synthesis and integration.

Figure 9 shows ingredients 1-3 in action for an example assurance claim taken from the European Train Control System (ETCS) use case.

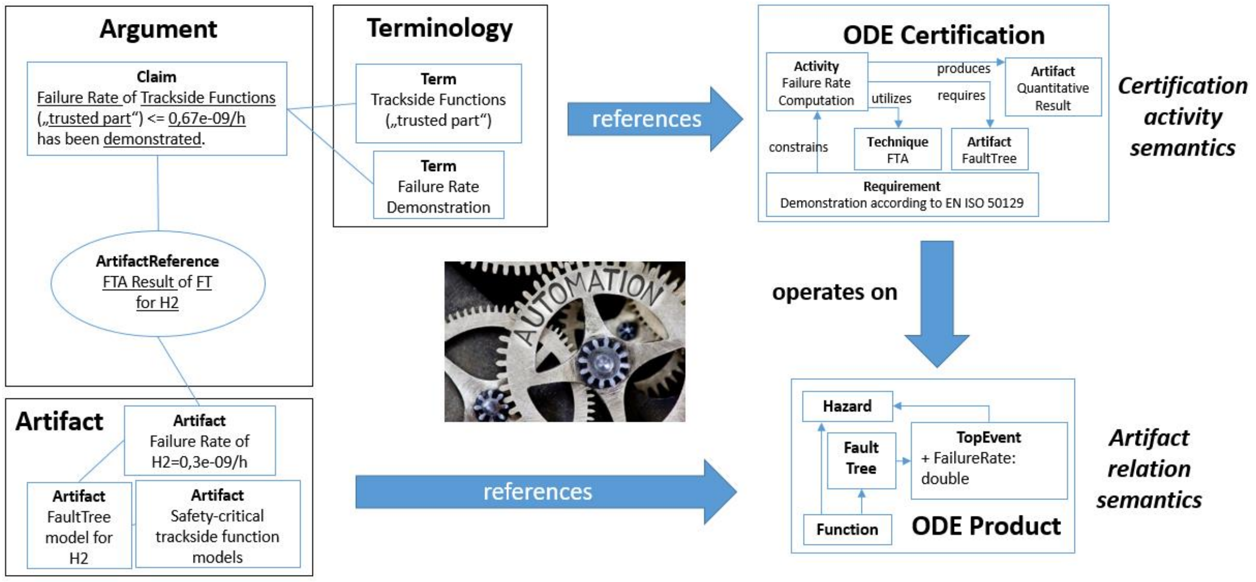


Figure 9 - An example DDI: Enriching the assurance case with process and product semantics

Ingredient 1 - SACM as assurance case definition language

The different parts of an assurance case fragment expressed in the Structured Assurance Case Meta-Model (SACM) is shown to the left of Figure 9. The argument package describes the argument that typically refines top-level claims, in a systematic and understandable way, into sub-claims until such time as a sub-claim can be supported by concrete evidence (i.e. one or a set of artefacts). Note that the claim in Figure 9 represents a leaf claim and the higher-level argument is omitted for clarity.

The argument package refers to concrete artefacts organized in the artefact package by using the *ArtifactReference* element. Note that the relations depicted between artefacts represent abstract relations, i.e. SACM does not specify semantics for the artefact relation, since it claims to be an exchange format leaving the definition of semantics open by design. *Artefact* elements play the role of placeholders for concrete evidence material, which typically occur in very diverse shapes, e.g. documents, reports, process descriptions, models, etc.. Therefore, *artefacts* offer a mechanism to reference the concrete external materials to establish traceability.

The terminology package allows the formalisation of expressions and terms, which are used as ingredients to define claims. While *expressions* are a means to realize structured language (i.e. reusable text blocks) to achieve a certain level of conformity, *terms* additionally incorporate means to reference external material such as documents or models referring to a definition of more precise semantics to explain, what the *term* should represent within the claim. More detailed information on SACM as well as the referencing mechanisms can be found in sections 2.2.4 and 3.2.

Ingredient 2 – The ODE product meta-model packages for defining artefact relation semantics

In order to provide (semi-)automatic engineering support for processing existing evidence artefacts, it is crucial to enable the respective algorithms to understand how those artefacts relate to each other. Within the DDI, artefacts that relate to the dependability aspects of systems engineering are grouped and integrated. Note that the dependability models typically model different aspects of the same system and therefore it is important to make the relations between the aspects explicit. An example for the semantical relation between different aspects is the relation between a function (behavioural design aspect of a system) and a hazard (a state of the system realizing the function potentially leading to an accident). By making such relations explicit in the ODE product models, it is possible to programmatically navigate from functions to hazards, which can be seen as a very basic requirement to (semi-)automatic reasoning about adequate mitigation of hazards.

As of ODEv2, the ODE product meta-model contains integrated meta-model packages for the following aspects: architecture design (ODE::Design, Section 3.3.1), hazard and risk analyses and dependability requirements modelling (ODE::HARA, Section 3.3.2), failure logic modelling with fault trees, failure modes and effect analyses (FMEA) and Markov chains (ODE::FailureLogic, Section 3.3.3) as well as security-related threat and risk analyses (ODE::TARA, Section 3.3.4). Note that the failure logic modelling package also incorporates means for security-related attack tree modelling as the structure for both model is similar.

Note that the dependability aspects covered by ODEv2 do not cover all possible aspects required to instantiate and evaluate a complete assurance case of a cyber-physical system. Therefore, the DDI mechanisms have been designed specifically to be open for extension, i.e. for adding new aspects as the need for them emerges. It is expected that for dependability assurance of cyber-physical systems (of systems), new types of claims will be required to be assured with confidence, in particular those related to runtime assurance approaches. Since these are still subject of research, the idea is to react to such new emerging claims with new ODE product packages to specifically support the (semi-)automated assurance of those new claims. In this way, the DDI contents and capabilities will be able to evolve in line with future assurance demands.

Ingredient 3 – The ODE certification meta-model packages for defining certification activity semantics

The SACM exchange format, together with the ODE product meta-models, build a solid formal data basis for the definition of an assurance case with a formal traceability to evidence produced from dependability aspect models. One missing ingredient for enabling (semi-)automated processing of the DDI is the specification of what DDI shall be processed, and in what sequence, to achieve the desired result. The desired result is to help the engineer with the generation of useful new information (e.g. the validity of claims, instantiated assurance case fragments, …) during the execution of a specific engineering task.

The ODE certification meta-model aims to provide a suitable means for modelling engineering *activities* that consume *artefacts* of different dependability aspects (ODE product models), or assurance case elements (SACM), and transform them into new information by means of defined *techniques* and activity step descriptions. Such new information can be for instance a Boolean result about the validity of a claim, or a new model such as an integrated assurance case fragment.

Figure 10 exemplifies the interplay of assurance case, certification model and product model by depicting an automated claim validity assessment activity for a failure rate demonstration claim.

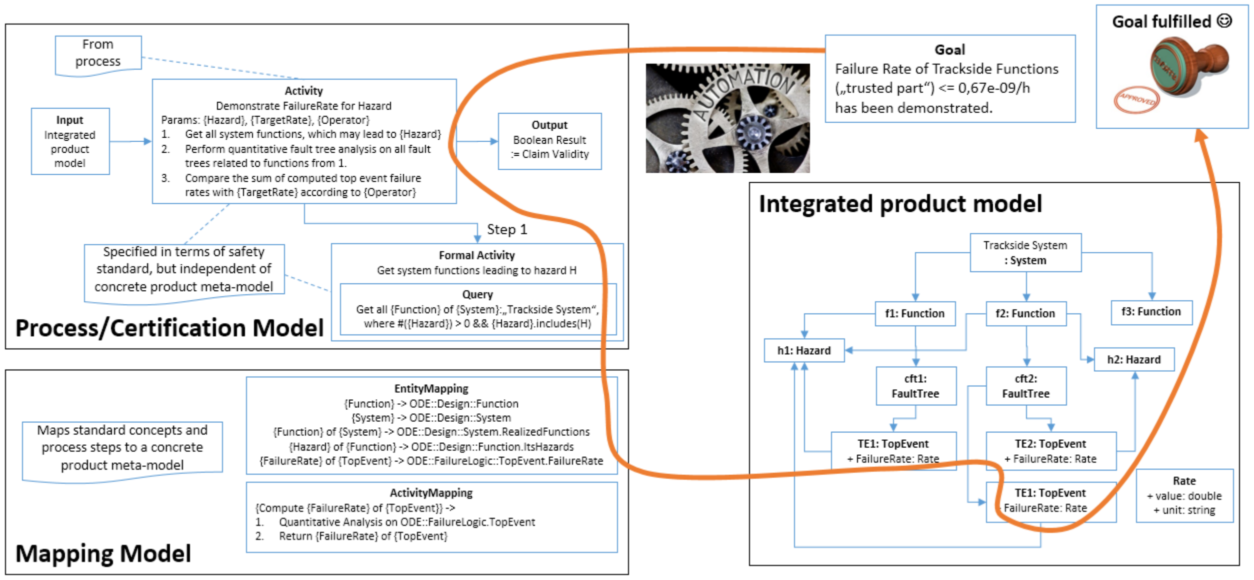

From process
Activity
Demonstrate FailureRate for Hazard
Params: {Hazard}, {TargetRate}, {Operator}
1. Get all system functions, which may lead to {Hazard}
2. Perform quantitative fault tree analysis on all fault trees related to functions from 1.
3. Compare the sum of computed top event failure rates with {TargetRate} according to {Operator}
Input
Integrated product model
Output
Boolean Result := Claim Validity
Step 1
Specified in terms of safety standard, but independent of concrete product meta-model
Formal Activity
Get system functions leading to hazard H
Query
Get all {Function} of {System}:„Trackside System“, where #({Hazard}) > 0 && {Hazard}.includes(H)
Process/Certification Model
Maps standard concepts and process steps to a concrete product meta-model
EntityMapping
{Function} -> ODE::Design::Function
{System} -> ODE::Design::System
{Function} of {System} -> ODE::Design::System.RealizedFunctions
{Hazard} of {Function} -> ODE::Design::Function.ItsHazards
{FailureRate} of {TopEvent} -> ODE::FailureLogic::TopEvent.FailureRate
ActivityMapping
{Compute {FailureRate} of {TopEvent}} ->
1. Quantitative Analysis on ODE::FailureLogic.TopEvent
2. Return {FailureRate} of {TopEvent}
Mapping Model
AUTOMATION
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
Goal fulfilled ☺
APPROVED
Integrated product model
Trackside System : System
f1: Function
f2: Function
f3: Function
h1: Hazard
h2: Hazard
cft1: FaultTree
cft2: FaultTree
TE1: TopEvent + FailureRate: Rate
TE2: TopEvent + FailureRate: Rate
TE1: TopEvent FailureRate: Rate
Rate + value: double + unit: string

Figure 10 - Automated claim validity assessment activity

The activity model contains two representations of the desired engineering activity *Demonstrate FailureRate for Hazard*: One describes the processed modelling elements (*Hazard, TargetRate, (Comparation)Operator "<=")* mapping to certain formalized concepts of the dependability domain, as well as the different steps of the activity in natural language. The other representation of the activity is a formal one that can be understood and processed by a machine automatically. The intention behind providing both representations is twofold: One the one hand, those engineers modelling and reviewing the activities must be confident that the formal representation is indeed matching the intention of the activity. Process engineers are seldom experts for formal languages and as such, a human understandable format is beneficial. On the other hand, dependability engineers who should be supported by the automation, have to effectively understand *how* the results of the automation are produced in order to judge the adequacy of using it in a particular context.

In fact, the parameters of an engineering activity are referring to either model elements of the product models (e.g. *Hazard, Function, System* in Figure 10), or to steps that are performed on these model elements (e.g. navigation, filtering, reading properties, comparing properties with values). While the latter ones are expressed in the activity itself (*Get all …, where …*), the former ones have to be mapped to the ODE product package elements explicitly.

One may ask why this mapping is necessary as the activity parameters are equal to the product model elements depicted on the right side. This is due to the fact that certain dependability aspects (such as the concept of a hazard or a function) are quite stable across a domain, but its realization within a specific product meta-model might differ from company to company, or depend on the modelling language used (e.g. SysML vs. UML). To illustrate this, let us look at an example of the most basic difference between different meta-model representations of the same domain concept: The name of the meta-model element for describing a link between two architectural ports of a system. Still being the same concept, there may exist a variety of meta-model element names for it such as *Connector, Link, Connection, Propagation,* etc. Although the example explains the mechanism for a non-dependability concept, it is particularly useful for dependability domain concepts, as no commonly accepted meta-modelling languages similar to SysML for system design modelling exist for dependability aspects to date.

To account for this diversity in modelling similar concepts differently across languages and companies, the ODE contains a mapping mechanism aiming at decoupling concrete product meta-models from the activities operating on them, which should be specified in terms of domain concept placeholders. If new modelling languages will emerge known dependability domain concepts differently, already specified activities can be easily reused by providing a mapping model relating domain concept placeholders to product meta-model elements. Sometimes this process is also referred to as *aspect weaving* in literature. Illustratively spoken, this means the mapping model allows for the weaving of new product model elements into the already existing domain-specific activity models without having to remodel the activity.

As of ODEv2, no concrete language meta-model has been produced or selected that allows modelling of activities in the depicted way. However, the required properties of such a language have been outlined and the DEIS consortium sees a good fit with the concepts of the *Common Assurance and Certification Meta-*

model developed within the European AMASS project[1]. More details on this aspect can be found in Section 3.4.

Ingredient 4 – Means to (semi-)automatically execute engineering activities on DDIs

Having described the three core parts of the ODE which represent the anatomy of a DDI (see Figure 11), there is still one missing piece left to enable the DDI to do useful things for the engineer: A higher-level mechanism that is capable of orchestrating the execution of certification activities on a set of DDIs. This mechanism might seem similar to the certification activity itself, but the ODE certification packages provide means for the *definition* of activities, while the mechanism described in this section is rather a concrete execution environment.

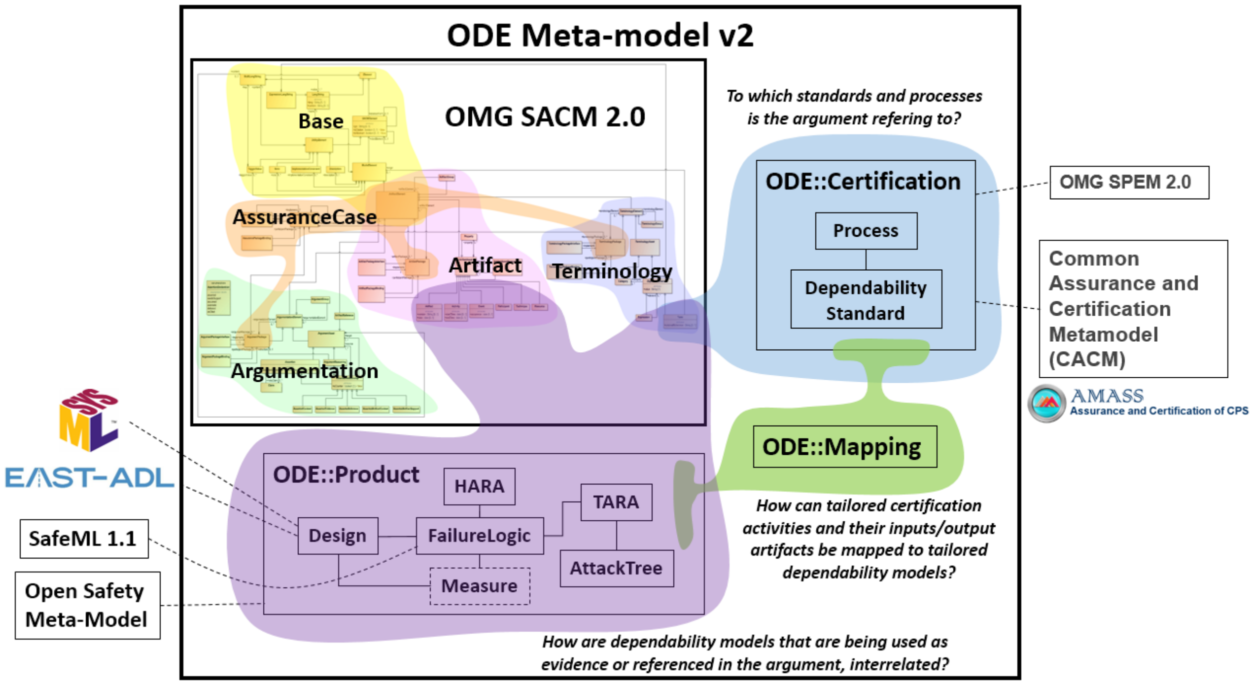

ODE Meta-model v2
OMG SACM 2.0
Base
AssuranceCase
Artifact
Terminology
Argumentation
To which standards and processes is the argument refering to?
ODE::Certification
Process
Dependability Standard
OMG SPEM 2.0
Common Assurance and Certification Metamodel (CACM)
AMASS
Assurance and Certification of CPS
ODE::Mapping
How can tailored certification activities and their inputs/output artifacts be mapped to tailored dependability models?
ODE::Product
HARA
TARA
Design
FailureLogic
AttackTree
Measure
EAST-ADL
SafeML 1.1
Open Safety Meta-Model
How are dependability models that are being used as evidence or referenced in the argument, interrelated?

Figure 11 - The Open Dependability Exchange Meta-Model (ODE) v2

This DDI execution component allows for the (semi-)automatically execution of dependability certification activities (defined as activity models conforming to packages in ODE::Certification), which operate on dependability aspect models (conforming to packages of ODE::Product) in order to synthesise, integrate or assess a dependability assurance case (conforming to OMG SACM).

Technically, the DDI execution component is envisioned to have an underlying computational model that can execute DDI certification activities with different degrees of formalism: This means that, at design time a lower level of formalism is required as the human dependability engineer can still serve as backup for getting information or decisions that are too difficult to evaluate in a fully automated manner. At runtime however, full formalism will be required as all decisions have to be computed by machines. Some of the consortiums initial ideas on what a DDI execution component can look like are elaborated in Section 2.2.5.

2.2.4 Interfaces of SACM to define artefact and process semantics

The Structured Assurance Case Metamodel (SACM) is a specification issued by the Object Management Group (OMG). SACM is developed by the specifiers of existing system assurance approaches (e.g. GSN and CAE), based on the collective knowledge and experiences of safety and/or safety practitioners. SACM provides a richer set of features in addition to the features provided by GSN and CAE, such as fine-grained modularity, controlled vocabulary, and argument-evidence traceability.

SACM is organised in five fundamental packages, as shown in Figure 12 below. The Base package provides a rich sets of features on atomic model elements of SACM which will be explained later. SACM organises Assurance cases in AssuranceCase packages. An AssuranceCase package contains several Argumentation packages, Artifact packages and Terminology packages.

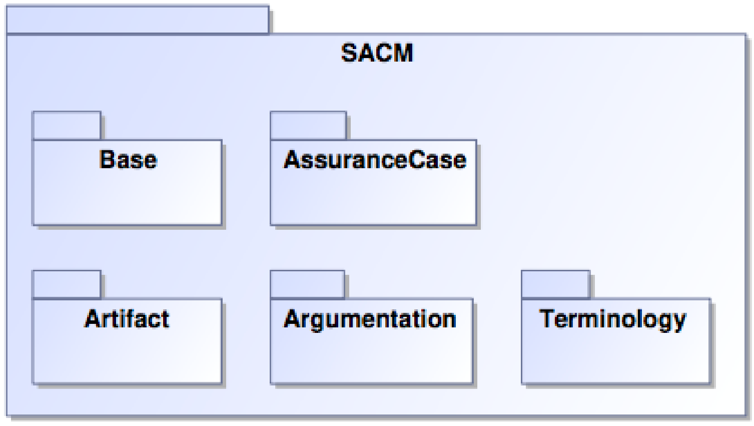


Figure 12 – SACM fundamental packages

Argumentation packages store information about the argumentation part of an assurance case, where safety/security claims are broken down into sub claims until they are directly backed by evidence.

Evidence used in the argument packages can be modelled and organised in artefact packages. For example, a hazard analysis model can be recorded in an artefact package, where the user may also specify when the analysis is performed, who participated in the analysis process and what techniques are used in the process.

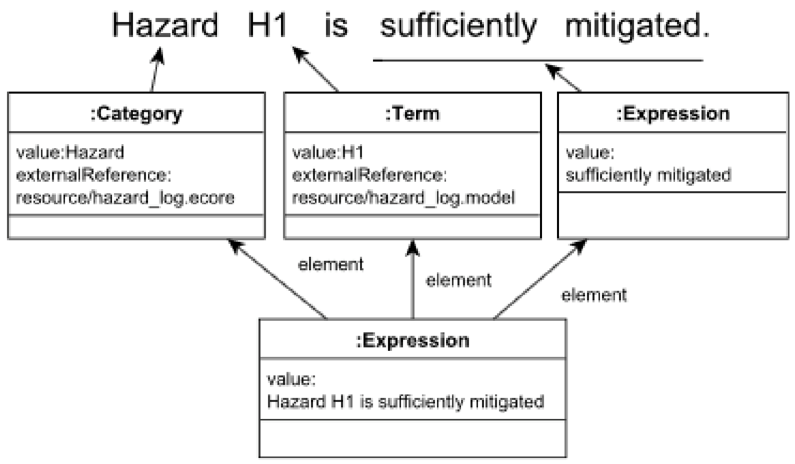


Figure 13 - SACM terminology package example

SACM also provides the mechanisms to create controlled natural languages so that the users can establish a finer grade of reference to system models. The Terminology package of SACM provides the mechanism to create *Expressions*, *Categories* and *Terms*. An example of controlled language is shown in Figure 13. The upper part of the figure is a claim: **Hazard H1 is sufficiently mitigated**. In this claim, the user can refer to expression elements in their terminology packages. For example, *Hazard* may refer to a *Category* in the terminology package, which in turn points to a hazard log metamodel through its *externalReference* property. In this way, hazard log metamodel provides a definition of what a *Hazard* is. Then, hazard *H1* can refer to a *Term* in the terminology package, which in turn refers to an instance hazard log model (that conforms to the hazard log metamodel). The hazard log model may then contain information on how H1 is identified, its cases and consequences, etc. The *Expression* **sufficiently mitigated** is recorded in the terminology package so that it can be reused. The user is also free to add any explanatory information to the *Expression* so that it better explains what **sufficiently mitigated** means.

Finally, an overall *Expression* which references the three previous elements is created. This expression can be referenced in the argumentation package (e.g. as a description of a Claim).

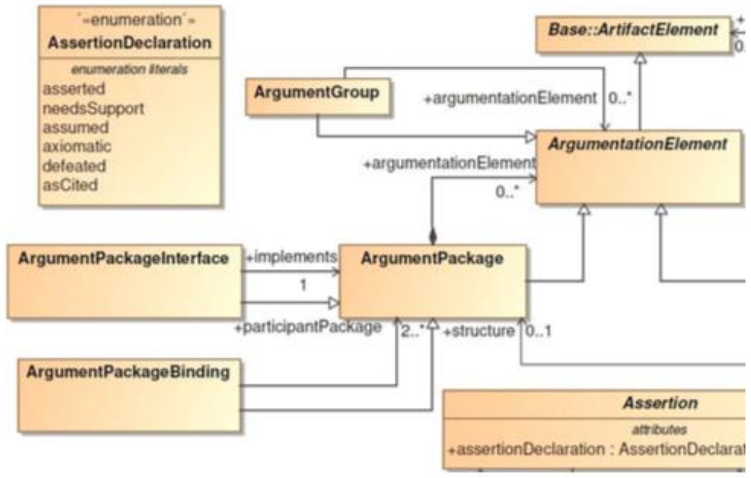


Figure 14 - SACM argument package segment

SACM promotes modularity, in the sense that elements are organised in different packages. To refine modularity, SACM provides three different types of packages. Figure 14 shows a segment of the meta-model for the argument package of SACM, illustrating the three types of packages in the argumentation package of SACM. ArgumentPackage is the main package in which structured argumentation is stored. The users can disclose part of the argumentation externally with the use of ArgumentPackageInterfaces. To do this, in ArgumentPackageInterfaces, citation elements need to be created which cite to original elements in ArgumentPackages. Figure 15 shows a segment of SACM for the citation mechanism. All SACMElements have the capability of citing other SACMElements via the +citedElement reference. If an element cites another, it automatically becomes abstract and citation via its +isAbstract and +isCitation features. ArgumentPackageInterface only contains citation elements, it should be enforced by constraints on the metamodel.

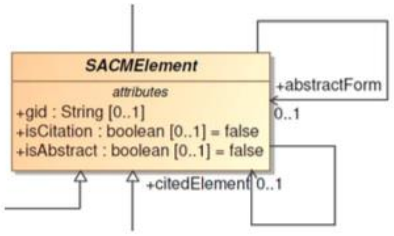


Figure 15 - SACM citation mechanism segment

Whilst SACM provides traceability features between argumentation and evidence, it does not impose a strong traceability link from SACM to other models. This is due to the fact that SACM is a generic assurance case metamodel and is not bound to any specific domains. This can be seen from the mechanisms designed for the Terminology package where externalReferences for *Expressions* are only described as Strings. In this sense, SACM provides a weak link to external models. However, the link is robust since it does not rely on any other metamodels and does not need to evolve with other models if the models referenced in SACM change.

2.2.5 Initial ideas for an action language to enable automated reasoning based on DDIs

To this point, the DDI engineering process has focused on design and development of the necessary infrastructure for creating, managing and exchanging DDIs. The work towards the release of ODE v1 and v2 and the initial engineering tools are indicative of this focus.

To extend the DDI functionality further and introduce increasingly larger degrees of automation as per the initial vision, a computational model which evaluates the DDI is needed (refer to section 2.2.3 for a more detailed motivation). In this section, we present a preliminary description of such a model, which follows directly from the work presented in DEIS so far. Furthermore, the computational model provides a starting point for the advancement of the DDI engineering tool concept.

Under this view, the evaluation of the DDI can be modelled as a Directed Acyclic Graph (DAG); Each node of the graph can represent a section of the DDI being parsed by the evaluation process. An example of such a model is given in Figure 16. In the figure, the DAG represents the instantiation of an abstract assurance claim at the top. In this case the arrows indicate the direction of the support of the claim provided by following nodes. Alternative directions and interpretations are also possible, depending on the context of the computation.

Abstract assurance claims can be modelled in SACM. Such claims abstract from the details of the argumentation (e.g. what is the subject for which the claim is made), its constituent properties and elements etc. Instantiating the claim means replacing the abstract references within the claim with concrete ones, requisitioned from the appropriate ODE elements within the rest of the DDI.

Thus, in the case of Figure 16, the 'Instantiation Script' elements attached to each node of the DAG control how the abstract references in the descriptions, denoted using the '{' '}' brackets, are replaced. Specifically, in the top node, {X} is replaced with 'System A', being the name of the ODE element being referenced. In the supporting node, properties of constituent elements of X form an assertion supporting the previous claim. The constituent elements can be referenced using a universal quantifier that ranges over the domain of constituent elements of X. In practical terms, this can be realized using a programmatic 'loop' to iterate over the constituent elements of X.

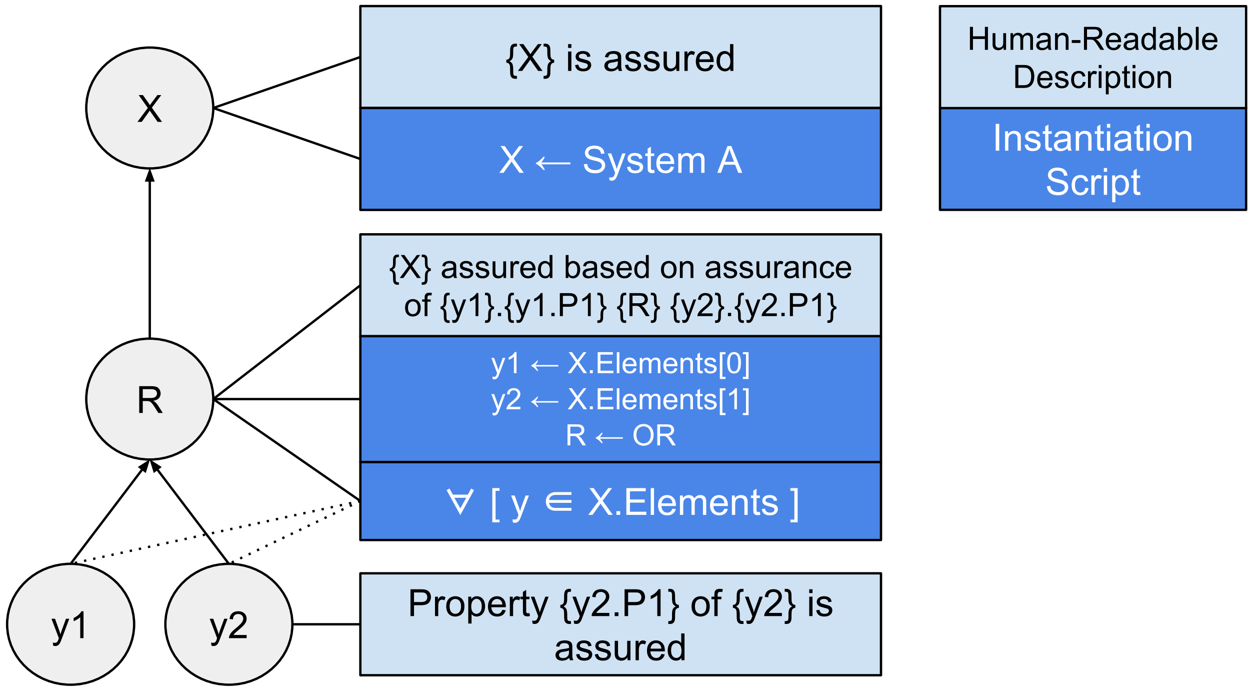

X
R
y1
y2
{X} is assured
X ← System A
Human-Readable Description
Instantiation Script
{X} assured based on assurance of {y1}.{y1.P1} {R} {y2}.{y2.P1}
y1 ← X.Elements[0]
y2 ← X.Elements[1]
R ← OR
∀ [y ∈ X.Elements]
Property {y2.P1} of {y2} is assured

Figure 16 - DDI Instantiation as a DAG

Using this understanding of the DDI evaluation, its support for introducing further automation can be explained. We can immediately consider two scenarios based on the use cases found in DEIS so far.

One scenario is applicable during the development of the Cyber-Physical System (CPS), where an integrator requires constituent subsystems to be integrated into a superordinate system. In this case, the suppliers of the constituent systems also provide their automatically generated DDIs to the integrator. The integrator has designed a DDI for the superordinate system with abstract assurance claims within it, comparable to those in Figure 16. To certify the integrated system, all that is required is to merge the DDIs received with the superordinate DDI and instantiate the combined abstract assurance claim. If the claim is fully instantiated with no errors or missing references, then the integrated system has now been successfully argued to be adequately dependable.

An alternative scenario can be considered where design-time DDIs have been manually created and integrated for a given CPS up to completion of development. Now let us assume this CPS is required to cooperate with other CPSs in the field, with similarly available DDIs. To certify that their combined operation is accepted as adequately dependable (or safe/reliable/secure depending on the context), they need to exchange, merge and evaluate their (appropriate parts of their) combined DDIs. This means that the DAG presented above would build up increasingly larger DAGs, extending the assurance claim appropriately.

The two scenarios essentially differ only in the timing of the evaluation, the former having the evaluation happen during development and the latter during operation. In both cases, there is a need for a computational language which describes the order, the terms and the individual expressions that form each step of the DDI evaluation. Fortunately, the SACM metamodel provides within its Terminology package appropriate structures to model all of the above elements. We are currently researching options for an appropriate programming language for executing the computational model.

2.2.6 Advances regarding runtime dependability assurance

Although it has not been a principal objective in DEIS so far to conceptualize how runtime DDIs will concretely look like, we nevertheless designed the ODE meta-model v2 in a way so that it contains the required foundations for runtime DDIs that will be subject of upcoming work.

Figure 17 is an extension of Figure 6 in that it adds the runtime dimension to the anticipated engineering process. It is envisioned that design time DDIs still form an essential foundation for the synthesis of runtime DDIs, as only a subset of dependability engineering activities will be carried out in a fully automated manner at runtime. Therefore, step 5 in Figure 17 seamlessly continues the design time engineering process by a synthesis step that transforms the set of design time models into their runtime representations. Afterwards, the human engineers are taken out of the loop and all synthesis (step 6) and integration (step 7) activities are from now on only performed automatically by system ECUs.

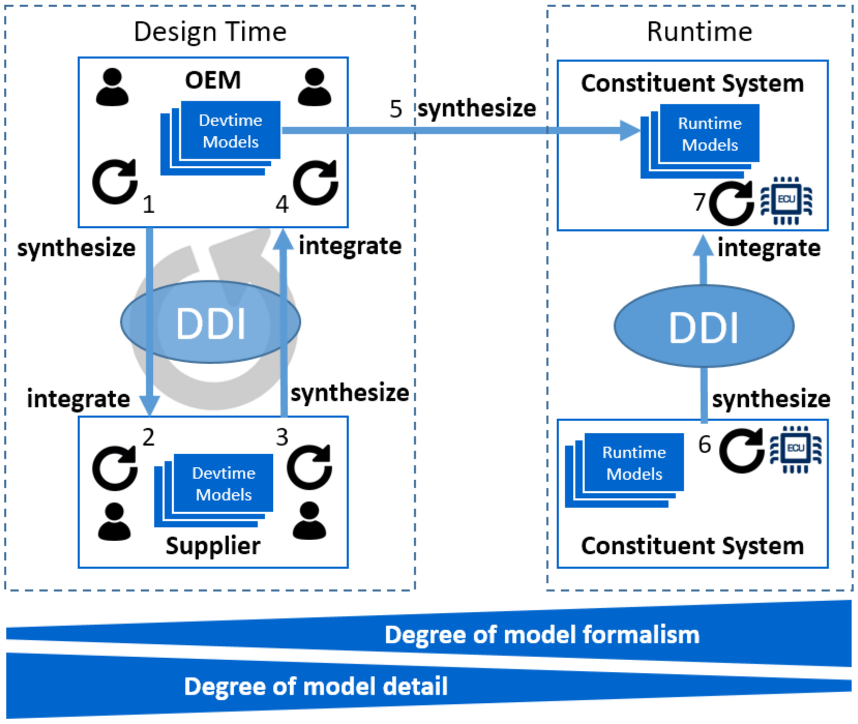


Figure 17 - Transitioning from design time to runtime DDI execution

The kind of models to be used at runtime will not be just fully formalized representations of the design time models such as hazard, failure or architecture models. Instead, runtime scenarios demand different models to enable a dependable operation of cyber-physical systems. In particular, runtime models will allow:

1. to monitor the operational context of the system;
2. to perform a situation-specific risk assessment;

3. to allow context-specific reasoning on dependability properties with the goal to always maintain dependable operation on the one hand and to reach an optimal functional performance on the other hand;
4. to execute adaptation scenarios based on the reasoning performed in 3.

These steps have been already conceptualized in an approach called *Dynamic Safety Management* (DSM) (Trapp, Weiss, & Schneider, 2018), which aims at shifting parts of the safety engineering activities into runtime in order to continuously react to changing context in a dependable way. A first step in this direction were *Conditional Safety Certificates* (ConSerts) (Schneider & Trapp, 2013) that modularize safety interfaces for configurations including variants of safety guarantees with different levels of required confidence. These ConSert models are evaluated at runtime to check, whether a set of configurations of multiple systems exists that allows a dependable operation.

In this way, ConSerts as a first constituent for a dependability runtime model, and Dynamic Safety Management as a more sophisticated approach for executing context-dependent dependability reasoning in general, will be an excellent basis for conceptualizing runtime DDIs.

3 The Open Dependability Exchange Meta-model (ODE) v2

Having introduced the concepts behind the engineering framework for the generation and integration of design time DDIs in section 2, this section contains the reference for the open dependability exchange meta-model in its second version. Note that the textual description limits itself to the changes that have been done since the last ODE meta-model version v1. For the sake of completeness, the contents of all currently available ODE packages are presented graphically.

3.1 The complete ODE meta-model

Figure 18 presents an overview of the ODE v2, encompassing both the SACM (highlighted in purple) and the product meta-models (highlighted in green). The overview indicates that while there has been some reduction and simplification, the ODE remains a quite complex metamodel, spanning across a plethora of metamodeling elements.

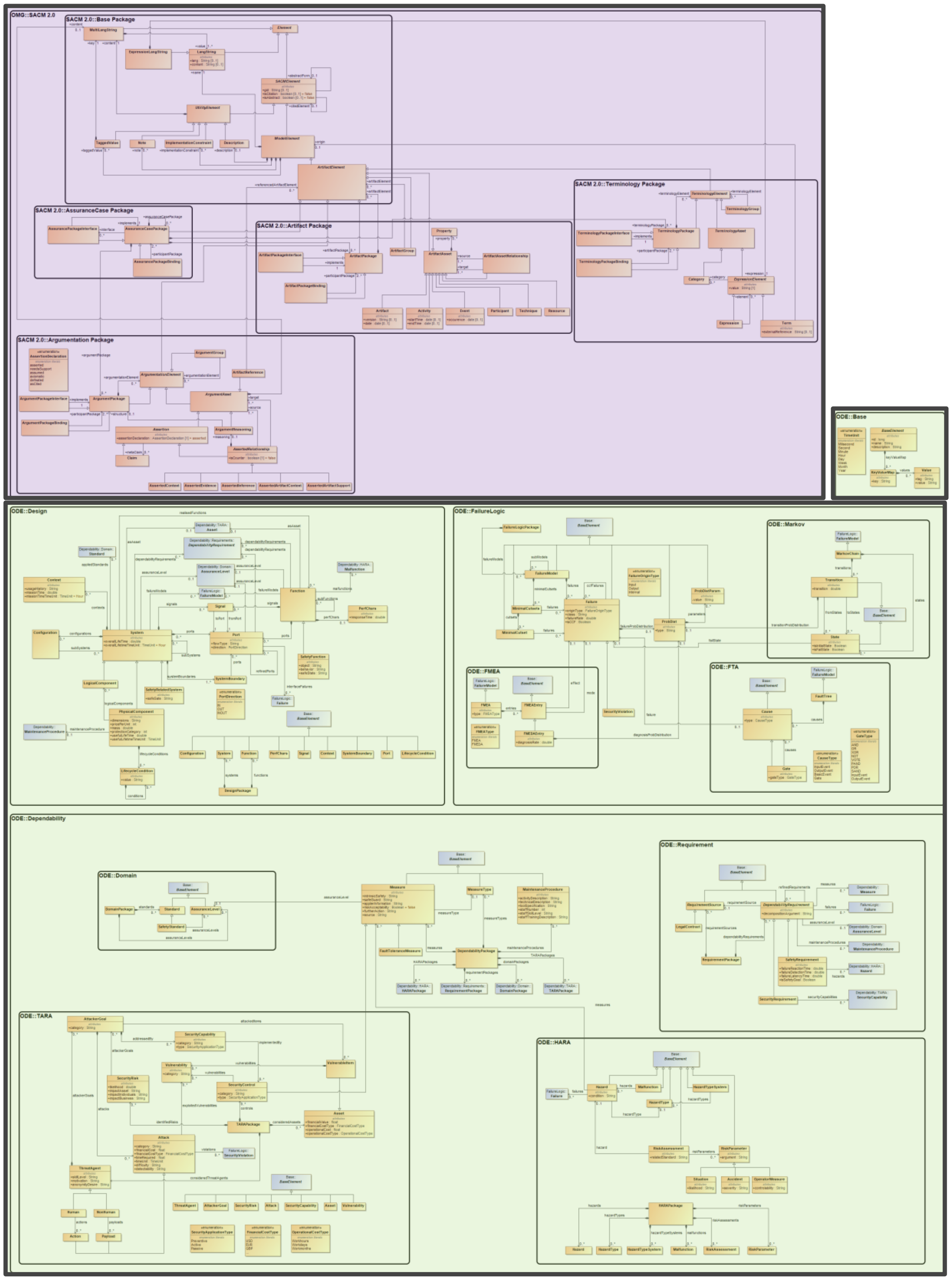


Figure 18 - The complete ODE Meta-model v2

3.2 Structured Assurance Case Meta-Model (SACM) 2.0

The Structured Assurance Case Meta-Model (SACM) is a modelling language specified by the Object Management Group (OMG) to create model-based assurance cases. SACM supports existing system assurance case approaches such as the Goal Structuring Notation (GSN) and Claims-Arguments-Evidence (CAE).

An assurance case is a set of auditable claims, arguments and evidence created to support the claim that a defined system/service will satisfy typical requirements such as safety and/or security. An assurance case in this context is a machine-readable model that facilitates the exchange of information between various systems stakeholders such as suppliers and integrators, and between the operator and regulator, with the knowledge related to the safety and security of the system being communicated in a clear and defendable way. Each assurance case should communicate the scope of the system, the operational context and the safety and/or security arguments, along with the corresponding evidence.

3.2.1 SACM Assurance Case Component

In general, the SACM enables the user to create assurance cases by combining structured argument(s) into *ArgumentPackage*(s) with their corresponding evidence defined in *ArtifactPackage*(s), as well as the controlled vocabularies used within the scope of the assurance case with regards to the information of the system/service for which the assurance case provides assurance for, in *TerminologyPackage*(s). The structure of SACM AssuranceCase component is shown in Figure 19.

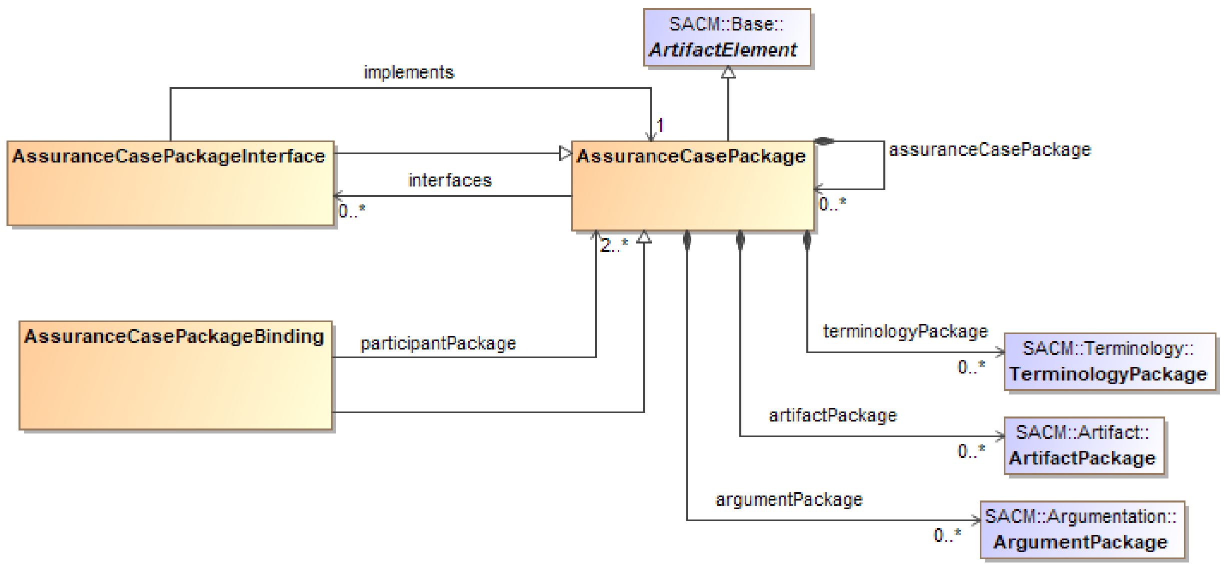


Figure 19 - SACM::AssuranceCase Component

Considering the possibility of exchanging assurance cases (or simply exchanging system information), the SACM provides the notion of Interface. The creator of an assurance case can decide to reveal part of its information by using the *AssuranceCasePackageInterface*. In this sense, systems with SACM-based assurance case models can be exchanged at runtime for higher-level engineering requirements.

The design of the SACM also takes into consideration scenarios where systems form a system of systems; in such cases, systems with SACM-based assurance case models can determine whether they are compatible (by using the *AssuranceCasePackageInterface*, as previously discussed). When the systems are compatible, a binding/contract (which contains the argumentation, if necessary, regarding why the systems are compatible and why they satisfy their safety/security requirements) can be created to bind/link assurance cases together to form a compound assurance case. For this purpose, the SACM provides the notion of binding, which is used to bind two or more interfaces at any given level (a binding also provides possible structured argumentations showing the logic underlying the integration).

3.2.2 SACM Machine-Readable Design

The SACM takes into consideration that machine-readable assurance cases can be created. The Base component (shown in Figure 21) of the SACM provides the necessary means such that not only names/descriptions can be described in natural language, they can also be described in computer languages (e.g., formal notations) to enable automated argument reasoning in future.

At the same time, the SACM provides various facilities (subclasses of *UtitlityElement* in Figure 20) allowing the user to define necessary constraints, notes, additional attributes etc.

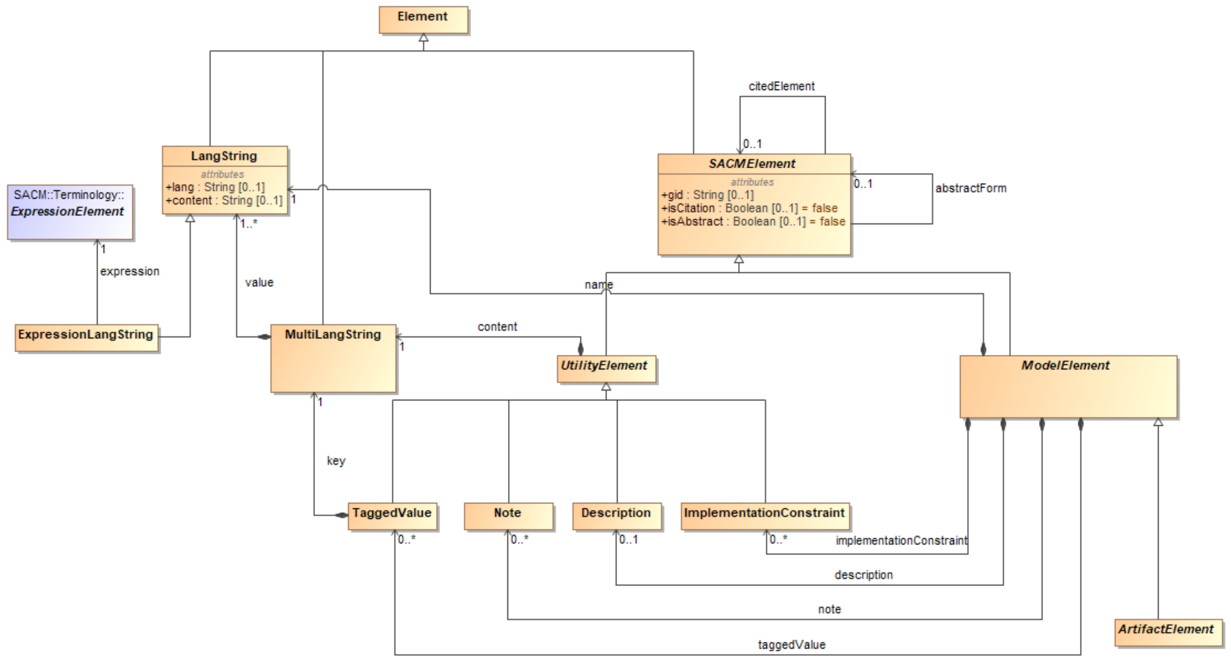


Figure 20 - SACM::Base Component

3.2.3 SACM Argumentation Component

As previously discussed, an assurance case created using the SACM contains a number of argument packages which contain structured argumentations. The SACM Argumentation component provides the facilities for creating structured argumentations, as shown in Figure 21. The user of the SACM can make a number of different types of claims which provide means of assertion, context, assumption and justification. The user can also make use of the *Artifact* component to refer to corresponding evidence (internal/external to the SACM model) to support the claims. There are various types of *AssertionRelationships* to link claims to sub-claims, contexts, assumptions, justifications etc.

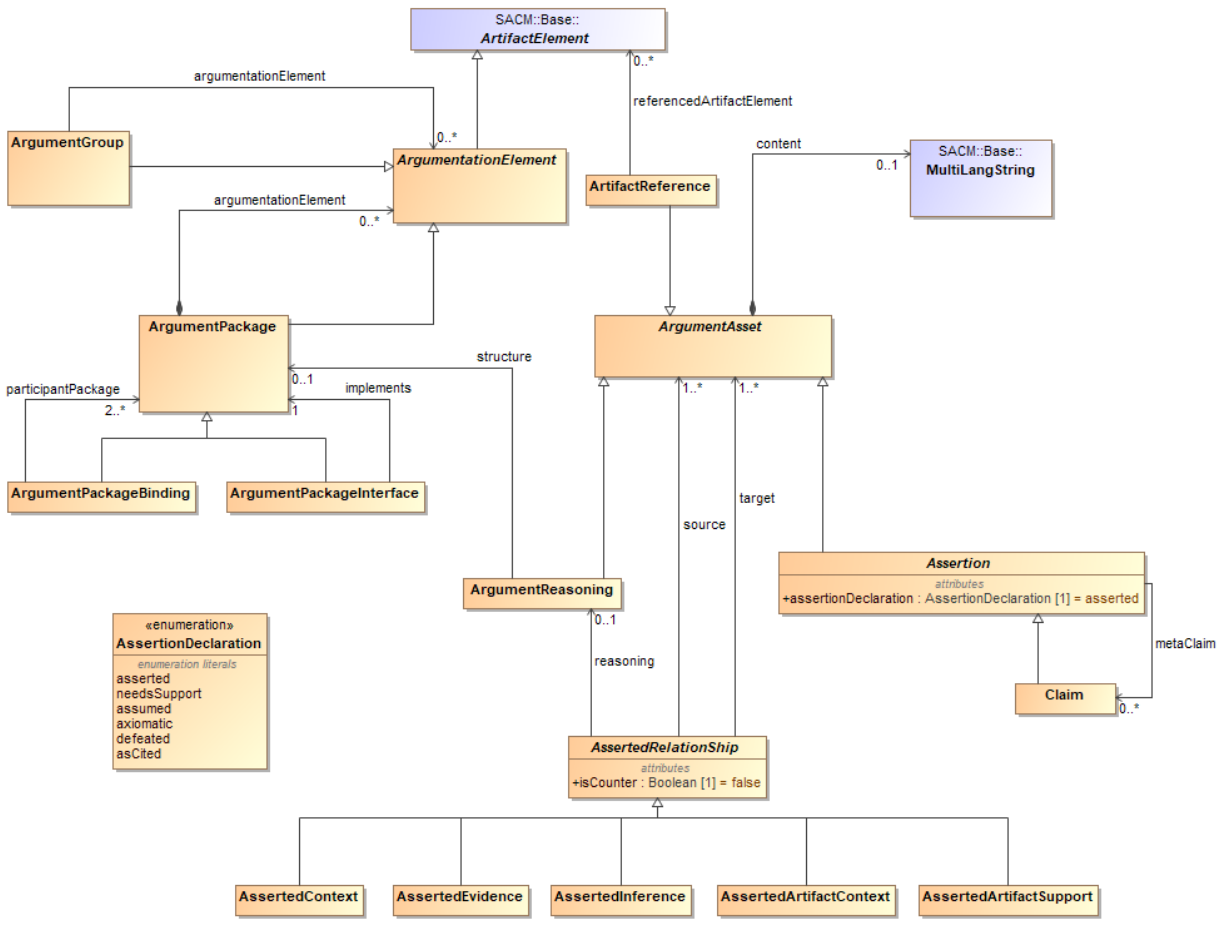


Figure 21 - SACM::Argumentation Component

As previously discussed, SACM provides the possibility for two systems to exchange information with regard to their structured argumentations about system assurance, via the *ArgumentPackageInterface*. In this sense, the creator of the structured argumentation can decide what information can be accessed externally (e.g., a safety requirement that is asserted to have been fulfilled), so that external users can make use of such information. Obviously, the notion of *interface* leads to the question of trust; the SACM also provides facilities for structurally arguing the level of trust embedded in the information provided in the interfaces (in the same manner as structured argumentation, via *metaClaims*).

With the interface present, systems can integrate and form a compound structured argumentation, by using *ArgumentPackageBinding*. The *ArgumentPackageBinding* used to integrate systems contains the underlying logic (in the form of structured argumentation) of the binding. This provides the possibility for systems to integrate at the level of argumentation.

3.2.4 SACM Artifact Component

The SACM provides the means for maintaining the evidence associated with the structured argumentations. The SACM Artifact component provides the facilities to maintain evidence such as *Resource*, *Artifact* as well as *Activities*, *Event*, *Participant* and *Technique*, as illustrated in Figure 22. SACM enables the user to point to external files/URLs of the related artifacts via the use of Property. In this sense, the SACM provides the necessary abstraction, as it does not demand the use of *models* for argumentation evidence. This abstraction provides a significant degree of openness regarding its adoption in open systems (i.e., Cyber-Physical Systems). However, the SACM does provide the necessary means for the Artifacts to be linked to model elements in the sense that constraints (described in model querying languages such as the Object Constraint Language) can be embedded into each one of the Artifacts, which, in turn, would be executed at runtime and retrieve the value of the referenced model elements.

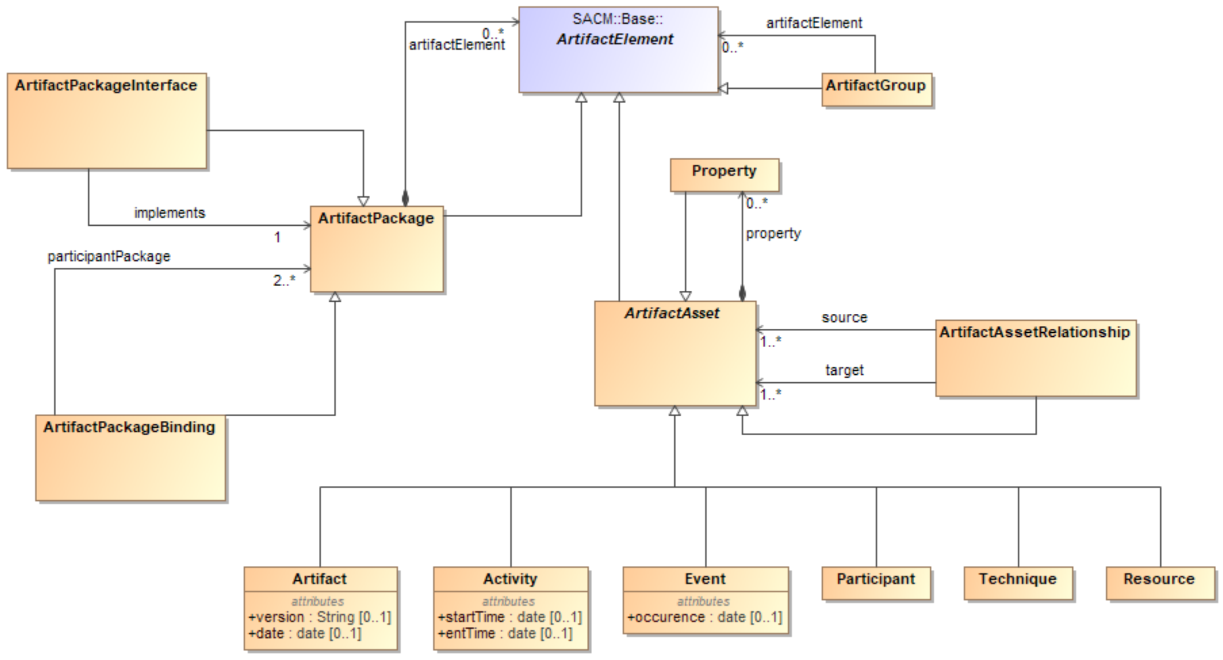


Figure 22 - SACM::Artifact Component

With respect to assurance case integration, there is also a need to exchange information at the level of evidence. The SACM provides the *ArtifactPackageInterface* to enable the exchange of Artifacts among assurance cases. The user can choose what evidence (inside an *ArtifactPackage*) can be accessed externally in an *ArtifactPackageInterface* associated with the *ArtifactPackage*. With the ArtifactPackageInterface, it is possible to bind ArtifactPackages by using ArtifactPackageBinding, so that system integration can be performed at the level of ArtifactPackage.

3.2.5 SACM Terminology Component

Without context, structured argumentation is meaningless. In SACM, the Terminology component provides the necessary means for defining controlled vocabularies which in turn link system information to the structured argumentation in the *ArgumentPackages,* as shown in Figure 23. Concerning system information, the user can define *Term*s, *Expression*s and *Categories*, which are the terminologies in the system for which the assurance case provides assurance. At this point SACM also provides the necessary abstraction so that external system information (such as system models, failure logic models, FMEA models, FTA models etc.) can be referenced. Note that the SACM does not demand the use of models to provide openness regarding its adoption in open systems (i.e., Cyber-Physical Systems).

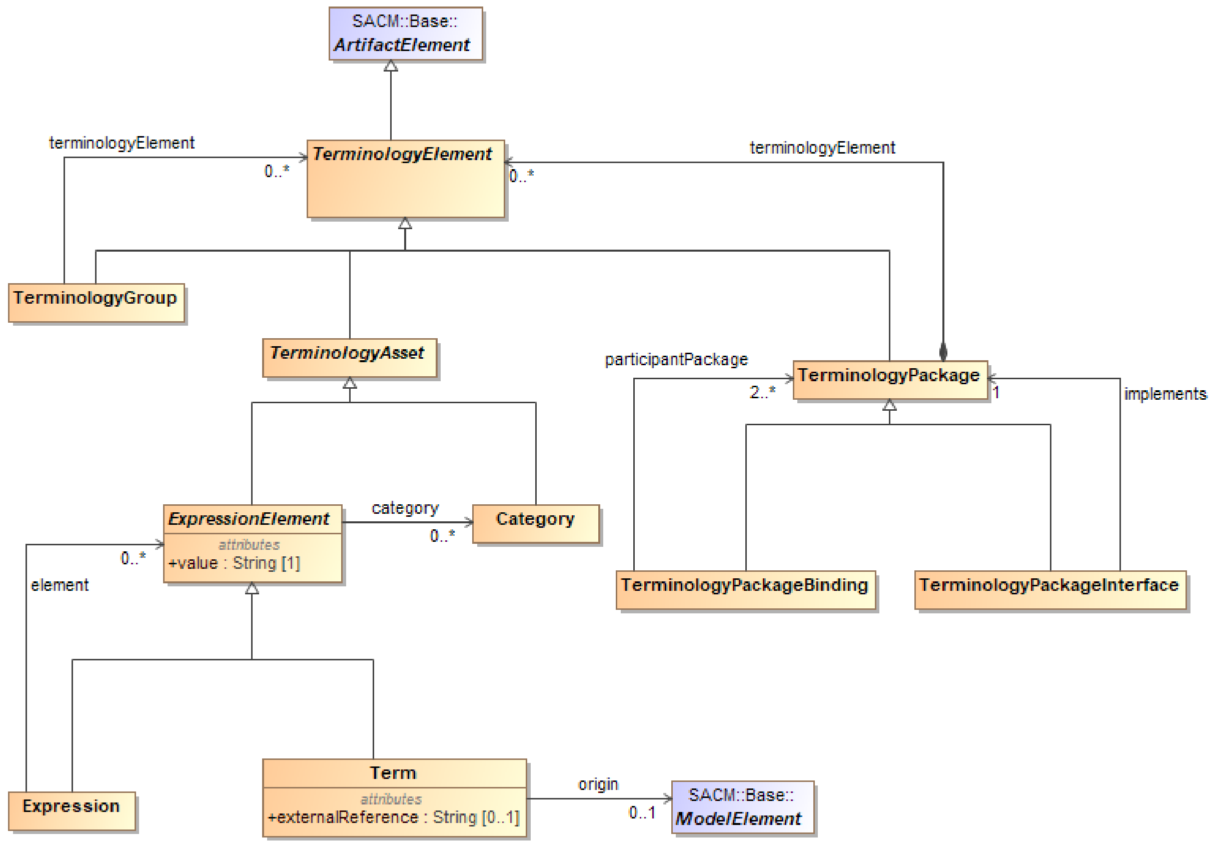


Figure 23 - SACM::Terminology Component

The creator of a *TerminologyPackage* can also decide to expose system information by using the *TerminologyPackageInterface* for system integration so that system information (e.g., system properties) can be accessed externally.

With *TerminologyPackageInterface* present, system integration is performed by using *TerminologyPackageBinding*, so systems are integrated at the Terminology level.

With standardised *TerminologyPackages*, a typical task to perform is to extend a standardised *TerminologyPackage* to create new standard/non-standard *TerminologyPackage*s. We can, once again, make use of the *+abstractform* of the *SACMElement* on both the level of the *TerminologyPackage* and the level of the elements contained inside *TerminologyPackage*s. In this sense, a *TerminologyPackage* can be extended. Standardised *TerminologyPackages* can be stored in publicly accessible repositories for reference, which is in line with the DEIS vision of the application of DDIs.

3.2.6 Summary

SACM is a complex metamodel defined by the specifiers of existing system assurance case approaches (i.e. GSN and CAE), based on the collective knowledge and experiences of safety and/or security practitioners over the period of 20 years. SACM contains more features than GSN and CAE, and is therefore more powerful in terms of expressiveness. The full specification of SACM can be found at https://www.omg.org/spec/SACM.

A paper explaining the usage of SACM via examples has also been submitted to the Journal of Systems and Software, entitled "Model Based System Assurance Using the Structured Assurance Case Metamodel".

3.3 Product meta-models

3.3.1 Architectural modelling package

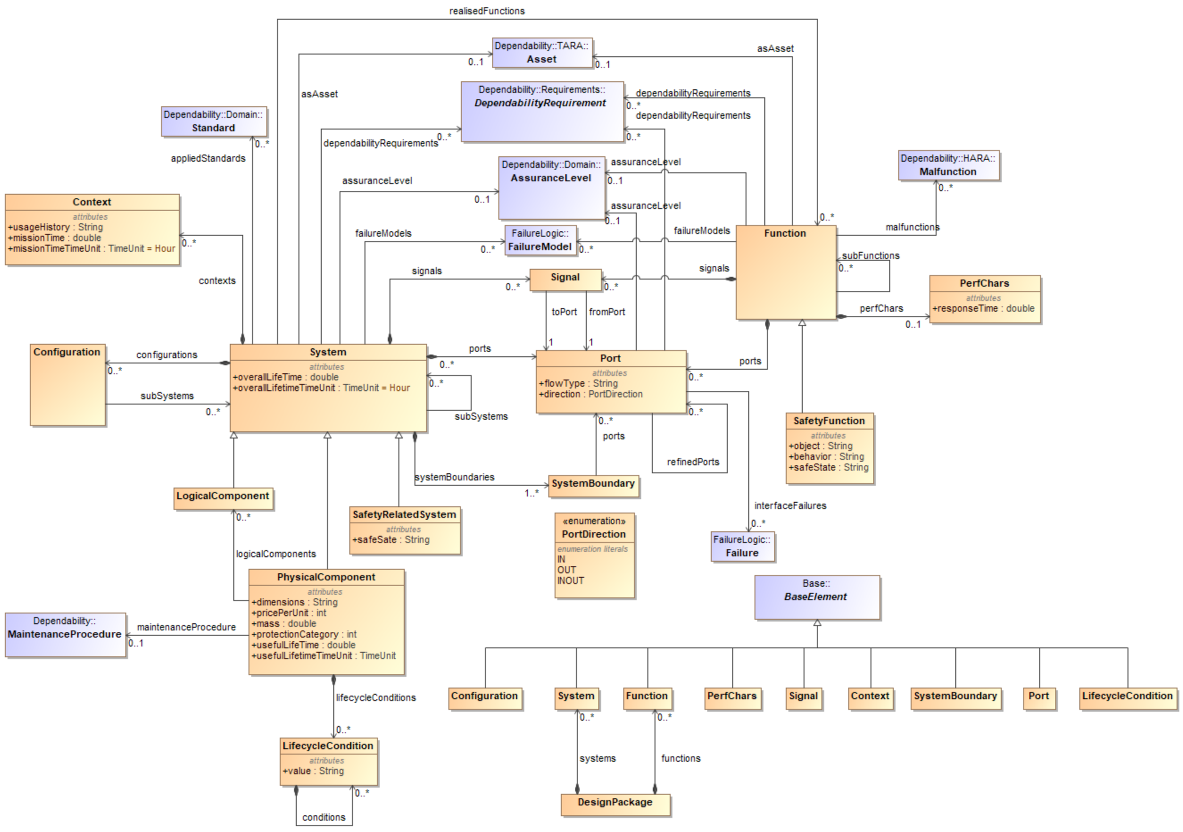


Figure 24 - ODE::Design Package

The ODE::Architecture package has undergone some simplification. Its updated state can be seen in Figure 24. To begin with, it has been renamed to 'Design' and its internal containment element 'DesignPackage'. The System and Function elements' role has been enhanced, as all other elements within the package compose onto or inherit from them. This change shifts the focus onto the structure of the system architecture as opposed to its properties.

Another significant change is the association of ODE::FailureLogic::FailureModels with Systems and Functions, whereas the more specific ODE::FailureLogic::Failures associate with individual Ports. This change clarifies and makes more precise the relationship between the functional/systemic failure analysis and individual failure behavior at the level of function/system/component interfaces.

3.3.2 Hazard and risk analysis (HARA) modelling package

The ODE::Dependability package remains largely the same from v1. Figure 25, Figure 26, Figure 27 and Figure 28, provide an overview of the current version of the package and its sub-packages. The main change is the association of a Hazard with zero or more ODE::FailureLogic::Failures, as opposed to the previous version's limit of one. This change simplifies the modeling of Hazards which can be caused by a combination of failures. In the previous version, in such situations, an intermediate ODE::FailureLogic::OutputFailure caused by a combination of the other failures was required to delegate failure propagation to the Hazard.

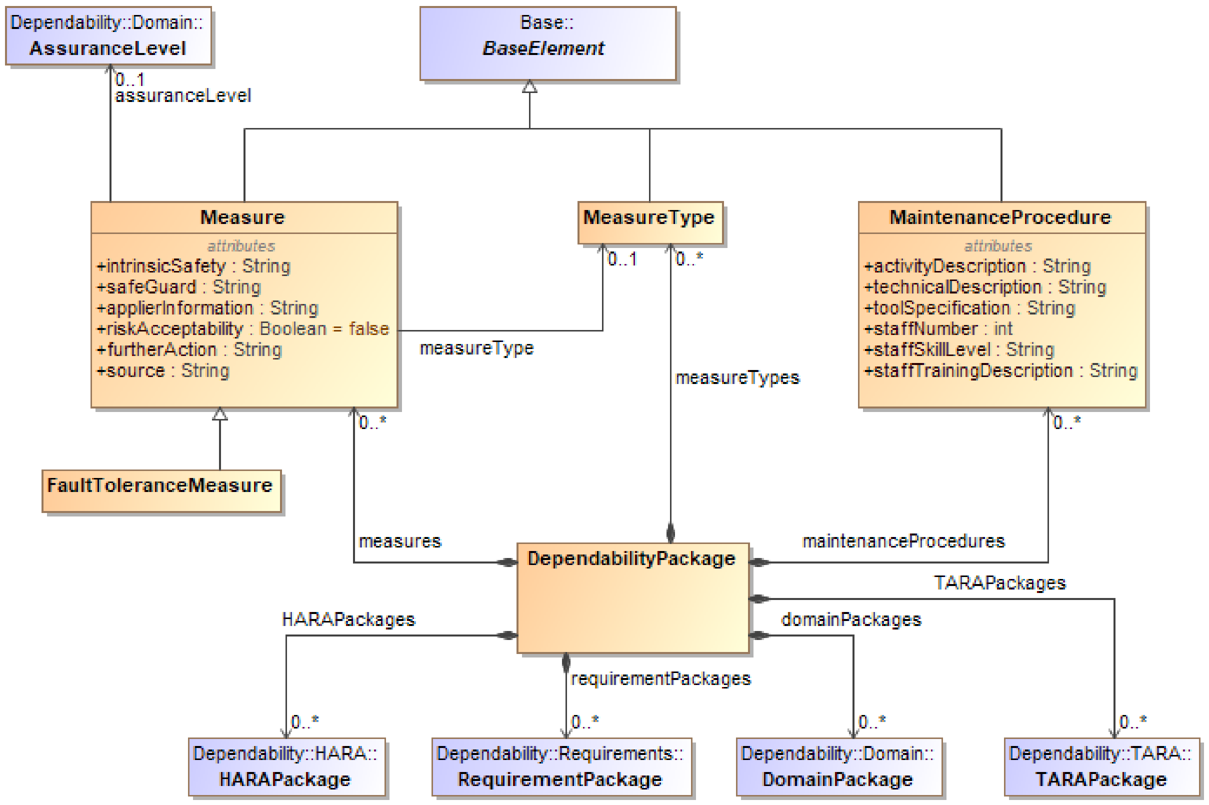


Figure 25 - ODE::Dependability Package

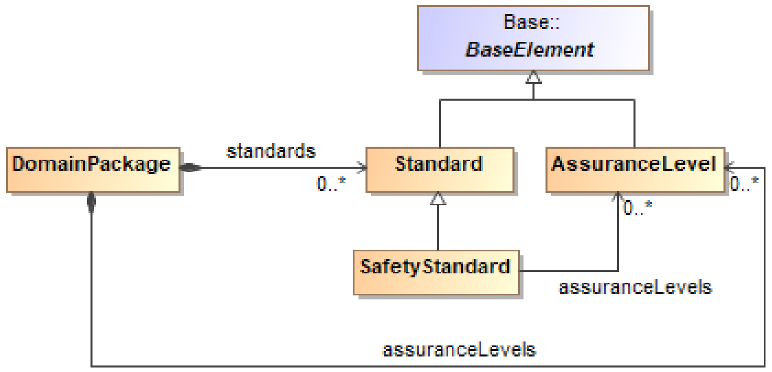


Figure 26 - ODE::Dependability::Domain Package

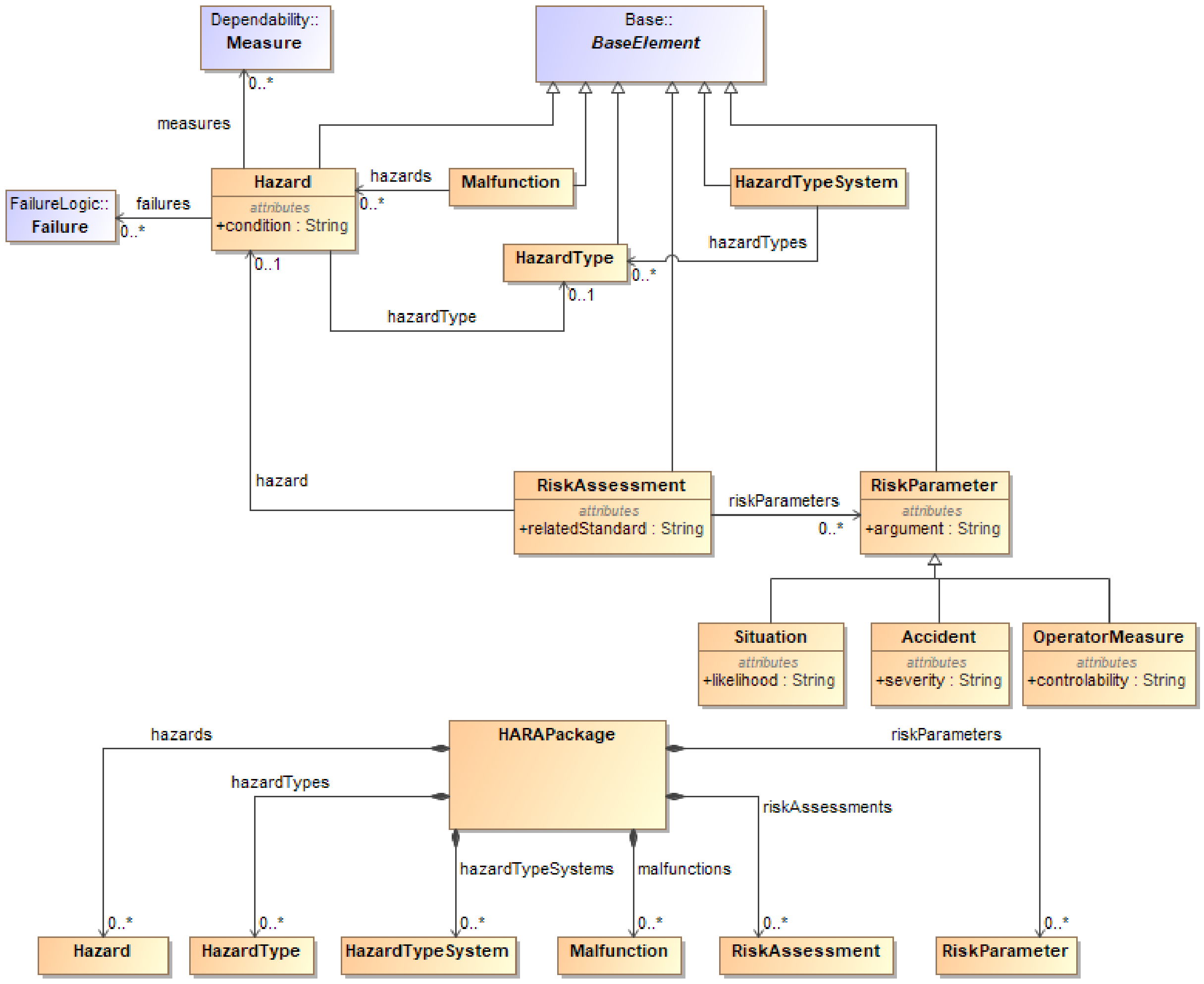


Figure 27 - ODE::Dependability::HARA Package

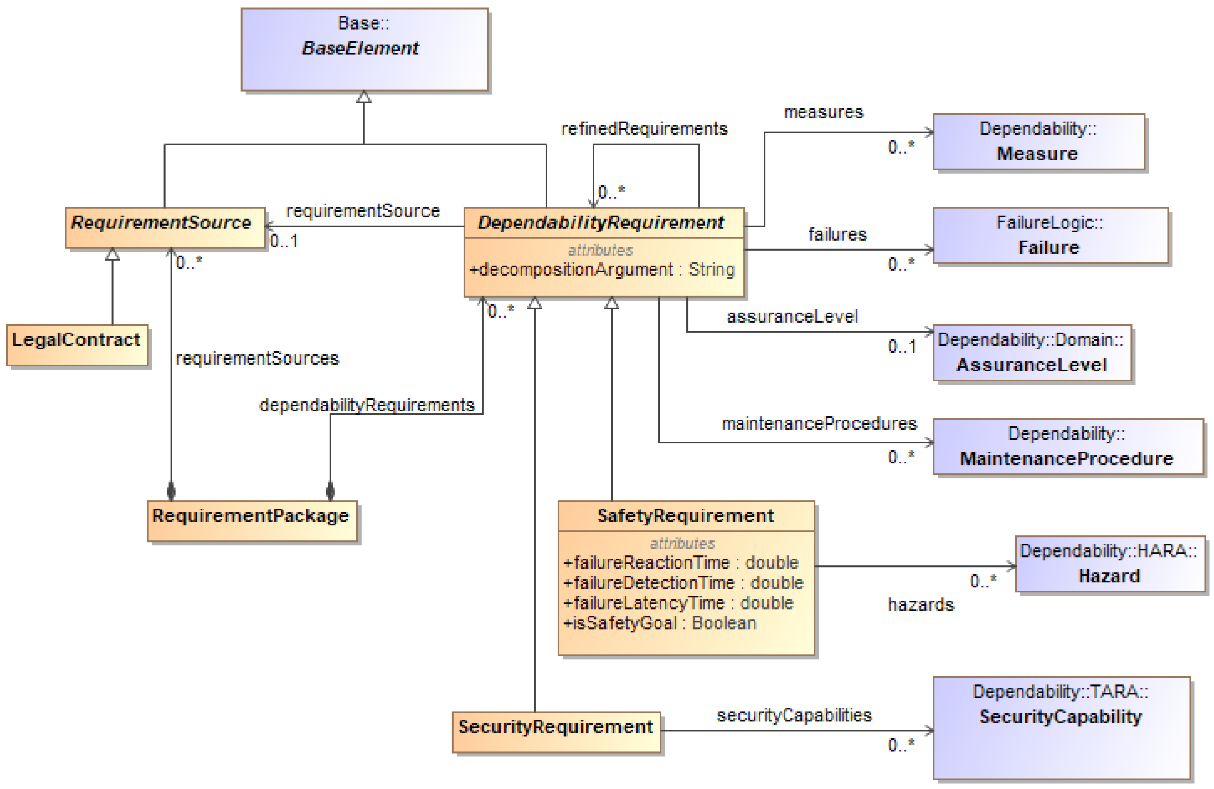


Figure 28 - ODE::Dependability::Requirements Package

3.3.3 Failure logic modelling package

The ODE::FailureLogic package has undergone significant restructuring and simplification. Figure 29, Figure 30, Figure 31 and Figure 32 present the current state of the package and its subpackages. Of particular note is the reduction of the separate Failure subtypes (InternalFailure, InterfaceFailure, InputFailure, OutputFailure) into a single type. The single Failure type maintains the semantics captured by the previous subtypes within its properties. For instance, the Failure::originType describes whether the failure is internal within a function/system/component or at the interface level.

Similarly, the FailureMode and CCF (Common Cause Failure) elements have been absorbed within the Failure type and represented with its class, isCCF and ccfFailures properties. The latter two denote whether the Failure represents a CCF and which are the other Failures that can be caused by the CCF.

The element MinimalCutSets now accurately describes the synonymous concept without the need to employ elements from the ODE::FTA package. In the previous version, complex combinations of minimal cut sets needed the ODE::FailureLogic::FTA::Gate element to be represented.

The SecurityViolation element is a new addition to the FailureLogic package. By inheriting from Failure, it enables modeling the direct effect a security ODE::TARA::Attack has on the system (see TARA package below).

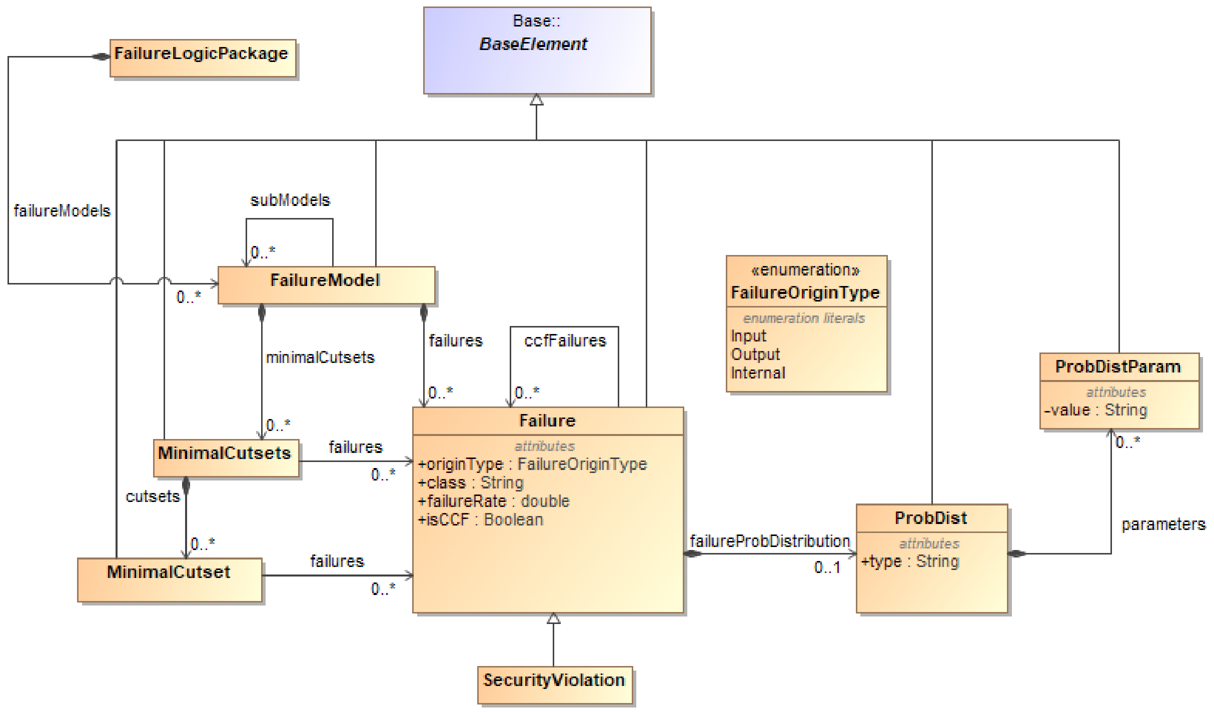


Figure 29 - ODE::FailureLogic Package

The ODE::FMEA package has undergone some simplification too, see Figure 30. Propagation is now represented with the association of each FMEAEntry with a ‘mode’ and ‘effect’ Failure. This change contrasts the previous version’s propagation modeling with explicit elements such as FMEAPropagation and DiagnosableFailurePropagation, which were deemed redundant. The specialization of the Failure type for FMEAs was also removed due to redundancy.

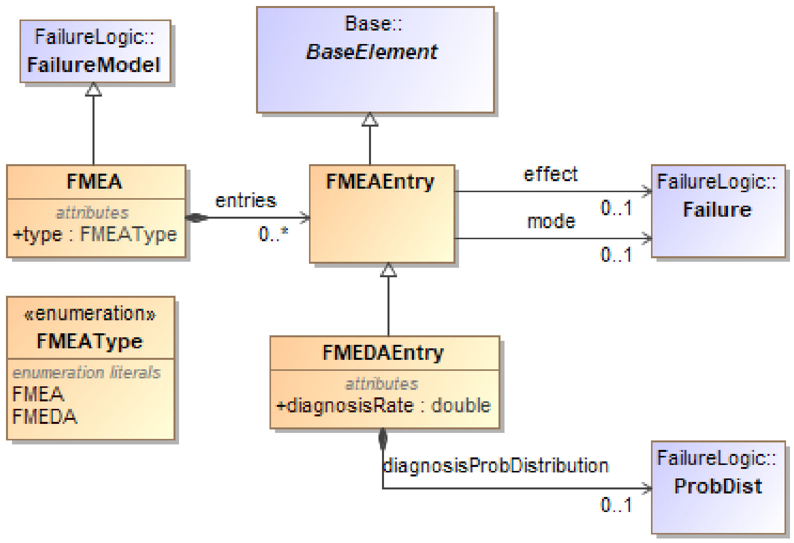


Figure 30 - ODE: FailureLogic::FMEA Package

The ODE::FTA package in Figure 31 has been simplified by reducing the various types of events represented in a fault tree down to a generic Cause element. Boolean logic event connectors are represented by the Gate element, which, as a subtype of Cause, can be used to chain hierarchies of Causes together into a fault tree.

Note that since generic Causes can reference both Failure and Security Violations as their inherited type, it is now possible to represent fault trees from classical dependability analysis as well as security attack trees. Furthermore, if a FaultTree is composed of Causes referencing both Failures and Security Violations, hybrid safety-security analysis can also be modeled.

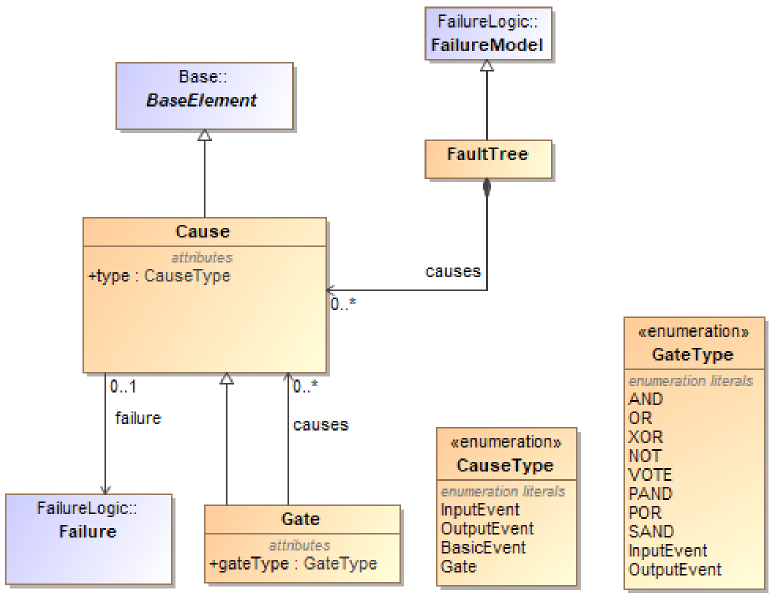


Figure 31 - ODE::FailureLogic::FTA Package

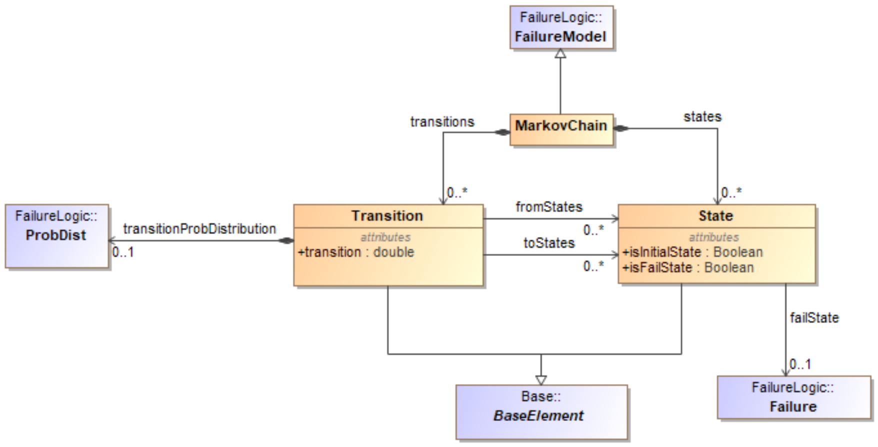


Figure 32 - ODE::FailureLogic::Markov Package

3.3.4 Threat and risk analysis (TARA) modelling package

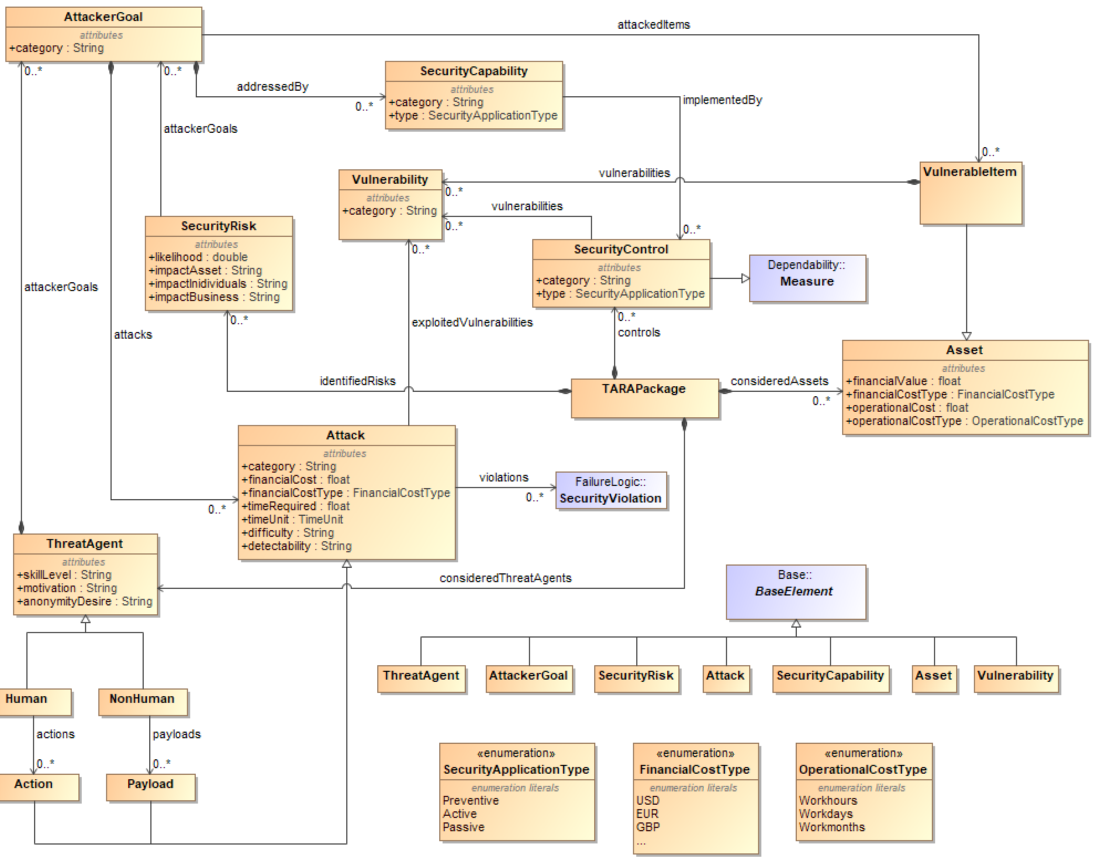


Figure 33 - ODE::TARA Package

The TARA package models the results of a Threat And Risk Analysis for security, seen in Figure 33. ThreatAgents are either Human or NonHuman (typically electronic) sources of Attacks. While individual Attacks may serve many purposes, ThreatAgents will also feature some higher-level AttackerGoal, representing the overall goal of the attacker. The AttackerGoal revolves around negatively impacting the Assets being considered for security, often being the system's operation and its data for example. Attacks exploit Vulnerabilities of the system.

Cumulatively, the above elements contribute towards the SecurityRisk posed by the various threats identified during the TARA. To combat these threats and reduce risk, SecurityCapabilities and SecurityControls are established. Respectively, these are high-level and low-level security safeguards/counter-measures. After applying these measures, risk is reduced accordingly.

Finally, more detailed analysis of the propagation of effects of Attacks on the system are modeled by linking individual Attacks with ODE::FailureLogic::SecurityViolations. This link further enables hybrid security-safety analysis, as complex ODE::FTA::Causes can be associated with Failures from classical dependability analysis.

3.4 Certification meta-models

For project cluster purposes, DEIS reaches out to the AMASS project for the metamodels developed for process and certification modelling.

In the AMASS project, a Common Assurance & Certification Metamodel (CACM) has been developed. CACM is a cluster of metamodels that captures concepts in various aspects of system assurance, such as system specification, argumentation, evidence, process, standards and the mapping between process and standards. The overlap between AMASS and DEIS is quite observable, therefore, it is best practice to re-use part of the CACM from AMASS to model process and certification.

CACM has a subset of metamodels called Compliance Management Metamodel, which contains the following aspects:

- Assurance project definition. It is used to define the assets produced during the development, assessment and justification of a safety-critical system;
- Process Definition. It is used to model general reference processes (e.g., Waterfall Process, Agile Process, V-model process), or company-specific processes (e.g., the Thales process to develop safety-critical systems).
- Standard Definition. It is used to model standards (IEC 61508, ISO 2626, DO-178C, EN 50126, etc.) and any regulations (either as additional Requirements or model elements in a given model representing a standard or a new reference standard). For the implementation another metamodel is added, the Baseline Metamodel, to capture what is planned to be done or to be compiled with a defined standard, in a concrete assurance project.
- Vocabulary Definition. It is used to provide a Thesaurus-type vocabulary, which defines and records key concepts relevant to safety assurance within the target domains and the relationships between them.
- Mapping Definition. It is used to capture the nature of the vertical and horizontal mappings between the different levels of model in the AMASS Framework and between the concepts and vocabulary used in these models. There are two types of mapping: equivalence mapping that maps process models with models of standards; and process mapping, which maps process models to project specific models.

The compliance management metamodels are helpful for DEIS for their ability to model processes and standards (and their relationships using the mapping definition). However, vocabulary definition and assurance project definition metamodels are overlapped with SACM. The plan next is to isolate the useful metamodels (hopefully they are independently implemented) from CACM and use them in conjunction with ODE and SACM.

3.5 Crosscutting Aspects

3.5.1 Information abstraction and IP Hiding concept

There are two main aspects to IP hiding currently featured in the design of the ODE. The first is imported from SACM and the second relates to the design of the ODE::FailureLogic package.

SACM features a collection of -PackageInterface elements. These elements enable IP hiding by allowing the user to only publish parts of the assurance case (including contained artifact models). Most elements in SACM inherit SACMElement's 'isCitation' attribute, which is a flag denoting whether the subject element cites another one. If the flag is 'true' then SACMElement's 'citeElement' attribute provides a reference to the cited element. SACM -PackageInterfaces contain citation elements to elements from other packages of the same type (except in the case of AssuranceCasePackageInterfaces, they contain other -PackageInterfaces). For example, a TerminologyPackageInterface contains citations to elements from a TerminologyPackage. By sharing the TerminologyPackageInterface rather than the actual package, only the elements cited are visible to the receiver, assuming there is no other means of accessing the cited package.

The ODE::FailureLogic package contains sub-packages which expose in more detail how a system's dependability can be compromised from failures and security violations. These are the FMEA, FTA and Markov sub-packages. These sub-packages include information that can directly or indirectly describe the failure behavior of the system and/or reveal key elements of its design. To avoid revealing the inner workings of the system while still sharing as much information to external parties for evaluation, the detailed analyses found in the sub-packages of ODE::FailureLogic, such as ODE::FailureLogic::FTA, are separated. Instead, the provider can opt to share only the system's Minimal Cut Sets, which provide describe the minimal combinations of necessary and sufficient causes of system failure.

3.5.2 Harmonization of the ODE with common meta-modeling languages

The design of the ODE, while aiming to be reasonably concise and flexible, is focused on addressing the requirements drawn from the DEIS project's use cases. However, neither the DDI nor the ODE concepts should be viewed as monolithic constructs but instead, as solutions adaptable to the problem at hand. In this section, we aim to highlight the ODE's flexibility by briefly mentioning the potential for interoperation with preexisting, established modeling languages. In doing so, the options for adapting the DDI concept to current practice should become clearer.

We will be discussing interoperability with the SafeML, SysML, EAST-ADL2 and AADL modeling languages. The Safety Modeling Language (SafeML) enhances the management of safety information produced during the development lifecycle of safety-critical systems (Biggs, Sakamoto, & Kotoku, 2014). The Systems Modeling Language (SysML) is a popular choice for defining, analyzing and verifying generic systems and features numerous extensions for domain-specific applications such as SafeML. The EAST Architecture Description Language (EAST-ADL2) is the 2nd version of the modeling language targeting embedded systems for the automotive domain. EAST-ADL2 models feature multiple levels of abstraction, with each level addressing different development stages and views. The Architecture Analysis and Design Language (AADL) focuses on performance-critical systems development and initially targeted the aerospace domain.

3.5.2.1 SafeML

SafeML targets several issues that typically appear during design process. For example, information needs to be communicated among different stakeholders with different backgrounds. This typically leads to duplication and often to ambiguous and inconsistent artifacts. Moreover, this information tends to be documented in different formats, which makes maintenance difficult and costly whenever changes are required.

SafeML targets the mentioned issues with a model-based solution, in which the safety information is added to existing design models. The authors of the initiative offer on their website (Geoffrey Biggs/AIST, 2017) profiles for the commercial modeling tools Enterprise Architect and Magic Draw, as well as links to other resources like documentation and the Object Management Group (OMG) language specification.

SafeML is designed to be used in conjunction with SysML (Friedenthal, Moore, & Steiner, 2008). SysML provides the diagrams and element types necessary for design modelling, while SafeML provides the element types used to add safety information to the model, see Figure 34.

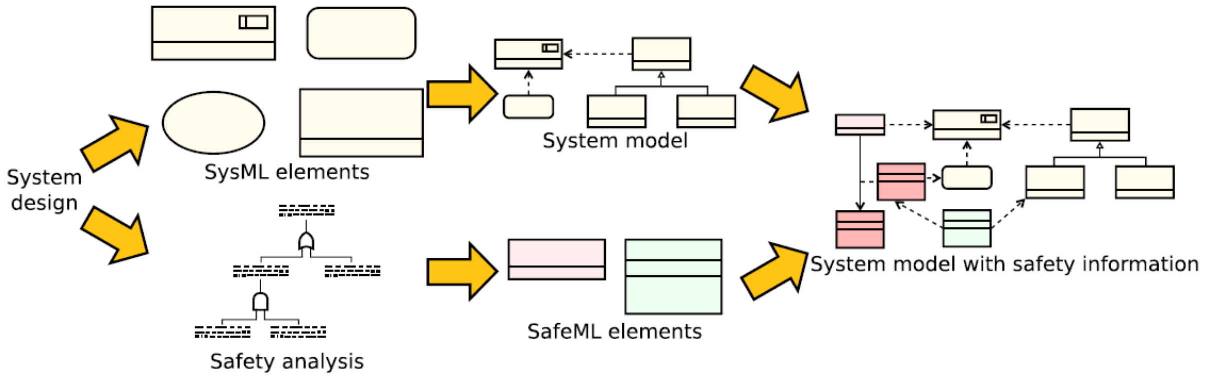


Figure 34 - SafeML usage concept

The modeling approach is organized in two parts. The first deals with hazards, harms and the context necessary for harms to occur. The second deals with safety measures, which in SafeML are targeted at preventing the hazardous event necessary for a hazard to lead to harm, or mitigating harm should a hazardous event occur. According to the specification, this is the list of modeling elements:

Table 1 - SafeML to ODE Mapping

SafeML Modeling element	ODE element	Comparison
Hazard	Hazard	Concept is understood exactly in the same way
Harm	Accident	Similar concepts used to describe physical injuries and damage to people. Main difference is in the attributes and usage. An Accident in ODE is used in the context of the Hazard and Risk analysis and defines only the severity in case of occurrence. Harms in SafeML are more global aspects. They include a quantification of the overall risk of the harm, including the use of measures.
Harm Context	Risk assessment	Both are used to depict how harm might be cause by a hazard. Therefore, both include typical parameters for risk assessment like occurrence, severity and controllability.

ActiveDefence, PassiveDefence	Measure	Very similar concept with the difference that in SafeML it is distinguished whether a measure (i.e. defence) need to be activated explicitly or not to protect against a HarmContext. In ODE this will be reflected partially in the design when the measure has been made part of it.
Context Detector	NONE	It represents the capability to monitor for a hazardous situation/event. There is no exact matching for this concept, but it could be represented in different ways through the ODE depending of the interpretation. E.g. by a measure, or a function.

One of the most remarkable aspects of the SafeML approach is that it is a concrete modeling language. It defines modeling elements and relationships and, more over it relies on SysML, which is by itself also a modeling language. Contrary to SafeML and SysML, ODE does not define how the information is created, nor concretely define processes or languages. It primarily defines containers and structure to allocate information. The ODE has been conceived in this way having in mind data exchange. It does not place constraints or guidelines on how the information is obtained.

The purpose of SafeML is to extend SysML models with safety information, by focusing on the definition of hazards and measures. This covers however just a portion of the actual information needs for dependability systems. One could even consider the language as incomplete, since there are no means to perform a complete hazard and risk analysis. For instance, there are no means for identifying malfunctions. Therefore, this process should basically occur somewhere else and only the resulting hazards will be documented. Contrary, ODE has been structured to cover the entire lifecycle and it is therefore far more complete and structured. Consider for instance the following aspects: Fault analysis, security concerns, argumentation, etc.

In summary, SafeML does not offer much more to what is already existing in SysML. Nevertheless, models created with SafeML will be, as shown before, translatable into ODE models.

3.5.2.2 SysML, EAST-ADL2, AADL and HiP-HOPS

For SysML, EAST-ADL2 and AADL, transformations to HiP-HOPS-compatible models have been described in past research. These transformations enable us to invoke the HiP-HOPS automated dependability analysis and output its results in its standard format. Outputs from HiP-HOPS can be converted to ODE-compliant models i.e. DDIs. Therefore, by transitivity, we can extend the transformation process to include models from SysML, EAST-ADL2 and AADL, transform and process them via HiP-HOPS and finally transform the outputs into DDIs.

We will briefly summarize the semantic correspondence between key elements of each language and HiP-HOPS. The mappings from the EAST-ADL2, SysML and AADL metamodels to HiP-HOPS are summarized in Table 2, Table 3 and Table 4 respectively. We should note that current tool support only offers transformation from external languages to HiP-HOPS and not vice versa. More information on the mappings can be found in (Biehl, Chen, & Torngren, 2010) for EAST-ADL2, in (Mian, Bottaci, Papadopoulos, & Biehl, 2012) for AADL and (Andrews, Payne, Romanovsky, Dider, & Mota, 2014) for SysML.

Table 2 - EAST-ADL2 to HiP-HOPS Mapping

EAST-ADL2	HH
ErrorModelType	System
ErrorModelType.errorConnector	System.Lines
ErrorModelType.parts	System.Component
ErrorModelPrototype.type.errorPort	System.Component.Ports
ErrorModelPrototype	System.Component.Implementation
ErrorModelPrototype.type. errorBehaviorDescription.internalErrorEvent	System.Component.Implementation. FailureData.basicEvent
ErrorModelPrototype.type. errorBehaviorDescription.failureLogic	System.Component.Implementation. FailureData.outputDeviation
ErrorModelPrototype.type	System.Component. Implementation.System (recursion)

Table 3 - SysML (for COMPASS Artisan Studio) to HiP-HOPS Mapping

SysML (via COMPASS Artisan Studio)	HH
Fault Analysis Model	Model
SoS	System
Optimisation Parameters	Optimisation Parameters
Objective	Objective
Constituent	System
Implementation	Implementation
Component	Component
Port	Port
Line	Line
Connector	Line
LineEnd	Line.Port
Propagation Logic	Line.PortExpression
FailureClass	OutputDeviation.Name (partially)
BasicEvent	BasicEvent

Table 4 - AADL to HiP-HOPS Mapping

AADL	HH
SystemInstance	System
ComponentInstance	System.Component.Implementation
ComponentErrorModelProperty	System.Component.Implementation.FailureData
ComponentInstance.FeatureInstance	System.Component.Ports
ConnectionInstance	System.Lines

In Table 5, the symmetric mapping between HiP-HOPS and the ODE is summarized. Unlike the other mappings, the semantics in the latter are defined to enable bi-directional conversion. This means that conversion from both HiP-HOPS to ODE models and vice versa is possible.

Table 5 - HiP-HOPS to ODE Mapping

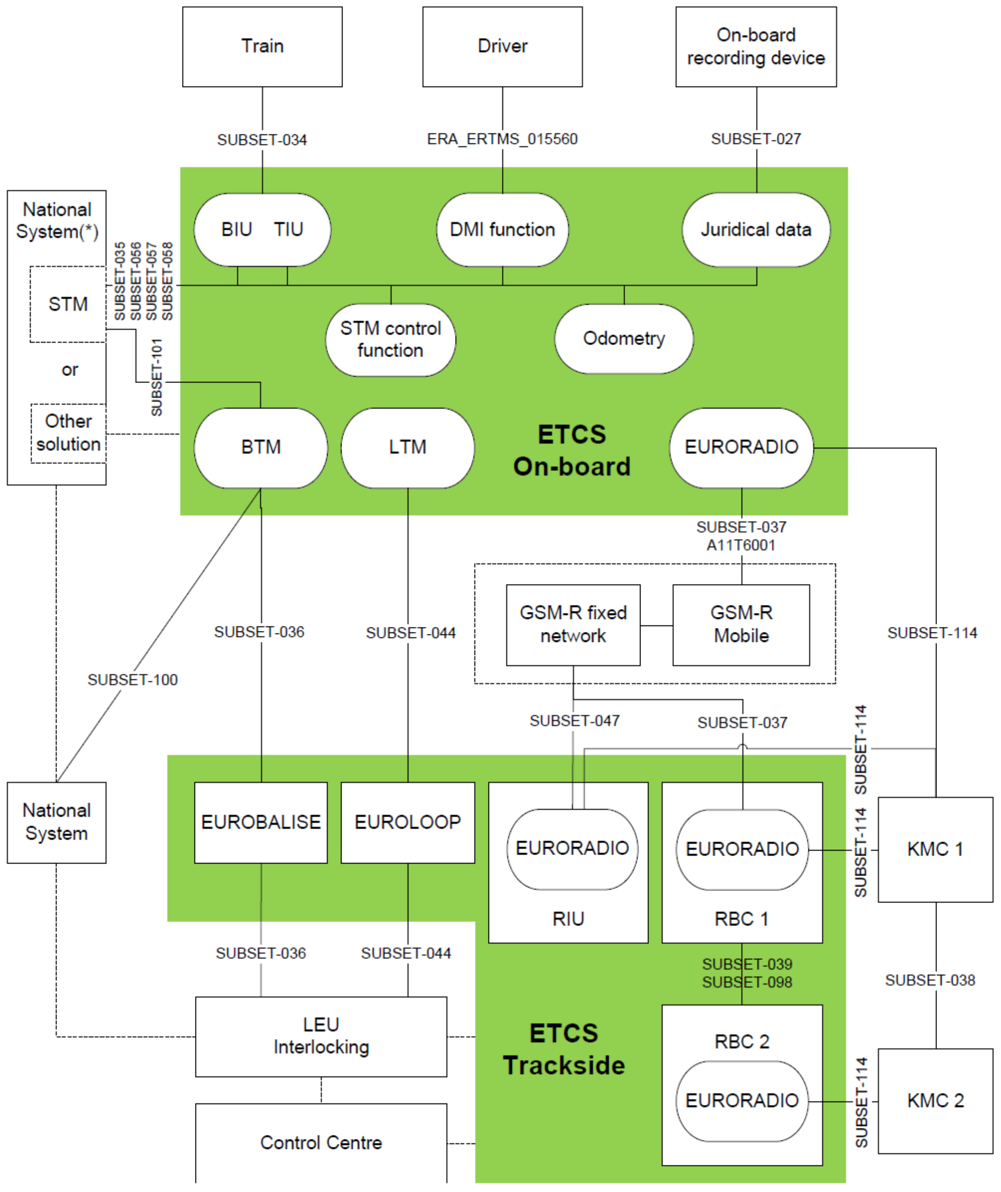
HH	ODE
Model	Integration::ODEPackage
Perspective	Architecture::ArchitecturePackage
System	Architecture::ArchitecturePackage/System
Component	Architecture::System/Logical/PhysicalComponent
Implementation	Architecture::System/Logical/PhysicalComponent
FailureData	FailureLogic::FailureLogicPackage/FTAPackage
BasicEvent	FTA::BasicEvent
PotentialCCF	FailureLogic::CCF
OutputDeviation	FTA::OutputEvent/FailureLogic::OutputFailure
Port	Architecture::Port
Line	Architecture::Signal
FMEA	FMEA::FMEAPackage
FMEA-Component-BasicEvent	FMEA::FMEAFailure
FMEA-Component-BasicEvent-Effect	FMEA::FMEAFailure
FMEA-Component-BasicEvent & Effect	FMEA::FMEAPropagation
FaultTree	FTA::FTAPackage
TopNode	FTA::OutputEvent
Gate (And, Or, …)	FTA::Gate
AllCutSets	FailureLogic::MinimalCutSets

4 Utilization of DDIs in concrete engineering activities of DEIS use cases

4.1 Utilization of DDIs in an OEM-TIER integration scenario in the context of the ETCS

4.1.1 Use case overview

The European Train Control System (ETCS) provides standardised train control in Europe and makes it easier to travel by train across the borders of all countries in Europe.



(*) Depending on its functionality and the desired configuration, the national system can be addressed either via an STM using the standard interface or via another national solution

Figure 35 - ERTMS/ETCS Reference Architecture

ETCS itself is a system of systems, consisting of an on-board and a trackside sub-system (see Figure 35). In the railway domain, the realisation of such a system typically involves different stakeholders in the value chain (railway undertaking, OEMs, suppliers etc.). Often a railway company orders an ETCS system from one vendor while another one provides the trackside equipment. Moreover, the on-board sub-system must interact with the train, which is either built by another department in the same company or by another OEM.

In order to satisfy the laws and regulations in each European country, it must be proven that the overall ETCS system is sufficiently safe. Therefore, both sub-systems must fulfil the safety requirements as defined in Subset-091 (Safety Requirements for the Technical Interoperability of ETCS in Levels 1 & 2) of the ERTMS/ETCS specification (ETCS/ERTMS, 2016). This standard defines specific hazards, and tolerable hazard rates are apportioned to each sub-system. Moreover, interoperability between the trackside and the on-board systems must be ensured.

4.1.2 Semi-automated ETCS assurance case instantiation

The suppliers of the on-board or the trackside part of the ETCS system provide assurance case information about the trackside or on-board ETCS sub-system in the form of DDIs.

During the system integration phase the information provided in the DDIs of the sub-systems are integrated to generate an overall assurance case. Hence, an on-board or trackside system can be integrated into an existing railway system and interact in a safe manner with the pre-existing systems within the railway system.

In order to enable the (semi-)automated integration of safety case fragments, the modular safety arguments (also called fragments here) of the sub-systems have to be combined and additional argumentation may be added, taking into account the integration context of the overall system, which was only assumed during the development of the sub-systems. Therefore, the content and the structure of the safety argumentation must be formalized, so that algorithms can help to perform the integration of the fragments in a (semi-)automated way. By enabling semi-automated assurance case instantiation, not only the content and structure of the safety argumentation is formalized but also the fulfilment of safety goals can be semi-automated.

For instance, to show that the safety goal defined in the UNISIG Subset-091 (ETCS/ERTMS, 2016) (Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.) is fulfilled for the ETCS trackside equipment, a number of question must be answered (see Figure 36).

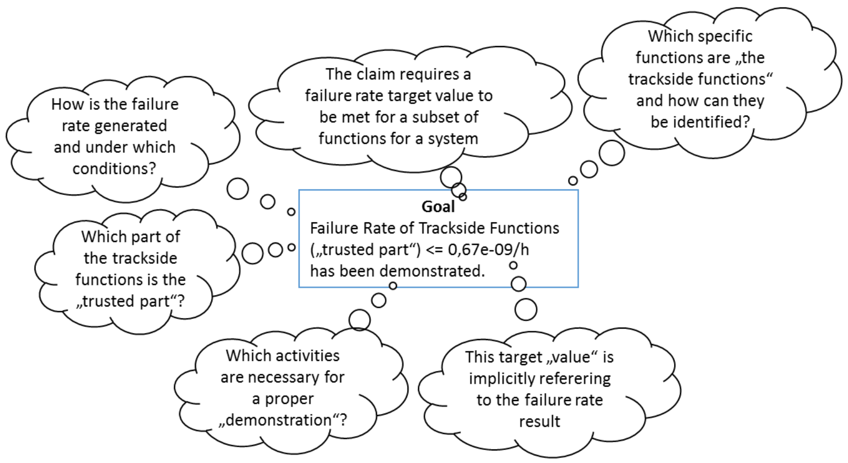

How is the failure rate generated and under which conditions?
The claim requires a failure rate target value to be met for a subset of functions for a system
Which specific functions are „the trackside functions“ and how can they be identified?
Which part of the trackside functions is the „trusted part“?
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
Which activities are necessary for a proper „demonstration“?
This target „value“ is implicitly referering to the failure rate result

Figure 36 - Safety goal of the ETCS trackside part as defined in UNISIG Subset-091

In current industrial practice all the questions shown in Figure 36 are answered by safety experts in a manual process of creating the safety case. Moreover, other design artefacts (such as process descriptions, safety analyses, hazard lists, specifications) are referenced to provide evidence that the goal is fulfilled. The resulting safety argumentation is written in the safety case document using natural language or GSN diagram (e.g. as depicted in Figure 37).

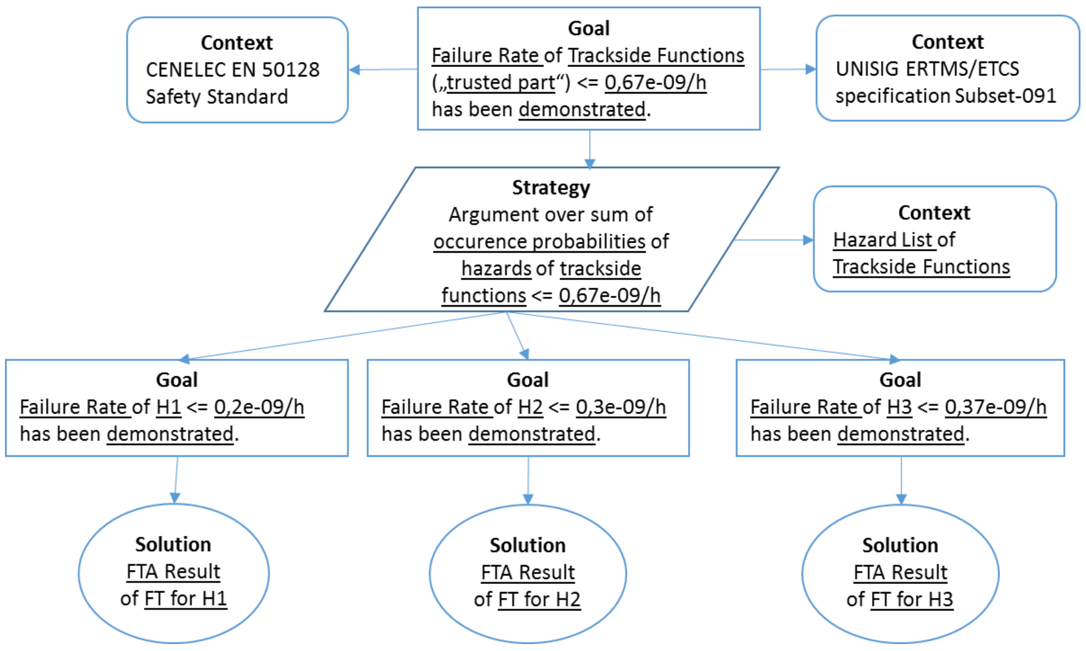

Context
CENELEC EN 50128 Safety Standard
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
Context
UNISIG ERTMS/ETCS specification Subset-091
Strategy
Argument over sum of occurence probabilities of hazards of trackside functions <= 0,67e-09/h
Context
Hazard List of Trackside Functions
Goal
Failure Rate of H1 <= 0,2e-09/h has been demonstrated.
Goal
Failure Rate of H2 <= 0,3e-09/h has been demonstrated.
Goal
Failure Rate of H3 <= 0,37e-09/h has been demonstrated.
Solution
FTA Result of FT for H1
Solution
FTA Result of FT for H2
Solution
FTA Result of FT for H3

Figure 37 - Example GSN diagram of the safety argumentation fragment of the ETCS trackside sub-system

The ODE extends SACM by adding product- and process-related model semantics to enable (semi-) automated assurance case integration/evaluation support. Thereby, SACM provides the foundation of formalizing context and structure of the safety argumentation as well as mechanisms to:

1. link an artifact element to an external resource (e.g. model);

2. link different artifacts together (ArtifactAssetRelationship) to express their relation;
3. specify structured language as terms/expressions with the possibility to link the terms to arbitrary external resources (e.g. elements from ArtifactPackage or completely external).

Moreover, the process model describes the activities necessary (e.g. described in the safety standard EN50129 (CENELEC, 2003)) to show the safety of a specific system in a formal and machine-readable way. The product model describes the artifacts of the system as well as their relationship. These elements are referenced as artifacts in the SACM argumentation. Since the process model is independent of concrete product meta-model, a mapping model is needed which maps standard concepts and process steps to a concrete product meta-model (see Figure 38).

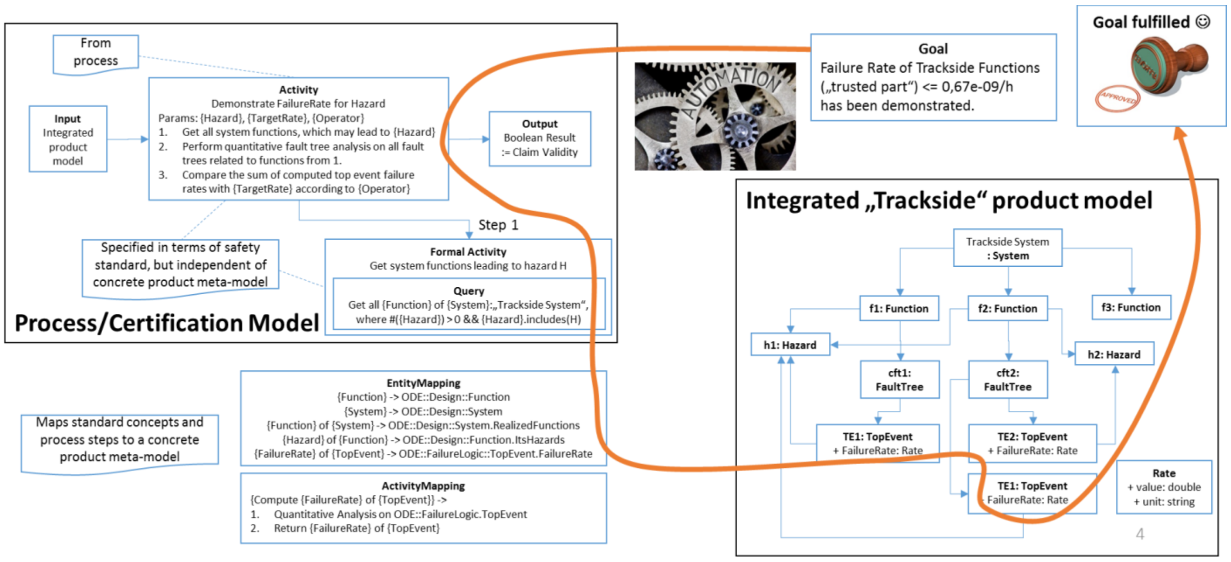

From process
Activity
Demonstrate FailureRate for Hazard
Params: {Hazard}, {TargetRate}, {Operator}
1. Get all system functions, which may lead to {Hazard}
2. Perform quantitative fault tree analysis on all fault trees related to functions from 1.
3. Compare the sum of computed top event failure rates with {TargetRate} according to {Operator}
Input
Integrated product model
Output
Boolean Result
:= Claim Validity
Step 1
Specified in terms of safety standard, but independent of concrete product meta-model
Formal Activity
Get system functions leading to hazard H
Query
Get all {Function} of {System}:„Trackside System“, where #({Hazard}) > 0 && {Hazard}.includes(H)
Process/Certification Model
Maps standard concepts and process steps to a concrete product meta-model
EntityMapping
{Function} -> ODE::Design::Function
{System} -> ODE::Design::System
{Function} of {System} -> ODE::Design::System.RealizedFunctions
{Hazard} of {Function} -> ODE::Design::Function.ItsHazards
{FailureRate} of {TopEvent} -> ODE::FailureLogic::TopEvent.FailureRate
ActivityMapping
{Compute {FailureRate} of {TopEvent}} ->
1. Quantitative Analysis on ODE::FailureLogic.TopEvent
2. Return {FailureRate} of {TopEvent}
AUTOMATION
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
Goal fulfilled ☺
APPROVED
Integrated „Trackside“ product model
Trackside System : System
f1: Function
f2: Function
f3: Function
h1: Hazard
h2: Hazard
cft1: FaultTree
cft2: FaultTree
TE1: TopEvent + FailureRate: Rate
TE2: TopEvent + FailureRate: Rate
TE1: TopEvent FailureRate: Rate
Rate + value: double + unit: string
4

Figure 38 - Process / product model and mapping model

Based on these different parts of the ODE semi-automated assurance case instantiation can be realized as depicted in Figure 39. As an input for the semi-automated instantiation the ODE provides:

1. Product models (e.g. hazards, failure logic propagation, functional model, etc);
2. Set of GSN/SACM patterns;
3. Process model describing the activities of safety standards;
4. Mapping model specifying how to automatically perform activities from the ODE process model on ODE-conformant product models.

Taking these models as input, a SACM AssuranceCasePackage can be generated semi-automatically, in which the GSN/SACM patterns are instantiated to generate the SACM ArgumentPackage. Moreover, a SACM TerminologyPackage and a SACM ArtifactPackage can be generated based on the ODE product model. Thus, interlinking safety argumentation and product model based on the information provided in the ODE process model and the mapping model. An example, for a safety argumentation for the ETCS trackside sub-system created by this semi-automated assurance case instantiation approach is depicted in Figure 40.

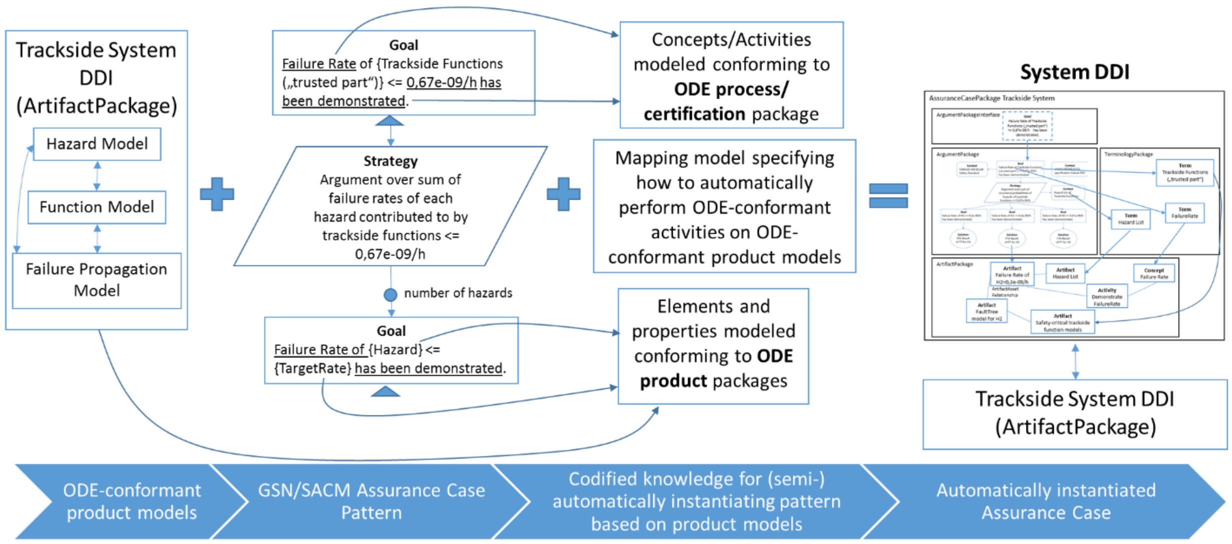

Trackside System DDI (ArtifactPackage)
Hazard Model
Function Model
Failure Propagation Model
Goal
Failure Rate of {Trackside Functions („trusted part“)} <= 0,67e-09/h has been demonstrated.
Strategy
Argument over sum of failure rates of each hazard contributed to by trackside functions <= 0,67e-09/h
number of hazards
Goal
Failure Rate of {Hazard} <= {TargetRate} has been demonstrated.
Concepts/Activities modeled conforming to ODE process/ certification package
Mapping model specifying how to automatically perform ODE-conformant activities on ODE-conformant product models
Elements and properties modeled conforming to ODE product packages
System DDI
Trackside System DDI (ArtifactPackage)
ODE-conformant product models
GSN/SACM Assurance Case Pattern
Codified knowledge for (semi-) automatically instantiating pattern based on product models
Automatically instantiated Assurance Case

Figure 39 - Process of the semi-automated assurance case instantiation based in the ODE

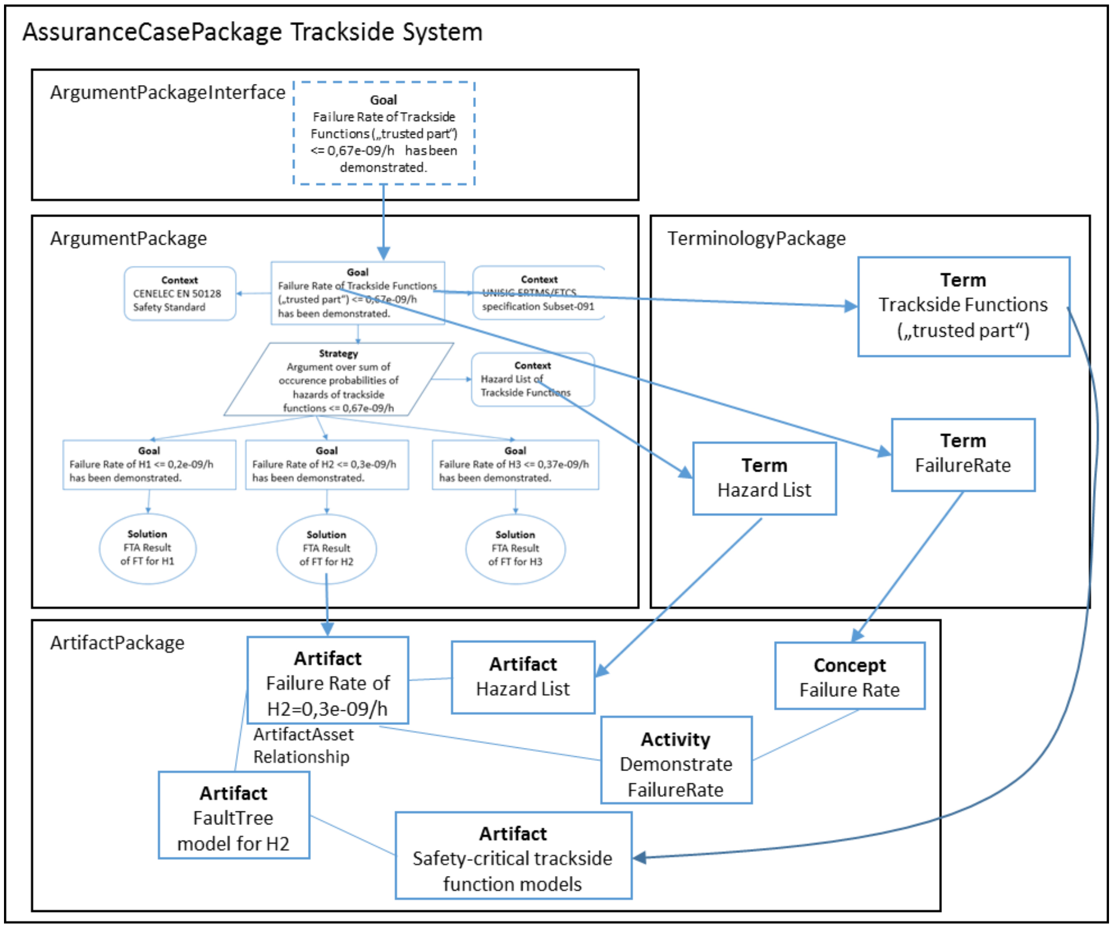

AssuranceCasePackage Trackside System
ArgumentPackageInterface
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
ArgumentPackage
Context
CENELEC EN 50128 Safety Standard
Goal
Failure Rate of Trackside Functions („trusted part“) <= 0,67e-09/h has been demonstrated.
Context
UNISIG ERTMS/ETCS specification Subset-091
Strategy
Argument over sum of occurence probabilities of hazards of trackside functions <= 0,67e-09/h
Context
Hazard List of Trackside Functions
Goal
Failure Rate of H1 <= 0,2e-09/h has been demonstrated.
Goal
Failure Rate of H2 <= 0,3e-09/h has been demonstrated.
Goal
Failure Rate of H3 <= 0,37e-09/h has been demonstrated.
Solution
FTA Result of FT for H1
Solution
FTA Result of FT for H2
Solution
FTA Result of FT for H3
TerminologyPackage
Term
Trackside Functions („trusted part“)
Term
Hazard List
Term
FailureRate
ArtifactPackage
Artifact
Failure Rate of H2=0,3e-09/h
Artifact
Hazard List
Concept
Failure Rate
ArtifactAsset Relationship
Activity
Demonstrate FailureRate
Artifact
FaultTree model for H2
Artifact
Safety-critical trackside function models

Figure 40 - DDI in form SACM AssuranceCasePackage for the ETCS trackside part

4.1.3 Optimized supplier component selection based on DDI

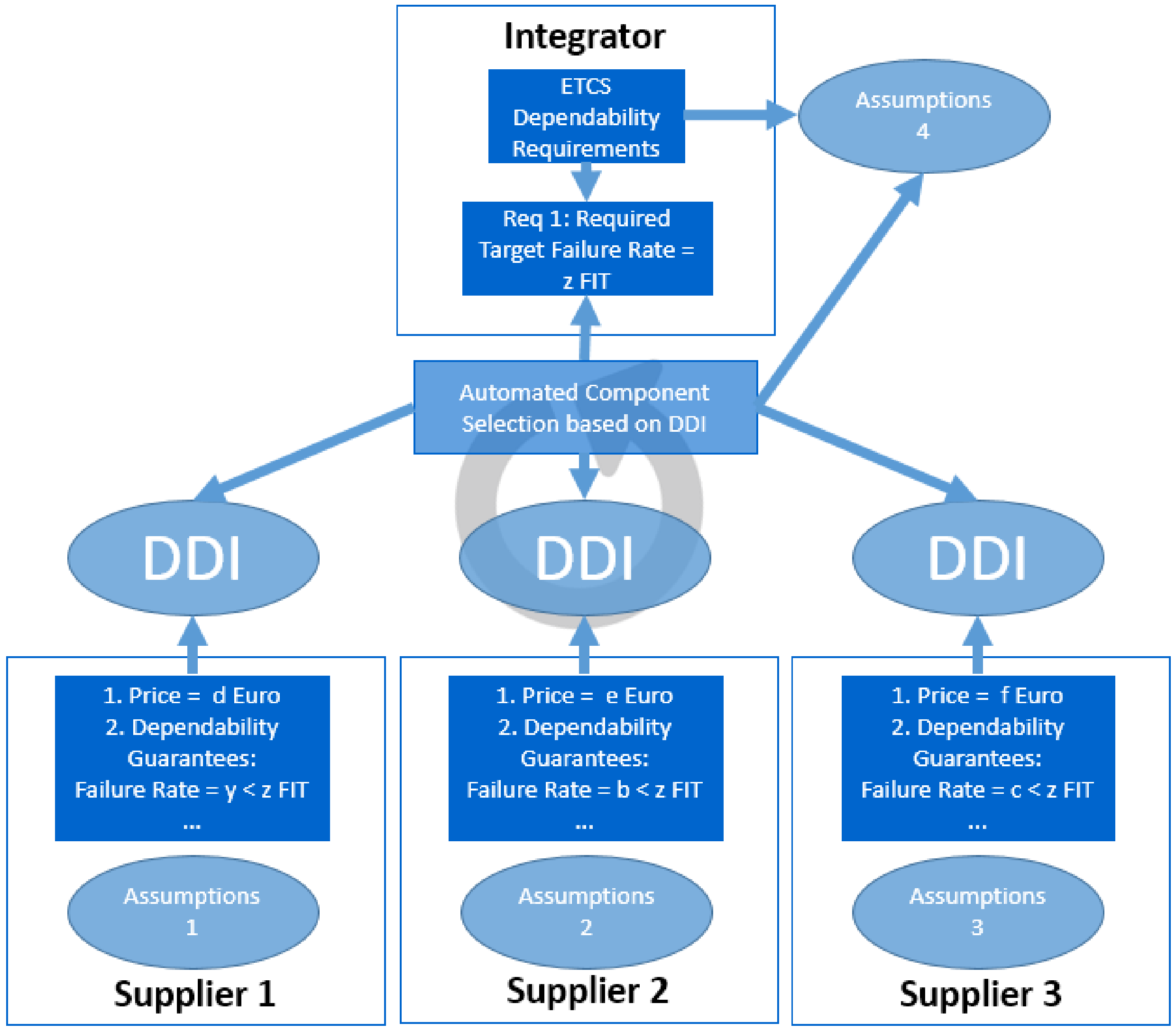

Integrator
ETCS Dependability Requirements
Req 1: Required Target Failure Rate = z FIT
Assumptions 4
Automated Component Selection based on DDI
DDI
DDI
DDI
1. Price = d Euro 2. Dependability Guarantees: Failure Rate = y < z FIT ...
1. Price = e Euro 2. Dependability Guarantees: Failure Rate = b < z FIT ...
1. Price = f Euro 2. Dependability Guarantees: Failure Rate = c < z FIT ...
Assumptions 1
Assumptions 2
Assumptions 3
Supplier 1
Supplier 2
Supplier 3

Figure 41 - Selection of the suppliers based on DDI

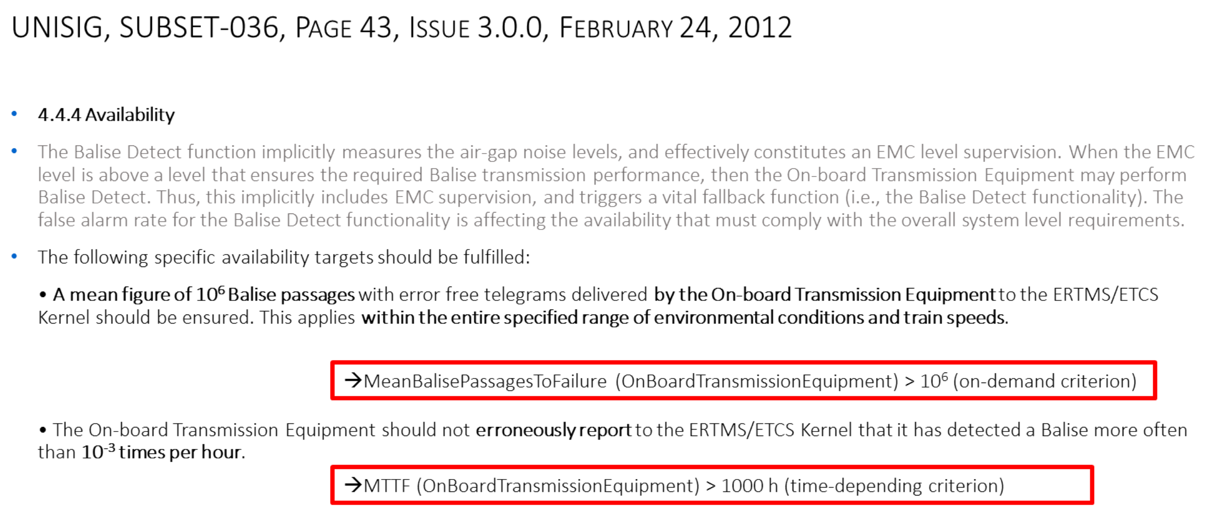
As depicted in Figure 41, attributes of a DDI (such as cost, minimum operation life time, logistic requirements etc.) will be utilized, for example, to select suppliers of system components. The criterion is that the target values (activity) of all attributes shall be fulfilled. DDI could be seen in this scenario as a formalized component datasheet filled with values of aforementioned attributes from suppliers. This datasheet will be used by OEM (integrator) to optimize the component selection. The process begins with providing suppliers' DDIs to the OEM. Afterwards, the OEM compares the attributes values of the different suppliers' datasheets (DDIs) and the target requirements stored in the OEM database. For instance, in the ETCS use case, the selection of a Balise sensor could be done by this comparison based on checking the attribute values of DDI.

Additionally, the individual assumptions of the supplier's failure rate shall be compared with the assumption of the target failure rate. In UNISIG Subset-091 (ETCS/ERTMS, 2016) the operational assumptions are related to the drivers' reactions for certain events. Such requirements shall be considered during the decision making. Other operational parameters such as time spent for certain driving mode, frequency of radio signal between track and train etc. shall also be taken into account during the attribute comparison.

Figure 41 demonstrates also the decomposition of the overall failure rate into subsystem/subfunction/subcomponent failure rates. In this figure, there are 3 subcomponents that are working in parallel. This means, there 3 subcomponents are connected by a "or" connector (not visualized in the picture). The decomposition will be performed by the integrator (the OEM). After the decomposition, the failures rates of the subfunction/subsystem/subcomponent are assigned or calculated based on system information such as architecture.

In addition to the operational assumptions, environment conditions (operational environment in UNISIG Subset-091) shall be checked. Such environment conditions are temperature, vibration, electromagnetic interference etc (ETCS/ERTMS, 2016). For instance, if we assume the target failure rate is 2 fit at a temperature condition of 40 °C, a supplier's component failure rate of 3 fit at 30 °C will obviously not be selected.

In , the requirement of availability in the sense of MTTF of "OnBoardTransmissionEquipmnt" shall be greater than 1000 h for certain environment conditions. The related environment conditions are essential to perform the comparison, otherwise this comparison could lead to total wrong decision



UNISIG, SUBSET-036, PAGE 43, ISSUE 3.0.0, FEBRUARY 24, 2012

- **4.4.4 Availability**
- The Balise Detect function implicitly measures the air-gap noise levels, and effectively constitutes an EMC level supervision. When the EMC level is above a level that ensures the required Balise transmission performance, then the On-board Transmission Equipment may perform Balise Detect. Thus, this implicitly includes EMC supervision, and triggers a vital fallback function (i.e., the Balise Detect functionality). The false alarm rate for the Balise Detect functionality is affecting the availability that must comply with the overall system level requirements.
- The following specific availability targets should be fulfilled:

• **A mean figure of 10^6 Balise passages** with error free telegrams delivered **by the On-board Transmission Equipment** to the ERTMS/ETCS Kernel should be ensured. This applies **within the entire specified range of environmental conditions and train speeds.**

→MeanBalisePassagesToFailure (OnBoardTransmissionEquipment) > 10^6 (on-demand criterion)

• The On-board Transmission Equipment should not **erroneously report** to the ERTMS/ETCS Kernel that it has detected a Balise more often than **10^{-3} times per hour.**

→MTTF (OnBoardTransmissionEquipment) > 1000 h (time-depending criterion)

Figure 44 - Environment conditions check for certain MTTF requirements

.

A possible workflow of attribute comparison could contain:

- Checking MTTF;
- Checking environmental conditions of the corresponding context;

- checking driver reaction assumption;
- comparison of MTTF regarding assumption and condition.

It is meaningful to observe the dependency between the different operational parameters and the target value e.g. for availability analysis. An example usage of DDI is to facilitate the cross references of data elements, such as attributes, between documents. The references are defined for example in the ODE meta-model of the DDIs. In order to automate data exchange between DDIs, the formalized component datasheet (as develop time DDI) will be integrated in the DDI attribute comparison workflow. Through this integration, optimization or automation support can be executed. On the integrator side, the selection of supplier/s can be made more effective based on the comparison results. Nevertheless, the safety argumentation (in form of GSNs represented in the DDI) of the suppliers can be integrated by the ODEM, if the suppliers provide own GSN with consideration of same operational assumption and operational condition among other attributes. In additional to failure rate check, the overall system goals and compatible sub goals, along with different dependability requirements, can be checked regarding their fulfillment.

4.2 Utilization of DDIs in security and safety co-engineering in the context of the Physiological Health Monitoring Use Case

4.2.1 Use case overview

The Dependable Physiological Monitoring System (DPMS) is an advanced technology applied to a connected vehicle and is capable of measuring physiological parameters and evaluating the health condition of driver and other occupants. The proposed Single-Photon Avalanche Diode (SPAD) array imaging system can identify Heart Rate (HR), Respiration Rate (RR), Heart Rate Variability (HRV), Inter-Beat Interval (IBI) and Oxygen-Saturation (SpO2) in real time.

This brand-new feature contains a comprehensive package of technologies, tools and services that support the drive session in evaluating the health status. Being introduced to new generation connected vehicles, with (even limited) autonomous capabilities in the case of health emergency condition, it shall process hard decision making at run-time and trigger different “actions” to mitigate the risk of vehicle accidents and people injuries.

The vehicle decision mechanism allows the DPMS to take control over autonomous driving features when needed, moving the vehicle to the emergency lane or the nearest safety area, or in case of great danger, driving the vehicle to the Hospital / First Aid Services.

The decision-making computation may result in one of three different paths (Figure 42- DPMS decision-making action paths).

From a safety perspective, all the possible paths shall evaluate the risk to health. As shown in the model (Figure 42), part of the safety argumentation requirements enable different features on the vehicle, for example, enabling autonomous driving features, moving the vehicle to the emergency lane / the nearest safe area, or in case of high danger, drive the vehicle to the Hospital / First Aid Services (depending on autonomous level capabilities).

The physiological data (including the streaming video) is sensitive with regards to GDPR, therefore the requirements from the recent GDPR regulation need to be implemented.

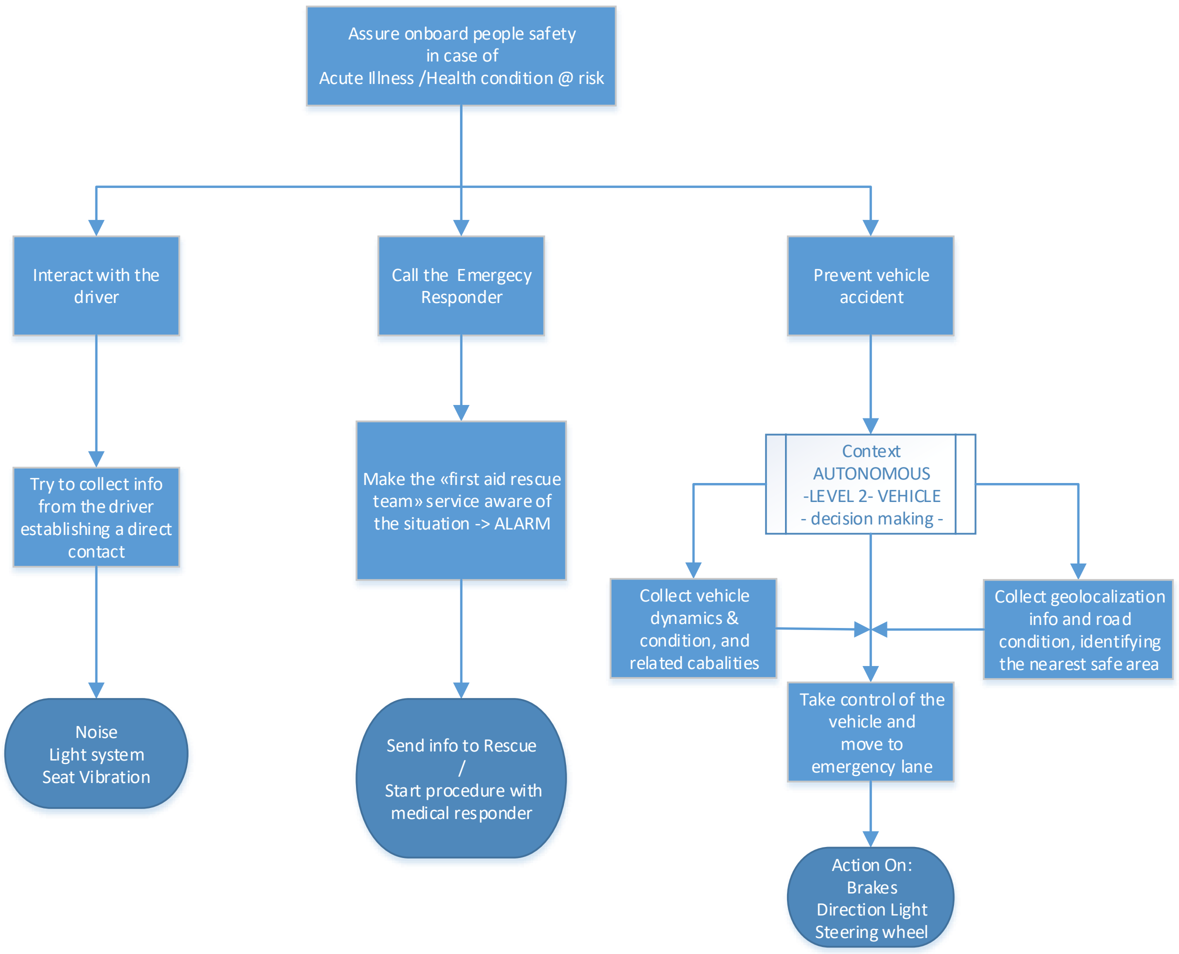

Assure onboard people safety in case of Acute Illness /Health condition @ risk
Interact with the driver
Call the Emergecy Responder
Prevent vehicle accident
Try to collect info from the driver establishing a direct contact
Make the «first aid rescue team» service aware of the situation -> ALARM
Context AUTONOMOUS -LEVEL 2- VEHICLE - decision making -
Collect vehicle dynamics & condition, and related cabalities
Collect geolocalization info and road condition, identifying the nearest safe area
Take control of the vehicle and move to emergency lane
Noise
Light system
Seat Vibration
Send info to Rescue / Start procedure with medical responder
Action On:
Brakes
Direction Light
Steering wheel

Figure 42- DPMS decision-making action paths

4.2.2 Addressed challenge in safety-security co-engineering

In order to integrate the DPMS features with a CPS which already has predefined safety ASIL level and ISO standard compliance, the DPMS shall guarantee accuracy and robustness at automotive grade level, like all the other vehicle on-board systems.

The dynamic nature of the CPS introduces the potential for security threats to contribute risks towards safety as well. In current practice, safety and security analysis are handled by different methods, organizational units and techniques. So, this integration process addresses safety/security co-engineering challenges between two or more domains.

As per the GM legacy process, the DPMS project has to be verified and validated across different technical review boards that have expertise in different competencies, such as Security and Safety.

On the security side, DMPS requirements have been analysed by a Cybersecurity technical board. After the requirements review, the experts identified the criticalities by employing the Threat Assessment and Remediation Analysis (TARA) methodology and related Attack Trees models. This approach identified and assessed cyber vulnerabilities and selected countermeasures which were effective at mitigating those vulnerabilities.

Figure 43 portrays the interaction of project requirement, threat analysis and identified countermeasure in a graphical perspective.

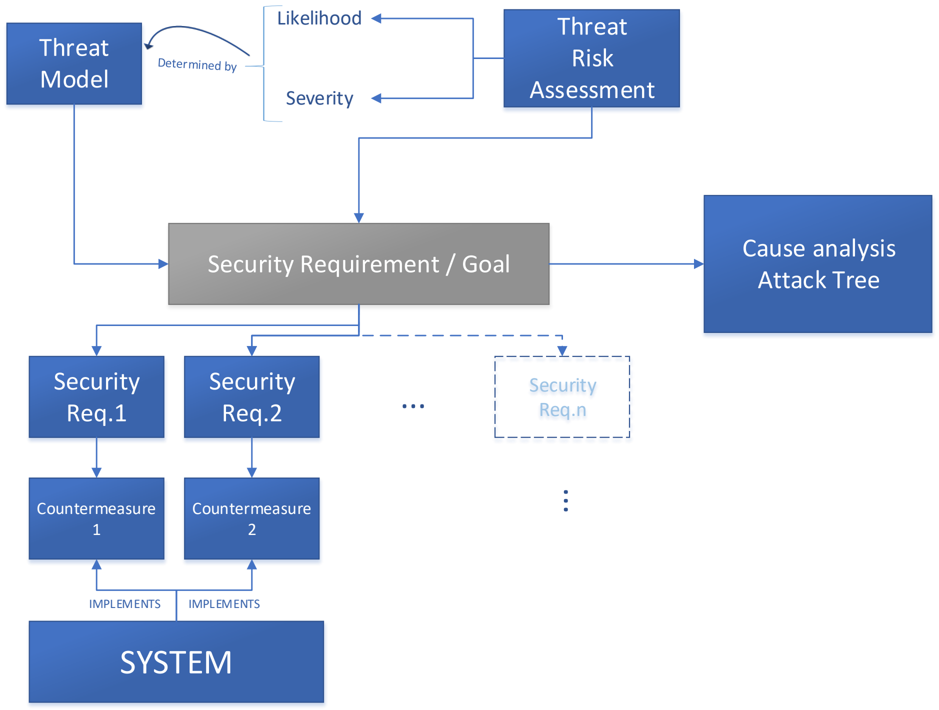


Figure 43 - DPMS general security model

The TARA methodology enables detailed modelling of the potential security threats to the system's critical elements and identifies appropriate requirements and counter-measures to mitigate the cumulative risk.

Further refinement of the causes that can lead to an attack being successful, and highlight vulnerabilities, is performed using an appropriate qualitative analysis technique, such as security attack trees.

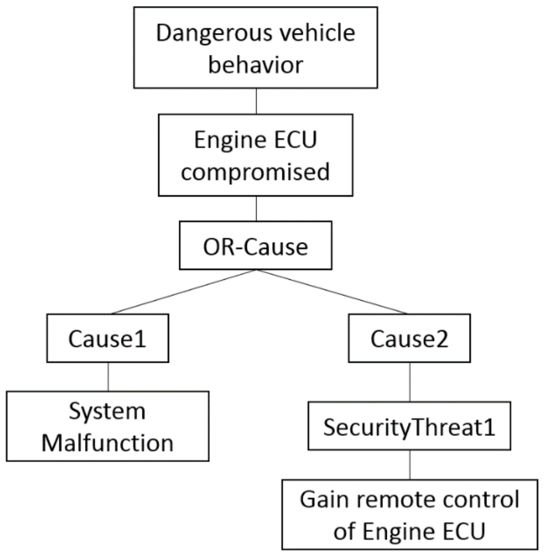


Figure 44: Attack Tree example

Attack trees have been used as conceptual diagrams to highlight assets/targets that might be attacked and the related safety effect (see engineering stories in the next section). Security attack trees share a similar structure with fault trees, differing mainly in the semantics of the base and intermediate causes and their effects.

From this point on, a Safety-Security Hybrid Analysis can qualitatively link to Security Violations and to system Failures and safety Hazards.

AS shown in Figure 44, dangerous vehicle behavior can be caused by either a System malfunction, typical RAMS issue (Failure), or a Security Threat (Attack). This can lead to gaining control of vehicle stability controller.

This may also affect the DPMS, where being hijacked, it may produce misleading results and trigger an “unexpected action” that causes the vehicle to unexpectedly takes over the control of the vehicle.

Hybrid analysis enables better communication of issues spanning different dependability domains.

The new version of the ODE v2 seen on section 3 exploits the similarity seen within attack trees, enhancing the definition of fault trees containing both security and safety-related events. This enables security threats identified by TARA to be analysed via security attack trees and linked to fault trees, all under the singular DDI structure.

4.2.3 Long cycle times integration processes at development time

The legacy process requires verification and validation approval from different review boards that have expertise in different competencies, such as Security and Safety.

Each board works as a standalone entity and analyses each project through a legacy process, with different levels of (gate) approval.

Being two (or more) different company entities with different processes, the integration between these competencies (e.g. security and safety), typically results in additional work for the team, who have to synchronize the (different) implementation guidelines provided by both boards, and double check the potential impact (Figure 45).

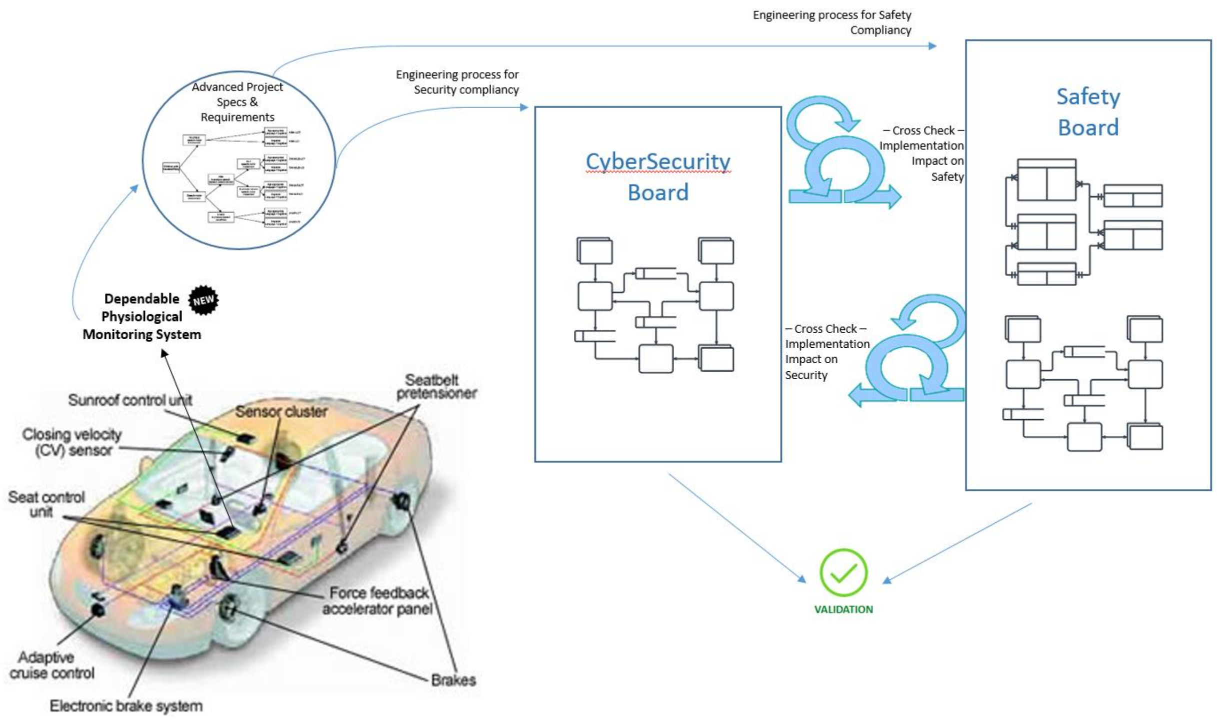

Engineering process for Safety Compliancy
Engineering process for Security compliancy
Advanced Project Specs & Requirements
CyberSecurity Board
Safety Board
– Cross Check – Implementation Impact on Safety
– Cross Check – Implementation Impact on Security
Dependable Physiological Monitoring System
NEW
VALIDATION
Sunroof control unit
Sensor cluster
Seatbelt pretensioner
Closing velocity (CV) sensor
Seat control unit
Force feedback accelerator panel
Adaptive cruise control
Brakes
Electronic brake system

Figure 45 - Internal co-engineering process

This is a very time consuming activity and may result in increased time-to-market.

The adoption of the DDI methodology at development time shall help the team during the design phase, on the following aspects:

- adopting the security-safety co-engineering by design;
- helping identifying criticalities and marginalities of the project without the needs to involve technical board in early phase;
- implementing state-of-the-art cybersecurity solution already available on the market.

The process shall result in faster development time, cutting out intermediate approval loops, allowing the team to start work early, and submitting a first "security proof " design to review board. The boards are still responsible for approval at the end of this first design round, because they are still the centre of expertise and responsible for the validation.

4.2.4 Engineering Story 1: Unauthorized Emergency Brake

In this case, we utilize DDIs to protect against a safety-relevant attack (hijacking), during the development lifecycle of the DPMS, and evaluate how the DEIS approach eases the development process in achieving the security standard required at production level.

The engineering story analyses a condition where the malware threat is intended to corrupt the vehicle safety & stability onboard system, activating the automatic breaking features, typically used to prevent potential accident. The attack hijacks the algorithm that processes the vehicle stability data coming from the sensors, and the effect is the unauthorized activation of the emergency brake mechanism, which in this case may cause collision or loss of control of the vehicle (Figure 46).

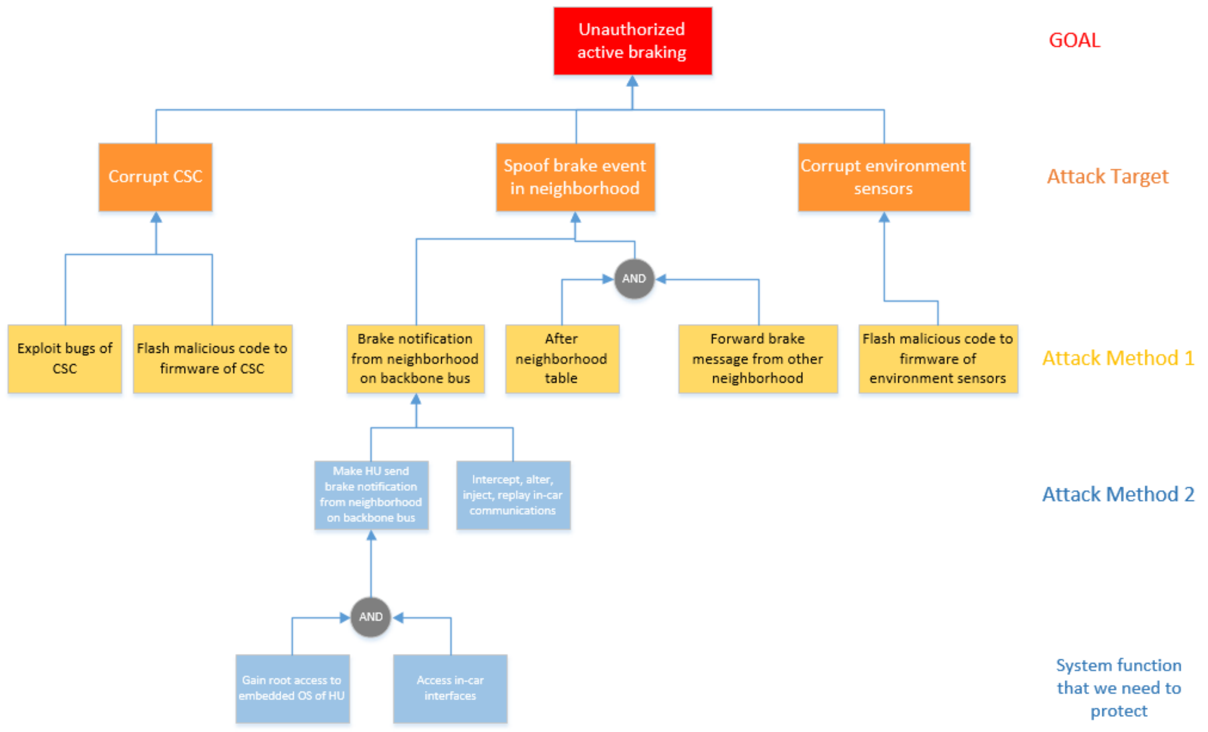

Unauthorized active braking
GOAL
Corrupt CSC
Spoof brake event in neighborhood
Corrupt environment sensors
Attack Target
AND
Exploit bugs of CSC
Flash malicious code to firmware of CSC
Brake notification from neighborhood on backbone bus
After neighborhood table
Forward brake message from other neighborhood
Flash malicious code to firmware of environment sensors
Attack Method 1
Make HU send brake notification from neighborhood on backbone bus
Intercept, alter, inject, replay in-car communications
Attack Method 2
AND
Gain root access to embedded OS of HU
Access in-car interfaces
System function that we need to protect

Figure 46 - Unauthorized Emergency Brake Attack Tree

From a more generic perspective, an attack against an embedded OS like the one that is running on vehicle HMI, is intended to gain control of a privileged (ID=root) process and to act as admin. The hijack allows the attacker to perform unauthorised action, for example, stealing private date coming from the vehicle (drive profile, scoring system, vehicle position), or tampering with the vehicles' sensor information which is exchanged through the vehicle network (Figure 47).

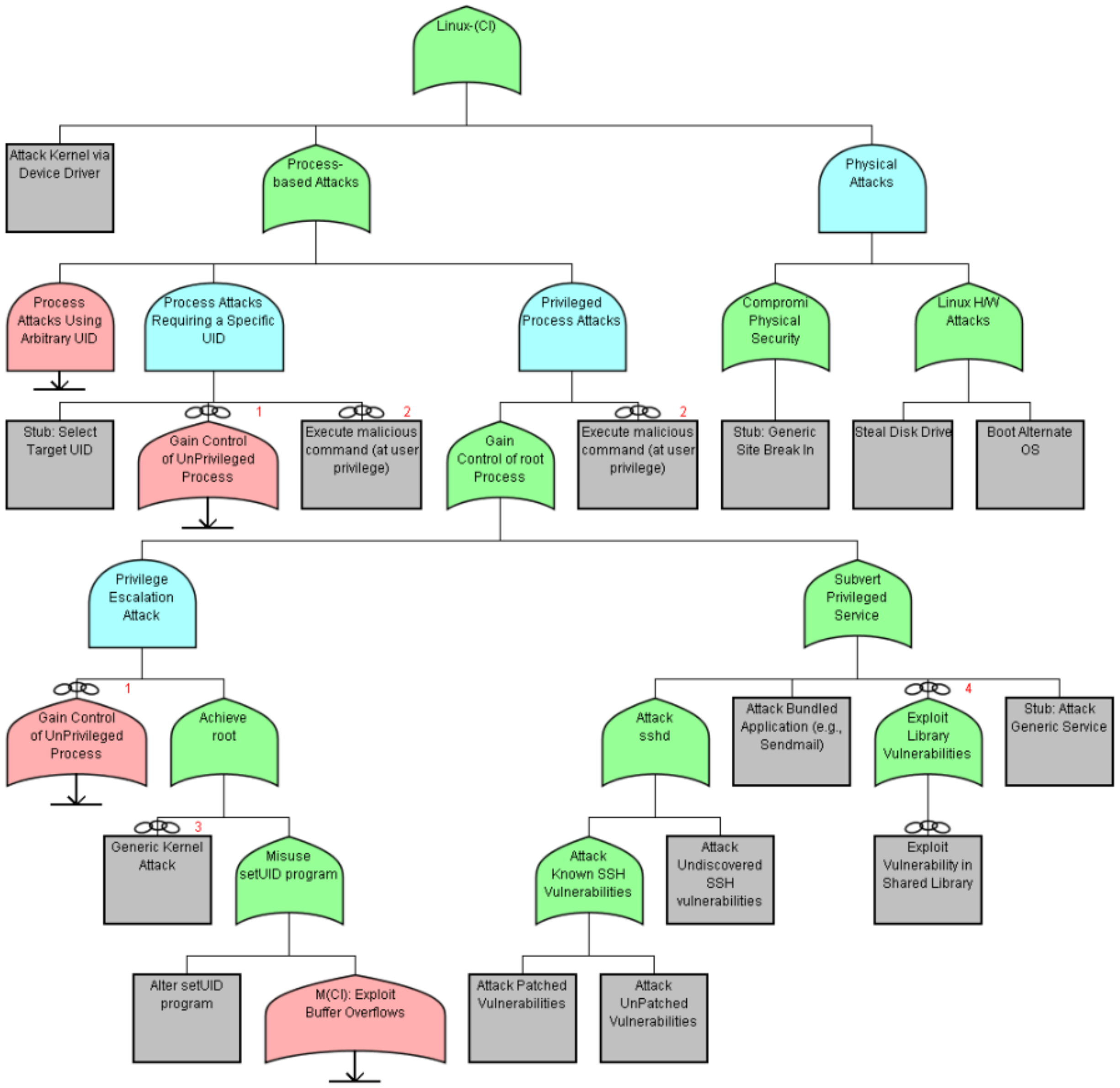

Linux-(CI)
Attack Kernel via Device Driver
Process-based Attacks
Physical Attacks
Process Attacks Using Arbitrary UID
Process Attacks Requiring a Specific UID
Privileged Process Attacks
Compromi Physical Security
Linux H/W Attacks
Stub: Select Target UID
1
Gain Control of UnPrivileged Process
2
Execute malicious command (at user privilege)
Gain Control of root Process
2
Execute malicious command (at user privilege)
Stub: Generic Site Break In
Steal Disk Drive
Boot Alternate OS
Privilege Escalation Attack
Subvert Privileged Service
1
Gain Control of UnPrivileged Process
Achieve root
Attack sshd
Attack Bundled Application (e.g., Sendmail)
4
Exploit Library Vulnerabilities
Stub: Attack Generic Service
3
Generic Kernel Attack
Misuse setUID program
Attack Known SSH Vulnerabilities
Attack Undiscovered SSH vulnerabilities
Exploit Vulnerability in Shared Library
Alter setUID program
M(CI): Exploit Buffer Overflows
Attack Patched Vulnerabilities
Attack UnPatched Vulnerabilities

Figure 47 - Linux-Embedded-OS Attack Tree

The Linux embedded operating system, may also be affected by injection of malicious software (Figure 48) and result in a system’s unpredictable behavior, vehicle hijacking, and broken integrity and confidentiality.

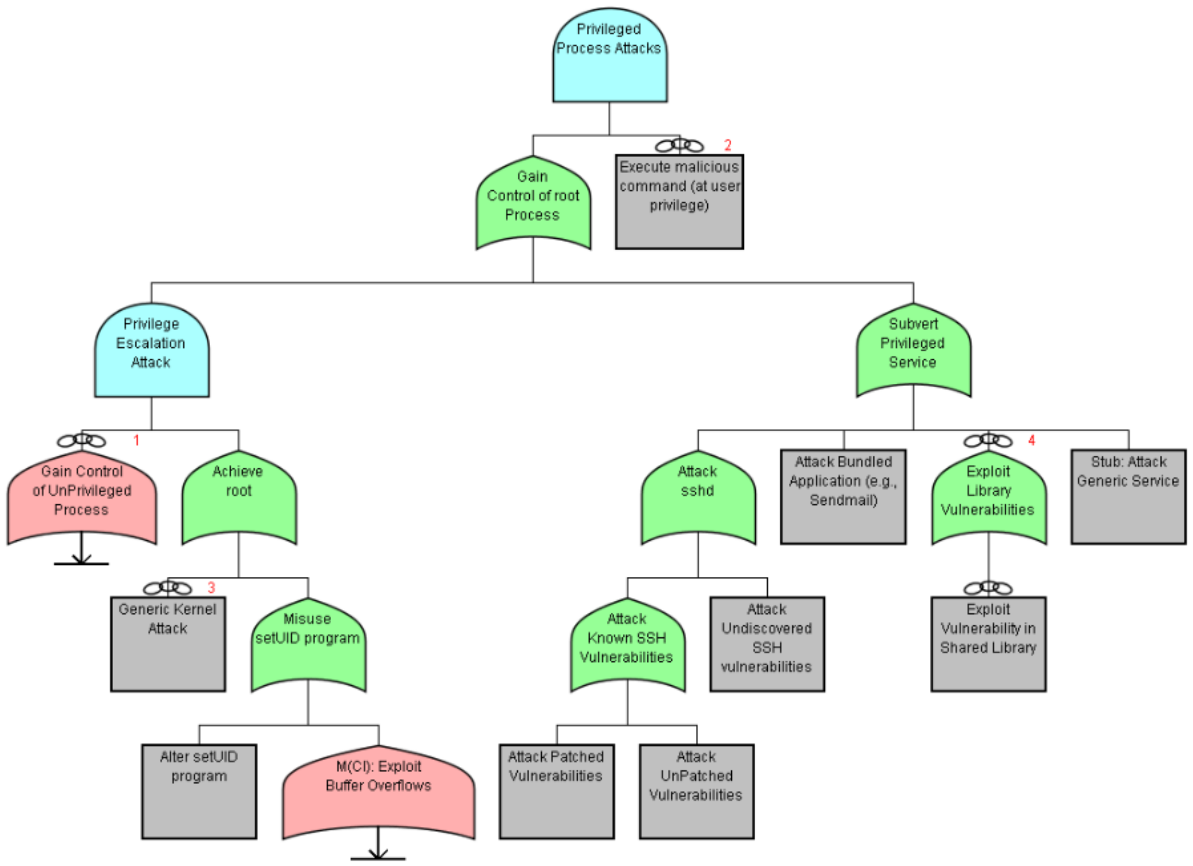

Privileged Process Attacks
Gain Control of root Process
2
Execute malicious command (at user privilege)
Privilege Escalation Attack
Subvert Privileged Service
1
Gain Control of UnPrivileged Process
Achieve root
3
Generic Kernel Attack
Misuse setUID program
Alter setUID program
M(CI): Exploit Buffer Overflows
Attack sshd
Attack Bundled Application (e.g., Sendmail)
4
Exploit Library Vulnerabilities
Stub: Attack Generic Service
Attack Known SSH Vulnerabilities
Attack Undiscovered SSH vulnerabilities
Exploit Vulnerability in Shared Library
Attack Patched Vulnerabilities
Attack UnPatched Vulnerabilities

Figure 48 - Privileged Process Attack Tree

4.2.5 Engineering Story 2: Vehicle unintended behavior

The engineering story is supposed to protect against an attack intended to corrupt the emergency manager onboard system that takes the control of the vehicle (activating autonomous LV2 features) to prevent a potential accident due to the driver's acute illness condition.

The attack hijacks the algorithm that processes raw physio parameters of the driver and extracts altered health condition, or corrupts raw data (heart rate, breath rate, SpO2) before being processed.

The effect is the unauthorized activation of features strictly linked to autonomous LV2, used from the emergency manager to move the vehicle in safe condition (Figure 49). This may cause collision or loss of control of the vehicle.

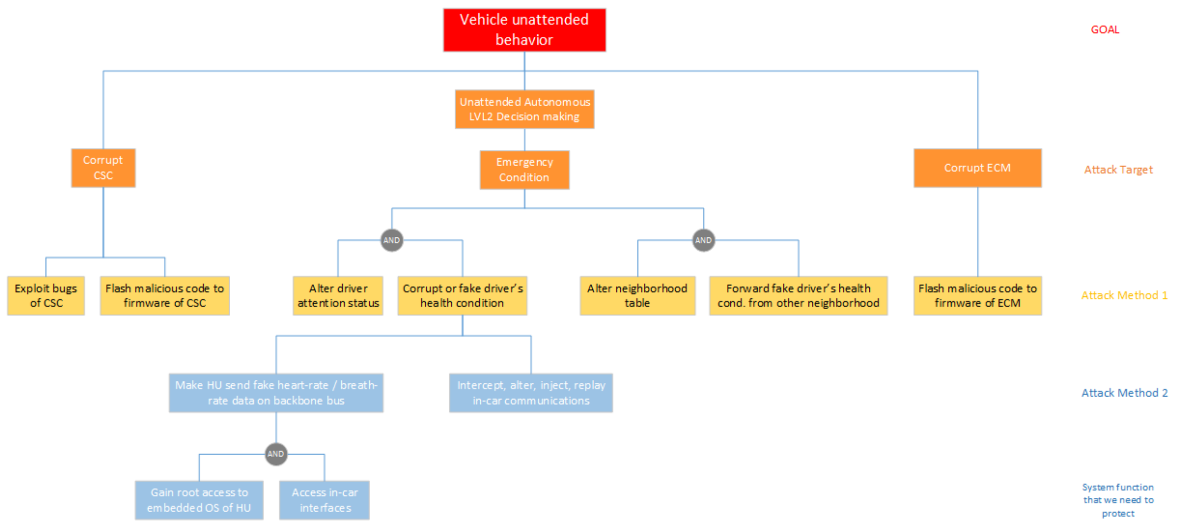

Vehicle unattended behavior
GOAL
Unattended Autonomous LVL2 Decision making
Corrupt CSC
Emergency Condition
Corrupt ECM
Attack Target
AND
AND
Exploit bugs of CSC
Flash malicious code to firmware of CSC
Alter driver attention status
Corrupt or fake driver's health condition
Alter neighborhood table
Forward fake driver's health cond. from other neighborhood
Flash malicious code to firmware of ECM
Attack Method 1
Make HU send fake heart-rate / breath-rate data on backbone bus
Intercept, alter, inject, replay in-car communications
Attack Method 2
AND
Gain root access to embedded OS of HU
Access in-car interfaces
System function that we need to protect

Figure 49 - Vehicle unattended behavior Attack Tree

4.2.6 ES3: Compromise driver's privacy

The engineering story is supposed to deal with a potential attack in the vehicle CPS intended to steal drivers' data, for example:

- Raw physio parameters (Hear Rate, Breath Rata, SpO2);
- Health data post processed locally by physio board;
- Single frame (image) of the face of driver stored in volatile memory used to extract physio parameter;
- GPS date coming from the infotainment system;
- Drive style data (used for driver score algorithm).

In this case the privacy aspect is impacted, and personal and sensitive data may be stolen for unauthorized usage, both inside and outside the CPS. This implies the DDI shall guarantee the proper application of GDPR regulation that specifies the action to be taken in case of data breach inside a system (Figure 50).

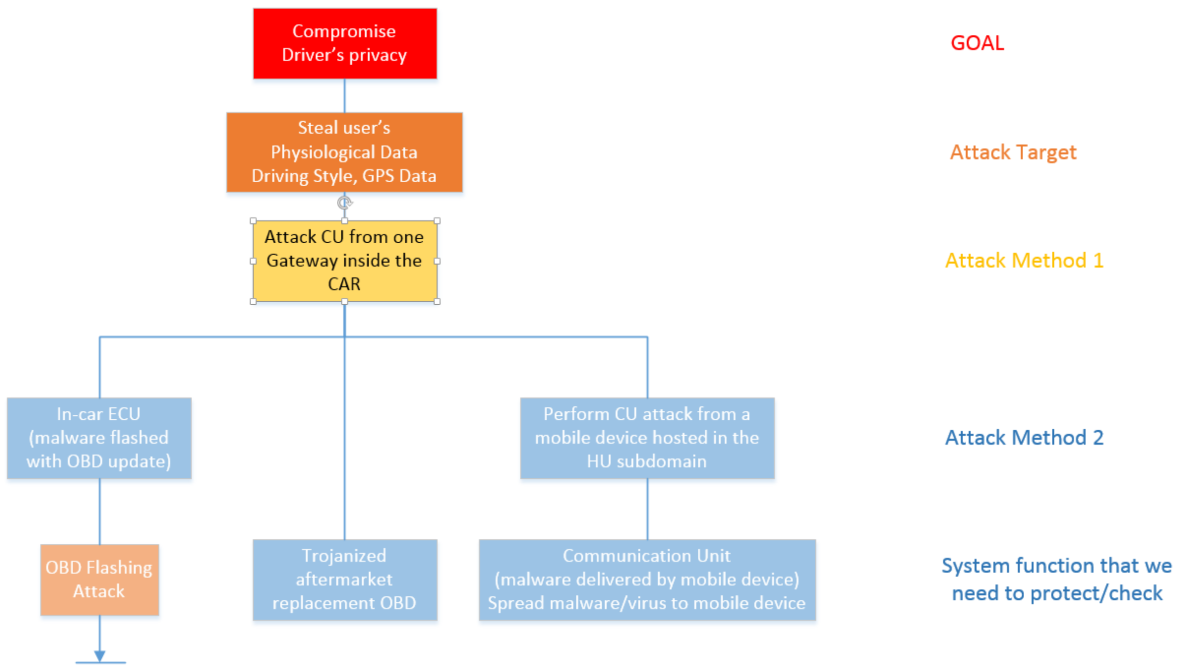

Compromise Driver's privacy
Steal user's Physiological Data Driving Style, GPS Data
Attack CU from one Gateway inside the CAR
In-car ECU (malware flashed with OBD update)
Perform CU attack from a mobile device hosted in the HU subdomain
OBD Flashing Attack
Trojanized aftermarket replacement OBD
Communication Unit (malware delivered by mobile device) Spread malware/virus to mobile device
GOAL
Attack Target
Attack Method 1
Attack Method 2
System function that we need to protect/check

Figure 50 - Compromise driver's privacy Attack Tree

4.3 Utilization of DDIs in applications involving General Data Protection Regulations (GDPR)

4.3.1 Introduction

The GDPR lays down rules relating to the protection of natural persons with regard to the processing of personal data and rules relating to the free movement of personal data. GDPR lays out responsibilities for organisations to ensure the privacy and protection of personal data, provides data subjects with certain rights, and assigns powers to regulators to ask for demonstrations of accountability or even impose fines in cases where an organisation is not complying with GDPR requirements.

The organisations that need to be EU GDPR compliant are:

Companies (controllers and processors) established in the EU, regardless of whether or not the processing takes place within the EU;

Companies (controllers and processors) not established in the EU offering goods or services within the EU or to EU individuals.

A '**Processor**' means a natural or legal person, public authority, agency or other body which processes personal data on behalf of the controller. A '**Controller**' means the natural or legal person, public authority, agency or other body which, alone or jointly with others, determines the purposes and means of the processing of personal data. **'Processing'** means any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means, such as collection, recording, organisation, structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, restriction, erasure or destruction. **'Personal data'** means any information relating to an identified or identifiable natural person (data subject); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person. 'Personal data breach' means a breach of security leading to the accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to, personal data transmitted, stored or otherwise processed.

Section 4.3.2 provides the GDPR requirements. Section 4.3.3 provides the means as to how GDPR requirements shall be implemented within DDI.

4.3.2 GDPR Requirements

The requirements detailed in Table 6 are the GDPR requirements that have been elicited. It is envisaged that the number of GDPR requirements will increase and be further refined in upcoming work.

Table 6 - GDPR requirements (Refined Project Requirements for Semi-automation)

Requirement (ID and Name)	Category	Description
RQ_034 - Privacy by Design	Non-Functional	DDI should incorporate organizational and technical mechanisms to protect personal data in the design of new systems and processes; that is, **privacy and protection aspects should be ensured by default.**
RQ_035 - Data breach notification	Functional	DDI shall keep track of any **personal data breach**, to allow the regulator and data subject to be informed within 72 hours from the breach event.
RQ_036 - Data portability identification	Functional	In case of personal data, DDI has the accountability to identify and ensure the protection and privacy of personal data when that data is being **transferred outside to a third party and / or other entity.**
RQ_037 - Data perimeter	Functional	GDPR Privacy policy are applied considering the **origin of the personal data** (from where data comes, and where data has been collected). DDI that manages personal data shall **keep track about data origin**, to facilitate the regulator to apply data protection policy properly, country by country.

4.3.3 DDI Implementation of GDPR

In safety risk management, harm is considered as physical injury or damage to the health of people, or damage to property or the environment. However, because harm (in a security sense) can also include reduction in effectiveness, or breach of data and systems security, it is appropriate to create a security risk management process (as companion to safety risk management process) to allow the organization to assess the additional risks associated with effectiveness and system/data security. If the processes are integrated, there could be an inclination to drop the evaluation of those risks that do not lead to harm (in the safety sense), which can lead to incomplete or inconsistent security controls. Additionally, security risk assessment models typically use assessment factors that are different from safety risk assessment (while safety risk involves evaluating the probability and severity of a hazard leading to harm, security risk is based on an assessment of the likelihood that a threat will successfully exploit a device vulnerability, an event that could lead to an adverse impact due to a compromise of system confidentiality, integrity, and/or availability). Furthermore, integrating safety and security risk assessment into a single general risk management process may result in major modifications to a well-functioning safety risk management process. The relationship between security and safety risks is depicted in Figure 54.

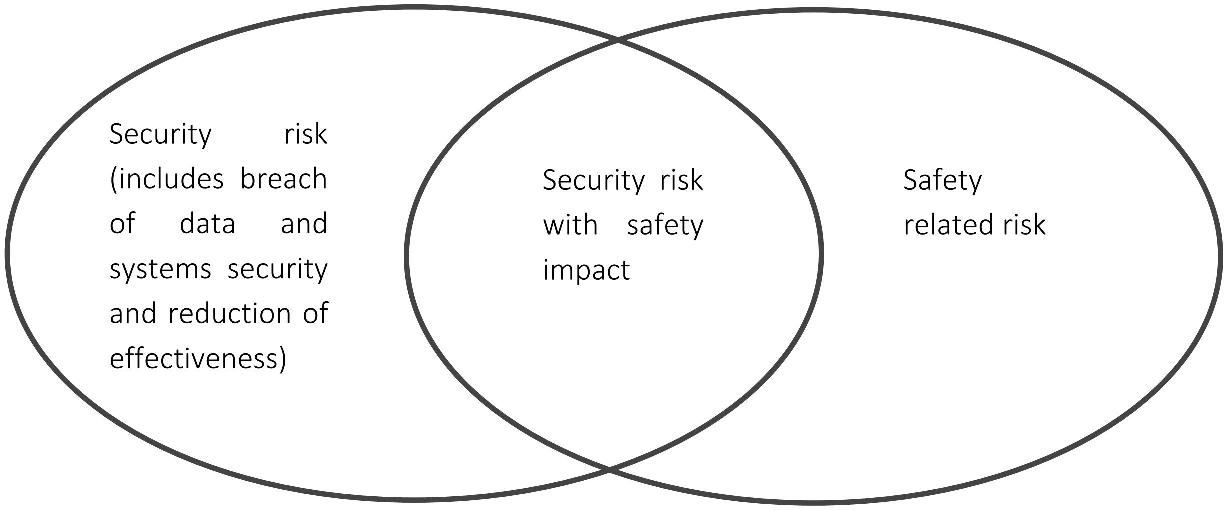

Security risk (includes breach of data and systems security and reduction of effectiveness)
Security risk with safety impact
Safety related risk

Figure 51 - Relationship between security and safety risks

Security risks that impact safety, should also be captured in the organization’s safety risk management process. A specific risk assessed as “must mitigate” in one model might be assessed as “does not need further mitigation” in the other. Risk control measure(s) should be applied to bring the risk into the acceptable range in both assessment models. There will be risks managed in the security risk assessment that are not propagated to the safety risk management process. An example would be a risk of compromise of the confidentiality of protected health information that is not considered harm (in the safety sense), but clearly requires mitigation by the security risk management process. There are also business and reputation risks associated with a security compromise that are not considered harm in the safety sense.

A security compromise that leads to harm (in the safety sense) should be managed within the security risk management process and propagated for assessment using the organisation’s safety risk management process. An example of a security risk that is also a safety risk is a malicious attacker gaining access to a medical device’s code, altering that code, and causing the device to malfunction. This malfunction may have the potential to cause harm to the patient.

As the purpose of GDPR is to ensure the privacy and protection of personal data, and provide data subjects with certain rights, it is proposed that GDPR requirements are handled by the DDI’s security package

(TARA). Any compromise of GDPR requirements that could lead to harm in the safety sense should be propagated for assessment in the DDI's HARA.

As exemplar, consider the following Portable Medical engineering story:

Context: Compromise ONCOassist User data

Description: ONCOassist user data is captured on initial signup to the platform. All details are stored on server located in Republic of Ireland. PMT is both a 'processor' and 'controller' of personal data, and under GDPR rules PMT must ensure the privacy and protection of personal data.

Challenge: Protect against an attack intended to steal client data such as name, address etc.

DDI implementation: GDPR requires a risk based approach to data security. Article 35 requires companies to perform data protection impact assessments (the controller shall, prior to processing, carry out an assessment of the impact of the envisaged processing operations on the protection of personal data) to assess and identify risks to individuals' data. In this engineering story the potential for theft of personal data is considered a security risk and will be dealt with using the **TARA** which assesses the *likelihood* of a successful attack, in addition to its *impact*. If the result of this assessment warrants threat mitigation then controls that ONCOassist consider include data minimisation, pseudonymisation etc.

A guide to assessing an organisations compliance with GDPR is included in Appendix A.

5 Summary and Outlook

This document presented our approach to conceptualizing and designing the engineering framework for the semi-automated generation and integration of design time DDIs.

To this end, section 2 first described a set of representative engineering scenarios that exemplify, at which points in the engineering lifecycle DDIs can support which specific engineering activities that are either inefficient or not achievable without the use of DDIs. Afterwards, the fundamental concept for realizing the engineering activities by (semi-) automated DDI execution is described along with the following four technical building blocks: 1. SACM-conformant assurance case*;* 2. the ODE product meta-model describing meta-model packages for different dependability aspects such as hazard, failure logic, architecture or security-related models; 3. The ODE certification meta-model describing a meta-model for modeling engineering activities with variable degree of formalism; 4. A DDI execution component that allows for (semi-)automatically executing dependability certification activities (conforming to packages in ODE::Certification), which operate on dependability aspect models (conforming to packages of ODE::Product) to synthesize, integrate or assess a dependability assurance case (conforming to OMG SACM).

Section 3 presented the reference for the second version of the Open Dependability Exchange (ODE) Meta-model by depicting all existing meta-model packages graphically along with textual description about changes to ODE v1. In addition, the harmonization with other commonly used meta-modeling languages such as SysML or EAST-ADL(2) was described.

The abstract engineering tasks that have been used in Section 2 for exemplifying DDI usage were concretized in Section 4 in the context of the DEIS use cases. For the European Train Control System (ETCS), an OEM-Tier integration scenario has been examined. For the Physiological Health Monitoring use case, security and safety co-engineering aspects have been considered. In addition, thoughts have been described on the utilization of DDI in applications, where general data protection regulations (GDPR) are relevant.

The DDI engineering framework concept described in this document is the follow up from the initial DDI concept, and its operationalization through the ODE meta-model, that have been presented in an earlier stage of the project. It builds the basis for using DDIs to automate engineering tasks at design time. The most notable innovations described in this document have been the creation of a package to formally model certification activities and a package to address security aspects such as threat and risk analyses or attack tree modeling. Moreover, the engineering framework has been developed with the upcoming step towards runtime dependability assurance in mind.

Upcoming work in the concept work package will focus on: 1. the concretization of the certification activity meta-model; 2. the refinement of the DDI execution component specification; 3. the conceptualization of runtime DDIs. Apart from conceptual research, the concepts of this document will be implemented in the dependability collaboration workspace (safeTbox, HiP-HOPS, ComposR and ACME as the dependability tools considered within DEIS) to demonstrate technical feasibility of the DDI engineering framework.

References

Andrews, Z., Payne, R., Romanovsky, A., Dider, A., & Mota, A. (2014). *Static Fault Analysis Support - Technical Manual.* COMPASS Project EU. Retrieved July 2018, from http://www.compass-research.eu/Project/Deliverables/D33_3b%20final.pdf

Biehl, M., Chen, D., & Torngren, M. (2010). Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems. *LCTES '10 Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems* (pp. 125-131). Stockholm, Sweden: ACM.

Biggs, G., Sakamoto, T., & Kotoku, T. (2014). A profile and tool for modeling safety information with design information in SysML. *Software & Systems Modeling, 15*(1), 147-178. doi:https://doi.org/10.1007/s10270-014-0400-x

CENELEC. (2003). EN 50129 Railway Applications: Safety related electronic systems for signalling.

DEIS Consortium. (2017). D2.1: Project Requirement.

DEIS Consortium. (2018). D3.1 Specification of the ODE metamodel and documentation of the fundamental concept of DDI.

ETCS/ERTMS. (2016, 05 12). Safety Requirements for the Technical Interoperability of ETCS in Levels (Subset-091, Issue: 3.6.0).

Friedenthal, S., Moore, A., & Steiner, R. (2008). *A Practical Guide to SysML The Systems Modeling Language* (1st ed.). Palo Alto, CA, USA: Elsevier. doi:https://doi.org/10.1016/b978-0-12-374379-4.x0001-x

Geoffrey Biggs/AIST. (2017). *SafeML 1.1.1 Documentation*. Retrieved September 15, 2018, from SafeML Introduction: https://staff.aist.go.jp/geoffrey.biggs/safeml/

Mian, Z., Bottaci, L., Papadopoulos, Y., & Biehl, M. (2012). System Dependability Modelling and Analysis Using AADL and HiP-HOPS. *14th IFAC Symposium on Information Control Problems in Manufacturing. 45 (6)*, pp. 1647-1652. Bucharest, Romania: Elsevier. doi:https://doi.org/10.3182/20120523-3-RO-2023.00334

Schneider, D., & Trapp, M. (2013). Conditional Safety Certification of Open Adaptive Systems. *ACM Trans. Auton. Adapt. Syst. (ACM Transactions on Autonomous and Adaptive Systems), 8*(2), pp. 1-20. doi:10.1145/2491465.2491467

Trapp, M., Weiss, G., & Schneider, D. (2018). Towards safety-awareness and dynamic safety management. *Proceedings of IEEE 14th Eurpean Dependable Computing Conference (EDCC).*

Zoe, A., Payne, R., Romanovsky, A., Dider, A., & Mota, A. (2014). *Static Fault Analysis Support - Technical Manual.* COMPASS Project EU. Retrieved July 2018, from http://www.compass-research.eu/Project/Deliverables/D33_3b%20final.pdf

Appendix A

The following questions provide a guide to assessing an organisations compliance with GDPR

1. Are both the legal basis and the purpose for each processing activity documented?
2. Will the personal data be processed for a purpose other than what was intended at the time of collection?
3. Do consent-collecting mechanisms require some action (e.g., ticking a box) or affirmative statement by the data subject?
4. If the legal basis for collecting data is consent, is explicit consent obtained?
5. Has a representative within the European Union been designated (for organisations outside the EU)?
6. Do contracts with third parties specify that the third party must have data protection and security protection clauses/annexes in place?
7. Are records kept of all processing activities your company engages in?
8. Are all data transfers documented, including cross-border transfers?
9. Is a Privacy Notice provided to data subjects at every point of data collection?
10. If data is to be processed for a secondary purpose, are data subjects notified of the new purpose prior to processing?
11. Does the Privacy Notice clearly specify how data subjects can exercise their rights under the GDPR?
12. Are internal policies in place defining what is considered to be a data breach and when and if notification to data subjects or Supervisory Authorities is required?
13. Do agreements/contracts with third parties specify that the third party has to notify you (the controller) without undue delay after becoming aware of a data breach or potential data breach involving personal data?
14. Is a log kept of all data breaches that occur, along with the effects and remedial actions taken?
15. Are assessments of processing activities conducted by the relevant personnel to determine the data protection measures that should be in place, proportionate to the risks involved with the processing activity?
16. Is privacy assessed at the beginning stages of development of any processing activity?
17. Are measures such as data minimisation and pseudonymisation implemented across all applicable organisational units?
18. Are Data Protection Impact Assessments (DPIAs) completed for processing activities involving special categories of information, automated decision making, or profiling?
19. Are DPIAs completed prior to implementing new technologies, processes, or projects?
20. Are there processes in place for
    a. responding to a data subject's request for access to information?
    b. rectifying/deleting information about a data subject?
    c. communicating updates of personal data to third parties who have received the data?
    d. allowing a data subject to revoke consent for a particular processing activity at any time?

e. ensuring processing is stopped, including any processing by third parties when consent is revoked?
f. complying with requests to restrict the processing of data if requested by a data subject?
g. complying with requests from a data subject to have their personal data transferred directly to another controller?
h. stopping processing for direct marketing purposes when an objection is received?
i. allowing a data subject to request a manual review of the decision or profiling activity (in the case of automated decision making)?
j. transferring personal data to a third country or international organisation?
k. ensuring the appropriate Supervisory Authority is notified within 72 hours of a confirmed data breach?