MAENAD

Grant Agreement 260057

Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles

Report type — Deliverable D3.1.1

Report name — Language Concepts Supporting Engineering Scenarios

Dissemination level	**PU**
Status	**Final**
Version number	**3.0**
Date of preparation	**2012-08-31**

Authors

Editor	E-mail
Mark-Oliver Reiser	moreiser@cs.tu-berlin.de

Authors	E-mail
Anders Sandberg	anders.sandberg@mecel.se
David Parker	D.J.Parker@hull.ac.uk
DeJiu Chen	chen@md.kth.se
Fulvio Tagliabò	fulvio.tagliabo@crf.it
Henrik Lönn	Henrik.Lonn@volvo.com
Hans Blom	Hans.Blom@volvo.com
Juha-Pekka Tolvanen	jpt@metacase.com
Peter Lindqvist	peter.lindqvist@systemite.se
Renato Librino	renato.librino@4sgroup.it
Sandra Torchiaro	sandra.torchiaro@crf.it

The Consortium

Volvo Technology Corporation (S) Centro Ricerche Fiat (I)

Continental Automotive (D) Delphi/Mecel (S) 4S Group (I)

MetaCase (Fi) Pulse-AR (Fr) Systemite (SE) CEA LIST (F)

Kungliga Tekniska Högskolan (S) Technische Universität Berlin (D) University of Hull (GB)

Revision chart and history log

Version	Date	Reason
0.1	2010-11-26	Initial outline.
		Various contributions by authors of individual chapters.
0.9	2011-02-07	Integration of author contributions (i.e. accepted all changes in Word's "track changes" mode). Some layout fixes.
	mid Feb 11	Review.
1.0	2011-03-02	Intermediate (Final version for 1st delivery at MS3)
1.0.1	2011-08-30	Intermediate (Work-in-progress version at end of Y1) Next delivery due at Dec 1, 2011.
1.1.0	2011-12-20	Intermediate (Final version for 2nd delivery at MS4.5)
	May to Aug 2012	Update of individual chapters by chapter authors.
1.2.0	2012-08-28	Final version.
1.2.1	2012-08-30	Updates in ISO26262 and Analysis chapters.
3.0	2012-08-30	Same as 1.2.1 but correct version numbering for M6 delivery' Integration of update of Timing Analysis chapter. Minor corrections

Table of contents

1 Introduction

Welcome to MAENAD deliverable D3.1.1! This deliverable describes additions and changes to EAST-ADL's modeling elements required in order to support ISO 26262, optimization, analysis, and the other MAENAD objectives. Also, the aim is to give a brief summary and overview of EAST-ADL's support of ISO 26262, optimization, analysis, etc. (depending on the chapter).

The primary purpose of this document, according to the MAENAD Description of Work, has been to document the ongoing conceptual work and help project partners working on a particular topic in planning their future work in MAENAD and help others in the project to catch up with the current status of the topic and join discussions.

It is important to note that this deliverable is not intended to provide a comprehensive introduction to EAST-ADL that is understandable to persons outside MAENAD. Instead, it mainly served the communication within the project and to feed language requirements and change requests to WP4. However, there material from this deliverable is planned to be partly reused in such dissemination and tutorial documents planned in WP7.

Structure

The document is structured based on the cross-work-package work groups identified in MAENAD. Each of these work groups is focused on a particular project objective – as defined in the MAENAD Description of Work – plus an additional work group on language consolidation, which deals with an overall refinement of all parts the language (consistency, etc.). Each cross-work-package group has its own chapter. Consequently, you could say there is a chapter for each MAENAD objective plus one on consolidation. In addition, Chapter 8 goes into detail a selected topic, i.e. fault injection, and Chapter 9 lists all project requirements that are related to modeling concept and provides comments on their coverage.

Scope

This deliverable differs from deliverable D4.1.1 (the EAST-ADL language specification), which is also focused on language concepts, in that we here provide more background, more motivation and document investigations and discussions that were taking place while working out the concepts. Also, some alternatives might be documented that did not make their way into the final language for some reason. In contrast, D4.1.1 will only document the final outcome of the work on the language concepts.

The MAENAD Consortium

2 Modeling Concepts for Supporting ISO 26262

This chapter discusses language refinements related to MAENAD Objective 1, “Develop capabilities for modeling and analysis support, following ISO 26262”. In this section we will describe which ISO 26262 concepts are supported by EAST-ADL at the moment. So we will discuss on how the current support has to be improved and which are the ISO 26262 concepts not yet covered by EAST-ADL.

The ISO 26262 requires that the application of the “functional safety approach”, starts from the preliminary vehicle development phases and continuing along the complete life-cycle of the product. This approach ensures the design of a safe automotive system. Furthermore it provides an automotive specific risk-based approach for determining the risk classes, called ASILs (Automotive Safety Integrity Levels). The standard uses the ASILs for specifying necessary safety requirements on each corresponding item for achieving an acceptable residual risk. ISO 26262 also provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety being achieved.

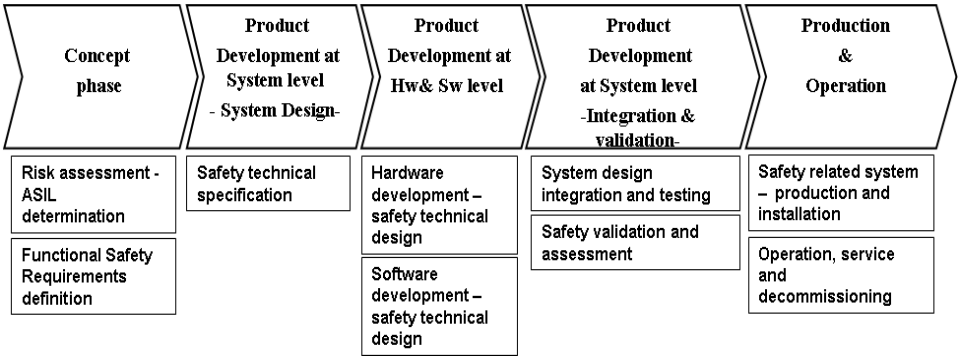


Figure 1. ISO 26262 Safety life-cycle

The ISO 26262 safety life-cycle includes the following phases:

- Concept phase, (Part 3)
- System level development – specification, (Part 4)
- Hardware level development, (Part 5)
- Software level development, (Part 6)
- System level development – integration and validation (Part 4)

2.1 Relation of ISO 26262 to EAST-ADL

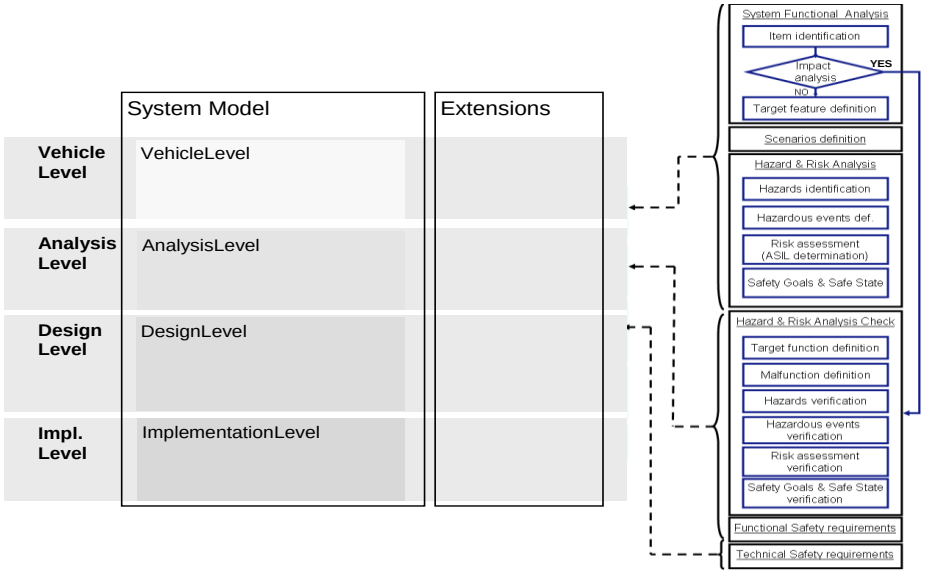
The EAST-ADL supports several of the safety life-cycle phases defined in ISO 26262. EAST-ADL provides support for the safety design flow and related safety design concepts such as item, hazard, and safety concept according to ISO 26262.

This information corresponds to the Dependability extension in EAST-ADL. Following a top-down approach, the safety analysis can start at the Vehicle level, beginning from the item’s “target feature” definition (the feature description in terms of the vehicle’s output(s) behaviour), and the feature flaws definition, as anomalies of the item's outputs on Vehicle Level. Therefore, on Vehicle

level, it is already possible to perform a Hazard analysis and Risk assessment to evaluate the "safety relevance" of the Item under safety analysis. For this purpose, the hazards should be evaluated in different scenarios, assessing Severity, Controllability and Exposure for each hazardous event. The hazard under analysis, when applied to the various operational situations (operative & environmental conditions), results in the so called "hazardous events".

Each Hazardous Event has to be classified in terms of associated risk defined by its Automotive Safety Integrity Level (ASIL). The ASIL level is captured as an attribute in the Safety Goal element and the safety goal itself is defined by a referenced requirement and if applicable a safe state.

To verify the correctness and completeness of the preliminary Hazard analysis and risk assessment performed on VehicleLevel, a complementary analysis can be performed by looking at the architectural level. Therefore the target function (the function description in terms of its output(s) behaviour) on AnalysisLevel should be defined by deriving it from the target feature introduced at the upper abstraction level. At this point it is possible to define the malfunction as anomalies of the item's outputs. These anomalies may be foreseen by the engineer or found by analyses such as Failure Modes and Effects Analysis of the architectural solution.



This serves as a more concrete basis for hazard identification and risk assessment, and therefore offers an opportunity for validation. Note that this process may be iterative and parallel: hazards and risks may be identified and assessed at any abstraction level, but the information is solution independent and Hazards, the Safety Goals and the Safe States are managed as Vehicle level information.

The top-down approach described is intended to be applicable whether or not the item/function is a new development. In the case of a modification of an already existing item an impact analysis is required and a tailored safety lifecycle is advisable. Therefore, with the hypothesis that the safety analysis on VehicleLevel is already available (inherited from original item), the most convenient approach is the bottom-up one, i.e. by entering directly on the AnalysisLevel and by verifying the impact in terms of differences in hazard list and risk assessment outcomes. The VehicleLevel abstracts away all implementation details of a function. This means that even if you have a rough architecture on analysis level to start with, it is easy to present this on VehicleLevel where you express the Item. It is reasonable to assume that the normal functionality is developed in parallel and that makes it equally reasonable to assume that an AnalysisLevel model of the function is present as the safety engineering is performed. This assumption also supports the modelling of the functional requirements needed to find and review the possible feature flaws present in the target function.

On the vehicle level the Item is typically one or more VehicleFeature elements in the Technical Feature tree. To enable a clear definition of what functionality that is associated with an individual VehicleFeature we need not only understand the elements of the EAST-ADL language but also the meaning of relations between Vehicle Feature's in the feature tree. Without this semantic definition it is impossible to make a unambiguous definition of the requirements allocated to a feature and the possible feature flaws that can appear.

For each safety goal resulting from the preliminary hazard analysis, at least one functional safety requirement must be specified. The definition of functional safety requirements is appropriate at the EAST-ADL AnalysisLevel. Note that what is expressed in the ISO 26262 standard as "preliminary architectural assumptions" is exactly the purpose of analysis architecture in the EAST-ADL language. At this level, the goal is to verify that the functional safety concept realizes all safety goals defined at VehicleLevel. More than one safety requirement could be associated with the same Function.

Once the functional safety concept is specified, the item can be developed with a system perspective that includes detailed functional solutions and hardware platform on the EAST-ADL Design Level. This corresponds to the "system design specification" according to ISO 26262. The functional safety requirements are refined to technical safety requirements allocated to the architectural elements on the Design Level.

2.2 ASILs (Automotive Safety Integrity Levels)

Safety Integrity Levels (SILs) are abstract classification levels that can be used to indicate the integrity features of the systems (or elements thereof) obtained with prober safety measures and development processes. SILs have been adopted as part of safety standards such as IEC 61508 and - in the automotive domain - ISO 26262. In the context of the upcoming ISO 26262, SILs are known as ASILs - Automotive Safety Integrity Levels - and form a major part of the standard: ASILs are used to specify the necessary safety requirements for achieving an acceptable residual risk, as well as providing requirements for validation and confirmation to ensure the required levels of safety are being achieved.

Safety requirements in these standards are intended to ensure the system being designed is free from unacceptable risk (assuming the requirements are met) and are derived through a process of analysis and risk assessment. The aim of the process is to determine the critical system functions - those which have the potential to be hazardous in the instance of failure - and the requirements necessary to mitigate the effects or reduce the likelihood of those hazards. These safety requirements are often associated with integrity requirements that apply to those critical functions to indicate, in essence, what level of contribution they have towards the overall system safety and thus what level of safety they should implement to avoid system failures. A low ASIL therefore indicates that the element is not a major contributor to severe system failures, while a high ASIL indicates that it is potentially is a major contributor, and this allows a means of verifying that system safety requirements are being achieved by ensuring that the ASILs allocated to system elements are also being met.

Therefore ASILs play a dual role in the development of safety-critical systems: they allow for top-down allocation of safety requirements to different elements of the system according to their contribution to risk, and they allow for bottom-up verification to show that the safety requirements are being met by the developed system.

ASILs are divided into one of four classes, see Table 1 below, to specify the item's necessary safety requirement for achieving an acceptable residual risk, with D representing the highest and A the lowest class. QM (Quality Management) can be applied to non-safety critical elements to indicate that there are no specific safety requirements in place. The ASIL-Level shall be determined for each hazardous event using the estimation parameters severity (S), probability of exposure (E) and controllability (C)

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Table 1 - Determining ASILs

QM (Quality Management) → the function has no impact on safety - it is not necessary to define any safety requirement

Top-level safety requirements

During the concept phase a *safety goal* shall be defined for each hazardous event. This is a fundamental task, since the safety goal is the top level safety requirement, and it will be the base from which the functional and technical safety requirements are defined. The safety goal leads to item characteristics needed to avert the hazard or to reduce the risk associated with the hazard to an acceptable level. Each safety goal is assigned an ASIL value to indicate the required integrity level according to which the goal shall be fulfilled. For every safety goal, if applicable, a *Safe state* shall be identified in order to declare a system state to be reached or maintained when the failure is detected, to allow a failure mitigation action without any violation of the associated safety goal. For each safety goal that are the results of the risk assessment, at least one safety requirement shall be specified.

2.3 SEooC Concept

The automotive industry develops generic elements for different applications and for different customers. These generic elements can be developed in respect to the functional safety approach as Safety Elements out of Context SEooC (ref. ISO 26262 - Road vehicles — Functional safety — Part 2: Management of functional safety - Clause 6.4.5.6 and Part 10: Guideline - Clause 9).

The SEooC is a generic element(s) developed independently by different organizations. It is a safety-related element not developed in the context of a specific vehicle.

To develop a SEooC it is necessary to define a set of assumptions to which the SEooC aims.

The assumptions can be categorized into two main categories: the External requirements, related to the reference vehicle target (e.g. E/E architecture, system(s), environment…) and the Internal ones, related to the application, that are placed on the element by higher levels of design. The assumptions allow the correct integration of the SEooC into a specific Item; this is allowed by checking the consistency of the assumptions in respect with the specific interfaces of the item.

If the SEooC assumptions do not fulfil the item requirements, it is necessary to apply a change(s) to the SEooC or a change(s) to the Item.

The assumptions give consistency to the application of the ISO26262 on developing of the generic element(s) during the item integration phase.

Each SEooC can be developed at many Safety Life Cycle levels, depending on the functionalities and types. In other terms it's possible to :

- develop a System as a Safety Element out of Context (e.g. Stop & Start system)
- develop a Hardware component as a Safety Element out of Context (e.g. microcontroller)
- develop a Software component as a Safety Element out of Context (e.g. AUTOSAR basic software modules).

A structured modelling performed in EAST-ADL can support the SEooC application.

The SEooC assumptions can be captured in EAST-ADL dependability model. During the integration, the assumptions are matched vs. Item requirements (Safety goal, Functional Safety Concept, Technical Safety Concept depending on SEooC abstraction level).

Morerover, EAST-ADL model reflects the character of the SEooC development at many Safety Life Cycle levels.

When a SEooC is developed, no step of the safety lifecycle can be omitted; the EAST-ADL methodology steps and the existence of the required work products secure this constraint.

2.3.1 Language support for function definition

As stated in section 2.2 it is essential to understand the function definition for the Item on vehicle level. The EAST-ADL language has all the language elements to support an unambiguous function definition. It lacks however a clear definition of what the links between elements on vehicle level mean when looking at requirements allocated to individual nodes. What is needed is an explicit definition on how the FeatureLink relation in the vehicle level feature tree extends the validity of the requirements associated with the features.

For a single VehicleFeature the meaning is simple, the function definition is defined by the requirements directly associated with the VehicleFeature.

A normal feature tree on the vehicle level consists of a tree structure with VehicleFeature elements linked with FeatureLink associations. This structure makes the Item's function definition unclear as no semantic meaning on how requirements on either side of a FeatureLink is valid is present in the language. For ISO26262 support this semantic meaning is essential as it is the only association linking Vehicle features.

The proposal is to say that the following semantic definition for 'Requirement' elements inheritance:

1. A VehicleFeature element inherits all requirements directly associated with all its parents VehicleFeature elements.

2. The parents are defined as the ‘start’ element of a FeatureLink where the target VehicleFeature is the ‘end’ element of the same FeatureLink.

In the figure below Feature A has no parents. Feature B has Feature A as its parent and Feature C and D both share Feature A and B as parents.

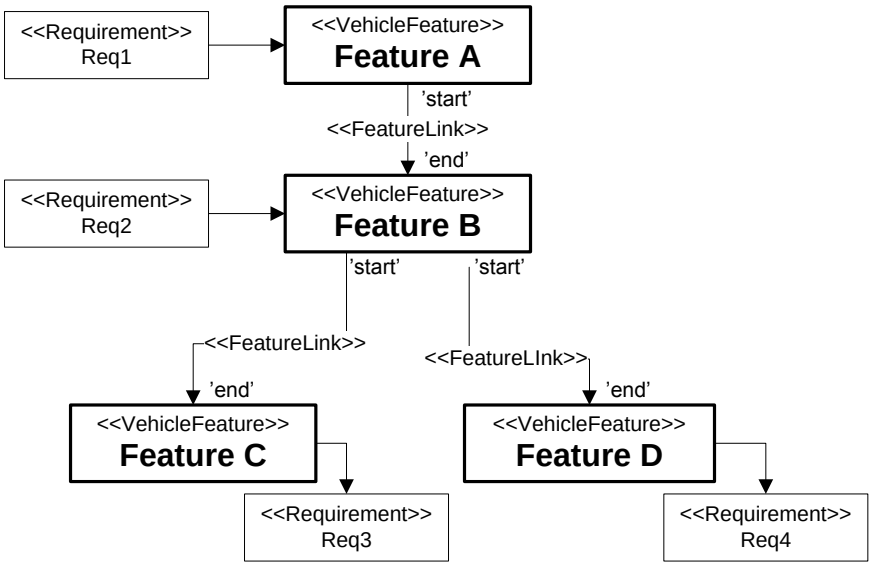


Figure 2. Vehicle feature tree example

This leads to the following functional definition:

- Feature A is defined by requirement Req1.
- Feature B is defined by requirements Req1 and Req2.
- Feature C is defined by requriements Req1, Req2 and Req3.
- Feature D is defined by requriements Req1, Req2, and Req4.

This semantic meaning is only valid for Requirement elements. But it has the strength in that it makes for a clear, unambiguous definition of which requirements that are valid for every Vehicle Feature in the vehicle level model. It is not reasonable to extend the semantic meaning in general. For requirements it has been discussed and makes sense.

It is also important to state that the inheritance only applies to the vehicle level. For several reasons there is no inheritance capabilities built into the language in lower abstraction levels and hence no semantic rules are possible to define. To state the requirements on analysis level for a VehicleFeature you must follow the links from the set of requirements on vehicle level down to the analysis level and state all the refined requirements.

The FeatureLink semantic limits how the vehicle level can be modeled, at least when it comes to how requirements are linked to VehicleFeature elements. Since inheritance is only from ‘end’ to ‘start’, common requirements at a ‘start’ need to be kept in the VehicleFeature element pointed to.

This semantic definition has been disseminated in the ATESST2 project at SAFECOMP2010.

This paper also states a follow-on definition of the links between <<Realization>> links between VehicleFeatures and AnalysisLevel elements. If functionality is associated with all parent element requirements, realization is top down. This becomes obvious when you look at the top node in the feature tree. This node likely has no realization links to the analysis architecture, it is implemented by the union of all realization links by all its ‘end’ elements. This holds for each sub-tree in the feature tree.

2.3.1.1 Next steps in support for function definition.

Since the ATESST2 work in function definition was limited to requirements only these elements were review with respect to inheritance on vehicle level. With the development of the language and the behavior part the new elements that can be used to model behavior on vehicle level should be reviewed in the same way as requirement to find the semantic meaning of the links.

2.3.2 Language Support for ASILs

ASIL decomposition and allocation is an important objective for MAENAD and a major requirement in order to be able to fully support ISO 26262-compatible safety-driven design. D3.2.1 details a newly developed algorithm that enables the automatic decomposition and allocation of ASILs across independent elements of the system by building upon earlier work on FTA; ASILs assigned to hazards can then be decomposed to the minimal cut sets that cause those hazards. By enumerating the different permutations of those ASILs assigned to multi-event cut sets in a recursive process, it is possible to determine all possible valid ASIL allocations for the basic events of a system while ensuring that the resulting allocations are still capable of meeting the original safety requirements. EAST-ADL language support is relatively mature and language elements for both hazard analysis and ASILs are present. However, it may be that these need tweaking or extending to streamline the process in response to practical experience gained during the project.

At present, much of the infrastructure required to support ASIL decomposition and analysis is already present in both EAST-ADL and HiP-HOPS. In particular, EAST-ADL supports:

- Hazard analysis and definition of Hazards, HazardousEvents, and SafetyGoals. HazardousEvents and SafetyGoals can both store ASIL values.
- SafetyConstraints can be used to assign ASILs to elements of the error model.
- SafetyConstraints may also provide a mechanism for linking the resulting ASIL allocations back to the faults & failures of the error model after decomposition.

2.3.2.1 Next steps to be made in Language support for ASILs

EAST-ADL therefore has sufficient language support to enable ASIL decomposition and the next step is to ensure nothing is missing by developing the algorithms and performing tests on case studies. The main obstacle at present is the link between the two: namely, the Papyrus plugin, which needs extending to enable the output of hazards and ASIL information to HiP-HOPS and allow the starting of the decomposition process. This will enable us to begin testing the decomposition of actual EAST-ADL models with the algorithm, which will highlight bugs to be fixed and other areas for further work (e.g. any streamlining or clarification of the existing language elements, any additions necessary to the language, what sort of bugs exist in the tools, and how efficient and scalable the algorithm is).

A secondary issue is that of storing the results back in the model. This is a general unsolved issue not specific to ASILs (e.g. many other analysis results are currently completely external as well) but it is something to be investigated further.

2.4 Modeling concepts for ISO26262 – gaps analysis

In the following table a description of what language concepts (i.e. modeling elements, attributes, associations, etc.) are already present in EAST-ADL and what are required to cover ISO 26262 has been provided.

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
Part 3 - Clause 5	**Item definition**		
Part 3 -Clause 5.4.1	Description of the item's purpose and functionality, including operating modes and states	Item references Features Realizing Artifacts and define SystemBoundaries. Features describe purposes and functionality, including operating modes and states on user level	Semantic definitions on links on vehicle level are needed.
Part 3 -Clause 5.4.1	Description of the interactions with other items or elements	Features (its use cases, requirements and refined requirements) describe interactions with other items or elements on user level. Realizing Artifacts describe interactions with other items or elements of solution	Semantic definitions on links on vehicle level are needed.
Part 3 -Clause 5.4.1	Applicable laws and regulations, national and international standards	Features (its requirements) define Applicable laws and regulations, both national and international standards.	Semantic definitions on links on vehicle level are needed.
Part 3 -Clause 5.4.1	The operating scenarios which impact the functionality of the item. Expected or required environmental conditions	OperatingScenario on Item describes operating scenarios which impact the functionality of the item. Requirements on Features define expected or required environmental conditions that	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
		are independent of solution, Requirements on Artifacts define expected or required environmental conditions that are dependent of solution	
	Known failures and hazards	ErrorModels linked to artifacts identify known failures Hazards identify known Hazards	No additional requirements
Part 3 -Clause 5.4.1	Behavior achieved by similar functions, items or elements, if any. Pre-trials information	Requirements on Feature can be compared with other feature's requirements. For more detailed aspects, behaviour achieved by similar functions, items or elements can be investigated, if any are defined on the respective element.	No additional requirements
Part 3 – Clause 6.4.1.1	Determine if the Item is a new development, or if it is a modification of an existing item or its environment.	Feature models can be used to characterize the new item. Use V&V concepts to refer to "Proven in use"	Add an 0..1 attribute to "Item"
Part 3 – Clause 6.4.2.1	Impact Analysis: - Definition and description of the intended modifications, in terms of design modifications and/or implementation modifications - Identification of the areas affected by the intended modifications - Implications of the intended modifications with regard to functional safety - Identification and description of the affected work products	Feature models can be used to characterize the new item. Realize links define how the architecture relate to the item. Tool support can assist the engineer to identify the architecture elements related to the item under safety assessment represent. The impact of those element on the rest of the architecture or their significance in their own rights can be assessed.	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
Part 3 – Clause 7	**Hazard Analysis and Risk Assessment**		
Part 3 – Clause 7.4.2.1.1	The operational situations and operating modes in which an item's malfunctioning behaviour will result in a hazardous event shall be described, both for cases when the vehicle is correctly used and when it is incorrectly used in a foreseeable way.	Operating Mode; Operational Situation – traffic, environment; Operational Situation – Use Case.	No additional requirements
Part 3 – Clause 7.4.2.2.1	- The hazards shall be determined systematically by using adequate techniques; - Hazards shall be defined in terms of the conditions or behavior that can be observed at the vehicle level.	FeatureFlaw, Hazard	No additional requirements
Part 3 – Clause 7.4.2.2.3	The hazardous events shall be determined for relevant combinations of operational situations and hazards.	HazardousEvent metaclass	No additional requirements
Part 3 – Clause 7.4.3	- All hazardous events identified shall be classified, except those that are outside the scope of ISO 26262; - The severity of potential harm shall be estimated based on a defined rationale for each hazardous event; - The probability of exposure of each operational situation shall be estimated based on a defined rationale for each hazardous event; - The controllability of each hazardous event, by the driver or other traffic participants, shall be estimated based on a defined rationale for each hazardous event	SeverityClassKind, ControllabilityClassKind, ExposureClassKind (Enumeration Metaclass) Rationale element can be used to justify the S/E/C parameters and relate them to analyses and assessments underlying the selected value.	No additional requirements
Part 3 – Clause 7.4.4.1	An ASIL shall be determined for each hazardous event using the parameters "severity", "probability of exposure" and "controllability"	ASIL parameter, typed by ASILClassKind can be used.	Undefined is missing from the current enumeration to allow differentiation between e.g. QM and concious decisions.
Part 3 – Clause 7.4.4.3	Safety goal shall be determined for each hazardous event. Possible grouping of safety goals	SafetyGoal (EAElement)	No additional requirements
	Safety goals should be possible to combine. For example several safety goals could be assessed and replaced by a common safety goal that match all of them.	Use Derive relation between requirements to trace the derived safety goal's requirements.	
Part 3 – Clause 7.4.4.5	Safe state shall be defined, if possible	For every Safety Goal, a safe state should be defined as attribute of SafetyGoal (safeStates : String [0..1]).	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
		The safe state should be a state or a reference to a state.	
Part 3 – Clause 8	**Functional Safety Concept**		
Part 3 – Clause 8.4.2.3, 8.4.2.4, 8.4.2.5, 8.4.2.6	Safety requirements, including: fault tolerant time interval (FTTI), warning and degradation concept, driver's actions Functional architecture, including: functional redundancies, safe states, emergency operation, driver actions, external measures (if any), ASILs Preliminary physical architecture, in which functionality is allocated	Requirements in a FunctionalSafetyConcept related with Satisfy links to FAA without redundancy or safety measures OR Requirements in a FunctionalSafetyConcept related with Satisfy links to FAA with redundancy and safety measures (In case safety solutions are modelled as a modified FAA, this structure may also linked with a refine relation to a Functional Safety Requirement. The original FAA stays non-redundant in that case) ("Preliminary physical architecture, in which functionality is allocated" appears to be too early. ISO26262 does not mention preliminary hardware, it only mentions architectural elements which can stay purely functional in concept phase) Regular elements such as modes, functions, etc. can be used. The functional safety concept. The required concepts should be requirements with roles in the Functional/Technical safety concept.	External measure concept is to be added. Role names for driver actions, emergency operation, external measure, fault tolerant time interval, etc. are to be added.

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
		Timing-related requirements can be formalized using timing concepts, some events needs to be added. Acceptance criteria can be recorded in VVIntendedOutcome	
Part 4 – Clause 6	**Technical Safety Requirements**		
Part 4 – Clause 6.4.1.1	Interfaces including communication and HMI (if applicable)	FDA for Interfaces including communication and HMI (if applicable)	No additional requirements
Part 4 – Clause 6.4.1.1	Environmental and functional constraints	EnvironmentModel; FDA for functional constraints	No additional requirements
Part 4 – Clause 6.4.1.1	Configuration requirements	Requirements on FDA, Requirements on variability mechanisms	No additional requirements
Part 4 – Clause 6.4.1.1	Response to stimuli	FDA behavior or Requirements on FDA	No additional requirements
Part 4 – Clause 6.4.2.3	Safety mechanisms (fault detection and control): - detection, indication and control of faults of the item - detection, indication and control of faults in external devices that interact with the system - measures that enable the system to achieve or maintain a safe state - measures to detail and implement the warning and degradation concept - measures to detail and implement the warning and degradation concept	Requirements on FDA and HDA elements represent Safety mechanisms (fault detection and control); RequirementContainers with appropriate names can be used to identify requirements related to safety mechanisms, warning, degradation, etc.	No additional requirements
Part 4 – Clause 6.4.2.3	For each safety mechanism that enables an item to achieve or maintain a safe state the following shall be specified: the transition to the safe state, including the requirements to control the actuators	FTTI and other timing measures can be modelled as timing constraints. Explicit roles can be defined on the technical safety	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
	- the fault tolerant time interval - the emergency operation interval, if the safe state cannot be reached immediately - the measures to maintain the safe state.	concept.	
Part 4, 6.4.2.2	Degraded modes, limp home strategy, fault mitigation mechanisms, driver warning	Regular elements such as modes, functions, etc. can be used. The functional safety concept and technical safety concept are used	No additional requirements
Part 4 – Clause 6.4.4	Measures which prevent faults from being latent shall be defined	Measures which prevent faults from being latent can be represented using regular constructs such as functions and requirements. Their role as latent fault prevention mechanism can be identified using a requirment with the role “latent fault prevention”	No additional requirements
Part 4 – Clause 7.4.1; 7.4.5	**Technical Safety Concept**		
Part 4 – Clause 7.4.1.1	The system design shall be based on the functional concept, the preliminary architectural assumptions and the technical safety requirements	Requirements on FDA, refined by SafetyConstraints represent Safety requirements	No additional requirements
Part 4 – Clause 7.4.1.5	The technical safety requirements shall be allocated to hardware and software elements (ASIL Allocation)	SafetyConstraint represent ASIL allocation	No additional requirements
Part 4 – Clause 7.4.1-7.4.4	**System design specification**		
Part 4 – Clause 7.4.4	'- Measures for control of random hardware failures: - Specifications of the measures to detect, control or mitigate the random failures - Target values for metrics - Evaluation procedures of violation of the safety goals - Diagnostics and coverage targets at element level	FDA and HDA represent: - Measures for control of random hardware failures. Methodology issue: Instruction for Engineers to instruct that "Target values for metrics", etc.	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
		are covered; - Requirements and related VVCase define how to "evaluate procedures of violation of the safety goals"; RequirementContainers with appropriate names can be used to identify requirements related to control of random hardware failures. Methodology issue: Engineers should add Requirements/steps that reduce the failure rate of the solution.	
Part 4 – Clause 7.4.3	Measures to eliminate or to mitigate the effects of internal and external systematic failures	Requirements define the "Diagnostics and coverage targets at element level" FDA and HDA requirements specify some "Measures to eliminate or to mitigate the effects of internal and external systematic failures" RequirementContainers with appropriate names can be used to identify requirements related to systematic failure control.	No additional requirements
Part 4 – Clause 7.4.6	Hardware software interface specifications: - the relevant operating modes of hardware devices and the relevant configuration parameters	- HWFunction, BSWFunction and LocalDeviceManager specify "Hardware software interface"; - A ModeGroup can be owned by HW element to define "relevant operating modes of hardware devices" while variability mechanisms may define "relevant configuration parameters" RequirementContainers with appropriate names can be used	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
		to identify the HSI requirements.	
Part 4 – Clause 7.4.6	Hardware software interface specifications: - the hardware features that ensure the independence between elements and that support software partitioning	Requirements on HDA define "hardware features that ensure the independence between elements and that support software partitioning"	No additional requirements
Part 4 – Clause 7.4.6	Hardware software interface specifications: - shared and exclusive use of hardware resources - the access mechanism to hardware devices	FDA and requirements on FDA define "shared and exclusive use of hardware resources" and "the access mechanism to hardware devices"	No additional requirements.
Part 4 – Clause 7.4.6	Hardware software interface specifications: - the timing constraints defined for each service involved in the technical safety concept	Resources for software (memory, I/O, etc) are treated on Implementation level. Timing constraints on FDA are used to represent "Timing constraints defined for each service involved in the technical safety concept"	No additional requirements
Part 4 – Clause 7.4.6	Hardware software interface specifications: - the hardware diagnostic features - the diagnostic features concerning the hardware, to be implemented in software	FDA and requirements on FDA and HDA specify "the hardware diagnostic features " and "the diagnostic features concerning the hardware, to be implemented in software" RequirementContainers with appropriate names can be used to identify the HSI requirements.	No additional requirements
Part 4 – Clause 7.4.7	Specification of requirements for production, operation, service and decommissioning: - Assembly instructions requirements - Safety-related special characteristics - Requirements dedicated to ensure proper identification of systems or elements -Verification methods and measure for production - Service requirements including diagnostic data and service	Requirements for production, operation, service and decommissioning can be represented using the regular requirements constructs. Due to the wide range of requirements perceivable RequirementContainers with	No additional requirements

ISO26262 ref.	Requirement of the standard	Requirement to system description and modeling already covered in EAST-ADL	Requirement to system description and modeling to be added in EAST-ADL
	notes - Decommissioning requirements	appropriate names can be used rather than pre-defined requirement categories.	

2.5 System/Environment model interface implications for ISO26262 support

One of the issues with modeling the system – environment interface and their relation to ISO26262 is the open meaning of ports for functional devices. To enable a clear distinction of what is the target of a hazard when analyzing a model some changes are suggested. The background for the changes are given in section 2.5.1 and the proposal for new structure of Functional Devices are given in 2.5.2

2.5.1 Functional devices in current language definition

From a modeling point of view there is no difference between a FunctionalDevice and an AnalysisFunctionType. There are nothing special about it other than the semantics. A lot of this discussion relates to the discussion on the environment model in section 6 and there is some overlap on the issues that relate to functional devices.

The issues that have been detected are the following.

1. Can a FunctionalDevice have an error-behavior or is it just a mapping between physical and logical world.
    a. If a functional device can have logical errors, is it then more than a functional device.
2. Can a FunctionalDevice be monitored by a safety mechanism. (The output is in the physical world.)
    a. It is a modeling challenge to get the right level of detail if functional devices as they can be complex or simple. They are often complete systems that could contain a full system model.
    b. A safety mechanism monitoring the logical output of an actuator is difficult to envision with ISO terminology if there is no error in the functional device and the output to the physical world is a direct mapping of the input.

Some of these views are conceptual discussions based on how to handle sensors and actuators from a safety point of view.

The semantics in the language spec gives some hints on the scope of a functional device: The behavior associated with the FunctionalDevice is the transfer function between the environment model representing the environment and an AnalysisFunction. The transfer function represents the sensor or actuator and its interfacing hardware and software (connectors, electronics, in/out interface, driver software, and application software).

Does this mean that the logical port of a functional device should encapsulate the unknown implementation of the transfer, or that the transfer function becomes a constraint on how the lower abstraction levels manage the transfer from logical level to environment? It is not sure that the second part of the semantics is necessarily known at the time the functional device is defined and it seems strange that the analysis level even discusses artifacts that are mapped to hardware and software. The first sentence is the basic notion of a functional device on analysis level.

There is nothing that really prevents the current language to address the above mentioned problems but the semantics is too open for comfort. This is especially true for the semantics on the port definitions. Hence after the extract of the relevant parts of the language a proposal for change is made that clarifies the specialization of a functional device from its ancestors.

Extract from the domain model:

FunctionType (from FunctionModeling) {abstract} «atpType»

Generalizations

Context (from Elements)

Description

The abstract metaclass FunctionType abstracts the function component types that are used to model the functional structure, which is distinguished from the implementation of component types using AUTOSAR. The syntax of FunctionTypes is inspired from the concept of Block from SysML.

FunctionBehavior and FunctionTrigger in the Behavior package are associated to a FunctionType.

Attributes

isElementary : Boolean [1]

True, when this type must not have any parts.

Associations

port : FunctionPort [*] (from FunctionModeling)

Owned ports.

connector : FunctionConnector [*] (from FunctionModeling)

The connectors that connect ports of parts as assembly connectors or ports of this type and ports of parts as delegation connectors.

portGroup : PortGroup [*] (from FunctionModeling)

Grouping of ports owned by this element.

Constraints

No additional constraints

Semantics

The FunctionType abstracts the function component types that are used to model the functional structure on AnalysisLevel and DesignLevel.

Leaf functions of an EAST-ADL function hierarchy are called elementary Functions.

Elementary Functions have synchronous execution semantics:

1. Read inputs

2. Execute (duration: Execution time)

3. Write outputs

Execution is defined by a behavior that acts as a transfer function.

Subclasses of the abstract class FunctionType add their own semantics.

If a behavior is attached to the FunctionType, the execution semantic for a discrete elementary FunctionType complies with the run-to-completion semantic. This has the following implications:

1. Input that arrives at the input FunctionPorts after execution begins will be ignored until the next execution cycle.

2. If more than one input value arrives per FunctionPort before execution begins, the last value will override all previous ones in the public part of the input FunctionPort (single element buffers for input).

3. The local part of a FunctionPort does not change its value during execution of the behavior.

4. During an execution cycle, only one output value can be sent per FunctionPort. If consecutive output values are produced on the same FunctionPort during a single execution cycle, the last value will override all previous ones on the output FunctionPort (single element buffers for output).

5. Output will not be available at an output FunctionPort before execution ends.

6. Elementary FunctionTypes may not produce any side effects (i.e., all data passes the FunctionPorts).

AnalysisFunctionType (from FunctionModeling)

Generalizations

FunctionType (from FunctionModeling)

Description

The AnalysisFunctionType is a concrete FunctionType and therefore inherits the elementary function properties from the abstract metaclass FunctionType. The AnalysisFunctionType is used to model the functional structure on AnalysisLevel. The syntax of AnalysisFunctionTypes is inspired from the type-prototype pattern used by AUTOSAR.

The AnalysisFunctions may interact with other AnalysisFunctions (i.e., also FunctionalDevices) through their FunctionPorts.

Furthermore, an AnalysisFunction may be decomposed into (sub-)AnalysisFunctions. This allows the functionalities provided by the parent AnalysisFunction to be broken up hierarchically into subfunctionalities.

A FunctionBehavior may be associated with each AnalysisFunction. In the case where the AnalysisFunction is decomposed, the behavior is a specification for the composed behavior of the subAnalysisFunction. If the AnalysisFunction is not decomposed (i.e., if the AnalysisFunction is elementary), then the behavior is describing the behavior of the subAnalysisFunction, which is to be used when building the global behavior of the FunctionalAnalysisArchitecture by composition of the leaf behaviors.

Attributes

No additional attributes

Associations

part : AnalysisFunctionPrototype [*] (from FunctionModeling)

The parts contained in this AnalysisFunctionType.

Constraints

No additional constraints

Semantics

The AnalysisFunctionType represents a node in a tree structure corresponding to the functional decomposition of a top level AnalysisFunction. The AnalysisFunction represents the analysis function used to describe the functionalities provided by a vehicle on the AnalysisLevel. At the AnalysisLevel, AnalysisFunctions are defined and structured according to the functional requirements, i.e., the functionalities provided to the user.

FunctionalDevice (from FunctionModeling)

Generalizations

AnalysisFunctionType (from FunctionModeling)

Description

The FunctionalDevice represents an abstract sensor or actuator that encapsulates sensor/actuator dynamics and the interfacing software. The FunctionalDevice is the interface between the electronic architecture and the environment (connected by ClampConnectors). As such, it is a transfer function between the AnalysisFunction and the physical entity that it measures or actuates.

A Realization dependency can be used for traceability between LocalDeviceManagers and Sensors/Actuators that are represented by the FunctionalDevice.

Attributes

No additional attributes

Associations

No additional Associations

Constraints

No additional constraints

Semantics

The behavior associated with the FunctionalDevice is the transfer function between the environment model representing the environment and an AnalysisFunction. The transfer function represents the sensor or actuator and its interfacing hardware and software (connectors, electronics, in/out interface, driver software, and application software).

2.5.2 Suggested changes to FunctionalDevice.

Since Hazards occur when a failure is propagated to the output of an actuator in a specific situation, they all originate through the *output* of a FunctionalDevice acting as an actuator. But an instance of a FunctionalDevice can be both sensor and actuator and there is no distinction that makes it possible to find locations in a model where propagated failures can cause hazards. Hence it might make more sense to use the roles ‘logical’ and ‘physical’ for the ports on a functional device to state where it is connected. The direction of the physical port could then indicate whether it is a sensor or actuator when an analysis is performed. The difference between a functional device and an analysis function could be seen as the fact that the functional device has a connection to the physical world, something that an analysis function cannot have.

The suggested change to the FunctionalDevice class is a new port. The connection to the physical world, the place where hazards occur or where sensors are connected:

Associations

physicalPort : FunctionPort [*] (from FunctionModeling)

This could be both a FlowPort or PowerPort depending on the needs.

This port serves two purposes in the analysis of models. It gives information on the type of functional device given the direction of the port. Secondly it serves as the connection point between Safety goals and the logical architecture, through the error model.

The current port given by the FunctionType attribute ‘port’ would then be limited to being logical ports not allowed to be connected to the environment. Depending on the type of functional device input ports would be stimuli from the logical world and outputs could be nominal values or logical feedback. This enables the possibility to make functional devices hierarchical as the internal structure could feed not only the physical port with data but also the logical ports which makes it perfectly plausible to use AnalysisFunctionTypes in the decomposition of a FunctionalDevice.

Having the capability to do logical feedback would make it possible to address safety mechanisms as there would be a logical path that could be specialized when decomposing the functional device in a more detailed view.

Functional devices can be seen as either very complex system, especially if you are focusing on them in your modeling. Or as trivial data producers if you are interested in the logic manipulating of data.

3 Modeling Concepts for Supporting the Analysis of Behavior-Centric Properties

The reasoning and analysis of dependability & performance involve many aspects in a system's lifecycle. In system development, this requires not only information about the system's topologies, but also an understanding of system behaviors in reacting environmental stimuli and in managing the deployment of internal communication and computation resources. While providing necessary modeling support for capturing important performance and dependability constraints (e.g. end-to-end timing, reliability and safety constraints), current EAST-ADL provides only a rather limited support for capturing the behaviors underlying the generations of such analytical models.

This chapter presents the proposals that have been developed in MAENAD to enhance EAST-ADL for allowing advanced analysis of dependability & performance. An overview of potential EAST-ADL support for analysis and a review of the previous EAST-ADL behavior annex proposal can be found in D3.2.1. In this chapter, we focus on the recent advances towards a final language upgrade. The proposed EAST-ADL enhancement on native behavior modeling can bring in many important benefits. Besides the decisions underlying the assignments of time budgets and error behaviors, such an enhancement will also improve the EAST-ADL support for safety requirements, function/component contracts, fault injection, and test case generation. It will also constitute a necessary step towards the integrations of external mature formalisms and tools for advance prediction of dependability& performance. See Figure 1 for an illustration of the key factors that have affected the language enhancement proposal.

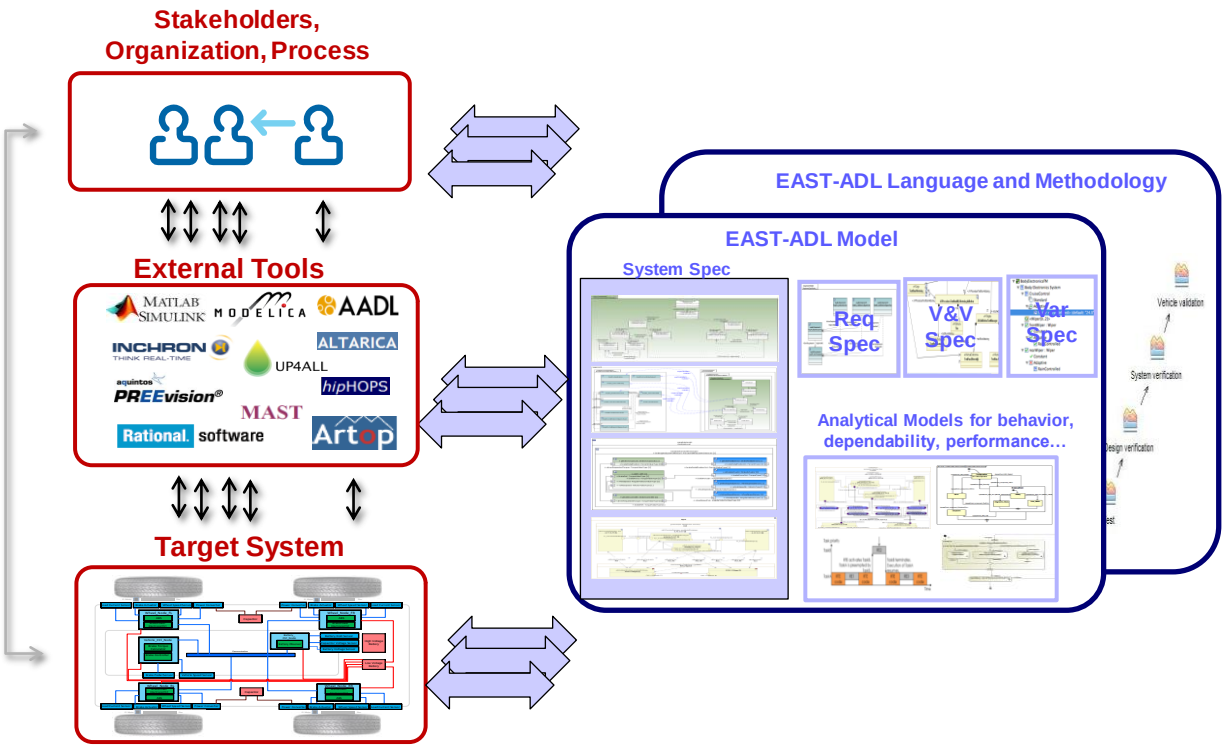


Figure 1. The scope of EAST-ADL enhancement and related contextual factors.

3.1 Background

In FEV (Fully Electrical Vehicles), embedded systems play important roles in regard to advanced control and mode management and have stringent dependability and performance constraints. A specification of the expected EAST-ADL language support, together with the related FEV specific engineering scenarios, can be found in the MENAD deliverable D2.1.1. It is concluded that an enhanced language support for behavior specification is necessary for many reasons, such as

unambiguous interpretation of requirements and early quality predictions. In particular, the following categories of language features are considered important for the engineering of FEV:

- To support precise definitions of temporal characteristics for the definition and analysis of safety constraints (4SG#0050, 4SG#0057, 4SG#0058, 4SG#0059)
- To support the assessment of completeness and correctness of the safety requirements (4SG#0048)
- To support the descriptions of driving profiles (CON#2001), physical dynamics (CRF#0006b, CRF#0007b), power management procedures (CRF#0010b, CRF#0011b, CRF#0013b, CRF#0014b, CRF#0015b), fault tolerance design (CRF#0017b, CRF#0018b)
- To support the generation and precise definition of test cases (4SG#0049a, 4SG#0050)
- To support the integration with external formalisms (CON#0017, CON#0018, CON#0019)

In regard to system behaviors, current EAST-ADL provides language support for specifying the executions of system functions, together with related allocations, triggering policies, and timing constraints. The specification of actual behaviors of system functions relies on external tools (e.g., Simulink/Matlab). This means that behavior models, simulation, analysis, and code generation for the final software synthesis are all maintained and carried out based on external tools. This kind of black-box approach to behavior specification is considered sufficient for implementation design, such as in regard to multitasking and final software configuration. See Figure 2 for an overview of related modeling constructs.

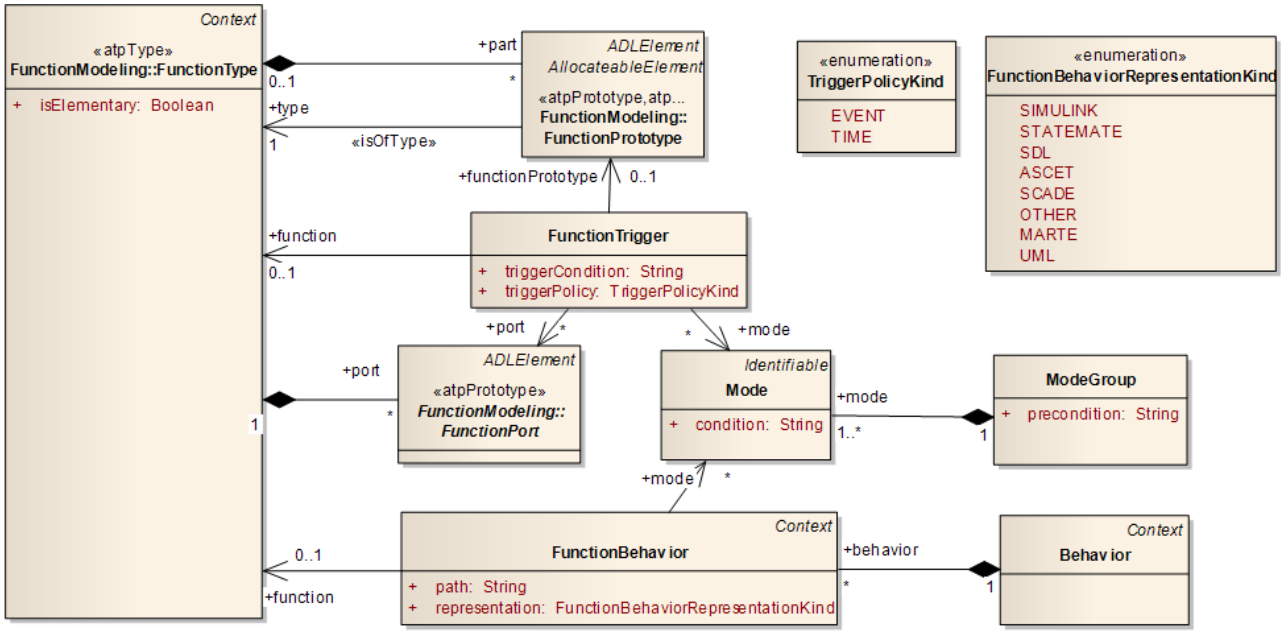


Figure 2. An overview of the meta-model definitions in current EAST-ADL2 Behavior Modeling Package.

From a system design point of view, the behavioral issues that can be of particular concern include not only the execution scheme (e.g., time- or event-triggered execution of system functions), but also the system's operational situations, the dynamics of plant under control, the nominal and erroneous behaviors of functions and components (e.g., their internal state transitions), as well as the compositions of various behaviors and related mode assignments. A precise specification of such issues is fundamental for many overall design decisions, including requirements definitions and refinements, function structuring, the synthesis of analytical models and test cases, and safety engineering, etc. It is seldom the case that a single analysis tool would cover all these. Even if the actual behaviors of system functions are captured in external tools, there is still a need of explicitly

annotating related bounds (e.g., invariants of data, internal states and state transitions) and permitting the traceability of behavior concerns at different levels of abstraction, such as for the control of consistency and completeness.

To facilitate the predication of dependability and performance, it is expected that EAST-ADL as a system architecture description language would constitute the basis for consolidating various kinds of behavior information. Such behavior information can for example be associated to requirements, architectural and analytical models, or V&V cases.

Current EAST-ADL supports the annotations of error behaviors of system functions and components through error models. The aim is to provide analytical information for fault-tree analysis and safety constraint assignment. The specifications of error logics are directly based on external formalisms, such as expressions in Boolean logic. There is still a lack of support for consolidating the error logics in different external formalisms. Moreover, an enhancement of the language in regard to the temporal aspects of anomalies is necessary for allowing advanced safety analysis (e.g., model-checking and fault injection). The aim is to support precise definitions of faulty conditions in both value- and time- domain, the transitions across nominal and erroneous states, and thereby the reasoning of emergent properties due to compositions.

3.2 EAST-ADL Enhancement Proposals

This proposal further refines the EAST-ADL behavior annex proposed in the ATESST2 project. The aim of the behavior annex is to allow a more precise specification of behavioral constraints, which are implied by requirements and satisfied by functions, hardware and environmental components, etc. See the MENAD deliverable D3.2.1 for an introduction of the proposed behavior annex. Major improvements in MAEAD include:

- A harmonization with the syntax and semantics of current EAST-ADL support for specifications of architectural structures, execution behaviors (e.g., Triggers, FunctionEvents) of functions and components, and execution specific timing constraints;
- A consolidation of proposed behavior constraints in regard to their definitions and relations.
- Support for type-prototype pattern, allowing the instantiating of behavior types in particular contexts.
- An investigation of alignment with time-automata semantics and the transformation to the model-checking tool UPPAAL.

We introduce the related key concepts in the following parts of this section. See the table below for a summary of the proposed language updates for the EAST-ADL BehaviorAnnex (Annexes:: BehaviorConstraints).

Table 2. An overview of main updates for an enhanced behavior description support.

Old Definition	Update(s)	Comment
BehaviorAnnex	**removed**	This top-level container is now removed. The composition support is now given by *BehaviorConstraintType*
BehaviorConstraint	**Replaced** by *BehaviorConstraintType* and *BehaviorConstraintPrototype*	To align with the type-prototype pattern
-	**Added** *BehaviorConstraintType*	(See above)
-	**Added** *BehaviorConstraintPrototype*	(See above)
	Added *BehaviorConstraintType.part* :	(See above)

	BehaviorConstraintPrototype	
	Added *BehaviorConstraintPrototype.type* : *BehaviorConstraintType*	(See above)
-	**Added** *BehaviorInstantiationParameter*	To support the parameterization and instantiations of *BehaviorConstraintPrototype*
-	**Added** *BehaviorConstraintType.parameter* : *BehaviorInstantiationParameter*	(See above)
-	**Added** *BehaviorConstraintType. partBindingParameter* : *BehaviorConstraintBindingParameter*	(See above)
-	**Added** *BehaviorConstraintBindingParameter*	(See above)
-	**Added** *BehaviorConstraintPrototype*. *instantiatedWithParameter* : *BehaviorInstantiationParameter*	(See above)
-	**Added** the specialization of *BehaviorInstantiationParameter* to *BehaviorConstraintBindingParameter*	(See above)
ParameterConstraint	**Renamed** to *AttributeQuantificationConstraint*	"Parameter" is an overloaded term.
Parameter	**Renamed** to *Attribute*	(See above)
ParameterCondition	**Renamed** to *Quantification*	A more exact definition of the role.
StateMachineConstraint	**Renamed to** *TemporalConstraint*	Better support for other constraints on the history of behaviors, which are not directly expressed in SM (e.g. in temporal logic)
Specialization of *BehaviorConstraint* to *ParameterConstraint*	**Replaced** with the aggregation from *BehaviorConstraintType* to *AttributeQuantificationConstraint*	Better support for the internal structuring of content of behavior constraint annotation.
Specialization of *BehaviorConstraint* to *StateMachineConstraint*	**Replaced** with the aggregation from *BehaviorConstraintType* to *TemporalConstraint*	(See above)
Specialization of *BehaviorConstraint* to *ComputationConstraint*	**Replaced** with the aggregation from *BehaviorConstraintType* to *ComputationConstraint*	(See above)
-	**Added** *LogicalEvent*	To support explicitly events related to values.
State.denote* : *ParameterCondition	**Replaced** with *State.quantificationInvariant*: quantificationInvariant	A more exact definition of the role.
-	**Added** *EventOccurrence*	A key concept introduced to integrate existing EAST-ADL constructs for the specifications of various behavior constraints.
-	**Added** the aggregation from *TemporalConstraint* to *EventOccurrence*	(See above)
-	**Added** *EventOccurrence.occurredExecutionEvent* : *Timing::Event*	(See above)
-	**Added** *EventOccurrence.occurredLogicalEvent* :	(See above)

	LogicalEvent	
-	**Added** *EventOccurrence. occurredFeatureFlaw*: *FeatureFlaw*	(See above)
-	**Added** *EventOccurrence. occurredAnomaly*: *Anomaly*	(See above)
-	**Added** *EventOccurrence. occurredHazardousEvent* : *HazardousEvent*	(See above)
Transition.read*:*Parameter	**Replaced** by *Transition.readEventOccurrence?* : *EventOccurrence*	(See above)
Transition.write*:*Parameter	**Replaced** by *Transition.writeEventOccurrence?* : *EventOccurrence*	(See above)
-	**Added** *LogicalTimeCondition*	A key concept introduced to allow fine-grained specification of timing constraints for behaviors, while reusing the support of execution timing for the semantics.
-	**Added** the aggregation from *TemporalConstraint* to *LogicalTimeCondition*	(See above)
-	**Added** *Quantification.timeCondition* : *LogicalTimeCondition*	(See above)
-	**Added** *State.timeInvariant* : *LogicalTimeCondition*	(See above)
-	**Added** *Transition.timeGuard* : *LogicalTimeCondition*	(See above)
-	**Added** *LogicalTransformation.timeInvariant* : *LogicalTimeCondition*	(See above)
-	**Added** *TransformationOccurrance.timeCondition*: *LogicalTimeCondition*	(See above)
-	**Added** *LogicalTimeCondition.upper*: *Timing*::*TimeDuration*	(See above)
-	**Added** *LogicalTimeCondition.lower*: *Timing*::*TimeDuration*	(See above)
-	**Added** *LogicalTimeCondition.width*: *Timing*::*TimeDuration*	(See above)
-	**Added** *LogicalTimeCondition. startPointReference*: *EventOccurrence*	(See above)
-	**Added** *LogicalTimeCondition. endPointReference*: *EventOccurrence*	(See above)
Transformation	**Renamed** to *LogicalTransformation*	A more exact definition of the role.
-	**Added** *LogicalTransformation. clientServerInterfaceOperation* : *Operation*	To merge with existing related constructs.
Transformation.incomingFlow : Flow	**removed**	Unnecessary (due to the new *TransformationOccurrance*).
Transformation. outgoingFlow: Flow	**removed**	(See above)
Flow	**Renamed** to *LogicalPath*	A more exact definition of the role.
-	**Added** *TransformationOccurrance*	Concept introduced to support the invocations of

		logical transformation.
-	**Added** *TransformationOccurrance. invokedLogicalTransformation*: *LogicalTransformation*	(See above)
-	**Added** *Transition.effect* :*LogicalTransformation*	(See above)
-	**Added** *LogicalPath. transformationOccurrance*:*LogicalTransformation*	(See above)
Flow.sinkParameter* : *Parameter	**removed**	Unnecessary (due to the new *TransformationOccurrance*).
Flow.sourceParameter* : *Parameter	**removed**	Unnecessary (due to the new *TransformationOccurrance*).
Flow.orderedSegment* : *Flow	**Replaced** by: *LogicalPath.segment*{ordered} **:** *LogicalPath*	A more exact definition.
-	**Added**: *LogicalPath.strand* **:** *LogicalPath*	(See above)
-	**Added**: *LogicalPath.correspondingExecutionEventChain*: *Timing::EventChain*	To allow the merge of control flows and timing chains.
-	**Added**: *LogicalPath. precedingExecutionEventChain*:*Timing::EventChain*	(See above)
-	**Added**: *LogicalPath. succeedingExecutionEventChain*:*Timing::EventChain*	(See above)

3.2.1 Behavior Constraint Types and Their Targets

The proposed behavior extension provides a language basis for allowing a more precise declaration of various behavior concerns, such as assumed or implied by requirements and quality constraints, assigned to system environment, functions and components, or test procedures. To capture those concerns, three categories of behavior constraints are proposed. It is up to the users of EAST-ADL, in their particular design and analysis contexts, to decide the exact types and degree of constraints to be applied. These categories of behavior constraints are:

- **Attribute Quantification Constraint** – relating to the declarations of value attributes and the related acausal quantifications (e.g., U=I*R).
- **Temporal Constraint** – relating to the declarations of behavior constraints where the history of behaviors on a timeline is taken into consideration.
- **Computation Constraint** – relating to the declarations of cause-effect dependencies of data in terms of logical transformations (for data assignments) and logical paths.

As shown in Figure 3, we distinguish the types of behavior constraints from their prototypes. The latter represent the instantiations of the types in particular context (BehaviorContraintPrototype). The language extension for behavior constraints are currently managed in the BehaviorAnnex. The meta-model integration is done in a modular way such that no existing EAST-ADL constructs are modified by the extension. Also shown in Figure 3, the proposed language extension for behavior constraints can be applied to address a variety of behavioral concerns in a system. The advantages are introduced in the following sub-sections.

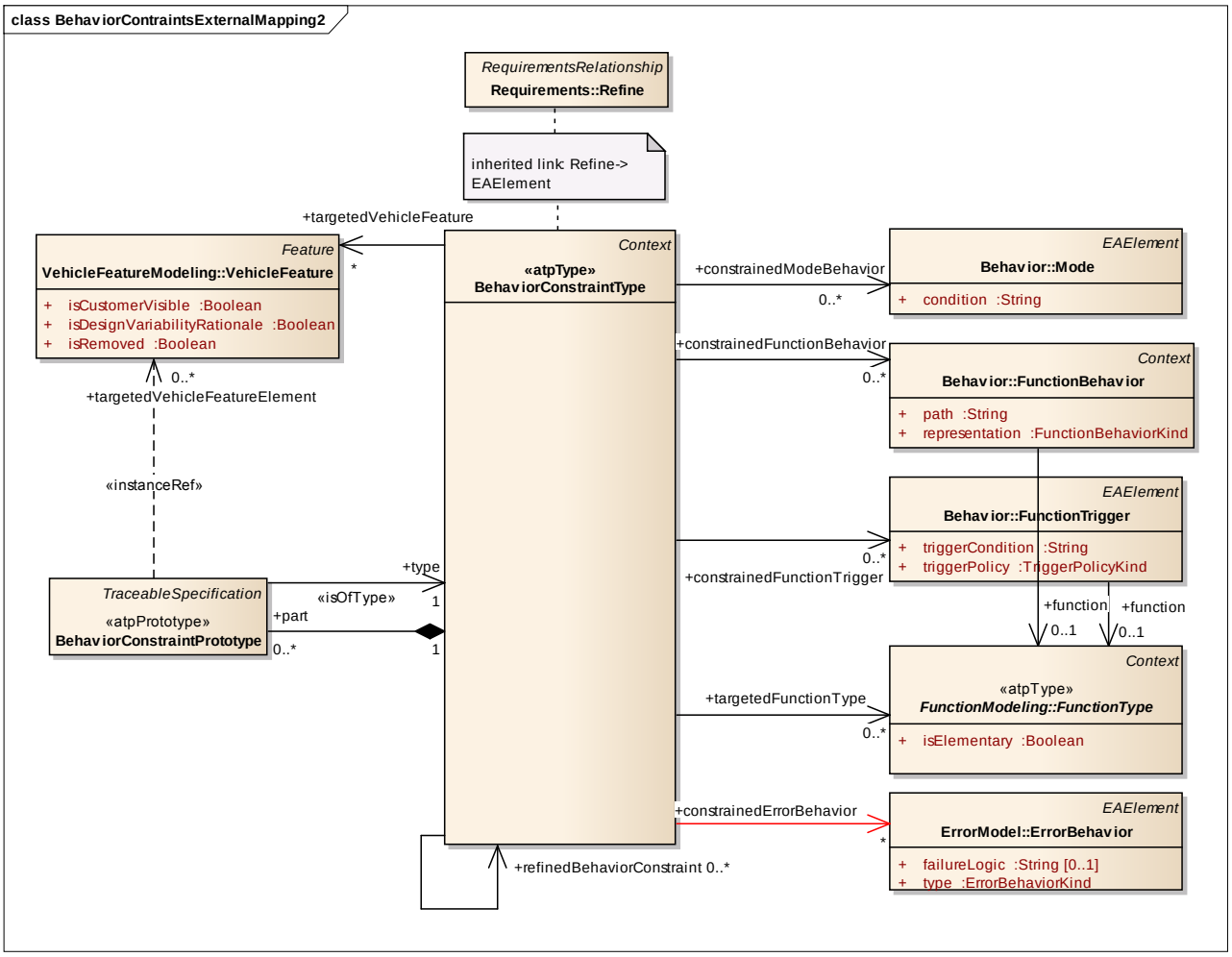


Figure 3. BehaviorConstraintType and the constrained properties in the proposed EAST-ADL2 Behavior Annex.

3.2.1.1 Behavior Constraints for Refinements of Requirements

Through requirement refinement (requirement::refine) relations, behavior constraints can be used to refine the textual statements of requirements, use cases, as well as the assumed operation situations. Such refinements formalize the related behavioral concerns (e.g. the boundary conditions and invariants of variables, states and state transitions) for a more rigorous verification and validation of requirements.

3.2.1.2 Behavior Constraints for Vehicle Features

In EAST-ADL, system functions at the topmost level of abstraction are referred to as vehicle features (VehicleFeatureModeling::VehicleFeature). When assigned to such system functions, behavior constraints are used to capture the related data and behavior characteristics that have to be fulfilled by the target feature. This would constitute a basis for having a more precise reasoning about the configuration of features in terms of feature tree. For example, an assignment of parent-child relation between features may also imply the inheritance of related behavior constraints.

3.2.1.3 Behavior Constraints for Functions and Components

Behavior constraints provide support for specifying the bounds or contracts of acceptable behaviors of functions in a system or its environment. This is achieved by assigning behavior constraints to the function behaviors (which is a container with references to external models and has the run-to-completion semantics), or the function triggers (which declares a triggering policy for the execution of functions) of the target function type. See Figure 4 for a user-model example of applying the behavior constraints to a design function type.

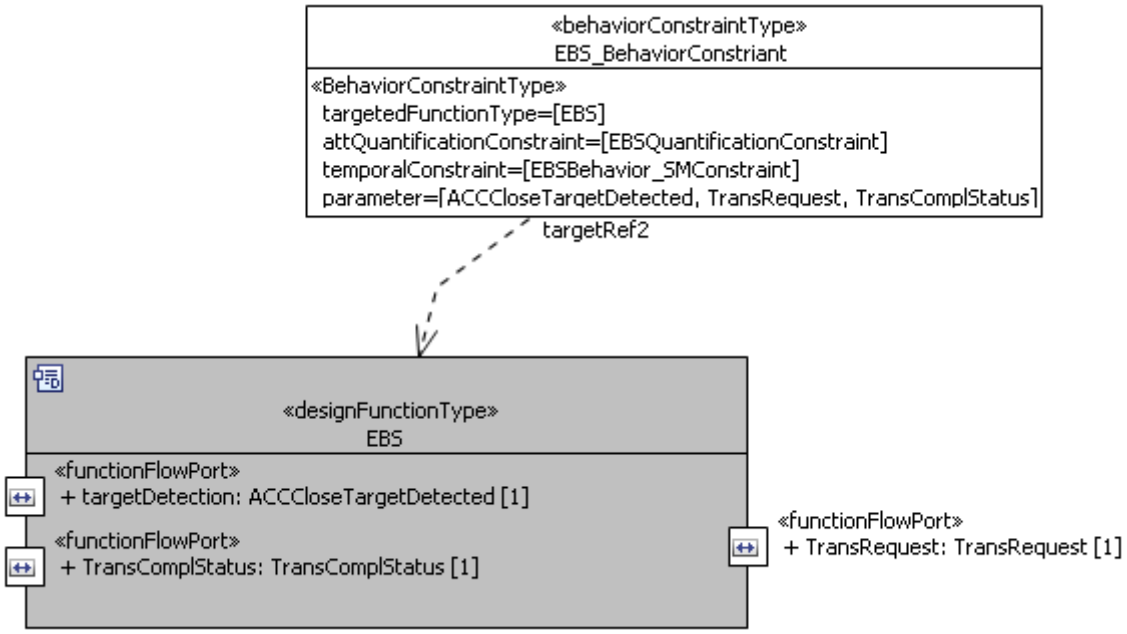


Figure 4. Declaring the behavior constraints of a design function type.

3.2.1.4 Behavior Constraints for Modes

Behavior constraints can also be applied to mode declarations. This modeling feature is not only useful for precisely specifying the mode logics (e.g., to relate modes and the transitions with operational states and resource conditions), but also for specifying the impacts of modes on application behaviors and the system support for quality-of-service management.

3.2.1.5 Behavior Constraints for Error Estimation

While currently focusing on nominal behaviors, the proposed behavior contain extension can also be applied to strengthen the EAST-ADL support for error modeling. When targeting error behaviors, behavior constraints refine the estimated anomalies declared in the error models. This would then allow: precise definitions of faulty conditions in value and time, erroneous states and their transitions, formal analysis of emergent properties due to the compositions. A behavior constraint can be associated to nominal and error behaviors simultaneously. This modeling feature is useful for the specification of fault-injection by allowing transitions across nominal states and errors (for such transitions certain probabilistic attributes will be added).

3.2.2 Attribute Quantification Constraints

Attribute quantifications provide support for the definitions of acausal behavior constraints. They are useful for stating the required value attributes such as the input-, output- and internal

variables, as well as expressing the expected value conditions or invariants in terms of equations like F=m*a, U=I*R. This is comparable with the Modelica approach to behavior specification, where system behaviors are primarily declared based on equations instead of data assignment statements. When necessary, the corresponding data transformations for a quantification can be declared through computation constraints, which are introduced in Section 3.2.4. See Figure 5 for an overview of the related meta-model definitions.

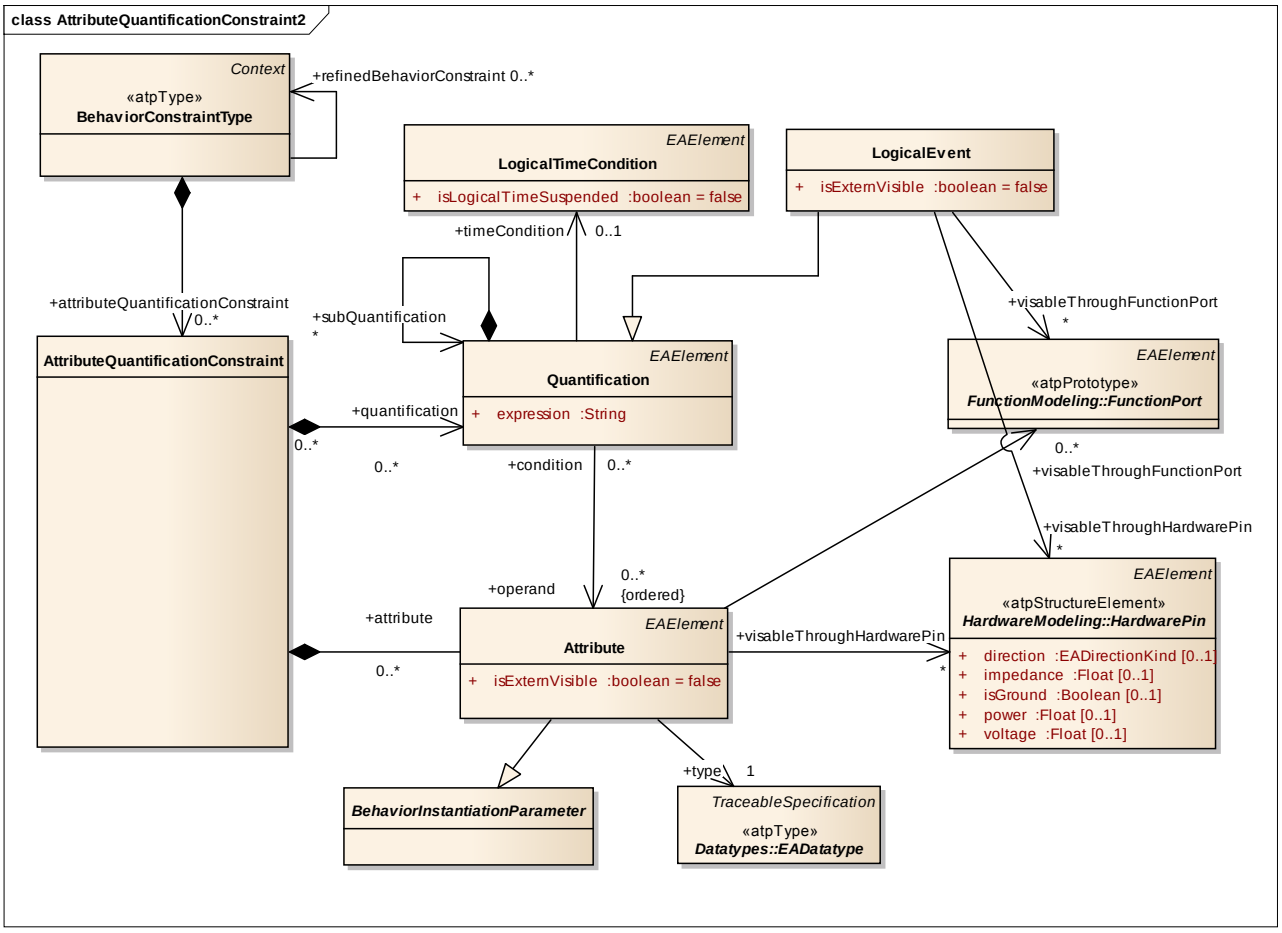


Figure 5. AttributeQuantificationConstraint and its properties in the proposed EAST-ADL2 Behavior Annex.

A specification of attribute quantification constraints is based on the following constructs:

- **AttributeQuantificationConstraint** – the modeling construct for grouping the attribute and quantification declarations in a behavior constraint.
- **Attribute** – the modeling construct for the declarations of the in-, out-, or local variables to be processed or owned by a behavior. Each Attribute is typed by a data type (DataType), specifying the related meta-information like unit, valid range, required accuracy, etc. If an attribute is externally visible (isExternVisble == true), it denotes an input or output variable and has associated function ports or hardware pins for the external accesses. Attributes are instantiation parameters (BehaviorInstantiation-Parameter), to which certain values can be assigned when behavior constraint types are instantiated as behavior constraint prototypes in a context.
- **Quantification** – the modeling construct for the declarations of the value conditions or invariants that an attribute have to obey. For example, a quantification may state that a monitored environmental variable must fall within particular segments in the spectrum of its possible value range during different modes. Each quantification can also be associated with time conditions for stating the time instances or time intervals where the

quantification is valid or takes place. A quantification can be composed of one or several sub-quantifications. (The expression statement is a placeholder for the upcoming support for logical and arithmetic operators and equations.)

- **Logical Event** – the modeling construct for the declarations of the value conditions that, when fulfilled, may trigger state transitions. If a logical event is externally visible (isExternVisble == true), it is disseminated through function ports or hardware pins.

3.2.3 Temporal Constraints

Temporal constraints provide support for capturing the dependency that a behavior has in regard to its own history and other behaviors on a timeline. They can be expressed by means of temporal logic or state-machines. The semantics is based on timed-automata and thereby comparable with approaches like Promela/Spin and UPPAAL in regard to analysis leverage. Compared to those analytical models, the proposed temporal constraints integrate the existing EAST-ADL support for function, communications, executions, and timing, and provide thereby a more exact definition of semantics in regard to the notions of time, events and events synchronizations. See Figure 5 for an overview of the related meta-model definitions.

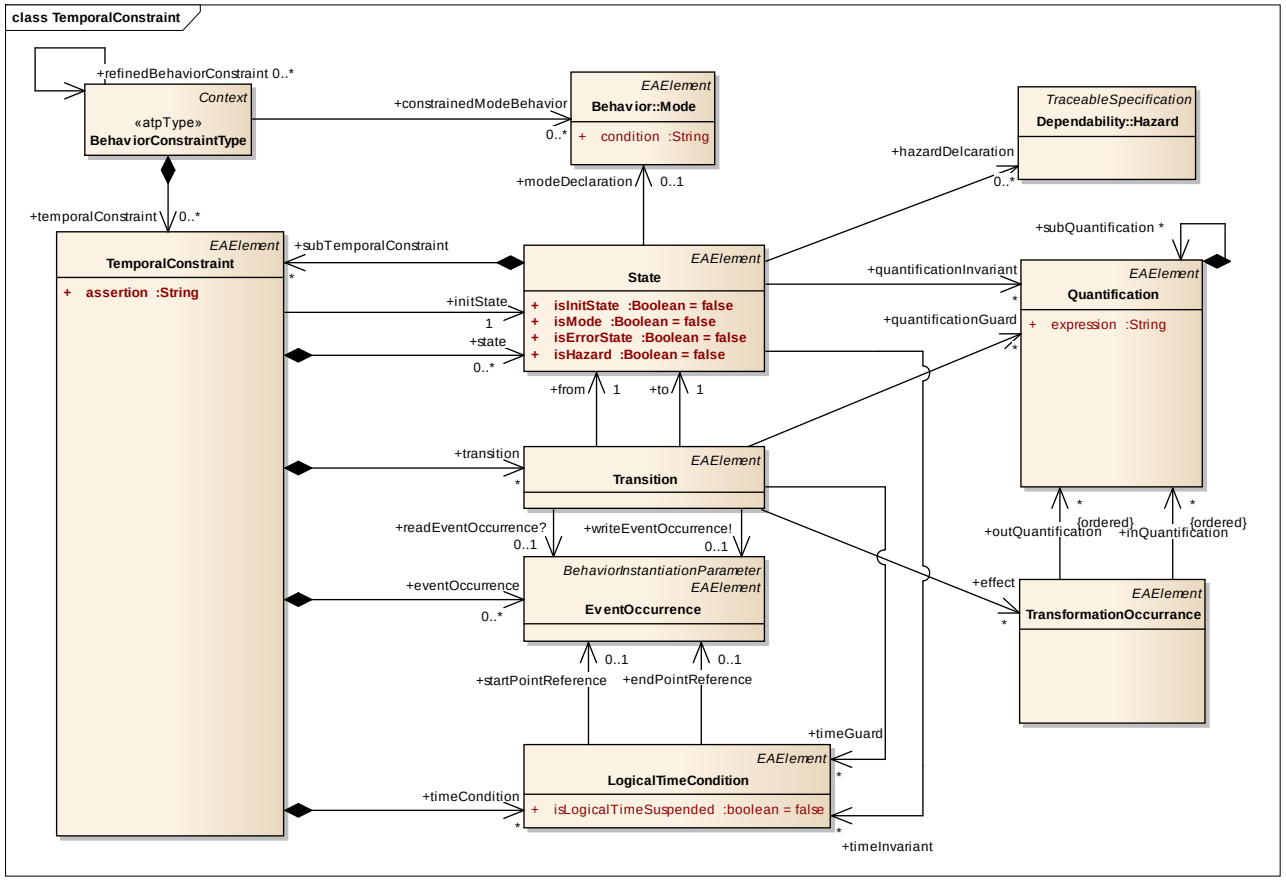


Figure 6. TemporalConstraint and its properties in the proposed EAST-ADL2 Behavior Annex.

A specification of attribute quantification constraints is based on the following constructs:

- **TemporalContraint** – the modeling construct for grouping the declarations of states, transitions, event occurrences, and logical time conditions in a behavior constraint. It can contain assertions in temporal or modal logics when desired. (The assertion attribute (assertion) is a placeholder for the upcoming support for expressions based on temporal/modal logics)

- **State** – the modeling construct for the declarations of states that represent the situations where certain quantifications (Quantification) in terms of value conditions or invariants hold. A state can also have time invariants, representing the time conditions that must be true (e.g., the time duration of a state). In a state-machine based specification, there is always one init state. Each state can have subordinate state machines or other temporal constraint definitions (subTemporalConstraint). Besides nominal operation situations, a state can also represent errors (isError == true), or modes (isMode == true)), or hazards (isMode == true). In the two latter cases, the corresponding declarations of modes (modeDeclaration) and hazards (hazardDeclaration) have to be specified.
- **Transition** – the modeling construct for the declarations of transitions between two states. When the related guard conditions both in time and value domains are met, a transition can be fired to respond to the occurrence of an event (readEventOccurrences?) or to signal the occurrence of an event (writeEventOccurrance!). A transition, when fired, can also invoke one or more logical transformations (TransformationOccurrance).
- **EventOccurrence** – the modeling construct for the declarations of occurrences of events that are either logical events (occurredLogicalEvent), execution specific events (occurred-ExecutionEvent), or fault and failure related (occurredFeatureFlaw, occurredAnomay, occurredHazardEvent). Event-occurrences declared in a behavior constraint type are also instantiation parameters (BehaviorInstantiationParameter), which allow a behavior constraint type to be instantiated as behavior constraint prototypes in different contexts. During the instantiation, such parameters are mapped to some global/external event-occurrences. An occurred event can be purely logical or execution specific.
    - The occurrence of a logical event (LogicalEnvents) denotes a value condition that takes place at a particular time instance and becomes valid in a certain time interval. The semantics is given by the definition of corresponding value condition.
    - The occurrence of an execution event (Timing::Event - an existing EAST-ADL construct from the Timing package) denotes a distinct form of state change in a running system, at distinct points in time, such as at the triggering of a function, or at the receiving/sending of data from/to ports. The definition of execution event itself only provides a description expressing its purpose instead of occurrence.
    - The occurrence of a fault or failure (Dependability::FeatureFlaw, ErrorModel::Anomay, Dependability::HazardEvent – all these are existing EAST-ADL construct from the dependability package) denotes a distinct form of deviation from nominal behaviors at distinct points in time. The definitions of those faults and failures provide the descriptions expressing their estimated existences.

  See Figure 7 for an overview of the related meta-model definitions.
- **LogicalTimeCondition** – the modeling construct for the declarations of time conditions in terms of time instances or time intervals. As shown in Figure 8, such time conditions can be assigned to attribute quantifications, states, transitions, logical transformations or the occurrences of such transformations, in order to characterize the related timing, causality, and synchronization. The logical time condition is an abstraction of real time with the semantics given by the associated occurrences (startPointReference and endPointReference) of execution events (e.g., the triggering event of a function). The value of logical time is defined by a time duration specification (Timing::TimeDuration -

an existing EAST-ADL construct from the Timing package) in the format of CseCode as in AUTOSAR and MSR/ASAM.

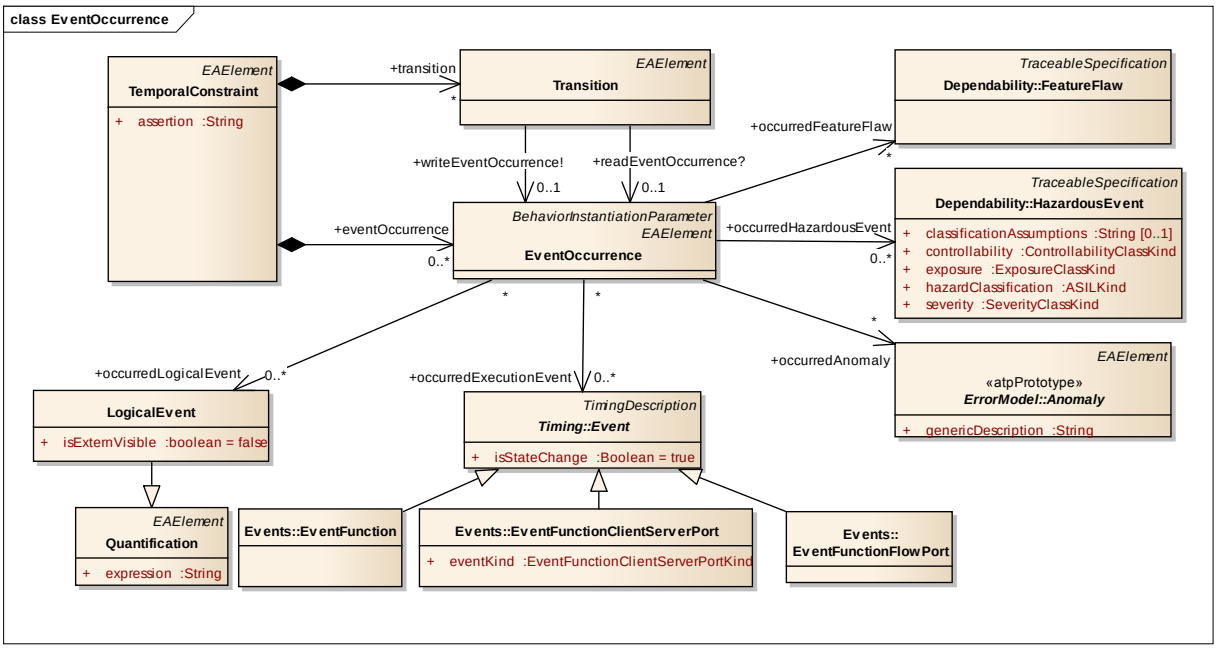


Figure 7. EventOccurrence and its properties in the proposed EAST-ADL2 Behavior Annex.

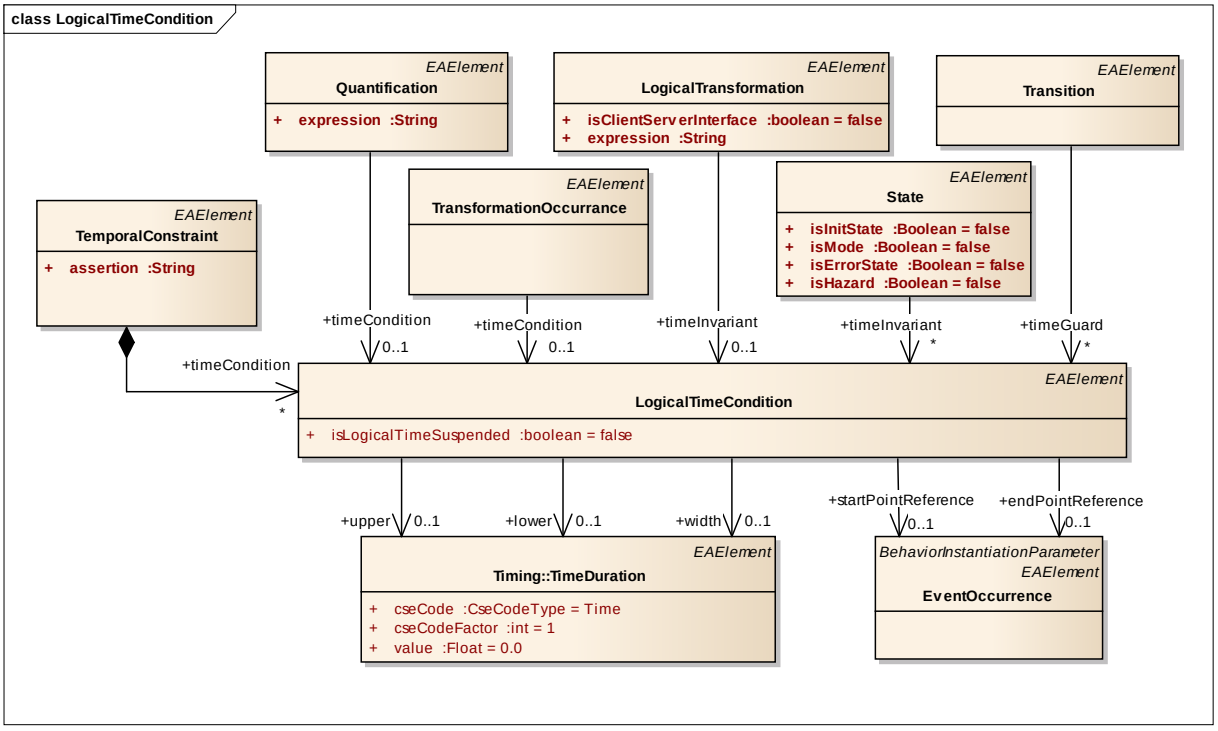


Figure 8. LogicalTimeCondtion and its properties in the proposed EAST-ADL2 Behavior Annex.

See Figure 9 for a user-model example that shows how event occurrences are declared based on execution behaviors. The example system consists of two design functions, a braking control function (pEBS) and a communication transceiver function (pEBSTransiver).

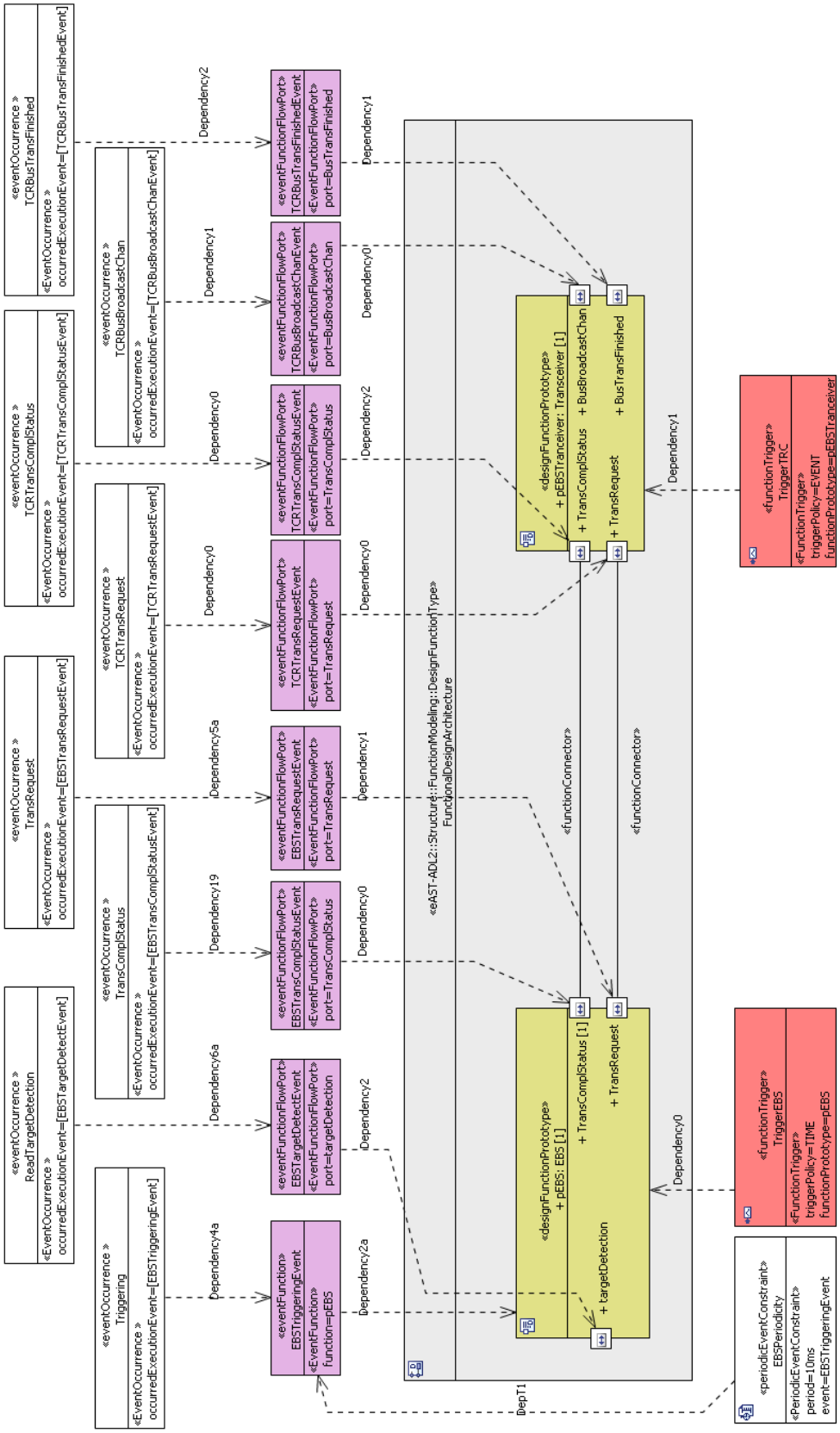


Figure 9. Declaring the occurrences of execution events that express the triggering, data receiving and sending of two functions in a system.

Figure 10 shows a state-machine based specification of the temporal constraint applied to the braking control function (pEBS).

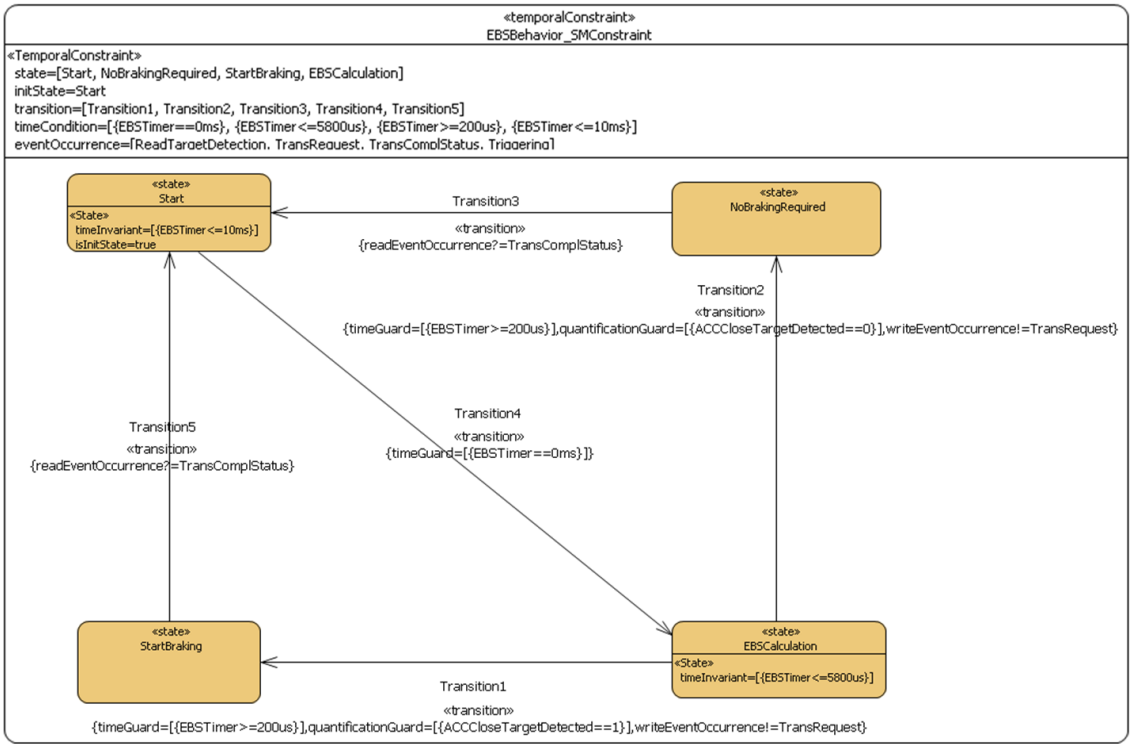


Figure 10. Declaring the temporal constraint of a system function.

The logical time conditions and read&write synchronizations in this temporal constraint are defined based on the occurrences of some execution events. See Figure 11 for a snapshot.

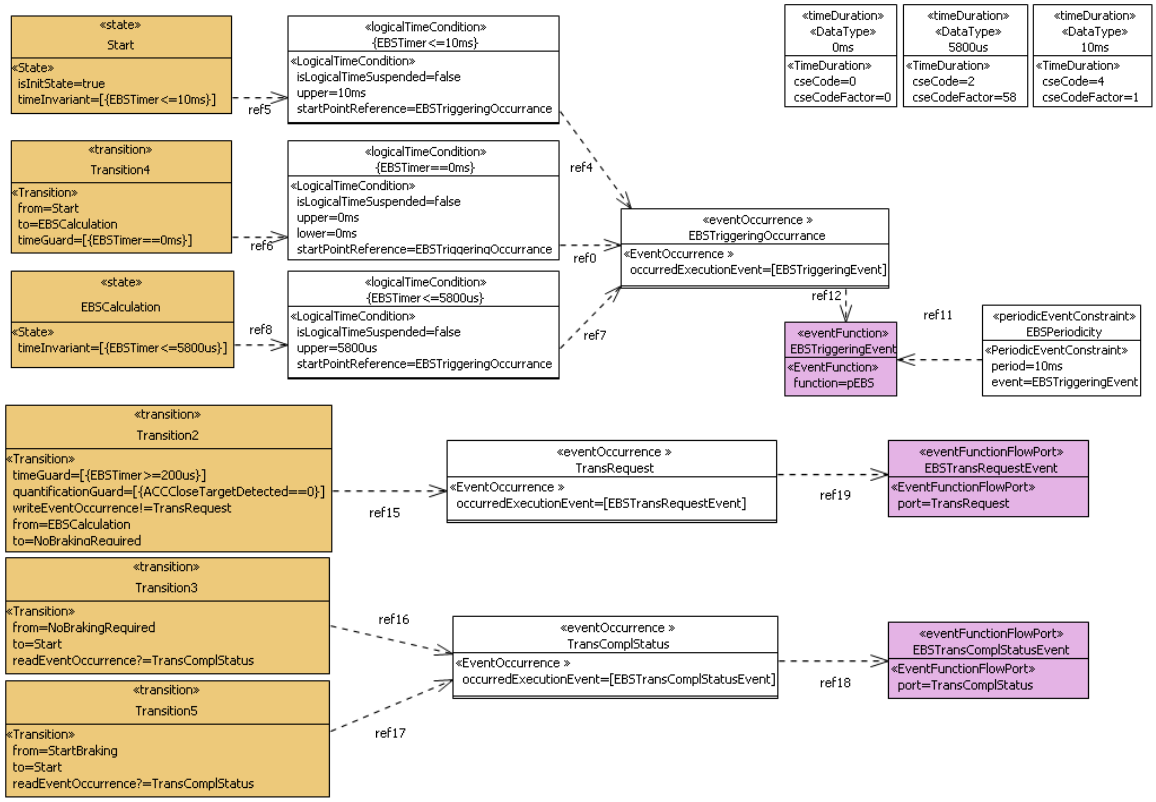


Figure 11. Defining the logical time conditions and read&write synchronizations for a state-machine.

Due to the semantics alignment, the specifications of temporal constraints can be exported and transformed to external models of automata and thereby analyzed through related model-checking

engines. Figure 12 shows the corresponding analytical model in UPPAAL for the temporal constraint specification shown in Figure 10. Compared to the analytical model in UPPAAL, the EAST-ADL temporal constraint declaration complements with detailed architecture information and allows an integration of many related architectural aspects for the purpose of architecture design and management.

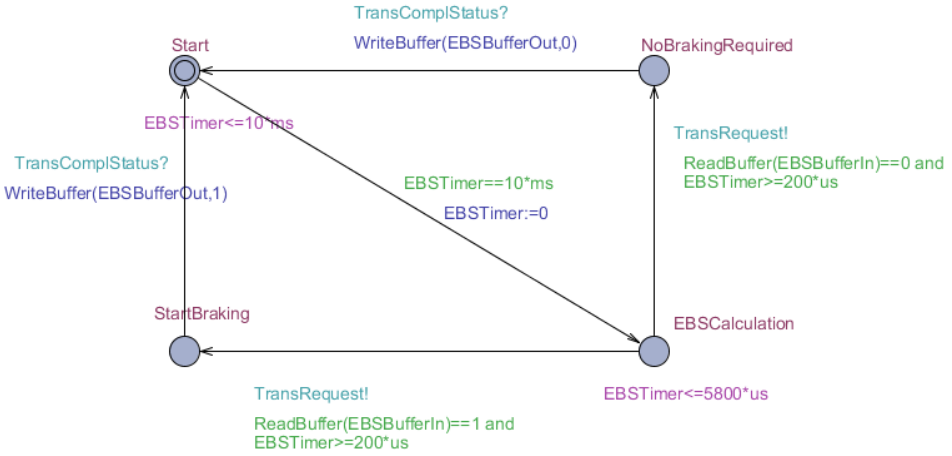


Figure 12. The corresponding UPPAAL analytical model for an EAST-ADL temporal constraint.

3.2.4 Computation Constraint

Computation constraints provide support for the declarations of computation restrictions. They are useful for defining the required logical transformations from input data to output data, as well as the expected causal sequences across such data transformations. Computation constraints are comparable with the UML activity and sequence behavior specification. See Figure 13 for an overview of the related meta-model definitions.

A specification of computation constraints is based on the following constructs:

- **ComputationConstraint** – the modeling construct for grouping the transformation and transformation path declarations in a behavior constraint.
- **LogicalTransformation** – the modeling construct for the declarations of logical computation transformations that determine some out-data (out) by processing some in-data (in) and local-data (contained). Such data are defined in terms of behavior attributes (Attribute). The corresponding value bounds of such data that must be hold before, after, and during the executions of a logical transformation are given by pre-, post-, and invariant conditions respectively. A logical transformation can also have time invariants (timeInvariants), stating the duration bounds when the transformation takes place. If a logical transformation is externally accessible (isClientServerInterface == true), it represents the operations declared in client-server interfaces (clientServerInterfaceOperation). A logical transformation can also have subordinate computation constraints (subComputationConstraint). (The expression attribute (expression) is a placeholder for the upcoming support for expressions of computation logics)
- **TransformationOccurrence** – the modeling construct for the declarations of invocations of logical transformations as the effects of state transitions and control flows. A transformation occurrence can also have a time condition (timeCondition), stating the time instances when the invocation happens. If a logical transformation is invoked, its in-data will be assigned with particular values by the invocation context

(inQuantification). As the consequence of transformation, the out-data will also be assigned with particular value (outQuantification).

- **LogicalPath** – the modeling construct for the declarations of the cause-effect paths connecting execution events, logical transformations, and logical events. One main advantage is that the internal causality of functions/components can now be merged explicitly with the related external execution events. When applied to a function/component, a logical path captures the control flow from some logical events (logicalStimulus) or execution events (precedingEventChain) to some other logical events (logicalResponse) or execution events (succeedingEventChain). In each logical path, some logical transformation can be invoked (TransformationOccurrence) either directly or in synchronization with state transitions. By describing the internal causality of a function/component, a logical path may refine an execution event chain (correspondingExecutionEventChain), which is primarily used to capture the causality of triggering, port reading and writing events. Logical paths can be combined in parallel (strand) or in sequence (segment).

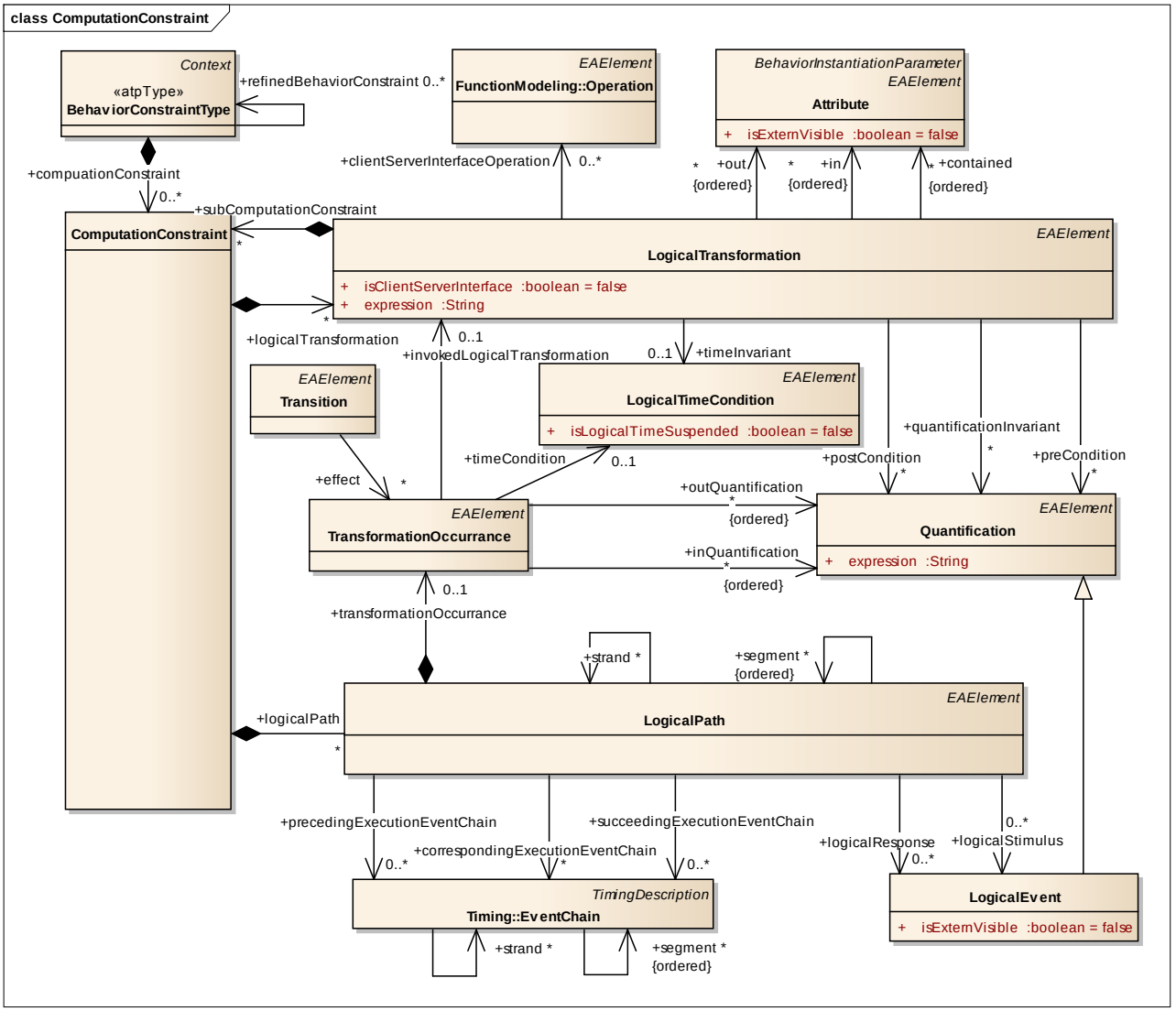


Figure 13. ComputationConstraint and its properties in the proposed EAST-ADL2 Behavior Annex.

3.2.5 Instantiations of Behavior Constraint Types

A behavior constraint has both type and prototype(s), following the type-prototype pattern in EAST-ADL. While a type definition provides the template for a range of behaviors, a prototype definition specifies a particular behavior instance in a context. See Figure 14 for an overview of the related meta-model definitions.

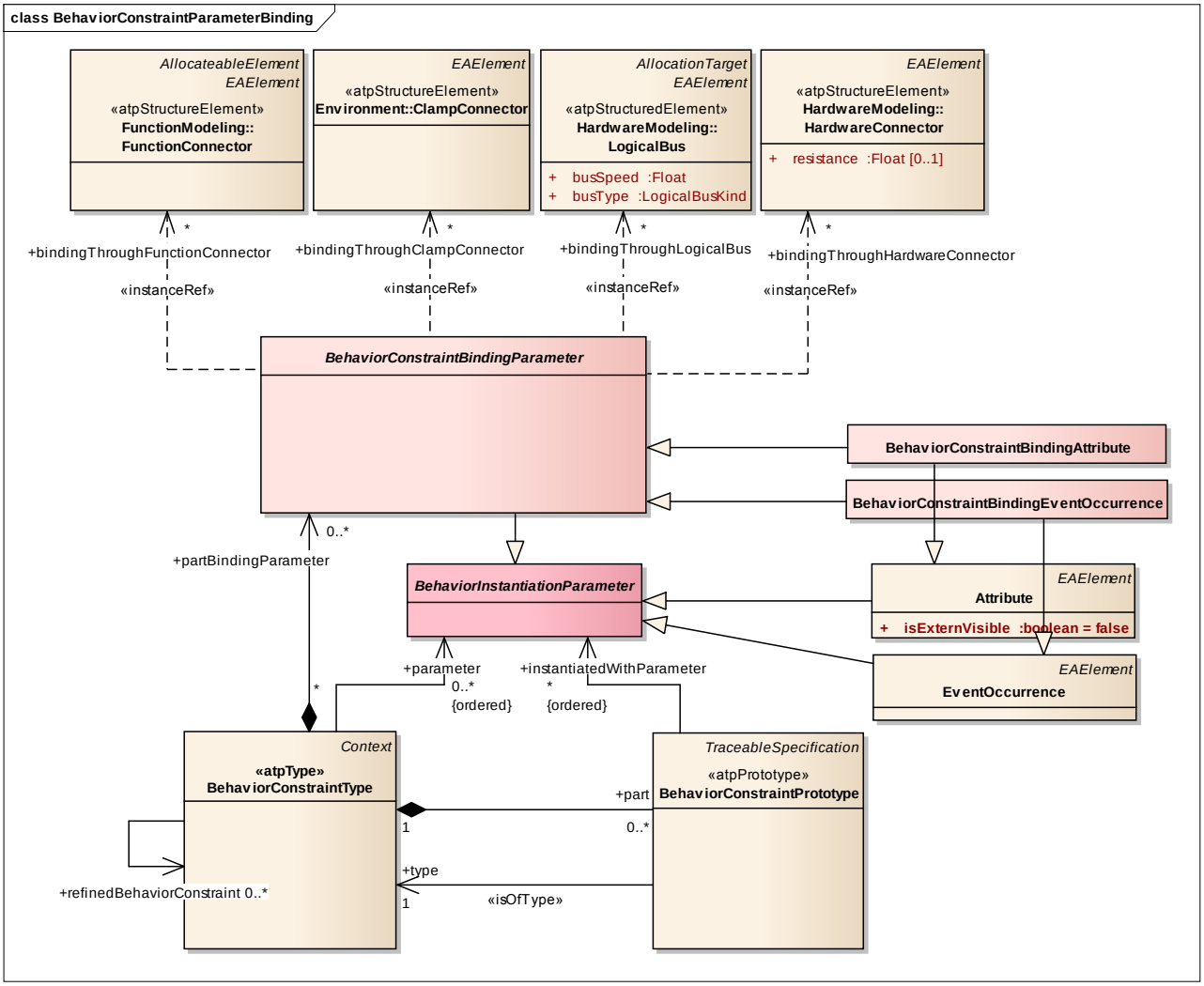


Figure 14. BehaviorConstraintType, BehaviorConstraintPrototype, and the related modeling instantiation constructs in the proposed EAST-ADL2 Behavior Annex.

The support for the instantiations of behavior constraint types is based on the following constructs:

- **BehaviorConstraintPrototype** – the modeling construct for the declarations of the occurrence(s) of a behavior constraint type (type) in a particular context where it acts as a part (part).
- **BehaviorInstantiationParameter** – the modeling construct for the declarations of the parameters (parameter) that a behavior constraint type offer for its instantiations in terms of prototypes. During the instantiation, the parameters of a behavior constraint type will be bound to some contextual parameters (instantiatedWithParameter) and thereby be assigned with the values of those contextual parameters. A BehaviorInstantiation-Parameter can be a value attribute (Attribute), an event occurrence (EventOccurrence), or an internal binding parameter (BehaviorConstraintBindingParameter).

- **BehaviorConstraintBindingParameter** – the modeling construct for the declaration of parameters (partBindingParameter) that a behavior constraint type has for binding its parts. In effect, such parameters can be shared by all parts in the same context. Each binding parameter can have a structural correspondence (bindingThroughFunctionConnector, bindingThroughClampConnector, bindingThrough-LogicalBus, or bindingThrough-HardwareConnector), stating the structural channels through which the binding takes place. In the meta-model, the abstract binding parameter is further specialized into
    - **BehaviorConstraintBindingAttribute** – the contextual parameters that are value attributes.
    - **BehaviorConstraintBindingEventOccurrence** – the contextual parameters that are event occurrences.

See Figure 15 for a user-model example that shows how the behavior constraint type (EBS_BehaviorConstraint) of the example braking control function (EBS) is instantiated as a prototype (eBS_BehaviorConstraint) in a particular context (FunctionDesignArchitecture_ BehaviorConstraint). The supported part binding parameters in the context are given as a set of sharable event occurrences. Such binding parameters play the roles of synchronization connectors (i.e. rendezvous) with structural correspondences in terms of functional connectors or physical channels.

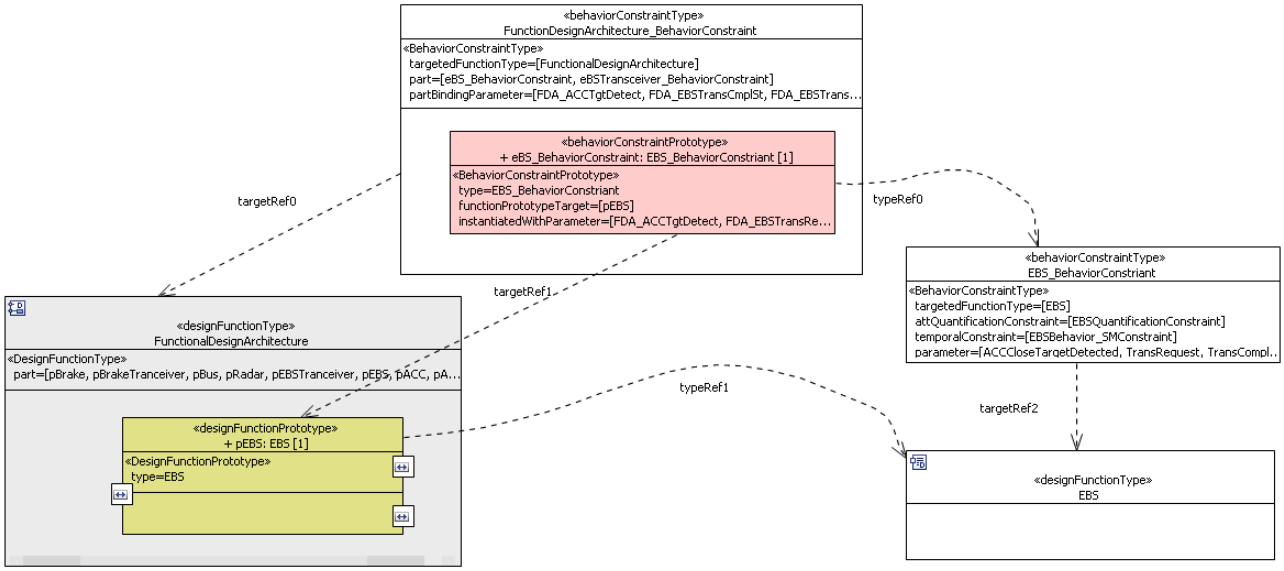


Figure 15. Declaring the behavior constraint of a system function and instantiating the constraint in an architecture context.

Two or more behavior constraint prototypes in the same context are bound if they share the same binding parameters. One user-model example is shown in Figure 16. The behavior constraint of the braking control function (EBS) declares an ordered set of instantiation parameters:

[ACCCloseTargetDetected, TransRequest, TransComplSt]

In its prototype instantiation, such parameters are assigned through an ordered set of binding parameters:

[FDA_ACCCloseTargetDetected, FDA_TransRequest, FDA_TransComplSt]

In such a way, a behavioral binding of the braking control function (EBS) and the communication transceiver (Transceiver) is established. This is illustrated in Figure 16. Assume that the behavior constraint of transceiver function (EBS) has the instantiation parameter declaration: [TransRequest, TransComplSt]. It is instantiated with: [FDA_TransRequest, FDA_TransComplSt]. Under the circumstance, the read&write event occurrences of transitions in the respective state-

machine constraints of these two functions will be sychronized. Figure 16 also shows the declarations of corresponding structural connectors of the binding parameters.

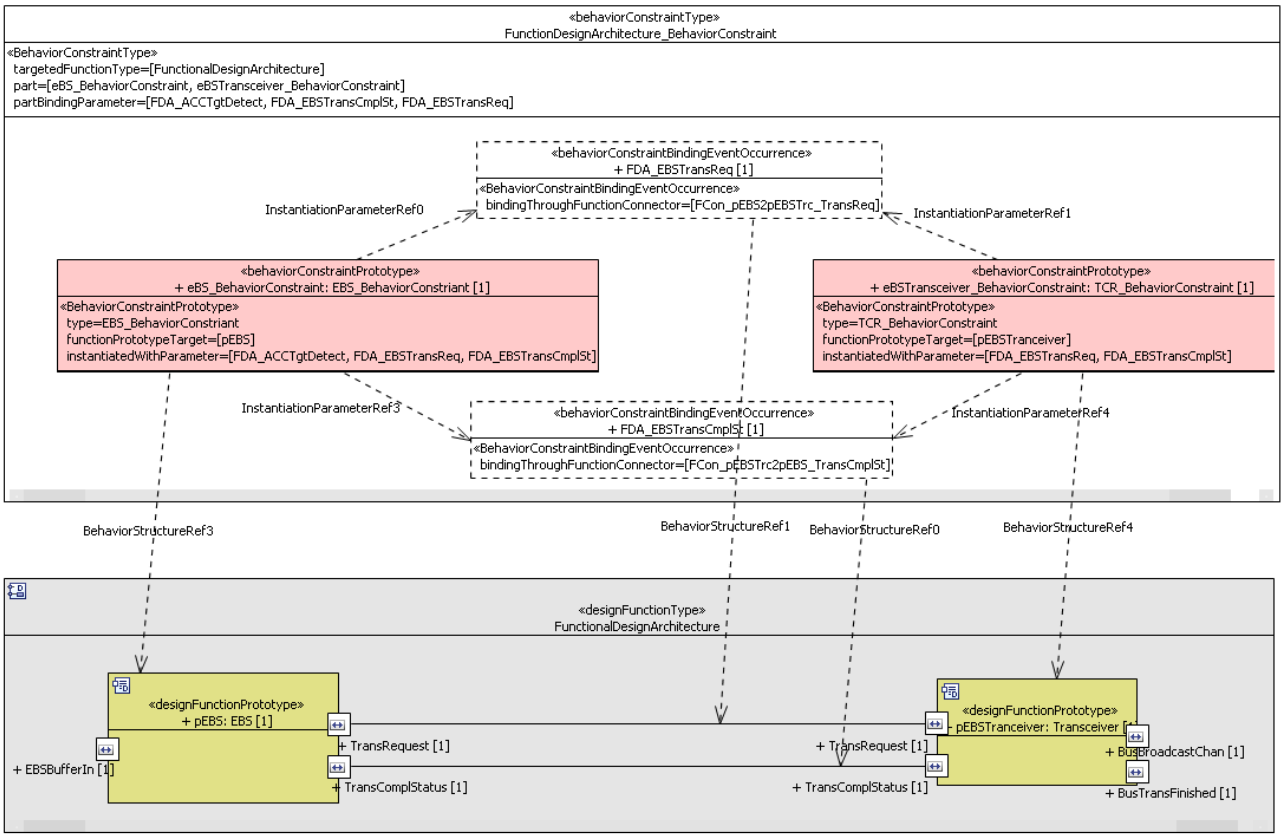


Figure 16. Declaring the binding of behavior constraint prototypes in an architecture context.

3.3 Upcoming Activities

EAST-ADL could provide many benefits for analytical and architectural decision-making, information exchange and management in the development of FEV. As an ontology and formalism of embedded systems, the language would also constitute an important basis for the integration of state-of-the-art analysis methods and techniques from computer science, electronics&electrical engineering, and other related disciplines. This would allow a seamlessly integrated analysis support in the entire lifecycle of system development. For many advanced analysis, it is of critical importance that an alignment of EAST-ADL analytical models with the related analysis methods and tools is clearly defined.

Upcoming EAST-ADL support for the analysis of FEV will address the assessment of FEV properties, either of the system itself or of its operational situation, in both logical and physical domains. Such a support will allow enhanced language support for the elicitation of safety goals, the descriptions, verification and validation of safety requirements, as well as for the assessments of mode-sensitive system compositionality and composabiltiy in general. In the upcoming project period, an alignment of the proposed behavior constraint specification support with external formalisms and analysis engines including UPPAAL, SPIN, Modelica will be supported. Current EAST-ADL provides analytical modeling support in regard to timing, faults and error propagation. Future work will investigate the analysis leverage by adopting such external formalisms and engines via the proposed modeling enhancement for behavior constraint description.

4 Modeling Concepts for Supporting Timing Analysis

4.1 Background

Automotive applications have to fulfill stringent timing constraints to function properly. For example, power train and chassis applications include complex (multi-variable) control laws, with different sampling rates, for use in conveying real-time information to distributed devices. One hard real-time constraint is ignition timing, which varies with engine position. The latter is defined by a periodic event characterizing the flywheel zero position. End-to-end response times must also be bounded, because a too long control loop response time may not only degrade performance, but also cause vehicle instability. As these constraints have to be met in every possible situation, there is a strong need to perform timing analysis and verification of these applications.

ISO 26262 requirements, moreover, impose to integrate system safety analysis in the development process. System safety analysis determines ASIL (Automotive Safety Integrity Level) levels according to 26262 standard. Based on a given level, devoted safety mechanisms and/or methods (e.g. software redundancy, graceful degradation, etc.) are recommended, to guarantee this level of safety. These mechanisms may highly impact timing behavior of the system since they generally involve additional resources consumption. For this reason, the ISO 26262 recommends to consider the verification of timing requirements, together with safety requirements, during the whole software development process

Automotive software development costs are sharply impacted by wrong design choices made in the early stages of development but often detected after implementation. Most timing-related verifications are addressed very late, in the development process, during implementation or in the system integration phase. The need of defining an approach that permits timing verification throughout the development process, starting from the early phases of design, is thus obvious. Such an approach would enable early prediction of system timing behaviour and would allow potential weak points in design to be detected as early as possible.

In this context, quantitative analysis techniques (such as scheduling analysis) [1] are good candidates for analyzing non-functional properties at the design stage. Using these techniques, designers could detect infeasible real-time architectures, and therefore prevent costly design mistakes, while providing an analytical basis for assessing the design tradeoffs associated with resources optimization.

Model-based engineering (MBE) represents a means for mastering system complexity and assessing system-level tradeoffs geared to achieving higher quality and dependability [2]. Specific advantages expected from this approach are the ability to employ correct-by-construction, but also incremental design processes (which rely extensively on automated transformations and synthesis) and to formalize computer-based correctness analysis [5]. In this context, one challenging problem in model-based timing analysis is the integration of commonly used architecture models with information that is relevant to analysis. In order to perform timing analyses, it is necessary to transform the representation of system architecture into some specific form that can be mathematically evaluated (denoted here as the "analyzable model"). Analysis tools accept such analyzable models as inputs, and then evaluate them mathematically to produce the results needed for successive refinements of architecture models.

In this context, many model-based approaches have been recently developed for the automotive domain. Projects as ATESST and ATESST2 [3], TIMMO [4], EDONA [5] have been carried out to provide concepts, tools and methodologies for the description of automotive architectures and timing properties.

As for schedulability analysis, a framework, developed in the context of the EDONA project enables performance of model-based scheduling analysis at the EAST-ADL Design level. This

framework is based on supplementing EAST-ADL with concepts from the modelling language MARTE [6] following the procedure shown in [7]. In this framework, scheduling analysis is performed based on an automatic transformation of MARTE models to an academic scheduling analysis tool called MAST [8].

This approach, however, suffers from the following limitations:

- The scheduling analysis framework defined at the Design level does not describe how the scheduling analysis results should exploited *to improve* the architecture designed at that same level, if these results reveal that the system is not schedulable.
- The scheduling analysis framework at the Design level lacks for a methodology describing how scheduling analysis results should be exploited *to refine* the system architecture at the subsequent Implementation level.

In order to overcome the above mentioned limitations, we need to re-think to the type of timing analysis that better adapt to EAST-ADL Design Level. In order to identify those analyses, the following general principles should apply:

Per-level timing analysis ***fitness****:* For each abstraction level, timing analysis only applies on design concepts allowed at that description level. For instance, at Design Level the concept of OS task is absent. In order to apply scheduling analysis at this stage, as proposed in [7] we should on the other hand define a task model as entry. This means that carrying out scheduling analysis at that level implies to make implicit refinement of the Functional Design Architecture towards a refined architecture that details the mapping of functions on tasks. We advocate that architecture refinements should always be explicit (traced between architecture models) and that timing analysis should work at the same level of abstraction the analysed model lies on. This is of paramount importance *to correctly feedback results on the entry model*. If results are related to implicit refinements, it is hard to extrapolate a feedback to improve the entry model (e.g. the functional architecture), as maybe it is only the refinement to improve (e.g. the mapping of functions on tasks) and not the architecture itself.

Inter-level timing analysis ***coherence***: At each level, applied timing analysis should provide 1) insights to rapidly discard wrong architectural choices and 2) to prepare more detailed timing analysis at the subsequent refined level. Positive results of timing analysis at some level should be further confirmed by subsequent analysis (as more detailed architectures will be defined), but negative results should be as much as possible ‘true negative’ to correctly prune the solution space.

In the context of the MAENAD project, we decided to support at EAST-ADL design level the following timing analyses [11]:

- *Early Stage Schedulability Analysis*. The allocation model of EAST-ADL defines on which ECUs, functions will be executed and on which buses, communication between functions will take place. Basing on this information, the following two interesting metrics, relevant from a schedulability point of view, can be considered [11]:
    - Resource Utilization. Resource utilization is a function of (i) the function’s activation rate and (ii) a time budget representing the time an execution/communication will take. Basing on utilization of single resources, other related interesting metrics can also be extracted, as load distribution and function/signals extensibility (function of processors/bus slacks).
    - Interference Time: represents the waiting time to access shared resources (CPU/Bus). This delay is caused by concurrent functions/signals that are allocated to the same execution/communication node. Small interference is desirable to minimize end-to-end latency.
- *Schedulability analysis*. As remarked above schedulability analysis takes as input a task model which is not defined at Design level. Schedulbaility analysis, thus, does not seem

meeting the fitness principle for the Design level, where the concept of task is absent. On the other hand for the special case of linear chains of activations running on a mono-processor system, the scenario-based mapping has proved to be the most appropriate [9]. In this case each linear activation chain is regrouped in a thread of execution and priorities are assigned following a rate monotonic assignment. Note that this is possible only if in the chain rates are harmonic. Once the task model is obtained, a response time will be computed for each end-to-end chain (thread) trough Rate Monotonic Analysis [10]. If the response-time does not meet end-to-end deadlines, the current evaluated allocation can be safely pruned, meeting the coherence principle. Let us also remark that the scenario-based mapping can be automatically applied behind the scene, being then transparent to the user.

4.2 EAST-ADL support for Timing Analysis

As highlighted in Section 3.1, we are interested in two types of analysis (i) early stage schedulability analysis (resource utilization and interference time analysis) and (ii) traditional schedulability analysis in case of mono-processor systems and linear end-to-end activation chains. In both cases the analyses work on the level of abstraction offered by the Design level, without the need of further refinements towards Implementation-like concepts. Fundamental concepts used by the analysis are (1) activation chains of functions, the rate of their activation, nominal execution times and end-to-end deadlines, (2) the topology of the hardware network in terms of processors and the buses that connect processors along with their maximal utilization capacity and (3) the allocation of functions on processors.

In the following we detail EAST-ADL concepts for analysis and their use. They mainly come following from EAST-ADL FunctionModelling, HardwareModelling and Timing.

4.2.1 EAST ADL concepts for Timing Analysis from FunctionModeling

EAST-ADL defines the concept of FunctionType which abstracts the function component types used to model functional structure. The leaf functions of an EAST-ADL function hierarchy are called Elementary Functions. Elementary Functions have synchronous execution semantics: each function's activation is divided in the following steps: inputs reading, execution (transfer function), output writing. If a behavior is attached to the FunctionType, the execution semantic complies with the run-to-completion semantic. This has the following implications:

1. Input that arrives at the input FunctionPorts after execution begins will be ignored until the next execution cycle.
2. If more than one input value arrives per FunctionPort before execution begins, the last value will override all previous ones in the public part of the input FunctionPort (single element buffers for input).
3. The local part of a FunctionPort does not change its value during execution of the behavior.
4. During an execution cycle, only one output value can be sent per FunctionPort. If consecutive output values are produced on the same FunctionPort during a single execution cycle, the last value will override all previous ones on the output FunctionPort (single element buffers for output).
5. Output will not be available at an output FunctionPort before execution ends.
6. Elementary FunctionTypes may not produce any side effects (i.e., all data passes the FunctionPorts).

This synchronous and run-to-completion semantics is taken into account in timing analysis.

The other concepts of interest for timing analysis are those concepts used when an allocation is defined. The FunctionAllocation concept represents an allocation constraint binding an AllocateableElement (computation functions or communication connectors) on an AllocationTarget (computation resource or logical bus). AllocateableElement is specialized by the concrete elements DesignFunctionPrototype/FunctionConnector in the FunctionModeling package and AllocationTarget is specialized by HardwareComponentPrototype in the HardwareModeling package (see next Section).

4.2.2 EAST ADL concepts for Timing Analysis from HardwareModeling

The HardwareComponentType represents hardware element on an abstract level, allowing preliminary engineering activities related to hardware. Relevant associations are the owned hardware connectors, owned parts, portGroups and ports. Hardware connectors represent wires that electrically connect the hardware components through its ports.

For our purposes, we are interested in one specialization of this concept, namely Node and in the concept of LogicalBus, as detailed as follows:

Node represents the computer nodes of the embedded electrical/electronic system. Node acts as a computing element executing Functions. Nodes consist of processor(s) and may be connected to sensors, actuators and other ECUs via a HardwareConnector. In case a single CPU ECU is represented, it is sufficient to have a single, non-hierarchical Node. The main relevant attribute for analysis activity is executionRate. A nominal execution time (attributed to functions allocated on the node, see Section 3.2.3 for details) divided by executionRate provides the actual execution time to be used for timing analysis.

LogicalBus represents logical communication channels. The LogicalBus groups a set of wires (the associated hardwareConnectors) and it represents a logical connection that carries data from any sender to all receivers. Senders and receivers are identified by the wires of the LogicalBus. The available busSpeed represents the maximum amount of useful data that can be carried.

Note that LogicalBus serves as an allocation target for connectors, i.e. the data exchanged between functions in the FunctionalDesignArchitecture. In order to allocate function on computing resource, the HardwareComponentPrototype must be used. The hardwareComponentPrototype allows for a reference to the occurrence of a HardwareComponentType when it acts as a part.

4.2.3 EAST ADL concepts for Timing Analysis from Timing

For timing analysis purposes, we need to represent end-to-end activation chains of functions. For each end-to-end activation chain, the stimulus for the activation chain must be specified. The stimulus should be characterized in terms of its arrival pattern, i.e. a characterization of stimulus occurrences over time. The end-to-end deadline for each end-to-end activation chain and execution time for each single function's activation must be specified as well.

EAST-ADL Timing offers the all above-mentioned concepts, structured in three different packages: *Timing* which defines core elements and their organization, *Events* which lists various kinds of events that can be associated to structural elements, such arrival of data on ports, and *TimingConstraints* which lists all possible constraints one can model – delays, synchronization, etc.

The modeling principle is the following: *TimingConstraints* are associated to an *EventChain*, which defines the scope of the constraint. An EventChain is composed by a stimulus and a response.

The stimulus/response of an EventChain is an event. *Events* refer to structural elements as functions/ports. Events can be of different types:

Event Function: it represents the case in which the function, the event refers to, is activated via a trigger. It is used in conjunction with FunctionTrigger (from Behavior) to define *a time-driven triggering* for a function. In this case the FunctionTrigger points to the EventFunction and defines a triggerPolicy set to TIME (enumerated value).

EventFunctionFlowPort: it represents the case when the function is activated by data sent/received at the function's flow port the event refers to.

EventFunctionClientServerPort: it represents the case when the function is activated by data sent/received at the function's client/server port the event refers to.

In order to specify how the event occurs over time, EAST-ADL provides the EventConstraint language element. Each eventConstraint is associated to an Event. In EAST-ADL, we have the following kinds of EventConstraints: PeriodicEventConstraint (it specifies that an event occurs periodically), SporadicEventConstraint (it specifies that an event occurs sporadically), PatternEventConstraint (it specifies that an event occurs following a certain pattern) and ArbitraryEventConstraint (specifies that an event occurs following an arbitrary pattern). Note that for timing analysis purposes we need to restrict EventConstraint to PeriodicEventContraint or SporadicEventConstraint where the minimum inter-arrival time is specified.

In order to specify end-to-end deadlines we need to define a TimingConstraint whose scope will be the end-to-end event chain representing the system response. Local deadlines can be also specified by decomposing the original end-to-end chain in segments. In particular DelayConstraint should be used. Delay constraints can be of two types**:** ReactionConstraint and Age Timing constraint. For timing analysis purposes, ReactionConstraint should be used.

In order to specify execution time for single functions, EAST-ADL offers the ExecutionTimeConstraint concept. ExecutionTimeConstraint expresses the execution time of a function under the assumption of a nominal CPU that executes 1 "function second" per second. Function allocation will decide the actual execution time by multiplication with the relative speed of the host CPU. The inherited attribute `lower` (from TimingConstraint) denotes the minimal best case execution time. The inherited attribute `upper`(from TimingConstraint) denotes the maximal worst case execution time. Additional associations allow assigning the ExecutionTimeConstraint to a DesignFunctionType or DesignFunctionPrototype, respectively. Another additional association, called `variation` denotes the allowed variation in execution time, i.e. maximal minimal execution time.

4.3 Discussion

A pointed out in section 3.1, timing analysis at Design Level is mainly concerned with early-stage schedulability analysis (resource utilization and interference time) analysis and schedulability analysis in the mono-processor and linear activation chains case. This conclusion is an achievement of the MAENAD project as at the end of ATESST 2, only schedulability analysis have been selected with the idea of refining the Design Level of EAST-ADL towards an implementation level. As explained in Section 3.1 this approach has several drawbacks and indeed violates the fitness principle. For this reason schedulability analysis is carried out only when an automatic refinement towards an implementation makes sense without biasing the evaluation of the design level. Being the early-stage schedulabiilty analysis, a quite new analysis concept, EAST-ADL proved to be a very mature language as almost all relevant concepts are covered. The only point not explicitly addressed is the characterization of the maximal resource utilization capacity for nodes/buses. In many cases the maximal authorized utilization for a resource must be specified, and resource utilization analysis should verify that the allocation choices met utilization constraints. For the moment we can use the existing GenericConstraint element in the EAST-ADL language (GenericConstraints extension) to make such assumptions explicit in the model. At most for the

time being, a potential revision of GenericConstraintKind with additional Literals might be envisioned (e.g. utilization). Let us note that in MARTE both ExecutionHost and CommunicationHost have a specific attribute to provide maximal utilization capacity.

Another point is about the possibility of storing analysis results in the model. Computed resource utilizations/response times, which are the actual values for the architecture under evaluation, should be recorded. This leads to a more general problem about the support of the language for architectural exploration. In the case several candidates must be evaluated and compared, a candidate should be annotated as special element in which analysis parameters/results might be stored. For the time being we are using MARTE concepts to support allocation exploration. MARTE provides the notion of AnalysisContext to actually define a candidate under evaluation through specific analyses. For a given analysis, the context identifies the model elements of interest (function activation chains and platform resources) and specifies global parameters of the analysis. An allocation can be also affected to the analysis context. As for analysis results, they can be stored using variables defined for the AnalysisContext.

References

1. B. Selic: "A Generic Framework for Modelling Resources with UML", IEEE Computer vol. 33 no.6, pp.64-69 2000.
2. A. Pretschner, M. Broy, I. H. Kruger, T. Stauner: "Software Engineering for Automotive Systems: A Roadmap", in 29th International Conference on Software Engineering (ICSE 2007), 2007
3. ATESST project website: http://www.atesst.org
4. TIMMO project website www.timmo.org
5. EDONA project website http://www.edona.fr
6. MARTE website www.omgmarte.org
7. S. Anssi, S. Tucci-pergiovanni, C. Mraidha, A. Albinet, F. Terrier, S. Gérard, "Completing EAST-ADL with MARTE for Enabling Scheduling Analysis for Automotive Applications", Embedded Real Time Software and Systems (ETS²2010), Toulouse, France, May 19th - 21st, 2010
8. MAST website Mast.unican.es
9. Zonghua Gu, Zhimin He: Real-Time Scheduling Techniques for Implementation Synthesis from Component-Based Software Models. CBSE 2005: 235-250
10. Liu, C. L. & Layland, J. W. "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment." Journal of the Association for Computing Machinery 20, 1 (January 1973): 40-61.
11. A. Mehiaoui, Sara Tucci-Piergiovanni, Jean-Philippe Babau. Optimizing the Deployment of Distributed Real-time Embedded Applications. The 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012). To appear.

5 Modeling Concepts for Optimization Support

Another objective of MAENAD is to extend the EAST-ADL language with support for multi-objective optimisation, including the definition of standard architectural patterns that can be used in optimisation to improve system characteristics like dependability and performance. This section first provides a brief overview of basic optimisation concepts before then describing EAST-ADL support for optimisation and detailing the various concepts introduced to the language to achieve this objective.

5.1 Overview of general optimisation concepts

Contemporary model-based systems analysis techniques (see section **Error! Reference source not found.**) allow a wealth of information to be obtained about a system. For example, dependability analysis can be used to both determine the probable causes of a system failure and also obtain an estimate of the probability of that failure occurring. Such information can be extremely valuable while designing a system and can be used to guide the future iterations of the design, e.g. to produce a more mature model in which the effects of a critical failure identified in earlier design models are mitigated or avoided.

This capability is enhanced due to the fact that many systems analysis techniques can now be semi-automated by software tools; this is especially true when such tools are compatible with the modelling language used to describe the system model, as the model can then be subjected to analysis directly. Automated analysis allows models to be rapidly evaluated according to a variety of different criteria, e.g. performance, safety, maintainability, cost etc. This allows designers to prototype and test out different design options as part of an iterative design process.

However, modern systems (particularly electronic ones) are typically sufficiently complex that only a small number of potential options can be investigated in this way, since it takes time for a designer to produce a new design variant, analyse it, and evaluate the resulting data. The set of all possible design variants is known as the **design space** or **search space**. This task is made harder when there are multiple design **objectives** - i.e. attributes of the system to be improved - which may conflict, e.g. the goal may be to maximise performance and safety while minimising cost as much as possible. This results in complex trade-offs which can be difficult to evaluate manually. Taken together, the act of searching a large potential design space to obtain a good solution that features a desirable balance of attributes is known as a **multi-objective optimisation problem**.

In multi-objective optimisation, automated algorithms are typically employed to search the design space according to various heuristics that aim to quickly find **solutions** - valid design variants - that offer optimal or near-optimal attributes without having to investigate all possible designs (a task which may be impossible to complete in a reasonable time even for modern computers and software). Different algorithms exist, but typically they operate in an iterative manner, evaluating a given set of solutions to determine which are the best and keeping them while discarding the rest, hopefully leading to a new iteration with a new set of superior solutions.

It is important to note that in a multi-objective optimisation problem, there is normally not one **optimum** solution since the different objectives may be mutually exclusive, i.e. to improve one objective attribute, it may be necessary to sacrifice another objective attribute. Therefore, multi-

objective optimisation algorithms typically produce a set of 'optimal' solutions that feature a range of attributes that balance the different objectives in different ways. These are known as the **Pareto solutions** and are based on the concept of **dominance**; a solution is a Pareto solution if it is better than any other solution in at least one objective attribute and no worse than any other solution in the others, in which case it is said to dominate the other solutions. Thus a Pareto set can include solutions with one objective maximised and the others minimised, or solutions with all objectives more evenly balanced, and all can be described as 'optimal'. This concept can be seen in the figure below:

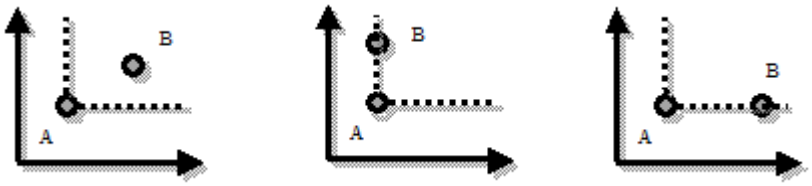


Figure 3. Pareto dominance

Here, solution A is always better in at least one axis (representing a particular optimisation objective, e.g. safety or performance or cost). For example, assuming the X axis is unavailability and the Y axis is cost, then in the middle graph, A has the same unavailability as B but is cheaper, while in the right-most graph, A is no cheaper but has a significantly smaller unavailability. A is therefore said to dominate B.

When the different solutions are plotted on a graph, the dominant Pareto solutions form a curve known as the **Pareto frontier**, as can be seen below:

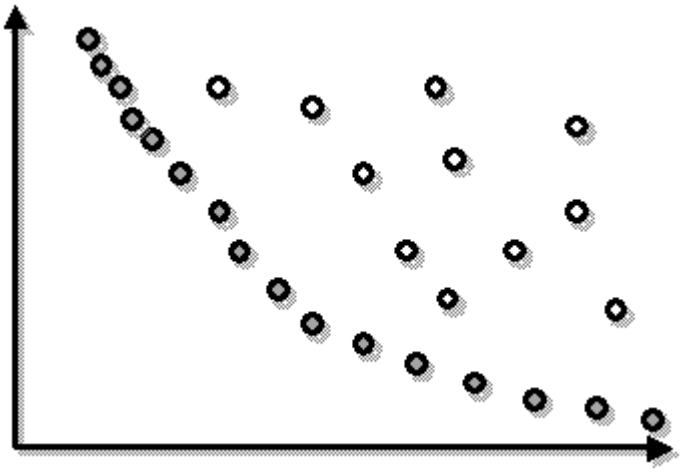


Figure 4: Pareto frontier

Here the shaded dots are Pareto solutions that dominate the non-shaded dots.

One of the key concepts in optimisation of this type, particularly when trying to automate it, is the ability to define and explore the design space. Different design **variants** can be created by altering

a particular aspect of a system; for example, dependability may be improved by replicating a critical component to achieve redundancy, but cost may also be increased as a result. The design space therefore contains a set of models that feature all possible design variants. Defining this design space means highlighting areas in the system model where other variants are possible. This can be achieved through the use of **variability** mechanisms, e.g. to indicate that there are different possible implementations of a given subsystem, each with different characteristics (so one may have better performance but cost more, another may be cheaper but may suffer in performance). Typically, variants can be defined in one of three ways:

1. **Substitution** - A given component or subsystem is substituted for another that has different objective attributes (e.g. safety, cost, performance). A substitute must be **functionally equivalent** to be substitutable, i.e. it must perform the same task (thus a substitute can never have less functionality than the element it replaces, only either the same or more functionality). This does not mean that the function must be carried out in the same way, however; for example, an electronic braking subsystem may be replaced by a hydraulic version. The different substitutable options may be thought of different **implementations** of a given element/subsystem.
2. **Replication** - A given component or model element may be duplicated to achieve redundancy. Such replicants are typically connected in a parallel configuration so that failure of one replicant does not lead to failure of the whole subsystem.
3. **Both** - More complex optimisation is possible when the design space features a combination of both substitution and replication. Thus, for example, a design variant may replace a given element with two parallel elements, one of which may be the same as the original, and the other of which is substituted for a different implementation.

These different approaches to creating design variation are illustrated in the figure below:

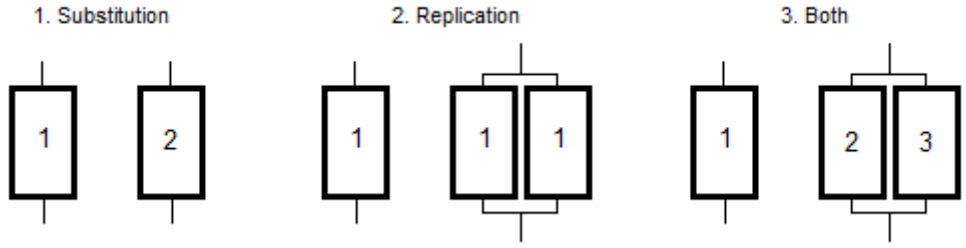


Figure 5. Substitution and Replication strategies

Note that in more complex approaches, substitution can also be hierarchical; for example, a single component may be substituted for another component or may be substituted for a subsystem with an entire sub-architecture, thus allowing for more complex replication strategies (like voters, monitors, or other parallelisms). Such configurations can often be stored in a library and reused.

To be used in the optimisation process, it must be possible to represent these design variants using an **encoding** strategy. A simple encoding may be to merely represent each component implementation with a number, thus 1111 could be a simple model in which no substitution has taken place, and 1112 a model in which the last component has been substituted for a second implementation/version. However, to allow more complex design spaces to be explored (e.g.

featuring substitution of subsystems or more sophisticated replication strategies), the encodings become correspondingly more complex. One option is to use a hierarchical encoding such as a **tree encoding**, which echoes the hierarchical structure of the design model.

Once the design space is defined and can be explored using algorithms and encodings, the algorithm must be able to **evaluate** a given design variant according to the optimisation objectives; this requires the use of system analysis techniques. These can range in complexity from simple summations of component costs to more elaborate timing and safety analyses. The algorithm needs to be able to analyse a design variant for each objective attribute (e.g. cost, performance) to allow it to compare that design variant against other variants, and thus determine whether it is a dominant or optimal design and decide whether it should be kept for future optimisation iterations or discarded.

The optimisation process itself can often continue almost indefinitely, assuming the design space is sufficiently large; therefore in general it is set to run for a given number of iterations or a given time period, after which it returns the best solutions it has discovered so far. These can then be examined by the designer and used as the basis for the next iteration of the overall design process.

Thus to be able to perform multi-objective optimisation of a system design, it must be possible to represent and make use of the above concepts; in particular, it has to be possible to:

- Create the design space by defining different possible design variants through the use of replication and substitution.
- The possibility to manually define design variants explicitly.
- Allow the optimisation algorithm to explore the design space by representing the variants as encodings.
- Enable evaluation of the different variants according to the optimisation objectives by means of model-based analysis techniques.
- The product variability space and the take rate of each combination shall be included in the optimization criterion.

5.2 Current EAST-ADL support for optimisation concepts

The main requirement for language support of optimisation is the ability to define the design space. To a significant degree, this was achieved by making use of pre-existing EAST-ADL semantics. In particular, EAST-ADL features some sophisticated variability mechanisms; while these were originally designed to be able to represent product line variability, they can also be used to represent different design variants in an optimisation context, as long as the two types of variability are kept distinct.

Variability in EAST-ADL is based - at the most abstract level - upon the concept of a **feature**, which can be present in some product lines and absent in others. EAST-ADL provides a range of variability management features to allow more complex configurations of features to be represented, including dependencies between features (possibly hierarchical) and duplication of

features. Variability of features is primarily defined using **feature models** and **configuration links**, which connect feature models and define the dependencies between them. Variability management concepts in EAST-ADL are more fully described in the following section of this document, but in the context of optimisation, they provide a useful mechanism for which to define different optimisation design variants by means of substituting one design element for another element or hierarchy of elements. Variability management is also designed to be able to ensure substitutability of features when required, which is vital when employed in an optimisation context.

The evaluation stage of optimisation consists of a combination of analysis techniques designed to analyse the different optimisation objective attributes. The use of model-based system analysis techniques is described in the preceding section and considerable work has already been done in developing concepts in EAST-ADL to support various analysis techniques, including behavioural analysis, performance and timing analyses, and safety and dependability analyses.

5.3 Discussion

The primary challenge in meeting the optimisation objective in MAENAD was to combine and link existing concepts, such as variability and the various analysis techniques, into an overall optimisation process. This meant extending and/or refining the variability mechanisms to allow them to support the definition of optimisation search spaces in addition to their primary role of modelling product line variability and enabling the analysis techniques to analyse these design variants. Most of this was supported via a tool framework known as the 'optimisation architecture' (for more information on this, see D3.2.1), but the overall process is described below.

5.3.1 Defining the design space

To achieve optimisation, an EAST-ADL design model - usually at the analysis and/or design levels, since sufficient data for evaluation must also be present - must be created that contains a number of different variability points and that allows for different design variants to be explored. Existing variability mechanisms proved to be largely sufficient for this task. One of the primary concerns for optimisation is that the designer must still ensure that the variants are substitutable, i.e. that one variant is functionally equivalent to another; this is not always the case in normal variability management, where e.g. a feature may be present in one product line but absent in another. Thus is it still important to distinguish between variations that represent such optional functionality (a.k.a. “product line variability”, that does not define an optimization choice) and variations represent alternative but functionally equivalent realizations of a feature (a.k.a. “design space variability”, that defines the optimization space). This was achieved through the use of the binding time attribute.

Furthermore, so variants can be automatically explored by the optimisation algorithm without unnecessary complexity, they need to be encoded in a hierarchical, structured way. In practice one major difficulty in automatic optimisation is the connections between substituted and/or replicated components, e.g. one implementation may not have the same interface (i.e. number and type of inputs & outputs) as another, and this has to be resolved for automatic optimisation to produce valid and sensible results. In EAST-ADL, the variability mehanisms already provide this capability via the Feature Model (see next section), which serves as a kind of hierarchical index of the different variability points and relationships between them. The feature model (or feature tree) is similar to the hierarchical encoding envisaged for the optimisation and broadly speaking can be reused for this purpose.

Finally, although the original design model used as input to the optimisation process should contain the necessary variability to define the design space (and product space with take rates), the design solutions identified by the optimisation need to have had this variability resolved such that the solution represents a single possible configuration of the system. This is similar to the concept of binding in variability and allows the optimisation process to be able to subject the resolved models to analysis. Resolving the variability in the model ideally needs to ensure that each resulting product variant is still a valid model and thus must still contain any necessary functionality and meet any potential requirements and constraints, e.g. safety constraints and requirements (like ASILs) in the safety domain, timing requirements in the performance domain etc. However, the optimisation algorithm allows for the imposition of penalties if necessary, allowing invalid candidates to be examined as stepping stones on the road to more optimal candidates but without allowing them to be considered as true results.

5.3.2 Evaluating the designs

To be used effectively in optimisation, each design variant has to be evaluated according to each of the objective attributes being optimized (there is a need to specify whether on objective is to be maximized, minimized and/or if it must satisfy a requirement to be “within bounds”; there is a need to identify which constraints are included in the optimization, which takes place through the VVCase construct). Thus the original design model has to contain all the different attribute data necessary to describe each variant. Note that this does not necessarily mean the model should itself contain all possible outcomes, but it provides enough information (through the use of variability and things like the error model etc) for the analysis techniques to determine the final attributes and thus allow for evaluation to take place. For example, it is infeasible for a single design model with, say, 10,000 variants to provide an estimate for unavailability for each variant within the model itself; instead, it should provide unavailability for each design option (i.e. for all possible component/function implementations), so that the safety analysis techniques can use this raw information to arrive at an estimate of the unavailability for that design variant as a whole.

Once evaluated according to each of the criteria, the optimisation algorithm can both determine its dominance and decide whether it should be retained as an optimal solution or not. If it is retained as a solution, the designer can also see what the attributes of the solution are. Note that this does not mean that a new design model gets created with all variability resolved; only the configuration options are recorded (together with the analysis results), and if a particular candidate needs to be identified, it can be generated on demand by resolving the variability in the original model according to the configuration options — in the form of the feature model (i.e., the encoding) — which are recorded for each retained candidate. Thus for example, a simple design model with only one function that can be implemented in one of three ways may produce three solutions, each of which representing a different trade off. Each solution records enough information for the designer to be able to see its overall objective attributes (e.g. timing, safety, cost) and if required, the designer can produce an actual EAST-ADL model by configuring the original model according to the encoding of the given design solution (in this example, by telling the tool which implementation of the function was chosen in each case).

5.3.3 Developing an optimisation algorithm

Although there are many different optimisation algorithms, each with various strengths and weaknesses, in MAENAD we focused on the use of particular forms of genetic algorithms to

perform the multi-objective optimisation, following the HiP-HOPS optimisation technique. HiP-HOPS uses genetic algorithms to support optimisation with different discrete objectives (compared to some other techniques that merge objectives into a single evaluation figure, for instance) and thus create a set of Pareto optimal solutions.

However, even for a given algorithm, there are many possible tweaks and variations available to enhance and optimise its performance and allow it to more efficiently explore the search space and obtain good solutions. Most optimisation algorithms have a variety of parameters that govern their usage, e.g. for genetic algorithms, the number of generations (i.e. iterations) and size of each population (i.e. set of solutions) can be tweaked, as can the rules governing the exploration of new solutions and the rules governing which solutions are kept. This work is ongoing, but in practice it may be that the optimisation parameters need to be tailored specifically to each model being optimised to produce the best results.

5.3.4 Language Concepts and Examples

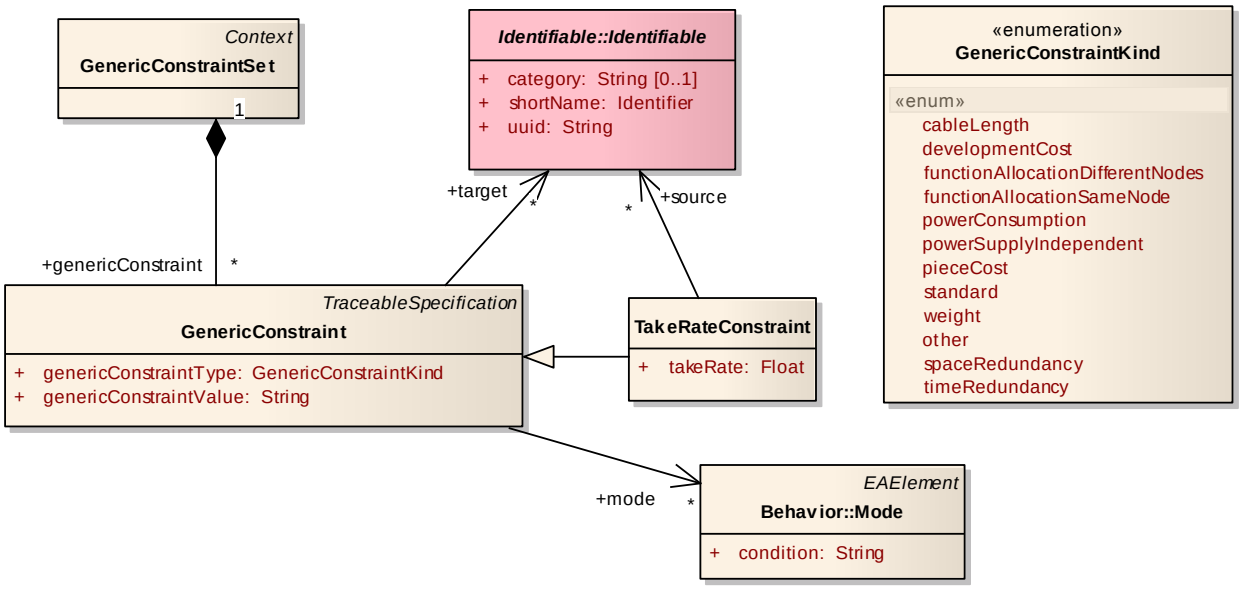


Figure 17. Constraints used for optimization

The number of components produced and their cost is a critical parameter for optimization. The total cost is development cost for each component type plus the sum of piececost multipled by piece count summed over all component types.

To know the piece count, it is necessary to define the absolute or relative number of elements in the feature tree or in the artifact model. A constraint solver can compute the number of elements of any given component based on such constraints, and it can also warn if the model is underspecified or inconsistent.

In the example in Figure 19, 20000 vehicles are produced. Since 16% of all vehicles go to the US market, and 50% of those have trim level Prime, it is possible to deduce that 1600 such vehicles with will be produced. 10% of all vehicles will be Plus, so these represent another 2000 vehicles. Both trim levels have ABSPlus which means 3800 PlusECUs will be produced and thus 16200 Standard ECUs.

Having established the number of components produced, it is possible to take this into account during optimization: Solutions with good price and performance for high volume products are favored before solutions that only benefit low volume products.

For example, if a high-volume vehicle needs an advanced component, while the low volume vehicle can do with a simpler component, it may well be better to equip all vehicles with the advanced component. This may also open up for after-sales revenue by selling upgraded functionality that relies on the better ECU. Figure 18 shows that the eliminated fixed cost for the low-end ECU compensates for the higher piece cost. This is consistent with Figure 19, as there is no configuration decision that states whether standard or plusECU shall be used. In the takerate model, it is undefined whether standardECU or PlusECU is used, unless the ABSPlus is selected in which case PlusECU is mandatory.

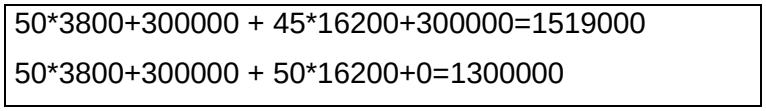


Figure 18. Total Product Line cost for same and different ECU

The assumption is that the stated rate constraints are part of the same GenericConstraintSet and thus consistent. Further, the semantics assumption is that the root element of each product feature tree represent the all vehicles.

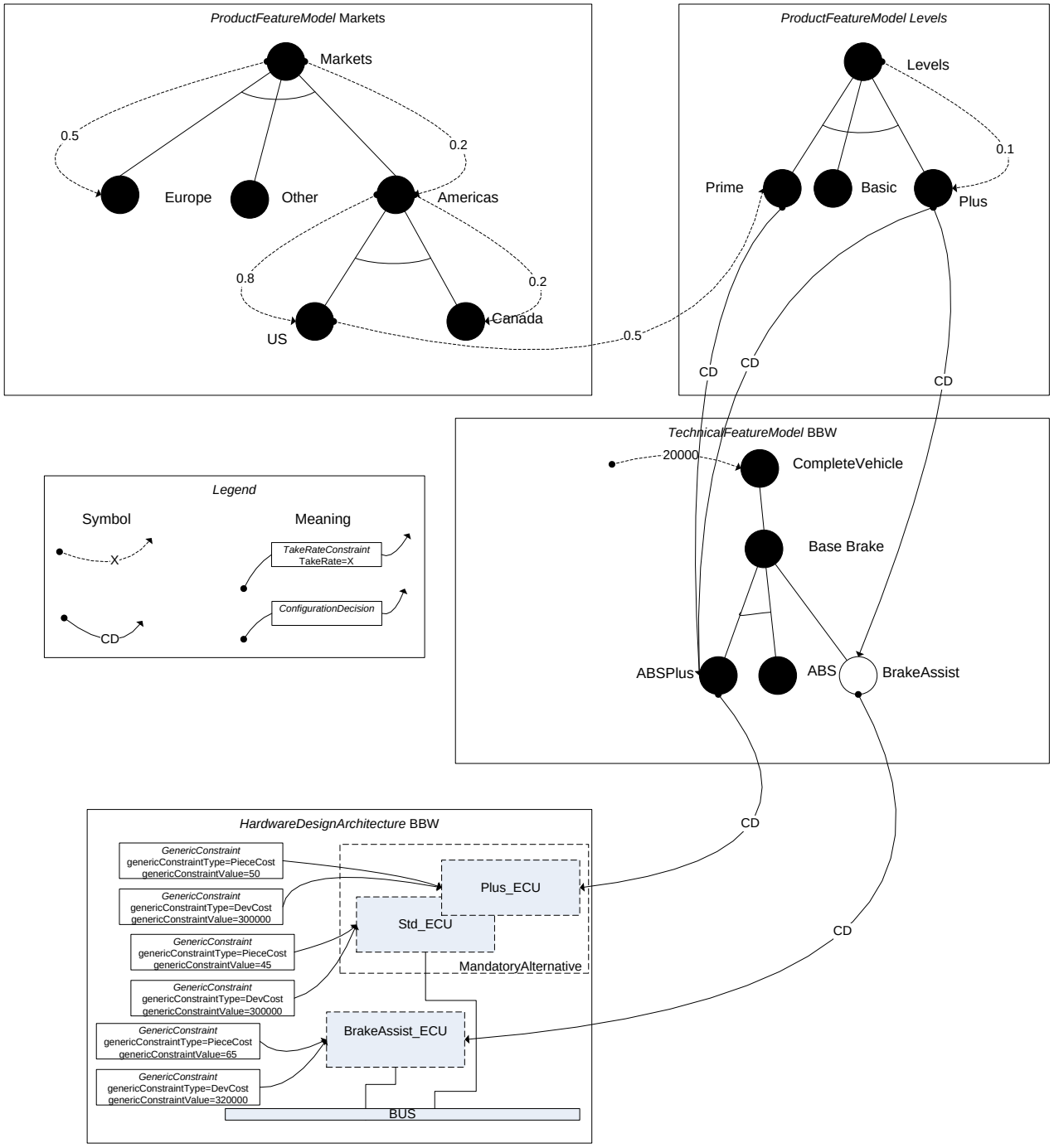


Figure 19. Feature models and constraints to consider in optimization

One way to resolve the take rate constraints is to translate it to a constraint programming problem and use a suitable solver. Figure 20 shows how a constraint programme in Prolog may look. It also contains a minimization criteria for cost.

```
% Prolog CLP(FD) pseudo-code

:- use_module(library(clpfd)).

% The feature tree as constraints

feature_tree(TotVehicles,Cost,Variables):-
        % All variables should be greater than zero
        TotVehicles#>=0,
        VehiclesOther#>=0,
        VehiclesEurope#>=0,
        VehiclesAmerica#>=0,
        VehiclesUS#>=0,
        VehiclesCanada#>=0,
        Prime#>=0,
        Basic#>=0,
        Plus#>=0,
        BaseBrake#>=0,
        ABSPlus#>=0,
        ABS#>=0,
        BrakeAssist#>=0,
        PlusECU#>=0,
        StdECU#>=0,
        BrakeAssistECU#>=0,
        PlusECUYesNo in 0..1,
        StdECUYesNo in 0..1,
        BrakeAssistECUYesNo in 0..1,
        Cost#>=0,

        % Constraints from Markets
        VehiclesEurope#=TotVehicles*0.5,
        VehiclesAmericas#=TotVehicles*0.2,
        TotVehicles#=VehiclesEurope+VehiclesAmericas+VehiclesOther,

        VehiclesUS#=VehiclesAmerica*0.8,
        VehiclesCanada#=VehiclesAmerica*0.2,
        VehiclesAmerica#=VehiclesUS+VehiclesCanada, % redundant constraint

        % Constraints from Levels
        TotVehicles#=Prime+Basic+Plus,
        Prime#=0.5*VehiclesUS,
        Plus#=0.1*TotVehicles,

        % Constraints from BBW feature tree
        BaseBrake#=TotVehicles,
        BaseBrake#=ABSPlus+ABS,
        ABSPlus#=Prime+Plus,
        BrakeAssist#=Plus,

        %Constraints from BBW architecture design
        StdECU#<=ABS
        PlusECU#>=ABSPlus
        PlusECU#<=ABSPlus+ABS
        StdECU#+PlusECU#=ABS+ABSPLus
        BrakeAssisECU#=BrakeAssist,

        % Objective function (single objective)
        PlusECU#>0 #<=> PlusECUYesNo,
        StdECU#>0 #<=> StdECUYesNo,
        BrakeAssistECU#>0 #<=> BrakeAssistECUYesNo,

        Cost#=PlusECUYesNo*300000+PlusECU*50+StdECUYesNo*300000+StdECU*45+BrakeAssistECUYesNo*320000+Brak
eAssistECU*65,

        Variables=[VehiclesEurope,VehiclesAmerica,VehiclesUS,VehiclesCanada,Prime,Basic,Plus,BaseBrake,ABSPlus,ABS,Bra
keAssist,PlusECU,StdECU,BrakeAssistECU].

% Find minimum cost solution for given number of vehicles

find_cost(Cost,Variables):-
        feature_tree(20000,Cost,Variables),
        labeling([minimize(Cost)],Variables).

% Find maximum number of vehicles for maximum given cost

find_vehicles(TotVehicles,Variables):-
        Cost#=<1600000,
        feature_tree(TotVehicles,Cost,Variables),
        labeling([maximize(TotVehicles)],Variables).
```

Figure 20. Prolog code corresponding to the TakeRate constraints

5.3.5 Summary

In summary, to achieve the objective of supporting automatic multi-objective optimisation of EAST-ADL models, the following steps were taken:

- Definition of the design space:
  - Use existing variability mechanisms where possible
  - Ensuring substitutability of one design variant for another
  - Developed an encoding strategy to allow the design space to be represented in the optimisation algorithm
  - Allowed the variability to be resolved/bound to a particular configuration to provide a design solution with a given set of objective attributes that can be evaluated
- Evaluation of the design variants:
  - Made use of existing developments towards system analysis techniques in EAST-ADL for the purposes of analysis, and started the development of new ones where existing analyses were not available
  - Enable the results can be used by the optimisation algorithm to determine dominance
  - Allow the designer to view the design candidates/results by configuring the original model according to the feature model/encoding
- Development of multi-objective optimisation heuristics to allow the algorithm to efficiently explore the design space and rapidly arrive at suitable Pareto optimal solutions
  - Refined the algorithm to take into account design space and evaluation requirements
  - Experimented with the different algorithm parameters to improve efficiency (although there may not be a one-size-fits-all solution to this)
  - Developed a initial methodology governing the use of optimisation in an EAST-ADL model

This is meant to meet the following project requirements:

- **DOW#0006 O3: Develop capabilities for design optimization**

  The concepts are now in place, although tool support is still being developed (see optimisation architecture in D3.2.1).

- **DOW#0014 O3-1**: **Extension of EAST-ADL2 language with semantics to support multi-objective optimization for product lines**
- By making use of the existing variability concepts in EAST-ADL, we are able to support both optimisation and product line variability as part of the model/design space definition. However, it requires further tool support to allow this kind of optimisation to actually take place (again, see D3.2.1).
- **DOW#0018 O4-3**: **Evaluation of different optimization approaches**
- A comparison of different algorithms is included in D3.2.1.
- **DOW#0015 O3-2: Definition of a library of standard architectural patterns & DOW#2000 Architectural Patterns**

- These are still ongoing and deal with the development of a series of architectural substitution and replication patterns that can be applied by the optimisation algorithm to achieve a better exploration of the design space.

6 Modeling Concepts for Variability Management

Variability management is not a primary objective of the MAENAD project. It is not explicitly mentioned in the list of project objectives in the MAENAD description of work (pages 9, 10), because at the end of ATESST2 the variability management in EAST-ADL was considered fairly complete. In this chapter we explain why variability has still received significant attention in the MAENAD project and explain the result of the variability-related project activities.

6.1 The Role of Variability Management in MAENAD

Despite not being a primary MAENAD objective, variability is still an important topic for MAENAD for these reasons:

1. Variability management is an important part of the EAST-ADL and the overall consolidation and maintenance of EAST-ADL is an aim in MAENAD. Therefore, also the maintenance and consolidation of the variability-related concepts is within the project's scope.
2. In automotive industry, most development projects are dealing not with a single system but a whole family of similar but distinct products. The resulting variability in system development poses a significant challenge to most of the primary objectives of MAENAD. Therefore the solutions devised in MAENAD to tackle the project's primary objectives also have to take into account variability.
   For example, the entities for hazard analysis in EAST-ADL also have to be feasible for analyzing variant-rich systems.
3. Variability management concepts in EAST-ADL can be helpful for achieving some of the primary objectives of MAENAD, even though these objectives may not be primarily concerned with variability management. For example, language concepts for defining design variations may also be used to define the optimization space for design space exploration in the context of system optimization (cf. Section 5.3.1).

As a consequence, variability has received special attention in MAENAD, but the work on variability management was mainly be driven and motivated by the other, primary objectives and the demonstrators (e.g. required refinements of the variability concepts that were identified during demonstrator modeling).

6.2 Topics Related to Variability

The uses of variability in the context of the primary project objectives is documented elsewhere in this deliverable (esp. in the chapter on optimization). However, some general consolidation issues and refinements related to variability are described in the following:

- Dependencies between variants across the containment hierarchy in FAA, FDA, etc.
- Evaluation of the support for storing system configurations in EAST-ADL models.
- Improved documentation of the overall variability management technique in EAST-ADL (in particular the Multi-Level concept).
- Alignment with AR variability management (postponed to end of 2011 when next version of AUTOSAR is expected to be available).
- "Feature Tree Semantics" (see below).

The last item in the above list has been addressed first; the remainder of this chapter summarizes the current status of this discussion as of June 2012.

6.3 "Feature Tree Semantics"

The discussion of this topic already started during the consolidation phase at the end of the ATESST2 project, but no final conclusion was taken at that time. It is important to note that despite the name "feature tree semantics", the topic does not deal with the core of the variability-related semantics of EAST-ADL feature modeling itself. Instead, the focus is on the precise semantics of certain Relationships between Features and other entities in the language, in particular Requirements and FunctionPrototypes.

6.3.1 Overview

This issue deals with the precise semantics of the two EAST-ADL relationships **Satisfy** and **Realization** in case there is a Feature on one side. Refer to the next two figures for details on how these relationships are defined in the EAST-ADL domain model.

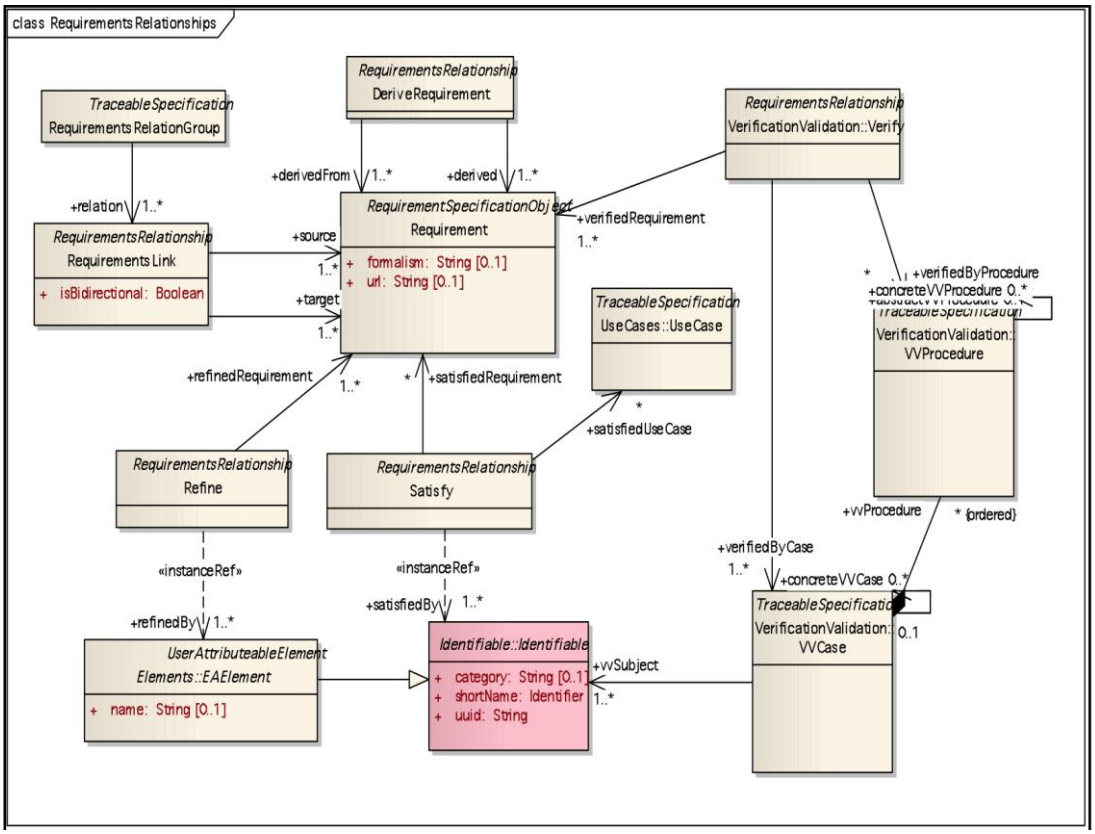


Figure 6. Diagram "Requirements Relationships" from EAST-ADL domain model.

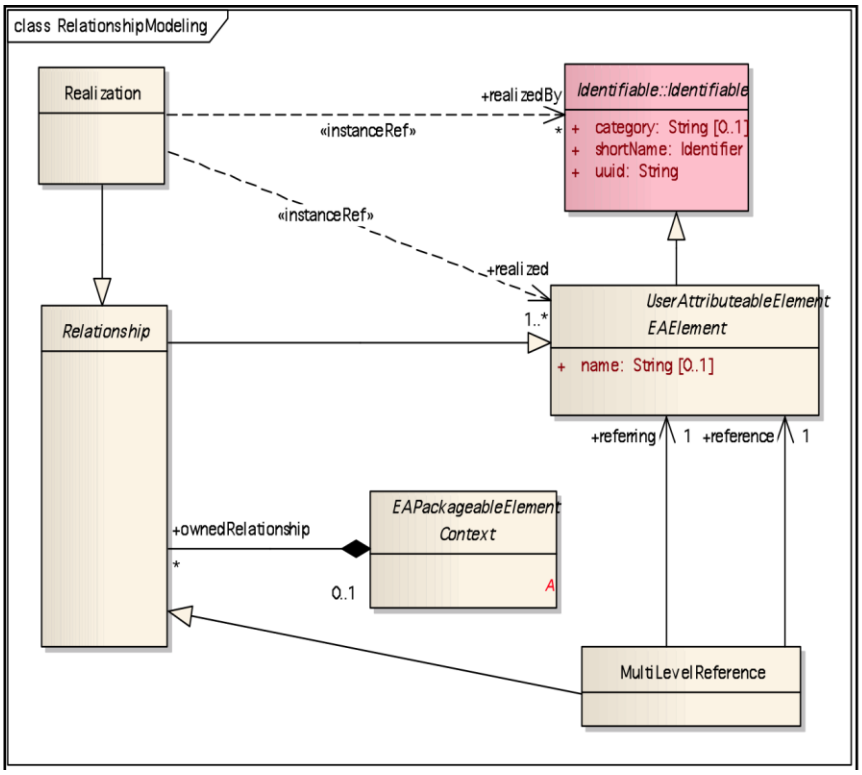


Figure 7. Diagram "Relationship Modeling" from EAST-ADL domain model.

In the remainder of this section we will focus on these cases:

- A Satisfy relationship between a Requirement (role name "satisfiedRequirement") and a Feature (role name "satisfiedBy").
- A Realization relationship between a FunctionPrototype (role name "realizedBy") and a Feature (role name "realized").

The overall semantics of these two relationships – when leaving aside the details – is quite clear in the above cases:

- Feature F → Satisfy → Requirement R:
  Feature F will heed Requirement R, i.e. it is responsible for making sure that Requirement R is fulfilled.
- FunctionPrototype FP → Realization → Feature F:
  FunctionPrototype FP provides an implementation of Feature F (in case of a DesignFunctionPrototype).

However, when inspecting these relationships more closely, the semantics becomes more intricate, especially when taking into consideration the parent/child relations between features. This is discussed in the next section.

6.3.2 Problem Description

In this section we try to highlight the difficulties in the semantics of the two aforementioned relationships by listing a number of questions in each case.

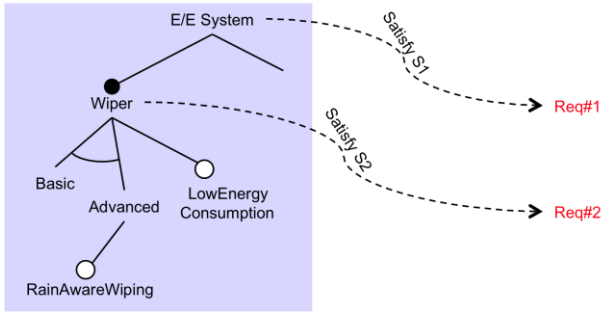


Figure 8. Sample Satisfy relationships.

Some considerations on the fact that Wiper satisfies Req#2, as defined by S2 (in the above figure):

- Does S2 mean that only the wiper-system (i.e. feature Wiper) has to heed requirement Req#2 and the climate-control system (not shown in figure) need not heed Req#2?
- What does S2 imply for predecessors (i.e. parent features, grand-parents, and so on …) and successors (i.e. child features, grand-children, etc.) of feature Wiper? For example, does S2 imply that also feature Advanced satisfies Req#2?

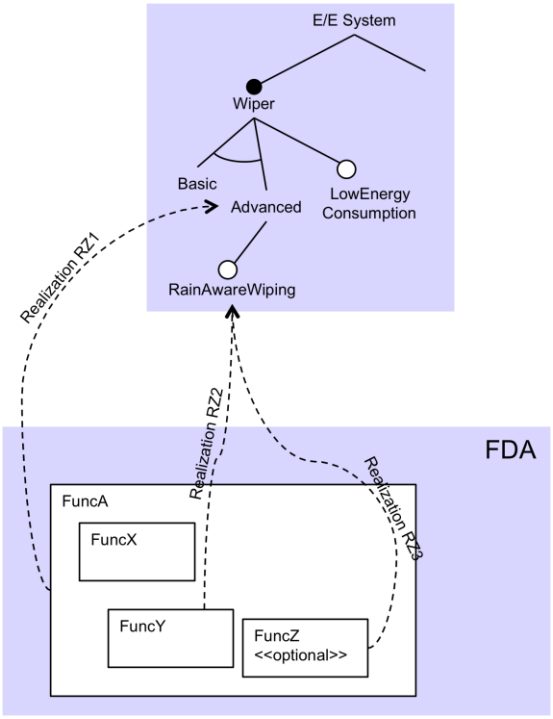


Figure 9. Sample Realization relationships.

Some questions regarding Realization relationships RZ1, RZ2 and RZ3 in the above figure:

- Does RZ2 mean that FuncX does **not** realize RainAwareWiping (i.e. is not at all involved in realizing RainAwareWiping)?
- What does it mean that both FuncY and FuncZ realize feature RainAwareWiping?

- What does RZ1 imply with respect to the realization of features E/E-System, Wiper and RainAwareWiping through FuncA? For example, does RZ1 imply that FuncA also realizes feature Wiper? Does RZ1 imply that FuncA also realizes feature RainAwareWiping (if it is selected)?
- What does RZ1 imply with respect to FuncX? Does FuncX, as a subfunction of FuncA, also (partly) realize feature Advanced?

6.3.3 Tentative Solution

In this section we provide a tentative semantics definition that may be used as a basis for further investigation and refinement based on more detailed examples and the MAENAD demonstrator models.

Semantics for case "Feature F → Satisfy → Requirement R":

The Satisfy relationship between a feature and a requirement defines that this particular requirement applies to this feature and its successors, i.e. the functionality and/or non-functional properties represented by the feature and its successors must collectively fulfill the requirement.

Points to note:

- Predecessors of a feature are its parent, grand-parent, and so on. Successors of a feature are its child features, grand-children, etc.
- This might mean that the feature provides some functionality that is required by a functional requirement or that the feature must comply with some constraint, restriction, etc. imposed by the requirement.
- Effect of parent/child relations in the feature tree (still referring to case "Feature F → Satisfy → Requirement R"):
  (a) When looking at the particular requirement R, then this requirement applies to F, the child features of F, the grand children of F, etc.
  (b) when looking at a particular feature F, then all requirements of its parent, those of its grand parent, etc. apply to F.
- The term "collectively" above is still being discussed at time of writing (MS3). Some project partners tend to this view: "Each successor must fulfill the requirement. It may be implemented in different ways but each child feature is individually responsible."

Semantics for case "FunctionPrototype FP → Realization → Feature F":

„The Realization relationship denotes the primary responsibility of an architectural element for realizing the functionality and/or non-functional properties represented by a feature. Several architectural elements defined to realize a single feature are collectively responsible for the realization."

Points to note:

- If several FunctionPrototypes realize the same feature they are all, collectively responsible for realizing the feature. No assumption is made how responsibilities are shared (equally or one function being more significant than the others) and which parts are realized by which function.

- The same applies to a higher-level function that contains subfunctions: the containing function and all its directly or indirectly contained subfunctions collectively realize the feature.
- The explicitly defined realizations (by way of Realization relationships) do not claim completeness in the sense that each and every contribution is explicitly defined. Otherwise also minor, very remote and indirect contributions would have to be defined with a Realization relationship.
  Instead, the Realization relationships define primary / major contributions to realization.

6.3.4 Further Steps

The initial, tentative definitions from the previous section should be evaluated and further refined based on concrete examples and the demonstrator models. Further refinement on this abstract, theoretical level would probably prove very difficult. Also the example in the SAFECOMP paper (one of the ATESST2 publications), where dependent functions are used as examples, can be used as a basis for further exploration. This focuses on the Satisfy relation as that one has a deeper impact on functionality definition on Vehicle level. In addition, the discussion on the system/environment model interface from Section 2.5 is to be considered in this context.

7 Language Consolidation Amendments

This section discusses language refinements related to general consolidation activities that are orthogonal to the specific project objectives. The language concepts that were refined are discussed below along with an explanation of modifications.

7.1 Overview

The motivation for these consolidation activities is very divers. In some cases mistakes had to be resolved and missing things had to be added, but in most cases the language changes were required in order to adapt the language to new requirements or to incorporate results from other research projects, for example in case of the timing-related concepts devised in the TIMMO2USE project.

However, despite the improvements described in this chapter, EAST-ADL has overall proved to be highly viable and already well-consolidated, thanks to the consolidation and refinement activities from the ATESST2 project. In fact, fewer changes were required than we had anticipated.

The amendments are related to:

- Type definition, values and expressions
- The Inheritance structure
- Environment Model
- HardwareArchitecture
- Semantics of Realization
- TADL2
- Improved Documentation

A main, overall consolidation activity had been conducted during the months from Dec 2011 to Feb 2012. This was based on an extensive review of the domain model and its documentation and a coordinated process of change request elicitation and resolution. All MAENAD partners were invited to review the EAST-ADL domain model and to provide feedback and change requests. In addition, several smaller, more specialized consolidation activities have been conducted since then, each focused on a particular topic.

At time of writing, some consolidation activities are still ongoing. It is planned to finalize a version 2.1.11 of EAST-ADL by October / November 2012 that will remain stable and will be used for a longer period of time. This is also of relevance for the tool implementations in the context of the upcoming EATOP Eclipse project.

7.2 Types and Values

The Datatype concept of EAST-ADL 2.1 requires further validation. Both the definition of Datatype and the use of Datatype as a type of various attributes in the language had to be revisited. The old Datatype packge had been devised at the end of the ATESST2 project but at that time there was no opportunity to evaluate these concepts thoroughly. Figure 21 shows the old metamodel of EADatatype.

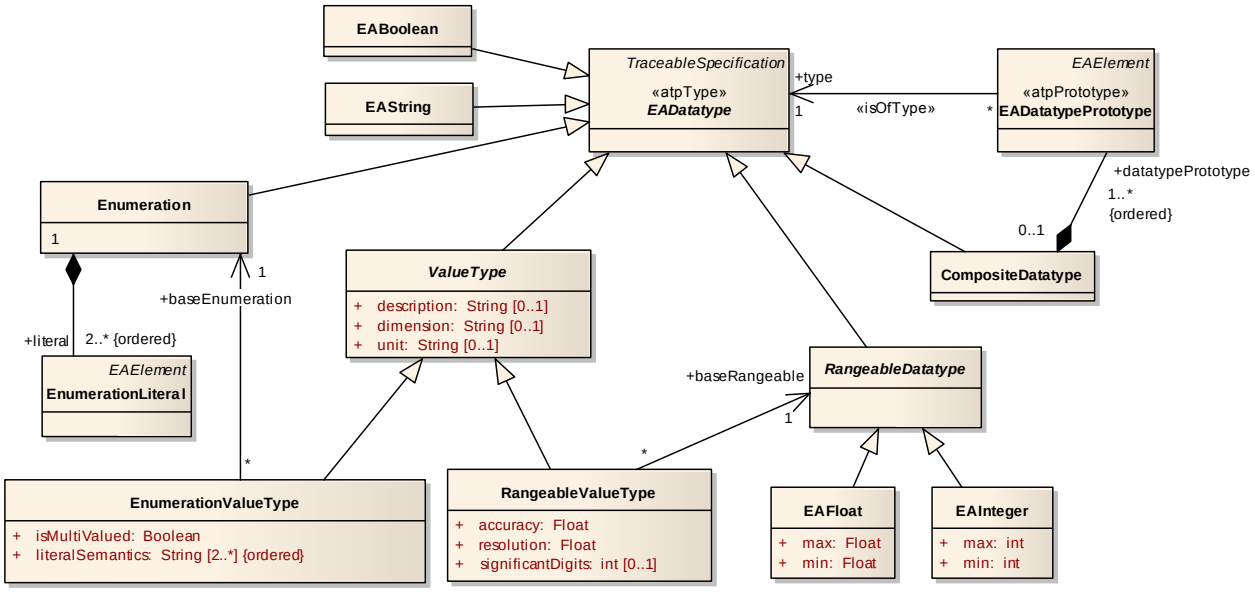


Figure 21. The EADatatype and related elements

There has not been a support for modeling values in the user model in a structural way. Inspired by UML and the MARTE Annex B, Value Specification Language a structure of EAValue in EAST-ADL is proposed.

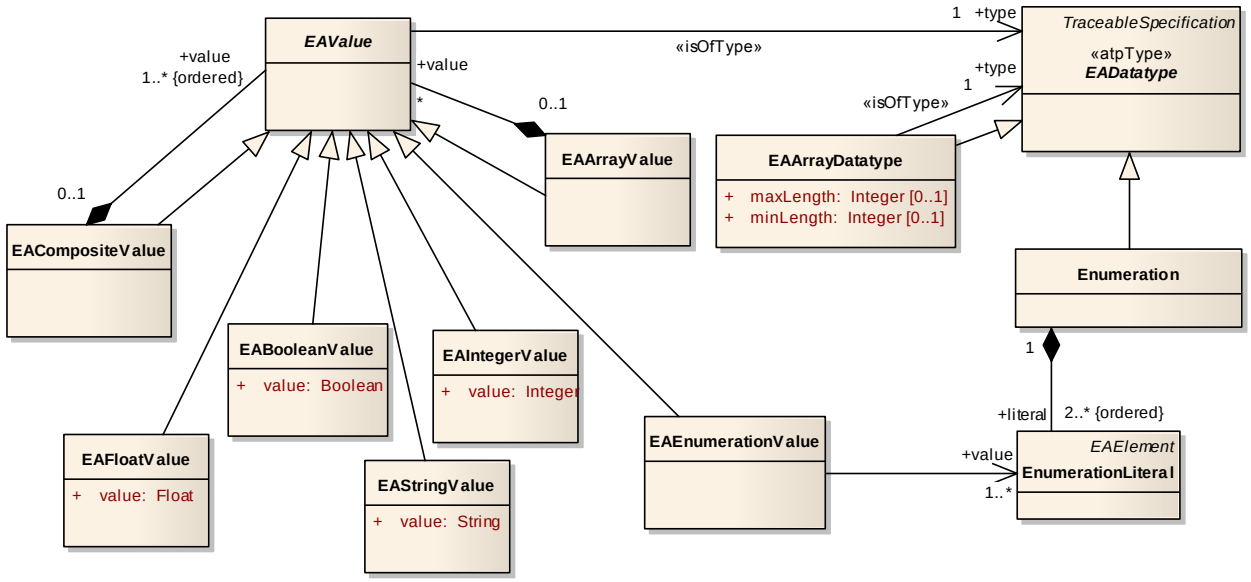


Figure 22. The new EAValue typed by EADatatype and concrete elements

In the current metamodel proposal the EAValue is used for:

- FunctionFlowPort defaultValue (this is a new concept)
- UserAttributeableElement uaValue (earlier this value was a UserAttributeValue, this concept can be removed)
- UserAttributeDefinition defaultValue (earlier this was a string attribute)

Eligible for using the EAValue are also:

- FaultFailure faultFailureValue (currently EADatatypePrototype is used)
- FeatureConfiguration value (this is a new concept). Also Feature featureParameter to use EADatatype instead of EADatatypePrototype?
- GenericConstraint value (currently this is a string attribute) the type of the value is a GenericConstraintKind where some EADatatype is implied.

New concepts in MAENAD that also would benefit from EAValue are

- Behavior Description Annex
- Timing (TADL2)

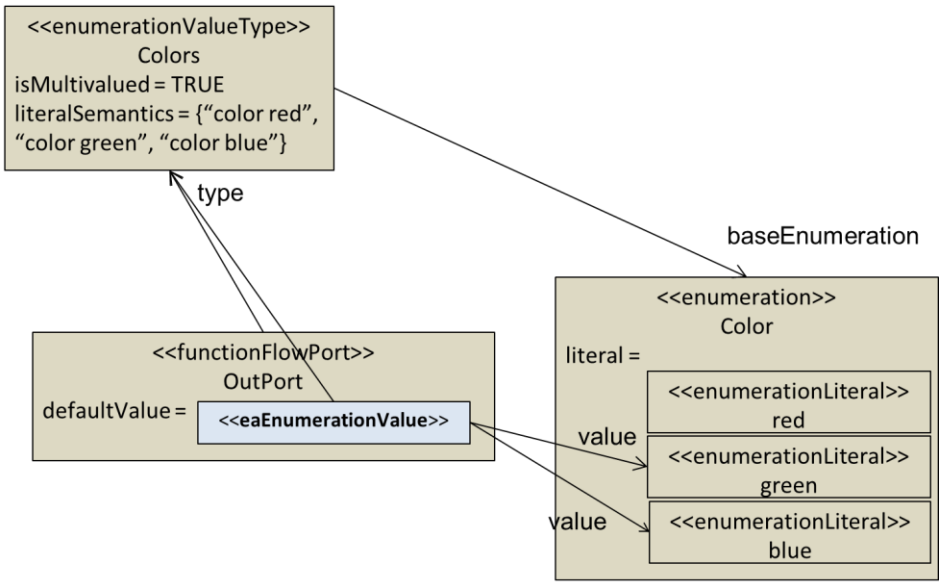
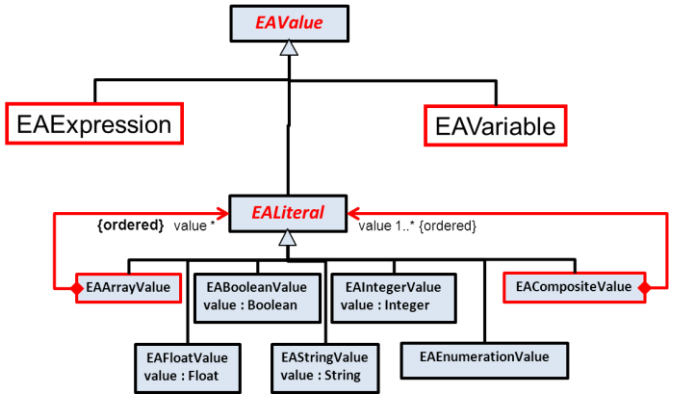


Figure 23. Example model where the EAEnumerationValue is used to model the default value of a port.

7.3 Expressions

Expressions have not been a part of EAST-ADL, but fit well in the EAValue framework. The timing-related extensions to EAST-ADL that were provided by the TIMMO project contained a solid basis for such an expression concept. The main effort in MAENAD was to integrate these expression in the EAST-ADL core. The below figure shows, how EAExpressions were integrated as special subclasses of EAValue.



7.4 Refinement of Inheritance structure

7.4.1 Inheritance from EAST-ADL Base Elements

Majority of the language elements are subtypes of a few base elements like EAElement, AllocationTarget, EAPackeableElement, Context and TraceableSpecification. The inherited attributes and associations needs to be assessed for some language concepts to ensure validity, see Figure 24.

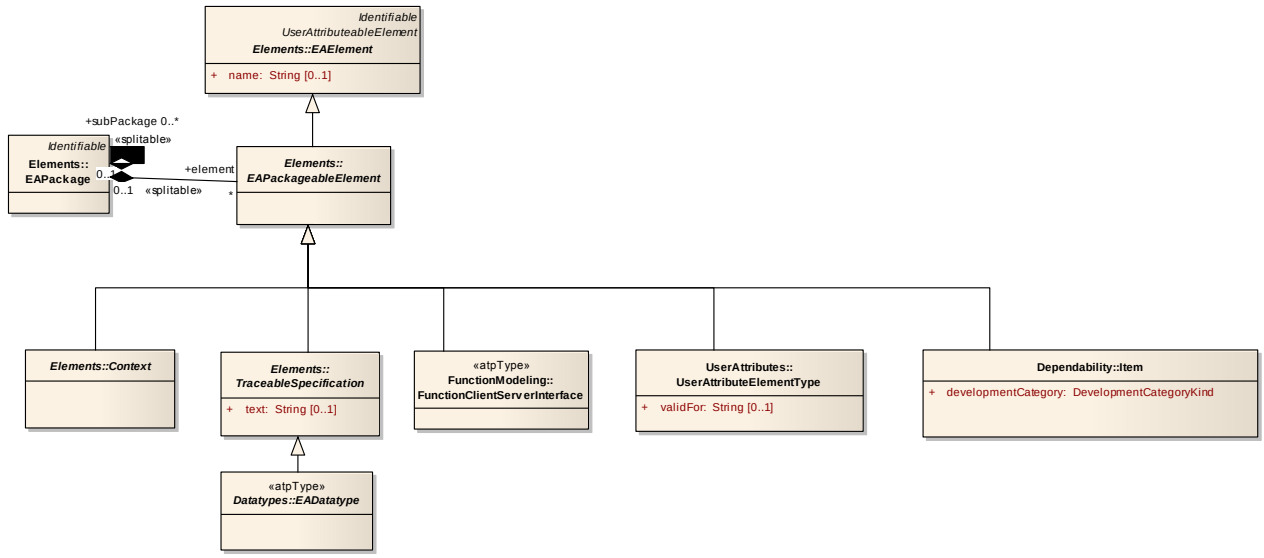


Figure 24. Inheritance structure of some selected elements

Figure 24 shows that FunctionClientServerInterface inherits directly from EAPackableElement. Instead it should inherit from TraceableSpecification like EADatatype.

Item should have the ability to own TraceableSpecifications and should thus inherit from Context.

UserAttributeElementType should inherit from TraceableSpecification like EADatatype.

7.4.2 Inheritance of FunctionType and related elements

Comparing with the metamodel structure of AUTOSAR introduced from version 4. There is an abstract structure of metaclasses like AtpType for elements previously only marked by stereotypes like <<atpType>>.

New additional abstract elements proposed for EAST-ADL to assist tool support of structural modeling are (and their specializations):

EAType

- ErrorModelType
- FunctionType
- HardwareComponentType

EAPrototype

- ErrorModelPrototype
- FunctionPrototype
- HardwareComponentPrototype

EAPort

- FaultFailurePort
- FunctionPort
- HardwarePin

EAConnector

- FaultFailurePropagationLink
- FunctionConnector
- HardwareConnector

7.5 Environment Model

The EnvironmentModel is currently a function hierarchy that is separated from the SystemModel and linked to the FAA and FDA through ClampConnectors, see Figure 25.

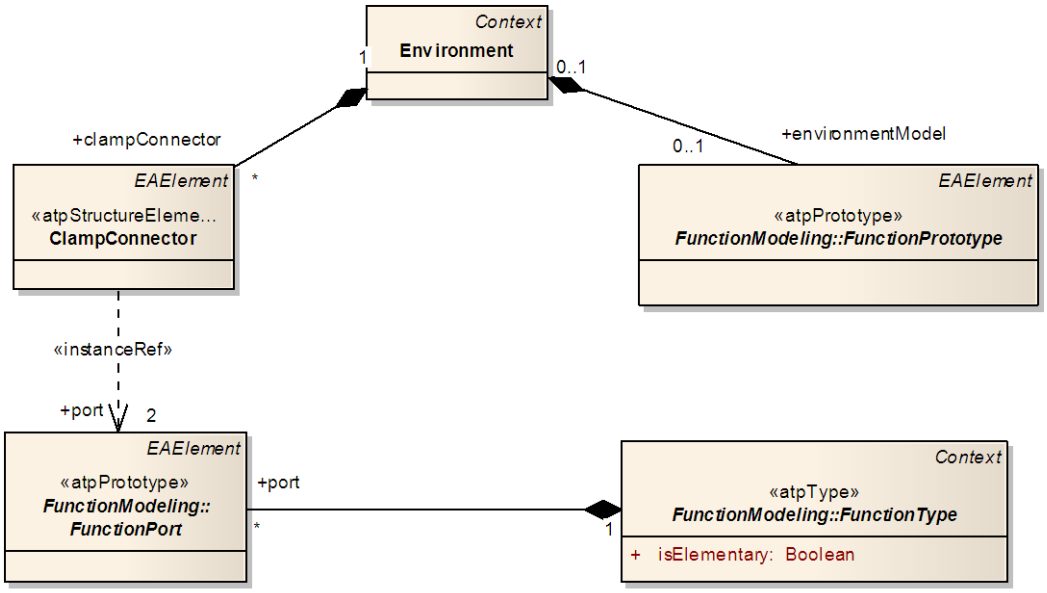


Figure 25. The EnvironmentModel and related elements

7.6 HardwareArchitecture

The HardwareArchitecture need to be refined to support FEV needs. In particular, the specification of electrical I/O need to be added to the metamodel. Currently the metamodel of HardwareArchitecture is as specified in Figure 26. Once this metamodel is entered as a metamodel (see Figure 27) in a modeling tool the language can be applied as shown in Figure 28 - albeit now is not possible to specify electric I/Os.

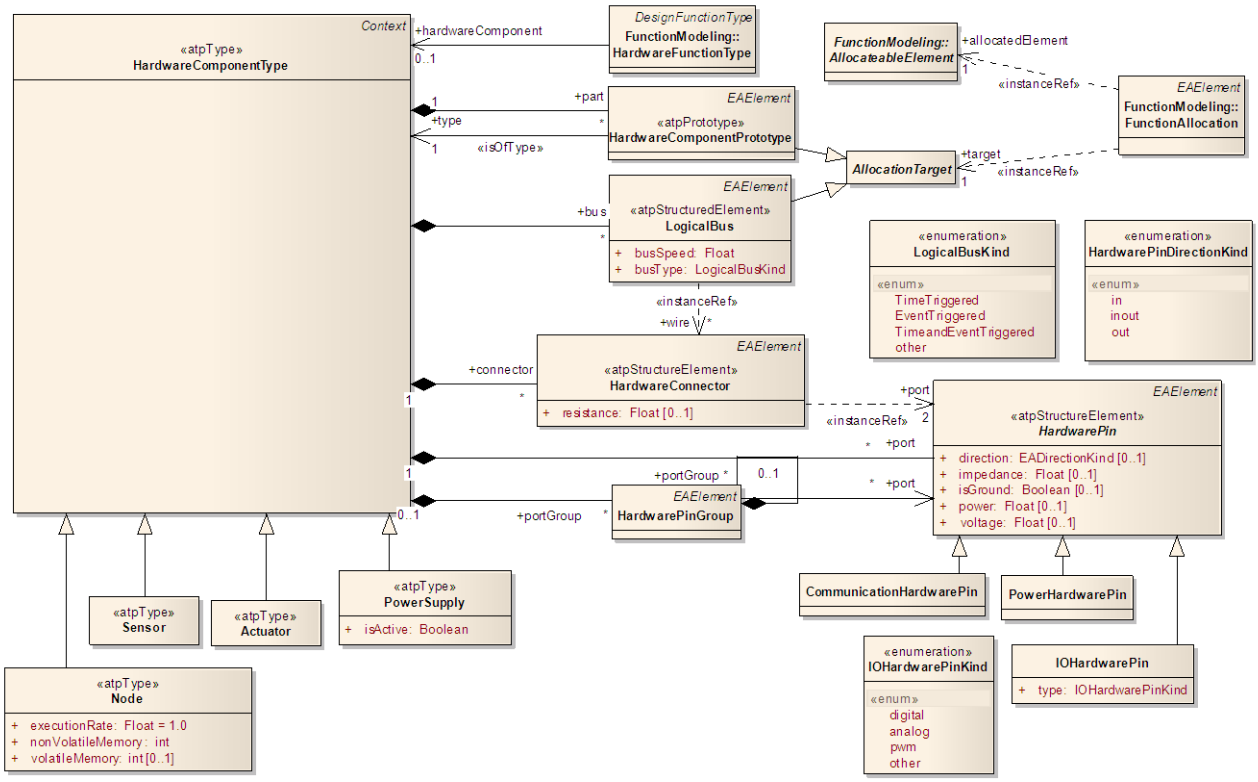


Figure 26. The Hardware Architecture

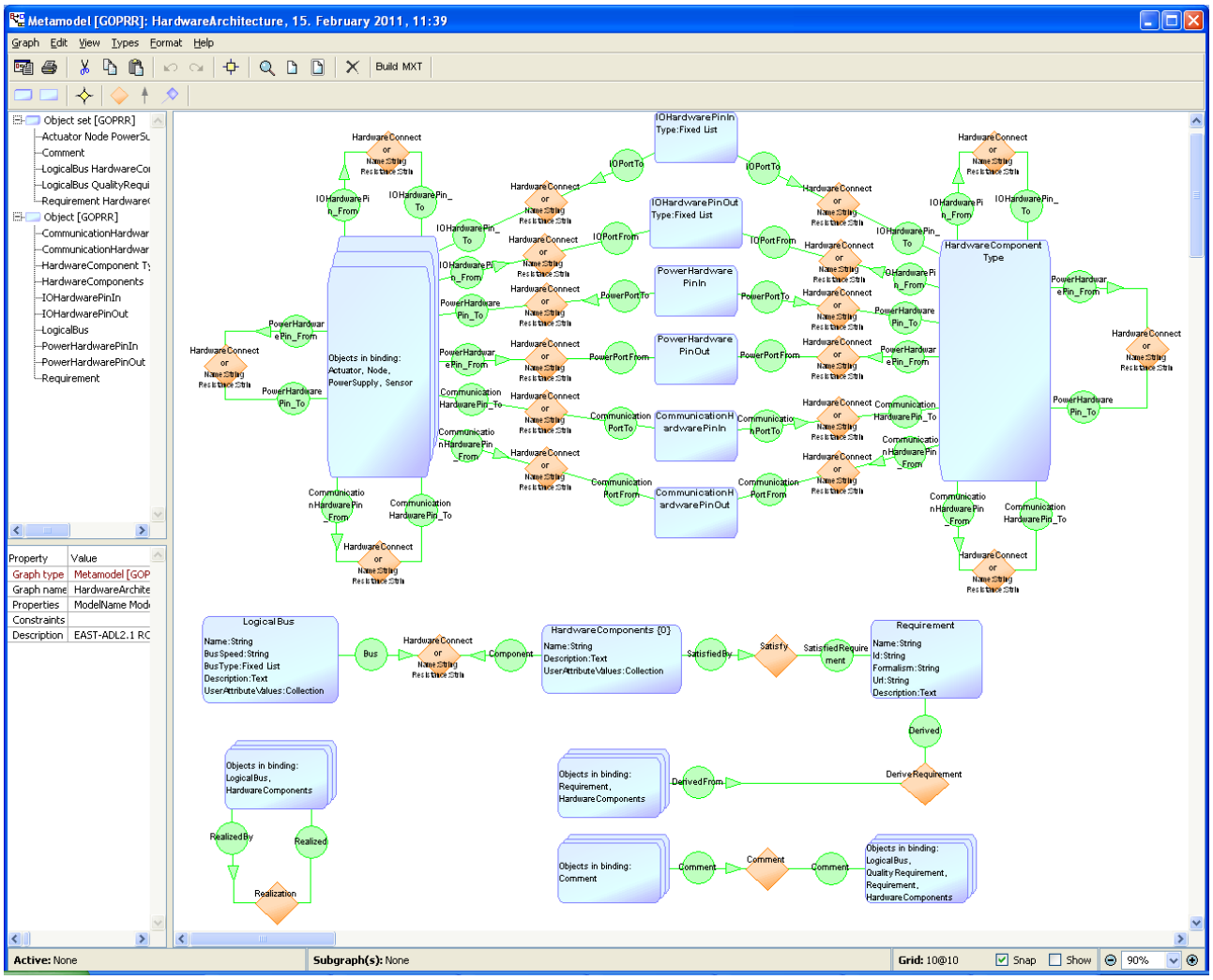


Figure 27. The metamodel of Hardware Architecture as implemented in MetaEdit+ for EAST-ADL2

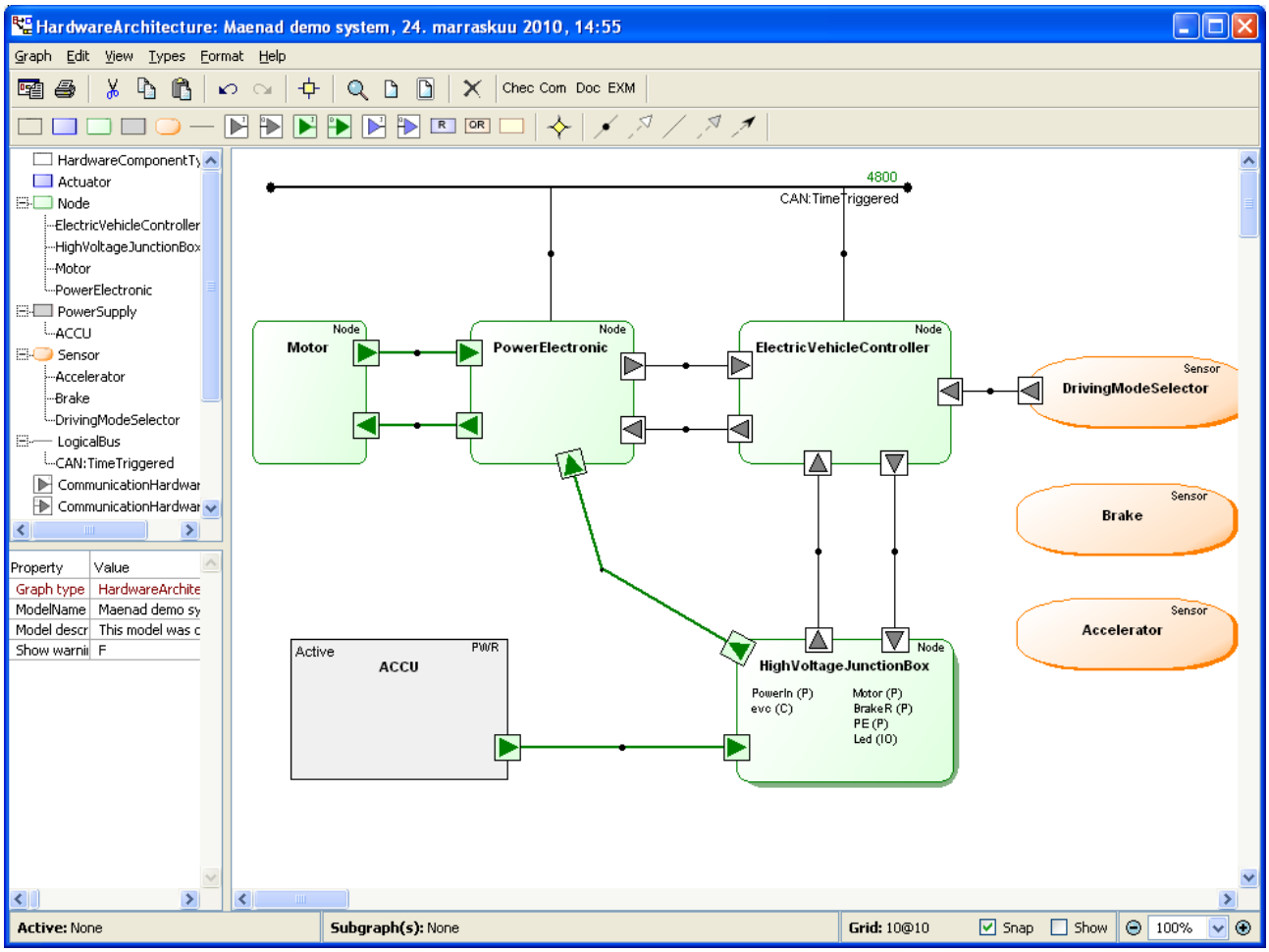


Figure 28. A sample model of an hardware architecture

To support capturing electric I/Os the metamodel must be extended with new kinds of ports and connections that enable specifying electric I/Os. The extension of electric I/Os is similar to other hardware connectors already described in the metamodels, but electric I/O connection and ports have own characteristics as follows:

- electric I/O port has an attribute called ‘Voltage’ to specify ‘Electric voltage used’
- Hardware connection for electric port has ….
- Etc.

Electric I/O can be connected only between electric I/O ports and their type are defined by hardware component types and they are used by prototypes similarly to other hardware connectors (power, hardware IO and communication).

After extending the metamodel as shown in Figure 29 the hardware architecture models can be presented in EAST-ADL2.

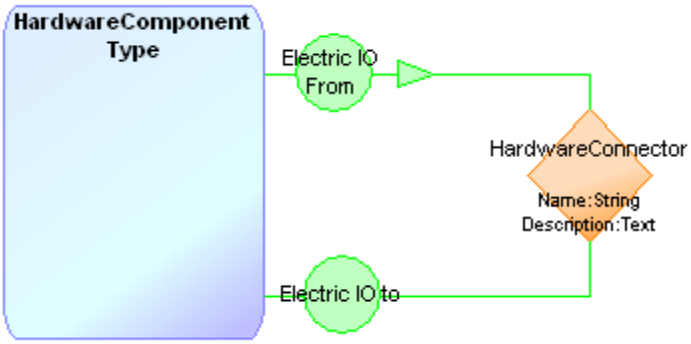


Figure 29. An extended metamodel of Hardware Architecture

A sample of hardware architecture modeling specifying electric I/O is illustrated in Figure 30 where ACCU (PowerSupply Prototype) is connected to HVJB (Node prototype) using the added language concept (electric I/O).

The notation for electric I/O distinguishes it from other ports and connections by using different coloring and line type. The ports and connection may also show relevant information about the electric I/O such as the voltage information as illustrated below (12V).

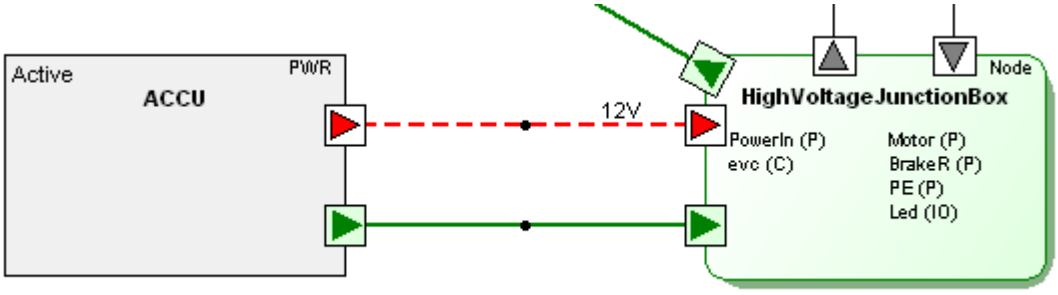


Figure 30. A sample of using the extension: electric I/O

The extended Hardware Architecture language will be tested in the pilots and refined based on the feedback from realistic usage scenarios. If change is accepted it will be incorporated to the next release of the EAST-ADL2 language.

7.7 Semantics of Realization

It shall be defined what the meaning of element X realizing element Y mean. This is particularly important for features referenced by an Item, as they define the scope of the safety element. Figure 31 shows the metamodel of the Realization relation.

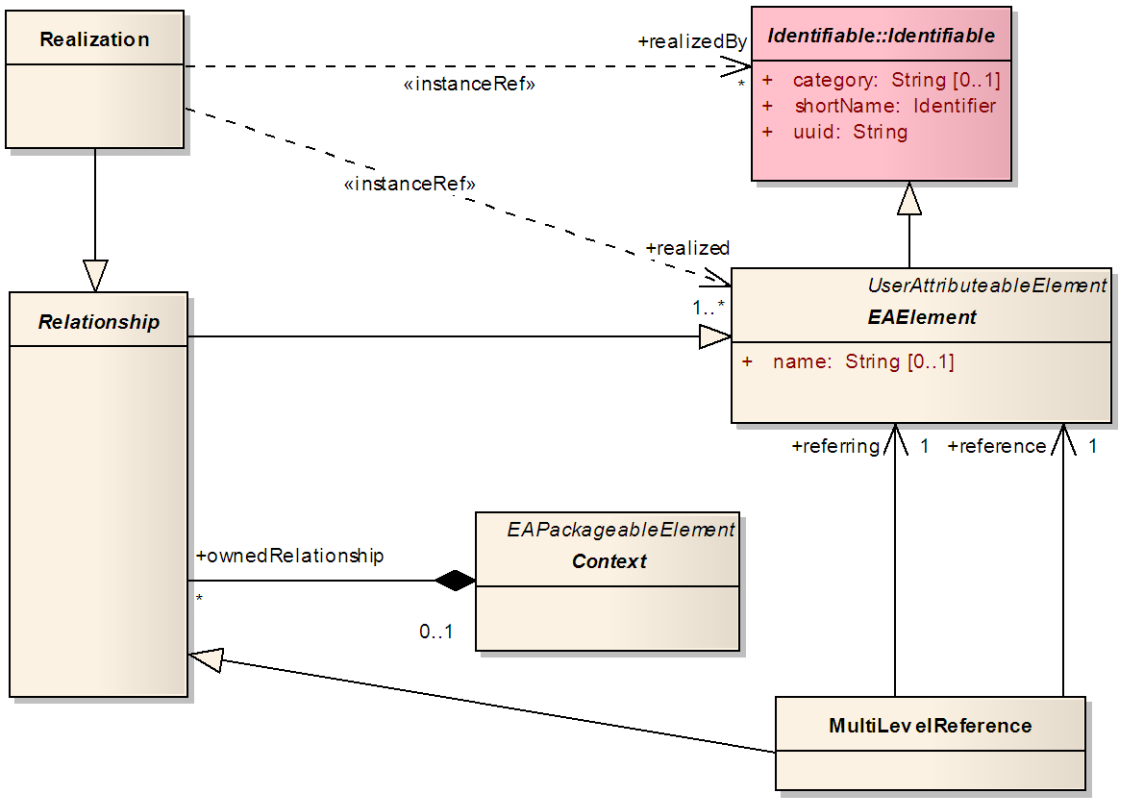


Figure 31. The metamodel of Realization

7.8 TADL2 from TIMMO-2-USE

TADL2 from the TIMMO-2-USE project has been finalized and refines TADL defined by the TIMMO project and available in EAST-ADL. This refinement also include alignment with the AUTOSAR Timing Extension.

The semantics and syntax of the timing constraints have been updated in TADL2.

New concepts in TADL2 are:

- Symbolic Time Expressions and Multiple Time Bases
- Probabilistic Timing

Using the proposed concepts of expressions mentioned in Section 7.3 also the EAST-ADL events can be modified to include a condition on when an event is observed and eligible for a timing constraint.

Also alignment with the proposed Behavior Annex from Chapter 3 has been performed.

8 Fault Injection

8.1 Background

Fault injection is the activity to manipulate a system, function, component, etc. or its interfaces to simulate faults. By observing the behavior of the component, it is possible to study its capability to handle faults and otherwise how they propagate through the component.

In the EAST-ADL context, Fault injection is relevant for several reasons:

- The EAST-ADL error propagation models are made to represent how faults propagate. Fault injection in a nominal system is a way to identify how faults propagate, and thus provide inputs for the definition of the error propagation model of each component.
- Fault injection in a system, function, component, etc. is a means of testing. This means that the EAST-ADL V&V constructs are useful for the representation of the requirements to be tested, the experiments and the outcome.

We will first discuss fault injection in relation to ISO26262.

8.2 Fault Injection and ISO 26262

EAST-ADL concepts provides support for verification and validation activities during the development phase of the safety lifecycle according to ISO 26262, including fault injection techniques.

ISO 26262 heavily relies on Verification and Validation activities to provide evidence that the obtained product complies with the safety requirements. V&V activities are carried out in a systematic way on each phase of the development: system development, HW development and SW development. At the system development phase, focus is on the integration of the item's elements and to provide evidence that the integrated elements interact correctly. Integration tests are performed at each stage of integration; software and HW integration, system integration and vehicle integration.

At the HW development level, tests are carried out to check the correctness of the HW safety mechanisms in relation to the HW safety requirements. The same apply for the SW development level, where tests are performed for SW unit and during the integration of SW unit to form a complete SW architecture.

Goals of the testing activities are

- test compliance with each safety requirement in accordance with its specification and ASIL classification;
- verify that the "System design" covering the safety requirements are correctly implemented by the entire item;
- correct implementation of functional safety and technical safety requirements;
- correct functional performance, accuracy and timing of safety mechanisms;
- consistent and correct implementation of interfaces;
- effectiveness of a safety mechanism's diagnostic or failure coverage;
- level of robustness.

ISO 26262 provides a set of testing techniques and methods that address specific goals; more precisely, a given test goal is addressed using different testing techniques. The table below summarizes the relationship between development phases, test goals and fault injection techniques used as a test method to achieve compliance with requirements. An R indicates that Fault injection is explicitly recommended to address a test goal, an O indicates optionality, i.e. that it is indirectly involved.

Table 3. Applicability of Fault Injection (FI) for test goals and phases (dark fields indicate that FI is recommended and light fields that FI is indirectly involved

			Test Goals				
		Sub phases	correct implementation of functional safety and technical safety requirements	correct functional performance, accuracy and timing of safety mechanisms;	consistent and correct implementation of interfaces	effectiveness of a safety mechanism's diagnostic or failure coverage	level of robustness
Development phases	System	HW/SW integration	R	O		R	
		System integration	R	O		R	
		Vehicle integration	R	O		R	
	HW	Unit	R	O		R	
	SW	Unit	R			R	
		SW integration	R	O		R	

8.3 EAST-ADL Support for Fault Injection

This section explains the EAST-ADL concepts for supporting Fault injection in particular, but also test and verification in general.

8.3.1 Modeling of Experiment Setup

Figure 32 shows the elements involved in the overall organization of V&V. VerificationValidation is the container element which helps to identify the V&V information. Verify is a relation that identifies the Requirement that is subject to verification through a VVCase. VVTarget is the concrete component, system, prototype, model, software, etc. that is subject to testing or any other means of verification.

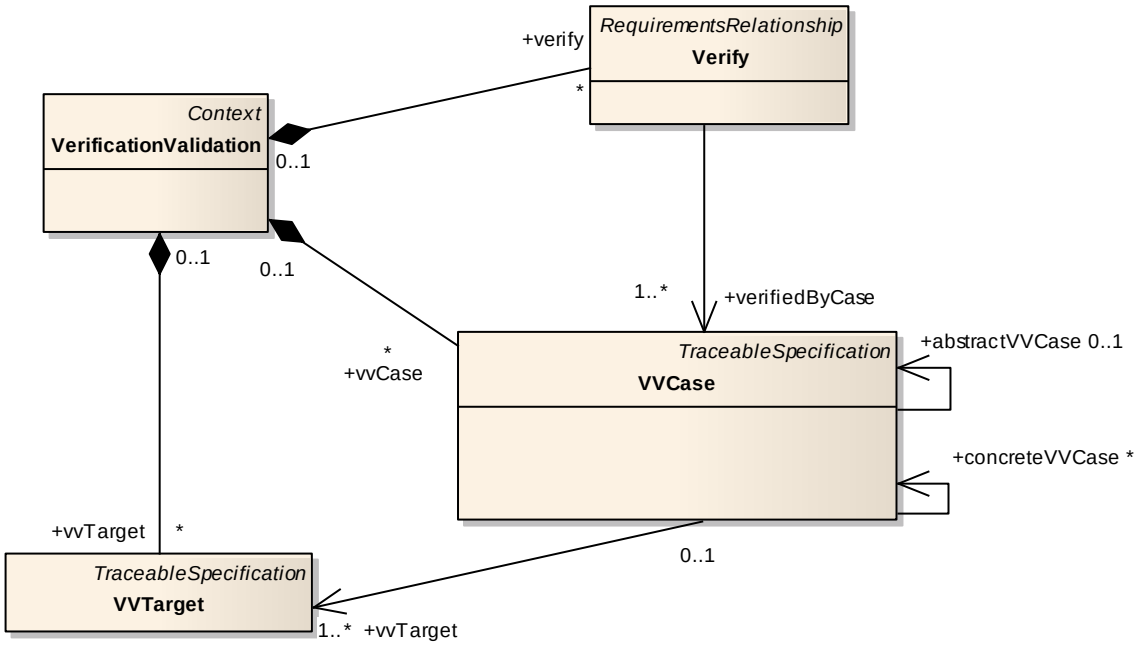


Figure 32. Overall Organization of V&V elements

The documentation of Setting up a test can be done using the VVCase construct, see Figure 33. The Verify construct relates one or several VVCase to one or several requirements. VVCase also, identifies the part of the model that is verified with vvSubject, and the concrete element that is verified with vvTarget. The VVCase is composed by several VVProcedures allowing a more fine-grained definition of the test or verification. Each VVProcedure allows stimuli and outcome to be defined. The vvLog element can be decomposed into vvActualOutcome, which can then be compared to the intended outcome.

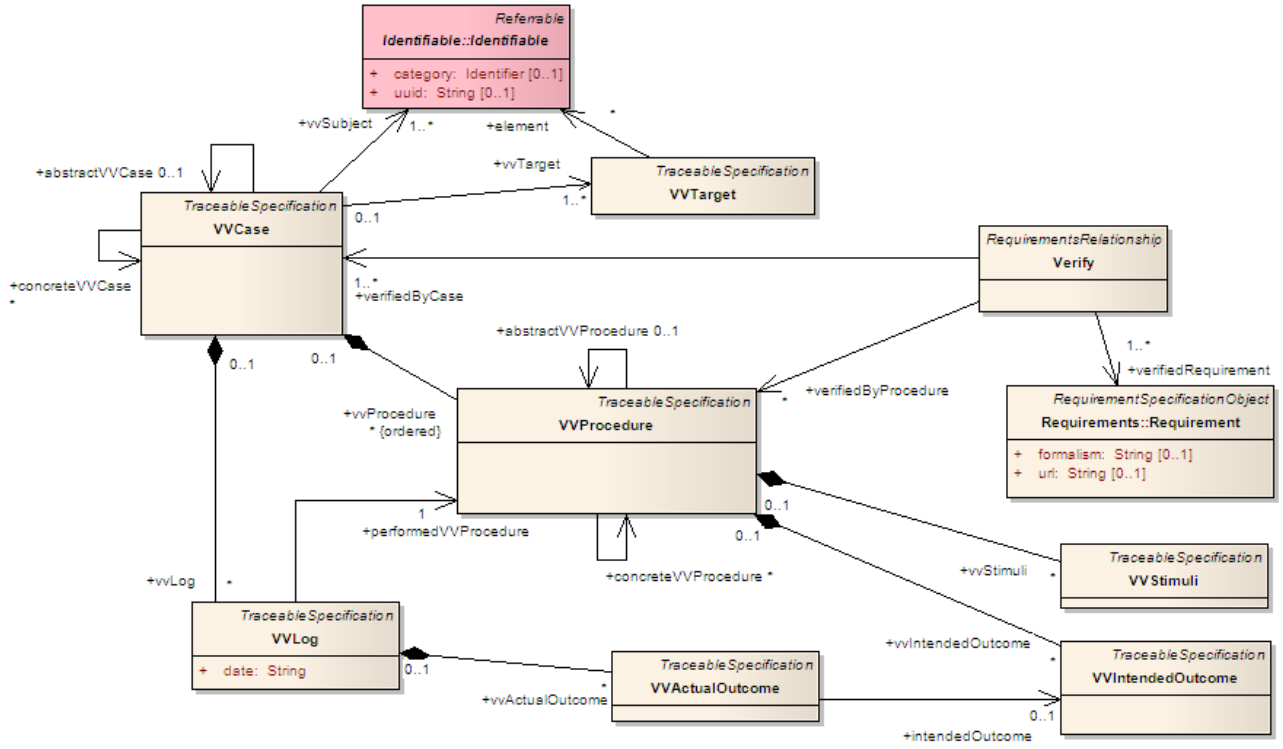


Figure 33. Elements related to setting up a test.

Test execution can be supported using behavioral definitions in Simulink, State diagrams, sequence diagrams and the like. There are also test languages like TTCN3. Depending on how the user organizes the test, a vvProcedure or its contained vvStimuli can be the placeholders of such behavior definitions. The behavioral definitions would be contained in an analysis- or design function which is linked to the vvProcedure or vvStimuli with a Refine relation, see Figure 34.

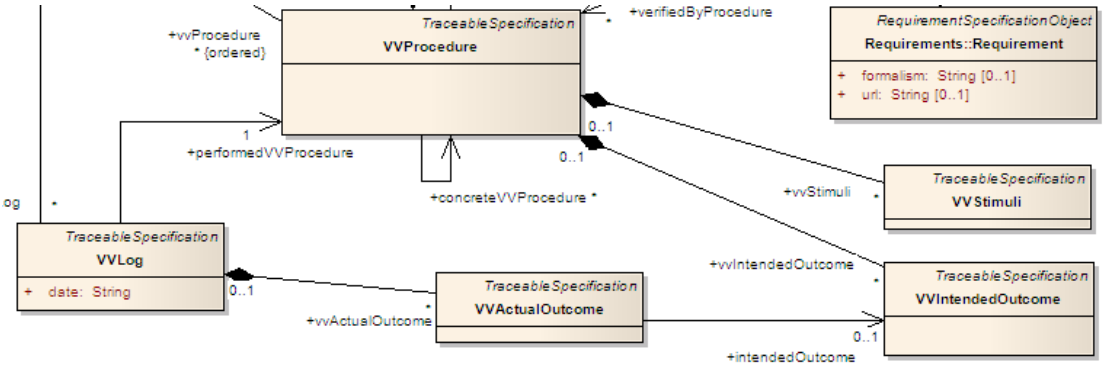


Figure 34. Details of test setup for Fault Injection

The EAST-ADL Error Models can be used to capture injected faults and resulting component failures. Figure 35 shows the elements involved in the Error Model. Typically, an ErrorModelType would be defined for the test subject of the Fault injection experiment, and FaultInPorts would be defined for each injected fault. The ErrorModelType can be associated to the AnalysisFunction, DesignFunction, HardwareComonent, etc. that is subject to fault injection, see Figure 36.

As results appear, the ErrorModelType can be refined with FailureOutports defining which failures are observed. An internal structure defining error propagation across components can also be added, if such observations can be done. Depending on how the user prefers to structure information, the ErrorModelType can be linked to vvLog or vvActualOutcome.

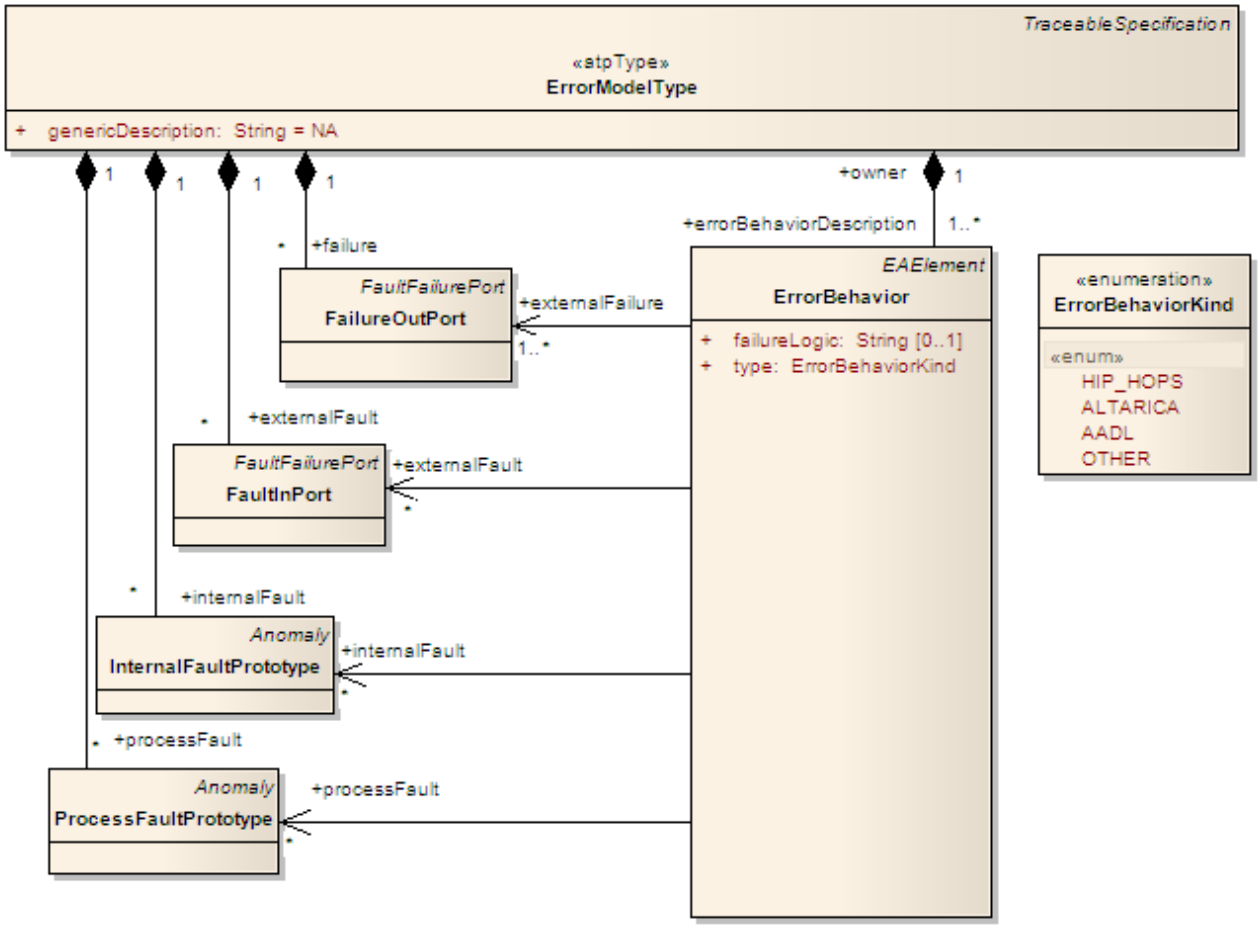


Figure 35. Error model elements

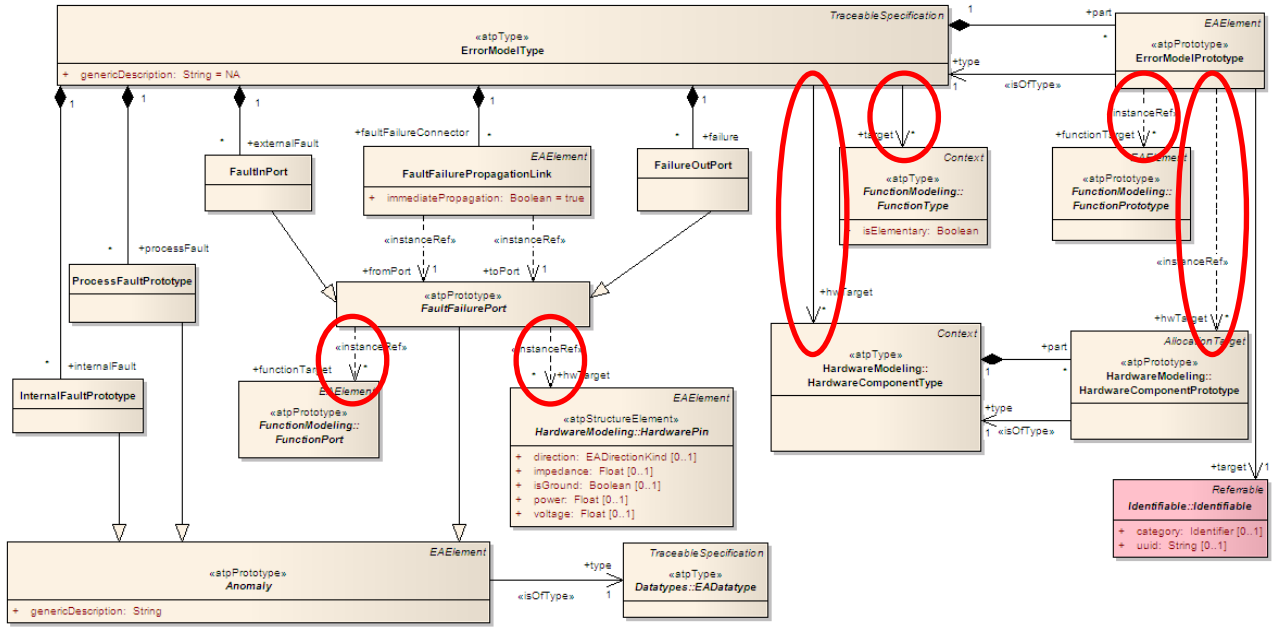


Figure 36. Linking ErrorModel to nominal elements

8.4 Discussion

EAST-ADL language and related tools could provide support for experimental V&V activities based on fault injection techniques with different scope. Experimental activities can be structured in three different sub-phases, and model based design could serve each of them in different way

- Test design: models of the systems are used to transfer information useful for the design of a test experiments. In this context, the model of the system is used as a container of data useful to derive information about the System Under Test (SUT), its boundary and to design test vectors
- Test bench setup: information is extracted from the model in order to support a semi-automatic setup of a test experiment and HW test plant. For complex systems to be analyzed, the capability to support the engineers in the semi-automatic setup of a test experiment could save days of works
- Test execution: the SUT is exercised through test equipment. Actual and intended test results are represented in the model using the V&V constructs.

The following section report a gap analysis related to the support that MAENAD language and tools could provide for fault injection experiment,

8.4.1 Addressed Requirements

The following requirements are relevant for Fault Injection:

Requirement	Addressed
VTEC#UC007 Model Fault Injection	Y
VTEC#UC008 Physical Fault Injection	Y

4SG#0049b Definition of testing	Y
4SG#0069 Enabling testing	Y
CON#0013: Fault injection and verification in HW environment	Y
CON#0014: Fault injection and verification in Modelica	Y
CRF#0036 fault injection	Partly (requirement concerns validator)
CRF#0067 Fault injection	Partly (requirement concerns validator)

In addition to the requirements above, support for test design, setup and execution is discussed below.

8.4.2 Test design

This section is mainly related on the capability of the MAENAD language and tools to support test engineers for the design of experiments. The focus will be on the support provided by the language for the formulation of tests. The first column reports the methods to derive test cases as they are expressed by ISO26262. Those methods are applicable to all type of experiment, and for fault injection as well.

Methods	Key point	Gap Analysis
Analysis of requirements		Supported through requirements packages
Analysis of external and internal interfaces	Capability to derive from the model functional/SW structure and decomposition	Full support
Analysis of equivalence classes	Capability to express partitions (valid, invalid) in the model for the input data of functions and SW components. Useful to derive test vectors reducing the total number of test cases that must be developed. Used in Black box testing and Gray box testing.	Equivalence classes in input requires behavioural model that captures the required behavior. Appropriate tooling can then establish the equivalence classes. Interface specifications are also relevant here. There is currently no such tool for EAST-ADL.
Analysis of boundary values	Derived from the equivalence classes	Appropriate tooling can assess behavioural models and interface specifications to identify boundary values. There is currently no such tool for EAST-ADL.
Error guessing based on knowledge or experience	Capability to transfer information on test vector derived from previous experiences	Supported by the V&V package
Analysis of functional dependencies	Capability to transfer information on EE architecture functionalities, their decomposition and the related dependencies	Full support due to the capability of the language to describe HW, functional and SW view of an embedded system and they relationship, functional allocation.

Methods	Key point	Gap Analysis
Analysis of common limit conditions, sequences, and sources of dependent failures		Full support for sources of dependent failures (Error modeling) Lack support to express limit conditions in the model
Analysis of environmental conditions and operational use cases		Use cases and behavioural definitions on vehicle level, plant models defined in the evironment model supports this activity.
Analysis of field experience		Field experience could be interpreted as a special kind of testing, and would then be supported by the V&V constructs.

8.4.3 Test setup

This section is mainly related on the capability of the MAENAD language and tools to support test engineers for the setup of experiments.

The focus is a gap analysis to derive plug-in that support the semi-automatic setup of an instrumented test.

Needs	Key point	Gap Analysis
Automatic generation of networks related setup	Plug in to automatically derive the information needed to setup network communications and interpretations of network data. This includes for each network signals: endianism, length, start bit, factors to obtain the physical value, message packing	The concrete network setup is defined in AUTOSAR and Fibex standards and not within the scope of EAST-ADL.
Extraction of subsystem test sets	Plug in to automatically derive test vector related to the subsystem under analysis	Implementation possible due to the hierarchical organization of the model and the capability of the model to link architectural elements and their dependencies

8.4.4 Test Execution - gaps analysis

Needs	Key point	Gap Analysis
Capability to support emulation of the environment		Link to external simulation tools capable to realize the necessary emulation is provided through dedicated bridge (Simulink gateway, Modelica exchange, Modelisar FMU import) All those external environments provide the
Capability to support emulation of the missing items		

Needs	Key point	Gap Analysis
		necessary capabilities to execute test vector, emulate plants, emulate missing items of a systems,… To be analyzed the effectiveness and suitability of the gateway starting from the above concerns

9 Electrical-Vehicle-Specific Needs

This section discusses how the needs regarding development of fully electrical vehicles are addressed by EAST-ADL. Many of these are of course general and shared with automotive embedded systems development in general. We will focus on the needs identified in the various standard and regulations related to electrical vehicles.

9.1 EAST-ADL Support for Electrical Vehicle Development

This section explains the EAST-ADL concepts for supporting Fault injection in particular, but also test and verification in general.

Hardware:

- Consider removing *impedence, power, voltage* attributes. Complex dependencies on different types of HardwarePins are possible by defining Constraints for these instead.
- Add attribute *isShield* on HardwarePin.
- Add package with annotation constraints for hardware elements. Include resistance, etc.
- attribute for nodeKind (RISC, ASIC, FPGA , CGRA and DSP.)?
    - Add documentation on OS?
    - Scheduling policy?
- Allow for complex memory modeling.
- battery, voltage regulator, DC/Dc converter, relay, fuse/interruption device, overvoltage protection device, other energy storage elements

9.2 Discussion

As reported in D2.1.1, a process was followed to define the requirements related to FEV development, in order:

- to verify the capability of the current version of EAST-ADL2 to cover the needs related to specific characteristics of FEVs, and to extend its features if necessary; and, similarly,
- to verify the capability of the analysis tools and to give inputs to adapt or, possibly, create specific tools to perform the necessary analyses;
- to define an extension of the basic E/E system development methodology resulted from ATTEST2, in order to help designers to perform the development activities required by the standards and the regulations, or those compliant to best practices or engineering needs for EV development.

Therefore, through a sequence of activities according to a bottom-up approach, three categories of requirements have been defined: language requirements, analysis requirements, and methodology requirements.

The requirements defined have been reported in an Excel sheet and, subsequently, in Enterprise Architect, to comply with the method followed for the collection of MAENAD requirements, thus allowing better traceability, uniform categorization, assignment to WPs.

The following table is an excerpt of the Excel file and includes only the language requirements.

Reference are given to the requirement codes used in EA; the field “subject” has been introduced to better identify the related engineering topic and to establish a link with the analysis and methodology requirements related to the same topic.

In addition, an empty field has been here introduced, which will be filled in to specify the technical requirements as to implement the language features. This new definition activity will be performed in the next months, both with the analysis of the requirements to verify which of the requirements can be met with the present EASTADL2 version.

It has to be pointed out that in the following table some language requirements are referred to a specific standard or regulation. However, the requirements, in some cases, can be referred to similar standards (not mentioned here, but only in the Excel sheet, which gives a more global view of the analysis conducted to define the requirements).

9.3 Requirements

In the following we list those project requirements that are relevant for EV development. The cells “Supported”, “Partly Supported” and “Not Supported” provide comments on if and how the respective requirement is supported by EAST-ADL’s modeling concepts. For some requirements of lower priority, no such comment is given. In these cases the language impact of the requirement was deemed insignificant but might be revisited in period 3 of MAENAD.

In general, the comments given below reflect the status at the end of period 2 of MAENAD.

4SG#0076: 6469-1 - Insulation modelling	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable modelling of insulation, including: - Insulation symbols - Insulation attributes (withstand voltage, resistance, presence of DC or AC parts, creepage distance, ref. to standards...) - Insulation devices (to describe the interconnection between isolated and not isolated physical parts, e.g. communication, power supply, drives) - High voltage parts (wrt physical view) in order to take note of the requirements regarding creepage distance, clearance, labeling, wire color, insulation.
Derived from	· 4SG#0007: ISO 6469-1
Partly Supported	Insulation symbols can be supported by putting a user-defined attribute on all elements requiring a specific symbol. It is then possible for a tool to visualize properly. Insulation attributes (withstand voltage, resistance, presence of DC or AC parts, creepage distance, ref. to standards...) can be defined using GenericConstraint Insulation devices (to describe the interconnection between isolated and not isolated physical parts, e.g. communication, power supply, drives) can be defined using HWComponents in conjunction with GenericConstraints that define the metrics. Requirements on High voltage parts regarding creepage distance, clearance, labeling, wire color, insulation, etc. can be defined with user defined attributes or Requirements
Not Supported	None

4SG#0077: 6469-1 - Insulation analysis	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support insulation analysis: overall resistance, voltage compliance.
Derived from	· 4SG#0007: ISO 6469-1
Partly Supported	EAST-ADL can represent the required attributes using genericConstraints and user-defined attributes. The project has currently no tool to do the analysis.

4SG#0078: 6469-1 - Insulation design and verification	
Alias	

Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the insulation design and verification, in particular: - Deployment of insulation resistance - Addressing insulation monitoring system - Hazard analysis and risk assessment concerning insulation monitoring - Design issues concerning recharging (grounding, communication) - Test planning concerning insulation - Production, operation and maintenance requirements during design phase (ISO 26262-4)
Derived from	· 4SG#0007: ISO 6469-1
Partly Supported	Representation: Insulation resistance can be modelled using HDA elements, an insulation monitoring system can be modelled using FDA and HDA elements. Hazard analysis and risk assessment is done using ISO26262 constructs Charging design issues are captured using FDA and HDA elements. Test planning concerning isolation is represented using V&V Production, operation and maintenance requirements during design phase are represented using requirements concepts. Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0079: 6469-1 - Prevention of danger due to heat generation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design and verification of a monitoring system to prevent dangerous effects to persons, in the case of failures producing heat generation.
Derived from	· 4SG#0007: ISO 6469-1
Partly Supported	Representation: Heat generation can be assessed based on the power, which is modelled using generic constraints. Fault detection and management functionality can be represented using FDA and HDA elements. Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0080: 6469-1 - RESS interruption device modelling	
Alias	
Status	Proposed

Type	
Priority	Medium
Description	The language shall enable modelling of an over-current interruption device, including power-flow paths and interruption characteristics (current-time characteristics). Note: RESS: Regenerative Energy Storage System
Derived from	· 4SG#0007: ISO 6469-1
Partly Supported	Representation: An over-current interruption function can be represented using FDA and HDA elements. Power-flow paths and current-time characteristics can be represented using HW functions allocated to HWComponents. Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0081: 6469-1 - RESS short circuit analysis	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support insulation RESS short circuit analysis (current and thermal effect analysis).
Derived from	· 4SG#0007: ISO 6469-1
Partly Supported	Representation: An short circuit analysis can be performed based on the HDA annotated with insulation resistance, voltage and connector resistance. Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0082: 6469-1 - Design of RESS interruption device	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the following activities: - the design and verification of an overcurrent interruption device - Hazard analysis in the case of short circuit of RESS - Planning of short circuit test
Derived from	· 4SG#0007: ISO 6469-1

4SG#0127: FMVSS No. 114 - Modeling keylocking device	
Alias	
Status	Proposed
Type	
Priority	Medium

Description	The language shall provide means to model a keylocking device with lock and unlock conditions
Derived from	· 4SG#0072: FMVSS No. 114 Theft protection
Supported	Representation: Key-locking device can be represented using FDA and HDA elements. Requirements and behavior constraints/behavior definition can be used to formalize the required behavior. Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0083: 6469-2 - Connection to off board power supply	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a means to make impossible to move the vehicle when connected to off-board electric power supply and charged by the user
Derived from	· 4SG#0008: ISO 6469-2
Partly Supported	Representation: The required behavior can be represented using regular FDA and HDA elements Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0084: 6469-2 - Warning of reduced power	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a warning to signal to the driver that the propulsion power is reduced, in the case this is done
Derived from	· 4SG#0008: ISO 6469-2
Partly Supported	Representation: A power warning function can be represented using FDA and HDA elements.

	Methodology: The identified concerns are managed in the FEV methodology swimlane

4SG#0085: 6469-2 - Driving backwards	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of means to prevent unintentional switching in reverse when the vehicle is in motion (two options are available, see the standard)
Derived from	· 4SG#0008: ISO 6469-2

4SG#0086: 6469-2 - Parking	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of - a warning to indicate whether propulsion is in the driving–enable mode, when user leaves the vehicle - a safety mechanism to prevent unexpected movements.
Derived from	· 4SG#0008: ISO 6469-2

4SG#0087: 6469-2 - Protection against failures	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology, and in particular the functional safety development, shall consider unintended acceleration, deceleration and reverse motion as hazards to be prevented or minimized.
Derived from	· 4SG#0008: ISO 6469-2

4SG#0088: 6469-3 - Protection of persons against electric shock	
Alias	
Status	Proposed
Type	
Priority	Medium

Description	The EV development methodology shall include: - the design of mechanical and electronics means according to the standard - the verification planning for measures protection (design verification, test plan)
Derived from	· 4SG#0009: ISO 6469-3

4SG#0089: 6469-3 - Protection of persons against electric shock (alternative approach)	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the conduction of an appropriate hazard analysis with respect to electric shock and establish a set of measures which give sufficient protection against electric shock
Derived from	· 4SG#0009: ISO 6469-3

4SG#0090: 6469-3 - Isolation resistance requirements	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the assignment of insulation resistance to high voltage components as to achieve the overall insulation resistance (dc, ac cases).
Derived from	· 4SG#0009: ISO 6469-3

4SG#0091: 6469-3 - Language requirements concerning potential equalization	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the representation of bonding/grounding of physical elements (proper symbols)
Derived from	· 4SG#0009: ISO 6469-3

4SG#0092: 6469-3 - Methodology requirements concerning potential equalization	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include: - the design of insulation barriers and bonded conductive equalization barriers

	- the planning verification of barriers, including bond testing.
Derived from	· 4SG#0009: ISO 6469-3

4SG#0093: 6469-3 - Analysis of charging inlet disconnection	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support the analysis of charging inlet voltage decrease when the connector is disconnected
Derived from	· 4SG#0009: ISO 6469-3

4SG#0094: 6469-3 - Methodologu for the charging inlet disconnection	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include: - the design of the charge system, as to ensure voltage decrease of inlet according to time requirements. - the verification by simulation, analysis and testing.
Derived from	· 4SG#0009: ISO 6469-3

4SG#0095: 6469-3 - Grounding and isolation resistance requirement for charging inlet	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the charging system as to meet insulation requirements in the case of ac and ac inlet.
Derived from	· 4SG#0009: ISO 6469-3

4SG#0096: EN 61851 - Types of EV connection	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - the definition of the charging system according to one of the 4 charging modes. - the definition of the control pilot mandatory and optional functions (modes 2-4), including charging operation states.

Derived from	· 4SG#0016: EN 61508

4SG#0097: EN 61851 - Protection against electric shock	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the definition and the design of measures to prevent electric shock both in normal service and in case of fault.
Derived from	· 4SG#0016: EN 61508

4SG#0098: EN 61851 - Analysis of stored energy – discharge of capacitors	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support the analysis of the voltage transient of any accessible part after EV disconnection
Derived from	· 4SG#0016: EN 61508

4SG#0099: EN 61851 - Methodology concerning the stored energy – discharge of capacitors	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the EV voltage input in such a way to control the voltage decay after EV disconnection
Derived from	· 4SG#0016: EN 61508

4SG#0100: EN 61851 - Detection of the electrical continuity of the protective conductor	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a monitoring system to detect the electrical continuity of the protective conductor during charging modes 2, 3 and 4.
Derived from	· 4SG#0016: EN 61508

4SG#0101: EN 61851 - Dielectric withstand voltage	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include: - the design of the on board charging equipment as to withstand the test voltage at any input connection (2U +1000 V, min. 1500 V a.c.). - the design of all vehicle equipment as to withstand a test voltage of 4kV between a.c. or d.c. input and low voltage inputs (if any).
Derived from	· 4SG#0016: EN 61508

4SG#0102: EN 61851 - Electric vehicle insulation resistance	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the verification of the insulation resistance (by analysis and testing). Minimum required: 1 Mohm.
Derived from	· 4SG#0016: EN 61508

4SG#0103: EN 61851 - Drive train interlock	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a system to detect the connection of the mobile connector or that the plug and the cable have been stored in the vehicle. The system shall also inhibit the drive train.
Derived from	· 4SG#0016: EN 61508

4SG#0104: J2289 - Vehicle operational modes	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the defining the vehicle operational modes according to those required by the standard and eventually justify the possible discrepancies
Derived from	· 4SG#0018: J2289

4SG#0105: J2289 - Key-on discharge	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to model - the power supply network including fault protection devices with their current-time characteristics - the auxiliary equipment including power requirements/ power profiles
Derived from	• 4SG#0018: J2289

4SG#0106: J2289 - Key-on discharge	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support: - Power and energy analysis to estimate range, taking into account auxiliares consumption - Time analysis of fault protection intervention
Derived from	• 4SG#0018: J2289

4SG#0107: J2289 - Key-on discharge	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - Assessment of battery capability to match the vehicle demand (range, supply of auxiliary equipment) - Designing means to detect and limit the overdischarge of individual cells - Providing fault protection devices (fuses, fast contactors)
Derived from	• 4SG#0018: J2289

4SG#0108: J2289 - Key-on Regen operation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to define -voltage limit data/ requirements of the drive components -recommended battery current and voltage profiles during high SoC

	and to model the battery for current-voltage transients analysis
Derived from	· 4SG#0018: J2289

4SG#0109: J2289 - Key-on Regen operation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support: the analysis of voltage transients during regenerative braking
Derived from	· 4SG#0018: J2289

4SG#0110: J2289 - Key-on Regen operation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - the assessment of the compliance of the voltage with the limits during regeneration - the design of means to avoid drive component overvoltage occurrence during regeneration - the verification of the compliance with current and voltage profiles - design means to limit battery current and voltage during regeneration according to the specified profiles
Derived from	· 4SG#0018: J2289

4SG#0111: J2289 - Key-on charge	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to model the electrical characteristics of the charge system components (e.g. current, voltage)
Derived from	· 4SG#0018: J2289

4SG#0112: J2289 - Key-on charge	
Alias	
Status	Proposed
Type	
Priority	Medium

Description	Maenad tools should support: the matching analysis of power equipment (current, voltage)
Derived from	· 4SG#0018: J2289

4SG#0113: J2289 - Key-on charge	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - the verification that all charge system components match w.r.t. electrical characteristics - the design of the charge algorithm to be performed with the battery supplier
Derived from	· 4SG#0018: J2289

4SG#0114: J2289 - Key-Off Parked Off Plug Operating	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to describe the power characteristics of the devices running in key-off mode
Derived from	· 4SG#0018: J2289

4SG#0115: J2289 - Key-Off Parked Off Plug Operating	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support: the power requirement analysis in key-off mode
Derived from	· 4SG#0018: J2289

4SG#0116: J2289 - Key-Off Parked Off Plug Operating	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - the realization the energy management to prevent excessive discharge due to vehicle

	equipment operating in key-off mode - the verification of the energy behavior in key-off mode by simulation/calculation - the design of charge algorithm with the battery supplier
Derived from	· 4SG#0018: J2289

4SG#0117: J2289 - Parked Off Plug IDLE/Storage Operation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to model the battery disconnect system (mechanical switch)
Derived from	· 4SG#0018: J2289

4SG#0118: J2289 - Parked Off Plug IDLE/Storage Operation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	
Derived from	· 4SG#0018: J2289

4SG#0118: J2289 - Parked Off Plug IDLE/Storage Operation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - the design of the contactor operation as to be deactivated in the case of crash or isolation fault - the design of the disconnect system for added safety during service or by first responders during
Derived from	· 4SG#0018: J2289

4SG#0119: J2289 - Parked Off Plug IDLE/Storage Operation	
Alias	
Status	Proposed
Type	
Priority	Medium

Description	The EV development methodology shall include - the design of contactor operation as to be deactivated in the case of crash or isolation fault - the design of the disconnect system for added safety during service or by first responders during
Derived from	· 4SG#0018: J2289

4SG#0120: J2289 - Discharge management - Performance limits	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to define the operation limits of the battery (temperature ranges, current, under-voltage)
Derived from	· 4SG#0018: J2289

4SG#0120: J2289 - Discharge management - Performance limits	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the BMS to protect for overtemperature, under-temperature, over-current
Derived from	· 4SG#0018: J2289

4SG#0122: J2289 - Charge management	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of communication in compliance with SAE J1772, SAE J1773, and SAE J2293
Derived from	· 4SG#0018: J2289

4SG#0123: J2289 - Key-on startup diagnostics and warning	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to represent different levels of warnings (depending on the fault severity)

Derived from	· 4SG#0018: J2289

4SG#0124: J2289 - Key-on startup diagnostics and warning	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of key-on running diagnostics and warning procedures
Derived from	· 4SG#0018: J2289

4SG#0125: J2289 - Service diagnostics	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of service diagnostics
Derived from	· 4SG#0018: J2289

4SG#0126: J2289 - Toxic emissions - Flammable gasses	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall consider toxic emissions and flammable gasses caused by battery damages
Derived from	· 4SG#0018: J2289

4SG#0128: FMVSS No. 114 - Design of keylocking device	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a keylocking system to prevent the activation of the motor and steering or selfmobility (or both).
Derived from	· 4SG#0072: FMVSS No. 114 Theft protection

4SG#0129: FMVSS No. 114 - Operation and performance of keylocking device

Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include: - the design of the operation of keylocking system according to the standard (see interaction with park command). - the verification (by calculation and testing) that the maximum movement of the vehicle when locked is less than the max. allowable limit.
Derived from	· 4SG#0072: FMVSS No. 114 Theft protection

4SG#0130: FMVSS No. 102 Transmission shift lever design	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the shift lever according to the sequence position and rotation requirements given by the regulationj.
Derived from	· 4SG#0073: FMVSS No. 102 Transmission shift lever

4SG#0131: R 116 Unauthorized use - Design of locking device	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of devices to prevent unauthorized use (deactivation of engine in combination with a system to lock other vehicle functions, see regulation)
Derived from	· 4SG#0075: R 116 Unauthorized use

4SG#0132: R 116 Unauthorized use - Functional safety analysis of locking device	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the conduction of functional safety analyses to cover the devices intended to prevents unauthorized use
Derived from	· 4SG#0075: R 116 Unauthorized use

4SG#0133: FMVSS No. 135 Regenerative brake system

Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include: - the development of braking system according to the operation mode of the RBS: control of RBS by ABS (if RBS is always active, also in neutral without any means to disconnect it by the driver, RBS is part of the service braking system); - item definition: consider the interactions between RBS and ABS (w.r.t. interfacing and system definition in ISO 26262)
Derived from	· 4SG#0071: FMVSS No. 135 Brake system

4SG#0134: FMVSS No. 135 Modeling of diagnostics and warning of brake system	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall provide means to model HMI interface for visual indicators
Derived from	· 4SG#0071: FMVSS No. 135 Brake system

4SG#0135: FMVSS No. 135 Design of diagnostics and warning system of brake system	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include - diagnostics task related to RBS, in order to transmit information to the visual warning indicator - design pof roper warning in the case of failure of brake power supply, reduced SoC, RBS failure
Derived from	· 4SG#0071: FMVSS No. 135 Brake system

4SG#0136: FMVSS No. 135 Analysis of brake system performance	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support the analysis of power management and warning of brake system supply battery, to ensure brake operation, motor shutdown and warning at battery depleted state of charge
Derived from	· 4SG#0071: FMVSS No. 135 Brake system

4SG#0137: FMVSS No. 135 Testing of brake system performance in depleted SOC battery	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the braking test in depleted battery state-of-charge condition
Derived from	· 4SG#0071: FMVSS No. 135 Brake system

4SG#0138: ISO 8715 - Performance testing - Terms and definitions	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the definition of vehicle performance characteristics according to the terms and definitions given by the standard.
Derived from	· 4SG#0019: ISO 8715

4SG#0139: ISO 8715 - Performance testing - Language for test cases definition	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the definition of the test cases according to the test conditions and test procedures required by the standard. Scope: to define test profiles for simulation
Derived from	· 4SG#0019: ISO 8715

4SG#0140: ISO 8715 - Performance testing - Simulation tools for vehicle performance analysis	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support the simulation of vehicle performance according to test condition and test case requirements
Derived from	· 4SG#0019: ISO 8715

4SG#0141: ISO 8715 - Performance testing - Testing activity	
Alias	

Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the vehicle performance testing according to test condition and test procedure requirements.
Derived from	· 4SG#0019: ISO 8715

4SG#0142: ISO 8714 - Energy and range testing - Terms and definitions	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the definition of vehicle energy consumption and range characteristics according to the terms and definitions given by the standard
Derived from	· 4SG#0020: ISO 8714

4SG#0143: ISO 8714 - Energy and range testing - Language for energy and range tes cases definition	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the definition of the test cases according to the test conditions and test procedures required by the standard. Scope: to define test profiles for simulation Include standard test cycle (European, Japan, USA cycles)
Derived from	· 4SG#0020: ISO 8714

4SG#0144: ISO 8714 - Energy and range testing - Simulation tools for energy and range analysis	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support the simulation of vehicle energy consumption and range according to test condition and test case requirements
Derived from	· 4SG#0020: ISO 8714

4SG#0145: ISO 8714 - Energy and range testing - Simulation of energy and range performance	
Alias	

Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the simulation of vehicle performance according to test conditions and test procedure requirements
Derived from	· 4SG#0020: ISO 8714

4SG#0146: ISO 12045-2 - Lithium batteries - Language for test purposes	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the definition of battery model parameters according to the test purpose (e.g. energy efficiency, charging and discharging resistance)
Derived from	· 4SG#0023: ISO 12405-2

4SG#0147: ISO 12045-2 - Lithium batteries - Modelling for test purposes	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The language shall enable the modelling in compliance with test conditions requirements (e.g. battery state of charge, power consumption of the auxiliaries, test mass, etc.)
Derived from	· 4SG#0023: ISO 12405-2

4SG#0148: ISO 12045-2 - Lithium batteries - Simulation according to test condition requirements	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the simulation of vehicle performance according to test conditions requirements(when applicable)
Derived from	· 4SG#0023: ISO 12405-2

4SG#0149: ISO 12045-2 - Lithium batteries - Simulation tool according to test procedure requirements	
Alias	
Status	Proposed

Type	
Priority	Medium
Description	Maenad tools should support the simulation according to test case and test procedures requirements
Derived from	· 4SG#0023: ISO 12405-2

4SG#0150: SAE J1277 Conductive charge coupler - Control pilot modeling	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Model communication protocol based on PWM and signal amplitude (by switching a resistor)
Derived from	· 4SG#0074: SAE J1277 Conductive charge coupler

4SG#0151: SAE J1277 Conductive charge coupler - Communication design	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the communication according to the standard (charging station status, power level, fault conditions)
Derived from	· 4SG#0074: SAE J1277 Conductive charge coupler

4SG#0152: SAE J1277 Conductive charge coupler - Management of connector signals	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the management of the connector detection signal: to start charge control, to engage drive train interlock, to reduce charge load during disconnection
Derived from	· 4SG#0074: SAE J1277 Conductive charge coupler

4SG#0153: SAE J1277 Conductive charge coupler - Desig of charging state machine	
Alias	
Status	Proposed
Type	
Priority	Medium

Description	The EV development methodology shall include the design of the charging state machine according to the standard, including safe states in the case of fault.
Derived from	· 4SG#0074: SAE J1277 Conductive charge coupler

4SG#0154: SAE J1277 Conductive charge coupler - Design charge indicators and disgnostics	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the definition of the charge status indicator, including diagnostic functions.
Derived from	· 4SG#0074: SAE J1277 Conductive charge coupler

4SG#0155: R13H Braking - Simulation tools to analyse brake compensation transients	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	Maenad tools should support the analysis (e.g. by simulation) of the the compensation transients to verify that it is attained within the required time and value limits
Derived from	· 4SG#0070: R13H Braking

4SG#0156: R13H Braking - Design of braking compensation transients	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	If the RBS is part of service brake, the EV development methodology shall include the design of the braking inputs, compensating the variations of the regenerative braking and ensuring breaking action in all wheels.
Derived from	· 4SG#0070: R13H Braking

4SG#0157: R13H Braking - Design interaction between ABS and RBS	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the development task to define and

	manage the interaction between ABS and RBS
Derived from	· 4SG#0070: R13H Braking

4SG#0160: J2234 - Electric isolation	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include activities to: - Design the high voltage insulation (100 ohm/V DC, 500 ohm/V AC) - Design barriers between AC and DC, if the DC limit is applied - Plan testing to demonstrate high voltage withstand capability - Design an isolation loss monitoring system
Derived from	· 4SG#0013: J2234

4SG#0161: J2234 - High Voltage Automatic Disconnect System	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include activities to: - Design an automatic disconnect system actuated: - by a crash sensor - in the case of loss of isolation, only in non-motoring mode - in the case of overcurrent condition, as a primary or secondary protection - according to the guidelines given by SAE J2344 - Design a crash sensor, properly qualified to operate in the crash tests. - Design the disconnect to be activate by the crash sensor and to maintain disconnection after crash.
Derived from	· 4SG#0013: J2234

4SG#0162: J2234 - High Voltage Manual Disconnect System	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a manual disconnect system actuated by an interlock loop
Derived from	· 4SG#0013: J2234

4SG#0163: J2234 - Grounding	
Alias	
Status	Proposed

Type	
Priority	Medium
Description	The EV development methodology shall include the design of the grounding of the conductive cases containing high voltage systems, also by means of indirect connection.
Derived from	· 4SG#0013: J2234

4SG#0164: J2234 - Fault monitoring	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of - a fault monitoring system - the vehicle operation in such a way that the vehicle operator is not allowed to persist in unsafe condition
Derived from	· 4SG#0013: J2234

4SG#0165: J2234 - Rechargeable Energy Storage System State-of-Charge	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of the operation in low state-of-charge in such a way that - the performance of the critical safety systems is not degraded - the state is indicated in a separate indicator if the vehicle performance is reduced
Derived from	· 4SG#0013: J2234

4SG#0166: J2234 - Mechanical safety	
Alias	
Status	Proposed
Type	
Priority	Medium
Description	The EV development methodology shall include the design of a lock system activated when the shift mechanism is in P position or the key is in “off” position.
Derived from	· 4SG#0013: J2234

CON#2001: Support driving profiles	
Alias	Support driving profiles

Status	Approved
Type	«Language»
Priority	Medium
Description	Clarify whether we need language extensions for supporting driving profiles Derived from Use Case CON#0001
Derived from	• CON#0001: Adopt ID4EV use cases

CRF#0004b Isolation	
Alias	ISO 6469-1and UNECE R100 / Isolation
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for the isolation resistance of the RESS (Rechargeable energy storage system). For a RESS not embedded in a whole circuit, the minimum requirement for the isolation resistance Ri divided by its maximum working voltage shall be 100 O/V, if not containing a.c., or 500 O/V, if containing a.c. without additional a.c. protection throughout the entire lifetime of the RESS. When the RESS is integrated in a whole electric circuit, a higher resistance value for the RESS may be necessary. The measurement shall be done following the recommended procedure after a preconditioning and conditioning period.
Derived from	
Partly Supported	Isolation Resistance can be modelled using constraints Verification measurements can be defined using V&V constructs Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0005b Creepage and clearance distance	
Alias	ISO 6469-1 / Creepage and clearance distance
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements on clearance and creepage distance between RESS terminals. a) In the case of a creepage distance between two RESS connection terminals: d W 0,25U + 5 b) In the case of a creepage distance between live parts and the electric chassis: d W 0,125U + 5

	where d is the creepage distance between the live part and the electric chassis, in millimetres (mm); U is the maximum working voltage between the two RESS connection terminals, in volts (V). The clearance between conductive surfaces shall be 2,5 mm minimum.
Derived from	
Partly Supported	Requirements on clearance and creepage distance can be formalized using constraints Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0006b Heat generation	
Alias	ISO 6469-1 and UNECE R100 / Heat generation
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account heat generation by the RESS under first-failure conditions. Heat generation under any first-failure condition, which could form a hazard to persons, shall be prevented by appropriate measures, e.g. based on monitoring of current, voltage or temperature.
Derived from	
Partly Supported	Requirements on heat generation control can be represented using regular requirements. Current, Voltage and temperature monitoring can be represented using regular FDA and HDA constructs. Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0007b Gases emission	
Alias	ISO 6469-1 and UNECE R100 / Gases emission
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account emission of hazardous gases by the RESS. No potentially dangerous concentration of hazardous gases and other hazardous substances shall be allowed anywhere in the driver, passenger and load compartments. Refer to the latest version of applicable National/International Standards or

	regulations for the maximum allowed accumulated quantity of hazardous gases and other substances. Appropriate countermeasures shall manage first-failure situations.
Derived from	
Partly Supported	Requirements on gass emission represented using regular requirements Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0008b RESS over-current interruption	
Alias	ISO 6469-1 / RESS over-current interruption
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for the interruption of RESS over-current. If a RESS system is not short-circuit proof in itself, a RESS over-current interruption device shall open the RESS circuit under conditions specified by the vehicle and/or RESS manufacturer,
Derived from	
Partly Supported	Requirements on overcurrent interrupt can be represented using regular requirements. Current monitoring can be represented using regular FDA and HDA constructs. Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0009b Crash-test requirements	
Alias	ISO 6469-1 / Crash-test requirements
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account specific RESS crash-test requirements. The following requirements shall be met in a crash test, in accordance with the test requirements of applicable National and/or International Standards or regulations or standards: a) If the RESS is located outside the passenger compartment, it shall not penetrate into the passenger compartment. b) If the RESS is located inside the passenger compartment, movement of the RESS shall be limited to ensure the safety of the occupants. c) No spilled electrolyte shall enter the passenger compartment during and after the test.
Derived from	
Partly	Requirements on crash aspects can be represented using regular

Supported	requirements. Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0010b Power-on procedure	
Alias	ISO 6469-2 / Power-on procedure
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements on power-on/power off procedure. At least two deliberate, distinct actions shall be performed in order to go from the “power-off” mode to the “driving enabled” mode. a) Power-off: the propulsion system is off; no active driving of the vehicle is possible in this mode. b) Driving enabled: only in this mode will the vehicle move when the accelerator device is applied. After an automatic or manual turn-off of the propulsion system, it shall only be possible to reactivate the system by the specified power-on procedure.
Derived from	
Partly Supported	Requirements on power on/off procedure can be represented using regular requirements. Modes can be used to manage requirements validity in different modes. Formalization of behavior can be made using behaavioural constructs including. Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0011b Propulsion system status indication	
Alias	ISO 6469-2 / Propulsion system status indication
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for the indication of the propulsion system status. An obvious device (e.g. a visual or audible signal) shall indicate permanently or temporarily that the propulsion system is ready for driving.
Derived from	
Partly Supported	Requirements on status indication can be represented using regular requirements. Specification of the indication can be made using regular FDA and HDA constructs. Methodology:

	The identified concerns are managed in the FEV methodology swimlane

CRF#0012b Connection to power supply	
Alias	ISO 6469-2 / Connection to power supply
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for the connection of the vehicle to an off-board electric power supply. Vehicle movement by its own propulsion system shall be impossible when the vehicle is physically connected to an external electrical network (e.g. mains, off-board charger).
Derived from	
Partly Supported	Requirements on power supply connection restrictions can be represented using regular requirements. Specification of inhibitor, etc. can be represented using regular FDA and HDA constructs. Methodology: The identified concerns are managed in the FEV methodology swimlane

CRF#0013b RESS state indications	
Alias	ISO 6469-2 / RESS state indications
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for the indication of reduced power and low energy content of RESS. If the power is automatically reduced to a significant extent (e.g. by high temperature of the power unit or of the energy source component), this shall be indicated to the driver by an obvious device such as a visual or audible signal. A low state of charge of the traction battery shall be indicated to the driver by an obvious device. At the indicated low state of charge specified by the vehicle manufacturer, the vehicle shall meet the following requirements: a) It shall be possible to move the vehicle out of the traffic area by its own propulsion system. b) A minimum energy reserve shall still be available for the lighting system as required by national and/or international standards or regulations, when there is no independent energy storage for the auxiliary electrical circuit.
Derived from	
Partly Supported	Requirements on RESS and low SoC degradation can be represented using regular requirements. Specification of the monitoring and control system can be represented using regular FDA and HDA constructs. Methodology:

	The identified concerns are managed in the FEV methodology swimlane

CRF#0014b Driving backward	
Alias	ISO 6469-2 / Driving backward
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for driving backward. If driving backwards is achieved by reversing the rotational direction of the electric motor, the following requirements shall be met to prevent unintentional switching into reverse when the vehicle is in motion: a) switching between the forward and backward (reverse) directions shall require either two separate actions by the driver, or b) if only one driver action is required, a safety device shall allow the transition only when the vehicle is stationary or moving slowly. The maximum reverse speed shall be limited.
Derived from	

CRF#0015b Parking	
Alias	ISO 6469-2 / Parking
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for parking. When leaving the vehicle, the driver shall be informed by an obvious device (e.g. a visual or audible signal) if the propulsion system is still in the driving enabled mode. If the electric motor continues to rotate when the vehicle is stationary, no unintended movement of the vehicle shall be possible after switching to the power-off mode.
Derived from	

CRF#0016b Electromagnetic compatibility	
Alias	ISO 6469-2 / Electromagnetic compatibility
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for electromagnetic susceptibility and emissions. The electric road vehicle shall be tested for susceptibility according to ISO 11451-2. The reference field strength shall be a minimum of 30 V/m rms or according to national standards or regulations. Care shall be taken to minimize electromagnetic emissions from the electric road vehicle, taking into account national standards or regulations and

	international standards. Vehicle functions enabled by the auxiliary circuits shall meet the relevant national and/or international standards or regulations during operation of the vehicle, particularly those related to lighting, signalling and safety functions.
Derived from	

CRF#0017b Protection against failure	
Alias	ISO 6469-2 / Protection against failure
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for fail-safe design, first failure response and unintentional vehicle behaviour. Unintentional acceleration, deceleration and reversal of the propulsion system shall be prevented. In the event of a single failure (e.g. in the power control unit) of a stationary, unbraked vehicle, the propulsion shall be cut off to prevent unintended vehicle movement. Unintended steering effects from different torques while driving or braking that are greater than those of IC enginepropelled vehicles shall not occur.
Derived from	

CRF#0018b Emergency response	
Alias	ISO 6469-2 / Emergency response
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for emergency response. The manufacturer of the vehicle shall have information available for safety personnel and/or emergency responders with regard to dealing with accidents involving a vehicle.
Derived from	

CRF#0019b Marking	
Alias	ISO 6469-3 and UNECE R100 / Marking
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for marking high voltage components and high voltage wiring. The outer covering of cables and harness for high voltage circuits, not within enclosures or behind barriers shall be marked with orange colour.
Derived from	

CRF#0020b Protection against electric shock

Alias	ISO 6469-3 and UNECE R100 / Protection against electric shock
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requiremements for basic protection measures and protection under first-failure conditions against elecrtic shock
Derived from	

CRF#0021b Insulation	
Alias	ISO 6469-3 / Insulation
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for insulation of high voltage live parts. If protection is provided by insulation, the live parts of the electric system shall be totally encapsulated by insulation which can be removed only by destruction. The insulating material shall be suitable to the maximum working voltage and temperature ratings of the vehicle and its systems. The insulation shall have sufficient withstand voltage capability.
Derived from	

CRF#0022b Barriers and enclosures	
Alias	ISO 6469-3 / Barriers and enclosures
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requiremements for barriers and enclosures to prevent electrical shock. If protection is provided by barriers/enclosures, live parts shall be placed inside enclosures or behind barriers, preventing access to the live parts from any usual direction of access. The barriers/enclosures shall provide sufficient mechanical resistance under normal operating conditions, as specified by the manufacturer. If barriers/enclosures are accessible directly they shall be opened or removed only by use of tools or maintenance keys or they shall have means to deactivate live parts with high voltage, e.g. interlock.
Derived from	

CRF#0023b Isolation resistance	
Alias	ISO 6469-3 and UNECE R100 / Isolation resistance
Status	Approved
Type	«Safety»
Priority	High

Description	The east-adl approach shall take into account requirements for the isolation resistance of the high voltage systems. If the protection measures chosen (see 7.3) require a minimum isolation resistance, it shall be at least 100 O/V for d.c. circuits and at least 500 O/V for a.c. circuits. The reference shall be the maximum working voltage.
Derived from	

CRF#0024b Withstand voltage	
Alias	ISO 6469-3 / Withstand voltage
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for withstand voltage capability of the high voltage components and wiring. The high voltage components and wiring shall fulfill the applicable sections of IEC 60664-1 or meet the withstand voltage capability according to the withstand voltage test described.
Derived from	

CRF#0025b Potential equalization	
Alias	ISO 6469-3 and UNECE R100 / Potential equalization
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for components and path for the potential equalization. All components forming the potential equalization current path (conductors, connections) shall withstand the maximum first failure current in a maximum fault clearance time. The resistance of the potential equalization path between any two exposed conductive parts of the high voltage electric circuit which can be touched simultaneously by a person shall not exceed 0,1 ?.
Derived from	

CRF#0026b Charging inlet	
Alias	ISO 6469-3 and UNECE R100 / Charging inlet
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for the vehicle charging inlet. One second after having disconnected the charge coupler, the voltage of the vehicle inlet shall be less than or equal to 30 V a.c. or 60 V d.c.. This condition is not necessary if vehicle inlet complies with the requirement of at least IPXXB.
Derived from	

CRF#0027b Isolation resistance test	
Alias	ISO 6469-3 and UNECE R100/ Isolation resistance test

Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements and procedures for the isolation resistance test
Derived from	

CRF#0028b Withstand voltage test	
Alias	ISO 6469-3 / Withstand voltage test
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements and procedures for withstand voltage capability test
Derived from	

CRF#0029b Potential equalization test	
Alias	ISO 6469-3 / Potential equalization test
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements and procedure for the potential equalization components and path test
Derived from	

CRF#0030b Protection against electric shock after crash test	
Alias	R94 new EV proposals and R95 new EV proposals / Protection against electric shock after crash test
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for protection of persons against electric shock after vehicle crash test
Derived from	

CRF#0031b Electrolyte spillage after crash test	
Alias	R94 new EV proposals and R95 new EV proposals / Electrolyte spillage after crash test
Status	Approved
Type	«Safety»
Priority	High

Description	The east-adl approach shall take into account requirements for electrolyte spillage after vehicle crash test
Derived from	

CRF#0032b RESS retention after crash test	
Alias	R94 new EV proposals and R95 new EV proposals / RESS retention after crash test
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements for RESS retention after vehicle crash test
Derived from	

CRF#0033b Test for protection against electric shock after crash test	
Alias	R94 new EV proposals and R95 new EV proposals / Test for protection against electric shock after crash test
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements and procedure for protection against electric shock test after vehicle crash test
Derived from	

CRF#0034b Test for electrolyte spillage after crash test	
Alias	R94 new EV proposals and R95 new EV proposals / Test for electrolyte spillage after crash test
Status	Approved
Type	«Safety»
Priority	High
Description	The east-adl approach shall take into account requirements and procedure for electrolyte spillage test after vehicle crash test
Derived from	

CRF#0035b test for RESS retention after crash test	
Alias	R94 new EV proposals and R95 new EV proposals / test for RESS retention after crash test
Status	Approved
Type	«Non-Function»
Priority	High
Description	The east-adl approach shall take into account requirements and procedure for RESS retention test after vehicle crash test
Derived from	

CRF#0046b SEooC	
Alias	ISO 26262 / SEooC
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support the ISO 26262 SEooC concept
Derived from	

CRF#0047b hazard analysis and risk assessment	
Alias	ISO 26262 - 3 / hazard analysis and risk assessment
Status	Implemented
Type	«Safety»
Priority	High
Description	Maenad approach shall support ISO 26262 hazard analysis and risk assessment
Derived from	

CRF#0048b ASIL determination	
Alias	ISO 26262 - 3 / ASIL determination
Status	Implemented
Type	«Safety»
Priority	High
Description	Maenad approach shall support ISO 26262 ASIL determination
Derived from	

CRF#0049b Safety Goal	
Alias	ISO 26262 - 3 / Safety Goal
Status	Implemented
Type	«Safety»
Priority	High
Description	Maenad approach shall support Safety Goal and safe state definition
Derived from	

CRF#0050b External measures	
Alias	ISO 26262 - 3 / External measures
Status	Approved

Type	«Safety»
Priority	High
Description	Maenad approach shall support external measures definition
Derived from	

CRF#0051b functional safety requirements	
Alias	ISO 26262 - 3 / functional safety requirements
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support ISO 26262 functional safety requirements definition, including all necessary parameters (Operating modes, fault tolerant time interval, eventually safe state, emergency operation interval, functional redundancies)
Derived from	

CRF#0052b functional safety requirements allocation	
Alias	ISO 26262 - 3 / functional safety requirements allocation
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support ISO 26262 functional safety requirements allocation
Derived from	

CRF#0053b technical safety requirements	
Alias	ISO 26262 - 4 / technical safety requirements
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support ISO 26262 technical safety requirements definition
Derived from	

CRF#0054b safety mechanism	
Alias	ISO 26262 - 4 / safety mechanism
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support ISO 26262 safety mechanism definition

Derived from	

CRF#0055b latent faults	
Alias	ISO 26262 - 4 / latent faults
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support safety mechanism definition to avoid latent faults
Derived from	

CRF#0056b random hw failures	
Alias	ISO 26262 - 4 / random hw failures
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support safety mechanism definition to avoid random hw faults
Derived from	

CRF#0057b systematic failures	
Alias	ISO 26262 - 4 / systematic failures
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support safety mechanism definition to avoid systematic faults
Derived from	

CRF#0058b ASIL Decomposition	
Alias	ISO 26262 - 9 / ASIL Decomposition
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support ASIL decomposition
Derived from	

CRF#0059b Safety case	
Alias	ISO 26262 /Safety case

Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support Safety case specification
Derived from	

CRF#0061b functional safety assessmnet	
Alias	ISO 26262 - 2 / functional safety assessmnet
Status	Approved
Type	«Safety»
Priority	High
Description	Maenad approach shall support functional safety assessment
Derived from	

DOW#2000 Architectural Patterns	
Alias	
Status	Approved
Type	«Non-Function»
Priority	Medium
Description	Standard architectural patterns for optimization and refinement shall be defined
Derived from	• WP3 • DOW#0015 O3-2

UOH#0001 Error_Model_Analysis_Support	
Alias	Error_Model_Analysis_Support
Status	Implemented
Type	«Safety»
Priority	High
Description	The EAST-ADL error model should fully support the necessary concepts to allow dependability analysis, including safety requirements/constraints (e.g. ASILs).
Derived from	• DOW#0004 O1: Develop capabilities for modelling and analysis support, following ISO 26262