Grant FFI 2013-01296

Synligare utveckling av inbyggda fordonssystem – visualiserad kravhantering och samverkan

Visibler Development of Embedded Automotive Systems - Visualised Requirements Management and Collaboration

SYNLIGARE

Report type	**Deliverable D2.1**
Report name	**Report on modelling concepts and methods**

Dissemination level	**Public**
Status	**Final Release**
Version number	**2.0**
Date of preparation	**2016-05-20**

Authors

Editor	E-mail
Urban Ingelsson	urban.ingelsson@semcon.com

Authors	E-mail
Henrik Lönn	henrik.lonn@volvo.com
Henrik Kaijser	henrik.kaijser@volvo.com
Magnus Skoog	magnus.skoog@autoliv.com
Mikael Stolpe	mikael.stolpe@arccore.com
Urban Ingelsson	urban.ingelsson@semcon.com

The Consortium

Volvo Technology Corporation AB - Autoliv AB - Arccore AB- Systemite AB - Semcon Sweden AB

Table of contents

1 Terminology

Below is a set of terms used in project Synligare. The list largely excludes meta-model elements from AUTOSAR and EAST-ADL, as these are defined in the respective specification. Most of this terminology was defined in Synligare Deliverable D1.1 [1].

Term	Description
Abstract view	A subset of a system description
AUTOSAR	AUTomotive Open System Architecture
Artifact	A general term for engineering elements that compose a solution
ARXML	AUTOSAR exchange format
Component	A general term for software, hardware and functional components
Concrete view	The information of an abstract view, as it is shown to a software tool user. For example, it can have layout, coloring, etc.
DIA	Development Interface Agreement
EAST-ADL	Automotive Architecture Description Language
EAXML	EAST-ADL exchange format
Element	A general term for parts of a model.
Embedded system	Software and electronics embedded in a product
FROP	Feature Roll Out Plan
Function Point	A measurement of business functionality
GraphML	Graph exchange format
Legacy tools	Tools that already exist in a specific context
Metric	A quantification of aspect(s) of a design, system, project, organization , etc.
OEM	Original Equipment Manufacturer
Realize	Relationship between requirement and system model element
Requirement	A singular documented physical and functional need that a particular design, product or process must be or do
Requirement management	The activities of documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders.
RFI	Request for Information
RFQ	Request for Quotation
Satisfy	Relationship between requirements
SEooC	Safety Element out of Context
SIL	Safety Integrity Level
Synligare	FFI Project Grant FFI 2013-01296
System Model	A set of elements and relations that represents a system
TRL	Technology Readiness Level, a measure introduced by NASA to characterize technology maturity from 1 (initial concept) to 9 (deployed technology)
Tier-n Supplier	A supplier that provides components directly to an OEM (Tier-1) or Tier-i

	supplier (tier-i+1)
User Stories	Short description of functionality in informal language
View	A representation of a whole system (or aspects thereof) from the perspective of a related set of concerns
Viewpoint	A specification of the conventions for constructing and using a view
WP	Work Package, one of five parts of Project Synligare, e.g. WP2
Work product	The tangible result of a particular methodology or process step
Well-formed	The property of a model that it complies with applicable syntactical rules
XML	Extensible Markup Language

2 Introduction

This document is the report on results of Work Package 2 (WP2) of Project Synligare. This work package and subsequently this report describe conceptual ways to address the needs that were identified in D1.1.

In Project Synligare Deliverable D1.1 [1], the increasing complexity of specifications for automotive embedded systems and the impact of this complexity on collaborative development were identified as the problems that Project Synligare is to address. The key idea behind Project Synligare is to provide overview and to ease exchange of system specification by means of defining structured information models for representing system specifications. The structured information models will enable us to keep the system specification in the form of a system model, and define viewpoints and metrics as well as algorithms for automatically generating them. The viewpoints and metrics will provide the overview of the system model to the engineers developing the embedded system.

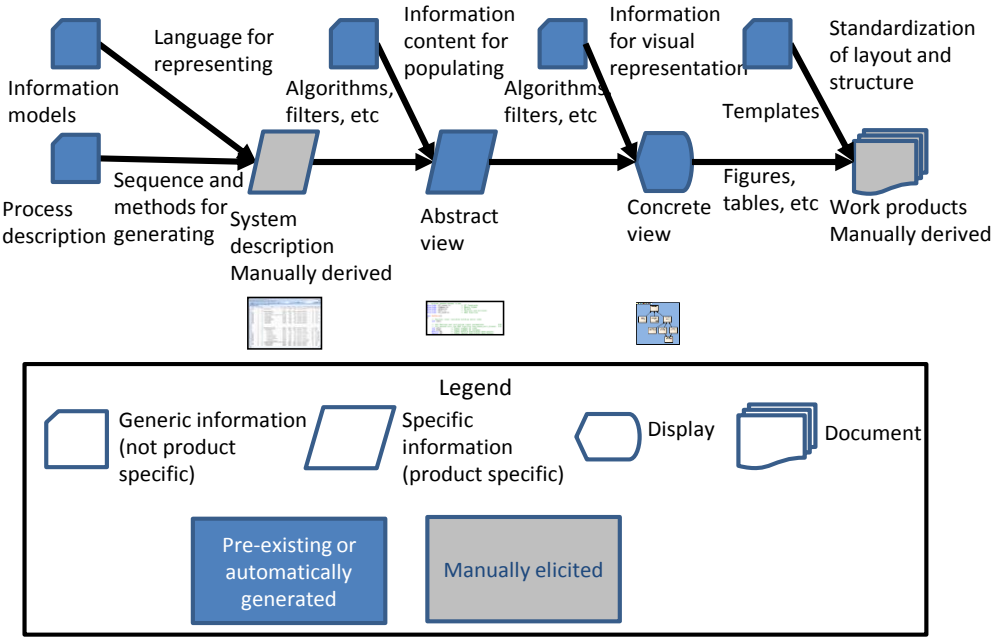

Figure 1 How information models relate to system description, views and viewpoints

It can be seen in Figure 1 how information models, process descriptions, templates and algorithms for generating graphs are not specific to the product. These types of information can be used by an organization to standardize how they conduct their system development projects to ensure compatibility with various tools and quality systems, to gain various benefits. For example, it can be beneficial to purposefully structure information to be easily communicated into a supplier's or a customer's system. **Information models** describe how the system model should be represented, i.e. what components, relationships and attributes there should be**. Process descriptions** describe in what order and by what method to generate the system model as well as criteria for passing milestones in the development project. An **abstract view** is a subset of the system model selected to support a particular activity. A view or **concrete view** contains the same information as the abstract view, but is represented to a stakeholder to give a user-friendly overview over the information to support the activity. Any view is created and populated with the information from the abstract view through a **viewpoint**, which is a specification of the conventions for constructing and using a view. A viewpoint can be a combination of several viewpoints, such as a combination of a

table and a diagram, where there is a viewpoint for creating and populating tables and a viewpoint for creating and populating diagrams. There are many aspects of the accessibility and visualization of information that can greatly contribute to understanding of a complex system. Similarly, there are **metrics**, i.e. values calculated from the system model to describe progress, performance, quality or other quantifiable aspect of the system. It is expected that **algorithms** or rule sets are required as part of the viewpoint to generate (or draw) the concrete view from the abstract view. However, these algorithms can be generic (they do not need to be system-specific), as long as the abstract view (and the rest of the system model) is represented in accordance with the information model. Typically, the concrete view is a set of graphs, tables and other illustrations. With the help of particular viewpoints such as **templates**, these graphs and tables can be used in **work products** such as the formal documentation of the system under development. It should be noted that describing the system, i.e. eliciting the system description, is a manual process. When information has been entered into the system description as specified in the information models, generation of concrete views can be automated by algorithmic viewpoints that assume information structured according to the information models. The assembly of figures and tables into work products is also a manual process, but is a good example of an activity that is made easier by automatic generation of concrete views.

Defining structured information models involves determining what information carriers (components, relationships and attributes) are required in the various design and verification activities. It should be considered what algorithms are needed to be carried out, like searches and sorting, to arrive at an information model that serves well to represent system specifications. A starting point is to consider the existing specification languages like EAST-ADL. Further, it should be considered how to do management of specifications and requirements in accordance with ISO 26262. The aim is that the outcomes of Project Synligare should be tried on an example system, to validate the structured information models, the viewpoints and the metrics.

The tool platforms that are considered in this work package are SystemWeaver, Enterprise Architect and EATOP. SystemWeaver is a modelling tool that can process EAST-ADL. Enterprise Architect is another modelling tool, originally for UML and SYSML. EATOP is an open source tool based on Eclipse, which uses EAST-ADL as the underlying information model.

This chapter introduces WP2 by defining purpose, scope and method.

2.1 Purpose

The work in WP2 will define structured information models that enable communication of system models in collaborative development of automotive embedded systems with the help of viewpoints and metrics that bridge complexity. In other words, WP2 will develop a representation for system models for communication between OEM and Tier-1 (marked as “information model” in Figure 1), as well as conceptual solutions for visualizing information from the system model (marked as “algorithms” in Figure 1). The conceptual solutions are to be handed on to WP3, for development of corresponding prototype tools. With the help of those prototype tools, the conceptual solutions will be validated with respect to an example system in WP4. The expected effect of having appropriate structured information models for specification and appropriate viewpoints and metrics is increased predictability of safety, quality and performance of the specified system. Further it is expected that with a common understanding of how the specification is represented, visualized and measured, communication of specification should involve less misunderstanding. The collaboration aspect shall be considered such that a specification according to the structured information models can be modularized and communicated appropriately.

2.2 Scope

The scope of WP2 is defined by the following categories of viewpoints and metrics (that were identified WP1 and reported in D1.1 [1]).

Table 1 View Categories

Topic	Viewpoint / Tool Support	Section in this report	Comment
All Elements	Generic Tree Viewpoint	7.1	Provides generic hierarchical view of all elements and their hierarchical relationships
	Generic Properties Viewpoint	7.2	Provides a table to relate the attributes of any small set of selected modelling elements.
	Generic Version Information Viewpoint	7.3	User interface to show and modify version information on model element level.
	Generic Table Viewpoint	7.4	Enables construction of tables that shows elements, their properties and attributes, including attributes inherited through relationships.
	Generic Diagram Viewpoint	7.5	Enables construction and viewing of various diagrams, showing hierarchy of and interconnectivity between elements.
	Generic Chart Viewpoint	7.6	Enables viewing of information in the form of charts, aggregating from metrics
	Create Connector Support	7.7	Tool support for selecting ports and creating modelling elements for the connectors between the ports
	Compare and Merge Support and Viewpoint	7.8	Aid in comparing two models, highlighting differences and allowing choices to be made as part of the process of merging two models
	Pilot View of Metrics	7.21	A combination of viewpoints that each illustrate a metric, by pie charts, etc.
Requirements	Requirements and Features Viewpoint	7.10	Requirements grouped by which feature they are (indirectly) allocated to.
	Requirements and Functions Viewpoint	7.11	Requirements grouped by which function they are (indirectly) allocated to
	Requirements and Components Viewpoint	7.12	Requirement grouped by which component they are (indirectly) allocated to
	Combined Viewpoint of Requirements and Architectural Components	7.16	A diagram representation of requirements and subsets of the architectural modelling on selected abstraction levels along with allocation components.
	Combined viewpoint for multiple aspects of requirements according to selected order and sorting	7.13, 7.16, 7.18, 7.19	Gives typical applications of selection of requirements allocated to a subsystem or having a particular attribute, which can be extended to encompass other applications of selection and sorting.
Safety	Safety Goals Viewpoint	7.18	Shows all safety goals for a given item along with the boundaries of the safety-critical system.

Topic	Viewpoint / Tool Support	Section in this report	Comment
	Functional Safety Concept Viewpoint	7.19	Visualizes what is designed to meet the safety goals, including functional safety requirements and functional modelling elements
	Technical Safety Concept Viewpoint	7.19	Visualizes how to meet the functional safety concept with technical design choices, including technical design modelling elements and hardware safety requirements and software safety requirements
	Safety Analysis Tool Support and Viewpoint	7.20	Error propagation modelling and fault tree analysis as a group of integrated viewpoints
System and Software Architecture	Feature Realization Viewpoint	7.15	Highlighting the modelling elements in a system model that realize a given feature.
	Function to Architecture Allocation Viewpoint	7.14	A table with architectural elements as rows and functions in columns.
	Software Architecture Viewpoint	7.17	Visualization of a given distributed software architecture with software components, ports and signals.
	Feature Tree Viewpoint	7.9	A variability breakdown of features
Progress	Requirements Coverage Viewpoint	7.21	For visualizing metrics, such as requirements coverage
	Elements Coverage Viewpoint	7.21	For visualizing metrics, such as elements coverage

Table 2 Metric Categories

Topic	Metric	Section in this report	Comment
Product Metric	Dependability	6.6	E.g. failure rate
	Impact of system model on production volume, income and cost	6.6	E.g. take rate of a feature configuration
Progress Metric	Number of completed analyses / artefacts / reviews	6.1, 6.2, 6.3, 6.5	Provides typical use examples of a metric to follow-up progress. E.g. progress on safety-related artefacts, requirement allocation progress or realization progress.
	Requirements Validation	6.1	Comparing a package of validated requirements with a package of not-yet-validated requirements
	Verification Progress	6.4	Counting the number of performed verification cases as compared to the full set of verification cases

2.2.1 Further scope

The viewpoint categories and metric categories listed above are addressed to ensure that the structured information models that are to be defined can support generation of those viewpoints and metrics. Information modelling results are presented in Chapter 3 of this document.

Given the focus on interaction between OEM and Tier-1, the interaction and exchanges between OEM and Tier-1 in a collaborative development project is described in Chapter 5 of this document. The viewpoints and metrics that are relevant for the RFQ-process are prioritized to focus on finding solutions that support collaborative system development.

The scope includes ensuring that the structured information models are adequate to specify an automotive embedded system.

2.3 Document Outline

The remainder of this report is structured as follows. Chapter 3 presents information modelling aspects. Chapter 5 discusses the interaction between OEM and Tier-1. Chapter 6 presents metrics as conceptual solutions. Chapter 7 presents viewpoints as conceptual solutions. Finally, Chapter 8 summarizes the report and Chapter 9.1 gives the bibliography.

3 Information Modelling

This chapter reports on the information models used. In order to handle a full scale model we need a model structure that matches the needs. Such needs are:

- Model Data Storage
- Model distribution
- Simplification of version control
- Simplification of model exchange and synchronization

3.1 Need: Model Data Storage

There are basically two alternatives with regard to where and how to store model data:

- Store model in a common database
- Use files for data exchange

There are advantages in using a common database, such as:

+ Models are in sync
+ No need for export/import
+ Faster workflow

But the disadvantages are:

- Need for real-time database performance with high performance database server
- Needs for a database manager and database is hard to set up
- Who owns the database?
- What about models that a part do not want to share?
- In the case of this project, which uses EAST-ADL as the structured information model and EATop as one of the software tools, it is a disadvantage that EATop cannot work with a data base, it is file-based

Therefor the project has decided to use a file-based approach.

3.2 Need: Model Distribution

The concept of model distribution concerns the capability to have a model on which several organisations or teams can work in parallel. The models have a common purpose, but different parties work on different parts. This is key in developing subsystems and in collaborative development, because involved organisations may want to withhold some information in the exchanges.

It is envisioned that the model is defined as the full system/sub-system model and that the OEM and the Tier-1 has different perspectives and different needs. In contrast, traditionally no models are shared or distributed. Only customer requirements are sent from the OEM to the Tier-1.

Consequently, sharing models leads to new needs, like:

- Model Integrity. Need for share and hide parts
- Parallel development with different versions of models
- Handling of shared interfaces

3.2.1 Model Integrity - Simplification of Model Separation

We define Model Separation as the distribution of a model over several files. Model Separation is not supported by EAST-ADL. The whole model is normally put into one file. Even as Model Separation is not explicitly supported in EAST-ADL, it is possible to find an work-around and thus obtain important functionality like:

- Hide model parts – These parts are not round-tripped. It stays within the local models of OEM/Tier-1. Reasons for this may be commercial or only due to the fact that OEM/Tier-1 is not interested in details that is out of their concern.
- Read only part – These parts are imported. They shall not be updated in the tool into which the model is imported. At subsequent imports of updates, read only information is overwritten. Read only information is not round-tripped.

In particular, it is useful to separate models according to the abstraction levels.

A problem has been identified. In the case of this project, that employs EAST-ADL as the structured information model, a low abstraction level such as the implementation level cannot stand alone as a separate model, since much relevant information is contained in higher abstraction levels. In EAST-ADL, as in most modelling languages, a key idea is to store information only once and make it available where it is needed by allowing a software tool to follow traceability links. If the information is stored on one abstraction level, then there are other abstraction levels that cannot be seen as complete models in themselves.

3.2.2 Parallel Development and Simplification of Version Control

Version control is important for exchange of information, to be able to refer to a history of system development. In particular, it is important to be able to determine if the shared information is up to date and synchronized. To make this simpler, a system model should be subject to a structure that allows parts of the model to be separate configuration items. Furthermore, version information is useful when searching/filtering information, when showing differences between models, when managing reuse and when planning.

Version control is not supported explicitly by EAST-ADL. It can be handled by putting all files into version control and to be strict in structure and naming. A solution for version control that employs strict structure and naming would name files according to a predefined formula that contains configuration item identifier and version number.

Handling of parallel development is more complex. When the OEM exports a version of the model to the Tier-1 it will take time before the Tier-1 exports an updated model. When the OEM is importing this there might be conflicts due to new updates. This means that the OEM must apply some of these techniques:

- Stop working until Tier-1 has responded
- Branch model and import updates to both branches
- Merge updates when they arrive

In Project Synligare we describe a way to keep version information for each modelling element by employing a user defined attribute and a tool feature to view and increment version information for a given element.

3.2.3 Handling of Shared Interfaces

EAST-ADL cannot strictly separate interfaces. They are a part of the design models. When a Tier-1 updates an interface it will lead to an inconsistent model. The solution for this problem is likely not

in the definition of the structured information model. These types of changes must be communicated and negotiated by other means.

3.3 Need: Model Exchange

These are typical model exchange scenarios:

- A typical scenario for exchanging models is export/import/sync. This means that a model is exported from tool A and imported to tool B. If tool B already contains the model the local model is synchronized during import. This simple scenario does not need any merging.
- A more complex scenario is round-trip. This scenario means that the model is re-imported. The re-imported model replaces the previous model, so merge is not needed.
- The most complex scenario is model merging. In this scenario the user chooses which elements that are merged into the existing model and which parts that are not merged.

In a round-trip scenario the UUID is needed. The package path is not enough. The reason is that there may already be local instances of a given element or package in the importing tool. Unless this is well handled using UUID, the result of renaming or moving an element may be that there is duplication of the given element or package.

3.4 EAST-ADL is an Example of a Structured Information Model

As an example of a structured information model, this project considers EAST-ADL. Here follows an introduction to EAST-ADL.

According to EAST-ADL models are put into four different architectural levels. The architectural levels are integrated and cannot be separated into stand-alone models. Consequently, EAST-ADL does not meet the need of model separation into architectural levels as is mentioned in Section 3.2.1.

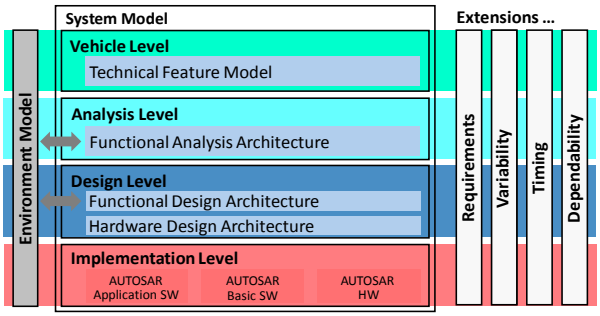

Figure 2 - EAST-ADL's breakdown in abstraction levels (vertically) and in core system model, environment and extensions (horizontally).

The functionality of the vehicle is described at different levels of abstraction and in different parts.

The four abstraction levels covered by the EAST-ADL are:

- **Vehicle Level**
  Feature trees characterizing the vehicle content as it is perceived externally.
- **Analysis Level**
  Abstract functional architecture defining systems from a functional point of view.
- **Design Level**
  Detailed functional architecture allocated to hardware architecture.

- **Implementation Level**
  Implementation of the embedded system represented using AUTOSAR elements

Extensions to the system model include requirements modeling, variability modeling, timing modeling for behavioral descriptions and dependability modelling including error modelling.

Each modelling element has dedicated properties and it is possible to assign further attributes to an element by relating it to an modelling element called User Defined Attribute.

EAST-ADL elements do not have a dedicated property for e.g. version, time stamp and status. Instead, User Defined Attributes can be used for this purpose. In Section 3.2.2 this is addressed by using a formal file naming convention and an external version control system.

3.5 Information Model Structure

In Project Synligare, the use of EAST-ADL and other structured information modelling is as follows.

The information is strictly separated into the following parts according to diagram in Figure 3:

- EAST-ADL full system model (package)
- EAST-ADL sub-system model(s) (sub-package)
  - In EAST-ADL, sub packages contain the function architecture and HW architecture from sub system perspective
- EAST-ADL Error model(s) (sub-package)
- Diagram models in Diagram-XML. Format is described in Section 3.7.
- AUTOSAR SW models

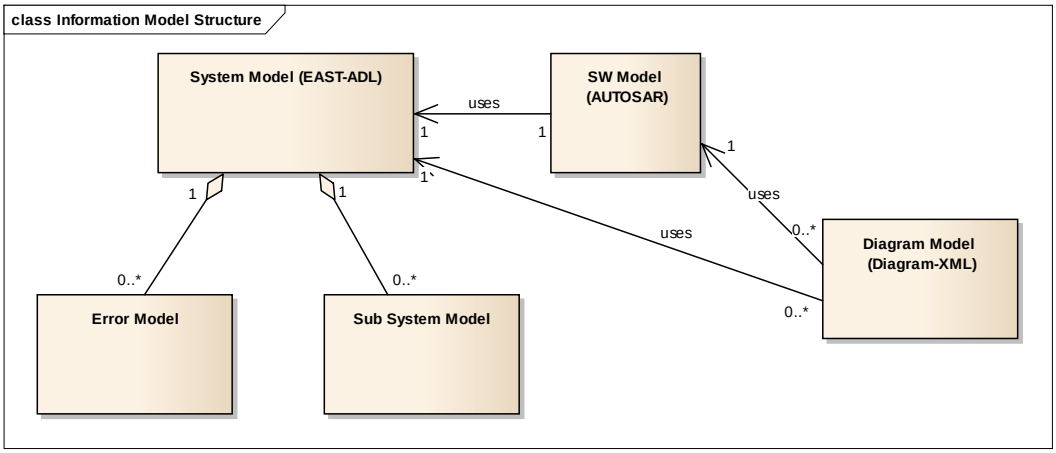

Figure 3 - Generic Model Structure

The motivation for the chosen structure is that the system model, error model and the sub-system models shall be possible to handle separately in EAST-ADL. This means that they can be loaded / saved as a separate configuration item. This can be seen as a way to meet the need of model distribution and a way to meet the need of simplification of version control. The separate configuration items can be handled separately and will have their own versioning. Further, the error model is traceable to the version of the system model for which it was generated or synchronized to.

3.6 Normative and Informative Elements

Requirements should specify what the final product should be and what it should do, its attributes and capabilities. In other words, requirements should not specify how to implement the final product. However, it does occur that textual requirements are incorrectly used to describe solutions.

A key potential benefit of model exchange is to replace textual requirements with architectural models To distinguish required content in an architectural model from example or negotiable content, elements can be marked informative or normative with a user defined attribute. The semantics for such user defined attribute should be that

- Default property is “Normative” meaning that the element must be included according to the given specification
- Children in a containment or type-prototype hierarchy inherit the property from its parent
- The element’s property overrides default property and inherited property

Figure 4 shows an example of how EAST-ADL, which does not explicitly specify how to modell normative and informative aspects, support the use of User Defined Attributes and they in turn can be employed to mark other modelling elements as normative or informative.

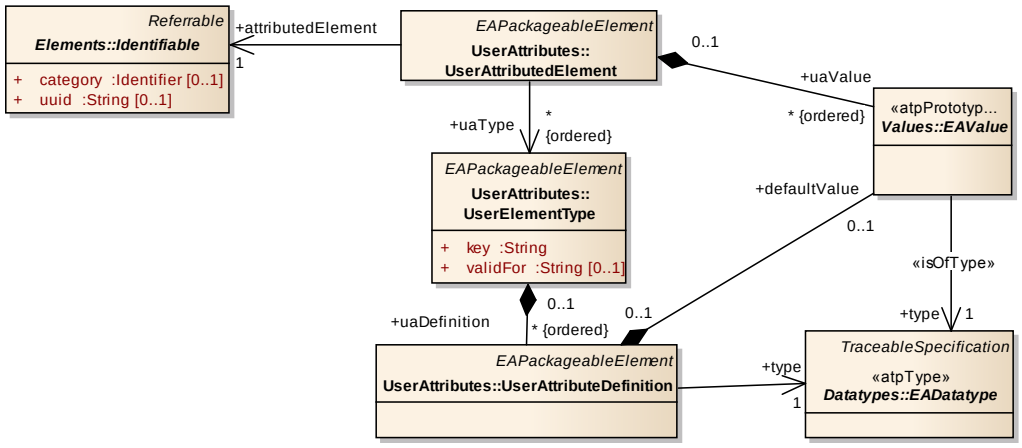

Figure 4 - The User Defined Attribute

3.7 Diagram Model

There is no information modelling for diagrams in EAST-ADL. Project Synligare has identified the need for handling graphical models as a way to communicate complex system models in an understandable way. When we use different tools we need a common format to exchange graphical information. In the project we support two similar formats for diagram exchange. The first format is a subset of the GraphML[11] extension, supported by the free yEd editor[12] supplied by the yWorks company. The second format, SGraphML, is another GraphML dialect that has been defined in the project to avoid relying on a proprietary format and make it possible to add any project specific extensions. SGraphML is strongly influenced by the yEd format, but only contains a subset of its graphical expressiveness.

In both formats, it is necessary to define references from the diagram objects to the underlying EAST-ADL model elements.

The limited GraphML diagram exchange format is shown in Figure 5. The exchange of this information is done after the EAST-ADL model exchange, but in sequence.

It is possible to import views without underlying model, but it is not recommended.

3.7.1 Implementation of GraphML Format

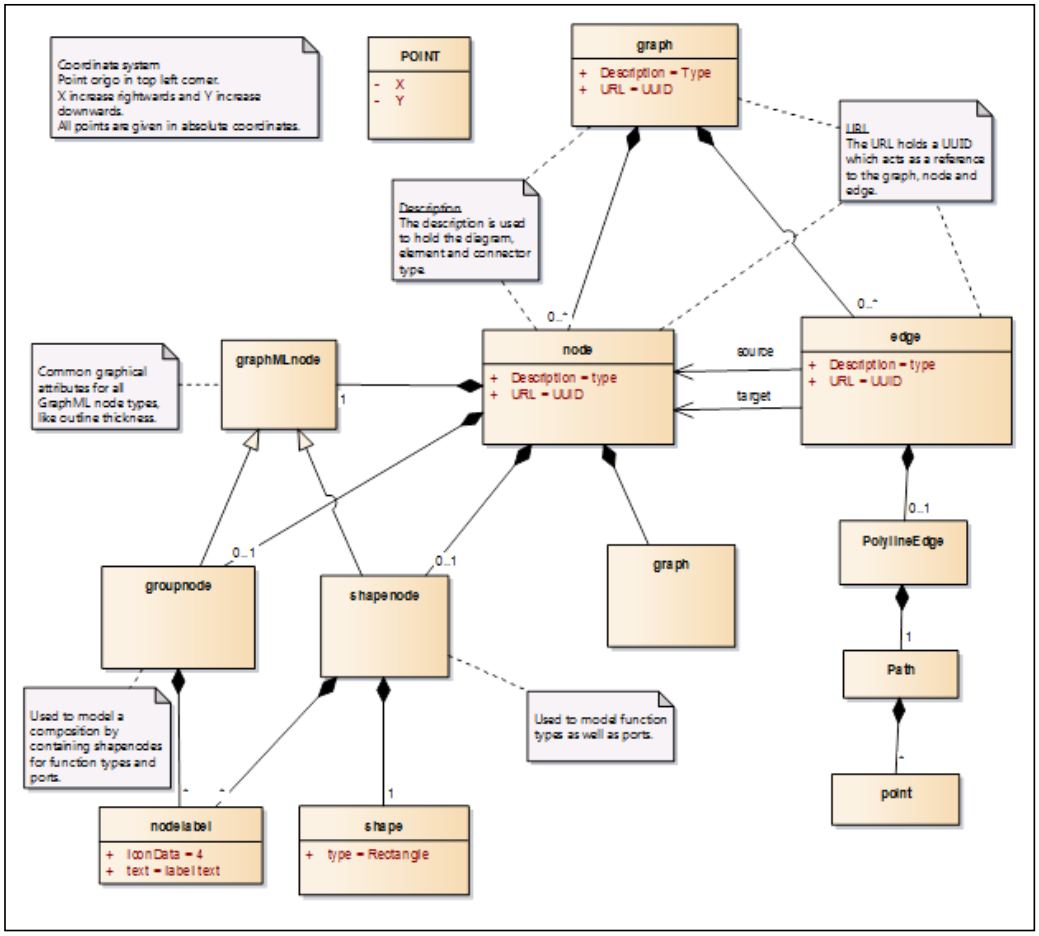

Figure 5 The subset of yEd GraphML elements used in Project Synligare

The *groupnode* is used as an invisible container element, whose graph contains *shapenodes* corresponding to a function type and port elements. This means that the ports of a function type are on the same level as the function type itself in the diagram model, i.e. ports are not nested under its container element.

Figure 6 and Figure 7 show the same diagram in two software tools, namely Enterprise Architect and yEd. The diagram was created in Enterprise Architect, exported to GraphML subset format and imported to yEd.

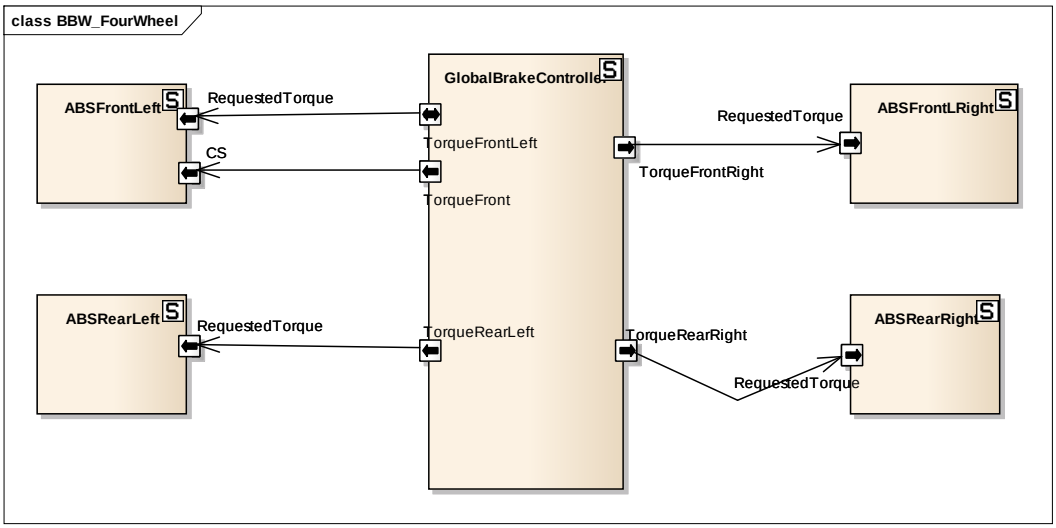

Figure 6 Diagram created in Enterprise Architect

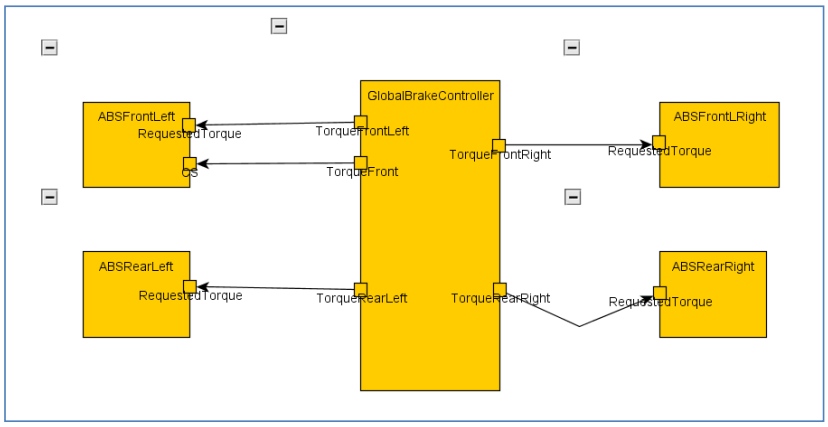

Figure 7 Diagram imported in yEd

3.7.2 SGraphML Format

Figure 8 shows a simplified version of the SGraphML metamodel. It is quite similar to the metamodel in Figure 5 but there are some differences. In this format, *groupnodes* are visible and used to model any EAST-ADL container element, like function types or packages. Function ports have their own *portnode* element and they are nested below the containing type element in the model.

Furthermore, there is a Resource class that contains various image data that are used to enrich the graphics. For example, *inports* and *outports* are illustrated with different kind of arrow images.

Instead of UUID, the reference to the underlying EAST-ADL model element is handled by package paths. This means that there is no way for the application that uses the format to make sure that the element pointed to has not been changed, so the diagram and the EAST-ADL model file needs to be manually synchronised.

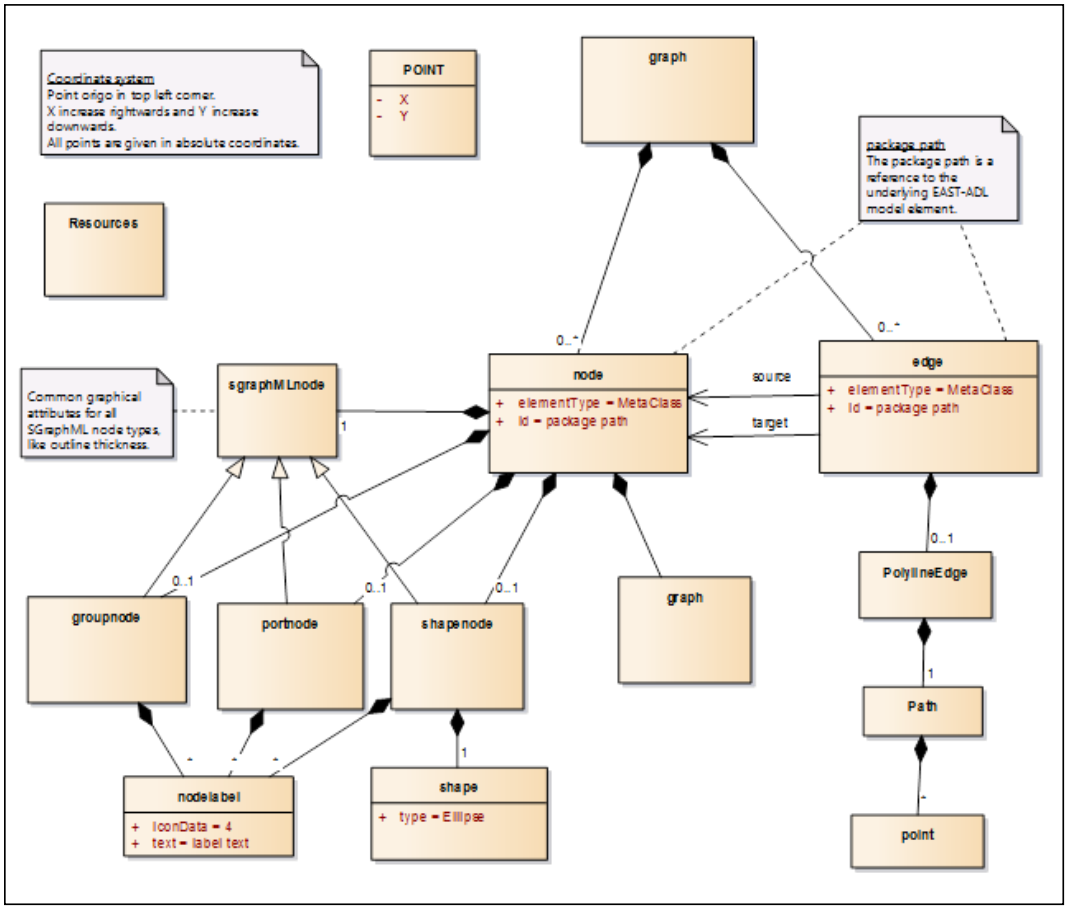

Figure 8 A simplified version of the SGraphML meta-model

Figure 9 and Figure 10 show the same SGraphML diagram in the EATOP and yEd applications. The diagram was created in EATOP and then exported to yEd format in EATOP. It is the same example as in Figure 6 and Figure 7, but different diagram representation formats.

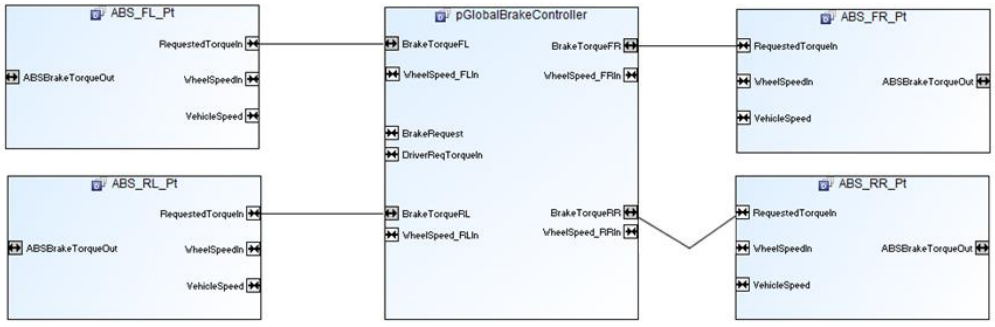

Figure 9 A diagram created in EATOP

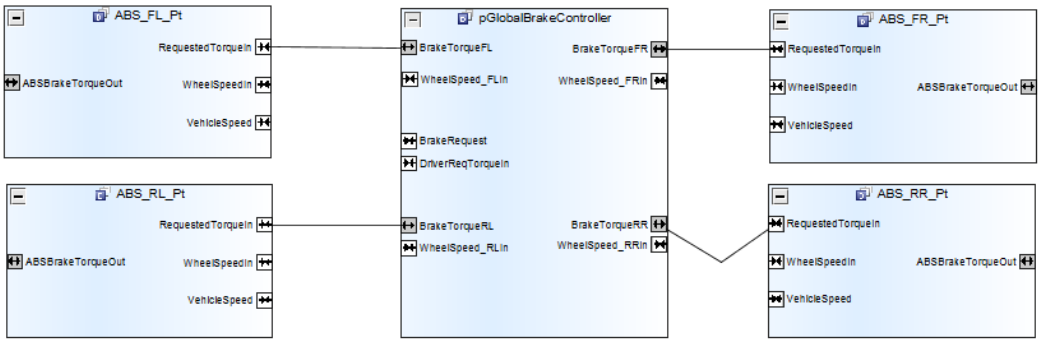

Figure 10 The diagram in Figure 9 exported from EATOP and then loaded in yEd

4 Tooling Support

The tools considered in Project Synligare are SystemWeaver (SW), EATop, ARTop and Enterprise Architect (EA).

Figure 11 shows the principles for data exchange of information models according to Section 3.5.

In different tools the models are held together in different ways:

- SystemWeaver: Model database
- EATOP: All .eaxml-files that are on the same directory are contained as a complete set of models.
- Enterprise Architect: There is a project file that refers to the individual packages that are implemented as files on the file system.

These various ways of keeping packages together or maintaining a database are considered in the project by working towards a round-trip validation experiment in which models are exchanged between organizations and the three tools named above. Thus Project Synligare will work towards having a working system of information models that withstand the second most complex scenario that is mentioned in Section 3.3.

SystemWeaver and Enterprise Architect have their own practice for version control. However, EATop does not have functionality for version control. Project Synligare is describing tool support for keeping version information for each element as a User Defined Attribute.

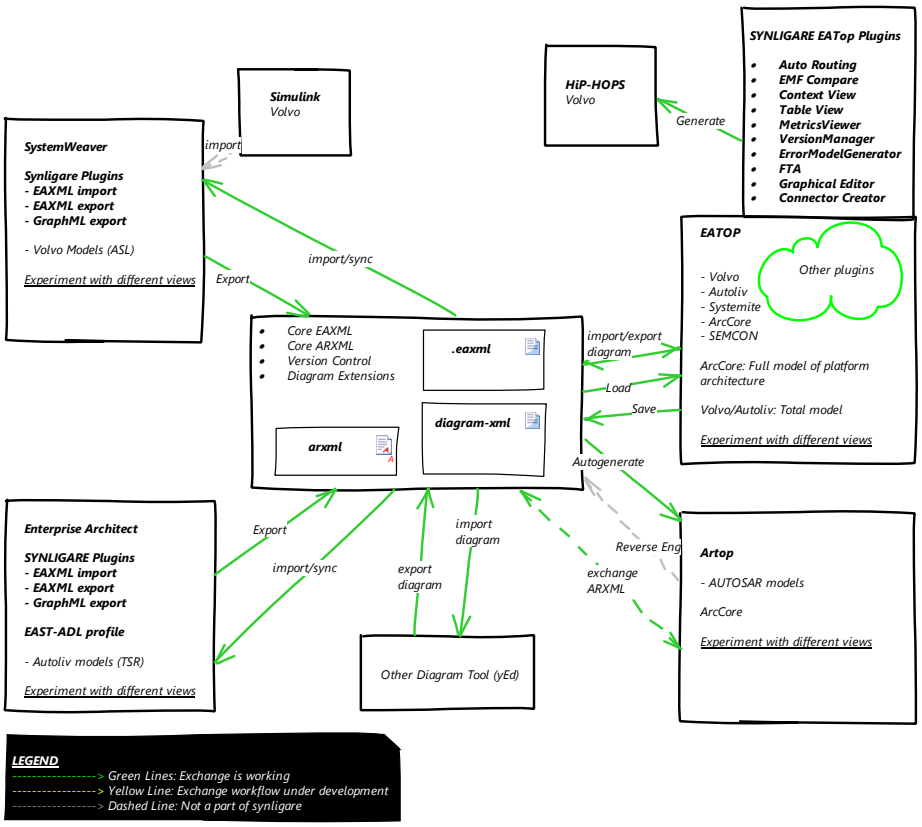

Figure 11 – Data exchange model

Figure 11 contains an example. Consider that an OEM, in this case Volvo, would develop an adaptive speed limiter system (ASL) and collaborate with a Tier-1, in this case Autoliv, on developing a traffic sign recognition subsystem (TSR).

Their respective tools contain their own models, stored separately on each part's configuration management tool. The model exchange functionality is implemented by each tool vendor. There is a core model for the system and there is a tool to access it.

This means that the OEM (e.g. Volvo) has the core system model for ASL. It is exported and can be used by Tier-1 (e.g. Autoliv), but it is normally not changed. The Tier-1 can make some updates in the interface and export it back to the OEM, and this can be compared to the core model by the OEM.

The Tier-1 works with the model for the TSR subsystem.

The Tier-1 and the OEM may choose to export only parts of the model. This is fine as long as the requirements, interfaces and the common function are synchronized.

All partners: OEM (Volvo), Tier-1 (Autoliv), Engineering Bureau (Semcon) and Tier-2 (ArcCore, Systemite) can work in EATop with the full model or parts of it.

4.1 Requirements on Tools used for Exchanging Models

To address the challenges involved in exchanging models, such as the challenges are described in Section 3.3, the following will specify requirements on each software tool that is involved in the exchange.

Each exchange tool must be able to handle the following:

- Import of a new model
    - created in any tool that can produce EAST-ADL models
    - saved in .eaxml-format (according to the EAST-ADL XML schema eastadl_2-1-12.xsd [2])
- When importing a model that has been imported before, synchronization should occur. It shall be possible to synchronize using one of these methods:
    - Either each change can be monitored by the user and be individually accepted/rejected.
    - Alternatively all updates can be shown and the user can select to accept the total import or stop the import process.

Other requirements:

- UUIDS uniquely identify objects in the core tool. The UUIDs should not be changed during import and shall be stored together with the other data.
- At synchronization or round-trip an element may have been moved from a package to another or renamed. If it still has the same UUID, the tool into which the model is imported shall rename it or move it instead of re-creating it. This is in order to minimize the updating and avoid duplication (and to do the same thing that was intended at the update in the exported tool).
- An export from an imported eaxml file shall produce a semantically identical eaxml file
- It shall be possible to differentiate between the old model and the updated, but merging is optional
- An object that no longer is a part of the new (imported) model is not removed during synchronization. A warning is printed and it is up to the user to remove manually.
- Regarding parsing of .eaxml-files that do not fully comply with the corresponding XML-schema, the EAST-ADL schema eastadl_2-1-12.xsd [2] is used for validation of the .eaxml-file. If there is an error a warning is printed out, but the model is still imported (for all items that match).
- Where possible, links whose target element is missing should be preserved. On validation, there should be a warning, but the reference to the missing element should be preserved until the element appears in the model or the link is voluntarily deleted or edited.
- The version concept should be compatible with established software tools. In particular SystemWeaver, EATop and Enterprise Architect are considered to be established software tools.

Even with the requirements that are listed above, the model exchange will have some imitations:

- No support for variability
- No support for version information. However, Project Synligare presents a way to store version information as User Defined Attribute on each modelling element.
- Timing of Import/Synchronization is set manually. This means that the imported model may not match well if synchronization is done seldom.

- Automatic import is prohibited if two elements have the same package name or two elements have the same UUID.
- Missing elements cause there to be corresponding hanging links. In other words, there will be relationships between existing modelling elements and missing elements. Models that relate must all be imported otherwise the links will not be possible to resolve and an export function will miss them. One alternative is to add dummy elements to model in order to make it possible the make the link. Conceptually, such dummy elements could be replaced when a model is imported that contains them.

Anticipated issues with established software tools.

- In this work package, no concept has been developed for supporting error modelling in the tool Enterprise Architect.
- Handling of hanging links (links whose target element is missing) may be problematic to solve in Enterprise Architect.
- Enterprise Architect will not support reading or writing version information to User Defined Attributes.

4.2 Handling of UUID

The figure below shows a suggested approach.

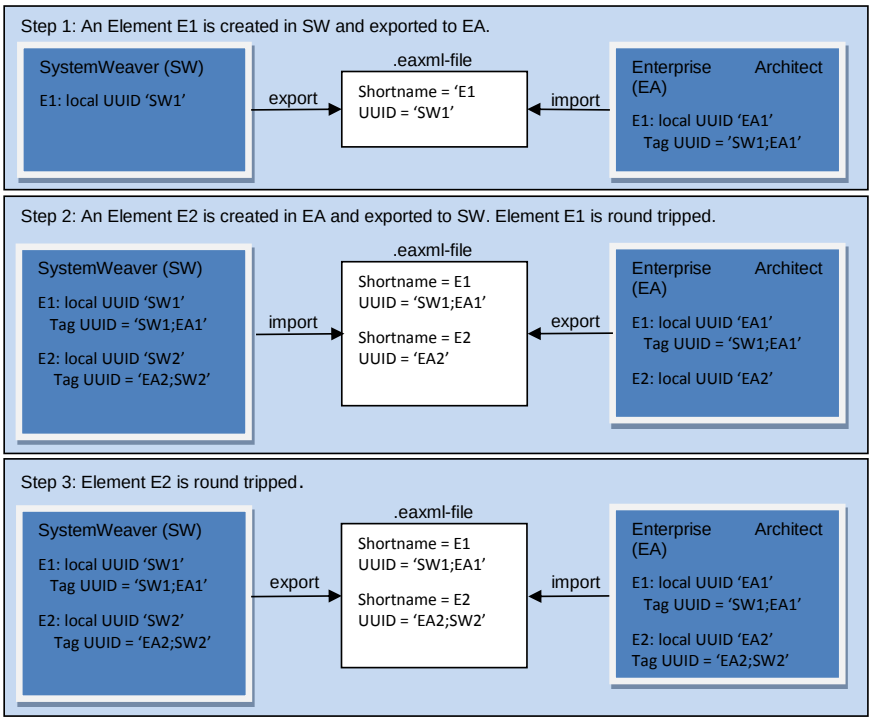

Figure 12 Illustration of shared model exchange problems at round-trip

The UUID syntax is as follows: UUID = 'UUID1;UUID2', meaning that both EA and SW adds the local UUID. This is needed to fulfil full roundtrip. The key idea is that the UUID is never updated by hand or changed by the tool by mistake.

A problem concerning handling of UUID in Project Synligare is that EATop does not use UUID. The need for using UUID is limited in EATop since EATop does not have a local model repository. It is

fully file-based. This lack of handling of UUID is not considered a problem to address within Project Synligare. Indeed, by not using UUID, it is ensured that UUID is never updated by hand or changed by mistake.

Example of UUID syntax:

- SW UUID = ’x1500000000187F7F’
- EA UUID = ’ {6CA88C2D-BCCE-4297-A845-1069983565D5}’
- Combined UUID= ’x1500000000187F7F;{6CA88C2D-BCCE-4297-A845-1069983565D5}’

4.3 Exchange between Models in EAST-ADL and Various Tools

It is considered typical that the OEM and Tier-1 have their own project areas and save the local models by using their local tool’s file format and their local version control systems.

It is assumed that exchange of models between OEM and Tier-1 is done via a common project area through export/import to/from eaxml files.

4.3.1 Example of Exchange of EAST-ADL Models Involving SystemWeaver

Consider a system model that is developed inside SystemWeaver. When the model is mature it is exported to eaxml. This model is put into the common project area.

When the Tier-1 has exported a sub-system model in eaxml format, it can be imported to SystemWeaver.

It is also possible to re-import parts of the system model (round-trip).

The workflow is shown in Figure 13.

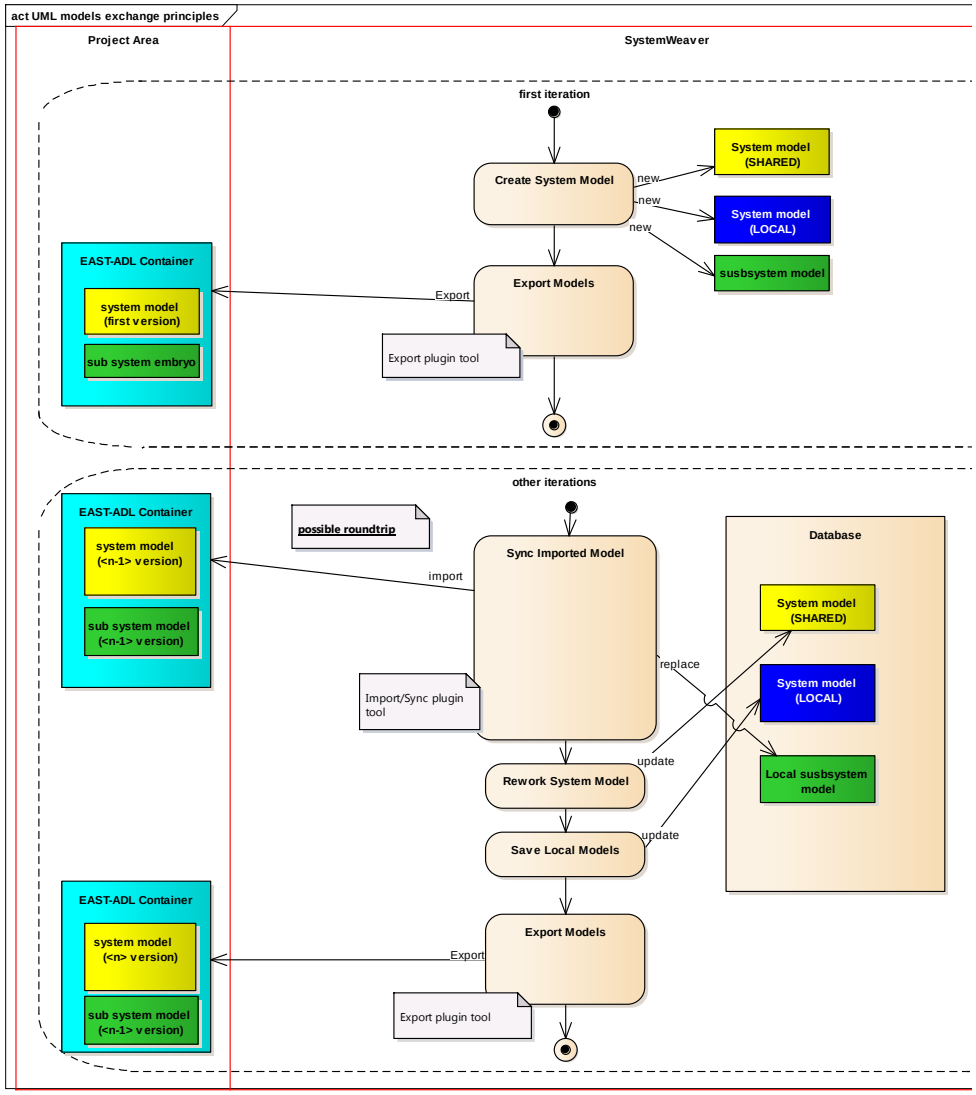

Figure 13 Model Import/Export/Sync/Round-trip from OEM perspective

4.3.2 Example of Exchange of EAST-ADL Models Involving Enterprise Architect

Consider a Tier-1 that has developed a sub-system. The Tier-1 can import a system model from eaxml format and add its sub-system to the system model.

The sub-system model is developed inside UML tool Enterprise Architect, adapted by Project Synligare to import and export EAST-ADL. UML has a lot in common with EAST-ADL. There are a couple of differences that are challenging:

- In EAST-ADL elements are referenced by its package path, but in UML elements are referenced by its UUID.
- In EAST-ADL unresolved relations can exist, but in UML unresolved relations cannot exist.

UML models can be split into configuration items based on packages that can be version controlled separately. Similarly, the EAST-ADL model can be split into several files.

The workflow is shown in Figure 14.

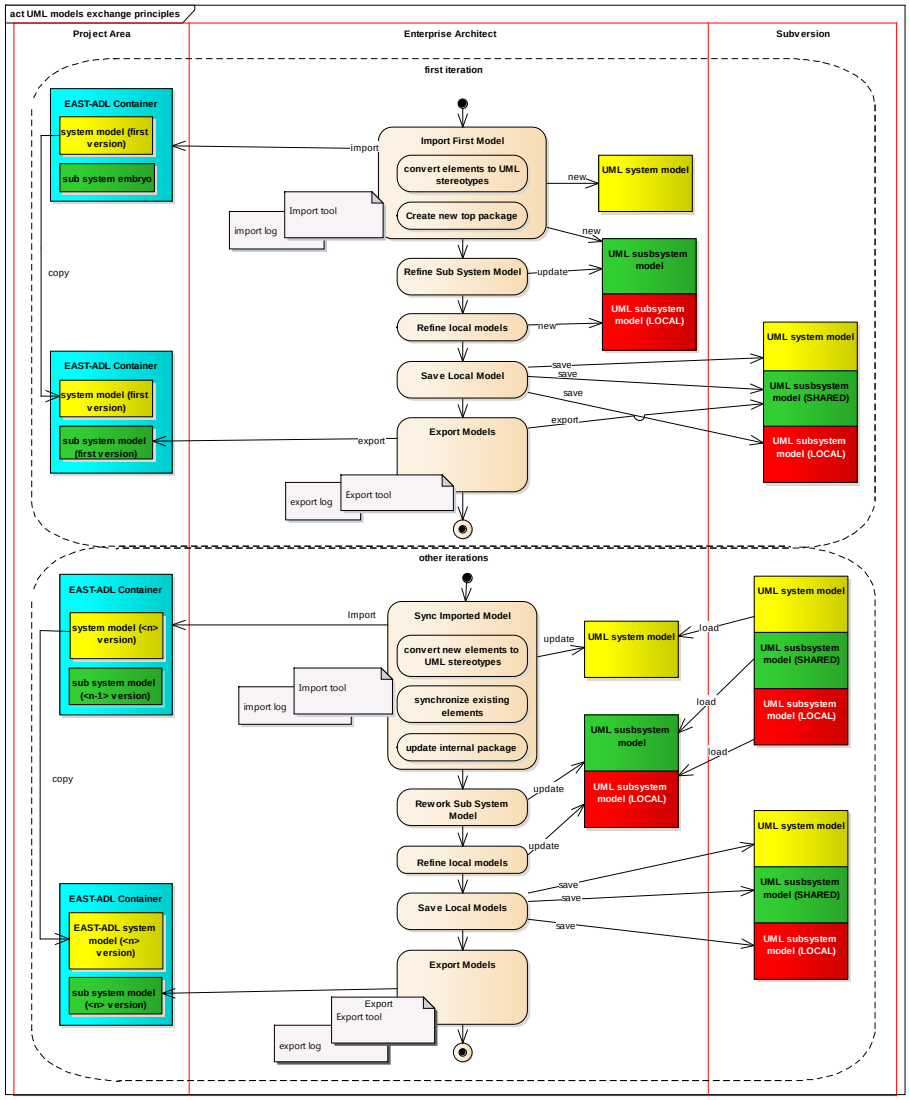

Figure 14 Model Import/Export/Sync/Round-trip from Tier-1 perspective

4.4 Requirements on Tools for Exchange of Diagrams

Diagram exchange between tools can be defined on different levels of interoperability:

- Export
- Export + Import
- Export + Import + Sync = Full Roundtrip

It is not a project need that all tools support all diagram exchange requirements. The important is that all functionality can be shown/tested with at least one tool. All tools support diagram export to sGraphML or yEd GraphML.

Basic demands for diagram Export:

- Each diagram element and the diagram connector have a unique object identifier. The object identifier can be UUID or package path which uniquely identify the diagram object.
- The diagram elements and the diagram connectors can be modified in the tool into which the diagram is imported. For example more details like colouring etc. can be added. This information is stored within the tool and may optionally be a part of the exchange. If elements and connectors are moved or resized, this shall be exported, as it is mandatory.
- If the diagrams are imported to a tool that also contains EAST-ADL models the UUID must point to an EAST-ADL element (for diagram elements) or EAST-ADL reference (for diagram connectors).

Each Diagram Exporting/Importing tool also must be able to handle the following:

- Import of a new diagram.
- Export a diagram
- The object identifier in the core tool shall be stored together with the other object data and must be possible to export when the diagrams are exported again.
- An export from an imported diagram file shall produce a semantically identical xml file. To make comparison of two diagram files easier, it is beneficial if the order of elements are preserved to the extent that it is possible.

Each Diagram Exporting/Importing tool that supports full roundtrip also must be able to handle the following:

- Previously imported diagrams shall be synchronized during import. It shall be possible to do this in one of these methods:
    - Either each change can be monitored by the user and be individually accepted/rejected
    - Alternatively all updates can be shown and the user can do the total import or stop it.
- It shall be possible to differentiate between the old diagrams and the updated but graphical merging is optional.
- A diagram object that no longer is a part of the new (imported) model is not removed during synchronization. A warning is printed and it is up to the user to remove manually.
- At synchronization or round-trip an element may have been moved from a package to another or renamed. If it still has the same UUID, the importing tool shall rename it or move it instead of re-creating it. This is in order to minimize the updating and avoid duplication (and to do the same thing that was intended at the update in the exporting tool).

Even with the requirements that are listed above, diagram exchange will have some limitations:

- No support for variability
- No support for handling versions
- The timing of Import/Synchronization is set manually. This means that the imported model may not match well if synchronization is done seldom.
- Automatic import is prohibited if two objects have the same UUID.

- It is up to the importer to check that the EAST-ADL models and diagrams are from the same baseline.
- It is up to the importer to first import EAST-ADL models before diagram import.

5 Manufacturer-Supplier Interaction

This chapter describes processes and process steps appropriate for the interaction between an OEM and a Tier-1. The scenario considered is that an OEM is developing a system and buys a key subsystem from a Tier-1. The OEM has a system model with an interface to the subsystem and a requirement specification for the subsystem. It may be that the Tier-1 is aiming to develop the subsystem as a new development project. An interesting alternative situation would be that the Tier-1 has a model of an existing generic subsystem, such that some development is necessary to modify the existing subsystem to meet the requirements of the OEM. The RFQ-process is conducted to agree on a collaborative development, with its associated cost and time-line. In this section, the following RFQ-related topics will be presented: A traditional RFQ process, a suggested RFQ-process and the exchanges of information that takes place, as well as the associated activities. The purpose is to describe activities that can be supported with viewpoints and metrics.

5.1 Traditional Interaction between OEM and Tier-1

Referring to a typical collaborative development project based on requirement specification as traditional, this section considers the interactions between OEM and Tier-1.

The traditional interactions between OEM and Tier-1 in a collaborative development project are shown in Figure 15. The OEM (marked as customer in the figure) and the Tier-1 (marked as supplier in the figure) play different roles. The OEM owns the system requirements and the system integration. The Tier-1 delivers SW, HW or both with increasing maturity as the development project progresses.

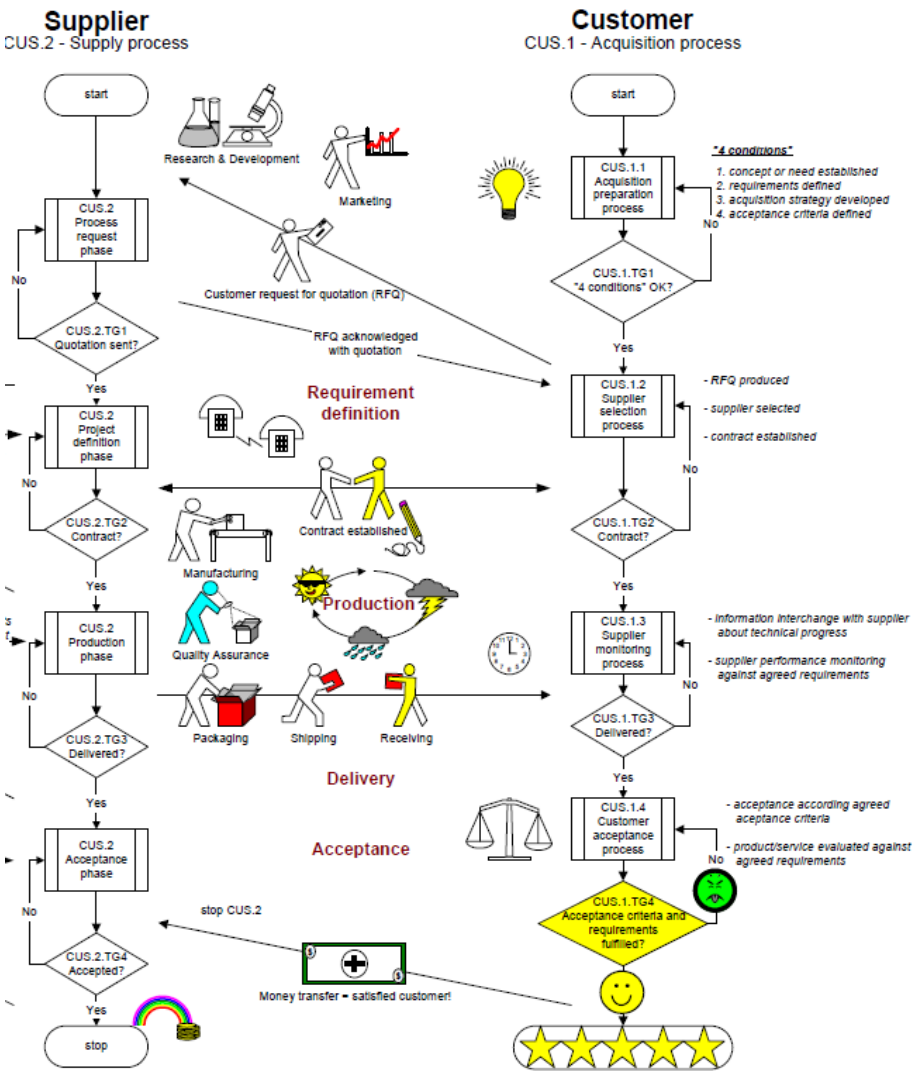

Figure 15 Customer / Supplier process

A first sub-process, called RFQ-process, starts from the first contact between the OEM and the Tier-1 and ends when the OEM acknowledges and accepts a quotation prepared by the Tier-1. In Figure 15, the RFQ-process corresponds to the steps between “start” (at both parties) and “supplier selection process” (for the customer) and “quotation sent” (for the supplier).

The interactions and activities between OEM and Tier-1 during the RFQ-process are listed in Table 3. For each interaction, activities at the OEM, activities at Tier-1 and identified viewpoints are listed in their respective columns.

Key to the RFQ process is the requirement analysis that is performed by the Tier-1. This analysis involves checking if the requirements can be fulfilled and negotiate with the OEM over requirements that cannot be fulfilled. If there is an existing subsystem that could potentially be adapted to the new requirements, the analysis is performed by attempting to map the requirements to the subsystem components. To correctly estimate cost of any changes that are required, the changes are categorized according to if they involve software, hardware or mechanical

development. Certain types of requirements are to be given extra consideration, since they have a more significant impact. Such types of requirements include functional safety requirements, legal requirements, requirements about diagnostic functions and requirements about performance and timing. If material or tooling needs to change, that is taken into account as a key cost factor. A part of the estimated cost is attributed to verification.

In this scenario, it is typical that the OEM has worked out the design of the subsystem to a certain level of abstraction. For example, the OEM might have described the subsystem's functions and have elicited all functional safety requirements. It is then up to the Tier-1 to implement the functions and meet the functional safety requirements.

The responsibility for each requirement should be determined, so that it is known which role in the project is to verify the fulfillment of each requirement, or if indeed the OEM will overtake the responsibility for verifying certain requirements.

Table 3

Interaction	OEM activity	Tier-1 activity	Viewpoint	Metric
Request for information	Ask for availability of a subsystem or capability to develop that subsystem, by communicating vehicle-level requirements and high-level characteristics	Answer by describing available subsystem or capability to develop the subsystem.	Safety Goals Requirements and Features	
Request for quotation	Ask for price and duration of development effort, by communicating specific requirements	Preliminary analysis of the requirements, possibly with regard to the existing subsystem, and planning of the development effort, to estimate cost and duration of development effort.	Requirements and Features Requirements and Functions Requirements and Components Combined viewpoint of requirements and allocation to various types of architectural components	Number of completed analyses / artefacts / reviews

The next sub-process is the requirement definition process. It continues until an agreement is found and a contract has been signed between the supplier and the customer, as is illustrated in Figure 15. During the requirement definition process the OEM and the Tier-1 can:

- Accept requirement
- Reject requirement
- Change requirement

After the contract agreement it is more complex to make changes in the agreed requirements. But in almost all projects this is necessary.

The requirement analysis in this requirement definition process is detailed in contrast to the analysis performed in the RFQ-process. Here the Tier-1 will explicitly map the requirements from the OEM to subsystem components. A plan is made for what changes are made to the generic subsystem. All changes as well as additional development on top of the generic subsystem are defined as a delta, and marked as such in the plan.

Each function is analyzed and an appropriate way to describe the outcome of the analysis is a block diagram with information flow, where events that trigger functional behavior are marked.

Further information to record in the function analysis include I/O mapping, signal specifications for interfaces and specification of any algorithms, as well as supporting functions, such as the operating system, the diagnostic functions and the I/O system.

The next step is component design analysis, the purpose of which is to get a detailed understanding of the delta development. In this analysis, HW is analyzed in terms of function, environmental conditions and mechanical aspects, as well as sensor (input) and actuator (output) devices. SW is analyzed in terms of function, timing and performance. The following aspects are specified: I/O interfaces, diagnostic functions, maintenance functions, operating system, communication protocols. The application software is further described, to take into account the changes between a generic algorithm and the new algorithm as required in the delta development. A plan is made for infrastructure that is required for the development work, such as simulation capability.

A test plan is created to ensure that important aspects of the changes are to be properly tested.

During this sub-process, the OEM as the customer go through a supplier selection process, as can be seen in Figure 15.

Table 4 lists activities at the OEM and at the Tier-1, with their respective views and metrics.

Table 4

Interaction	OEM activity	Tier-1 activity	Viewpoint	Metric
Accepting, rejecting and changing requirements	Answer the Tier-1's requests to reject or change requirements, by analyzing the impact of such changes	Analyze the requirements in detail and if necessary, reject or recommend a change of requirements.	Requirements and Features Requirements and Functions Requirements and Components Combined viewpoint of requirements and allocation to various types of architectural components Functional Safety Concept Technical Safety Concept	Number of completed analyses / artefacts / reviews
Supplier selection	Compare the quotations from different suppliers. If they have existing subsystems, they might differ in terms of performance and other quality attributes, as well as the cost and duration of development effort.			Number of completed analyses / artefacts / reviews

After the contract has been signed, the next sub-process is for development and production. It contains more than is illustrated in Figure 15. In this sub-process, there may be early integration and testing performed on early prototypes and for this purpose it is necessary to exchange detailed interface specifications, as well as progress reports to synchronize the time plans of both parts. As development progresses, reasons to adjust the requirements may be found. As design verification progresses, such status may be communicated. If the subsystem includes hardware, the quality of components may be reviewed. Further, functional safety-related artefacts may be communicated and design decisions may be discussed and agreed on. Table 5 lists activities at the OEM and at the Tier-1, with their respective views and metrics.

Table 5

Interaction	OEM activity	Tier-1 activity	Viewpoint	Metric
Exchanging interface specification	Integrating early prototypes	Selecting hardware components and verifying that the interface fulfills specifications	Requirements and Components	
Exchanging progress reports		Reporting the status of design, development and early verification	Illustrating metrics	Number of completed analyses / artefacts / reviews Verification progress
		Development and integration of subsystem components	Compare and Merge Support Software Architecture Requirements and Components	
Integrating and prototyping	Integrating prototype subsystem	Sharing integration information, release notes, verification reports, etc	Illustrating metrics Requirements and Components Requirements and Functions	Dependability metric Impact on production volute, income and cost metric Verification progress
		Functional safety analysis	Safety Analysis Tool Support	
Changing requirements		Discovering problems in meeting the requirements while selecting hardware components and while developing	Requirements and Features Requirements and Functions Requirements and Components Combined viewpoint of requirements and allocation to various types of architectural components	
Exchanging Functional Safety-related information	Requiring functional safety documentation for the purpose of in-vehicle trials of prototypes	Reporting functional safety documentation, as well as progress on implementation of diagnostic functions	Illustrating metrics Functional Safety Concept Technical Safety Concept	Number of completed analyses / artefacts / reviews

Next and final sub-process of interaction between OEM and Tier-1 is delivery and acceptance, as can be seen in Figure 15. At this point, the status of all artefacts should be complete and documentation should be mature.

Table 6

Interaction	OEM activity	Tier-1 activity	Viewpoint	Metric
Integration	Integrating the final version of the subsystem into the system	Supporting integration with relevant information	Illustrating metrics Requirements and Components Requirements and Functions	Dependability metric Impact on production volute, income and cost metric
Reporting final status		Delivering reports of completed artefacts	Illustrating metrics	Number of completed analyses / artefacts / reviews Verification progress Requirement validation
Delivering functional safety documentation		Delivering functional safety documentation	Illustrating metrics Safety Goals Functional Safety Concept Technical Safety Concept Safety Analysis Tool Support	Number of completed analyses / artefacts / reviews
Reporting documentation		Delivering various bits of documentation	Various views	Various metrics

5.2 Suggested Interaction between OEM and Tier-1

A key difference when comparing traditional interaction between OEM and Tier-1 with the interactions that are made possible by EAST-ADL is the model based design paradigm. EAST-ADL enables exchange of models. Instead of the traditional communication of textual requirements, it is plausible to envision exchange of normative and informative models complemented with textual requirements. For this purpose, it is not enough to communicate on a general modelling language. To make normative models effective in communicating clearly, a clear and industry-specific semantic is required. In this regard, EAST-ADL is more appropriate as information model than SysML and UML.

The introduction of a model based approach will impact this process in a number of ways, which both have pros and cons. The key benefit of introducing exchange of models in place of requirements is the transfer from a textual format to something more visual, which is expected to give overview. In the traditional exchange of requirements, the number of requirements could be excessive and this would make it hard to get an overview. In fact, it is likely that much of the information that was traditionally communicated as requirements is better represented in models. By giving more overview and reducing the number of requirements, it should be easier to communicate, analyze and review requirements.

When modelling is made on a deeper level than traditional, it is important to choose the right level of details. A lot of requirements could be moved to design, but they must still be implemented and verified.

A model-based development process that benefits from common information modelling is presented in Table 7. It should be noted that Table 7 is not as detailed as the tables in the previous section.

Table 7

OEM activity	Tier-1 activity	Viewpoint	Metric
Defines the intended functionality of the vehicle		Requirements and Features	
Defines requirements and use cases for the features		Requirements and Features	
Identifies parts of the functionality to buy from suppliers and marks them as normative based on a requirement specification process			
Defines system model and keeps parts to be bought as black boxes		Feature Tree Requirements and Functions Functional Safety Concept Technical Safety Concept Requirements and Components Software Architecture	Number of completed analyses / artefacts / reviews Dependability Impact of system model on production volume, income and cost
Sends system model to supplier as a request for information. Detail level corresponds to the OEM requirements.	Analyses system model	Requirements and Features Requirements and Functions Compare and Merge Support and	Number of completed analyses / artefacts / reviews
Analyses refined system model	Sends back refined system model, so far marked as informative. Detail level is restricted to what is available and what can be shared.	Requirements and Features Requirements and Functions Compare and Merge Support and	
Sends system model to supplier as a request for quotation	Analyses system model	Requirements and Features Requirements and Functions Requirements and Components Combined viewpoint of requirements and allocation to various types of architectural components Compare and Merge Support	Number of completed analyses / artefacts / reviews Requirement validation Dependability Impact of system model on production volume, income and cost
Analyses refined system model and quote. The OEM agrees on the quote. The OEM agrees on the normative elements.	Sends back refined system model and quote. Normative modelling elements are marked.	Requirements and Features Requirements and Functions Requirements and Components Combined viewpoint of requirements and allocation to various types of architectural components Compare and Merge Support	

The steps of sending system models and analyzing system models in Table 7 can be iterated until there is a consensus. The minimal content is a Feature Model defining the system content, but typically there will also be interface specifications to secure that the negotiated system can be integrated.

The same interactions as in the traditional interaction between OEM and Tier-1 can be expected, but based on models with complementary requirements. In the model based development context, we expect that it is possible to semi-automatically gather and export progress and product metrics, as well as some views. In other words, the manual effort of keeping up the communication between OEM and Tier-1 is significantly reduced.

6 Metrics

Various metrics on engineering documentation can be used to assess the product's properties or the development progress. In Project Synligare, we are investigating metrics that can assist the understanding of the system under development and the collaborative aspects of development, with particular focus on safety aspects. In general, all metrics that assist planning and follow-up engineering work are beneficial from a safety perspective, since faults caused by lack of resources and lack of time are impeded.

A metric is calculated on the basis of an entire system specification or parts thereof. Examples of delimitations include

- Architecture subset - The metric concerns a particular architecture or parts thereof
- Requirement subset - The metric concerns a particular requirement set
- Project - The metric concerns a particular project

A hierarchy of metrics based on the structural hierarchy of the system (system, subsystem, etc.) is conceivable. Depending on character, metrics can in a tool or in a report be visually represented as lists, flags and colors in addition to plain numbers.

Two main categories of metrics will be discussed below, namely progress and product metrics. Development progress can be assessed based on the status of the model or based on assessments of actual status that is reflected in the model.

6.1 Requirement Validation Progress

To use requirements to communicate product prerequisites and specifications, and to have an appropriate input for detailing the product, they have to be validated. As this is also a measure of progress, a Requirement Validation progress metric can be defined.

A requirement is considered valid when its content has been assessed. This may be marked using an attribute, or the requirement can be moved from a preliminary to its final package after validation. For the purpose of this metric, we will use the latter method: Initial requirements are placed in a preliminary package RequirementModel, and as they are validated they are moved to the correct package or RequirementModel. For the metric to be calculated, the package for preliminary requirements is marked with a "Preliminary" attribute using user defined attributes. An enumeration datatype with one value "Preliminary" is needed for this purpose. Figure 16 shows the relevant meta-model elements for attributing the preliminary package or requirement model (top right) and the requirement, package and requirement model for organizing the requirements (bottom).

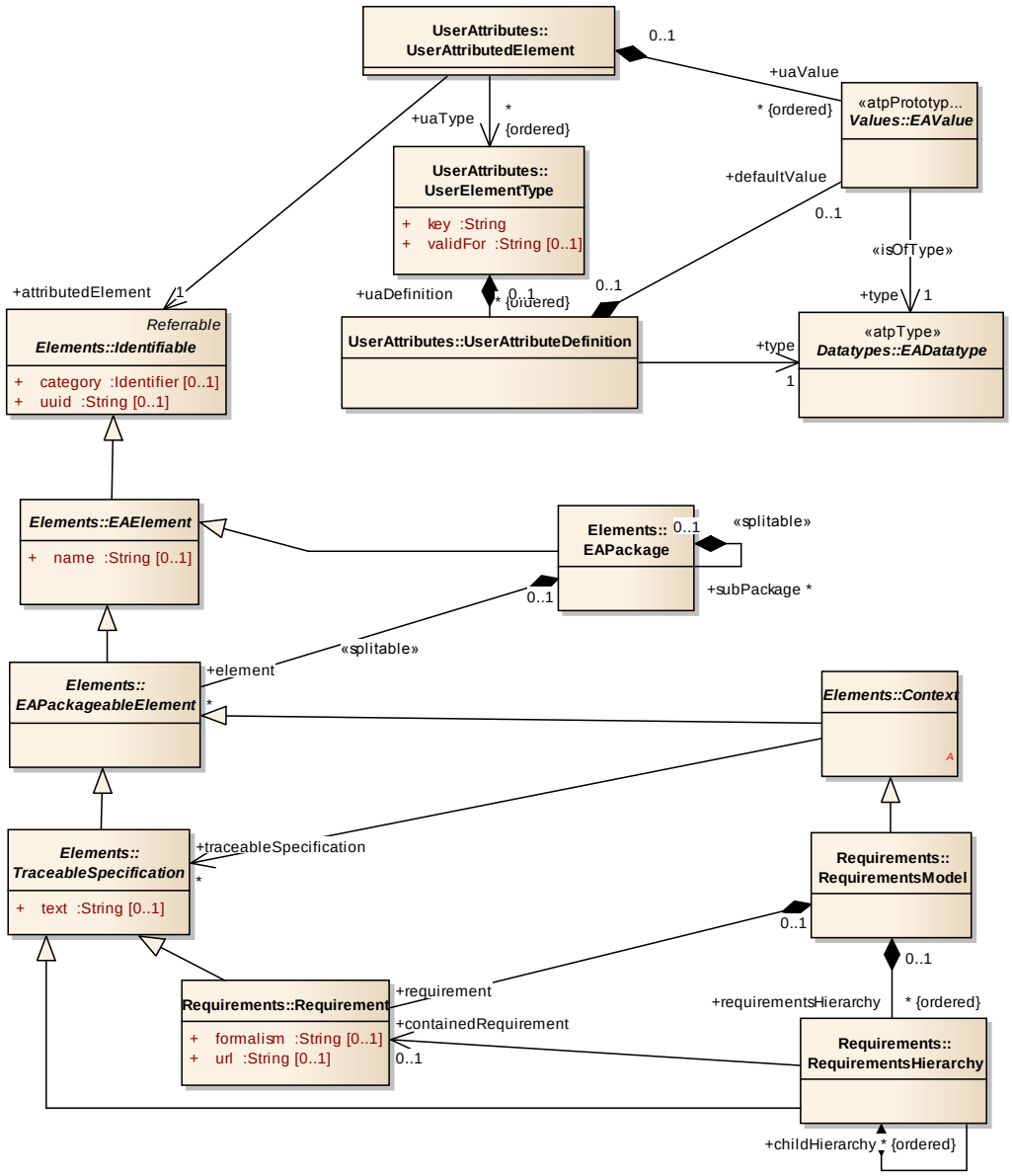

Figure 16 Non-validated requirements can be located in a package or requirements model marked with a user defined attribute with value "Preliminary"

6.2 Requirement Allocation Progress

Assuming that requirements are used to specify what the product shall do, they can also be the basis for assessing progress. Progress of requirement allocation is measured as the fraction of requirements allocated to architectural elements.

The metric for progress of requirement allocation provides an example of a metric that counts the number of completed tasks (see the scope defined in Section 2.2) with respect to covering the whole system. Typically, this type of metrics is used to follow the progress of a development project.

Figure 17 shows modelling elements for the satisfy relationship to illustrate that the requirement allocation progress metric is such that a requirement can be considered allocated when a) there is a satisfy link or b) when its derived requirements have satisfy links.

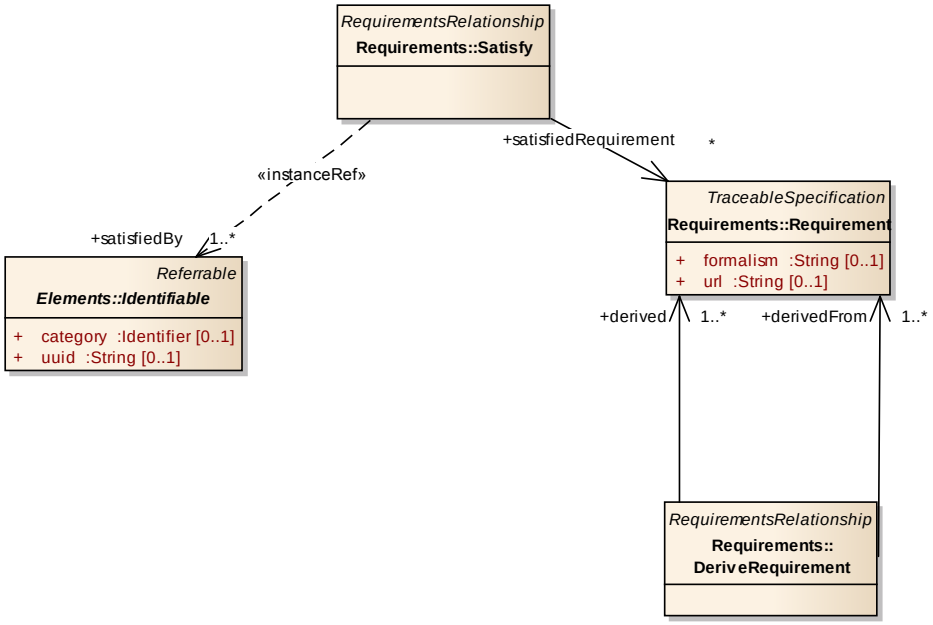

Figure 17 Requirements can be considered allocated when they have Satisfy links, or their derived requirements have Satisfy links.

Because requirements may be derived to more specific requirements, a requirement with one or more derived requirements is also considered as indirectly allocated when the derived requirements are allocated. Such “base” requirements are thus not counted among the allocateable requirements. The exception is when derived requirements are located in another package or requirements model. In this case, the requirements are considered to be at another abstraction level and therefore they do not qualify the “base” requirement as allocated.

Progress of requirement implementation is measured as the fraction of requirements that are implemented in the product. This criterion is not well-defined, as it may concern requirements with a final design, final software specification, final code, compiled code, etc. What we will assume is that a requirement that has derived requirements that are allocated to Implementation level elements is implemented.

On computing this metric, the model can be inspected for one of the criteria mentioned, or a specific flag that is manually set by the engineer who is performing the task of allocating requirements is used as a basis.

6.3 Realization Progress

The EAST-ADL model [3] provides means to relate model elements to their abstract counterparts. For example, a function or ECU is linked to the Vehicle Feature it is realizing. In the end all Vehicle Features should be realized down to the Software Component level. On computing realization

progress, one must decide which criterion to use, for example realization in software, function or hardware architecture. One must also be aware that it is not possible to know if the traceability through realize links is complete, i.e. how many concrete elements are required for realizing an abstract element. For this reason, an alternative approach is to use a specific flag which is manually set when a feature is deemed to be realized.

This metric of realization progress in an example of a metric used to count the number of completed tasks (see the scope defined in Section 2.2), in this case the task of modelling on a lower abstraction level and adding realization links between the elements of the more abstract model and the corresponding elements of the more specific model.

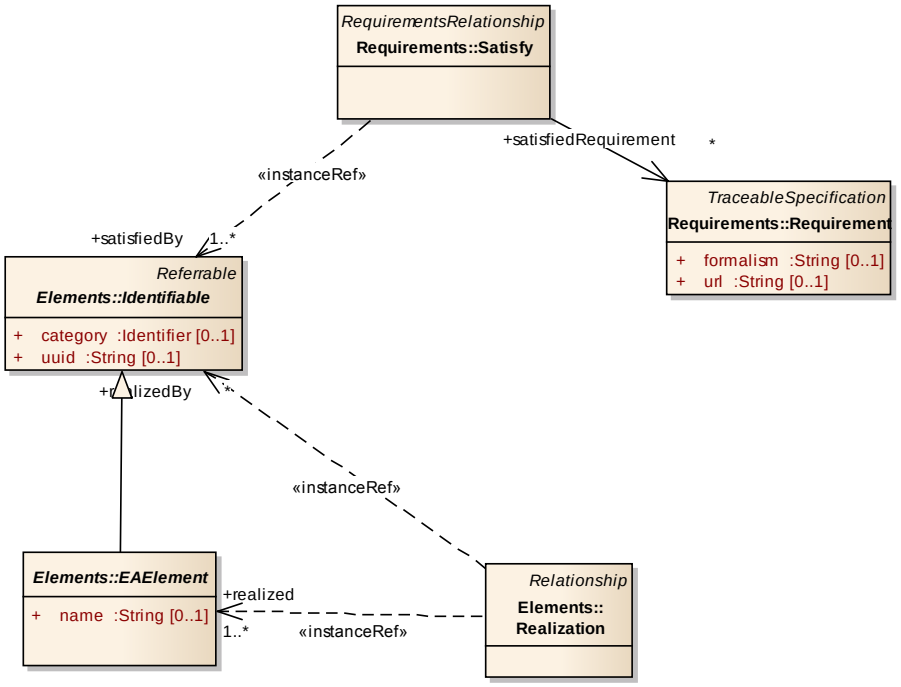

Figure 18 Requirements can be considered done when they are implemented, i.e. their target architectural element is realized.

Figure 18 above shows the realization relation between artifacts, and Figure 19 shows how to attach flags using the user attribute mechanism.

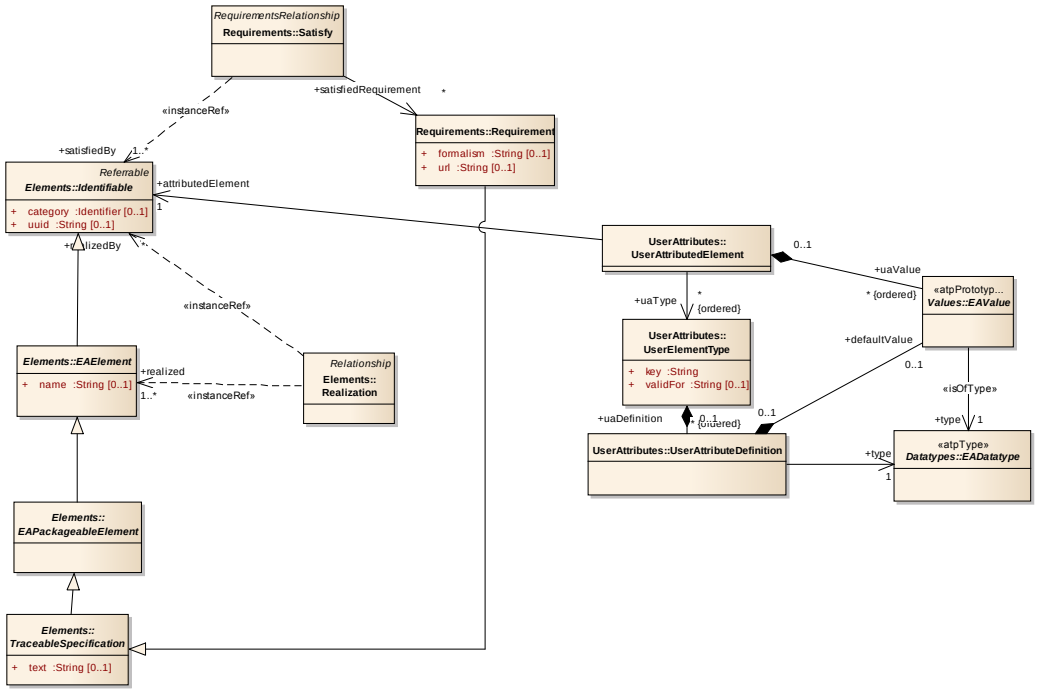

Figure 19 User Attributes can be used to put flags on model elements

6.4 Verification Progress

In EAST-ADL, the verification specification VVCase is linked to requirements. For each requirement one or several verification cases can be defined, each of which may pass or fail. The verification progress metric counts the number of performed verification cases with respect to the set of verification cases that are defined in a verification plan.

This metric for verification progress contributes to the scope for this report as defined in Section 2.2.

Figure 20 shows how requirements and verification are related. Note that there is an abstract VVCase defining the test or analysis steps required, and an abstract VVCase that holds one or several verification results.

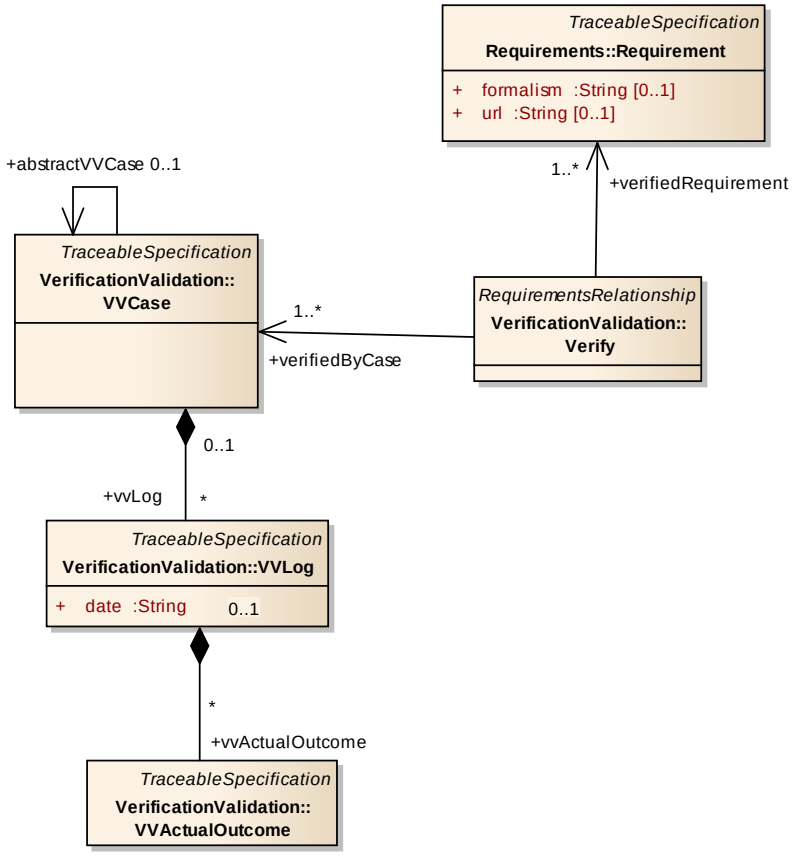

Figure 20 Verification specification for Requirements

Verification progress may thus concern the fraction of requirements that are verified or where a verification method is defined. If multiple verification cases are defined per requirement, the fraction may be based on verification cases instead. In case verification is done, a useful metric is the fraction of passed vs. failed requirements.

6.5 Safety-Related Progress Metrics

The ISO26262 functional safety standard [4] defines a number of engineering artifacts that shall be defined in the context of its Item. One of these artifacts is the safety goal, which is the starting point for organizing safety requirements. Examples of metrics include the fraction of Items with safety goals defined or the fraction of safety goals with functional safety requirements. Because it is system specific, it requires analysis to identify the safety goals or safety requirements that are needed. It is still a useful metric, since the amount must be non-zero.

These safety-relevant progress metrics are examples of metrics that count the number of completed tasks, as mentioned in the definition of this report's scope in Section 2.2.

Another safety-related metric is the fraction of functional safety concept components that are matched by technical safety concept components. This can be computed based on derivation relations between functional and technical safety requirements combined with realization relations between analysis functions and design functions making up the architectural aspects of the functional and technical safety concepts, respectively.

Figure 21 shows how the FunctionalSafetyConcept element identifies a set of Requirements. These requirements together with the elements to which they are related with a satisfy relation makes up the FunctionalSafetyConcept. The corresponding SafetyGoal is found through the Derive relation to the requirement identified as a safetyGoal.

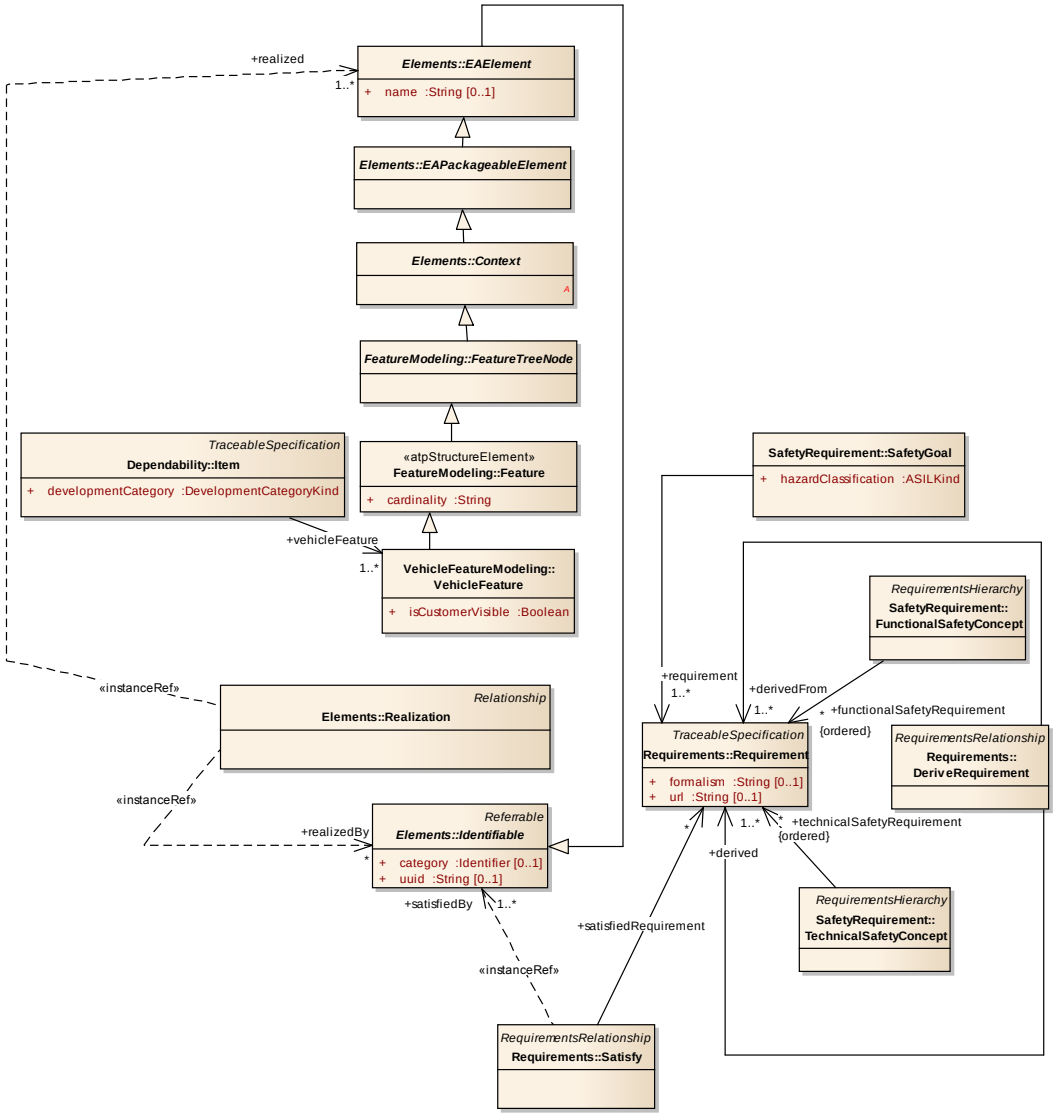

Figure 21 Elements and relations related to Functional and Technical Safety Concept

6.6 Product Metrics

As opposed to progress metrics, product related metrics are typically absolute. One metric that is relevant for strategic decisions regarding development rigor or priority, is the number of manufactured vehicles that are impacted by a certain engineering artifact. This can be computed

based on the take rate annotations of vehicle features realized by the engineering artifact considered. Similarly, total revenue or cost can be computed by combining take rate information with financial annotations.

When applied to estimate production volume and cost factors, the product metrics contribute to the scope defined for this report in Section 2.2.

With product metrics, it is possible to track dependability expressed in the expected lifetime of hardware components with respect to the expected stress that they will be exposed to. This is typically calculated in a structured way. It can be seen that such a metric contributes to the scope for this report as defined in Section 2.2.

Figure 22 shows how GenericConstraints of kind pieceCost can be used to define cost. The TakeRateConstraint defines with an absolute number how many elements are produced of the target, if the source is empty. If a source is also identified, the take rate is a fraction.

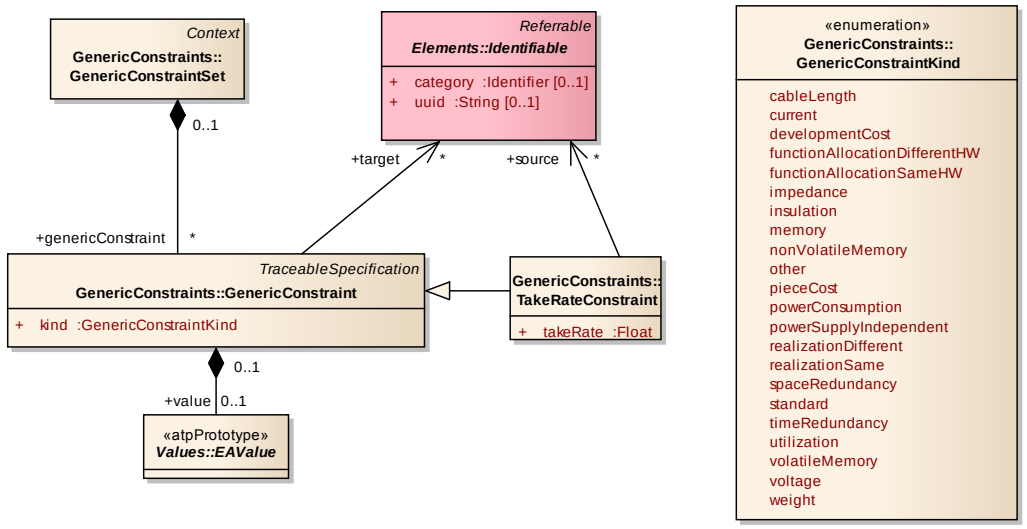

Figure 22 GenericConstraint can be used for annotating elements with a cost and take rate.

Various dependability metrics are relevant for developing safety-critical systems. These include failure rate and other hardware architecture metrics. Figure 23 shows elements relevant for failure rate annotation, based on which it is possible to calculate a hardware dependability metric.

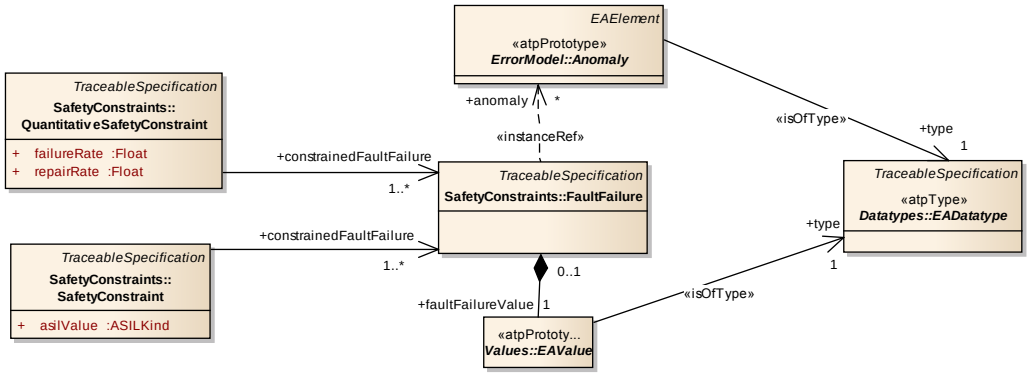

Figure 23 Failure rate constraint as example of product metric.

A metric for quantifying the complexity of a (distributed) software system should be considered. From literature, there is the Software Structure Metrics Based on Information Flow [5]. Simplified, it is about counting the number of information input flows to a software component and counting the number of information output flows from a software component. The information flows are categorized as local if they follow the control flow of the software system. Based on this information and the number of lines of code for a software component, the software component is assigned a complexity value defined by a formula. This metric can be extended to the whole software system.

To summarize, product metrics are handled according to their formulas, often depending on the structure of the components. Here are relevant examples of product metrics:

- Component Complexity
- Component Cost
- Expected Revenue
- Expected Lifetime
- Expected Error Rate
- Energy Consumption

6.7 Identifying Context

In general, it is necessary to be aware of context when computing metrics based on fractions. The AUTOSAR/EAST-ADL notion of package is one possible context. Another is to consider a certain feature, function, hardware component and compute a fraction among the related elements.

7 Viewpoints

A view is a concept where a subset of the available system information is collected. The information that is modelled in a view is dependent on the needs of the parties involved in development. A viewpoint is a specification of the conventions for constructing a view. I.e. the viewpoint has the data of a view as input and the user interface representation of the data as output. This chapter describes on a conceptual level viewpoints that are considered to satisfy the identified needs. The scope of viewpoints considered in this report is given in Section 2.2.1.

In this chapter, viewpoints that have been processed in WP2 are described and illustrated. View examples are marked , while metamodel diagrams showing the related modelling concepts are marked .

This chapter describes first generic viewpoints that can be applied to any part of a system model and without regard to role or task, namely:

Generic Tree Viewpoint

Generic Properties Viewpoint

Generic Version Information Viewpoint

Generic Table Viewpoint

Generic Diagram Viewpoint

Generic Chart Viewpoint

Subsequently the chapter describe viewpoints that are specific to a subset of the system model or related to some role or task.

7.1 Generic Tree Viewpoint

Elements in a model may be organized in a containment hierarchy based on packages. Typically, it is necessary to browse such model according to the containment hierarchy, but also along various relations. In a generic tree view, it is typically relations of the type parent-child that are shown, since this type of relations lend themselves well for a tree view. Other types of relations are not shown in the hierarchical tree view.

The preferred concrete view is categorized as a tree view with a set of features to:

- visualize the model
- navigate the model
- visualize and categorize model object types
- create model objects

7.1.1 Example View

EATOP holds an implementation of a tree view, see Figure 24. In Project Synligare, it is augmented according to navigation needs described above.

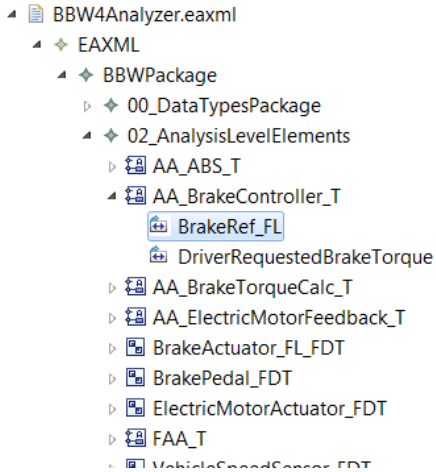

Figure 24 Example of a tree view

7.1.2 Related Modelling Concepts

Figure 25 shows relations that are relevant for navigation between elements in a tree view: Navigating the specializations of Relationship, such as Realization, Satisfy or Verify are important to get a quick overview of a model. The is-of-type relation is fundamental to access the specification of an element. Many relations are of type instance-ref, i.e. they refer to an element in a deep prototype hierarchy (an instance hierarchy of reused function, hardware or software types). Such navigation relies on a combination of the is-of-type and instance-ref relations.

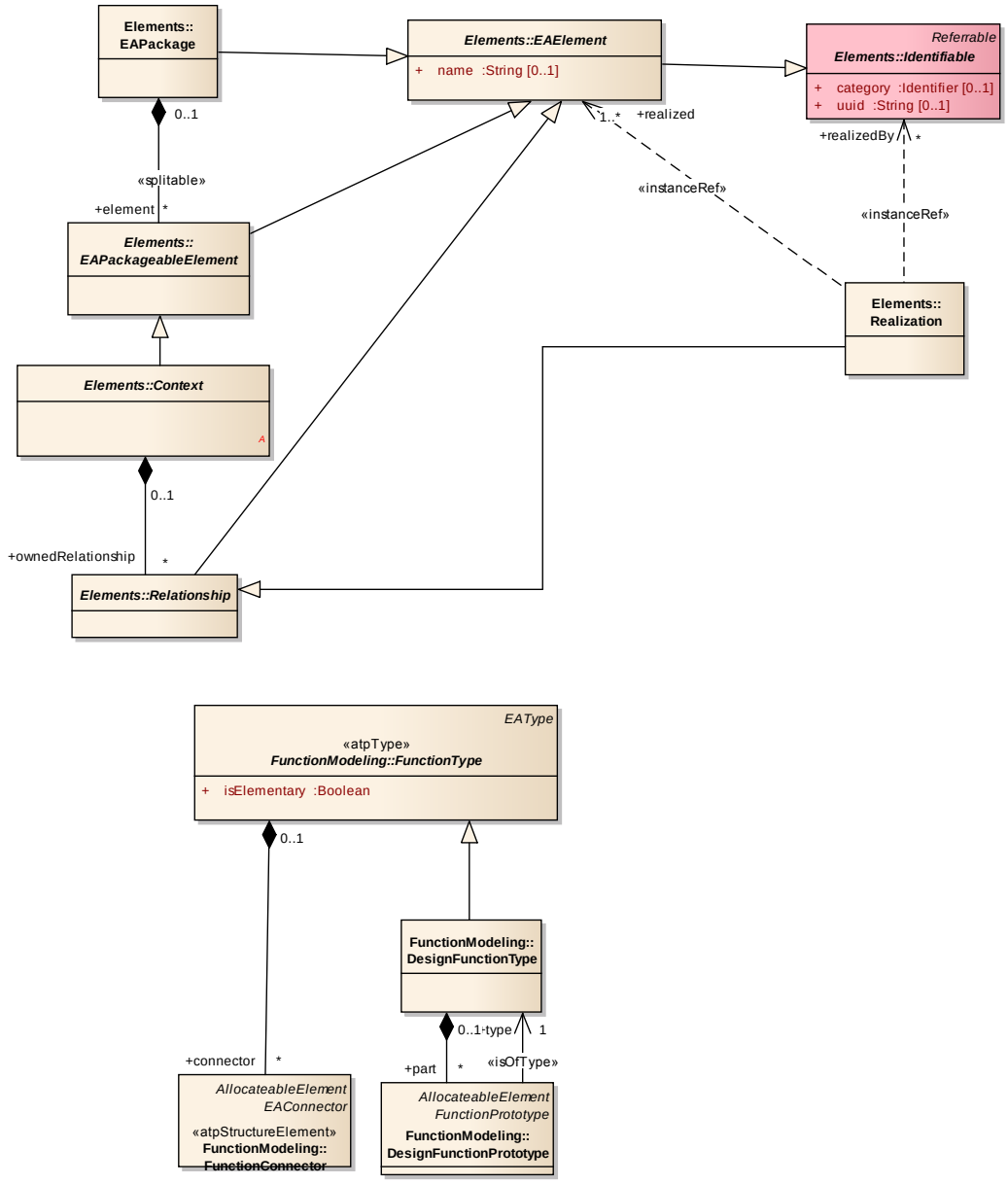

Figure 25 Examples of relations with relevance for visualization and navigation

7.2 Generic Properties Viewpoint

Elements have properties defined according to their types in the modelling language. The properties contain data about the element and references from the element. Viewing such data can be helpful whilst either trying to understand the model or looking for problems and errors. Conceptually, this is done by viewing one object at a time and the data is then presented in a table or list. Figure 26 depicts the properties of an AnalysisFunctionPrototype in EAST-ADL with example values. It should be noted that this view only shows the properties that are specific for the

given modelling element. The element also inherits data from its relations and these data are called attributes.

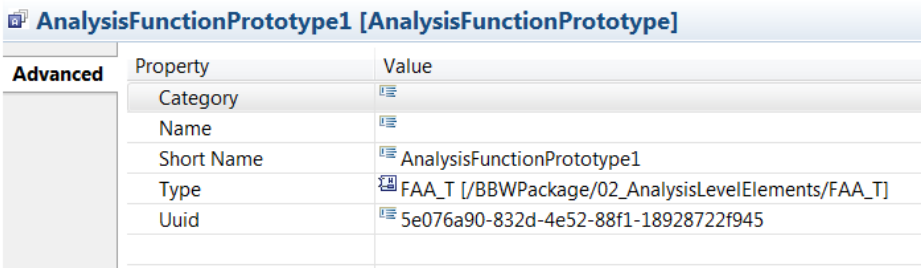

Figure 26 Example of a properties view from EATOP's EAST-ADL explorer

7.3 Generic Version Information Viewpoint

In Project Synligare, a concept for keeping a version number for each modelling element is introduced. The concept employs the EAST-ADL User Defined Attribute that can be added to modelling elements of all types. Tool support for this does not require much in the way of a visualization, but rather functionality to read and write the version information. It should be noted that this makes version information an attribute, not represented as a data owned by the element but rather data related to the element. Since User Defined Attribute are not standardized in EAST-ADL it cannot be expected to be used in the same way in all system models. Unfortunately, this means that version information is not shown in the Generic Property Viewpoint. This is why a Generic Version Information Viewpoint is needed.

7.3.1 Example View

Figure 27 shows how version information can be accessed in a tool through a context menu. Figure 28 shows the pop-up information box that is displayed if the menu option Version Property is selected in the context menu of Figure 27.

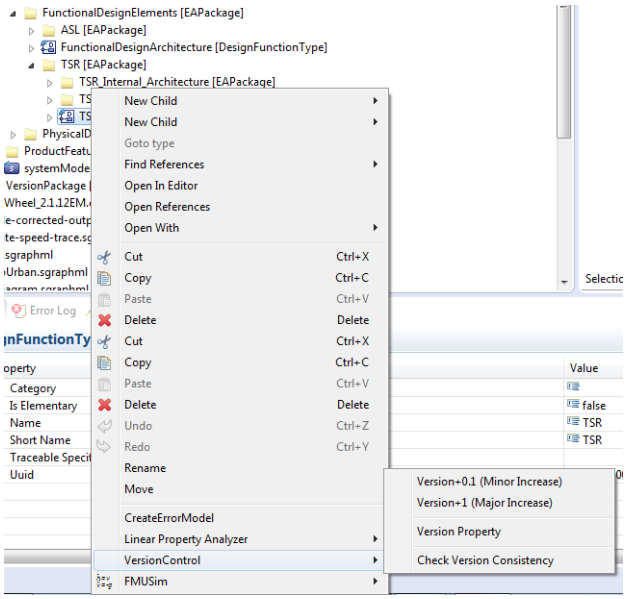

Figure 27 Example for Version Information Viewpoint in the form of a context menu

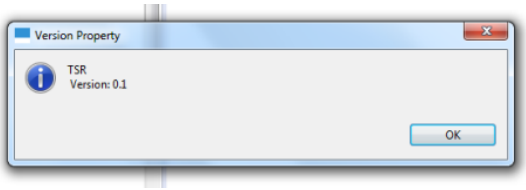

Figure 28 Example for Version Information Viewpoint in a pop-up information box

7.3.2 Related Modeling Concepts

Figure 29 shows the part of the EAST-ADL language that Project Synligare is employing to represent version information, namely User Defined Attribute. It should be noted that EAST-ADL does not have any explicit part of the language for representing version information.

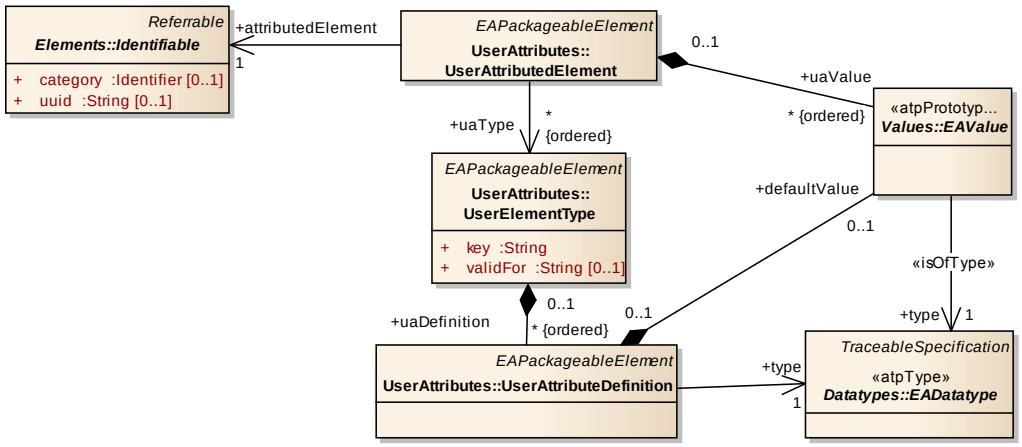

Figure 29 User Defined Attribute in EAST-ADL

7.4 Generic Table Viewpoint

It is often helpful to compare elements as well as the values of their properties and attributes. The Generic Properties Viewpoint listed only a given element's properties, and the Generic Version Information Viewpoint only handled a single attribute of a given element. Here we describe a Generic Table Viewpoint that can show elements on rows and values of properties and attributes in the columns. It should be noted that elements of different types have different properties in EAST-ADL, so the Generic Table Viewpoint is best suited when comparing elements of the same type. For example, the viewpoint can be employed to show which functions are allocated to what component of the hardware architecture. The viewpoint also provides editing capability, so that the values of properties can be edited.

There is an example in Figure 30. Further support for comparing parts of models are considered in Section 7.8.

Table View

	value	mode	kind	target	Satisfied Requirements	Incoming references
cableLengthUnit	1		CABLELENGTH	BrakeActuator_AT CPUTempSensor_T FlexRayWire1		
cableLengthUnit2	2		CABLELENGTH	Wheel_FlexECU_T WheelSpeedSensor_AT		
cableLenOverall	100		CABLELENGTH	HDA_T		cableLenRef
PowerConMax	100		POWERCONSUMPTION	HDA_T		powerConRef
PowerConUnit2	2	Mode2	POWERCONSUMPTION	Wheel_FlexECU_T WheelSpeedSensor_AT		
PowerConUnit3	10		POWERCONSUMPTION	CPUTempSensor_T		
PowerConUnitA	3	ModeA	POWERCONSUMPTION	HC1		
PowerConUnitB	7	ModeB	POWERCONSUMPTION	HC2		

Figure 30 Example of a table view in EATOP with GenericConstraints in rows and properties and associated elements in columns

This viewpoint contributes to the scope of the report (see Section 2.2) as a way to illustrate properties of modelling elements and to show how different elements are related.

7.5 Generic Diagram Viewpoint

There is a need to display the relations that are not visible in the hierarchical tree view of the Generic Tree Viewpoint. Relevant examples of such relations are:

- The of-type relation. A modelling element is of a type defined in another part of the model.
- Information flow from function to function.
- How connectors attach to hardware components.
- How requirements are satisfied by system model elements.
- How abstract models are realized by more specific models.

The typical way to show information such as these relationships is to have a diagram of the elements. The diagram typically takes the shape of a graph, with modelling elements as nodes and relationships as edges. Depending on the relationship, the edges may be directed or undirected. For example, a function diagram is typically arranged with inputs to the left and outputs to the right and information flow from the left to the right. To compare, it can be considered that the satisfy relationship between a requirement and a modelling element that satisfies it is not directed and it does not matter if the requirement is placed above, below, left of or right of the element.

Here are some requirements on a Generic Diagram Viewpoint:

- Elements of the same type should have a consistent visual appearance in terms of shape, color and other decoration.

- The elements should be placed appropriately in relation to each other so that elements that are related are close to each other, while unrelated elements can be placed far apart, as a way to accentuate the structure of the information.
- Elements should not overlap.
- Edges should if possible not overlap.
- The visual appearance of elements may be decorated with diagrams showing their internal structure. In other words, the diagram may be a hierarchical graph where the nodes may contain graphs.
- The diagram should show relations between elements that are contained in disjunct subgraphs (content of different elements) even if the connected elements are on different levels of hierarchy.

Diagrams can be classified according to if they are directed, undirected or showing a tree structure, or a mix of these. Often the most useful diagrams involve a few different types of elements and relationships, so that would be a mix of directed and undirected. It should be noted that it is common that directed diagrams, such as those containing functions with input and outputs, typically have relations routed horizontally and vertically to not overlap with the nodes that represent the elements.

Examples of use of Generic Diagram Viewpoints are available in several figures in the following, including Figure 34, Figure 35, Figure 36, Figure 37, Figure 38, Figure 44, Figure 45, Figure 46, Figure 47, Figure 48, and Figure 49.

7.6 Generic Chart Viewpoint

To show relative and absolute values of properties and calculated metrics, also over time, there should be a viewpoint for pie charts, plots and other ways to display values rather than elements and relationships. This viewpoint would typically be used in conjunction with a metric that supplies the values to be plotted or otherwise shown. This viewpoint requires the modelling elements and relationships that concern the metric, but in itself it does not need any additional modelling elements or relationships. Examples are seen in Section 7.21.

7.7 Create Connector Support

Using a tree view involves managing the model and creating additions to the mode, such as creating connectors and ports. Depending on the element type the process of creating a new element may have varying degrees of complexity. Creating a connector can be automated. This task involves the creation of the connector itself, creating the port reference elements and finally linking these to actual ports. Visualizing and automating part of this process simplifies and speeds up the process of working with models. A suggested concept is to create a tool which can parse the model and present options for the user of relevant ports, bringing down the process of creating connectors to a couple of steps.

7.7.1 Example View

Figure 31 shows an example view with ports from the local context that are accessible for selection and automatic connection.

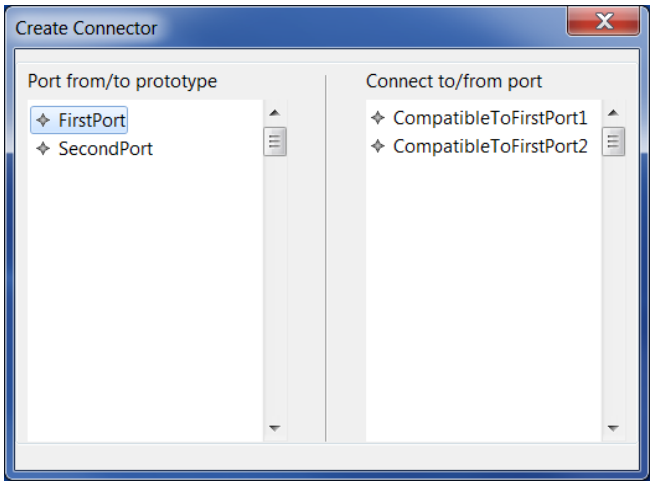

Figure 31 Concept of creating a connector

7.7.2 Related Modeling Concepts

Figure 32 shows the port and connector elements, which are the basis for a create connector support viewpoint.

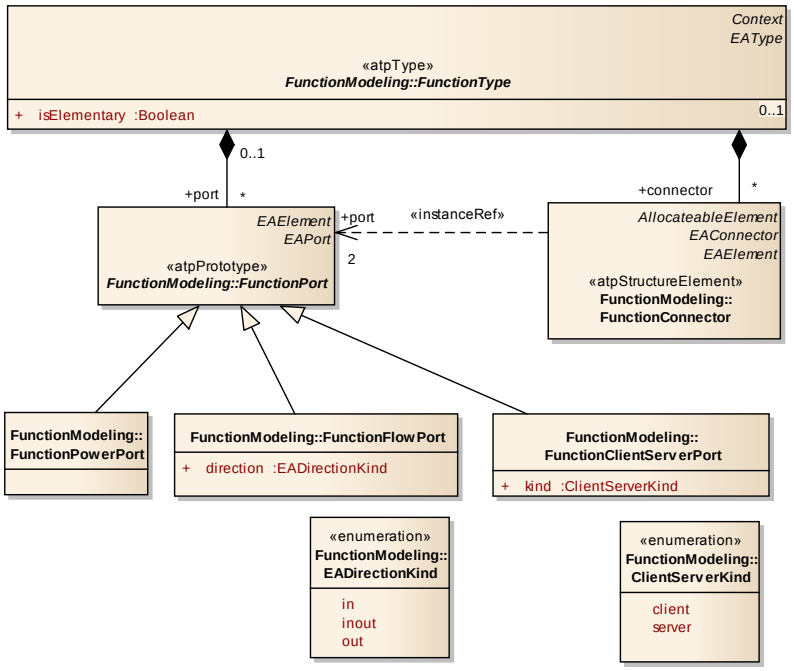

Figure 32 Ports and Connectors

7.8 Compare and Merge Support and Viewpoint

An important task during system development, in particular in a collaborative setting, is the merging of models of different origins. Models may represent a new version of something that

already existed, in which case it is a difficult task do identify changes and decide whether to accept or reject the change. The assessment is easier if version information is included, as the default choice would then be the most recent element.

Compare and merge is a view and editing support that assists the engineer during comparison of versioned or non-versioned changes to models and during updating the base model based on the comparison results.

7.8.1 Example View

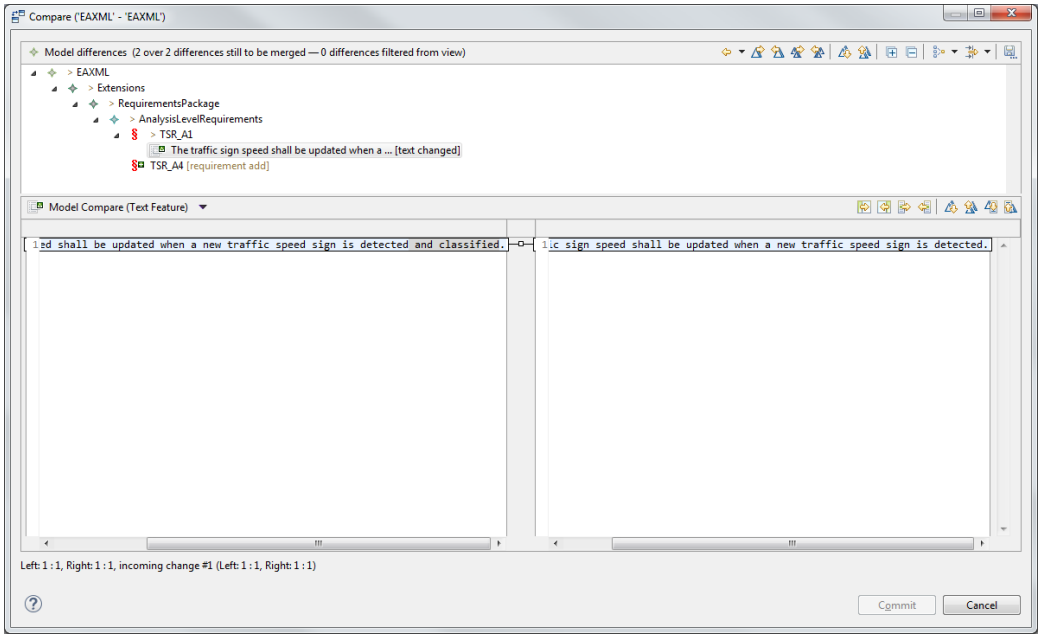

Figure 33 Screenshot of Compare and Merge.

7.9 Features Tree Viewpoint

On a feature-level modelling of a system, the main variability is handled. A typical way of illustrating a feature model is the feature tree. It has a semantic of mandatory features and optional features, where the optional features are included in the system on an exclusive-or basis.

The viewpoint that shows a feature tree contributes to the scope for the report as defined in Section 2.2.

7.9.1 Example View

Figure 34 illustrates a feature tree decorated with steps of incremental development in the form of numbers in the top left corner of each feature symbol and functional safety information in the form of ASIL. Further, there is the possibility to add comments. The comments here are written to explain the feature tree figure.

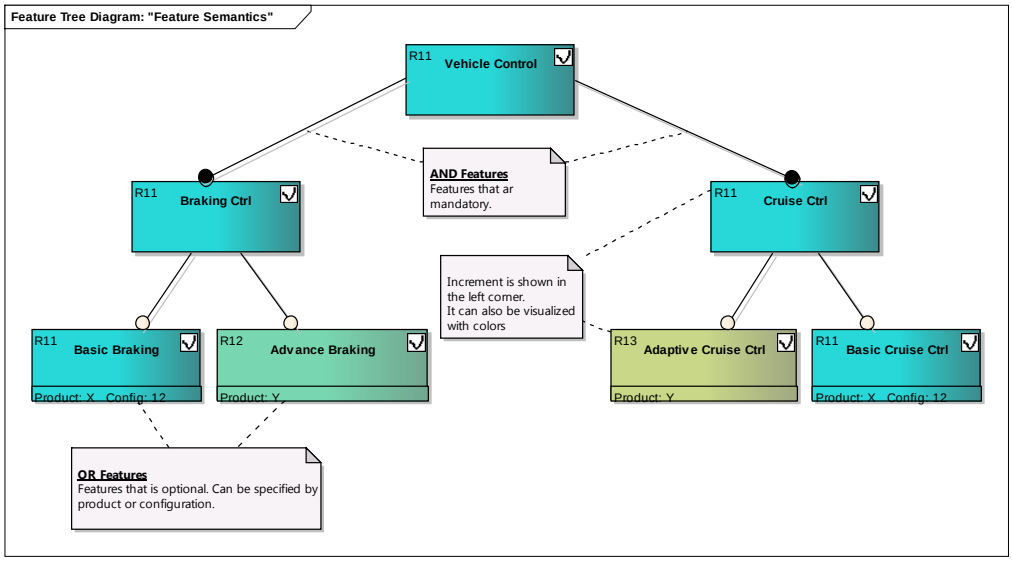

Figure 34 Feature tree decorated with steps of incremental development

The feature tree may in complex systems be rather large. Application of a filter to highlight relevant information and gray out other information may make the feature tree well understandable also in the case of a complex system. An illustration of applying a filter is given in Figure 35.

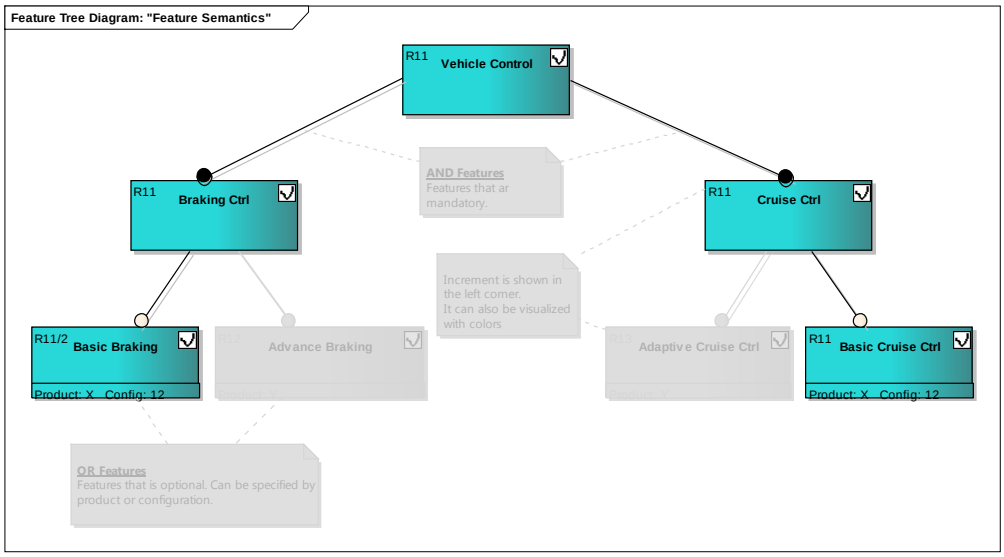

Figure 35 Feature tree with a filter to highlight desired information and gray out other

7.10 Requirements and Features Viewpoint

To understand a feature model for a system, the feature tree may be used in conjunction with requirements, grouped based on features, as illustrated in Figure 36. This would help to understand the feature model through the explanation of the requirements, and it would help to understand the requirements by seeing for which features they are relevant.

The viewpoint of a feature tree with requirements grouped on relevant features contributes to the scope as defined for the report in Section 2.2.

7.10.1 Example View

Figure 36 below shows features and their requirements according to Satisfy relationships. It is a viewpoint that does not scale to realistic sets of requirements, but filtering (showing only a subset of requirements) is a way to keep the number of requirements in the view manageable. Such views are also good entry points into a model: Once the overall picture is communicated and understood through a graphical view, a tree view or list view can be used for the full scale work.

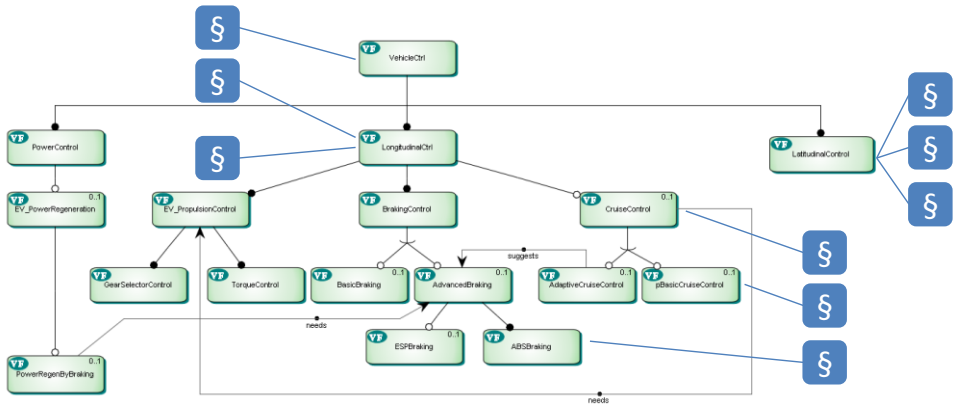

Figure 36 Feature tree decorated with requirements grouped on the relevant features

7.11 Requirements and Functions Viewpoint

To understand functions and their requirements, a view that illustrates (a subset of) the functions and the information flow between functions can be decorated with requirements grouped on the relevant functions. An illustration is given in Figure 37.

The view that shows requirements grouped on functions contributes to the scope of this report as defined in Section 2.2.

7.11.1 Example View

Figure 37 below shows functions and their requirements according to Satisfy relationships. It is a viewpoint that does not scale to realistic sets of requirements, but filtering (showing only a subset of requirements) is a way to keep the number of requirements in the view manageable. Such views are also good entry points into a model: Once the overall picture is communicated and understood through a graphical view, a tree view or list view can be used for the full scale work.

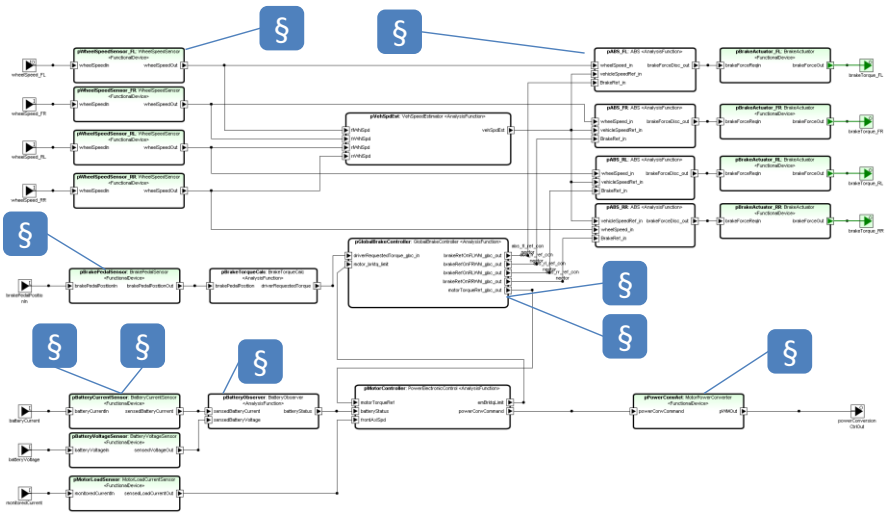

Figure 37 Functions with information flow, decorated with requirements grouped based on the relevant functions

7.12 Requirements and Components Viewpoint

To understand the modelling of the system on a hardware level and the corresponding requirements, a useful view is to show the hardware components and their signals in a schematic, and then decorate that schematic with requirements grouped on the hardware components that the requirements are allocated to. Figure 38 illustrates the view. Similarly, a viewpoint could be defined for all model components other than functions and features.

This view for showing requirements allocated on components contributes to the scope of the report, see Section 2.2.

7.12.1 Example View

Figure 38 shows hardware components and their requirements according to Satisfy relationships. It is a viewpoint that does not scale to realistic sets of requirements, but filtering (showing only a subset of requirements) is a way to keep the number of requirements in the view manageable. Such views are also good entry points into a model: Once the overall picture is communicated and understood through a graphical view, a tree view or list view can be used for the full scale work.

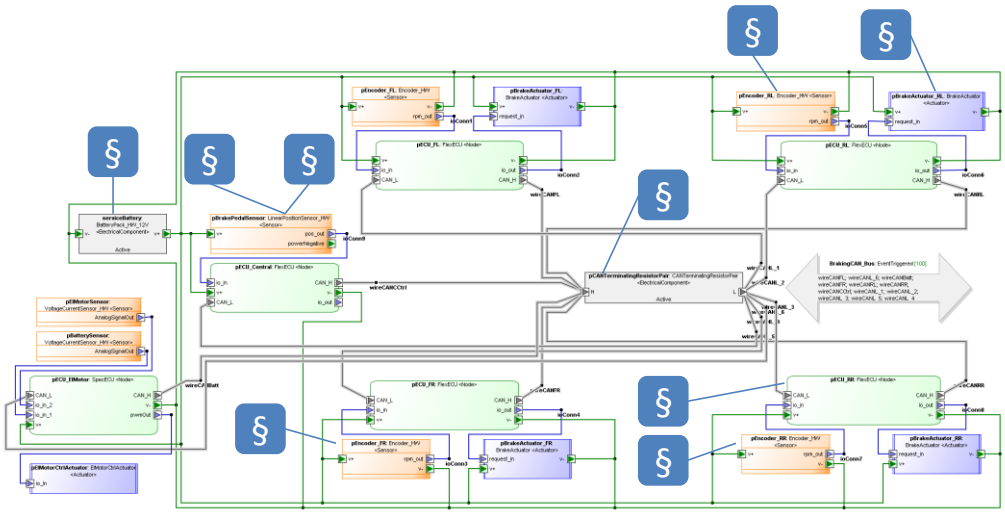

Figure 38 Hardware components and electrical signals in a schematic, decorated with requirements allocated to the hardware components and signals

7.13 Requirements Analysis Viewpoint

As described in the context of RfQ in Chapter 5, requirements analysis is a key activity when a supplier is responding to RfQ. It involves reviewing the requirements with respect to whether they are satisfied or can be satisfied by the system that the supplier is meaning to write a quotation for. To keep a record of the analysis, the requirements that are satisfied are allocated to the system model elements that satisfy them, using the Satisfy modelling element. Consequently, the requirements analysis activity produces the traceability that can be shown in the viewpoints discussed in Section 7.10, Section 7.11 and Section 7.12.

The analysis typically operates one or a few requirements at the time, beginning with requirements on an abstract level, because this is easier than to address more specific requirements. The previously allocated requirements on a higher abstraction level serve as guide to which parts of the system model the more specific requirements are likely to correspond to, employing the traceability links in the requirements specification and in the system model.

For analysis, it is considered appropriate to compare requirements specification and system model, side by side, while narrowing down what to show in the requirements specification and in the system model by applying search queries and filters.

7.13.1 Example View

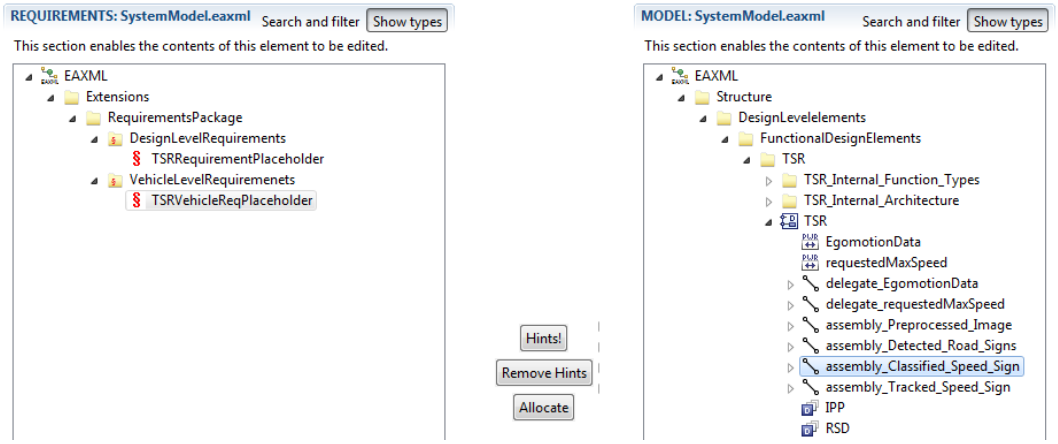

Figure 39 Example view for analysis of requirements

7.13.2 Related Modelling Concepts

Key modelling concepts for navigating requirements specifications and system models across abstraction levels are the relationships DeriveRequirement, for specifying requirements, and Realize for specifying system model elements. The relationship Satisfy is used to allocate requirements to system model elements that satisfy them.

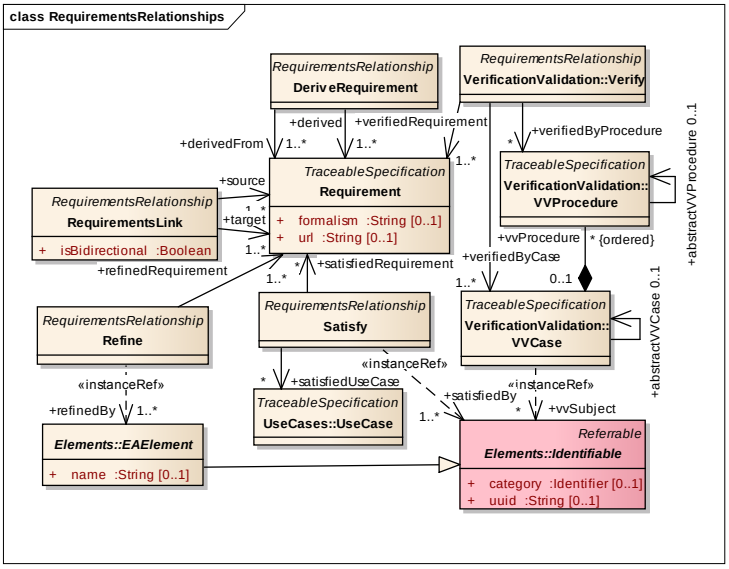

Figure 40 EAST-ADL concepts for the DeriveRequirement and Satisfy relationships.

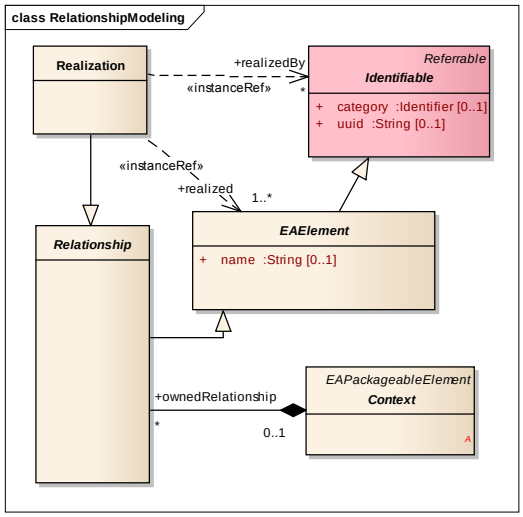

Figure 41 EAST-ADL concepts for Realize relationships that show how modelling elements specify elements on a higher abstraction level

7.14 Function to Architecture Allocation Viewpoint

To understand which functions are allocated to what component of the hardware architecture, a useful view is the table that shows the mapping between components of the hardware architecture (in rows) and functions (in columns). The same can be done for function allocation to the software architecture.

This view contributes to the scope of the report (see Section 2.2) as a way to illustrate how functions are allocated onto elements of architecture.

7.14.1 Example View

Figure 42 shows an example view, where hardware components in the columns/X axis are marked with a cross for each function (in the rows/Y-axis) allocated to it with FunctionAllocation relation.

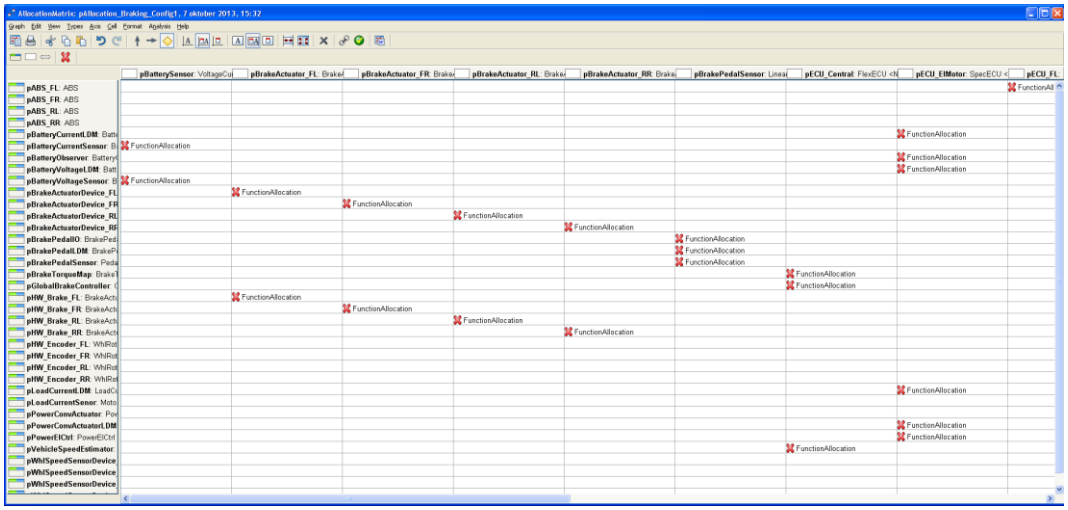

Figure 42 Functions allocated to nodes in a distributed system

7.14.2 Related Modelling Concepts

Figure 43 shows elements relevant for showing in an allocation view. The specializations of design functions and hardware components can also be used, but are omitted in the figure.

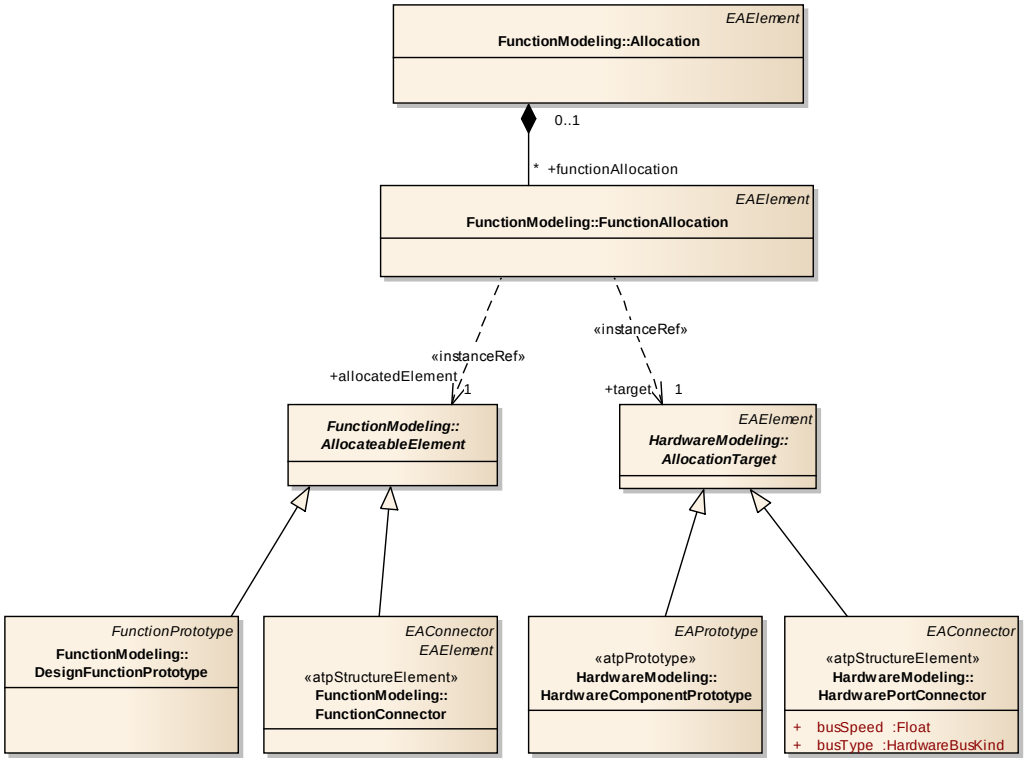

Figure 43 Allocation targets and subjects

7.15 Feature Realization Viewpoint

The viewpoint shows a given System and Software Architecture with regard to elements that realize a certain feature. This contributes to the scope of the report as defined in Section 2.2.

Architectural elements, such as functions, sensors or ECUs have the responsibility to realize one or several Vehicle Features. To identify which architectural elements are related to a certain Vehicle Feature, it is useful to have a view that hides other elements or highlights the relevant elements.

7.15.1 Example View

Figure 44 shows how elements that realize a selected feature are highlighted.

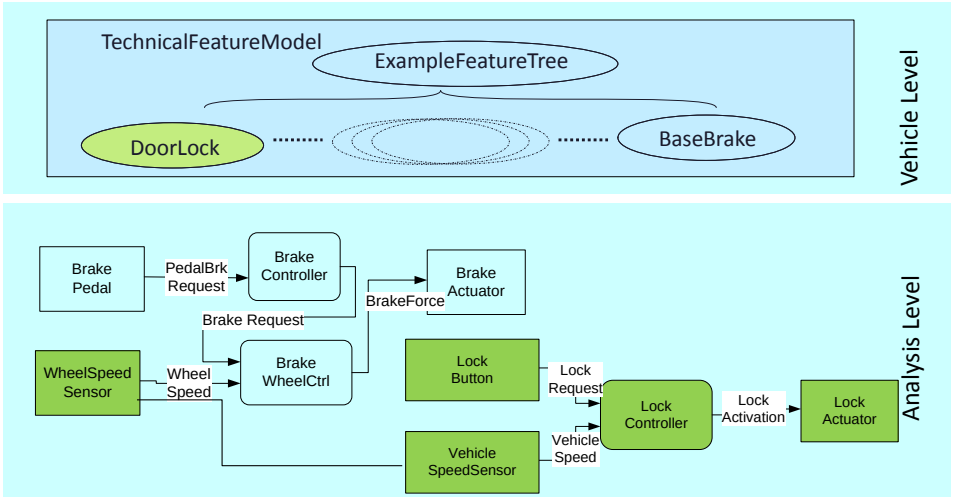

Figure 44 Highlighting elements that realize a specific feature.

7.15.2 Related Modeling Concepts

The elements that realize a specific feature are found by following the realize relations from the feature to elements on the analysis, design or implementation level. This contributes to the scope of the report as defined in Section 2.2.

Figure 45 shows the relevant meta-model excerpt.

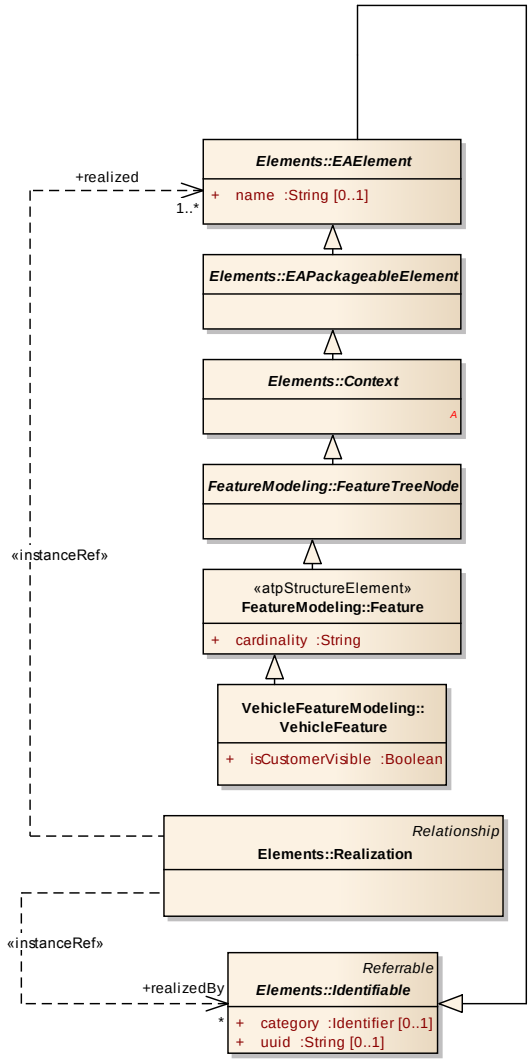

Figure 45 Realization relation between Identifiable and EAElement.

7.16 Combined Viewpoint of Requirements and Architectural Components

Requirements are normally related to specific parts of the functional, hardware and software architecture. It is expected that the allocation of requirements can make the requirement set easier to understand and manage. Viewpoints that present the requirements according to the system hierarchy are thus beneficial.

This combined viewpoint shows relevant attributes of the requirements. Furthermore, requirements are selected to be relevant for the functions and system modelling components that are shown in the combined viewpoint (see an example view in Figure 46). In these two aspects, this view contributes to the scope of the report as defined in Section 2.2.

7.16.1 Example View

Figure 46 shows requirements as they are allocated to architectural components.

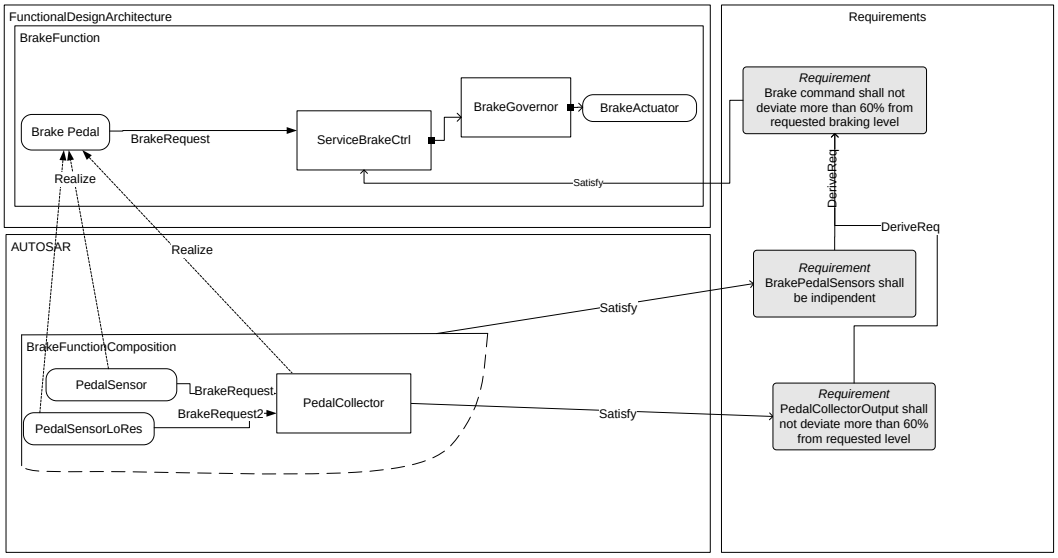

Figure 46 Example of requirements and architectural elements shown together.

7.16.2 Related Modelling Concepts

The combined viewpoint of requirements and architectural components relies on requirements (Requirement), architectural elements (Identifiable) and the Satisfy relation. Figure 17 shows how these elements relate.

7.17 Software Architecture Viewpoint

This viewpoint is there to show the architecture of distributed software. In doing so, it contributes to the scope of the report as defined in Section 2.2.

On the analysis, design and implementation level, functional and software components (for example a set of communicating applications on an AUTOSAR platform) are in some activities modelled as components with ports and connectors to represent connectivity and how data may flow. With many components, many signals and various types of ports (sender/receiver, client/server, modes) the complexity easily increases, leading to a system that is difficult to make sense of, unless something is done to provide overview.

7.17.1 Example Views

Depending on the size of the model the information shown by a viewpoint could have varying degrees of advantages. A small model is much easier to process than a large one. Furthermore, the larger the model the more information the user would be have to process and too large a model might be disadvantageous rather than providing any benefits. Figure 47 attempts to illustrate the main issue with providing an example of a view which illustrates a fairly large model. Although the view gives all the information the user might need in order to understand the flow of signals throughout the model the sheer amount of input one has to process makes the view nearly impossible to understand. The connections alone are so abundant they obscure whatever useful

information one might be able to gain from such a view. Certainly, information overflow is an issue here and it should be beneficial to limit the input provided to the viewpoint or give alternatives in how to view the model.

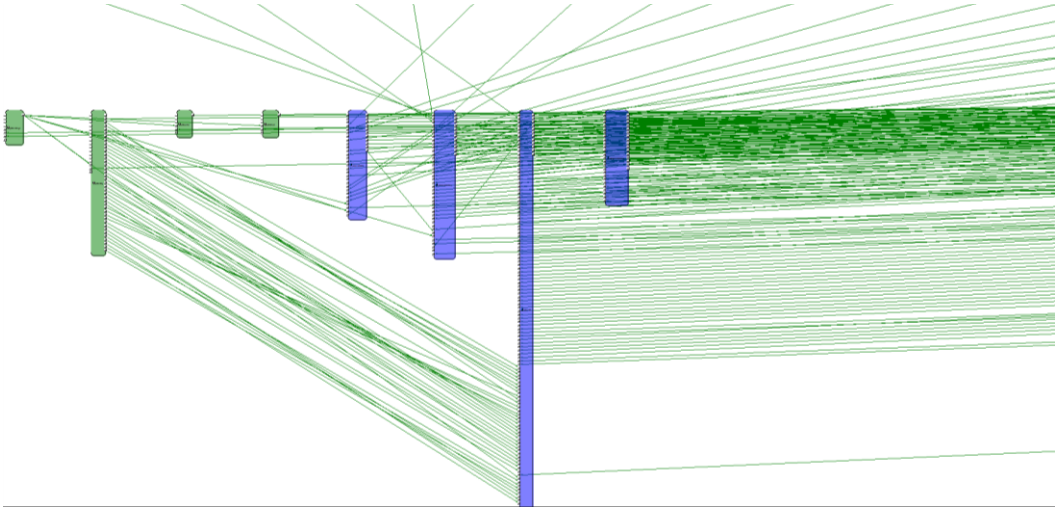

Figure 47 Large example of Components and Connections

Figure 48 demonstrates a situation which is much simpler to process. Here the components are not as large and the number of connections is rather limited. Despite many connections crossing components and other connections it is still fairly simple to gain an understanding of how the signals flow through the model. Obviously, not all models are as small as this one, Figure 47 clearly demonstrated a more complex System.

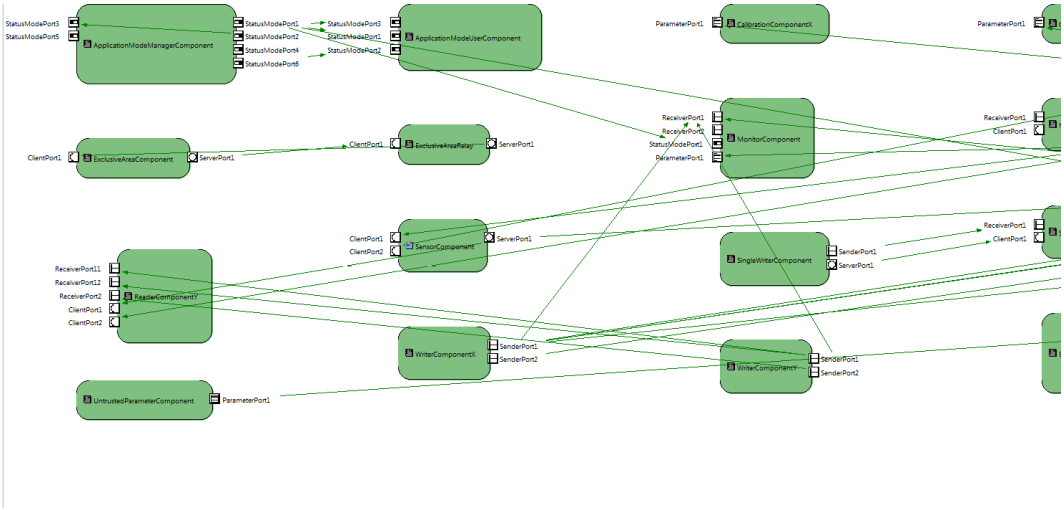

Figure 48 Small Example of Components and Connections

Consequently, here is where considerations could be made to improve how input is handled and limit the parts of the model shown by the viewpoint. Consider for example if one would only show a small part of the whole model demonstrated in Figure 47, perhaps the input to the viewpoint could be only a few select components which are relevant at the time. The user could specify more precisely which parts of the system which should be shown and the viewpoint would only render

those. Certainly, such improvements would simplify the generated view and make the handling of such a view a simpler matter for whoever needs to view it.

Furthermore, it might be beneficial to add usability tools to the viewpoint which could enable to user to obscure information, such as connections or ports which is at the time irrelevant. Figure 49 demonstrates a more focused view of that same example shown in Figure 47, here all irrelevant connections have been hidden, and the important component for this case has been put into focus. Consequently, a user is no longer burdened by information overload and can be more focused on processing the relevant data at hand.

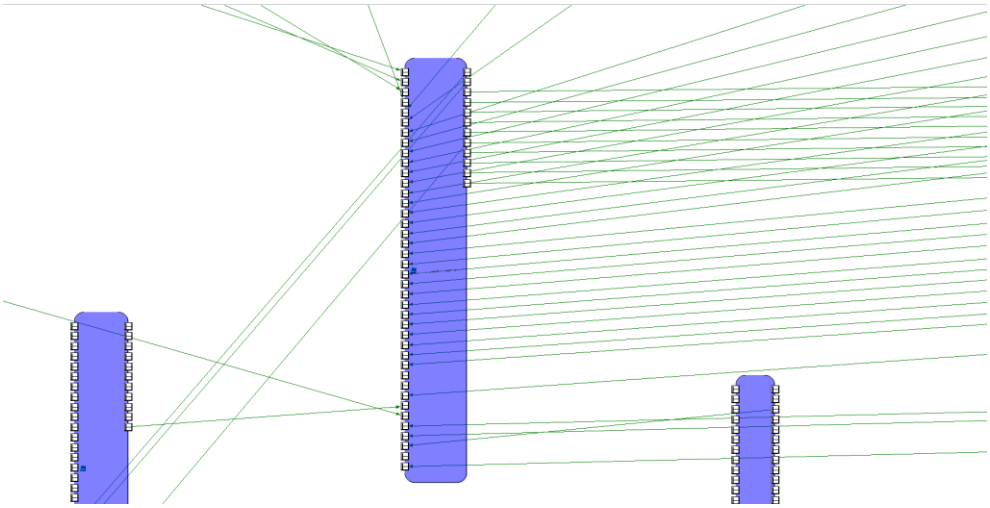

Figure 49 Focused view of large example in Figure 47

Even further tools could be provided to the user which could enable filtering away unnecessary information. Imagine if the user could simply open up a search window and type in a couple of key words and the viewpoint would then remove any components, ports and signals which do not match the result. It is important that it is context sensitive; certain restraints would have to be put in place in order to not obscure too much information. For example, if a user searches for a unique component name it would be disadvantageous to show only that component and hide everything else. Understandably, every connection and port associated with that component should also be shown.

A final mechanism to consider is the use of bus lines and goto which exists in Simulink. The EAST-ADL concept of port group could be used to deduce which connectors can be combined to a single, thick line: All connectors from a certain port group can be combined, and only split up when approaching destination ports and port groups. Goto blocks are not supported by EAST-ADL, but the concept could be reproduced as a diagram functionality in a specific tool: Instead of lines between ports, the connector is represented by lines plus goto blocks to avoid long and crossing lines.

7.17.2 Related Modeling Concepts

EAST-ADL contains the modeling elements for components, ports and signals. However, EAST-ADL does not specify the information elements to hold the necessary information to show the view. For example, EAST-ADL has no attributes for the position and size of a rectangle for the component. To show a concrete (decorated) view, information that dictates the visual appearance of the view has to be represented using some other information than EAST-ADL. The introduction of a structured information model for diagram information is thoroughly discussed in Section 3.7.

AUTOSAR can hold more concrete information of how a system can be modeled on an implementation level. AUTOSAR contains meta-model elements which can represent concrete components, connections and ports. Components hold ports which can be interconnected with other components via connections forming AUTOSAR applications. The AUTOSAR SW component template represents a component hierarchy that is a suitable basis for the described view. As for EAST-ADL the graphical information is missing in the AUTOSAR model, but can be stored in the same complementary structure.

7.18 Safety Goals Viewpoint

ISO26262 use the concept of Item to define the “system or array of systems to implement a function at the vehicle level, to which ISO 26262 is applied”. Elements of an Item are hardware (ECUs, sensors, cables etc.) and software (functions, connectors, software components, etc.) of various kinds and represented on various abstraction levels. During item definition it is therefore useful to show the elements concerned to secure the right boundary. Further, it is useful to show the safety goals for the item, as they are defined.

A viewpoint to show the safety goals for a certain item contributes to the scope of this report defined in Section 2.2.

In a FunctionalDesignArchitecture, HardwareDesignArchitecture or SoftwareComponentTemplate, it is thus useful to hide all elements that are not included in the item that is inspected.

7.18.1 Example View

Figure 50 shows an example of how features and functions relate to an item and gives an idea about how an item subset of a FeatureTree or Functional Architecture can be derived.

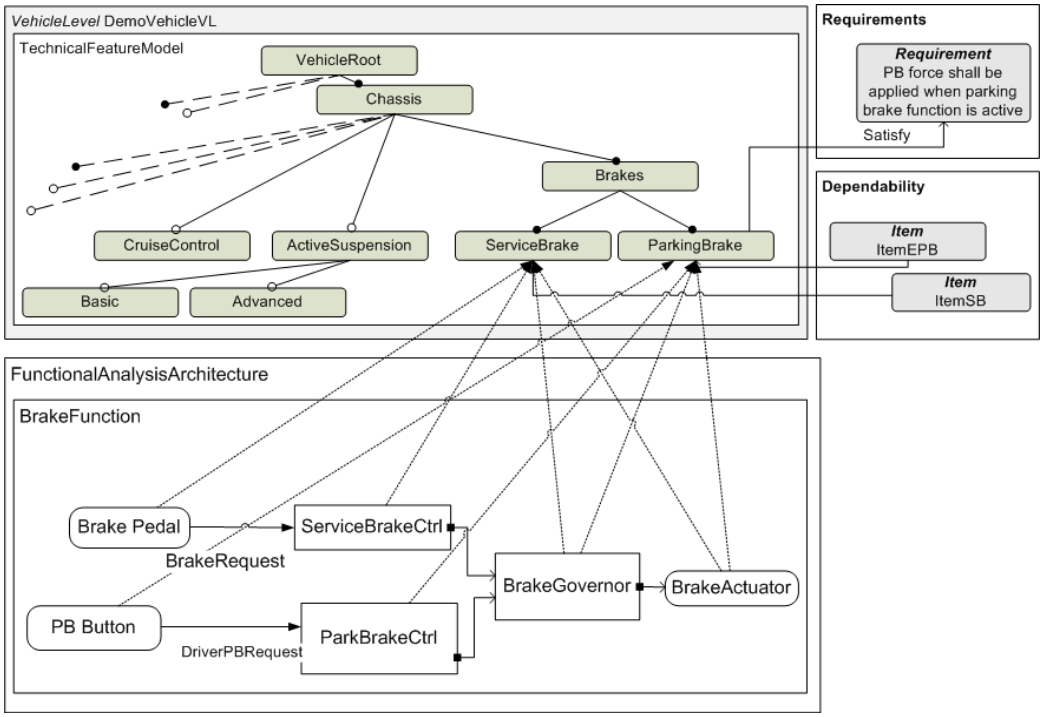

Figure 50 Example of an Item Definition View

7.18.2 Related Modeling Concepts

Figure 51 shows EAST-ADL model elements relevant for the view.

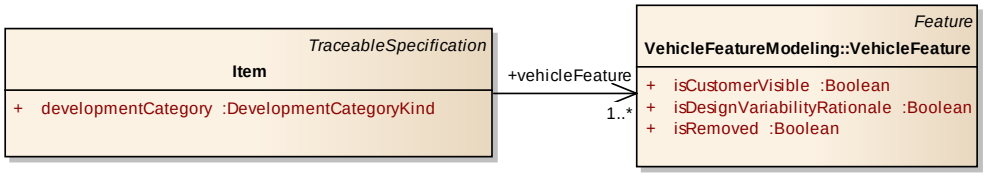

Figure 51 Item and its reference to one or several VehicleFeatures

The Item element references VehicleFeatures, which in turn are referenced by all elements that realize the feature. This means that any EAST-ADL or AUTOSAR element may be included in an item. Figure 52 shows how an identifiable (such as an AUTOSAR SWC or EAST-ADL Function) realizes an EAElement (such as Feature or Function).

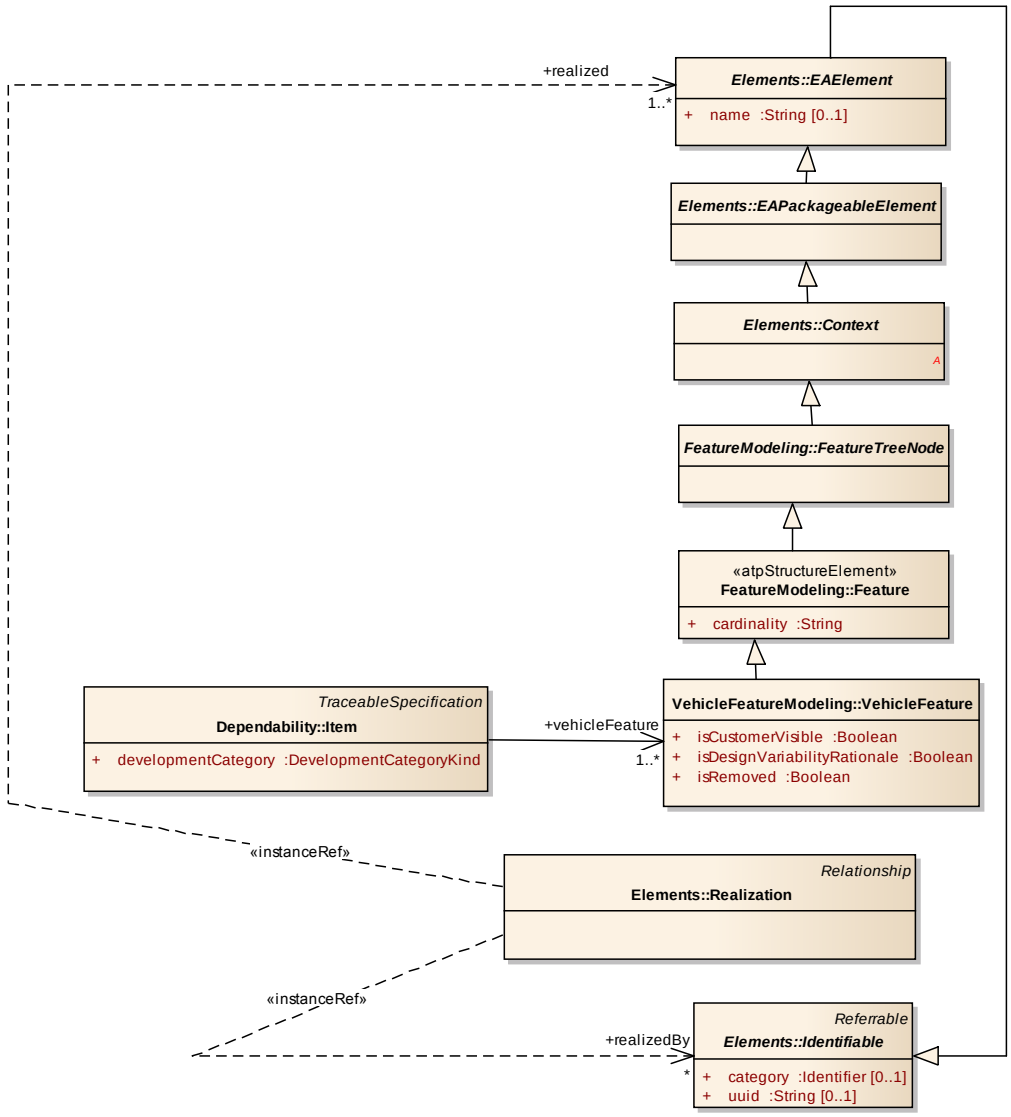

Figure 52 Realization relation between realized Element and realizing identifiable.

7.19 Functional Safety Concept Viewpoint and Technical Safety Concept Viewpoint

Visualizing the Functional Safety Concept or the Technical Safety Concept is important to assess what measures are taken to meet the safety goal. Functional and Technical Safety Concepts are composed of sets of safety requirements allocated to architectural elements.

Viewpoints for illustrating the part of the system model that correspond to the functional safety concept and the technical safety concept are contributing to the scope of this report as defined in Section 2.2.

7.19.1 Example View

Figure 53 shows how the functional safety concept may be visualized. In the upper part, its Item elements are also included, although they are not part of the functional safety concept.

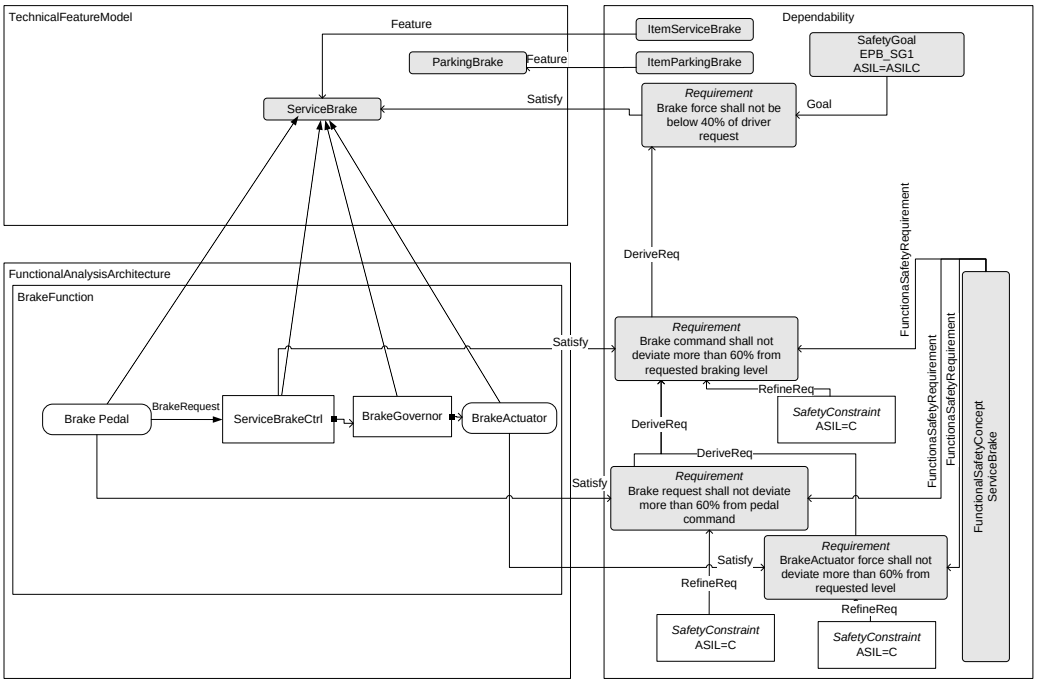

Figure 53 Functional Safety Concept

7.19.2 Modelling Concepts

The views of Functional Safety Concept and the Technical Safety Concept are based on the elements with the same names. Figure 21 above shows the elements and relations.

7.20 Safety Analysis Tool Support and Viewpoint

Error propagation models have the purpose of defining for each component (functional or physical, concrete or abstract) the expected internal faults and expected faults from adjacent components, the error propagation logic and the resulting potential failures of the component. This is closely related to views to support safety analysis, since the information of possible error propagation leads to the construction of a fault tree on with fault tree analysis is performed.

A view to support error propagation analysis and a view that shows a fault tree contribute to the scope defined for this report in Section 2.2.

7.20.1 Example View

Figure 54 shows a nominal model with one plausible error propagation model. The safety engineer may adjust the error propagation model further, which is why they are separated as two structures.

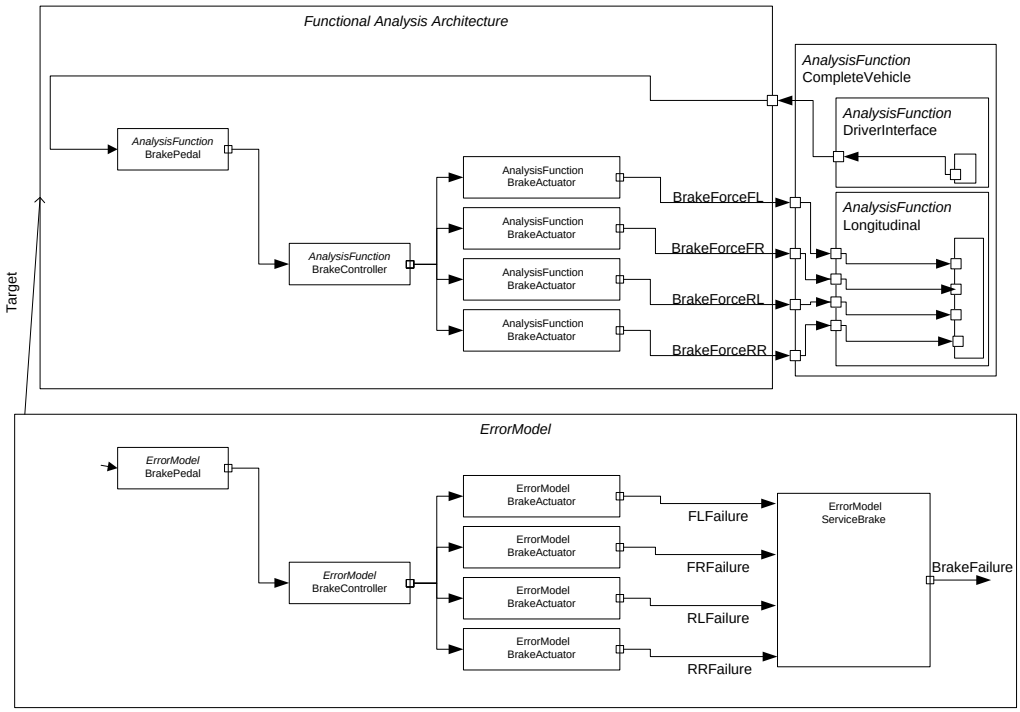

Figure 54 Example of nominal model (top) and its error propagation model

Another viewpoint of the error propagation model is to consider the failure logic of each block and derive a Fault Tree, see Figure 55

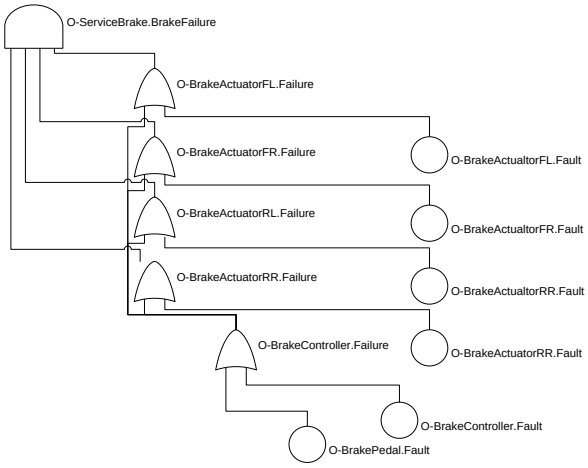

Figure 55 Fault Tree view of the ErrorModel

7.20.2 Related Modeling Concepts

Error propagation is modeled using the ErrorModelType hierarchy, as described in Figure 56.

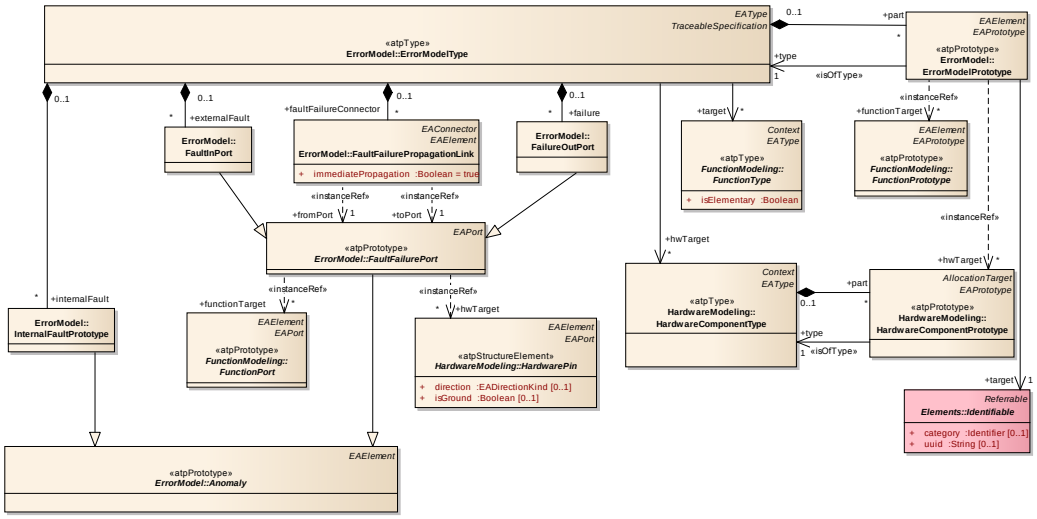

Figure 56 Error propagation modelling is based on the ErrorModelType hierarchy

To derive and show a fault tree, the ErrorBehavior of the error model types must be known. Figure 57 shows the corresponding EAST-ADL elements.

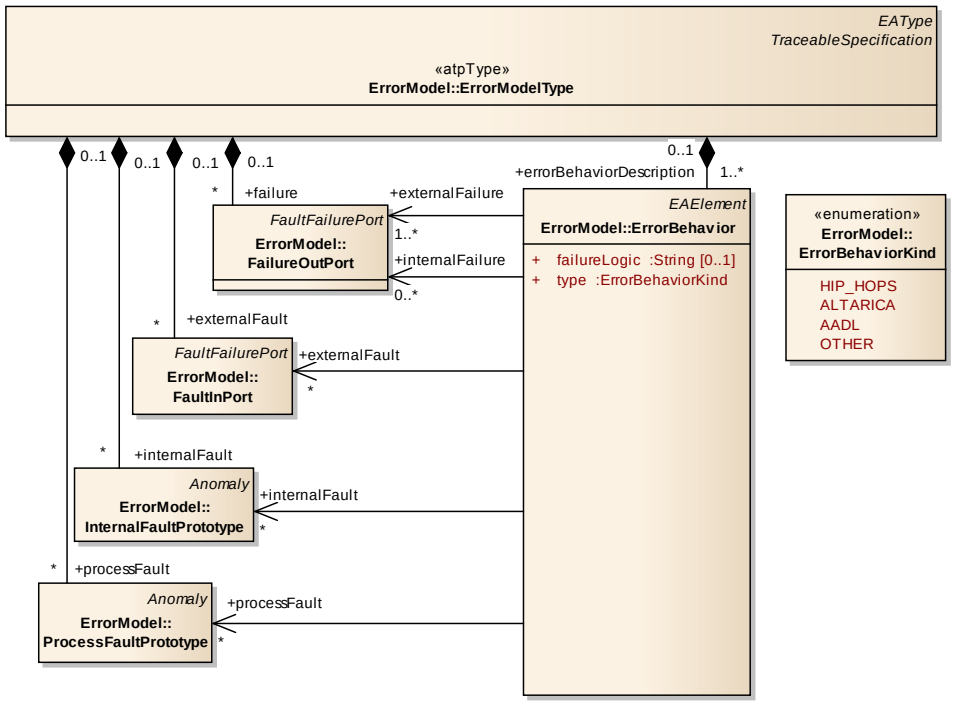

Figure 57 ErrorBehavior with failureLogic is the basis for defining error propagations.

The Error Propagation Model and the related Fault Tree Analysis are two views used to perform analysis which generates artefacts for the Functional Safety Concept and for the Technical Safety Concept.

7.20.3 Work-Phases when Eliciting Error Propagation Models

Working with Error Propagation models can be considered in four phases: Defining, Analysing, Understanding and Evolving.

7.20.3.1 Defining

The first step, defining, is to create the error propagation model from the nominal architecture model. Decisions are needed to determine which or what level of components are going to be used as a basis for error propagation model. There are also engineering decisions needed in order to define an efficient error propagation model, for example by identifying the right granularity. One or several Error Model components may capture the error propagation behaviour for a part of the architecture. The kind of errors considered will also be a critical engineering decision. While a lot of engineering decisions are involved, synthesis and micro-automation support from the tooling is important for efficiency.

A feasible approach to assist error propagation modelling is to start with a 1-1 auto-generation of an error propagation model based on the nominal architecture. Because error propagations may occur beyond direct inter-component links, a manual effort is required to modify the error propagation model. Similarly, where multiple nominal links connect two components, their error propagation characteristics may be captured by a single link and a set of nominal components may be represented by a single error propagation component. For these reasons, collapsing of components and ports need to be done and tool assistance is good.

7.20.3.2 Analysing

The second step is the analysing process, by means of fault tree analysis [5]. In this step, the fault propagation data in the error model would be synthesized and analysed into a fault tree. It should be noted that fault tree analysis is a wide-spread analysis method with a well-known view. For this reason, the viewpoint is not further described here. The analysis tool would automatically generate the corresponding fault tree once the error model is decided.

Different analysis tools and notations (AltaRica, HiP-HOPS, Rodelica, AADL) are available, and the failure logic must be defined accordingly.

7.20.3.3 Understanding

To understand the generated error propagation model and see its relation to a nominal model, efficient viewpoints are needed. This relates both to the error propagation model, and the corresponding fault tree or FMEA table. By visualizing failures, failure propagation paths, etc. it is possible to see (c.f. Figure 54) how the system elements relates to the identified faults and error propagations.

7.20.3.4 Evolving

After the initial definition of models, and subsequent analysis, there will be adjustments of the nominal model and of the error propagation model. To secure consistency, modifications of the nominal model should result in the modification of the error model and vice versa.

7.21 Illustrating Metrics

It is useful to decorate various views of the system model with annotations of metrics, to see the status or progress status of system model components as well as their contribution to product metrics such as dependability. There are also views just to show metric values, often called a cockpit view.

Viewpoints that annotate metrics to system model illustrations or provide a range of metrics contribute to the scope of this report as defined in Section 2.2.

7.21.1 Example View

Figure 58 shows an example of a cockpit view that illustrates profiling of six contributors to the subsystem's latency (left) and an overview of the relative number of requirements that have been rejected.

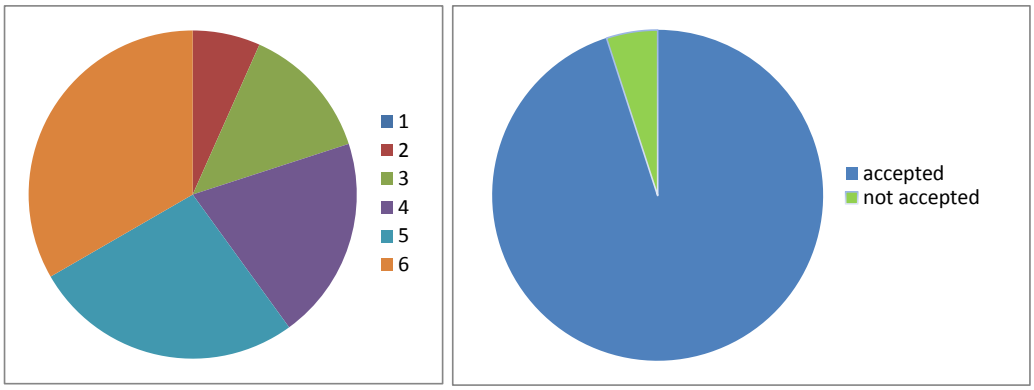

Figure 58 Aggregation of metrics in a cockpit view

Figure 59 shows how metrics can be used to give an overview of the progress on implementing the subsystem components. Green components have progress according to plan, while light-blue components are lagging behind.

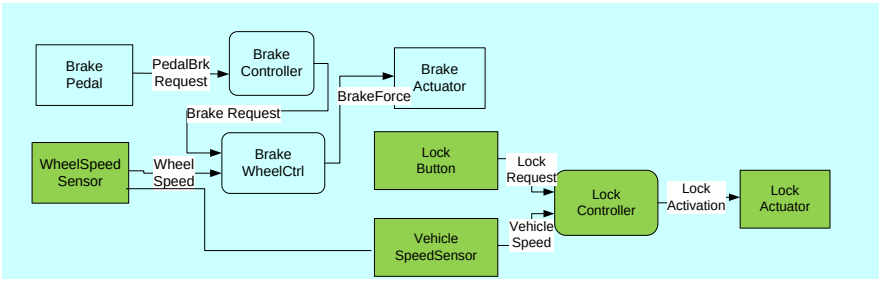

Figure 59 Alternative view for illustrating metrics

8 Efficient Development in the Presence of Complexity

Conceptually, complexity can have negative impact on development efficiency in various ways and here we categorize these ways.

- Lack of understanding of the developed system
  Impact: Tasks are incompletely done and mistakes are made that are caught only by extensive reviews. The impacts of changes to the developed system are difficult to assess, or it is time-consuming to determine the impacts of changes.
    - Addressed in Project Synligare by
        - Representing the developed system in accordance with a structured information model that enables a structured way of keeping information about the developed system, using hierarchies and traceability such that it is always possible to connect a piece of information with a higher level of abstraction where the system should be more understandable.
        - Enabling various views. Especially diagrams that can show the structure of the system and the relationships between system elements.
        - Enabling searching and filtering based on the structured information model, which helps focusing on relevant information and to learn about the system.
        - Making views and metrics automatically generated and calculated, to speed up the process of determining the impact of change.
- Lack of confidence in one's own understanding of the developed system
  Impact: Engineer working on developing the system will not feel confident that a task is completed.
    - Addressed in Project Synligare by:
        - Same support for seeing the structure of the system and learning about the system as described above.
- Misunderstandings due to discrepancy between the views of the developed system between communicating parts
  Impact: Engineers and managers who have different roles naturally work with different subsets of the information about the developed system. There is misunderstanding between them because they have different understanding of the system.
    - Addressed in Project Synligare by:
        - Well-defined structured information model that is clear about what modelling concepts mean.
        - Role-specific views that are adjusted for each role and task such that relevant information is shown. The semantics of the views are clearly defined so that there is no misunderstanding of what is shown.
- Lack of understanding of the progress of the development project
  Impact: Efforts are inefficiently invested based on people's flawed impressions of the status of various tasks.
    - Addressed in Project Synligare by:
        - Automatically calculated metrics annotated on a work breakdown structure show what parts of the system have been developed to acceptable progress and what parts are lagging behind. This enables better, more informed, decisions about how to invest efforts.

- Discrepancies between system information in various tools
  Impact: Each role or task has a task-specific tool which holds a local copy of a subset of the information about the system. A change in the local copy is not synchronized to other tools' local copies. Any errors that occur are caught by thorough reviews.
    - Addressed in Project Synligare by:
        - A single structured information model that can be introduced as each tools representation of system information. This enables more efficient synchronization and errors are easier to detect.
- Review time
  As mentioned above there process of reviewing is time-consuming. Further, due to the amount of information to review for a complex system, reviewing is extra time consuming.
    - Addressed in Project Synligare by:
        - A structured information model that encourages reuse and the principle to only represent the same information in one place. For example, identical redundant elements are described once and instantiated as many times as needed.

From the above it can be seen that the concepts described in this report indeed can be expected to increase efficiency in collaborative system development in the presence of complexity, in comparison with a system development project that does not employ the concepts.

9 Summary

This chapter summarizes the outcomes of WP2. In so many words, it was described how the work in this report has identified invormation modelling concepts that can be used towards the goals of Project Synligare. Further, it was described how viewpoints and metrics have been defined to ease collaboration and make system development more efficient in the presence of complexity. This has been explicitly related to the RfQ-processes of agreeing on collaborations between OEMs and Tier-1s.

The scope of the report has been detailed in Chapter 2, with a listing of what viewpoints and metrics that are mentioned in the report.

Information modelling concepts have been reviewed in Chapter 3. Four needs were mentioned, namely support for model distribution, simplification of version control, support for model exchange and model separation into architectural levels. EAST-ADL was reviewed and it was found that it provides model separation into architectural levels. However, EAST-ADL does not provide simplification of version control, since it does not have any dedicated property for version, time stamp or status. A model structure involving EAST-ADL, AUTOSAR and GraphML can be defined such that model parts can be handled as separate configuration items. This would be a step towards simplification of version control. Further, Project Synligare presents a way to use User Defined Attributes to hold a version number per modelling element, and a tool to read and write such information. In terms of model exchange, three scenarios with increasing complexity were defined. It was determined that Project Synligare should aim to towards the second most complex scenario, namely a round-trip of model information without merge of models. Concerning support for model distribution, it was shown how Project Synligare is working with three tools, namely SystemWeaver, Enterprise Architect and EATop to enable various parties to work on different parts of a system model.

Concepts for prototype tools to show the viewpoints and metrics have been described in Chapter 3. In particular, on top of EAST-ADL, there are identified needs for additional information modelling for version information (Section 3.4) and diagram modelling (Section 3.7).

The interaction between OEM and Tier-1 in a collaborative development project has been analyzed in Chapter 5 to identify exchanges of information, with and without employing the benefits of model based development and common information models in Section 5.1 and Section 5.2 respectively. Metrics and viewpoints have been conceptually described, in Chapter 6 and Chapter 7 respectively, such that they can support the activities that are relevant to the interaction between OEM and Tier-1.

Chapter 6 lists the metrics that have been defined. They are of two main types, namely progress metrics and product metrics. The progress metrics concern requirements, verification, realization, and safety-related progress. The product metrics concern quality attributes like performance.

Chapter 7 lists the conceptual viewpoints that have resulted from WP2. Notable examples of viewpoints that are described include Compare and Merge Support and Viewpoint, Feature Tree Viewpoint, views for requirements and modelling elements that satisfy them, Safety Goals Viewpoint and Safety Analysis Tool Support and Viewpoint.

Chapter 8 describes how Project Synligare addresses “bridging” complexity in collaborative system development. Potential negative impacts of complexity are categorized and for each category it is described how concepts from this report help to remove or mitigate that negative impact of complexity.

9.1 Inputs to WP3

The considerations for tooling in Chapter 3 list explicit requirements on tools that are to be based on the considered information modelling.

The metrics in Chapter 6 and the conceptual viewpoints in Chapter 7 set the starting point for developing the corresponding views in tool prototypes.

10 Bibliography

[1] Synligare Consortium, "Synligare Deliverable D1.1 Needs Identification," Volvo Technology AB, Gothenburg, 2014.

[2] EAST-ADL Association, "EAST-ADL Schema eastadl_2-1-12.xsd," [Online]. Available: http://www.east-adl.info/. [Accessed 29 09 2014].

[3] EAST-ADL Association, "Specification," [Online]. Available: www.east-adl.info. [Accessed 29 June 2014].

[4] International Organization for Standardization, "ISO/FDIS 26262 Road vehicles -- Functional Safety, part 1 to 9," 2011.

[5] S. Henry and D. Kafura, “Software structure metrics based on information flow,” *IEEE Transactions on Software Engineering,* Vols. SE-7, no. 5, pp. 510--518, 1981.

[6] C. Ericson, "Fault Tree Analysis - A History," in *Proceedings of the International System Safety Conference*, 1999.

[7] AUTOSAR Development Partnership, "AUTOSAR Specification," [Online]. Available: http://www.autosar.org/. [Accessed 29 09 2014].

[8] AUTOSAR Tool Platform User Group, "AUTOSAR Tool Platform," [Online]. Available: http://www.artop.org/. [Accessed 29 09 2014].

[9] EATOP Eclipse Open Source Project, "EAST-ADL Tool Platform," [Online]. Available: http://www.eclipse.org/eatop. [Accessed 29 09 2014].

[10] Synligare Consortium, "Synligare Project Website," [Online]. Available: http://www.synligare.eu/. [Accessed 29 09 2014].

[11] Synligare Consortium, "Synligare Deliverable D4.1 Validation," 2014.

[12] "The GraphML File Format," [Online]. Available: http://graphml.graphdrawing.org/index.html. [Accessed 26 June 2015].

[13] "The yEd editor," [Online]. Available: https://www.yworks.com/en/products/yfiles/yed/. [Accessed 26 June 2015].

Formatted: English (U.S.)

11 Acknowledgements

Figures in this deliverable were based on diagrams in EnterpriseArchitect from Sparx, MetaEdit+ from MetaCase as well as various tools and views developed by the Synligare partners.