EAST-ADL

EAST-ADL –

An Architecture Description Language for Automotive Software-Intensive Systems

White Paper

Version M2.1.10

Hans Blom, Henrik Lönn (Volvo Technology, SE), Frank Hagl (Continental, DE), Yiannis Papadopoulos (University of Hull, GB), Mark-Oliver Reiser (Technische Universität Berlin, DE), Carl-Johan Sjöstedt, De-Jiu Chen (KTH Royal Institute of Technology, SE), Ramin Tavakoli Kolagari (Ohm Hochschule, DE)

Abstract

This White Paper gives an overview to EAST-ADL. The document is intended for engineers that need a short introduction to the language, through descriptions and examples.

EAST-ADL is an Architecture Description Language (ADL) initially defined in the ITEA project EAST-EEA around 2000. Subsequently, several national and international funded projects have refined the language, and it is now aligned with the more recent AUTOSAR automotive standard. It provides a comprehensive approach for describing automotive electronic systems through an information model that captures engineering information in a standardized form. Aspects covered include vehicle features, requirements, analysis functions, software and hardware components and communication. The representation of the system's implementation is not defined in EAST-ADL itself but by AUTOSAR. However, traceability is supported from EAST-ADL's lower abstraction levels to the implementation level elements in AUTOSAR. In this article we describe EAST-ADL in detail, including a case study to show how it relates to AUTOSAR as well as other significant automotive standards and present current research work on using EAST-ADL. in the context of fully-electric vehicles, the functional safety standard ISO 26262 and for multi-objective optimization.

Table of contents

1 Introduction

EAST-ADL represents an Architecture Description Language (ADL) initially defined in the European ITEA EAST-EEA project. It was subsequently refined and aligned with the more recent AUTOSAR automotive standard [2] in national and international funded projects including the ATESST and MAENAD projects [1], [5]. It is maintained by the EAST-ADL Association [3].

EAST-ADL is an approach for describing automotive electronic systems through an information model that captures engineering information in a standardized form. Aspects covered include vehicle features, functions, requirements, variability, software components, hardware components and communication.

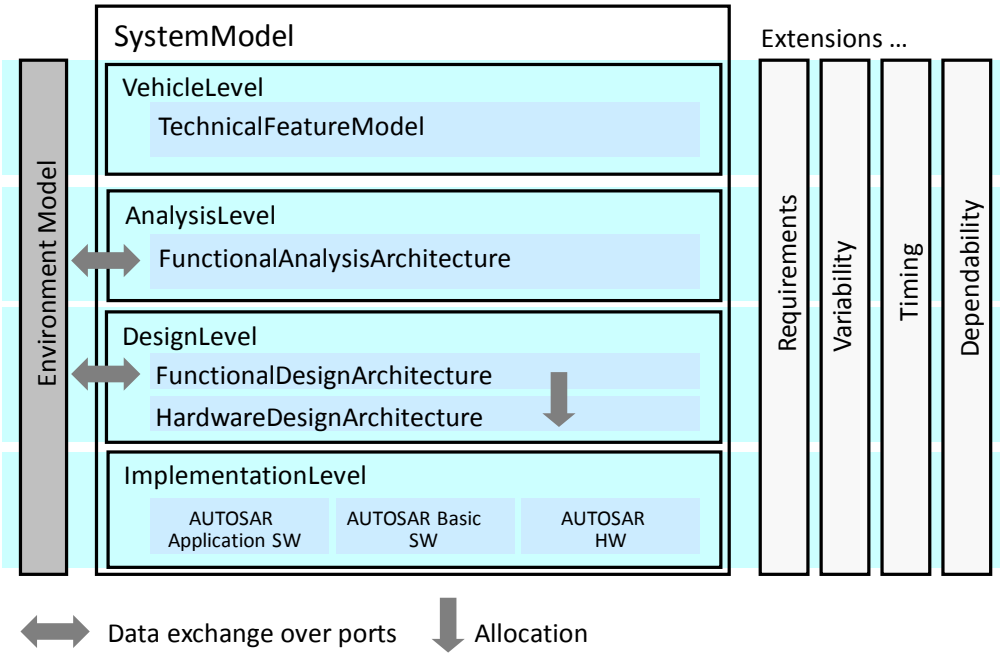

Figure 1: The EAST-ADL's breakdown in abstraction levels (vertically) and in core system model, environment and extensions (horizontally).

The software- and electronics-based 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 (see Figure 1):

- **Vehicle Level**
  Feature trees characterizing the vehicle content as it is perceived externally.
- **Analysis Level**
  An abstract functional architecture defining the embedded system from a functional point of view.
- **Design level**
  The detailed functional architecture allocated to a hardware architecture.
- **Implementation level**
  The implementation of the embedded system represented using AUTOSAR elements

Each of the abstraction levels has a specific role. From the Vehicle Level stating *what* the vehicle should do through analysis, design and implementation levels that define, at various level of abstraction, *how* this is done. Features on the vehicle level allow the organization of vehicle

content in a solution-independent way. Requirements can be linked to features such as markets and brands as well as to technical features such as Wipers or Brakes.

The proposed abstraction levels and the contained elements provide a separation of concerns and an implicit style for using the modeling elements. The embedded system is complete on each abstraction level.

The EAST-ADL extensions include requirements, variability, safety, behavior, timing, and generic constraints. Such elements reference the core elements in all abstraction levels. The behavior extension support modes, which allows different requirements, behaviors and constraints to be active at different times.

The core model at the Vehicle Level is organized around *features* in the *TechnicalFeatureModel.* These represent the vehicle from a top-level perspective without exposing the realization. It is possible to manage the content of each vehicle and entire product lines in a systematic manner using sets of feature models with relations like needs and exclude

A complete representation of the electronic functionality in an abstract form is modeled in the *Functional Analysis Architecture* (FAA). One or more entities (analysis functions) of the FAA can be combined to realize features in the *TechnicalFeatureModel*. The FAA captures the principal interfaces and behavior of the vehicle's subsystems. It allows validation and verification of the integrated system or its subsystems on a high level of abstraction. Critical issues for understanding or analysis can thus be considered, without the risk of them being obscured by implementation details.

The implementation-oriented aspects are introduced while defining the *Functional Design-Architecture* (FDA). The features are realized here in a function architecture that takes into account efficiency, legacy and reuse, commercial off-the-shelf components, hardware allocation, etc. The function structure is such that one or more functions can be subsequently realized by one or more AUTOSAR software component (SW-C). The external interfaces of such components correspond to the interfaces of the realized functions.

The *Hardware Design Architecture* (HDA) should be considered parallel to application development. On the design level and down, the HDA forms a natural constraint for development and the hardware and application software development needs to be iterated and performed together. There is also an indirect effect of hardware on the higher abstraction levels. Control strategies or the entire functionality may have to be revised to be implemented on a realistic hardware architecture. This reflection of implementation constraints needs to be managed in an iterative fashion.

The representation of the implementation, the software architecture, is not defined by EAST-ADL but by AUTOSAR. Traceability is supported from implementation level elements (AUTOSAR) to vehicle level elements. Further, the EAST-ADL extensions for ISO26262, requirements, variability, etc. can be applied to the AUTOSAR elements. Traceability through the extension elements are applicable through the abstraction levels all the way down to implementation level.

As a complement to the above, an environment model is required for verifying and validating features across all abstraction levels. Verification could for example be carried out using simulation or formal analysis techniques.

The environment model, sometimes referred to as plant model, captures the behavior of the vehicle dynamics, driver, etc. It represents all relevant elements interacting with the EE architecture (Electrics and electronics architecture including software). This includes for example a) the mechanical and hydraulical systems in the vehicle, b) the near environment including road surface and adjacent vehicles, and c) the far environment such as road traffic informatics.

Different tasks will typically require different detail and scope of the environment model. For example, there may be a detailed vehicle and powertrain dynamics model to assess gear change performance and a low-detail dynamic model for fuel consumption assessment. The alternatives

are selected using the variant management extension or using alternative environment models for each task. The same environment model can typically be used across several abstraction levels.

After this short introduction to the EAST-ADL concepts, we go on to discuss the motivation and modeling concepts in more detail.

2 Challenges for Modeling Automotive Embedded Systems

Automotive embedded systems have evolved enormously over the past decades. The use of electronics and software in automotive products has grown exponentially. For example, today vehicles in series production contain the same amount of electronics as an aircraft did two decades ago. To satisfy customer demands and competitiveness between vehicle manufacturers, innovation will further drive the significance of software-controlled automotive electronics over the next decade. It is obvious, that the vehicle's electronic architecture will continue to grow in complexity, criticality and authority.

To manage some of the challenges of automotive software, the AUTOSAR consortium has developed a standardized automotive software architecture. One of its main features is itss support for componentization of the application software architecture, to favor reuse and assist collaboration and integration aspects. The software development effort is no longer bound to a specific hardware platform or a particular provider. A standardized software architecture and methodology is a first step towards meeting the challenges connected with the development of automotive systems, often distributed over several suppliers with different responsibilities.

However, there still remains the critical issue of managing the overall engineering information to control the system definition from the early phases. The early phases of system definition involve the most decisive steps in meeting safety challenges, controlling complexity and avoiding development errors and delays. Many stakeholders are involved here, and development is distributed over several departments and locations and involves several suppliers.

While system modeling and model-based development is the trend in the automotive industry to solve this issue, there are diverse company-specific solutions. There is no standardized approach to support system modeling of the engineering information. A federation of different modeling language initiatives is required to develop an automotive domain-specific language that is also in line with non-automotive approaches. The modelling notations involved must be integrated in a way that correctly compose to a meaningful overall model, in terms of structure and behavior and additional aspects such as safety or variability annotations.

To support complexity and facilitate component development, an adequate organization of the system model is important. Representing the system in several "models" at different abstraction levels is a way to ensure separation of concerns and allow smooth interaction between disciplines. Supporting a functional decomposition of the system is also important to hide implementation aspects while the functional aspects are addressed.

Another aspect of system model organization concerns external annotations of the core stucture, for example timing and variability. By managing such information in external packages, the metamodel as well as the user models are modular.

Another challenge is the capability to use product line engineering. Today, component reuse is state of the art in the automotive industry. The organization and structuring of a product line approach, from feature selection up to decomposition into components, requires innovative and efficient techniques.

Finally, an important challenge is assessing the dependability of the application. What is needed are means for early evaluation of system architectures, in terms not only of functional properties, but also of non-functional ones (such as timing, resource and safety level). In this context, the application of the upcoming standard for functional safety (ISO DIS 26262) must be prepared by introducing new techniques and a structured development approach. An architecture description language provides means to represent the safety life-cycle information according to the requirements of the standard.

Last but not least, tool support for engineering development is organized today as a patchwork of heterogeneous tools and formalisms. A backbone environment using a standardized modeling language has to be harmonized to drive the tool market.

3 EAST-ADL Meta-Modeling Approach

The EAST-ADL language is formally specified as a meta-model that captures domain specific (i.e. automotive) concepts. The meta-model follows guidelines originating from AUTOSAR for definition of templates. Modeling concepts are represented by the basic notions of MOF (www.omg.org/mof/) supplemented by the AUTOSAR template rules. The meta-model thus fits as a specification of a domain specific tool environment, and also defines an XML exchange format. This *domain model* represents the actual definition of the EAST-ADL language and constitutes the heart of the EAST-ADL language specification.

The metamodel is organized in abstraction levels. On the lowest abstraction level, implementation level, AUTOSAR elements are used to represent the software architecture. The abstraction levels can be seen as a vertical layering of information. A horisontal structure is based on core vs. environment vs. extensions.

Depending on company needs, different strategies to apply the EAST-ADL are foreseen:

(1) As a reference model, where a company will adapt the EAST-ADL information model and implement what they find useful, e.g. through a DSL or custom tooling environment.
(2) A partial deployment of the metamodel where the core parts on one or several abstraction levels are used according to their definition. One or several exensions may be attached to these, according to which information is needed.
(3) EAST-ADL and corresponding exchange format is fully deployed.

In addition to the domain model, the EAST-ADL language is also implemented as a *UML2 profile*. UML profiles are standard extension mechanisms in the UML2 language, in which domain-specific concepts are provided as tags applicable to a selected subset of UML2 elements (such as classes, properties, ports, etc.) giving them different meaning and extra properties. The profile allows users to do system modeling according to the EAST-ADL semantics using off-the-shelf UML2 tools. Constraints are also part of the profile definition; this makes it possible to constrain the rich set of modeling constructs allowed by UML2 and to validate the conformance of the model. The EAST-ADL profile is delivered as an XMI file ready for use in UML2 tools.

In the definition of the EAST-ADL profile, the general strategy has been to provide stereotype properties even for properties already populated within the UML2 superstructure. In other words, the property values that appear when defining a UML2 user model are duplicated with semantic names in the stereotypes. This yields a user model that is quite complete even without a profile.

This approach is in line with the intention of UML2 that views and features of existing UML2 tools can be used readily, including for example, UML2 activity diagrams and related profiles such as SysML (www.omgsysml.org) and MARTE (www.omgmarte.org). The applied profile adds automotive semantics to this self-contained UML2 user model.

4 EAST-ADL Modeling Concepts

The modeling concepts of EAST-ADL fall into various areas, for example functional abstraction, timing modeling, requirements modeling, functional safety modeling, variability modeling and cooperative active safety systems. Below these will be elaborated with descriptions and conceptual figures. Some examples can be found in Section 7.

4.1 Functional Abstraction

EAST-ADL provides the means to capture the functional decomposition and behavior of the embedded system and the environment.

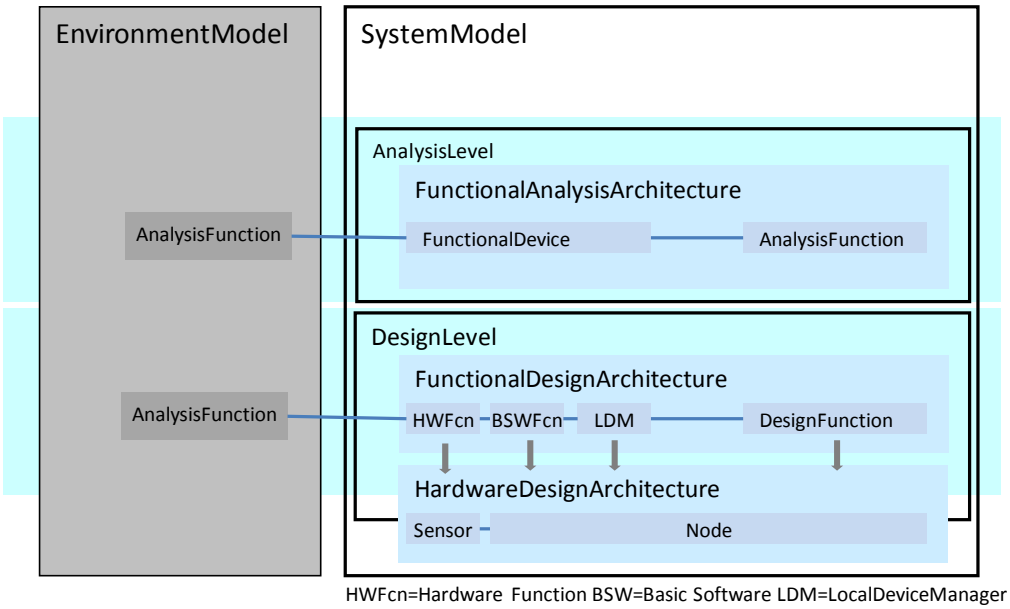

Figure 2:. Functional decomposition in EAST-ADL on analysis and design levels.

At the analysis level, the *Functional Analysis Architecture* contains *Functions* that can be hierarchically composed and connected to each other. Functional devices represent sensors and actuators with their interface software and electronics, and these are connected to the environment. Fig. 2 illustrates the entities involved and shows how they are connected.

The “Functions” can have two types of ports, *FlowPorts* and *ClientServer* ports to represent data exchange and client-server interaction respectively. FlowPorts provide or receive data according to its (single) datatype. ClientServer ports declare multiple operations, each with argument and return value with datatypes. The argument value is provided to the server function on the client call. A return value is received from to the called function on its completion. The data exchange semantics is single buffer overwrite, i.e. the last written value is always used.

The functions can be hierarchical, but the leaves have synchronous execution semantics, which means that they read inputs, calculate and provide outputs. They are triggered based on time or data arrival on ports. For FunctionFlow ports, input and output is provided asynchronously from sender to receiver. Calls to ClientServer ports on event-triggered functions are blocking until the execution time has expired. At this point, the return value is provided. If the server function is time triggered, the delay until triggering has to be added.

In addition to the triggering and execution semantics, the data transformation inside the function needs to be defined. Such transfer function is typically defined by external tools using their

respective notation for behavioral descriptions. A native behavioural notation has also been developed for EAST-ADL allowing direct definition of behavior, which is independent of which tool is used.

The behavior of the environment is captured in the *EnvironmentModel*. The environment model also contains *Functions*, but they represent vehicle dynamics, other vehicles, road-side IT systems, etc.

The design level (see Fig. 2) contains a more detailed functional definition of the system. *Functions* and *LocalDeviceManagers* represent application software in the Functional Design Architecture. "*BasicSoftwareFunctions*" are used to capture middleware behavior affecting application functionality. *HardwareFunctions* represents the logical behavior of hardware components and complete the logical path to the environment model with the controlled "plant" and surrounding elements. The *HardwareDesignArchitecture* represents the resources of the embedded computing platform, i.e. ECUs, communication networks, sensors, actuators and I/O to which the functions are allocated. The Hardware Design Architecture also reflects the physical topology of electrical elements and connectors.

4.2 Timing Modeling

EAST-ADL provides support for model-specific engineering information, including non-functional properties that are relevant for the timing of automotive functions. Conceptually, timing information can be divided into timing requirements and timing properties, where the actual timing properties of a solution must satisfy the specified timing requirements.

Modeling of timing requirements and properties on the functional abstraction levels of the architecture description language is done by means of the *Timing Augmented Description Language*, TADL developed by the TIMMO project [6]. In the implementation level, i.e. AUTOSAR, this is addressed by the *AUTOSAR Timing Extensions* which was introduced in AUTOSAR release 4.0 [2].

Timing constraints are defined separately from the structural modeling and reference the structural elements of the EAST-ADL. The requirements modeling support in EAST-ADL allows for tracing from solutions as modeled in the structural model to requirements, and from verification cases to requirements. The TADL constraints fit in the requirement support as refinements of the requirements.

The fundamental concepts for describing timing constraints are that of *Events* and *Event Chains*. On every level of abstraction, observable events can be identified, e.g. events that cause a reaction, i.e. a stimulus, and resulting observable event, i.e. a response.

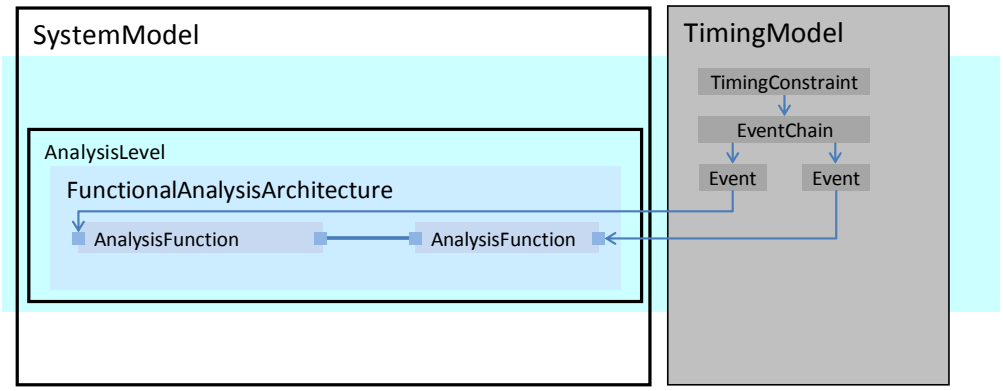

Figure 3. Event Chain with associated timing constraint.

Timing requirements can be imposed on Event Chains, for example, specifying that the time between the occurrence of a stimulus event and the occurrence of the expected response event

shall not exceed a specific amount of time, e.g. an end-to-end delay from a sensor to an actuator. In addition, requirements regarding the synchrony of events can be expressed, stating that a number of events shall occur „simultaneously“ in order to cause a reaction, or be considered as valid response of a system function. For example, in case of a passenger vehicle, its brake system shall apply the brakes simultaneously; or the exterior light system shall simultaneously turn on and off the rear and front turn signal indicators.

Fig. 3 shows a simple example of an event chain with a reaction constraint. The timing elements extend a basic Functional Analysis Architecture. The “in event” refers to the reading of data on the in port and the “out event” to the delivery of data on the out port.

4.3 Requirements Modeling

In order to comprehensively support the development of complex automotive systems, EAST-ADL provides means for requirements specification, i.e. for specifying the required properties of the system (at varying degrees of abstraction). The Requirements concepts are aligned with the SysML and Requirements Interchange Format standards, but adjusted to follow the metamodel structure of EAST-ADL.

The Requirement element is linked to any other EAST-ADL element using a Satisfy relation. Requirements are grouped and structured using the RequirementContainer construct. A Derive relation between requirements support tracing between an original and derived requirement. This way, requirements can be traced across abstraction levels and across hierarchies inside composed elements. Requirements can be formalized using the constraints of EAST-ADL, including timing, safety and behaviour. The Refine relation links the Requirement and the constraint, or other elements used to specify the textual requirement in more detail.

Methodically, EAST-ADL differentiates between *functional requirements*, which typically focus on some part of the "normal" functionality that the system has to provide (e.g. "ABS shall control brake force via wheel slip control"), and *quality requirements*, which typically focus on some non-functional property of the system (e.g. performance, "ABS shall reduce stopping distance on snow by 40%").

EAST-ADL offers detailed means to model artifacts of verification and validation activities and to relate these artifacts to requirements. This facilitates planning and tracking V&V activities and their impact on the system parallel to the system's development.

4.4 Functional Safety Modeling

The overall objective of the support for functional safety modeling is to enforce explicit considerations of safety concerns throughout an architecture design process, including all safety related information that is necessary for developing a safety critical system, in compliance with the Standard ISO 26262 (an international standard dedicated to functional safety for road vehicles, [4]).

As an overall system property, safety is concerned with anomalies (e.g. faults, errors and failures) and their consequences under certain environmental conditions with the goal to mitigate risks. Safety is one particular aspect of system dependability that normally also encompasses reliability, availability, integrity, maintainability and security. Functional safety represents the part of system safety that depends on the correctness of a system in performing its intended functionality. In other words, it addresses the hazardous events of a system during its operation (e.g. component errors and their propagations).

EAST-ADL facilitates safety engineering in terms of safety analysis, specification of safety requirements, and safety design. While promoting safety in general through its intrinsic architecture modeling and traceability support, EAST-ADL provides explicit support for efficient integration of functional safety activities along with the nominal architecture design and evolution.

As illustrated in Fig. 4, EAST-ADL provides language-level support for the concepts defined in ISO 26262, including vehicle-level hazard analysis and risk assessment, the definition of safety goals and safety requirements, the ASIL (Automotive Safety Integrity Level) decomposition and the error propagation. The information is included in the Dependability package, as an extension of the nominal architecture model.

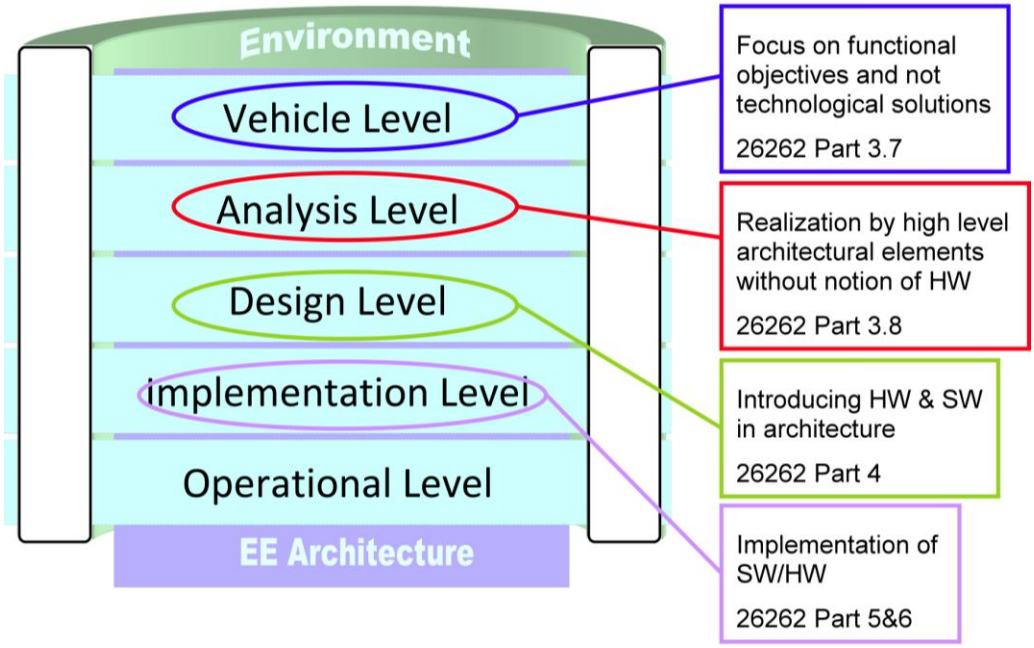

Figure 4. Mapping of ISO26262 information to EAST-ADL abstraction levels.

Following a top-down approach, the safety analysis starts at the VehicleLevel, beginning with the identification and description of the *item*. An item, as defined in ISO 26262, is a system or array of systems or functions that is of particular concern in regards to functional safety. Through hazard analysis and risk assessment activities, it is possible to preliminarily evaluate at VehicleLevel the "safety relevance" of the item under safety analysis, to define the safety goal (top-level safety requirement) for each hazardous event (hazard evaluated in different scenarios) and to classify them in terms of ASIL. Moreover, AnalysisLevel and DesignLevel of EAST-ADL support respectively the *functional safety concept* and the *technical safety concept* definition of ISO26262.

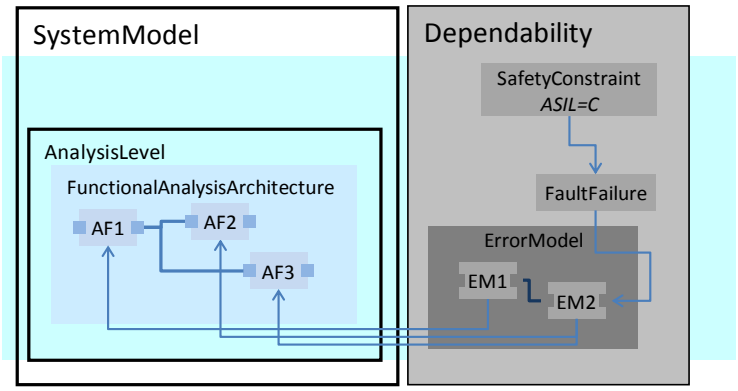

Figure 5. EAST-ADL error model as a separate architecture view extending the nominal architecture model.

EAST-ADL error modeling allows capturing detailed information about the failure behavior of the system and thus supports a safety analysis to determine whether technical safety requirements are being met. This *ErrorModel* describes the generation and propagation of failures through the system. The relationships of local error behaviors are captured by means of explicit error propagation ports and connections. Within an error model, the syntax and/or semantics of existing

external formalisms can be adopted for a precise description of the error logic. The specification captures what output failures of the target architecture component are caused by what faults of this component. This, together with the error propagation links, makes it possible to perform safety simulations and analyses through external analysis tools. In an architecture specification, an error is allowed to propagate via design specific architectural relationships when such relationships also imply behavioral or operational dependencies (e.g. between software and hardware).

The error modeling is treated as a separate analytical view (see Fig. 5). It is not embedded in a nominal architecture model but seamlessly integrated with the architecture model through the EAST-ADL meta-model. This separation of concerns in modeling is considered necessary in order to avoid some undesired effects of error modeling, e.g. relating to the comprehension and management of nominal design, reuse, and system synthesis (e.g. code generation).

Given an error model, the analysis of the causes and consequences of failure behaviors can be automated through tools. There is currently a (prototype) analysis plug-in in the Eclipse environment allowing the integration of the HiP-HOPS tool (Hierarchically Performed Hazard Origin and Propagation Studies) for static safety analysis in terms of FFA, FTA, and FMEA. The analysis leverage includes fault trees from functional failures to software and hardware failures, minimal cut sets, FMEA tables for component errors and their effects on the behaviors and reliability of the entire system.

In EAST-ADL, a safety requirement derived from the safety analysis has attributes specifying the hazard to be mitigated, the safety integrity level (ASIL), operation state, fault time span, emergency operation times, safety state, etc. The safety requirement is then traced to or used to derive other nominal requirements, e.g. relating to safety functions and performance.

4.5 Variability Modeling

EAST-ADL variability management starts on the vehicle level, where model range features and variability are represented. At this point, the purpose of variability management is to provide a highly abstract overview of the variability in the system such as the complete system together with dependencies between these variabilities. A *variability* in this sense is a certain aspect of the complete system that changes from one variant of the complete system to another. “Abstract” here means that, for an individual variability, the idea is not to specify how the system varies with respect to this variability but only that the system shows such variability. For example, the front wiper may or may not have an automatic start. At vehicle level, the impact of this variability on the design is not defined; only the fact that such variability exists is defined by introducing an optional feature named *RainControlledWiping*. This is subsequently validated and refined during analysis and design.

One or more feature models may be defined on the vehicle level: the so-called core Technical Feature Model is used to define the complete system’s variability on a global level from a technical perspective, whereas one or more optional Product Feature Models can be used to define views on this technical variability which can be tailored to a particular view-point or purpose, e.g. the end-customer perspective.

While the details of how variability is actually realized in the system are largely suppressed at the vehicle level, they are the focus of attention when managing variability in other areas of the development process. In fact, specific variability may lead to modifications in any development artifact, such as requirements specifications and functional models. Here, describing that a specific variability occurs is not sufficient; it is necessary to describe how each variation affects and modifies the corresponding artifact.

The purpose of feature modeling is to define the commonalities and variabilities of the product variants within the scope of a product line. Feature models are normally used on a high level of abstraction, as described above for vehicle level variability. However, in EAST-ADL, they are also used on analysis and design levels and acquire a much more concrete meaning there. Configuration decision modeling, on the other hand, is aimed at defining configuration: the

configuration of a feature model, f_T – i.e. the selection and deselection of its features – is defined in terms of the configuration of another feature model, f_S. A configuration decision model can thus be seen as a link from f_S to f_T that allows us to derive a configuration of f_T from any given configuration of f_S. In EAST-ADL, this mechanism is used to define how a certain configuration on a higher abstraction level affects the binding of variability in lower-level components.

Variability management on the artifact level is driven by the variability captured on the vehicle level. This means that the main driver for variability and also variability instantiation is the vehicle-level feature model. Variability on the artifact level essentially consists of the definition of variation points within these artifacts. In addition, feature models can be attached to functions in order to expose the variability within these functions and hide the actual structuring, representation and binding of this variability within a function. This way, the benefits of information hiding can now be applied to the variability representation and variability binding within the containment hierarchy of functions in the EAST-ADL Functional Analysis Architecture and Functional Design Architecture (called compositional variability management).

4.6 Behavior Constraint Modeling

The reasoning and analysis of dependability and performance involve many aspects in a system's lifecycle. To this end, EAST-ADL allows precise and integrated annotations of various behavioral concerns related to requirements, application modes and functions, implementation and resource deployment, and anomalies. The approach is architecture centric as all behavior annotations are formally connected to a set of standardized system artefacts and lifecycle phases. This is fundamental for many overall design decisions, such as requirements engineering, component compositionality and composability, design refinements, safety engineering, and maintenance. From a wider perspective, this language support enables an integration of many existing modelling and analysis technologies, such as from computer science and electronic engineering, by making it possible to trace and maintain the related engineering concerns and analytical information coherently using EAST-ADL.

Based on a hybrid-system semantics, the EAST-ADL support for the annotations of behavioural concerns consists of three categories of behavior constraints:

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

Each of these behaviour constraints can be associated to time conditions given in terms of logical time, of which the exact semantics is given by the existing EAST-ADL support for timing definition (e.g. the triggering, and port data sending and receiving events of a function). Owing to the formal semantics, one can explicitly define the model transformation from EAST-ADL behavior model to other model formats of external analysis methods and tools, such as hazard analysis, response time analysis, model checking, test-case generation, etc.

5 Methodology

The purpose of the EAST-ADL methodology is to give guidance for the adoption of EAST-ADL, i.e. how to use of the language for the construction, validation and reuse of models for automotive embedded software. The purpose is thus not to impose a specific development process, but to show sequences of steps that can produce sound and useful EAST-ADL models. The EAST-ADL methodology provides understanding for the EAST-ADL language and serves as building blocks for a more complex and complete process definition.

The EAST-ADL methodology is defined as a core part which is complemented by extensions. The core is a top-down description of the most central steps in each phase:

- The Vehicle phase involves analysis of external requirements based on which a Technical Feature Model is constructed. This tree structure shall be organized in an adequate way and also capture necessary or intended feature configurations. In addition, for each feature a set of requirements is specified.
- The Analysis phase results in a FunctionalAnalysisArchitecture which specifies a realization of the Features. The solution is a logical representation of the system to be developed and there is no distinction between hardware or software or about the implementation of communication.
- The Design phase involves defining the FunctionalDesignArchitecture specifying a solution to the requirements in terms of efficient and reusable architectures, i.e. sets of (structured) HW/ SW components and their interfaces, a hardware architecture, and a mapping from functional components to HW/SW components. The architecture must satisfy more detailed constraints.
- The Implementation phase results in the HW/SW implementation and configuration of the final solution. This part is mainly a reference to the concepts of AUTOSAR, which provides standardized specifications at this level of automotive software development.

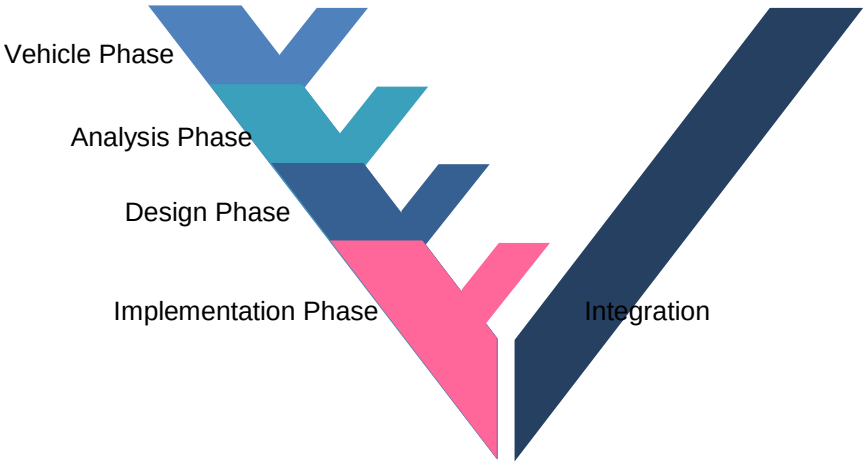

Figure 6. The typical basic structure of automotive system development according to the V-model.

The core methodology is extended into a comprehensive methodology for automotive development projects by adding additional and orthogonal activities to each of these phases:

- Specification of Requirements and corresponding V&V cases to be executed and evaluated during the corresponding integration phase. V&V cases are most typically test cases, but can also include reviews etc.
- Verification of the model on a given abstraction level to the requirements of the model at the abstraction level directly above.
- V&V activities on the model artefacts of a given level itself, i.e. peer reviews, consistency checks, check of modeling guidelines etc.

While the methodology tries to be comprehensive in handling the construction phases, the integration activities are only covered inasmuch they involve V&V activities and the relation to V&V-artifacts defined in the construction phases.

The EAST-ADL methodology is extended beyond the core activities by means of a set of methodology extensions. In the first approach for EAST-ADL methodology, different methodology versions were defined depending on scope. Examples of scope were:

Environment Modeling: modeling of the (typically analog or discrete-analog) environment of the system to be developed.
Safety Assurance: development of Safety-critical systems.
Timing: detailed handling of timing requirements and properties.
Variability Modeling: detailed handling of variability modeling.
Behavior modeling: detailed handling of behavioral modeling.

The initial EAST-ADL methodology definition is using concepts of the Software & Systems Process Engineering Meta-model SPEM (www.omg.org/spec/SPEM/), which means that the methodology is based on a set of elementary work tasks which are performed by a set of actors and produce a set output artifacts from a set of input artifacts. These tasks are structured into disciplines and then presented to the end user by a set of views. This leads to a highly linked network of methodological activities in which an end user can easily navigate to get information and guidance on the use of the language for particular development tasks.

The methodology definition is currently being restructured to follow a Generic Methodology Pattern, where the EAST-ADL phases are divided into a set of steps. The set of steps are the same for each phase, and also for each aspect (safety, timing, variability, etc.). The principle is that the user assesses the steps related to the core and each relevant aspect and then implicitly “weaves” an appropriate set of steps for his needs. The notation for this more recent methodology is Business Process Modelling Notation.

6 Related Approaches

One key aspect of the development of EAST-ADL is to benefit from existing methods and techniques and also to influence emerging approaches. Whenever possible, existing and state-of-the-art solutions were reused and integrated in the language. This favors the wide use of the language, allows the use of available tools and prepares for a sound standardization process.

Efforts like AUTOSAR [2], TIMMO [6], and ISO 26262 [4] are sources both for the alignment of domain specific challenges and for the integration of technologies and methodologies in the development of EAST-ADL.

As a future de-facto standard for automotive embedded systems, AUTOSAR addresses the needs for a process-safe integration of functions. It provides a standardized platform for the specification and execution of application software, an integration method for software components and hardware resources, and also the interchange formats that these require. While adopting AUTOSAR for the implementation level abstractions, the EAST-ADL language complements the AUTOSAR initiative by providing higher-level abstractions, analysis and lifecycle management support. In effect, it allows an AUTOSAR-compliant software architecture being extended with models relating to the design of functionality, timing and safety, the structuring and allocation of application, as well as the management of variability, requirements, traceability and verification and validation.

AUTOSAR is used to represent the final software architecture of automotive embedded systems. As such, it defines the software components, their interfaces, execution timing, middleware (basic software) interactions, etc. The model is sufficiently detailed to automatically generate and configure the platform software and integrate on ECUs.

The EAST-ADL function design architecture act as the functional specification of the AUTOSAR software architecture. As such, it carries functional and non-functional requirements stemming from user needs, control design, safety design, re-use, etc. Software components may have to be organized differently, have different interfaces, timing, interaction, etc. in order to meet the architectural constraints of the software architecture. Using an EAST-ADL functional model, such implementation specific adjustments can be made in the AUTOSAR model without replacing the fundamental properties.

Defining a software architecture requires a large effort with much detail in the solution. The EAST-ADL functional model is a way to quicker make architectural exploration. The design level models can then be used to autogenerate much of the content on implementation level.

EAST-ADL integrates the results of TIMMO, which is an ITEA project focusing on the timing constraints and timing properties in automotive real-time systems. TIMMO has developed a formal description language, TADL, and a methodology for dealing with the timing concerns on the basis of EAST-ADL1. It has been developed in a close collaboration with AUTOSAR. The follow-up project TIMMO-2-USE is further developing the TADL language, in close collaboration with the MAENAD project, developing EAST-ADL.

The emerging international standard ISO 26262 [4] is carefully considered in EAST-ADL. The key content includes an automotive safety lifecycle, an automotive specific approach for determining risk classes and deriving safety requirements based on ASILs (Automotive Safety Integrity Levels), and a set of requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety being achieved.

To support behavior modeling, EAST-ADL provides dedicated behavior elements that facilitate the description of the relationship between behavioral and structural models. The EAST-ADL functions have synchronous execution semantics, and language concepts are available to define their triggering and timing. By clearly distinguishing between component execution and component logical computation, EAST-ADL allows the integration of behavior models from off-the-shelf tools like SCADE, ASCET, Simulink, etc., according to lifecycle stages and stakeholder needs. For continuous-time behavior (e.g., for the vehicle dynamics under control), related modeling

techniques from Modelica, which combines acausal modeling with object-oriented thinking, have been adopted. The Functional Mockup interface, used for co-simulation and model exchange via Functional Mockup Units (FMUs) has been investigated, and a prototype transformation tool has been developed. EAST-ADL also provides tool prototypes for model transformation to Simulink and the SPIN (Simple PROMELA Interpreter) model checker.

A further standardization effort being taken into consideration is the SAE “Architecture and Analysis Description Language” (AADL), which has its roots in the avionics domain. Compared to EAST-ADL, AADL has a more narrow scope: no explicit support is provided for variability management or requirements refinements and traceability. Specifics for automotive systems such as the networks are weakly supported. The AADL is not designed for mass-produced systems and therefore has less emphasis on optimized overall solutions e.g. by considering compact runtime systems. For the automotive domain, the clash with AUTOSAR concepts is also a problem. However, wherever applicable, AADL concepts were reused, e.g. for dependability modeling.

EAST-ADL allows the adoptions of existing formalisms for the underlying semantics and provides support for model transformation and tool interoperability with the external safety analysis techniques. In particular, HiP-HOPS and the AADL’s Error Model Annex have been carefully considered in the development of EAST-ADL. They both enable the modeling of system failure behavior and allow analysis of that behavior using tools.

A tool plug-in for HiP-HOPS has been developed to support both FTA and FMEA. Other approaches to model-based safety analysis and verification that have been investigated for the development of EAST-ADL include ISSAC and its predecessor ESACS in the aerospace industries (where the goal was to develop a formal methodology and tools for the safety analysis of complex aeronautical systems), the ASSERT project (with similar goals but more focused on software intensive systems specified in AADL), the SETTA project (focusing on the use of time-triggered architectures in automotive systems), and the SAFEDOR project (which aimed to develop new practices for the safety assessment of maritime systems).

SPEEDS (Speculative and Exploratory Design in Systems Engineering) is a European project aiming at providing support for modeling and analysis of complex embedded systems through the usage of formal analysis tools. EAST-ADL complements the SPEEDS approach with automotive architecture and lifecycle information. The techniques of SPEEDS have been considered in EAST-ADL for behavior modeling (i.e., with the hybrid automata variant) and for a more formal specification of requirements and constraints (i.e., with temporal logics scripts for contracts of functionality, safety, and timing).

MARTE is a UML profile for Modeling and Analysis of Real-Time and Embedded systems. MARTE models real-time constraints and other embedded systems characteristics, such as memory capacity and power consumption. MARTE supports modeling and analysis of component-based architectures, as well as a variety of different computational paradigms (asynchronous, synchronous, and timed). The EAST-ADL UML-profile is released as an annex to MARTE, done in the ATESST2 and ADAMS projects.

The OMG Systems Modeling Language (OMG SysML) is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. Compared with EAST-ADL, SysML is more generic and high-level, so EAST-ADL can be seen as a specialization and subset for automotive embedded systems. In fact, the first versions of EAST-ADL and SysML were defined in parallel with some interaction between the teams. The EAST-ADL function architecture with ports and datatypes are influenced by SysML, as well as the requirements modelling.

7 Example Model

On the following pages, various diagrams from an EAST-ADL model will be shown. The model and diagrams are created using the Papyrus UML tool.

7.1 Overall Model

Figure 7 provides a package structure overview of the expected EAST-ADL modelling elements for the braking system architecture, as well as its associated requirements, variability and other non-functional constraints (e.g., timing and dependability), and verification&validation (V&V) cases. The *SystemModel* (within the *0_TopPackage*) contains the entire braking electrical/electronic system architecture, for which specifications at various abstraction levels are applied. Figure 8 provides a graphical representation of this multi-level braking electrical/electronic system specification and its related environment model (*EnvironmentBBW*).

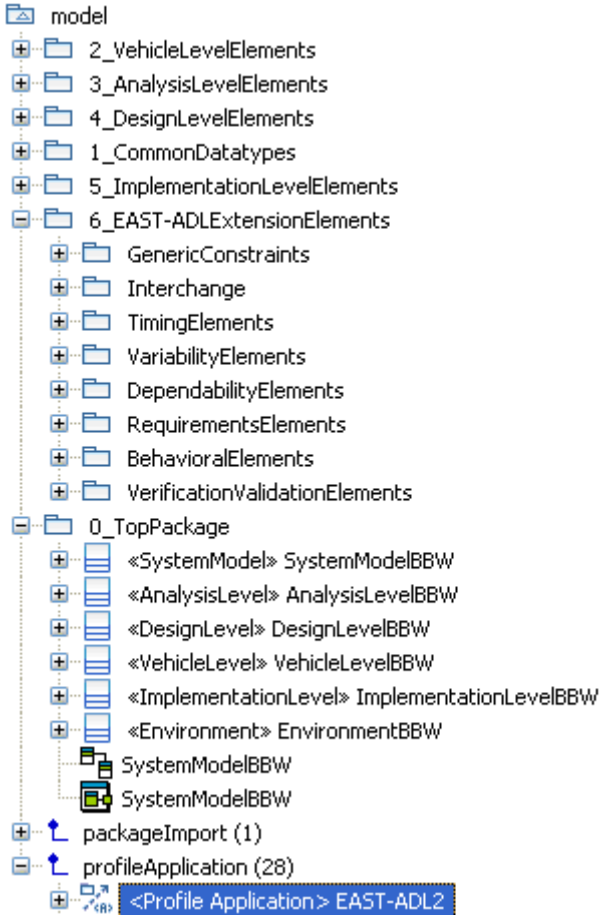

Figure 7: An overview of packages of an EAST-ADL model in Papyrus.

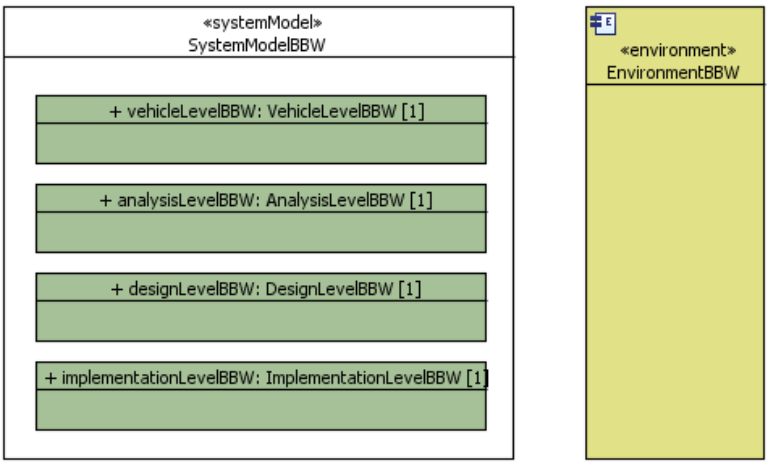

Figure 8: The braking electrical/electronic system and its environment in Papyrus.

EAST-ADL supports requirements, V&V cases, and the annotations of variability and other non-functional constraints through separate modeling packages shown in Figure 9. A requirement model specifies the conditions or capabilities that must be met or possessed by a system or its component. In a model-based approach, requirements are derived, refined, mapped, validated and verified along with the progress of system design. The specifications of variability and other non-functional constraints augment the multi-level system architecture specification with analytical information (e.g. timing, reliability, and safety integrity) for early quality predictions and contract declarations. Normally, an analytical model should have its level of abstraction according to its target artefacts.

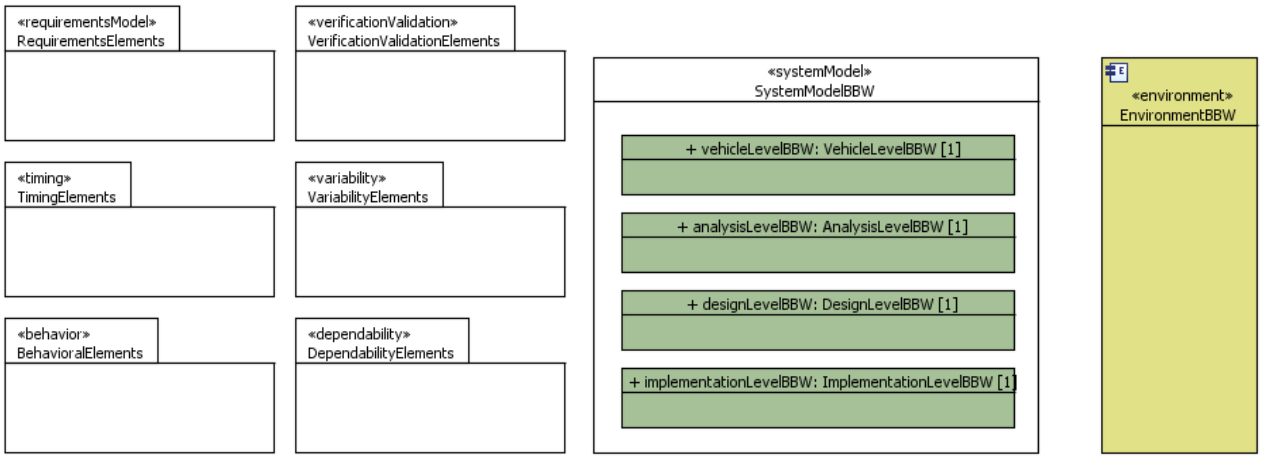

Figure 9: An overview of system model and related EAST-ADL packages for the specifications of requirements, V&V cases, and the annotations of variability and other non-functional constraints in Papyrus.

7.2 Vehicle level

A vehicle level architecture specification constitutes the topmost system description and manages the features of an entire product family. In Figure , the feature tree of the target braking system is shown. Each vehicle feature (*VehicleFeature*) denotes a functional characteristic, such as the functions, or non-functional properties, to be supported. While a braking control feature (*BrakingControl*) is needed for the vehicle longitudinal control, regenerative braking control

(*RegenerativeBrakingControl*) is a feature for power control in FEV, allowing the kinetic energy produced by braking to be converted to electrical energy and stored in capacitor or/and battery. As shown in Figure , the relations of features are supported by feature links (*FeatureLink*). In a feature link definition, the precise semantics of a feature relationship is given by the type attribute (*Kind*) and the direction attribute (*isBidirectional*).

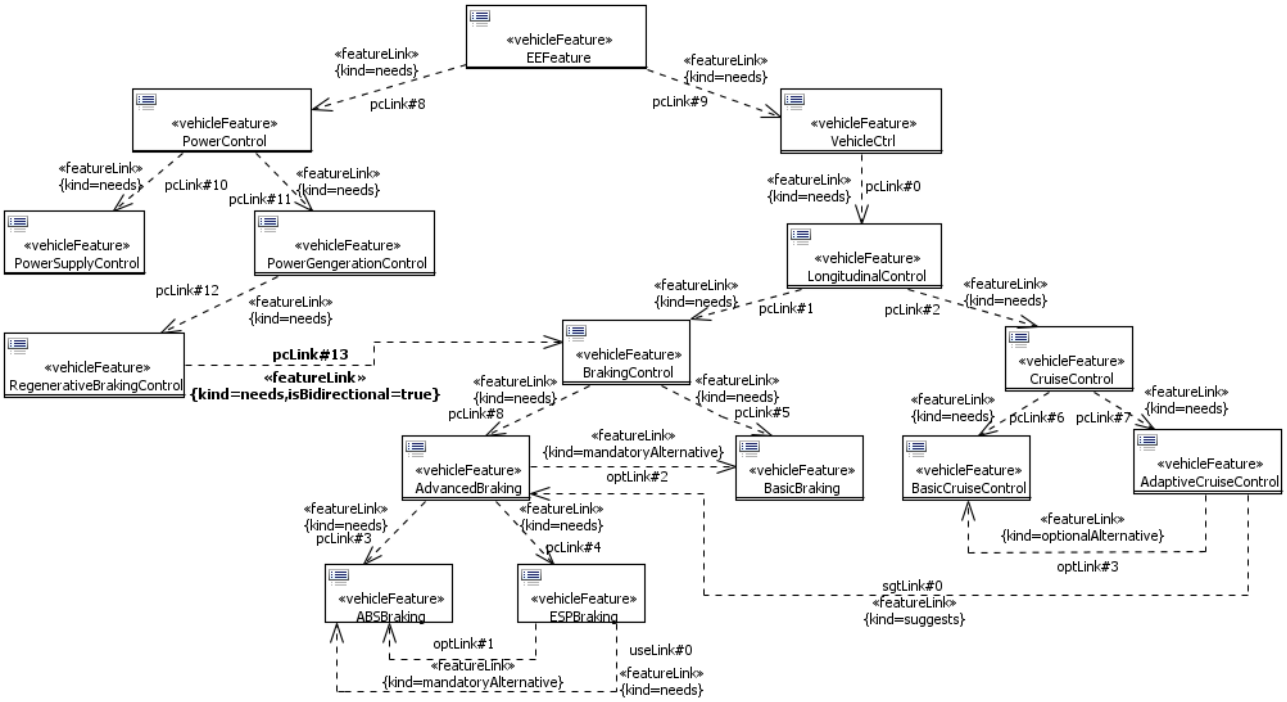

Figure 10: Vehicle Feature Model of the Regenerative Braking System in Papyrus.

Requirements at the vehicle level are directly based on system use cases and allocated to vehicle features denoting the expected system functions). See Table 1 for a list of requirements on braking control. By EAST-ADL, the relationships of a requirement in regard to other requirements, system artefacts, more detailed analytical models, and V&V cases are explicit supported.

Table 1: Top-level braking control requirements.

ID	Description
Req#1_BaseBraking	"The system shall provide a base brake functionality where the driver indicates that he/she wants to reduce speed and the braking system starts decelerating the vehicle"
Req#2_DriverBrakeRequest	"The driver shall be able to request braking"
Req#3_Anti-LockBraking	"The system shall be an anti-lock braking system (ABS) by preventing the wheels from locking while braking"
Req#4_BrakeReactionTime	"The time from the driver's brake request until the actual start of the deceleration shall be ≤ 300ms.(Value derived from expert judgment)"
Req#5_TimeToStandstill	"The time to stadstill shall follow the recommendations in EU braking systems Directive 71/320 EEC. The Swdish Road Administration claims that a factor of 3 (on braking distance) is acceptable for ice"
Req#6_OperationofBrakePedal	"The Operator shall be able to vary the desired braking force using the brake pedal. A fully pressed pedal means maximum brake force."
Req#7_BrakeRelease	"When the brake pedal is not pressed, the brake shall not be active."

While a feature tree model specifies composition of system functions and their logical dependencies, it often implies the refinement of vehicle level requirements. With EAST-ADL, the derived/derived by relationship of requirements is given by a dedicated requirement relationship: *DeriveRequirement*. When such a requirement relationship is declared, a modification of the

supplier requirement would have effects on the derived client requirements. Figure 11 shows the requirements model capturing four derived requirements and their relationships to a common supplier requirement and to each other.

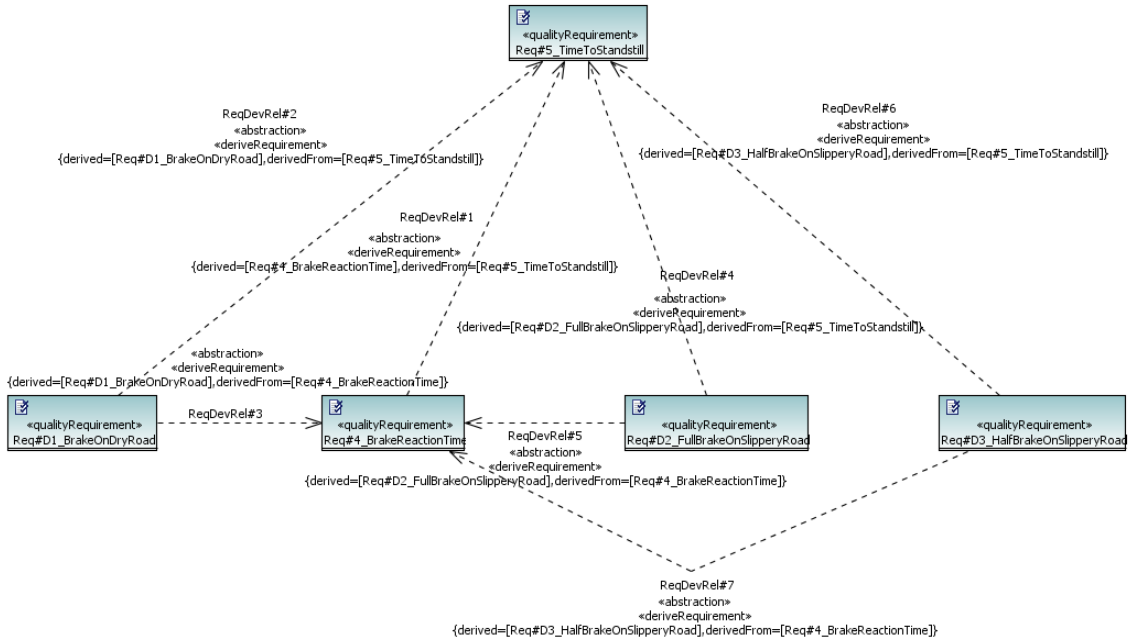

Figure 11: A model of braking performance requirements in Papyrus.

Figure 12 shows the allocations of functional and non-functional requirements to the braking control and its sub-features through the *Satisfy* links.

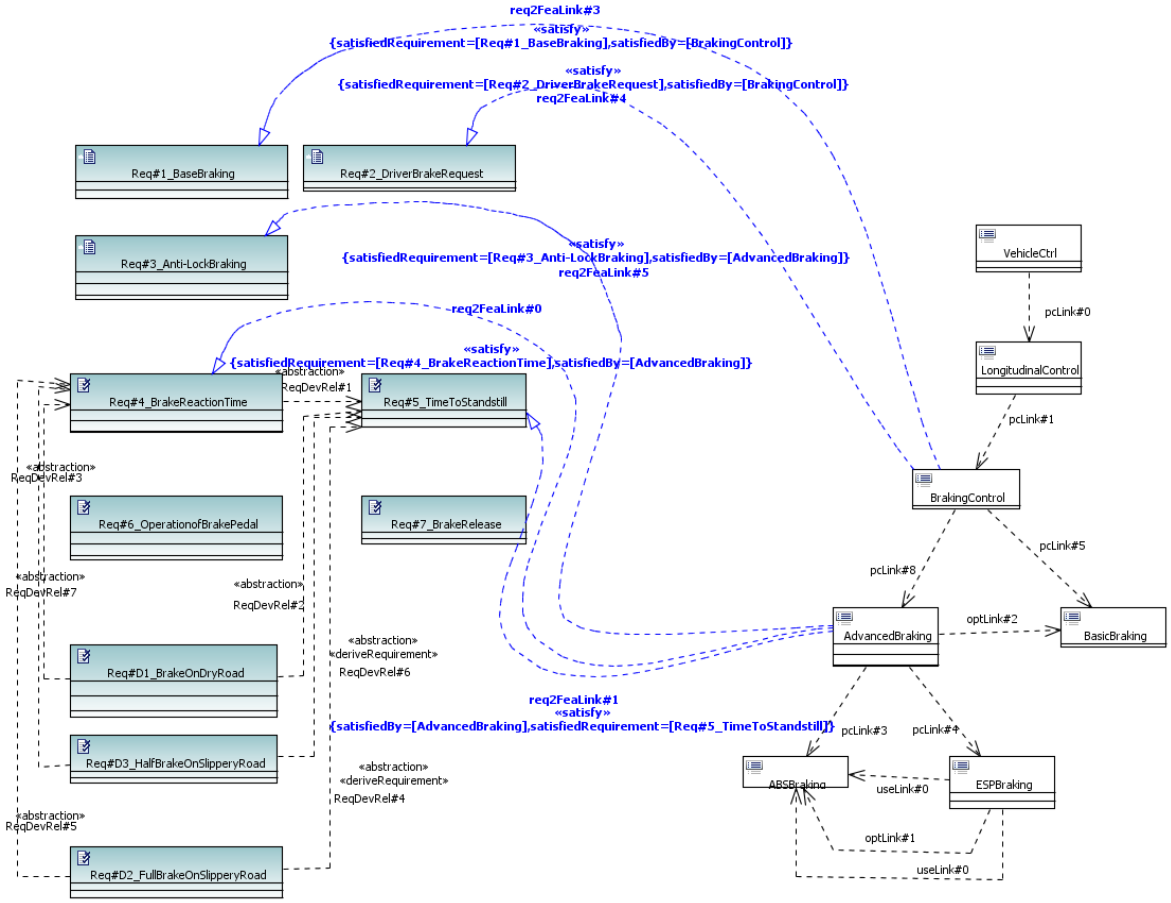

Figure 12: Allocations of braking requirements on vehicle features in Papyrus.

In EAST-ADL, a satisfy relationship signifies the relationship between a requirement and an architectural element intending to satisfy the requirement. Requirements can also be inherited along with the feature configuration hierarchy. For example, the requirements *Req#1_BaseBraking* and *Req#2_DriverBrakeRequest*, shown in Figure 12, should also be satisfied by the children of *BrakingControl*, such as the *AdvancedBraking* and the *BasicBraking.*

7.3 Analysis Level

As a step towards system realization, the vehicle level features are realised by some interconnected abstract functions at the analysis level, specifying the corresponding input functions, application functions, and output functions for each vehicle level function in an implementation independent way. For the target braking system, the vehicle features of concern are implemented by a set of analysis functions shown in Figure 13 and Figure 14.

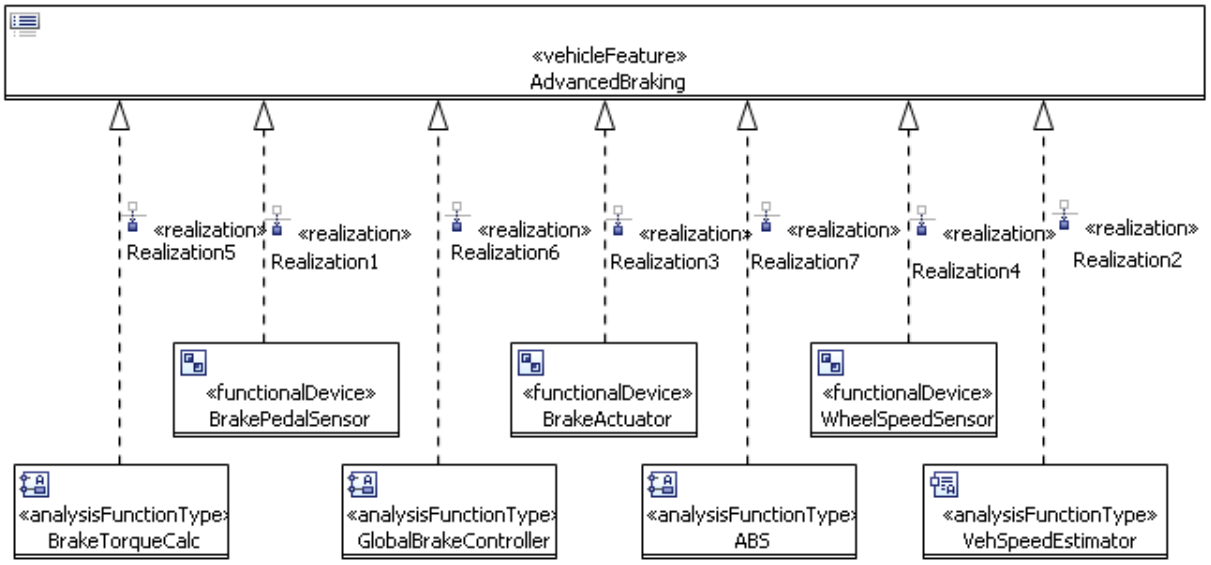

Figure 13: Advanced Braking feature and the specification of its functional realizations in Papyrus.

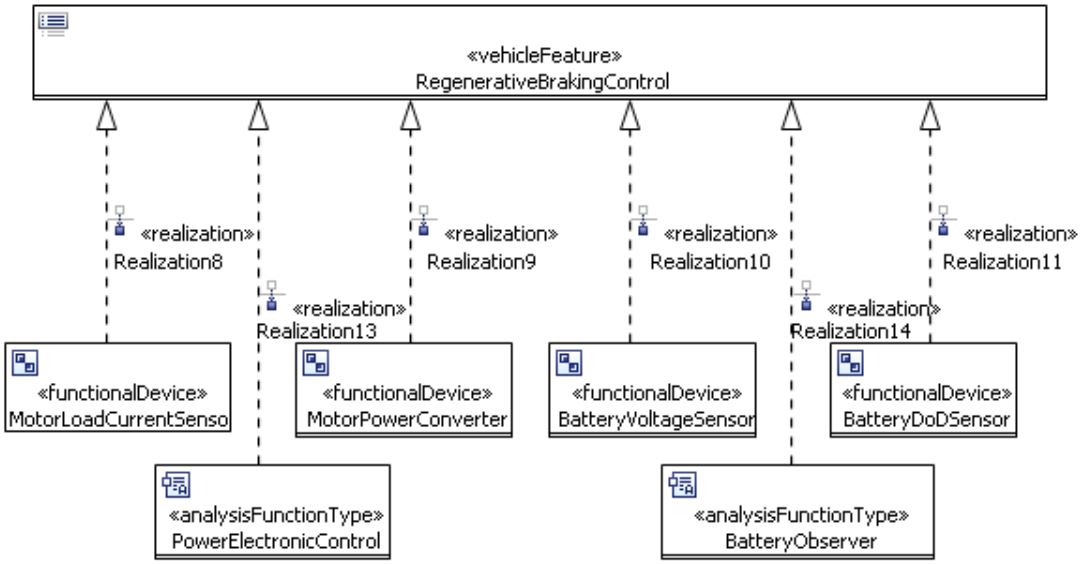

Figure 14:Regenerative Braking Control feature and the specification of its functional realizations in Papyrus.

Figure 15 shows the specification of functional architecture in EAST-ADL for the braking system (See also D6.1.1 for an overview the functional operation concept).

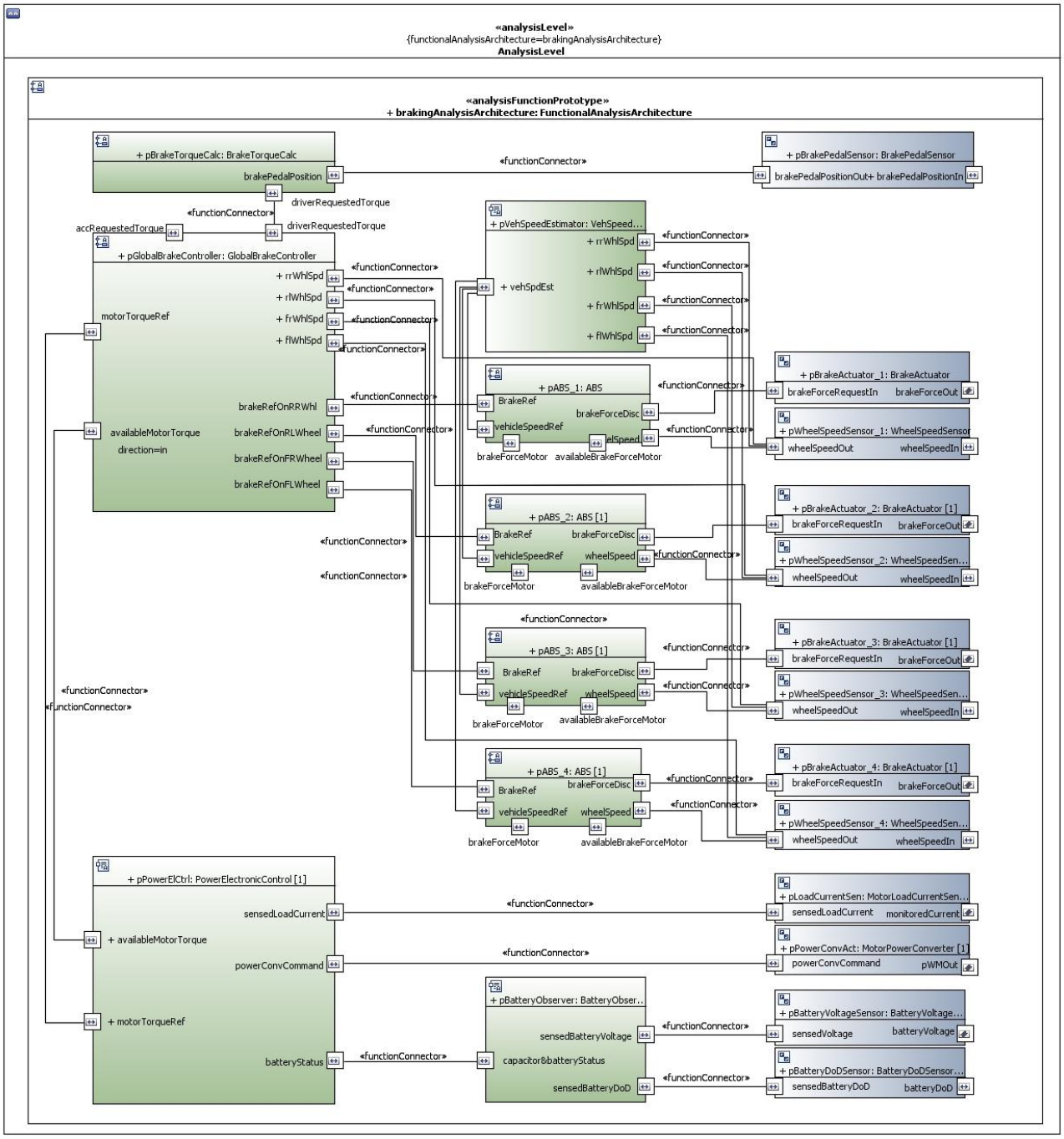

Figure 15: Functional Analysis Architecture specification of the Regenerative Braking System in Papyrus.

In EAST-ADL, system boundaries are explicitly defined by means of functional devices (*FunctionalDevice*). Through functional devices, an analysis function interacts with the physical environment. Figure 16 shows the connections between functional devices and the physical environment.

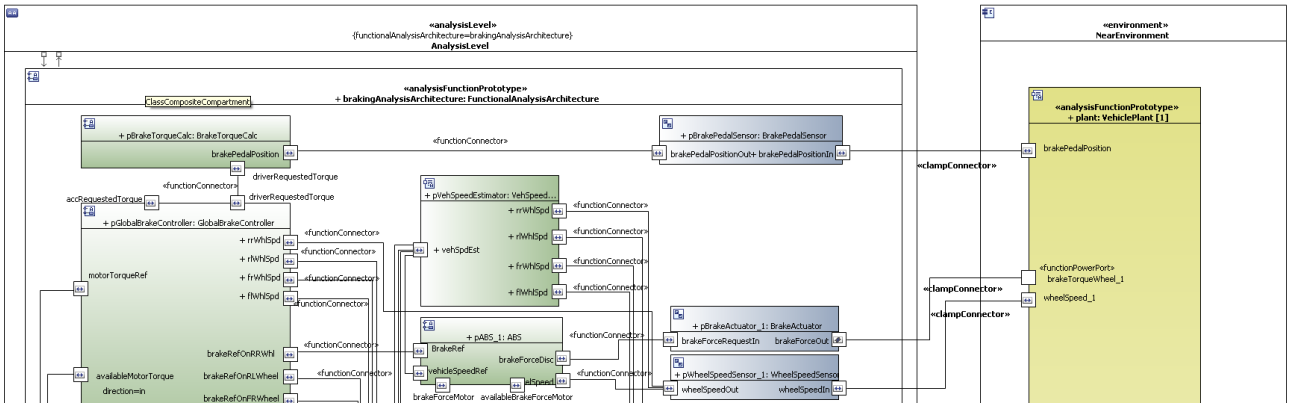

Figure 16: Connecting functional analysis functions with environment in Papyrus.

To define the timing requirements and timing design, constructs like TimingConctraint, EventChain and Event are available in EAST-ADL.

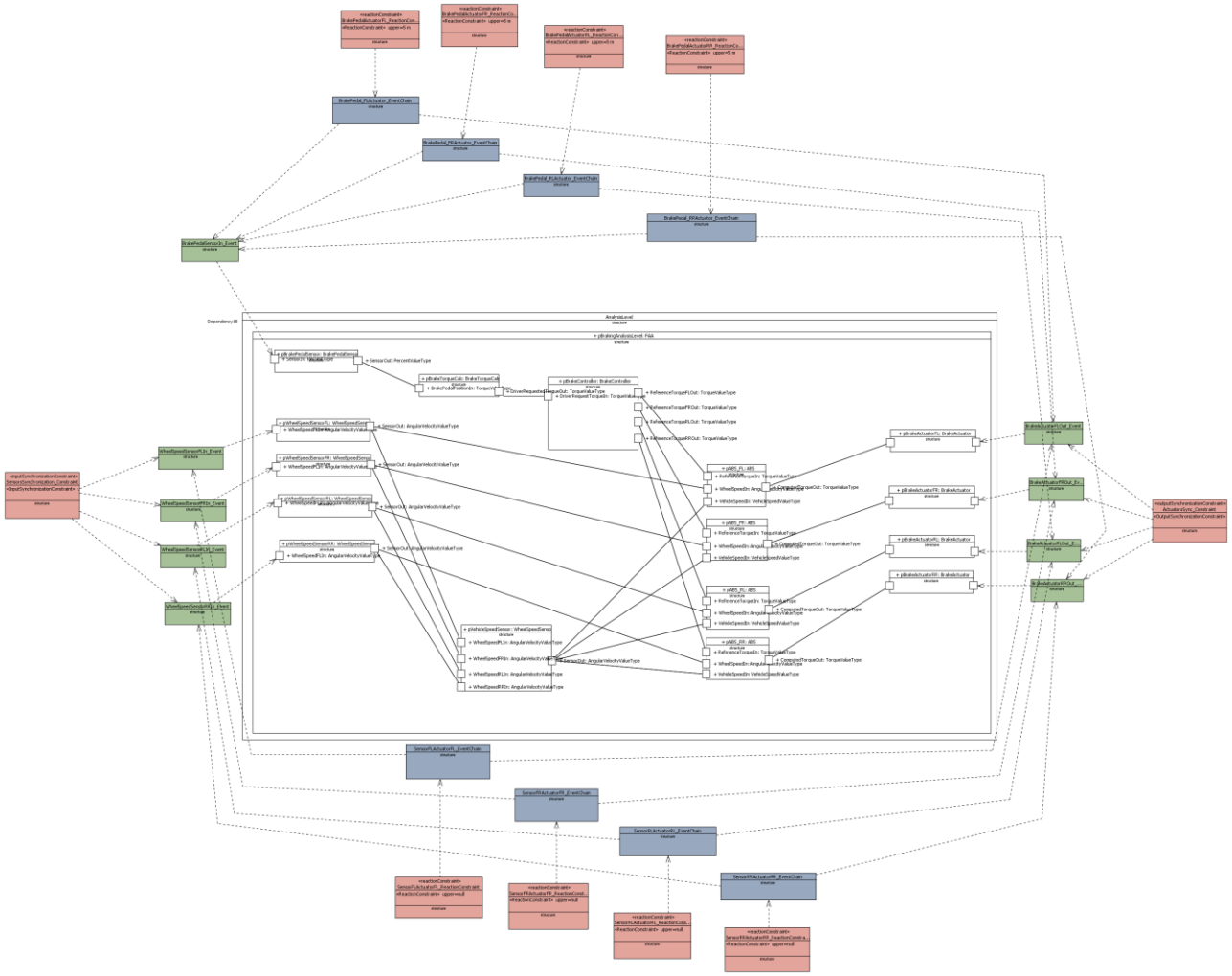

Figure 17. Synchronization and End-to-end timing from pedal to brake actuators

7.4 Design Level

The design level architecture further details the analysis level design by taking the software and hardware resources into consideration. (See also D6.1.1 for an overview the related design concept).

Currently, the documentation correspond to a single wheel brake by wire model. Work is under way to extend to a full four-wheel model.

7.4.1 Functional Design Architecture

Figure 18 shows the FunctionalDesignArchitecture. The model is focusing on base braking and does not include energy regeneration functionality.

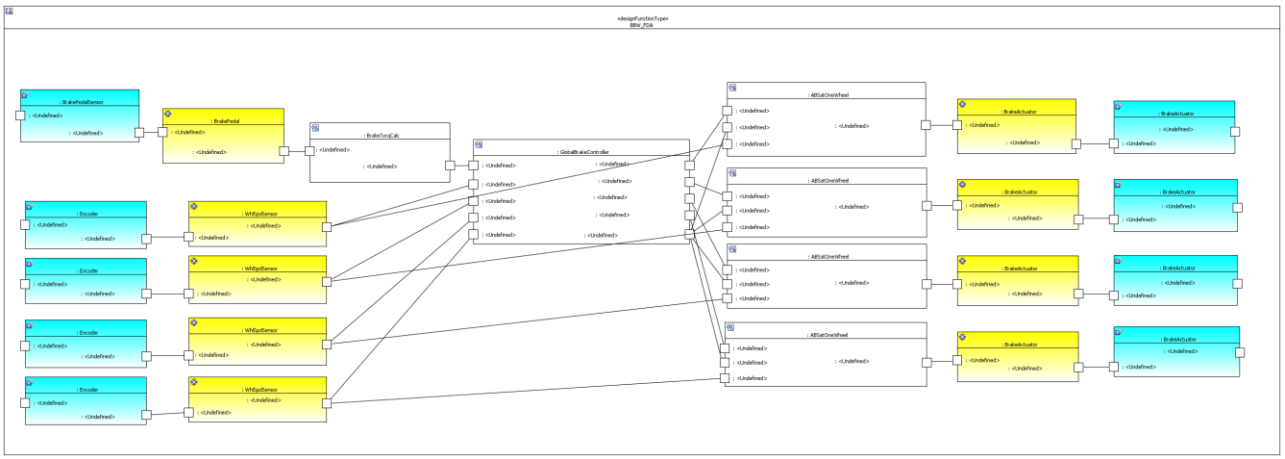

Figure 18. Functional Design Architecture of the Regenerative Braking System in Papyrus.

Figure 19 shows the period times of the included functions.

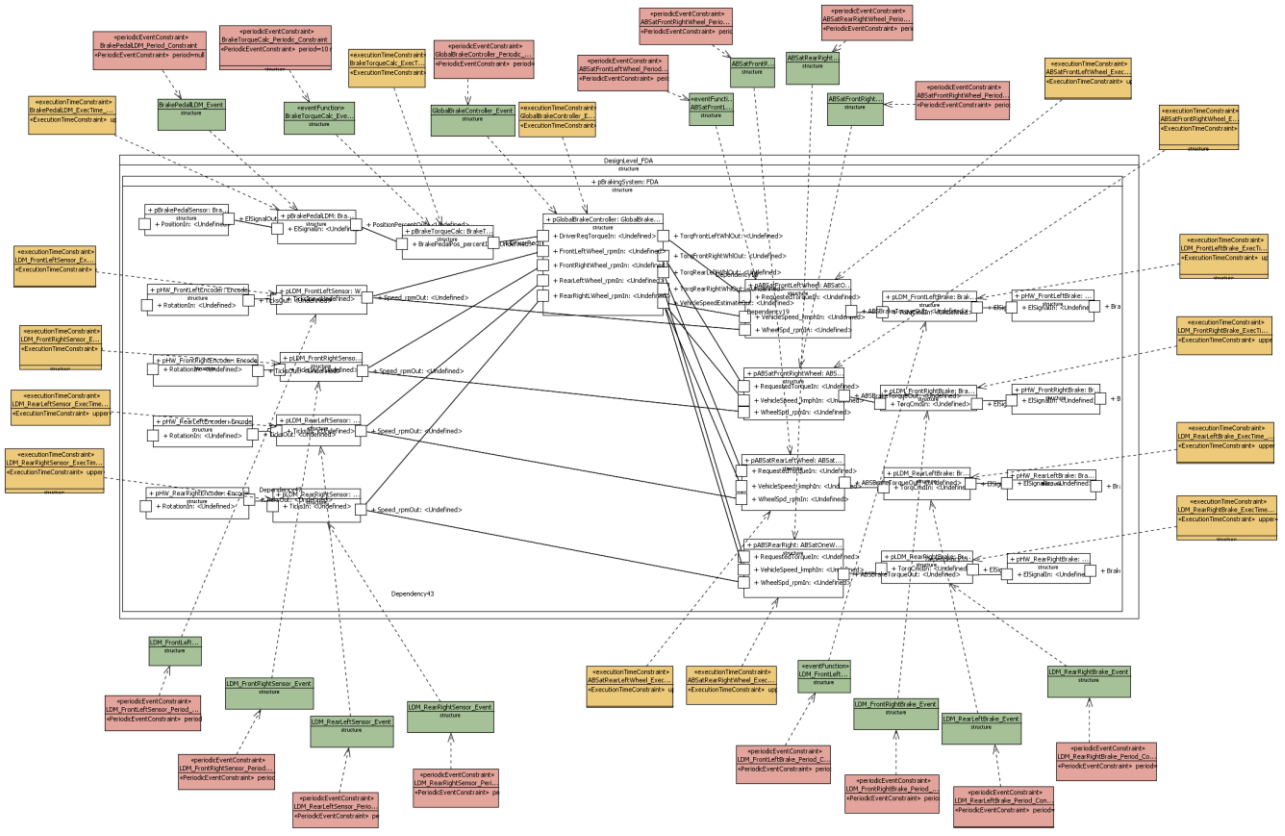

Figure 19. Period times of functions

Figure 20 (close-up) and Figure 21 (overall) shows timing constraints for end-to-end response requirements of the brake functionality. Figure 21 also show synchronization requirements and a brake-down of the end-to-end timing budget.

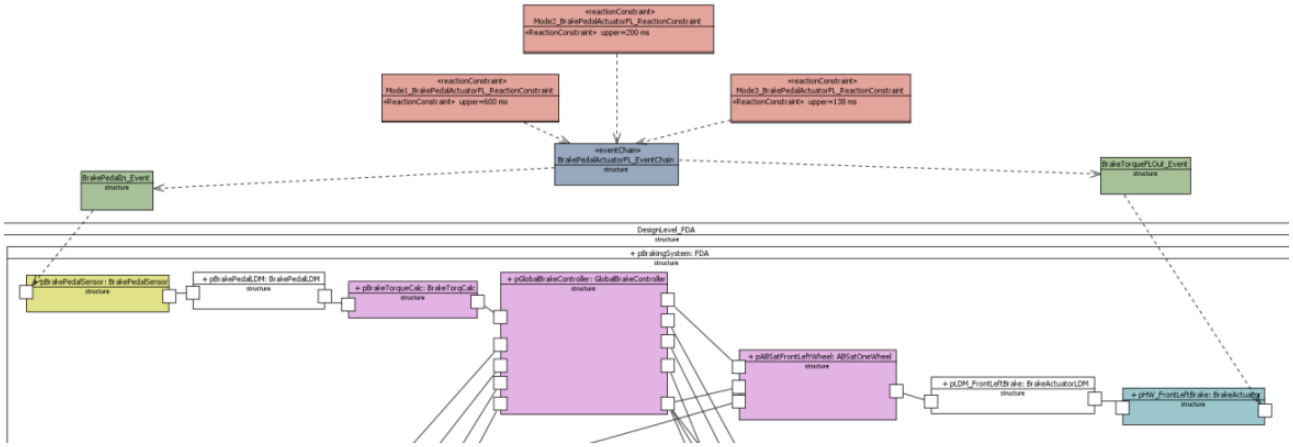

Figure 20. Functional Design Architecture with end-to-end timing

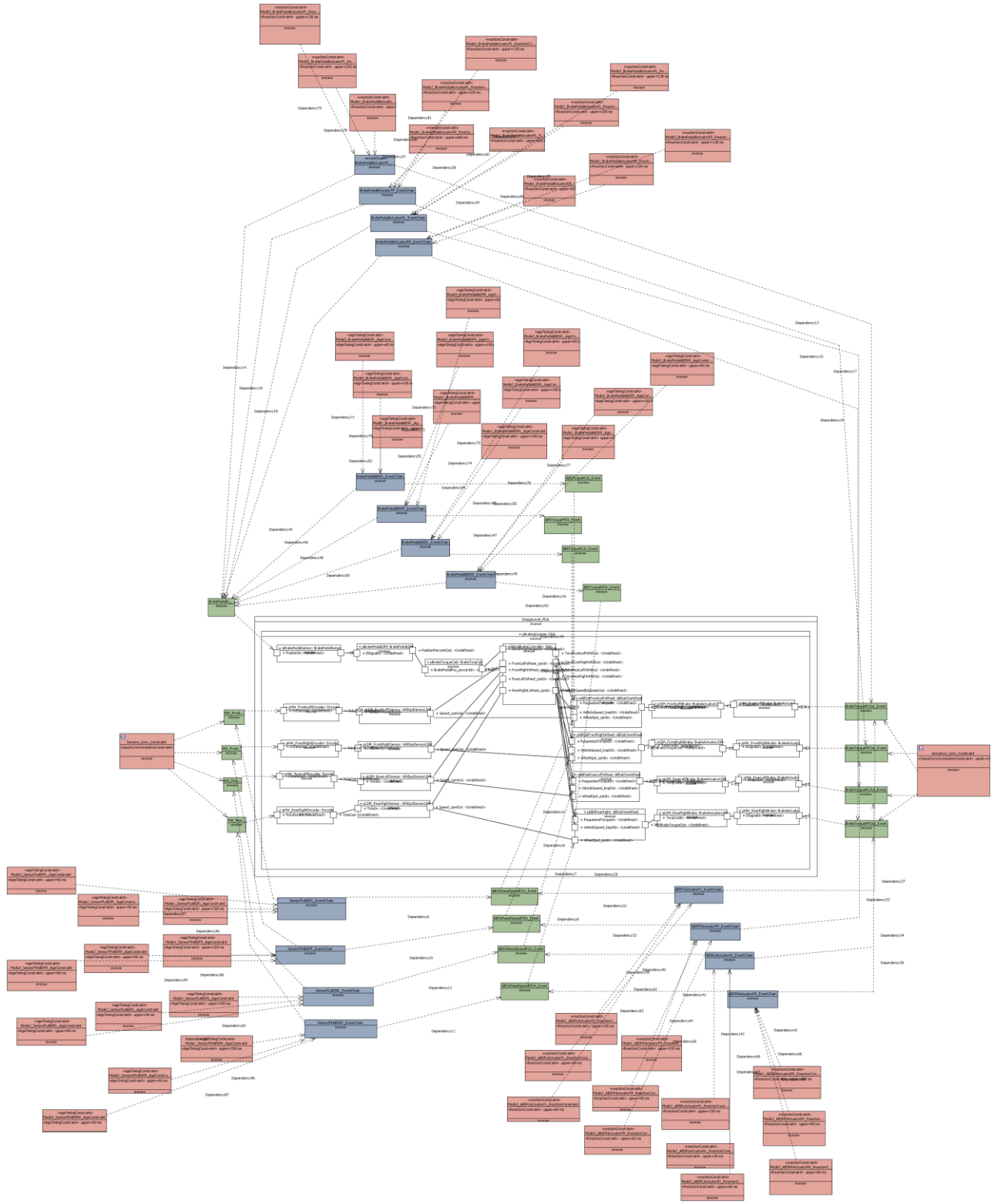

Figure 21. Functional Design Architecture with end-to-end timing

7.4.2 Hardware Design Architecture

Figure 22 shows an initial HardwareDesignArchitecture.

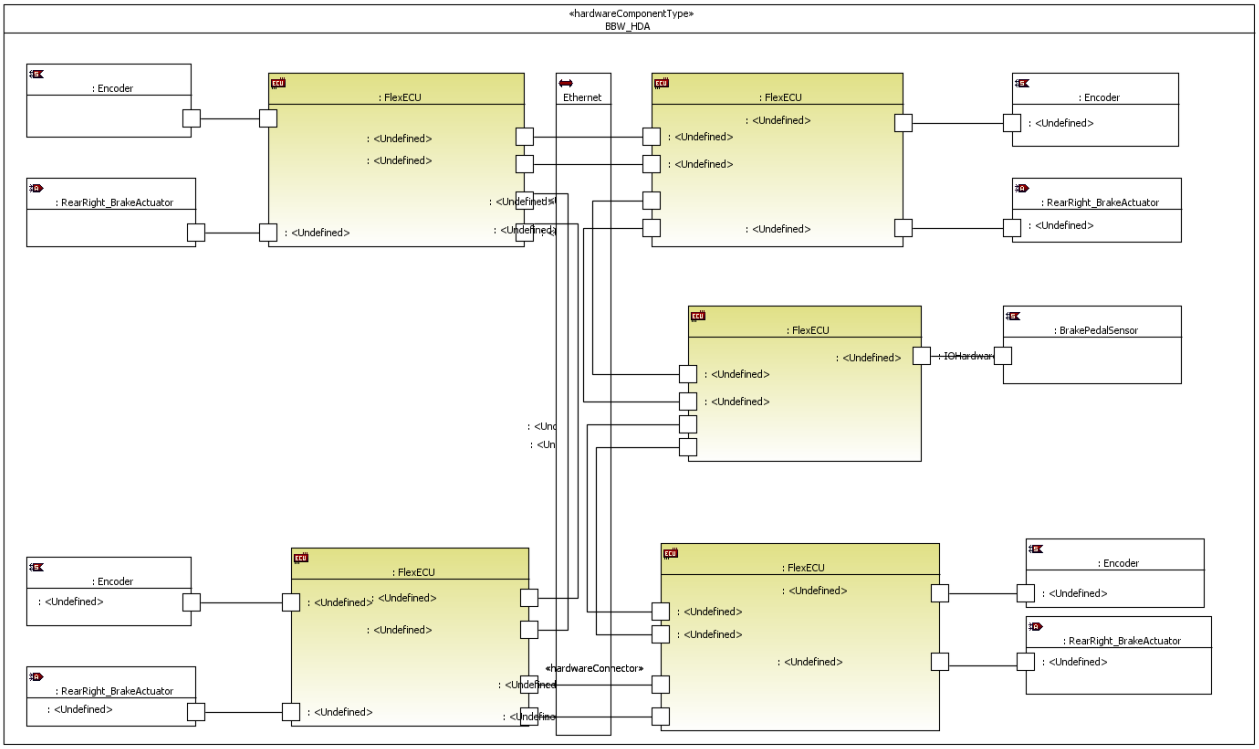

Figure 22: Hardware Design Architecture of the Braking System in Papyrus.

7.4.3 Allocation

Allocation on design level is represented in Figure 23, where function prototypes of the FunctionalDesignArchitecture are allocated to nodes in the HardwareDesignArchitecture.

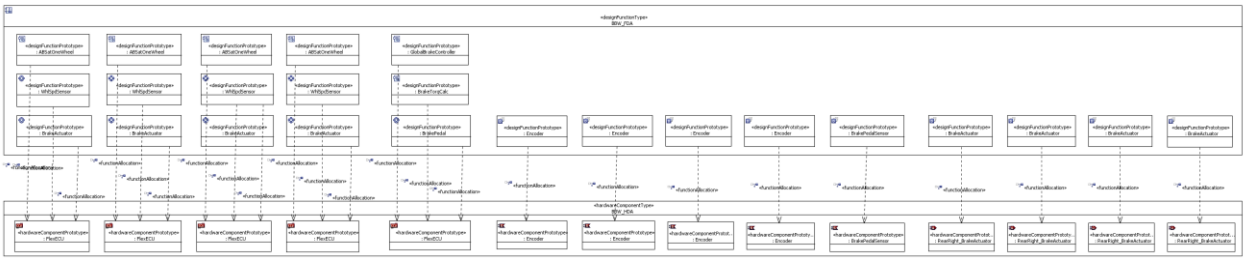

Figure 23: Function-to-node Allocation in the Braking System in Papyrus.

7.5 Implementation Level

The implementation level modelling is not yet complete. A partial model with a single wheel is shown below.

7.5.1 AUTOSAR Software Component Template

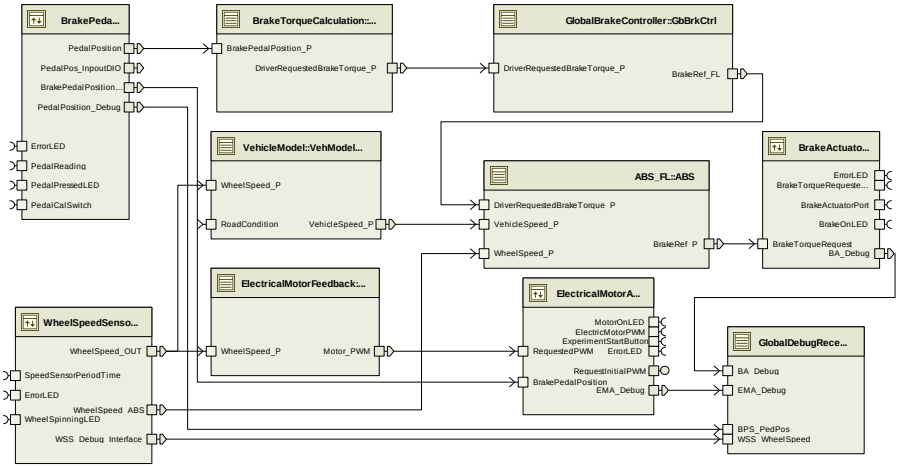

Figure 24. AUTOSAR Software Component Template of the Braking System

8 ACKNOWLEDGMENT

The authoring of this document was supported by the European FP7 project MAENAD (Grant 260057).

9 References

[1] ATESST2 Consortium (2010): ATESST2 Project web site. http://www.attest.org/

[2] AUTOSAR Development Partnership (2012): AUTOSAR web site. http://www.autosar.org/

[3] EAST-ADL Association Members (2012): EAST-ADL Association web site. http://www.east-adl.info/

[4] Internation Organization for Standardization (2011): *Road Vehicles – Functional Safety – Part 1 to 9.* International Standard ISO/FDIS 26262. November 2011.

[5] MAENAD Consortium (2012): MAENAD Project home page. http://www.maenad.eu/

[6] TIMMO Consortium (2012): TIMMO 2 USE Project web site. http://www.timmo-2-use.org/