Model-Based Analysis and Engineering of Automotive Architectures with EAST-ADL

Ramin Tavakoli Kolagari, Dejiu Chen, Agnes Lanusse, Renato Librino, Henrik Lönn, Nidhal Mahmud, Chokri Mraidha, Mark-Oliver Reiser, Sandra Torchiaro, Sara Tucci-Piergiovanni, et al.

Model-Based Analysis and Engineering of Automotive Architectures with EAST-ADL: Revisited

Ramin Tavakoli Kolagari, Nuremberg Institute of Technology, Nuremberg, Germany

DeJiu Chen, KTH Royal Institute of Technology, Stockholm, Sweden

Agnes Lanusse, CEA Saclay Nano-INNOV, Palaiseau, France

Renato Librino, 4S Group, Torino, Italy

Henrik Lönn, Volvo Group, Advanced Technology and Research, Gothenburg, Sweden

Nidhal Mahmud, University of Hull, Kingston upon Hull, UK

Chokri Mraidha, CEA LIST, Gif-sur-Yvette, France

Mark-Oliver Reiser, NumberFour AG, Berlin, Germany

Sandra Torchiaro, Centro Ricerche Fiat, Orbassano, Italy

Sara Tucci-Piergiovanni, CEA LIST, Gif-sur-Yvette, France

Tobias Wägemann, Nuremberg Institute of Technology, Nuremberg, Germany

Nataliya Yakymets, CEA Saclay Nano-INNOV, Palaiseau, France

ABSTRACT

Modern cars have turned into complex high-technology products, subject to strict safety and timing requirements, in a short time span. This evolution has translated into development processes that are not as efficient, flexible and agile as they could or should be. This paper presents the main aspects and capabilities of a rich model-based design framework, founded on EAST-ADL. EAST-ADL is an architecture description language specific to the automotive domain and complemented by a methodology compliant with the functional safety standard for the automotive domain ISO26262. The language and the methodology are used to develop an information model in the sense of a conceptual model, providing the engineer the basis for specifying the various aspects of the system. Inconsistencies, redundancies, and partly even missing system description aspects can be found automaticlally by advanced analyses and optimization capabilities to effectively improve development processes of modern cars.

Keywords: *Automotive Software Development, AUTOSAR, Dependability, EAST-ADL, Functional Safety, ISO 26262, Model-Based Software Development, Optimization, Software Product Lines, Timing Modelling*

DOI: 10.4018/IJCSSA.2015070103

1. INTRODUCTION

Commercial automobiles have become complex high-technology products in a relatively short time span. Different factors contribute to this complexity. One of them is the increasing number of vehicle functionalities supported by software, electronics and mechatronic technologies; a trend that does not seem to slow down. The involvement of carmakers in the development of these functionalities differs from one vehicle domain to the other (chassis, body, powertrain), ranging from black box integration to white-box developments. Another factor is the way in which car manufacturers have evolved from their historical mechanical and manufacturing background to the intricate organizations that develop the automobile products of today. The advent of the electrical vehicle makes this last two factors even more evident, not only because of the "untraditional" technologies that carmakers need to master, but also because the arrival of new stakeholders, actors and interests around the electrical vehicle mean that the traditional scope of the automobile has changed.

Generally, this evolution has translated into development processes that are not as efficient, flexible and agile as they could or should be (Chale et al. (2012)). The need to master these different complexity-inducing factors and improve the efficiency of product development, plus the arrival of the ISO 26262 standard (which besides from safety-related aspects, also raises issues concerning development processes of automotive systems, currently under-formalized) have motivated the adoption of model-based system engineering. Model-based system engineering advocates the use of models, conforming to a common semantic meta-model, all along the system development process. The meta-model specifies a common unambiguous semantics formalizing system engineering terminology and then providing a common language for system descriptions, i.e. models. Models, produced along the development process, provide system descriptions at different abstraction levels. Abstraction levels help human reasoning and analysis capabilities allowing system specifications to be refined and incrementally validated as long as the comprehension of the system increases. The meta-model approach is also attractive for system development as meta-models and their related models can be easily extended to support an open ended evolution of domain specific concepts. The resulting (information) models are conceptual models in the sense of a conceptual structure: the models impose a machine adequate outer structure upon the otherwise unbound creativity of the engineers, at the same time trying to be as flexible as possible to support a stimulating creativity process. The more concrete the abstraction level becomes (i.e., the more formally accessible system details are described) the lesser becomes the creativity freedom because the information model becomes more constraining until the code—as the final, formal specification—does not offer any more freedom as its respective set of instructions. This means that the information model described in this paper warily reduces the freedom that initially is of utmost necessity for the creativity process of the engineer along the abstraction levels in favour of the productivity of the computer, which in turn helps to find inconsistencies, redundancies, and partly even missing system description aspects by advanced analyses and optimization capabilities to effectively improve development processes.

Thanks to these capabilities, the adoption of model-based design has several benefits including an improvement of quality, through a more rigorous and costless traceability between requirements, design, analysis and testing. While the benefits of model-based design are widely understood, there is no COTS solution today providing a full-fledged model-based environment for automotive systems. The first problem is that many commercial solutions use proprietary meta-models that scarcely fit automotive design needs. Moreover, ideally, the meta-model should be shared in the entire automotive domain, and then proprietary languages should be avoided opting instead for standard languages. UML extensions as SysML, could be an option, but SyML, per se, does not support many concepts of vital importance for the automotive domain, as for instance, concepts for safety analysis, timing analysis and variability. To support these concepts UML needs to be specialized through specific profiles. Even though some efforts have been spent in that direction in literature—e.g. for safety (Cancila et al. (2009)), for timing (OMG MARTE (2011))—we did not reach the stage in which these efforts are unified and integrated in SyML.

EAST-ADL (EAST-ADL (2015)) (Electronics Architecture and Software Technology—Architecture Description Language) is an architecture description language specially targeting automotive systems. EAST-ADL, is a result of a series of consecutive projects: EAST-EEA (EAST-EEA (2001)), ATESST I, ATESST II (ATESST (2006)), and MAENAD project (MAENAD (2009)). Currently, EAST-ADL is managed by the EAST-ADL association (EAST-ADL (2015)). The main objective of EAST-ADL lies in encompassing all the relevant concepts to holistically support automotive engineering activities. To this end EAST-ADL aims at addressing: the ISO 26262 standard (ISO (2009)), which provides a general framework for functional safety handling in automotive systems, electrical vehicle specific concerns and related standards, along with timing, variability, and feature modeling. We believe that EAST-ADL successfully supports automotive engineering by providing methodological guidance on models to produce at different abstraction levels. Moreover, the methodology defines separated design flows (called swimlanes) following a separation of concern principle. A core design flow, only dedicated to mainstream system activities, is complemented by other three design flows, one to handle safety related activities (in conformance with the ISO26262), one dedicated to timing-related activities, and one called FEV (fully electric vehicle) swimlane to manage all the activities specific to fully electric vehicle sub-systems (not included then in the core design flow).

The language and methodology provided by EAST-ADL form the basic building blocks for a complete automation of the system development process. EAST-ADL comprises behavioral analysis of system functions, safety assessment, power analysis, and timing analysis. For safety and timing analysis advanced algorithms have been conceived, in order to provide fine predictions on system properties. Moreover a model-based optimization framework completes the panorama, by adding the capability of optimizing the system under conflicting objectives. A typical example comes when the safety assessment outcome suggests adding a software redundancy mechanism. The software redundancy mechanism, on the other hand, is time and resource consuming and can degrade system response time. Multi-objective optimization finds the right trade-off to not degrade too much system performance (or other safety conflicting goals as economic cost) and to assure the right level of safety.

This paper aims at presenting on one side the remarkable coverage of EAST-ADL as a conceptual model for the broad range of relevant concepts in the automotive domain, and on the other side the high level of integration of novel sophisticated analyses, for safety and timing in particular, and optimization capabilities to make adequate use of the productivity only computers can offer. To this end, we firstly present the EAST-ADL methodology, explaining in detail swimlanes and abstraction levels. We present all the models produced in the core swimlane using a case study running example. For the non-core swimlanes, we illustrate in the paper the safety and timing swimlane, in order to present the novel analyses developed during the MAENAD project. Principles for model-based optimization are also presented as the opportunity of dealing with conflicting goals.

The paper is organized as follows: Section 2 presents a literature review on modeling languages and related model-based design techniques. Section 3 presents the principles of the system development process depicting the abstraction levels EAST-ADL models conforms to. Section 4 presents the EAST-ADL methodology and swimlanes. Section 5 illustrates the core swimlane, presenting the models to be produced at each abstraction level and an example of verification activity based on model-checking. Section 6 presents main models to be produced in the safety swimlane and novel concepts for safety analysis. Section 7 presents main concepts for timing modeling and presents advanced analysis for system schedulability estimation. Section 8 presents recent research activities of a multi-objective optimization approach, while Section 9 concludes the paper.

2. BACKGROUND

In this section we will evaluate modeling languages with regard to their capacity to support non-functional properties covering safety aspects, timing aspects and FEV aspects. We focus on standard modeling languages.

The Table 1 lists some standard modeling languages and shows whether they provide support for safety, timing, multi-core, and FEV non-functional aspects.

The table presents the abstraction level each language can be used. At system abstraction level, vehicle features are specified and logical functions realizing the features are identified. At design abstraction level the functional architecture, the hardware architecture, and an allocation of the functions to hardware resources are defined. On the implementation level, the design architecture is implemented on a concrete platform.

AADL (SAE (2009)) (Architecture Analysis and Design Language) is an architecture description language standardized by SAE. AADL was first developed in the field of avionics and derived from MetaH, made by Honeywell. AADL is a Domain Specific Modeling Language designed for the specification, analysis, and automated integration of real-time performance-critical distributed systems. It allows analysis of designs prior to the implementation. AADL is adapted for systems architecture specification but lacks some standard support for requirements specification at system level.

UML (Unified Modeling Language) (OMG UML (2015)) is the most known modeling language. UML provides several views/diagrams for modeling structural and behavioral aspects of a system. UML is an object-oriented modeling language that is suitable for object-oriented software design. It is a general purpose modeling language, and while providing a basic support for timing specification (through timing observations and timing durations concepts), it lacks detailed modeling concepts for non-functional properties. For more specific modeling needs, UML provides a standard extension mechanism called profile. Examples of standard UML profile are SysML (OMG SYSML (2012)), MARTE (OMG MARTE (2011)) or QFTP (OMG QFTP (2008)).

Table 1. Standard modeling languages

Modeling Language	Safety	Timing	Multi-Core	FEV	Level
AADL	✓	✓	(✓)		Design/Implem
UML		✓			Design/Implem
SysML		✓			System
MARTE		✓		✓	Design/Implem
QFTP	✓				Design
AUTOSAR	✓	✓	(✓)	✓	Implem

SysML is a specialization of UML for system modeling. SysML provides support for requirements modeling, and a support for quantities and dimensions as the only support for non-functional aspects. MARTE is an extension of UML for real-time and embedded systems. It provides advanced concepts for the design and analysis of such systems, including modeling constructs for non-functional properties, time, resources. However, safety aspect is not addressed by the standard.

QFTP is a UML profile for modeling quality of service and fault tolerance characteristics and mechanisms, but it is not especially conceived to support safety activities prescribed by the ISO26262 standard.

Except for AUTOSAR, none of the presented languages provide support for the whole set of non-functional properties of interest. However, AUTOSAR targets the implementation of software parts of automotive systems and does not provide concepts for requirements specification, refinement and traceability; hindering then its application at system level.

Mixing different languages to cover all levels (system, design and implementation) with a support for all non-functional properties modeling may be inappropriate. In fact, a joint usage of modeling languages can raise semantic and/or syntactic problems (Di Natale et al (2010)). Transformations from one language to another can lead to a loss of semantics in the output model. This can be the case even between two UML profiles like SysML and MARTE (Espinoza et al (2010)).

3. PRINCIPLES OF THE OVERALL SYSTEM DEVELOPMENT PROCESS

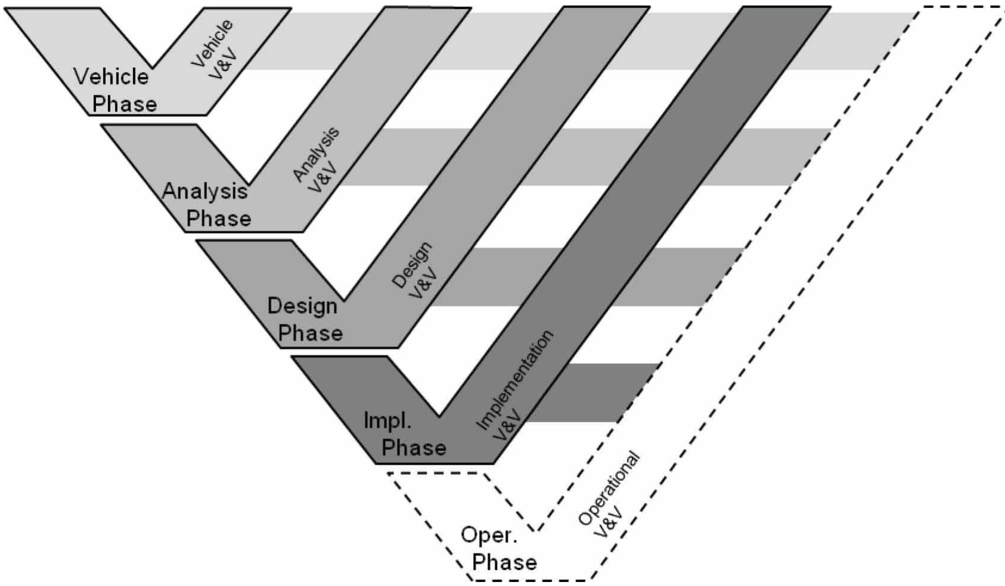
This section is devoted to the presentation of the main EAST-ADL principles guiding the development process of automotive systems. To explain how EAST-ADL can be used, a classical V model is used as reference. According to the classical V model, design steps from top level requirements collection to component realizations are represented on the left in the V. Bottom-up test and integration is represented on the right side of the V.

Four development phases, corresponding to four abstraction levels of a system model, are identified. Figure 1 shows the EAST-ADL phases in a V model context. Vehicle phase corresponds to the early stages in the product life cycle, while implementation phase corresponds to the final stages of development. The Operational phase corresponds to stages in which the concrete vehicle is operational.

The development phases can be detailed as follows:

- **Vehicle phase:** During this phase the analysis of external requirements is carried out with the objective of constructing a top-level feature model. The top-level feature model provides the most abstract definition of expected functionalities by a vehicular embedded system. Through a vehicle feature model, the expected system functionalities in terms of features are configured and linked to the corresponding specifications of system level requirements, verification and validation cases;
- **(Functional) Analysis phase:** Typically based on the inputs of automatic control engineering, system design at this level refines the vehicle level system feature specification by indentifying the individual functional units necessary for system boundary (e.g., sensing and actuating functions for the interaction with target physical plant) and internal computation (e.g. feedback control functions for regulating the dynamics of target physical plant). The design focuses on the abstract functional logic, while abstracting any SW/HW based

Figure 1. EAST-ADL phases in a V-model context



implementation details. Through an analysis level system model, such abstract functional units are defined and linked to the corresponding specifications of requirements (which are either satisfied or emergent) as well as the corresponding verification and validation cases;

- **Design phase:** System design at this level refines the analysis level model to obtain a logical representation of the system functional units that are now structured for their realizations through computer hardware and software. This representation is obtained by capturing the bindings of system functions to I/O devices, basic software, operating systems, communication systems, memories and processing units, and other hardware devices. Again, through a design level system model, the system functions, together with the expected software and hardware resources for their realizations, are defined and linked to the corresponding specifications of requirements (which are either satisfied or emergent) as well as the corresponding verification and validation cases. Moreover, the creation of an explicit design level system model promotes efficient and reusable architectures, i.e. sets of (structured) HW/SW components and their interfaces, hardware architecture for different functions. The architecture must satisfy the constraints of a particular development project in automotive series production;
- **Implementation phase:** This phase focuses on 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. However, the use of AUTOSAR concepts is not mandated by the methodology. Other, in particular more traditional implementation concepts can be used in this phase while leaving the other phases unchanged.

It is worth to notice that at the end of each phase, the main artifact to produce is a complete system definition in EAST-ADL. While being defined with the specific concepts used at each abstraction level (features, functions, etc.), each EAST-ADL model consists essentially of a set of abstract system constituent entities. Typically, “m-to-n” relationships (from n entities of a higher level to m entities of the lower level) allow refining models throughout the process for an incremental system concretization. In particular, the analysis architecture and its requirements is a refinement of the feature model. This results in “m-to-n” relationships between the vehicle feature model entities and analysis architecture entities. In its turn the design architecture is a refinement of the purely functional analysis architecture. Again “m-to-n” relationships between the analysis functional architecture entities and design architecture entities there exist. Figure 2 shows an example of ‘m-to-n’ relationships between features, analysis functions and design functions.

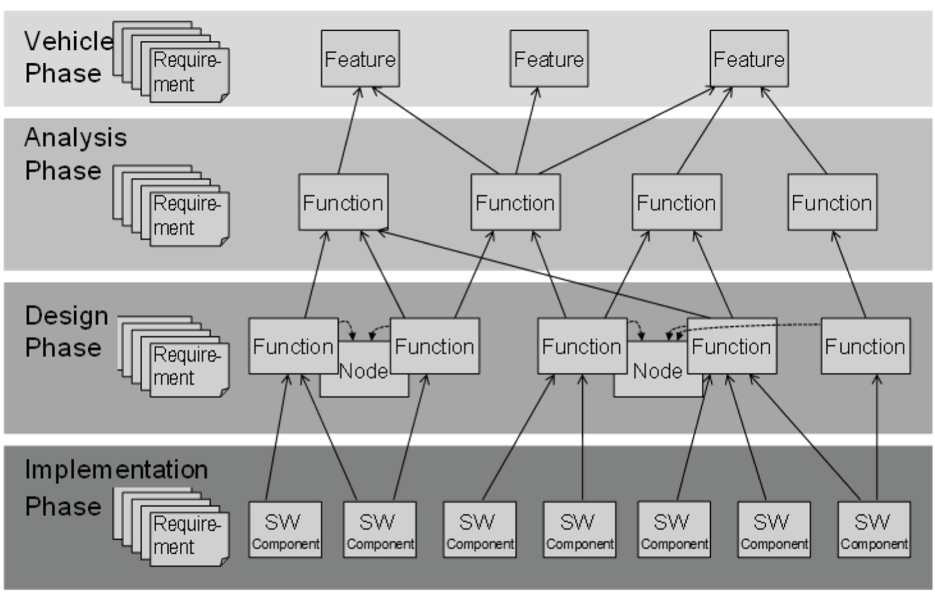
Summarizing EAST-ADL phases serve to explain the steps involved in engineering automotive systems, following a staged approach for integration, validation and verification. This staged approach not only allows addressing the requirements corresponding to the current abstraction level, but implicitly allows addressing, all the way back, top level requirements as the design evolves and gets more concrete.

Let us remark that, even if illustrated on a V cycle process, the involved engineering steps can be deployed in any overall framework, from waterfall to agile development. Together with the support of EAST-ADL language, this definition of development phases allows a common reasoning of engineering activities, related design artifacts, and thereby any need for traceability, reuse and safety lifecycle management.

4. EAST-ADL METHODOLOGY – SWIMLANES AND MAIN ACTIVITIES

In this section we aim at presenting the EAST-ADL methodology. The methodology elaborates system development process principles presented in Section 3, giving guidelines on the set of

Figure 2. EAST-ADL models and refinements



engineering activities that help in incrementally define and validate designs in compliance with relevant standards.

Interestingly, the EAST-ADL methodology defines and organizes development activities in so-called swimlanes (MAENAD Methodology (2013) and TIMMO2USE (2012)). The structuring of swimlanes follows a separation-of-concerns principle, in which core system design activities are separated from activities related to specific aspects, as safety, timing and FEV. This structuring results in four separated lanes:

- Core lane, covering design core activities, i.e. activities aiming at (i) representing core system structure and behavior and (ii) verifying functional properties of the system;
- Functional Safety lane, covering safety activities, i.e. activities aiming at representing and assessing system safety. The safety lane conforms to the ISO26262 standard;
- Timing lane, covering timing activities, i.e. activities aiming at representing and verifying timing properties of the system;
- FEV lane, covering FEV-related activities; i.e. activities aiming at representing and assessing system properties related to electric vehicles, in compliance with electric vehicle standards.

It is worth to notice that each lane covers all the EAST-ADL phases (feature, analysis, design, etc.). The possibility, however, to carry out some particular verification/assessment uniquely depends on the information available at each level of abstraction. For instance, timing activities related to schedulability estimation are not applicable until the design phase is reached, as information of hardware resources and allocation is needed. The same reasoning applies to most of the FEV related activities, which are mainly related to hardware properties. In the following, for each lane, assessment/verification activities are detailed for the Feature, Analysis and Design phases.

4.1. Safety Assessment Activities

The Functional Safety lane has been modeled by taking into account the ISO 26262 safety life-cycle. In the automotive domain, the ISO 26262 standard provides a complete set of process flow recommendations covering analysis, design and implementation of safety-critical systems and helping to respect the safety issues all along the life-cycle.

Following a top-down approach, the functional safety lane starts from the EAST-ADL Vehicle phase. The activities at this phase include Item definition in terms of target feature and the malfunction definition (feature flaws), as anomalies of the Item's outputs. On Vehicle phase, it is already possible to perform a Preliminary Hazard Analysis (PHA) and risk assessment, to estimate the level of risk associated with the Item (called Automotive Safety Integrity Level or ASIL), and to define a safety goal (and if it is possible the safe states) for each hazardous event identified, as well as, the set of essential safety requirements.

Once the functional safety concept (safety goals, ASILs, safety requirements, hazards and risks) is specified, the Item can be developed with a system perspective in the EAST-ADL Analysis Phase. The functional safety lane at the EAST-ADL Analysis Phase includes definition of the functional safety requirements and then their allocation to a preliminary architecture. At this phase, the System Hazard Analysis (SHA) can be conducted – in addition to the PHA and risk assessment – to study the propagation of failures across the system architecture. At this level the preliminary safety assessment can be conducted as well. Typical safety assessment at this phase employs Fault Tree generation and qualitative Analysis (FTA), Failure Mode and Effects Analysis (FMEA), Common Cause Analysis (CCA). FTA and FMEA are complementary methods to analyze propagation of faults through the system. Since there is no information on allocation to the hardware components available at the Analysis phase, only qualitative analysis can be performed. Expected results may include list of possible failure modes of analyzed system components, fault trees generated for the feared (or top) events corresponding to the priory defined hazards and risks, minimal cut sets, FMEA tables and a list of common cause failures (CCF).

During the Design phase, it is possible to define the technical safety requirements and allocate them to architectural elements. This allows performing quantitative analysis of system hardware components. Safety activities imply refining priory obtained FTA, FMEA and CCA results, performing quantitative FTA, and in particular, calculating probability of the top events and minimal cut sets, calculating Probability of Failure on Demand (PFD) and Probability of Failure per Hour (PFH), etc.

In Section 6 we present the main models prescribed by the EAST-ADL methodology in the safety swimlane along with advanced safety analyses developed during the MAENAD project (see Table 2).

4.2. Timing Assessment Activities

The Timing swimlane gives guidelines on timing related activities conducted on EAST-ADL phases. Timing activities start on the EAST-ADL Analysis phase where functions that realize the vehicle features are introduced. This functional model is enriched with timing properties (e.g. activation rates for chain of functions) and constraints (e.g. end-to-end deadlines). However, in the Analysis phase, functions worst case execution times (WCETs) are unknown. In fact these WCETs needed for timing properties verification are available only after the implementation phase, which is a quite late point in the process to detect design errors. As a workaround to the missing WCETs, an activity called Time Budgeting allows specifying so-called time budgets. A global budget that might come from end-to-end deadlines is decomposed and allocated to functions. The outcome of this time budgeting activity is to enrich each function with a budget for its execution time and a budget for its local deadline. These budgets represent the timing requirements used as input of the following Design phase (see Table 3).

Table 2. Safety lane main activites

Phase	Verification Activities & Methods	Expected Results
Vehicle	Preliminary Hazard Analysis Risk Assessment	Hazards & Risks, Safety goals and ASILs, Safety Requirements
Analysis	Preliminary Hazard Analysis and Risk Assessment System Hazard Analysis using FTA&FMEA Preliminary Safety Assessment using FTA, FMEA, CCA	Hazards & Risks, Safety goals and ASILs, Safety Requirements (refined) Qualitative FTA, Minimal Cut Sets, Top Events, FMEA, CCF
Design	Safety Assessment using FTA, FMEA, CCA	Qualitative & Quantitative FTA and FMEA, Top Events probability, PFD, PFH, CCF

Table 3. Timing lane main activities

Phase	Verification Activities & Methods	Expected Results
Vehicle	N/A	N/A
Analysis	Time budgeting (methods: constraints resolution)	Time budgets on functions
Design	Time budgeting, Schedulability estimation (model checking, schedulability analysis, simulation, optimization)	Time budgets on design functions, nodes and buses utilization, response-times

On the EAST-ADL Design phase, two main timing activities might be conducted:

1. The first one uses the function decomposition made in the Design phase to perform time budgets refinement. The result of this time budgeting activity is to enrich each atomic design function with an execution time budget. These refined budgets represent the timing requirements used as input of the implementation phase;
2. The second timing activity makes use of the refined time budgets, the allocation model and the timing specification, to achieve a first estimation of system schedulability. Actually, schedulability analysis applies at implementation level, once the application has been mapped on execution tasks. Nevertheless, at design level, simulation and verification techniques can detect timing errors on functional models. In TIMMO (TIMMO (2007)) and TIMMO-2-USE (TIMMO2USE (2012)) projects, some methodologies for timing validation & verification of EAST-ADL models have been proposed. In this context (Arda, G. et al. (2013)) provide a formal validation & verification approach based on simulation and model checking for the design phase. In MAENAD we proposed another approach for schedulability estimation of EAST-ADL models, based on optimization techniques and schedulability analysis. Section 7 presents such an approach along with main EAST-ADL models enabling schedulability estimation.

4.3. Fully Electric Vehicle Assessment Activities

The Fully Electric Vehicle swimlane is a guideline to develop FEVs by addressing the systems that are specific of this kind of vehicles. In particular, the functions and the systems considered have been grouped as follows: Electric propulsion, Regenerative Energy Storage, Regenerative Braking, Recharging, Energy conversion, Insulation and Protection, Anti-theft system andHuman Machine Interface (HMI). In order to provide an effective support to FEV development, for most of the activities of the process, the reference to the applicable standards and regulations, and some synthetic requirements of the norms are sufficient. The norm references include the relevant ISO, IEC, EN, SAE standards, and UNECE and FMVSS regulations. Table 4 illustrates specific analysis activities related to FEV development, such as energy flow analysis, vehicle

Table 4. Fev lane main activities

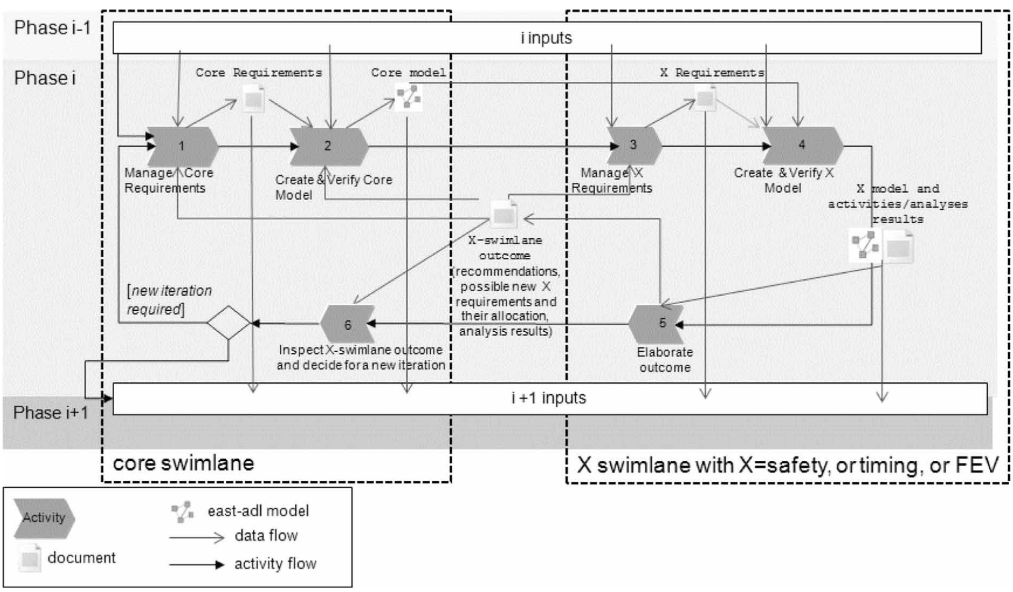
Phase	Verification Activities & Methods	Expected Results
Vehicle	N/A	N/A
Analysis	Analysis of vehicle performance , energy consumption and range through functional simulation	Verification of vehicle performance, energy consumption and range, according to test condition and test case requirements
Design	Insulation analysis through on-purpose analysis	Overall insulation resistance, voltage compliance
	Analysis of charging inlet voltage decrease through circuit simulation	Compliance with voltage-time requirements of charging inlet
	Analysis of the Regenerative Energy Storage System through multi-physics simulation	Current and thermal effects in the case of short circuit
	Matching analysis of power equipment through functional simulation	Matching verification of component current and voltage
	Power requirement analysis in key-off mode through power budgeting	Compliance with specification
	Analysis of the Braking System through functional simulation	Phasing correctness of braking sources, safe brake operation at battery depleted state of charge, compliance with battery limits

performance and range analysis, insulation resistance compliance analysis, etc. In the model-based framework developed in the MAENAD project, FEV-related analyses have been applied thanks to the integration between EAST-ADL models/tools and simulation models/tools, but no research efforts have been conducted on the analyses themselves. For this reason activities of the FEV swimlane are not further detailed in this section.

4.4. Interdependencies among Swimlanes

Swimlanes identify activities related to a specific concept (core, safety, timing, FEV), however, they cannot be considered as independent. In particular it is expected that core swimlane artifacts will be input for safety, timing, FEV-swimlanes and viceversa. Figure 3 shows a general process, formalized in SPEM, describing the interaction between the core swimlane and an X-swimlane (where X stands for Safety, or Timing or FEV) in any phase '*i*' (where *i* stands for Vehicle, Analysis, Design). This process is iterative: through activity 1 'manage core requirements', core requirements are defined (possibly refined from higher-level requirements and taking into account the outcome of the previous iteration, if any). Core requirements and the core model from the higher phase (if any) are the input of activity 2: "create and verify core model". Through this activity, a solution for the core model is defined. This core model is input for the X-swimlane. In the X-swimlane, X-related requirements are managed through activity 3 (possibly taking as input the X-requirements coming from the higher phase) and then an X-model is defined and verified through activity 4 'create and verify X-model'. The X-model captures all the information required to carry out verification activities/analyses prescribed for the given swimlane at the given phase. This model is typically built on the core model, by adding annotations to capture the X-related information. Once these activities are performed the X model can be enriched with activities/analysis results. Finally activity 5 'elaborate outcome' provides a complete document representing the outcome of the swimlane for the given phase. This document could serve to

Figure 3. Spem process describing interaction between core and the other swimlanes



refine/validate X-requirements (coming back to activity 3) and/or could contain recommendations for the core swimlane. In the core swimlane the X-swimlane outcome will be inspected through activity 6 'inspect X-swimlane outcome and decide for a new iteration'. Through this activity it will be decided if a new iteration is needed or it is possible to move down to the lower phase.

The methodology does not specify in detail how to proceed if the outcomes from different X-swimlanes are conflicting. For instance the safety swimlane can recommend adding a software redundancy mechanism to address a safety goal, whereas the timing swimlane will declare the software redundancy mechanism as exceeding the maximal CPU resource utilization. The resolution of this kind of conflicts is in general left to experienced system engineers. However, due to the increasing level of complexity of automotive architectures, system engineers cannot solve this kind of conflicts only relying on a manual approach. For this reason, automatic model-based optimization is going to play a central role in system development, helping system engineers to find appropriate trade-offs in case of conflicting goals. In the MAENAD project, a model-based optimization framework has been developed and will be presented in Section 8.

In the following we will detail modeling and verification support provided by the MAENAD model-based framework, focusing on the core, safety and timing swimlane.

5. MODELS AND VERIFICATION IN THE CORE LANE

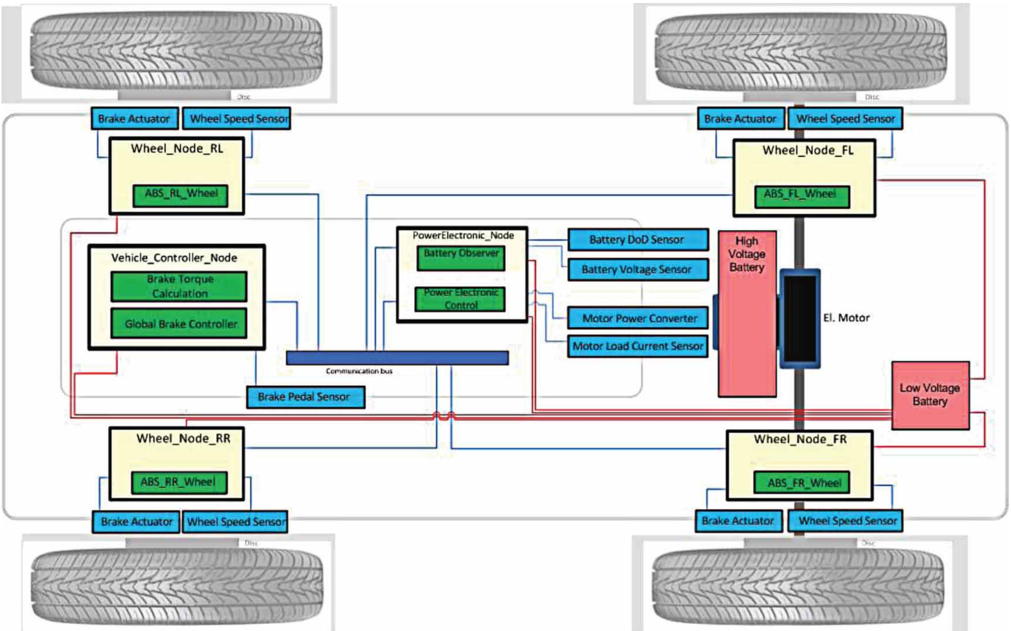
The main aim of this section is to present the modeling concepts and their use to build artifacts in the core swim lane through a case-study, i.e. a power regenerative braking system. It shows how EAST-ADL covers design concepts throughout the design process, from high-level features to hardware/software functions identification, to capture key architectural concerns.

5.1. The Power Regenerative Braking System

The system combines conventional braking with power regeneration, architectural concepts are shown in Figure 4. The driver request is monitored through a brake pedal position sensor. A rotation speed sensor, an ABS (Anti-lock Braking System) controller, and an electromagnetic brake actuator are placed at each wheel. When requested, the ABS controller regulates the braking through the electromagnetic brake actuator. By collecting the measured wheel rotation speed, an ABS controller also detects the occurrence of wheel slip by comparing the measured rotation speed with current vehicle speed. In the case of wheel slip, the controller adjusts the brake torque value for maximizing the traction and braking effectiveness. For the braking control of entire vehicle, a global brake controller receives the measured wheel rotation speeds and driver braking request and then sends an estimation of current vehicle speed and brake force request to each ABS controller. Instead of having the braking force completely realized by electromagnetic brake actuators, the regenerative braking allows a fraction or whole of kinetic energy of braking to be recovered and stored in battery. To support this, the global brake controller also receives the observed battery and motor status and estimates the maximum possible braking torque to be offered by an electrical motor. The controller then arbitrates the braking torques to be provided by the motor and brake discs.

The system implementation will be based on a distributed electrical architecture with 6 nodes: one central vehicle control node, four wheel brake control nodes, and one power electronic control node, as illustrated in Figure 4. The system has a communication network for distributing signals. For example, the estimated battery status information is fed from the battery observer to the power electronic control function for the torque estimation.

Figure 4. Architectural concepts of the regenerative braking system



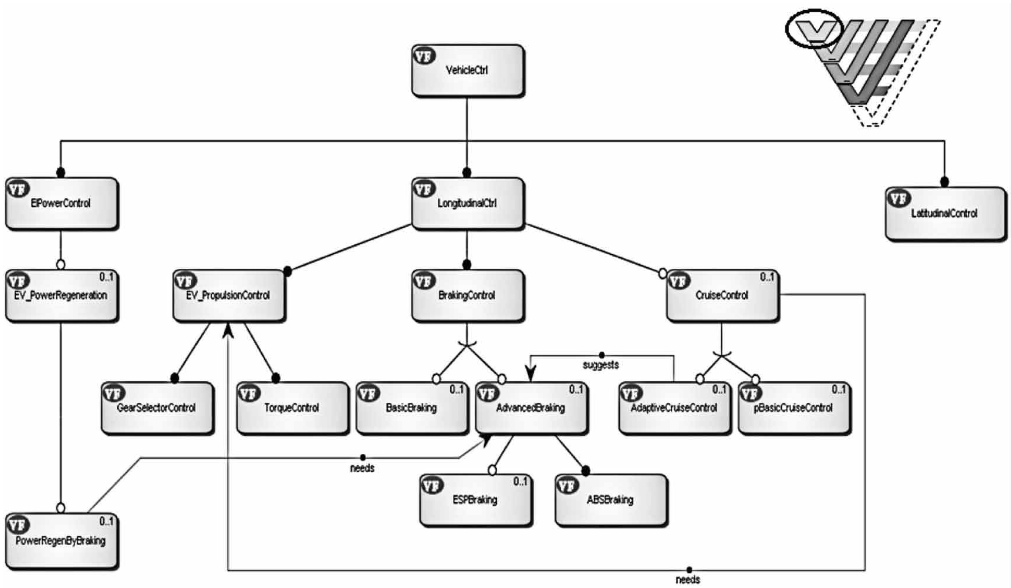
5.2. Core Models

5.2.1. Vehicle Feature Model

On Vehicle Level, the abstract system description is obtained by managing the features of an entire product line. In Figure 5, the feature tree of the braking system is shown. Each vehicle feature (VF) denotes a functional characteristic, such as the functional, non-functional, or even mechanical properties to be supported. As a child of the longitudinal control (LongitudinalCtrl) feature, the braking control feature (BrakingControl) is needed for the vehicle longitudinal control. The regenerative braking feature (PowerRegenerativeBraking) is a child feature of power control, allowing the kinetic energy produced by braking to be converted to electrical energy and stored in capacitor or/and battery. The interdependencies of vehicle 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).

In EAST-ADL, this feature model is the so-called core feature model, i.e., a technical feature model, describing on the topmost level the abstract boundaries of the system, although the architectural boundaries are, of course, not concretely defined, because the first architecture description happens on Analysis Level. This core feature model is connected via a Configuration Decision Model to the topmost feature models on Analysis Level and Design Level each. The Configuration Decision Model collects a set of configuration decisions. Each configuration decision expresses a connection between a specific (de)selection of features in the source feature model (here: the core feature model) and the required (de)selection of features in the target feature model (here: either the topmost feature model on Analysis Level or the topmost feature model on Design Level). By that, a pre-selection of features in the core feature model requires a specific pre-selection of features in the respective target feature models, i.e., the core feature

Figure 5. Vehicle feature model of the regenerative braking system



model is the main source for configuring the variability of the system. Let us note that feature modeling in EAST-ADL is an orthogonal concept, not only providing variability for Vehicle level then, but for the Analysis and Design levels as well. This capability is used for design space specification as explained in Section 8.

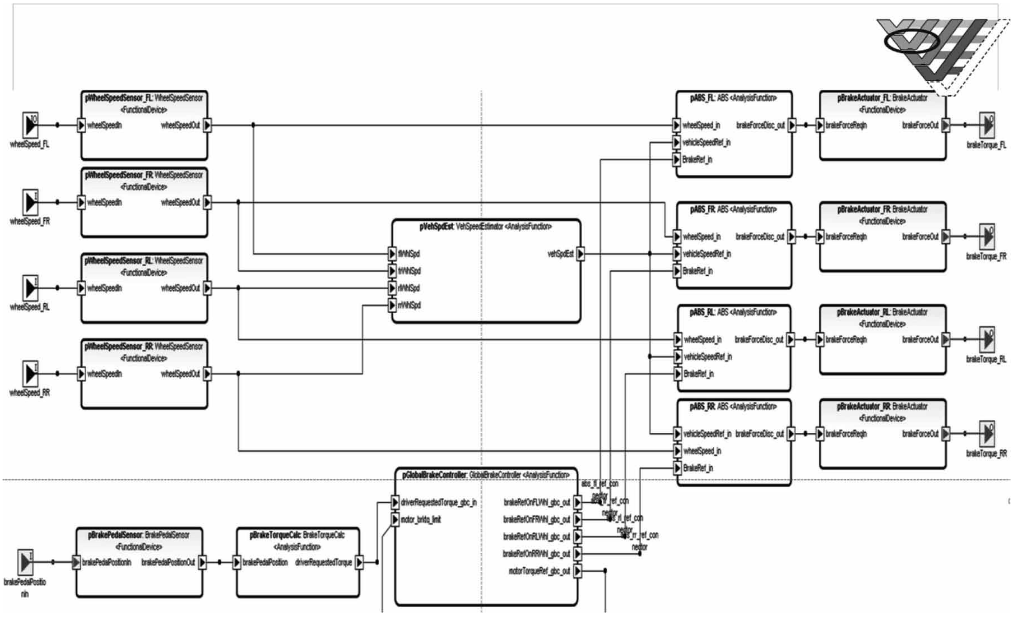
5.2.2. Functional Analysis Architecture Model

As the first step towards system realization, each vehicle feature (VF) of concern is refined into some functional design solutions, mainly from a control engineering perspective. Figure 6 shows an excerpt of the functional analysis architecture (FAA) for the vehicle feature ABS braking (ABSBraking). For design traceability, a realisation model is used to maintain the realisation mappings from vehicle features to the functional design solutions.

Compared to the vehicle feature models, a functional analysis architecture model provides additional information about the variables to be monitored and controlled, the internal computation blocks and their interactions. Here, the types of system internal computation blocks and the types of the system external I/O transformation blocks are classified by two different EAST-ADL constructs, i.e. AnalysisFunction and FunctionDevice, respectively. The composition description follows the basic *type-prototype* pattern of EAST-ADL, where a prototype represents a particular instantiation of a given type in a context. For example, the design shown in Figure 6 includes four prototypes (one for each wheel) of the same function device (WheelSpeedSensor).

In regard to execution, each function prototype runs according to a *run-to-completion* semantics (Chen et al. (2013)): when triggered, it reads all input parameters, executes the computation, and then writes the output parameters. If new data of the input parameters arrives during the execution or writing phase, it cannot be processed in the current executing cycle. Moreover, each port represents a one size buffer that does not block the sender when it is full or the receiver when it is empty. Each connector relates a pair of ports of the same type with a shared variable semantics.

Figure 6. Functional analysis architecture of the regenerative braking system (excerpt)



5.2.3. Design Architecture Models

The design level architecture further details the analysis level design by taking the software and hardware resources into consideration. Again, the design traceability is maintained by a realisation model. Figure 7 shows an excerpt of the functional design architecture for the braking system. As it can be noted each wheel speed sensor analysis function (WheelSpeedSensor) in Figure 6 is here refined in two different design level functions: 1. one hardware transfer function for the encoder hardware (WhlRotationEncoder), of which the type is classified by the construct HardwareFunction; and 2. one design function for the encoder software (WhlSpeedSensor-Device), of which the type is classified by the construct DesignFunction. While a hardware transfer function is realized directly by hardware, a design function has instead a software based implementation. For example, the WhlSpeedSensorDevice function is further decomposed into a local device manager for application interactions and a basic software module for lower level hardware control. Each prototype with the corresponding type classified by design function has the same run-to-completion execution semantics as for the analysis level functions. The execution of a hardware transfer function is however given by the corresponding physical hardware. EAST-ADL allows additional behaviour constraints in regard to the physical dynamics to be annotated (see Section 5.3.).

One particular architectural design decision is related to the deployment of functions on hardware. To this end, EAST-ADL provides necessary language support for hardware modelling. The focus is on the specification of available electronic and electrical resources as well as the circuit design, such as communication network, I/O devices, ECUs (electronic control units) and power supplies. Being the allocation targets of design functions, these hardware resources are characterized by properties like memory size, clock frequency, bandwidth, etc. Figure 8 shows an excerpt of the hardware architecture model for the braking system, including the encoder

Figure 7. Functional design architecture of the regenerative braking system (excerpt)

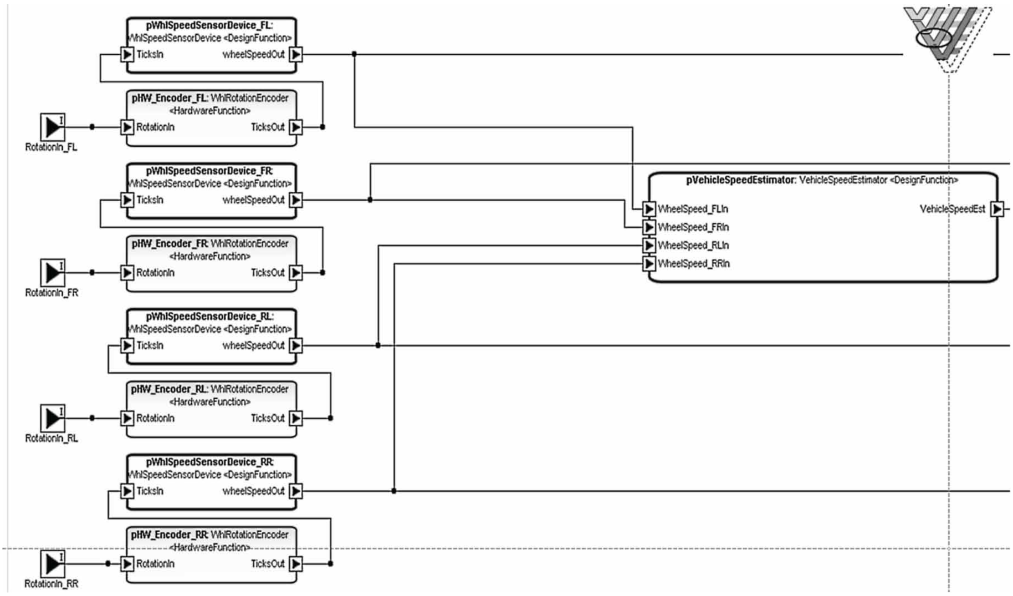
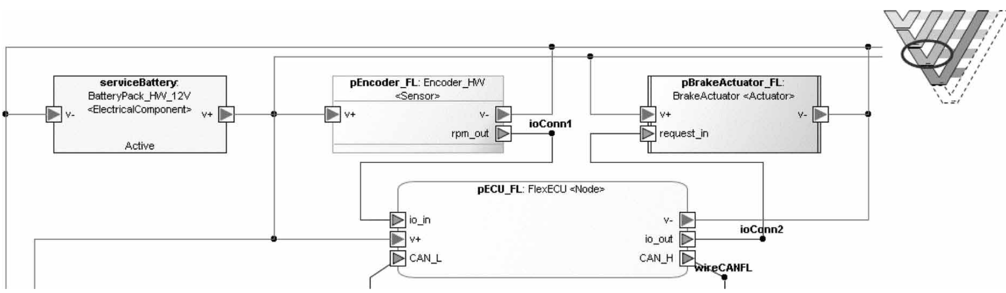


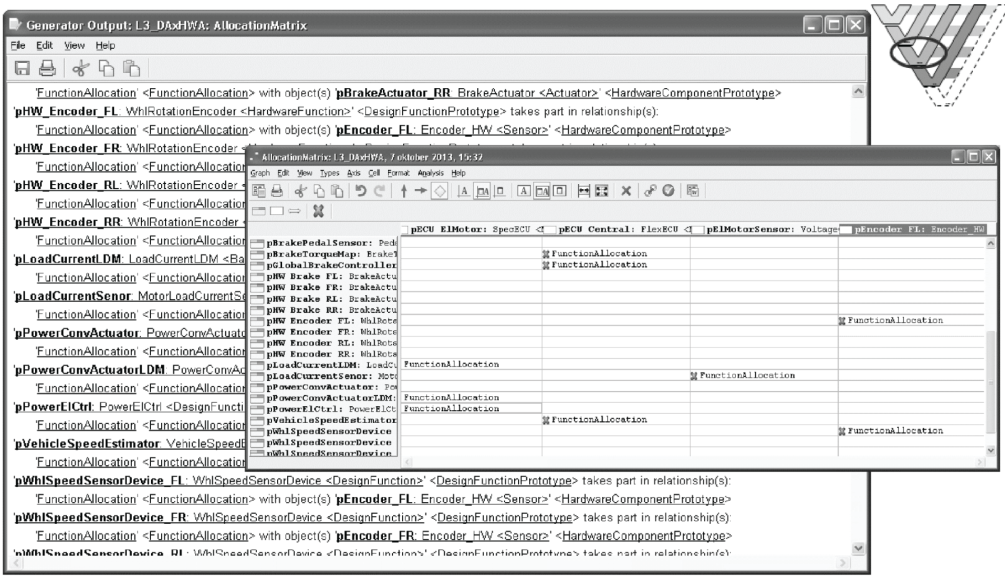
Figure 8. Hardware architecture of the regenerative braking system



device, the ECU, the brake actuator of a wheel, and the power supply unit. The connectors among these blocks represent the electrical wires, further characterized by physical circuit properties like length, resistance and insulation. Each wire connects a pair of hardware pins (HardwarePin) of the same type, representing the electronic or electrical connection points, such as the device power pins connected to the positive pole (v+) and minus pole (v-) of the 12V DC power supply (serviceBattery).

With EAST-ADL, dedicated allocation links (functionalAllocation) are introduced to specify the mapping from design functions to hardware resources. Figure 9 shows the EAST-ADL specification of allocations for some of the design functions with a matrix view and a textual view. Coming back to design functions shown in Figure 7, it can be noticed that the encoder device pEncoder_FL (with the column name checked in the matrix view) hosts now one prototype of the design function WhlRotationEncoder (pWhlSpeedSensorDevice_FL) and one prototype of the hardware transfer function WhlRotationEncoder (pMW_EncoderFL).

Figure 9. Allocation model for the regenerative braking system



5.3. Functional Verification via Model-Checking

EAST-ADL aims to obtain most of its analytical leverage through well established analysis methods and tools. To this end, the language package, referred to as Behavior Constraint Description Annex (Chen et al. (2013)), provides support for capturing and formalizing various behavioral concerns in the context of architectural design. On the basis of a formal semantics, several model transformations from EAST-ADL behavior constraint descriptions to several external tools (e.g. SPIN, UPPAAL, and Matlab/Simulink) have been developed. In the reminder of this section we introduce the basic EAST-ADL concepts for behavior descriptions through the ABS function.

5.3.1. Behavior Constraints for the ABS

In EAST-ADL, a behavior annotation, referred to as *behavior constraint*, can get different roles depending on the declared target associations. For example, such a behavior constraint can be used to capture the bounds of the acceptable behaviors of a system function. A behavior constraint can also be used to refine the textual statements of requirements including assumed system operational situations. Moreover, a behavior constraint can be introduced to provide a formalization of error annotations. The content of a behavior constraint is organized into the following three categories:

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

For example, the quantification constraint in regard to the slip rate estimation by ABS function is given in Figure 10. According to the constraint description, the estimated slip rate (*SlipRate*) should follow the slip rate quantification (*SlipRateQuantification*) with the expression: *SlipRate*=(*VehicleSpeedIn*- *WheelSpeedIn***WheelRadius*)/*VehicleSpeedIn*. Here, the *VehicleSpeedIn* and *WheelSpeedIn* are two variables received through the functional ports vehicleSpeedRef_in and wheelSpeed_in respectively. The *WheelRadius* is a constant with the value of maximum allowed wheel radius. The EAST-ADL Behavior Constraint Description Annex uses an abstract notion of time, referred to as logical time condition, as the time basis for quantifying physical dynamics by means of continuous- and discrete-time model, or for defining the timed guard conditions and invariants of state-machines or computations.

In Figure 11, the behavior constraint is further elaborated by a temporal constraint description in state machine (SM). The state invariants and transition guards are precisely defined by some attribute quantification specifications. In Figure 12, the specification of computation constraint declares two valid invocations to a transformation *Set_ABSBrakeTorqueOut* that calculates the ABS brake torque request.

As already mentioned, the EAST-ADL behavior constraint descriptions are transformed to external tools for the analysis support. This allows the engineers to exhaustively verifying the model against requirements. The verifiable requirements include assertion, freedom of deadlock, reachability of the desired state, avoidance of or compliance with given execution patterns, and linear temporal logic (LTL) statements. Figure 13 shows the PROMELA code section of the ABS function. Please note the assertion assert(0) is an added statement to guide the tool to search the execution path which indicates the locked condition of a wheel. In model checking terminology, the path is a counterexample, which gives the designer the hint on how the given state may be reached.

Figure 10. The attribute quantification constraint description for an ABS function

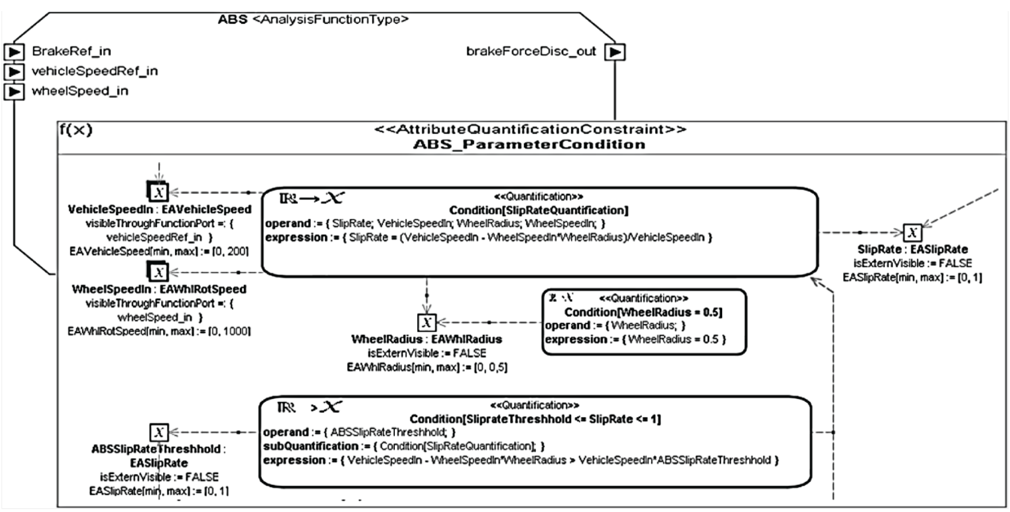


Figure 11. The temporal constraint description for an ABS function

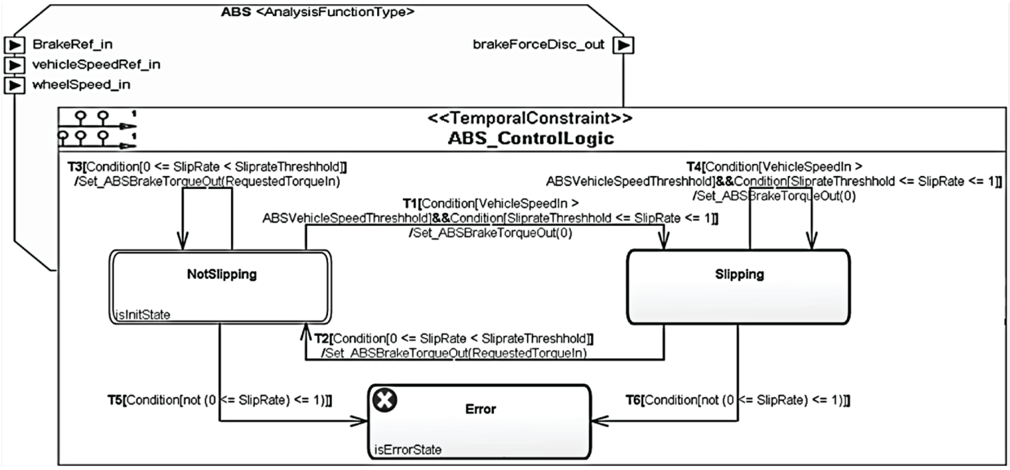
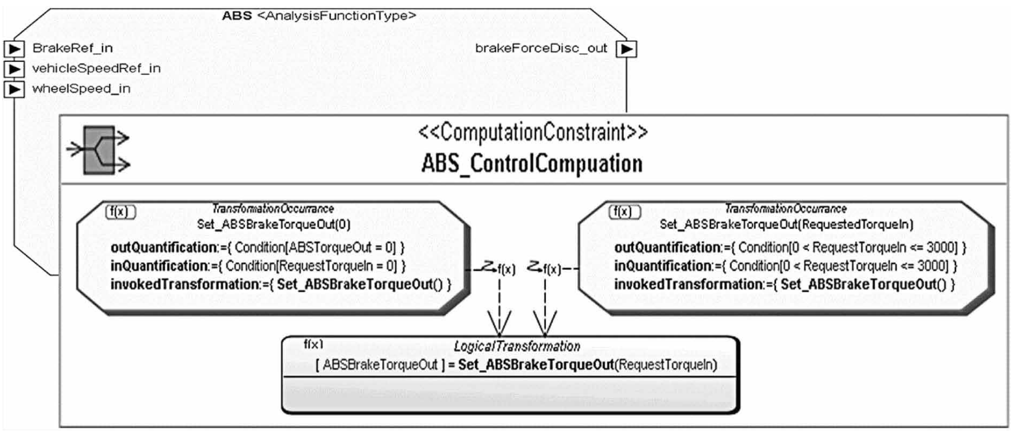


Figure 12. The computation constraint description for an ABS function



6. MODELS AND VERIFICATION IN THE SAFETY LANE

The safety swimlane focuses on the provision of methodology and modeling support allowing all safety related information according to ISO 26262 to be captured and managed seamlessly along with the core system design specification (Chen et al. (2011)). In this section we present the main models prescribed by the EAST-ADL methodology in the safety swimlane. Advanced safety analyses – developed during the MAENAD project – are also briefly presented.

6.1. Dependability Models

Through its Dependability package, EAST-ADL allows a wide range of functional safety related concerns (e.g. hazards, faults/failures, safety requirements) to be declared and structured seamlessly along with the lifecycle of core system development as shown in Figure 14. One key role

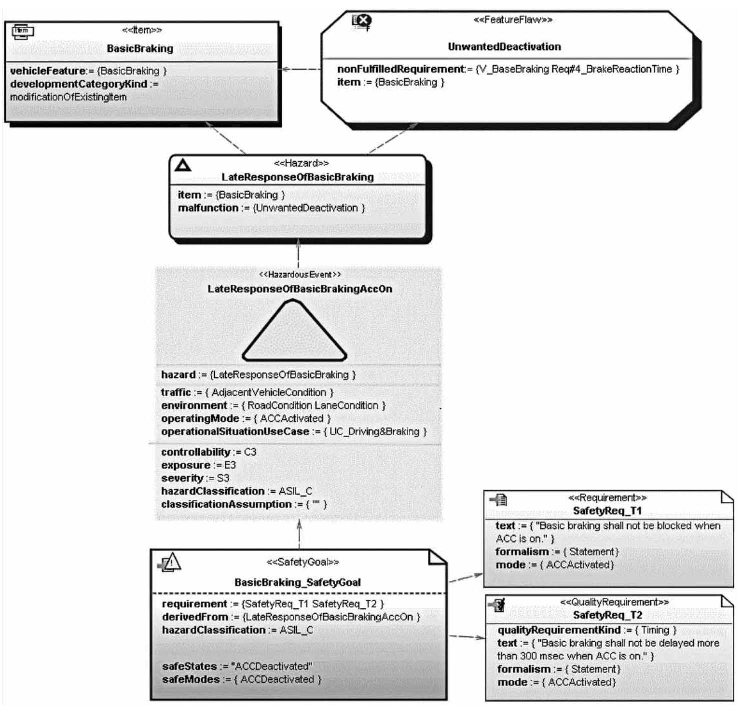
Figure 13. The PROMELA code of the ABS function

```
    /* The ABS function at each wheel satisfies the run-to-completion semantics */
proctype abs(byte id; chan RequestedTorqueIn, VehicleSpeedIn, WheelSpeedIn, ABSBrakeTorqueOut)
{
    byte reqTorq, vehSpd, whlSpd, ABSBrakeTorq; /* Local variables */

    do
    :: trig_abs[id] ? _ ->                /* The execution trigger */
        read_Inport(RequestedTorqueIn, reqTorq);    /* Read all inports and save the values */
        read_Inport(VehicleSpeeddIn, vehSpd);       /* in local variables */
        read_Inport(WheelSpeedIn, whlSpd);
                /* Note: the vehSpd in this process is the estimate from the average */
                /* of the speeds of the 4 wheels. */
        if
        :: vehSpd > ABSVehicleSpeedThreshold &&
            vehSpd - whlSpd*WheelRadius > vehSpd*ABSSlipRateThreshold ->
                                          /* If locked, release the brake */
            ABSBrakeTorq = 0;             /* Set_ABSBrakeTorqueOut(0) */
            assert(0)                     /* Flag an error when a wheel is locked */
        :: else ->      /* If not locked, apply the full torque demand */
            ABSBrakeTorq = reqTorq      /* Set_ABSBrakeTorqueOut(RequestedTorqueIn) */
        fi;

        write_Outport(ABSBrakeTorqueOut, ABSBrakeTorq)      /* Write the outports */
    od
}
```

Figure 14. Dependability model structuring information related to safety goals



of EAST-ADL dependability model is to capture the related system requirements and design information from which the safety requirements are elicited.

Along with the modeling support for the elicitation of safety requirements, EAST-ADL also allows the engineers to precisely defining the related error behaviors for the purposes of safety analysis through explicit error models. These analytical models provide the support for associating the annotations of error descriptions (i.e., faults and error propagations) within the target system. See Figure 15 for a modeling example, where the connection links represent the error propagations due to communication links or allocation relations in the design.

Each block in the error model contains the descriptions of plausible anomalies (*Anomaly*) in terms of faults and failures that a target system entity can have. The ports declare which faults the targeted system entity can receive from its environment and which failures the targeted system entity can propagate to the environment. Such ports are analytical and can be traced to the corresponding communication ports of functions or components.

Within each error model, there is a declaration of error behavior (ErrorBehavior) for relating the declared output failures to the declared faults. The exact formalism could be chosen according to the analysis methods of interest as well as the complexity of error logic. For example, the formalism can be directly based on Boolean logic expression as given in HiP-HOPS. For a state-machine (SM) based definition of error behaviors, the EAST-ADL temporal behavior constraint is used (Chen, Mahmud et al. (2013)). This is shown Figure 16, where the error model description targets a system control function with two internal parts func_a and func_b. The initial state is AB to indicate that the internal parts are both working and the failure of the system is sequence-dependent—i.e., func_a and func_b both need to fail in sequence for the whole system to fail (state FAILED). However, if func_b fails first or alone then the system enters a DEGRADED state. Such a sequence-dependent behavior is not uncommon in safety-critical systems; for example, in simple primary-standby architecture, if a sensor fails and the monitored component (the primary) fails afterwards, then the redundant component cannot be activated. But if the primary fails first or alone, then the system can still function in standby mode.

Figure 15. Error model defining the faults and error propagations of target system hardware and functions

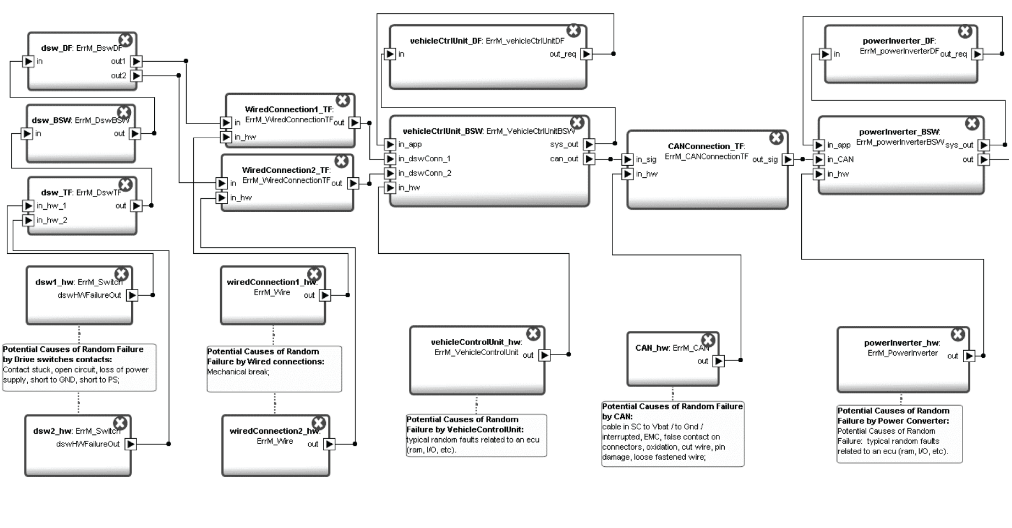
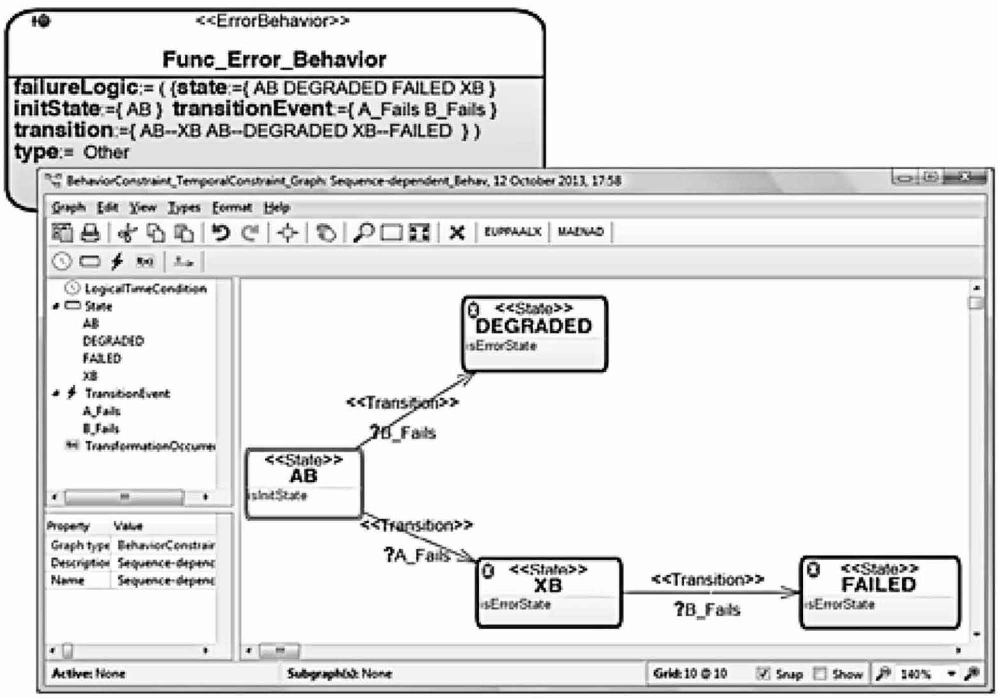


Figure 16. A state-machine based error logic description



6.2. Safety Analysis

There are several well-known approaches to automated safety analysis. Some of these approaches infer the effects of component failures on the overall system by means of model-checking and simulation techniques, such as the approaches described (Bozzano et al. (2003)). These approaches can be computationally expensive due to their inductive nature (i.e. from causes to effects), especially when combinations of failures need to be considered. Instead, a deductive approach (i.e. from effects to causes) can often be more efficient. One example is the HiP-HOPS method, originally described in (Papadopoulos et al (1999)) for a static safety analysis, but recently extended in (Walker et al. (2009)) and (Mahmud et al. (2010, 2012)) for a temporal analysis.

HiP-HOPS starts taking place early in the design lifecycle with exploratory FFA; but can be mainly used after a hierarchical model of the system has been developed. The failure behavior of components is analyzed using a modification of classical FMEA, called the Interface Focused FMEA (IF-FMEA). The application of this technique generates a model of the local failure behavior of the component which is represented as a table. The table provides a list of component failure modes observed at the component outputs. For each component output failure, the causes are determined as a logical combination of internal malfunctions or deviations of the component inputs. An IF-FMEA table records component reactions to failures that are generated by other components. Moreover, the table determines the failure modes that the component itself generates and may propagate to other components. Upon determination of local failure behavior of all components, HiP-HOPS can show how the functional failures (identified in the exploratory FFA) arise from combinations of the low-level component failure modes (identified in the IF-FMEAs).

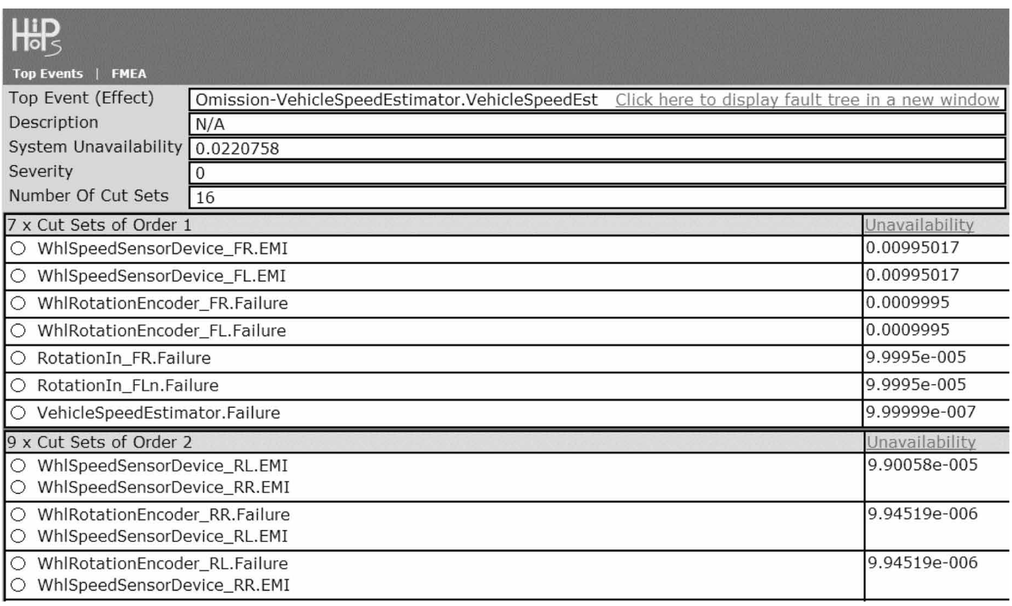
This is done by automatically synthesizing fault trees. A fault tree is generated incrementally by parsing the expressions, which are derived from the IF-FMEA, and encountered during a hierarchical traversal of the system model. The tool automatically performs minimal cut-set analysis and probabilistic calculations on the minimized fault trees to predict the reliability and availability of the system.

The failure annotations required by HiP-HOPS are originally Boolean-based. For example, with an error behavior of the vehicle speed estimator (see Regenerative Braking System in Figure 8) such that omission of output is caused either by an internal failure or by omission of any front wheel speed input or combined omission of rear wheel speed inputs. Furthermore, omission of output from any wheel speed sensor is caused either by an Electro Magnetic Interference (EMI) or by omission of input; and similarly, omission of output from any wheel rotation encoder is caused either by an internal failure or by omission of input. Figure 17 shows some cut sets (calculated by the tool) of the synthesized fault trees shown in Figure 18.

Figure 19 represents an FMEA table produced by the tool to show each failure mode, its further effect and the contributing failure modes.

As for SM-based failure descriptions, these can be basically compiled into fault trees such that each state that represents a system failure becomes the top event of a fault tree. Each branch of that FT represents the conjunction of the events that label a full path (from the initial state to the failure state). A conjunction of events represents one cut set of the fault tree, and if it contains no redundancies, then it is a minimal cut set (MCS). If there is a path with a loop (like in the SM describing an ABS function in Figure 16), then the conjunction of the events which label that path is not minimal and the loop needs to be removed. Therefore, the fault tree expression which corresponds to the error state of the SM of Figure 16 is as follows:

Figure 17. Cut sets displayed by HiP-HOPS for omission of vehicle speed estimator (excerpt)



Top Events | FMEA

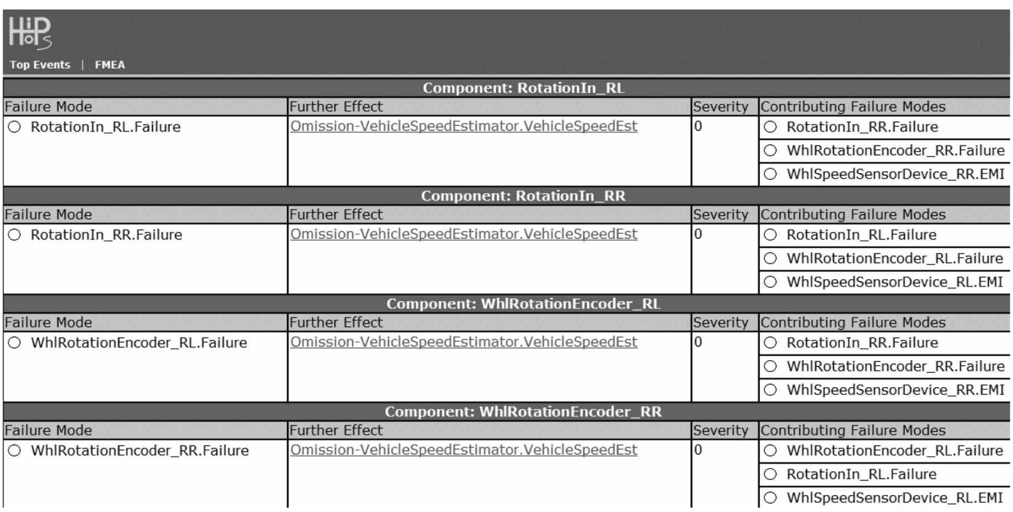
Top Event (Effect)	Omission-VehicleSpeedEstimator.VehicleSpeedEst Click here to display fault tree in a new window
Description	N/A
System Unavailability	0.0220758
Severity	0
Number Of Cut Sets	16

7 x Cut Sets of Order 1	Unavailability
WhlSpeedSensorDevice_FR.EMI	0.00995017
WhlSpeedSensorDevice_FL.EMI	0.00995017
WhlRotationEncoder_FR.Failure	0.0009995
WhlRotationEncoder_FL.Failure	0.0009995
RotationIn_FR.Failure	9.9995e-005
RotationIn_FLn.Failure	9.9995e-005
VehicleSpeedEstimator.Failure	9.99999e-007

9 x Cut Sets of Order 2	Unavailability
WhlSpeedSensorDevice_RL.EMI WhlSpeedSensorDevice_RR.EMI	9.90058e-005
WhlRotationEncoder_RR.Failure WhlSpeedSensorDevice_RL.EMI	9.94519e-006
WhlRotationEncoder_RL.Failure WhlSpeedSensorDevice_RR.EMI	9.94519e-006

Figure 18. The HiP-HOPS synthesized fault trees for omission of vehicle speed estimator

Figure 19. FMEA displayed by HiP-HOPS (excerpt)



Failure Mode	Further Effect	Severity	Contributing Failure Modes
Component: RotationIn_RL			
RotationIn_RL.Failure	Omission-VehicleSpeedEstimator.VehicleSpeedEst	0	RotationIn_RR.Failure
			WhlRotationEncoder_RR.Failure
			WhlSpeedSensorDevice_RR.EMI
Component: RotationIn_RR			
RotationIn_RR.Failure	Omission-VehicleSpeedEstimator.VehicleSpeedEst	0	RotationIn_RL.Failure
			WhlRotationEncoder_RL.Failure
			WhlSpeedSensorDevice_RL.EMI
Component: WhlRotationEncoder_RL			
WhlRotationEncoder_RL.Failure	Omission-VehicleSpeedEstimator.VehicleSpeedEst	0	RotationIn_RR.Failure
			WhlRotationEncoder_RR.Failure
			WhlSpeedSensorDevice_RR.EMI
Component: WhlRotationEncoder_RR			
WhlRotationEncoder_RR.Failure	Omission-VehicleSpeedEstimator.VehicleSpeedEst	0	WhlRotationEncoder_RL.Failure
			RotationIn_RL.Failure
			WhlSpeedSensorDevice_RL.EMI

Error = SlipRateOverstep + SlipRateOverstep . [(VehicleSpeedInGreaterThanABSThreshold) . (SlipRateGreaterThanThreshold)]

where '+' and '.' represent the Boolean 'OR' and 'AND' respectively. The events 'SlipRateOverstep' corresponds to the condition:

[not(0<=SlipRate <= 1)], 'VehicleSpeedInGreaterThanABSThreshold'

corresponds to the condition:

[VehicleSpeedIn>ABSVehicleThreshold]

and:

'SlipRateGreaterThanThreshold'

corresponds to the condition:

[SlipRateThreshold <=SlipRate <=1]

The failure expression of the corresponding fault tree can be simply minimized to:

Error = SlipRateOverstep

Concerning sequence-dependent failures, however, Mahmud et al., (2010, 2012) proposed an approach which extends the HiP-HOPS method for dynamic analysis by generating and synthesizing Pandora Temporal Fault Trees (TFTs) from the state machines. On the one hand, Pandora is designed for temporal qualitative analysis (Walker et al., 2009); it is equipped with temporal laws which are very useful in enabling the minimization of the TFTs. On the other hand, the proposed conversion approach generates fault trees extended only with the necessary temporal information—i.e., it detects and preserves the significance of the sequencing of faults during the conversion and all along the logical analysis. Thus, the approach remains as close as possible within the flexibility and ease-of use of the conventional fault trees.

To correctly capture the sequence-dependent failures, the approach mainly uses the Pandora Priority-AND gate (PAND, symbol '<') and the Priority-OR gate (POR, symbol '|'). A PAND gate represents a sequence of events typically from left to right, while a POR gate models a priority situation where one event (leftmost) must occur first and other events may or may not occur subsequently. To support a qualitative analysis, Pandora defines a set of temporal laws for identifying and removing redundant sequences of events. For example, $(A|B).\ B \Leftrightarrow A<B$, where the left hand side conjunction expression (A must occur first, but B must also occur) is equivalent to the right hand side expression (A occurs before B, both events have to occur).

The conversion algorithm which generates (temporal) FTs from the SMs performs backward traversals from each final state that represent a system (or a component) failure to the initial state. At each join state (a state at which paths diverge) during the traversal, if there is a common event with another divergent path, then the FT becomes temporal using POR. For example, the conversion of the SM of Fifure 16 generates temporal rather conventional FTs since the two divergent paths at the join state (the initial state in this case) share a common event, and hence:

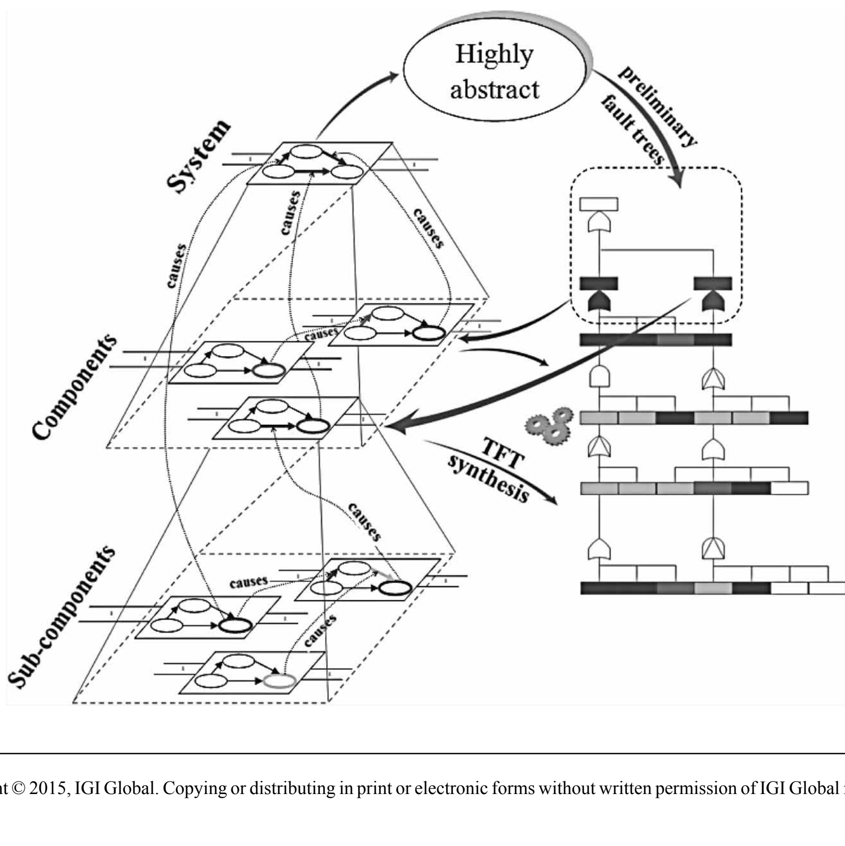
Degraded = B_Fails|A_Fails
Failed = B_Fails . A_Fails|B_Fails

The failure expression which corresponds to the Failed state is equivalent to A_Fails<B_Fails (using the temporal law described previously). The other expression specifies that only B_Fails needs to occurs, A_Fails may or may not occur afterwards.

The order in which failure events occur is captured in a probabilistically sound way, initially using POR only, then together with PAND, depending on the temporal law used. For example, in the case of exponential failure distribution for the basic events, the solution of the corresponding Markov model gives the same probabilistic results associated with the states FAILED and DEGARDED as those given in (Fussel *et al.*, (1976)) and in (Merle *et al.*, (2010)) for the PAND and POR-like respectively, see (Mahmud (2012)) for a detailed comparison.

This conversion approach is used as part of a novel compositional method SAFORA1, which has been developed in (Mahmud 2012) to increase scalability. Safora (Figure 20) is a top-down synthesis of (temporal) FTs generated during backward traversals of component SMs. The synthesis starts from a highly abstract SM describing the monolithic behavior of the system (top level SM in the hierarchy), and from which we generate preliminary fault trees. Then we start expanding these fault trees from the SMs local to the components. The fault trees get minimised when appropriate during synthesis. A final analysis takes place for each system failure

Figure 20. Overview of the Safora method



(temporal) FT which is completely synthesized— i.e., when no more expandable or non-atomic events remain in the fault tree.

7. MODELS AND VERIFICATION IN THE TIMING LANE

7.1. Timing Models

Timing modelling in EAST-ADL results from the work done in TIMMO project (TIMMO (2007)), which produced a dedicated language called TADL and from Timmo2Use, which produced a second version of the language called TADL2 (TIMMO2USE (2012)). TADL concepts were integrated in the course of the ATESST2 project in the EAST-ADL language. TADL2 concepts will be integrated in EAST-ADL during the third year of the MAENAD project, but in the current language version (2.1.10) TADL2 has not been integrated yet. For this reason we will refer to TADL in the reminder of the section.

EAST-ADL divides timing information into timing requirements and timing properties, where the actual timing properties of a solution must satisfy the specified timing requirements. EAST-ADL currently focuses on modelling of timing requirements on the functional abstraction levels of the architecture description language. The implementation level, i.e. AUTOSAR, is currently not explicitly considered, but it is expected that the information can be modelled in a similar way. The same holds for timing properties on both the functional abstraction levels and the implementation level.

Timing information on the functional abstraction levels is perceived as follows: timing requirements for a function can be captured on logical abstraction levels where no concrete hardware is yet available. This allows the specification of general timing requirements such as end-to-end delays from sensors to actuators regardless of how the final solution is built. This reflects the notion that a purely logical functional specification is not concerned with its technical realisation, i.e. how many ECUs or bus systems are ultimately involved. What matters from the functional perspective are the recurring end-to-end delays of a control application, which need to keep pace with the real plant. Specifying timing requirements on the implementation level might be both too late in the development process and rather difficult because of language complexity (e.g. AUTOSAR) and the number of details on this level. However, it sounds more feasible to refine the timing requirements on the implementation level from the timing requirements of the design functional model. Indeed AUTOSAR Timing Extensions (TIMEX) (AUTOSAR (2015)) allows timing specifications in the AUTOSAR implementation level. It is important to note that the short semantic distance between TADL and TIMEX facilitates this refinement of timing requirements.

Timing properties are characteristics of a solution, e.g. actual response times, and should be reflected in the functional abstraction levels.

In EAST-ADL, timing requirements are divided into various kinds of delays (or latencies) for single time-consuming modelling entities as well as specific requirements for temporal synchronisation of input or output data. The delays are either end-to-end delays, which are subject to segmentation along the functional decomposition track (e.g. end-to-end delay for a top-level function), or the delays form part of an end-to-end timing chain, and thus constitute segments of such an end-to-end timing chain. More precisely, EAST-ADL Timing concepts are based on:

- **Events:** Relate to EAST-ADL entities and depict observables: e.g. data arriving on a port, triggering of function execution, etc.;

- **Event Chains:** Bind together events to establish sequences/relations between events e.g. to capture a complete end-to-end flow requirement between data sent by a sensor to output by an actuator;
- **Constraints:** Put temporal constraints on sets of events or on event chains, e.g. deadlines to be met, expected delays, patterns of data arrival, synchronisation of outputs on a set of ports, etc. Note that these constraints represent measured/computed properties when attached to a VVactualOutcome, concept coming from Verification&Validation constructs of EAST-ADL.

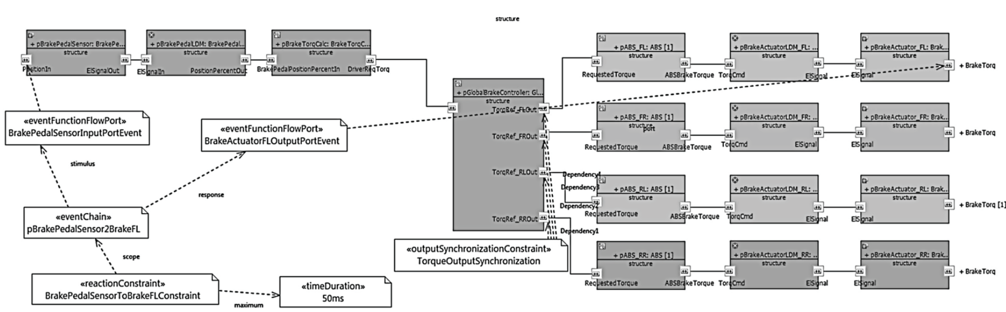
One example is featured in Figure 21 in which an excerpt of the functional design architecture of the Regenerative braking is shown. The timing model here must capture maximal end-to-end delays that must hold from the instant of time in which data arrives at the pedal sensor to the instant of time in which the output is produced by brakes actuators. The figure shows how to model this maximal delay for the output produced at the brake actuator of the front left wheel. To identify the instants in which data is available at input/output ports, two events are here defined. The first event ('BrakePedalSensorInputPortEvent') represents the data arrival at the input port of the brakeSensor function, while the second one (BrakeActuatorFLOutputPortEvent') represents the data availability at the output port of the actuator. These two events represent respectively the stimulus and the response of the event chain. The event chain is subject to a reactionConstraint whose maximal tolerated value is 50ms.

Other common constraints/properties characterizing timing models are execution timing constraints. The difference between a reaction constraint and an execution timing constraint is appreciable at design level in which hardware allocation is specified. Execution time constraint concerns the time needed by the function to be executed in isolation. It can be considered as a minimal bound for the function to be executed. A reaction constraint is a requirement that set the upper bound for the function to complete its execution; it must be greater than the execution time.

Timing constraints on data availability can be specified as well. For instance, an output synchronization constraint expresses a timing constraint on the output synchronization among the set of response events. On Figure 21, for readability concern we have omitted to display flow port events and event chains that are constrained by the output synchronization constraint 'TorqueOutputSynchronization'.

The functional model of the design phase depicted in Figure 21 specifies a graph of functions. Activation semantics is added to that functional graph through events definition and timing constraints among these events. This gives a timed partial order of execution for functions which is needed to make timing analysis. The following subsection shows timing analysis made on this BBW timed functional architecture model.

Figure 21. Timing model for the regenerative braking



7.2. Timing Analysis

The validation of timing properties during the development is a continuous process. Since refinements are top down, deadlines can only be validated under assumptions abstracting the lower-level details, such as the worst-case execution times (WCETs) of functions. Unless the implementation of functions is reused from previous systems, precise knowledge of WCETs of functions is not available before code implementation in the design process. To define a system architecture supporting functions with time constraints and provide the specification of its components to the suppliers, automotive developers and domain experts propose a time budgeting activity as part of the development methodology. The system integrator specifies and assigns time budgets. These time budgets are constraints on worst-case execution time that must be respected by the suppliers delivering the functions implementation. If all the delivered functions fulfill their local constraints, the end-to-end deadlines are satisfied. In the meantime, based on the budget assumptions, the system integrator can evaluate and optimize a functional architecture design.

The TIMMO-2-USE project (TIMMO2USE (2012)) discusses the need for time budgeting in the context of the process stages dedicated to the refinement of the system architecture. The project deliverables discuss a set of guidelines for budgeting the worst-case response times (WCRTs), based on the designer experience and do not provide a specific algorithm.

The concept of time budgeting in the integration of automotive systems is among the research topics of the ALL TIMES project (ALLTIMES (2009)). The approach proposed in the project deliverables takes as an input an already deployed architecture, i.e. the decision about functions allocation to hardware resources is already made. However, since the objective here is to determine time budgets (which are not determined yet), the function allocation decision cannot exploit time budget information of functions. On the other hand a functional allocation is only possible if the hardware resources have enough capacity to execute the function: a very time-consuming function would probably need a more powerful resource in terms of resource capacity. This is a typical egg-and-hen problem, in which functional allocation depends on time budgets and time budgets estimation depends on functional allocation.

We propose to break this tie by an interleaved approach in which we iteratively solve the time budget estimation problem and the functional allocation problem. Note that the overall goal is to obtain a design-level architecture which is schedulable: in which end-to-end deadlines on functional chains are respected despite delays induced by execution and communication mechanisms at software and hardware level.

In the following we will proceed bottom-up, discussing design phase activities and then analysis phase activities. We firstly introduce the schedulability notion (pertaining to the design phase), then we propose an algorithm to solving respectively the functional allocation problem (design phase) and the time budget estimation problem (analysis phase). Finally we will discuss the interleaved approach.

7.2.1. Schedulability Estimation

Schedulability estimation at design level is able to detect resources overload situations that will prevent the schedulability of the system, computing bus and processor utilization. Following formulas are used for the computation of processors utilization (on the left hand side) and buses utilization (on the right hand side). In these formulas:

- s_i is the i^{th} signal of the system;
- f_i is the i^{th} function of the system;

- β_i is the i[th] communication bus;
- c_i is the i[th] processor;
- The → relation indicates that the left term (function/signal) is allocated on the right term (processor/bus);
- $\omega_{i,j}$ represents the WCET/WCTT of the function/signal i on the processor/bus j;
- P_i is the activation period of the function/signal i:

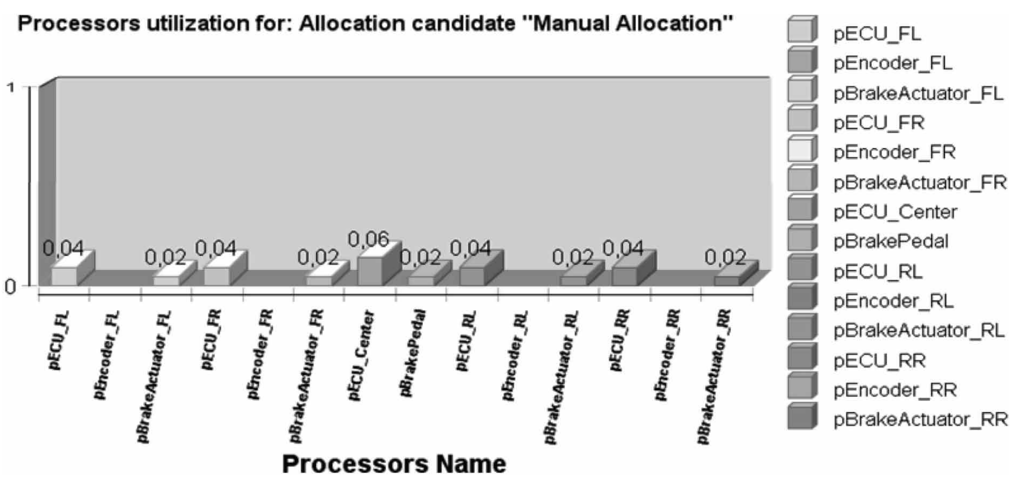
$$U_{c_i} = \sum_{f_i \to c_i} \frac{\omega_{f_i,c_i}}{P_{f_i}}$$

$$U_{\beta_i} = \sum_{s_i \to \beta_i} \frac{\omega_{s_i,\beta_i}}{P_{s_i}}$$

Based on this formulation for the Regenerative Braking we obtained a bus utilization of 36%. For processor utilization we obtained the results shown in Figure 22 provided by the Qompass tool (MAENAD (2013)).

Even though, bus and processor utilization are key indicators for system schedulability, the respect of maximal resource utilization capacities does not imply that the system is schedulable. System schedulability can be verified only by computing end-to-end latency of each path from sensors to actuators and comparing latencies against end-to-end deadlines. So-called response-time analysis is usually employed to compute end-to-end latency. Response time analysis, however, considers tasks chains instead of function chains. Functions and tasks, however, are not the same thing: a task is the software resource that executes functions. Response time is very sensitive to the way in which functions are partitioned into tasks. For this reason response-time analysis is usually reserved to the implementation phase. During the MAENAD project, however, we investigated solutions to anticipate response-time analysis to the design phase. The idea is to make an "optimal" assumption about how functions are partitioned into tasks, by running an

Figure 22. Resources utilization for a manual allocation of functions to ECUs



optimization algorithm solving the partioning problem. The output given by this optimization algorithm can be then used to estimate end-to-end responses. As a result of this investigation, three different advanced optimization approaches able to provide a prediction on latency of end-to-end chains at design level and to use this information to solve the functional allocation problem as detailed in the next section.

7.2.2. Functional Allocation

During the MAENAD project we elaborated three approaches to solve the functional allocation problem subject to schedulabilty constraints. The first approach lies in computing latency assuming the best partitioning of functions to tasks (from now on called simply partitioning) and best priority assignment to tasks (from now on called scheduling) with respect to latency minimization. Operationally, this means: to start from a manual allocation of functions to nodes (available at design level), to find a partitioning and scheduling that minimize latency via an optimization method, and then to output minimal latency as the latency for the initial manual allocation. The other two approaches aims at improving the first approach treating as well the allocation problem (functions to nodes) in the optimization loop. RTCSA (Mehiaoui et al (2012)) is a two-stage optimization approach. First stage of this approach deals with allocation optimization with respect to resource utilization. Allocation found in the first stage is given as input to the second stage that deals with partitioning and scheduling optimization with respect to latency. LCTES (Mehiaoui et al (2013)) is an improvement of RTCSA. Concretely, like RTCSA an optimized allocation is found and an optimized partitioning and scheduling is found for this allocation, but here the allocation is optimized with respect to latency already in the first stage, by combining two classical optimization strategies: divide and conquer and iterative improvement. Divided-and-conquer consists in dividing the allocation, partitioning and scheduling (PPS) problem in two sub-problems solved in cascade, in which allocation is solved first (PP stage), and then partitioning and scheduling (PS stage). In both sub-problems, we minimize the latency. Iterative improvement is used to move towards the optimum. Figure 23 shows the main algorithmic steps of the approach.

Each iteration starts from a PP optimization with an initial (valid) configuration for PPS as an input. PP provides the allocation of functions/signals to nodes/buses in an implicit manner. Namely, during this stage, tasks and messages are allocated on nodes/buses; however, knowing the partitioning of functions/signals at this stage, their allocation can be derived. Next, the PS stage tries to find a new partitioning and scheduling solution that improves the solution found in the PP stage.

The inner loop tries to find an optimal system configuration by applying iteratively an optimization sequence until convergence (two successive solutions are the same). Depending on the selection of the initial configuration, the PP+PS solution may be a local optimum. To move away from local minima, the outer loop selects random initial configurations for expanding the exploration space.

Based on an MILP (Mixed Integer Linear Programming) formulation of these stages this algorithm has been implemented in Qompass tool (MAENAD (2013)). Figure 24 and Figure 25 give formulas for response time, *computed on chains of functions and signals*. These formulas are an adaptation of traditional formulas for chains of tasks and messages, assuming pre-emptive tasks and non-preemptive messages. These formulas are used to set the objective function and the latency constraint of the MILP formulation (not shown here for space reasons).

Within these formulas in addition to the previously described terms the following ones mean:

Figure 23. Overview of the LCTES optimization approach

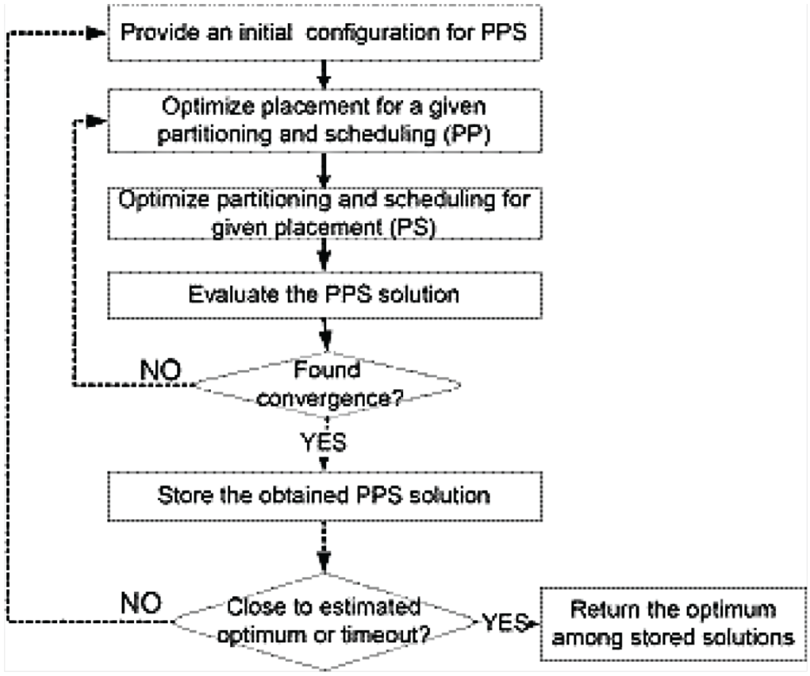


Figure 24. Formulation of functions response time computation

$$R_{f_i} = J_{f_i} + W_{f_i}$$

$$J_{f_i} = \max_{\substack{f_j \in F: \\ \tau_{f_i} = \tau_{f_j}}} \left[\max_{\substack{s_i \in \Phi: \\ f_j \in rec(s_i)}} [R_{s_i}] \right]$$

$$W_{f_i} = \omega_{f_i,c_j} + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k}}} \omega_{f_k,c_j} + \sum_{f_k \in hp(f_i)} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil \cdot \omega_{f_k,c_j}$$

- R_i is the Worst-Case Response Time (WCRT) of function/signal *i*;
- J_i is the jitter of the activation of function/signal *i*;
- W_i is the completion time of function/signal *i*;
- *hp(i)* is the set of functions/signals with a priority greater than the priority of function/signal *i* that are allocated on the same processor/bus;
- B_i is the blocking time of function/signal *i*;
- Φ is the set of signals of the system;
- *F* is the set of functions of the system;

Figure 25. Formulation of the response time computation for signals

$$R_{s_i} = J_{s_i} + \omega_{s_i,\beta_j} + \sum_{\substack{s_k \in \{\Phi \setminus s_i\}: \\ \mu_{s_i} = \mu_{s_k}}} \omega_{s_k,\beta_j} + W_{s_i}$$

$$J_{s_i} = \begin{cases} \max\limits_{\substack{s_j \in \Phi: \\ \mu_{s_i} = \mu_{s_j}}} [R_{snd_{s_j}}] & s_i \text{ is transmitted between processors} \\ R_{snd_{s_i}} & s_i \text{ is transmitted between tasks} \\ J_{snd_{s_i}} & s_i \text{ is transmitted within a task} \end{cases}$$

$$W_{s_i} = B_{s_i} + \sum_{s_k \in hp(s_i)} \left\lceil \frac{W_{s_i} + J_{s_k}}{P_{s_k}} \right\rceil \omega_{s_k,\beta_j}$$

- τ_{fi} is the task on which function f_i is allocated;
- μ_{si} is the message transmitting signal s_i;
- snd_{si} is the function sending the signal *i*;
- $rec(s_i)$ is the set of functions that receives the signal *i*.

The global WCRT for a function f_i (Figure 24) is computed as the sum of the jitter (representing the delay between the instant of the external event arrival, the instant of function activation) and the function completion time (representing the delay between the instant of its activation and the instant of its termination).

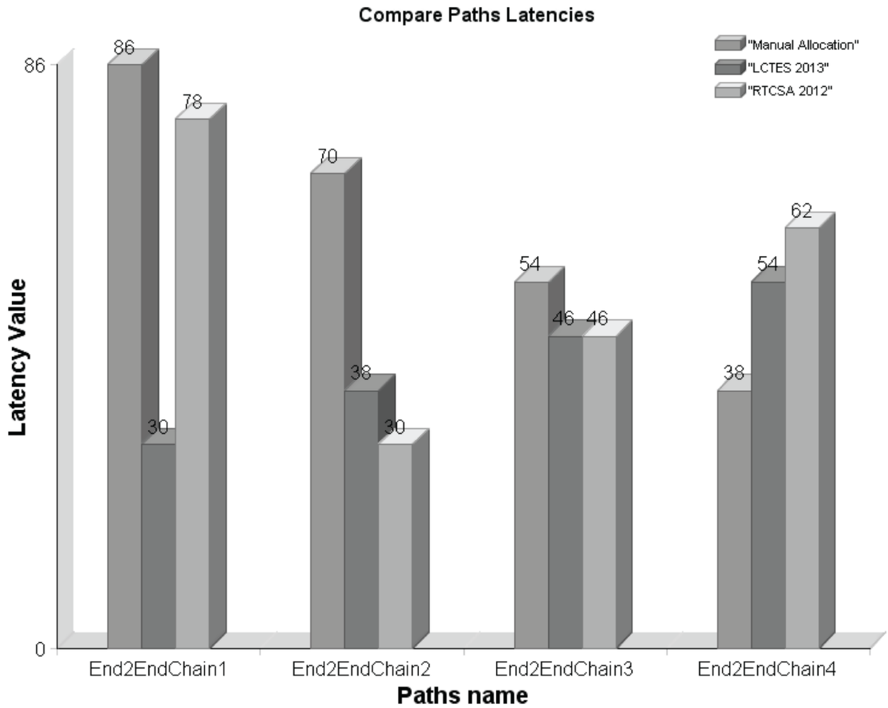
The global WCRT of a signal (Figure 25) is equal to the sum of its jitter, its worst-case transmission time, the worst-case transmission time of all signals that are allocated to the same message, and the longest duration of waiting in the queue before its election for a transmission on the bus.

Figure 26 gives end-to-end chains latencies for the three approaches and allows their comparison. We can notice that latencies of the manual allocation have been improved by the two other approaches (LCTES 2013, and RTCSA 2012). We can also notice that LCTES 2013 approach has improved latencies of RTCSA 2012 approach.

Figure 27 gives processors and bus utilizations obtained with the three approaches. We can observe that in terms of utilization ‘Manual Allocation’ and ‘RTCSA 2012’ are comparable, whereas ‘LCTES 2013’ improves bus utilization which is the cause of the global latencies improvement observed in Figure 23. This is due to the fact that distant communications are source of a large latencies value.

As can be seen results of these advanced analyses give hints on how to improve core models (allocation in particular). In the regenerative brake example, the new allocation found with the LCTES method can be part of the recommendations for the core swimlane in order to improve global latency.

Figure 26. Comparison of end-to-end chains latencies for the three approaches



7.2.3. Time Budget Estimation

In the previous section we presented an optimization approach to solve the functional allocation problem under schedulability constraints. This approach is supposed to be applied during the design phase. The approach takes as input WCETs of functions, however this information might be missing for new functions (never implemented before). To cope with this issue several approaches suggest to parform an time budget estimation at analysis level, to provide time budgets for functions missing WCET.

In this section we present a time budgeting algorithm (Wozniak, E. et al (2014)). Note that this first algorithm takes as input a functional allocation (as done in the ALL TIMES project). Before giving details of the algorithm, let us first introduce concept notations used for its definition (see Table 5).

This work considers an optimization metric expressing the relaxation of time budgets within the end-to-end deadline constraints. The function $f_{tb}\left(TBA\right)$ in equation bellow requires as an input the set TBA of functions with the specific valuation for their time budgets. It is defined as the minimum time budget value for all functions in RB normalized with respect to the target range $\left(tb_{r_k}^{m}, tb_{r_k}^{M}\right)$. The optimization objective is to maximize $f_{tb}\left(TBA\right)$, or equivalently, to maximize the minimum normalized time budget among functions in RB :

Figure 27. Comparison of: (a) Processors; and (b) Bus utilization for the three approaches

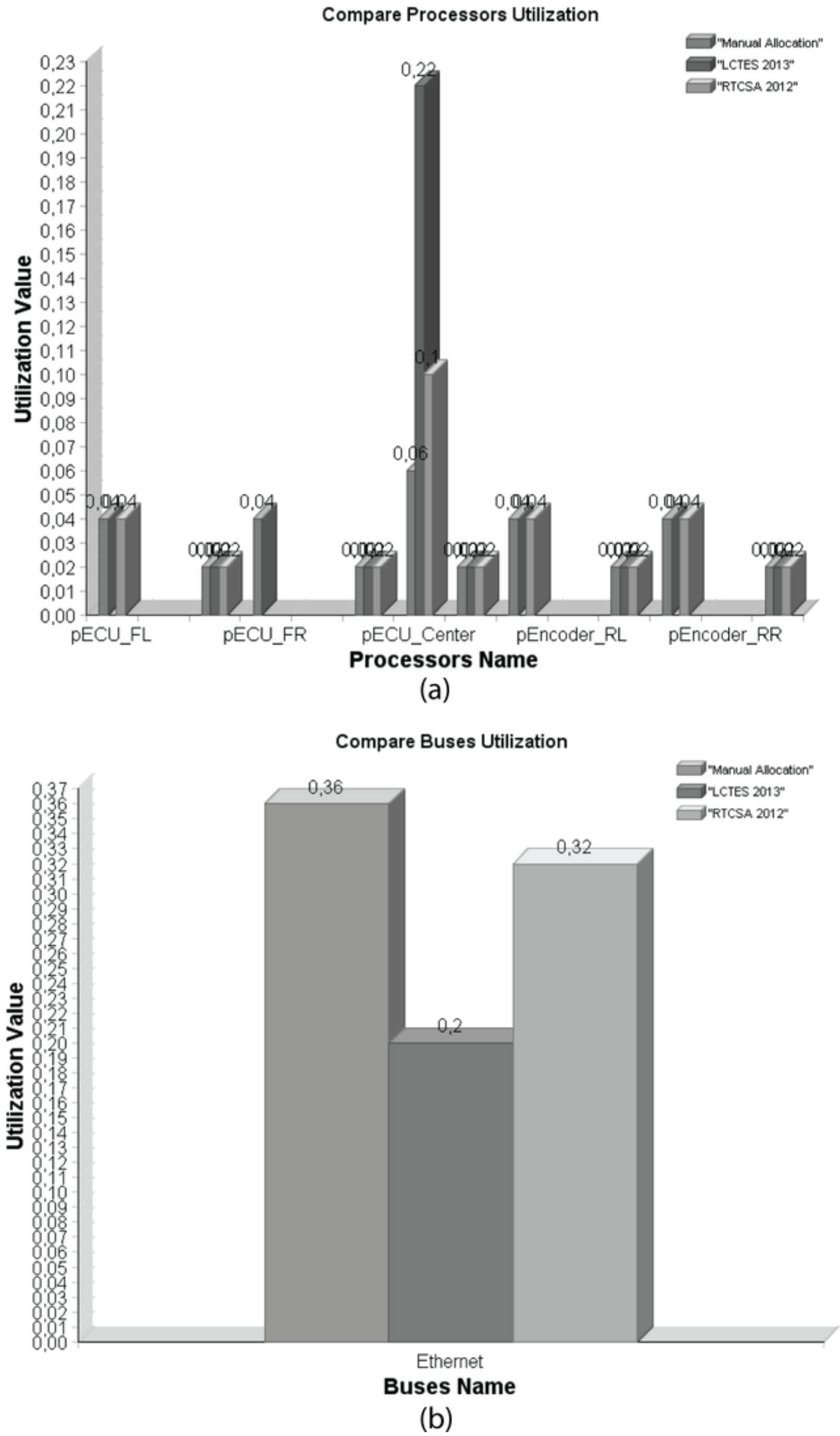


Table 5. Definitions of each concept notation

Concept	Definition
tb_{r_i}	Time budget for the function r_i. Time budget in our case represents the constraint imposed on the WCET of a function.
$tb_{r_i}^m$	Minimal time budget for the function r_i. The designer has the option to provide a minimum value for tb_{r_i} as $tb_{r_i}^m$. Its intuitive meaning is a preliminary evaluation of the minimum required execution time for the functionality, based on the experience of the designer. If it is not specified, then $tb_{r_i}^m = 0$.
$tb_{r_i}^M$	Maximal time budget for the function r_i. Its intuitive meaning is a preliminary evaluation of the maximal execution time for the functionality, based on the experience of the designer. If not explicitly set, it is assigned with the period of the end-to-end chain to which r_i belongs.
RB	Set of functions for which a budget assignment must be provided.
TBA	This set represents specific Time Budget values Assignment, i.e. the valuation. Namely each element is a value assigned for a corresponding time budget tb_{r_j}.

$$f_{tb}\left(TBA\right) = \min_{r_k \in RB} \frac{tb_{r_k} - tb_{r_k}^m}{tb_{r_k}^M - tb_{r_k}^m}$$

Algorithm 1 is executed for a deployment configuration to compute the corresponding optimum set of time budgets based on which value for the metric function $f_{tb}\left(TBA\right)$ can be calculated. The time budgeting algorithm has four inputs: Ψ (functional allocation), RB, ϵ and M_{RB}. ϵ is the maximum error on the computed budgets that controls the terminating condition (line 15). The lower is the value of ϵ, the more accurate are the time budgets, and the larger is the runtime of the algorithm. M_{RB} is a set of upper bounds on the functions budgets computed for a specific functional allocation Ψ, where $M_{RB}\left(i\right)$ is the maximum value for r_i. The values in M_{RB} are computed before running Algorithm 1, based on the end-to-end deadlines, utilization bounds, and the constraints $tb_{r_i}^m$ and $tb_{r_i}^M$.

Algorithm 1 tries to relax the time budgets for all the functions in RB according to the metric $f_{tb}\left(TBA\right)$ using a binary search algorithm. The upper bound values are tried first, giving the maximum possible value of $f_{tb}\left(TBA\right)$ (lines 6-8). If the corresponding configuration is schedulable, it is returned as the optimum value (line 9). If not, then the algorithm assigns to each r_j in RB a budget value that is the medium value between the minimum $tb_{r_j}^m$ and the upper bound $M_{RB}\left(i\right)$ (lines 11-13).

From this point on, Algorithm 1 continues by iteratively reducing the range of the time budgets, defined as $\left[Ltb_{r_j}, Utb_{r_j}\right]$ for function r_j. The algorithm works as a binary search. In each iteration, if the current budget values, at the midpoint between the upper and lower bounds result in a schedulable solution, the upper bound Utb_{r_j} remains the same, and the lower bound Ltb_{r_j} is updated to be midpoint (line 20), and the range is reduced to be half of the size. If the current settings result in a non schedulable solution, it means that the time budget value is too large, and the next iteration will search within the lower half of the range (line 22).

7.2.4. Interleaving Time Budget Estimation and Functional Allocation

In (Wozniak, E. et al (2014)) we proposed an interleaved approach in order to solve budget estimation and functional allocation in a coordinated manner. The proposed approach goes through two stages: in the first stage functional allocation is solved as shown in Section 7.2.2

Algorithm 1. Algorithm for the one-dimensional binary search

```
Requires: Ψ, RB, ε, M_RB
1:  forall r_j ∈ RB do
2:      Utb_{r_j} = M_RB(j)
3:      Ltb_{r_j} = tb^m_{r_j}
4:  end forall
5:  if isSchedulable(Ψ, M_RB) then
6:      forall r_j ∈ RB do
7:          TBA.set(r_j, M_RB(j))
8:      end forall
9:      return TBA
10: else
11:     forall r_j ∈ RB do
12:         tb_{r_j} = (Utb_{r_j} − Ltb_{r_j})/2
13:     end forall
14: end if
15: while ∃_{r_j∈RB} Utb_{r_j} − Ltb_{r_j} ≥ ε do
16:     forall r_j ∈ RB do
17:         TBA.set(r_j, tb_{r_j})
18:     end forall
19:     if isSchedulable(Ψ, TBA) then
20:         forall r_j ∈ RB do
21:             Ltb_{r_j} = tb_{r_j}
22:         end forall
23:     else
24:         forall r_j ∈ TB_r do
25:             Utb_{r_j} = tb_{r_j}
26:         end forall
27:     end if
28:     tb_{r_j} = Utb_{r_j} − (Utb_{r_j} − Ltb_{r_j})/2
29: end while
30: return TBA
```

The second stage tries to optimize the time budgeting only. The two stages are computed sequentially inside a loop until there is no further improvement. The staged algorithm implements an iterative improvement strategy that is in essence a local search. Starting from an initial solution, the current best solution is tentatively improved in the iterations of an inner cycle that includes the two optimization stages. If at any iteration, the two stages fail to produce a better result, the algorithm terminates and returns the best solution found until that point. The rationale for this choice is that we do not want the algorithm (a local search) to end prematurely and we try to ease schedulability (and provide for maximum allocation freedom) as much as possible in the first step. Details of the algorithm can be found in (Wozniak, E. et al (2014)).

8. MULTI-OBJECTIVE OPTIMISATION

The current challenge in the field of research is not only the exploration of additional use cases for software applications, but also the optimization of existing and future system architectures taking a multitude of different quality attributes, i.e. design objectives, into consideration (Walker et al. (2013)). This challenge is complicated by the fact that many real-world objectives conflict with each other, which necessitates a mechanism for trade-off resolution.

The resulting trade-off analysis is a hard combinatorial problem, which can often not be resolved by hand for all but the most trivial cases. In absence of adequate automated solutions, industry standard often relies on manual decision making when it comes to finding optimal or near-optimal configurations for variant-rich architectures. Due to the complex interdependencies and decision ramifications in sufficiently large design spaces, it is reasonable to assume that a large number of currently employed system designs in the automotive domain are based on suboptimal configuration decisions.

In order to establish a robust foundation for automated optimization approaches, a system's information model has to facilitate a high degree of expressiveness in regard to its variability management capabilities and its support of relevant quality attribute data. The utilization of a domain-specific ADL like EAST-ADL allows for model-based analyses on an information model of ample expressiveness comprised in a single monolithic model structure (Blom et al. (2013)). A comprehensively engineered EAST-ADL model can provide all relevant information necessary for an optimization analysis; including the system's variability information and a variety of data eligible for the assessment of quality attributes.

To make use of such information repositories for the purpose of architecture optimization, our goal is to create an approach for generating multi-objective programs from variant-rich EAST-ADL models. This can be realized by representing quality attribute data as objective functions and variability information as program constraints. We are also taking care in creating a heuristic variability interpretation process, which allows for product-line-aware optimization analyses, i.e. the possibility of family-based optimization of entire product lines in one sweep, as opposed to the optimization of individual products (Thüm et al. (2012)).

In this section, we provide a summary of our ongoing work on the subject of generating multi-objective programs from variability information and quality attribute data of given EAST-ADL models and showcase our progress in regard to different aspects of the intended implementation.

8.1. EAST-ADL Variability Language Concepts

Variability modelling in EAST-ADL can generally be divided into two different categories. One category, variability on the Vehicle Level, is represented in the form of cardinality based feature models (CBFM) (Czarnecki et al. (2005)). Its purpose is to establish an abstract view of the system's variable content, including the interdependencies between variable elements. This view offers a distinction between a technical perspective and customer-specific perspectives, but does not disclose any particulars about implementation details (please refer to 5.2, especially page 12*f*). The other category, variability on the artifact levels, can be represented as part of the systems FAA, FDA, and HDA models by means of both CBFM and concrete variation points. This view is concerned with the actual technical implementation of variable content, by defining the relationships and interdependencies between variable architecture entities, i.e. actual functional components or hardware elements on the artifact levels of the system architecture.

In order to relate variability information of different feature models, as well as to allow for logical bindings from feature models to variation points on the artifact level, the language also supports configuration modelling by means of configuration links. These links contain atomized rules – so called configuration decisions – for the configuration of target model entities based on a given configuration of source model entities. Configuration links can be used to relay configuration information across EAST-ADL layers; but also in a more local context, for creating bindings of the internal variability of a container element with feature model representations of that internal variability – a so called *public feature model*. The application of these variability concepts facilitates the use of *compositional variability management* (CVM) as a systematic language concept of the EAST-ADL language (Reiser et al. (2009)).

8.2. Product Line Variability versus Architecture Design Space

In addition to the different variability description methods described in section 3.2, a distinction between product-line-oriented variability (Metzger et al. (2007)) and the system's architecture design space, also called *architectural degrees of freedom* (Aldeida et al. (2013)), is essential for realizing a product-line-aware architecture optimization approach. Table 6 gives a summary of the conceptual differences between these two kinds of variability.

The intended output of our proposed optimization approach is a product line with optimal architectural decisions, as opposed to an optimally configured product based on the given product line. It is therefore necessary to establish a way to differentiate between the two types of variability, in order to be able to mark product-line-oriented variations as restricted for optimization. The approach presented in this paper achieves this distinction by tracing the propagation of individual variation points through the different layers of the given model.

In case of product line variability, a regarded variation point can always be traced all the way up to the features on the model's Vehicle Level by means of configuration decision connections. The features represent product line variations and therefore allow for configuration of variation points located on underlying abstraction levels, i.e. all variability that ultimately is rooted on Vehicle Level is product line variability.

If no such traceable connectivity to the Vehicle Level exists, the variation point is instead considered as part of the model's architectural design space. Such variation points are not con-

Table 6. Characteristics of product line variability and architecture design space

Product Line Variability	Architecture Design Space
Purpose: satisfies needs of a particular market segment or mission (e.g. different models of cars)	Purpose: represents different implementation variants (e.g. multiple suppliers for a subsystem)
Variations have functional influence on the target system	Variations only have influence on non-functional system characteristics like cost, performance or dependability

figured as part of a product line configuration, but are instead decided at design time. Finding optimal allocations for these architecture variation point is the primary objective of the intended optimization approach.

8.3. Optimization Approach

In order to establish a robust foundation for optimization analyses of variant-rich EASTADL models, an expedient course of action is to first educe a rigorous mathematical model, which reduces the complexity of the initial problem down to the fundamental drivers of the optimization process. This distillation procedure results in a lean representation of the optimization problem, which can be easily transformed into fitting input data for existing optimization software.

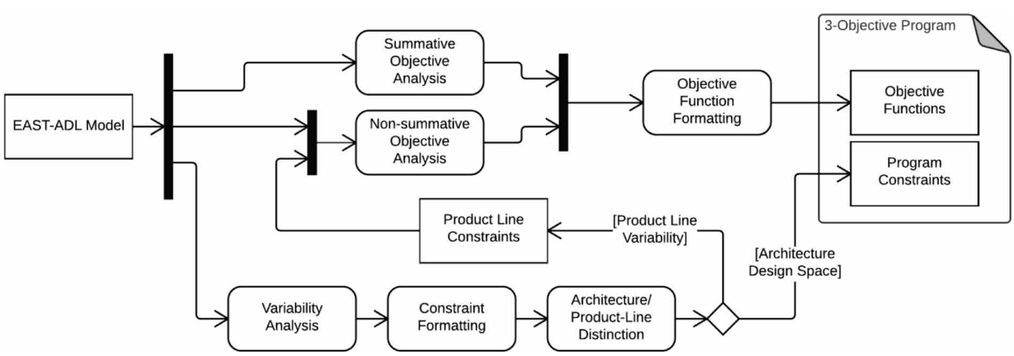
In EAST-ADL models however, the information relating to variability and design objectives is interwoven with other aspects of the given architecture description. There exists no separate depository for this kind of supplementary data; the identification and interpretation of the relevant information together with the transformation into a proper mathematical formulation is therefore part of the research presented in this section.

In order to accomplish this goal, we propose an approach for generating multi-objective programs from given EAST-ADL models, by representing the design objectives as a set of objective functions, and the system design space as a set of program constraints. Figure 28 illustrates the basic workflow of deriving such a program from a given model in terms of relevant activities. In order to allow for product-line-aware optimization, a number of specific considerations have to be made in regard to the examined variability information (cf. section 8.2.).

Our efforts to devise a comprehensive approach are part of an ongoing work in progress; our current, preliminary method still has certain restrictions in terms of the kind of variability and the kind of design objectives that can be handled and converted into a mathematical form. These restrictions are (a) the limitation to strictly boolean solution spaces, as opposed to non-boolean aspects introduced in particular by parametrization, and (b) the limitation to strictly linear design objectives, like unit cost and weight minimization.

The effect of these limitations on the generated output is that the resulting program is both linear and disjunctive. Future iterations, assuming that the current limitations can be lifted, may instead produce multi-objective non-linear mixed-integer programs.

Figure 28. Generation of a multi-objective program with linear and non-linear objectives



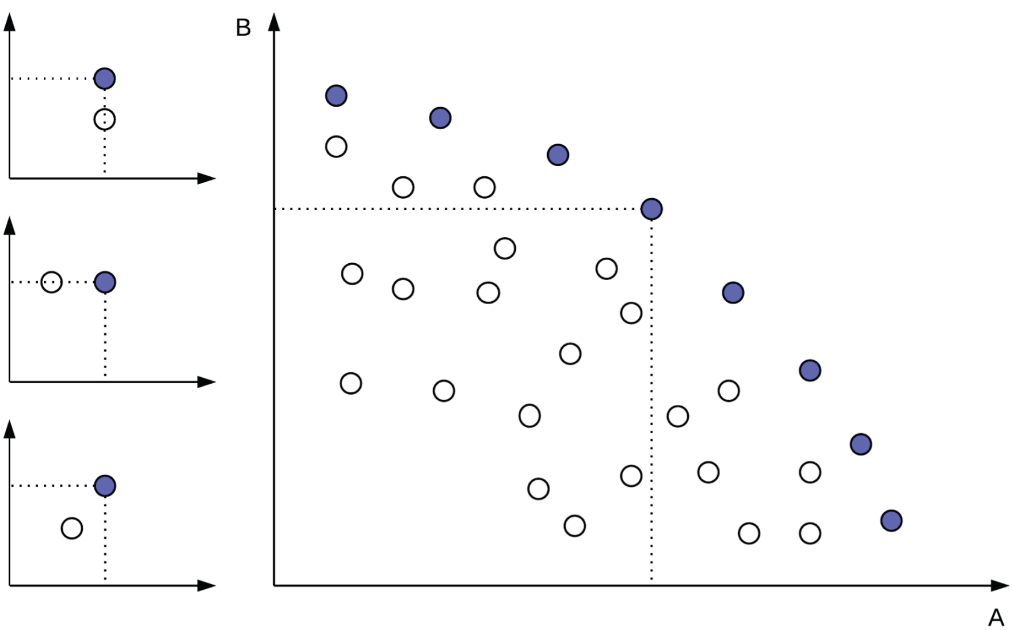
8.4. Pareto Optimality

When considering multiple objectives as part of an optimization analysis, it is usually impossible to find a single perfect solution which is truly optimal with respect to all objectives. This complication is caused by the often conflicting nature of typical design objectives, e.g. improving safety will decrease performance, and vice versa. One possible solution for this dilemma is to convert the multi-objective analysis into a single-objective analysis, by formulating an entirely new objective function from a weighted aggregation of the original objectives – a process known as scalarization or weighted normalization (Grodzevich et al. (2006)). This approach, while widely used, has significant drawbacks, in particular the fact that the weighting factors for the objectives have to be defined beforehand, i.e. before there is tangible knowledge about the properties of possible optimal solutions.

An alternative (and exhaustive) approach involves the simultaneous evaluation of all distinct objective functions in a single optimization sweep. This approach is usually regarded as genuine multi-objective optimization, also called *Pareto* optimization due to the particular composition of the analysis result. The typical outcome of a Pareto optimization is a Pareto set, also called Pareto frontier, which constitutes the set of all Pareto-optimal solutions of the optimization analysis (MAENAD (2012), p.54f).

Pareto optimality is based on the concept of dominance. A solution is called dominant if and only if no other solutions exist, which outperform it with respect to at least one objective without degradation in any other objective (Burke et al. (2005), p.414ff). Figure 3 depicts a solution space for an optimization analysis with two conflicting design objectives A and B plotted against the axes. In this example, both objective functions were to be maximized – think safety and performance. The Pareto set is represented by the filled bullets in the main graph. The smaller graphs depict typical scenarios for solution dominance (see Figure 29).

Figure 29. Pareto dominance for an exemplary solution space



8.5. Problem Representation as Multi-Objective Program

In order to attain a suitable mathematical representation of a given EAST-ADL optimization task, we propose an approach which makes use of the formulations of multi-objective programming (Benayoun et al. (1971), Marler et al. (2004)). Our current exclusion of non-boolean variabilities and non- linear design objectives, as noted in section 3, results in a program in which the constraints and objective functions are (a) linear, and (b) strictly binary. The application of such programs is called 0-1 integer linear programming, or disjunctive programming (Balas (1979)). Such a program can be formulated as follows:

$$Minimize\ Cx\ subject\ to\ Ax \geq a_0\ x \in \{0,1\}$$

where C is a (m;n)-Matrix representing a mapping of objective coefficients; A and a_0 are a (p;n)-Matrix and a p-vector, respectively, representing the mapping of the variability space to constraint coefficients; and x is an n-vector of strictly 0-1 binary architecture decisions.

The Matrix Cx is in principle just an alternative representation of a set of linear functions $F(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T$, where each function represents one distinct design objective, and thus one of the target parameters in the Pareto trade-off analysis. The coefficients can be parsed directly from extension constraints affiliated to the respective variable architecture elements in the given EAST-ADL model (cf. section 8.1.). Each of the Pareto-optimal solutions to this mathematical program represents a corresponding Pareto-optimal configuration of the initial system.

9. CONCLUSION AND FUTURE WORK

This paper presented model-based analysis and engineering capabilities offered by the EAST-ADL. Thanks to the omni-comprehensive nature of the EAST-ADL language, a wide range of advanced analyses have been seamlessly integrated in the framework, providing fine predictions of system properties at different abstraction levels, while initially stimulating the creativity of the engineer. These predictions allow front-ending verification activities at early stages of the system development process. Beside functional properties, verified by well-known model-checkers, the paper presented as well novel analysis methods for safety, timing and multi-objective optimisation analysis. Power analysis capabilities, even if not shown in the paper, could be easily provided thanks to available transformations towards simulation tools. The possibility of encompassing in one single framework different analyses has the further advantage of making possible optimization activities, where different analyses results are gathered by a centralized optimization framework able to rank analyzed model candidates in Pareto-optimal configurations.

While we think that model-based optimization is an emerging trend in both industry and research domains, multi-layered design flows pose several challenges from an optimization perspective (Broy et al. (2012)). It is not clear, in particular, how front-end optimization activities impact lower level designs, where new design elements (not taken into account in the higher-level optimization) emerge.

REFERENCES

Aldeida, A., ... (2013). Software architecture optimization methods: A systematic literature review. In IEEE Transactions on Software Engineering (pp. 658–683).

All Times. (2009). FP7 Project. Retrieved from http://www.mrtc.mdh.se/projects/all-times/

Arda, G. et al.. (2013). Analysis Support for TADL2 Timing Constraints on EAST-ADL Models.*Proceedings of the 7th European Conference on Software Architecture (ECSA)* (pp 89-105).

Atesst. (2006). FP7 ATESST 1 & ATESST 2 Projects. Retrieved from http://www.atesst.org

AUTOSAR. (2015, July). AUTOSAR Specification of Timing Extensions, Release 4.2.2. Retrieved from http://www.autosar.org/fileadmin/files/releases/4-2/methodology-and-templates/templates/standard/AUTOSAR_TPS_TimingExtensions.pdf

Balas, E. (1979). Disjunctive Programming. In Annals of Discrete Mathematics (Vol. 5, pp. 3–51). North-Holland Publishing Company. doi:10.1016/S0167-5060(08)70342-X

Benayoun, R., ... (1971). Linear Programming with multiple objective functions: Step method (STEM). In Mathematical programming (Vol. 1, pp. 366–375). Springer.

Blom, H., ... (2013). EAST-ADL – An Architecture Description Language for Automotive Software-Intensive Systems (White Paper Version 2.1.12).

Bozzano, M. et al.. (2003). Improving System Reliability via Model Checking: the FSAP /NuSMV-SA Safety Analysis Platform.*Proceedings of SAFECOMP* (pp. 49-62). doi:10.1007/978-3-540-39878-3_5

Broy, M., ... (2011). Cross-layer analysis, testing and verification of automotive control software. *Proceedings of the 11th International Conference on Embedded Software, EMSOFT* (pp 263-272). doi:10.1145/2038642.2038683

Burke, E. K. et al.. (2005). *Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Spinger. doi:10.1007/0-387-28356-0

Cancila, D., ... (2009) SOPHIA: a Modeling Language for Model-Based Safety Engineering. *Proceedings of the2nd International Workshop On Model Based Architecting and Construction Of Embedded Systems* (pp 11-26).

Chale, G., ... (2012), Towards an Architectural Design Framework for Automotive Systems Development. *Proceedings of the Third International Conference on Complex Systems Design & Management CSD&M* (pp 241-258).

Chen, D., ... (2011). Integrated Safety and Architecture Modeling for Automotive Embedded Systems. *e&i - elektrotechnik und informationstechnik*, 128(6).

Chen, D. et al.. (2013). *An Architectural Approach to the Analysis, Verification and Validation of Software Intensive Embedded Systems. Computing, 95(8), 649-688*. Springer.

Chen, D., Mahmud, N., Walker, M., Feng, L., Lönn, H., & Papadopoulos, Y. (2013). Systems Modeling with EAST-ADL for Fault Tree Analysis through HiP-HOPS.*Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems*.

Czarnecki, K., ... (2005), Cardinality-based feature modelling and constraints: A progress report. *Proceedings of theInternational Workshop on Software Factories* (pp. 16–20).

Czarnecki, K., ... (2005). Formalizing Cardinality-based Feature Models and their Specialization. In Software Process: Improvement and Practices (pp 7-29).

Di Natale, M., & Sangiovanni-Vincentelli, A. L. (2010). Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. *Proceedings of the IEEE*, 98(4), 603-620. doi:10.1109/JPROC.2009.2039550

EAST-ADL. (2015). Retrieved from http://www.east-adl.info/

EAST EEA. (2001). ITEA Project reference 00009.

Espinoza, H., ... (2010). Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. In R.F. Paige, A. Hartman & Arend Rensink (Eds.). *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '09)*. doi:10.1007/978-3-642-02674-4_8

Fussel, J. B. et al.. (1976). On the quantitative analysis of Priority-AND failure logic. *IEEE Transactions on Reliability*, *R-25*(5), 324–326. doi:10.1109/TR.1976.5220025

Grodzevich, O. et al.. (2006). Normalization and other topics in multi-objective optimization.*Proceedings of the Fields–MITACS Industrial Problems Workshop* (pp. 89–102).

ISO. (2009). Road vehicles — Functional safety, ISO/DIS 26262.

MAENAD. (2009). FP7 Project. Retrieved from http://www.maenad.eu/

MAENAD. (2012). Language Concepts Supporting Engineering Scenarios, D5.3.1. Retrieved from http://www.maenad.eu/public/Deliverables/MAENAD_Deliverable_D3.1.1_V3.0.pdf

MAENAD. (2013). MAENAD Analysis Workbench. D5.2.1. Retrieved from http://www.maenad.eu/public_pw/MAENAD_Deliverable_D5.2.1_V3.0.pdf

Mahmud, N. (2012). *Dynamic Model-based Safety Analysis: From State Machines to Temporal Fault Trees* [Ph.D. dissertation]. Univ. of Hull, UK.

Mahmud, N., ... (2010). A Translation of State Machines to Temporal Fault Trees. *Proceedings.of 40th Annu. IEEE/IFIP Int. Conf. Dependable Syst. and Netw*. (pp. 45-51). doi:10.1109/DSNW.2010.5542620

Mahmud, N. et al.. (2012). Compositional synthesis of Temporal Fault Trees from State Machines. *ACM SIGMETRICS, Performance Evaluation Review (Special Issue on Modeling Dynamic Behaviors of Complex Distrib. Syst.)*, *39*, 79–88.

Marler, R. T. et al.. (2004). Survey of multi-objective optimization methods for engineering. In *Structural and multidisciplinary optimization* (Vol. 26, pp. 369–395). Springer. doi:10.1007/s00158-003-0368-6

Mehiaoui, A. et al.. (2012). Optimizing the Deployment of Distributed Real-Time Embedded Applications. *Proceedings of International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)* (pp 400-403). doi:10.1109/RTCSA.2012.61

Mehiaoui, A., ... (2013). A Two-step Optimization Technique for Functions Placement, Partitioning, and Priority Assignment in Distributed Systems. Proceedings of Languages, Compilers and Tools for Embedded Systems (LCTES) (pp121-132).

Merle, G., Roussel, J.-M., Lesage, J.-J., & Bobbio, A. (2010). Probabilistic Algebraic Analysis of Fault Trees With Priority Dynamic Gates and Repeated Events. *IEEE Transactions on Reliability*, *59*(1), 250–261. doi:10.1109/TR.2009.2035793

MAENAD Methodology. (2013). MAENAD Design Methodology. D2.2.1. Retrieved from http://www.maenad.eu/public_pw/MAENAD_Deliverable_D2.2.1_V2.0.pdf

Metzger, A. et al.. (2007), Variability management in software product line engineering. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, IEEE Computer Society (pp. 186–187). doi:10.1109/ICSECOMPANION.2007.83

OMG MARTE. (2011, June). UML Profile for Modelling and Analysis of Real-Time and Embedded systems (MARTE), formal/2011-06-02, Version 1.1.

OMG QFTP. (2011, April). UML Profile for QoS and Fault Tolerance (QFTP), formal/2008-04-05, Version 1.1,.

OMG SYML. (2012, June). System Modeling Language (SysML), formal/2012-06-01, Version 1.3.

OMG UML. (2015, March). Unified Modeling Language (UML), formal/2015-03-01, Version 2.5.

Papadopoulos, Y., ... (1999). Hierarchically performed hazard origin and propagation studies. In *Proceedings of SAFECOMP, LNCS* (Vol. 1698, pp. 139-152).

Reiser, M.-O. et al.. (2009), Compositional variability – concepts and patterns.*Proceedings of the 42nd Hawaii International Conference on System Sciences*. IEEE.

SAE. (2009, January). Architecture Analysis and Design Language (AADL) AS-5506A, The Engineering Society for Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 2.0.

Thüm, T. et al.. (2012). *Analysis strategies for software product lines (Technical report)*. Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik.

TIMMO2USE. (2012). Timing Model – Tools, algorithms, languages, methodology, USE cases. D13. Retrieved from http://www.timmo-2-use.org/deliverables/TIMMO-2-USE_D13.pdf

TIMMO. (2007). ITEA2 Project. Retrieved from http://www.timmo-2-use.org/timmo/index.htm

Walker, M., ... (2013). Automatic optimisation of system architectures using EAST-ADL. *Journal of Systems and Software*.

Walker, M., & Papadopoulos, Y. (2009). Qualitative temporal analysis: Towards a full implementation of the Fault Tree Handbook. *Control Engineering Practice*, *17*(10), 1115–1125. doi:10.1016/j.conengprac.2008.10.003

Wozniak, E. et al.. (2014). Assigning time budgets to component functions in the design of time-critical automotive systems.Proceedings of*ASE* (Vol. *2014*, pp. 235–246).

ENDNOTES

[1] State Automata to Fault-trees extended (if necessary) with temporal information (ORA in Greek).

Ramin Tavakoli Kolagari works at the Nuremberg Institute of Technology at the faculty of Computer Science being responsible for the software engineering field of teaching. He studied computer science and business administration at the Technical University of Berlin and received his PhD at the University of Ulm on the topic of automotive requirements engineering and software product lines. Tavakoli lectured at the University of Ulm and at the Technical University of Berlin. He researched and published in the area of software reuse in general and specifically in the area of software product lines for the automotive domain. Today, his research focus lies in the area of adaptive autonomous automotive systems and model-based development for describing the system and software architecture of embedded automotive systems, e.g., with EAST-ADL. After his studies, Tavakoli worked in the research departments of internationally operative carmakers in Germany and Sweden in the areas software development, software architecture, and model-based development. Beside his work in the business units he was involved in many European research projects in the context of his research interests.

DeJiu Chen received his PhD degree in mechatronics with a research on embedded computer control systems from KTH. His research interests are in the areas of embedded control systems (ECS), cyber-physical systems (CPS), and system-of-systems (SoS) with active research on: 1. Methods and tools for model-based engineering; 2. Architecture design and quality management; 3. Safety Engineering; and 4. Design for Self-X properties. From 2007 to 2009, Dr. DeJiu Chen worked for Enea Data AB, Sweden, as a senior technical instructor. He is currently an associate professor at KTH.

Henrik Lönn has a PhD in Computer Engineering from Chalmers University of Technology, Sweden, with a research focus on safety-critical real-time systems. At Volvo, he has worked with prototypes, architecture modelling and communication aspects on vehicle electronic systems. He is also participating in national and international research collaborations on embedded systems development. Previous project involvement includes X-by-Wire, FIT, EAST-EEA and TIMMO and the coordination of FP6 and 7 projects ATESST, ATESST2 and MAENAD.

Chokri Mraidha is a researcher at the Laboratory of Model Driven Engineering for Embedded Systems of the CEA LIST institute in France. He got a master degree in distributed computing in 2001, a PhD in Computer Science in 2005 and a "Habilitation à Diriger les Recherches" in 2015. His research and development interests include dependable real-time and embedded systems model driven development, modelling languages design, instrumentation and control architectures design, optimized software real-time architecture synthesis and model compilation. He is involved in UML-based OMG standards for design of real systems like SysML and MARTE and the AUTOSAR standard for automotive. He is working in European and French research projects developing model-based approaches for design and verification of architectures for critical real-time systems for automotive, railway, aerospace and nuclear power plants.

Sara Tucci-Piergiovanni is research coordinator at the CEA LIST's department of system and software engineering. She received her MS and PhD degrees in computer engineering from the Sapienza University of Rome, respectively, in 2002 and 2006. Her master's thesis was awarded in 2002 with the national prize for best ICT master thesis in ICT from Confederation of Italian Industry. At Sapienza, she held a lecturer position teaching a course on distributed systems till 2008. Since her master degree, she publishes regularly in peer-reviewed scientific forums and journals in the areas of distributed-, real-time-, mission critical- systems and software. She serves regularly as reviewer in internantional journals (TPDS, JACM, TCS, JSS, SoSym, etc) and conferences (ACM SAC, ETFA, EDCC, etc.). She co-chaired the International Conference in Distributed Event Systems in 2008. She is also involved in numerous research and industrial projects developing model-driven engineering approaches for the automotive domain, among these projects she led the tooling workpackage in the EC funded MAENAD project.