



MAENAD



Grant Agreement 260057

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

| | |
|----------------------------|----------------------------------|
| Report type | Deliverable D5.2.1 |
| Report name | MAENAD Analysis Workbench |
| Dissemination level | PU |
| Status | Final |
| Version number | 4.0 |
| Date of preparation | 2014-02-25 |

Authors**Editor**

DeJiu Chen

E-mail

chen@md.kth.se

Authors

Carl-Johan Sjöstedt

E-mail

carlj@md.kth.se

Matthias Biehl

biehl@md.kth.se

Hans Blom

hans.blom@volvo.com

Mark-Oliver Reiser

mark-oliver.reiser@tu-berlin.de

Sara Tucci-Piergiovanni

sara.tucci@cea.fr

Martin Walker

martin.walker@hull.ac.uk

Frank Hagl

frank.hagl@continental-corporation.com

DeJiu Chen

chen@md.kth.se

Henrik Lönn

henrik.lonn@volvo.com

Tobias Wägemann

tobias.waegemann@th-nuernberg.de

Reviewers

Mark-Oliver Reiser

E-mail

mark-oliver.reiser@tu-berlin.de

Hans Blom

hans.blom@volvo.com

Nigsti Ayele

nigsti.ayeale@systemite.se

The Consortium

Volvo Technology Corporation (S)

Centro Ricerche Fiat (I)

Continental Automotive (D)

Delphi/Mecel (S)

4S Group (I)

ArcCore AB (S)

MetaCase (Fi)

Systemite (SE)

CEA LIST (F)

Kungliga Tekniska Högskolan (S)

Technische Universität Berlin (D)

University of Hull (GB)

Revision chart and history log

| Version | Date | Reason |
|----------------|-------------|-------------------------------|
| 1.0 | 2011-05-31 | First release |
| 2.0 | 2012-08-31 | Second release |
| | 2013-08-22 | Internal review |
| 3.0 | 2013-08-31 | Third release |
| 4.0 prel | 2014-02-18 | General M42 update for review |
| 4.0 | 2014-02-25 | Final release |

| Approval | Date |
|-----------------|-------------|
| Henrik Lönn | 2014-02-25 |

Table of contents

| | |
|--------------------------------------------------------------|----|
| Authors..... | 2 |
| Revision chart and history log | 3 |
| Table of contents | 4 |
| 1 Introduction | 6 |
| 2 The components of the MAENAD Analysis Platform | 7 |
| 2.1 AUTOSAR Gateway | 8 |
| 2.1.1 Objectives | 8 |
| 2.1.2 Related Project Requirements..... | 8 |
| 2.1.3 Background and Version History | 8 |
| 2.1.4 Key Features..... | 8 |
| 2.2 Qompass Timing Analysis Gateway..... | 13 |
| 2.2.1 Objectives | 13 |
| 2.2.2 Related Project Requirements..... | 13 |
| 2.2.3 Background and Version History | 13 |
| 2.2.4 Key Features..... | 14 |
| 2.3 Simulink Gateway | 20 |
| 2.3.1 Objectives | 20 |
| 2.3.2 Related Project Requirements..... | 20 |
| 2.3.3 Background and Version History | 20 |
| 2.3.4 Key Features..... | 20 |
| 2.4 HiP-HOPS Safety Analysis Gateway..... | 22 |
| 2.4.1 Objectives | 22 |
| 2.4.2 Related Project Requirements..... | 22 |
| 2.4.3 Background and Version History | 22 |
| 2.4.4 Key Features..... | 22 |
| 2.5 Architecture optimization and configuration..... | 24 |
| 2.5.1 Objectives | 24 |
| 2.5.2 Related Project Requirements..... | 24 |
| 2.5.3 Background and Version History | 24 |
| 2.5.4 Key Features..... | 27 |
| 2.6 ASIL allocation with EPM/HiP-HOPS | 30 |
| 2.6.1 Objectives | 30 |
| 2.6.2 Requirements from WT2.1: Identification of needs | 30 |
| 2.6.3 Background and Version History | 30 |
| 2.6.4 Key Features of ASIL allocation | 30 |
| 2.7 UPPAAL/SPIN Model-Checking Gateway..... | 32 |

| | | |
|--------|------------------------------------------------|----|
| 2.7.1 | Objectives | 32 |
| 2.7.2 | Related Project Requirements | 32 |
| 2.7.3 | Background and Version History | 32 |
| 2.7.4 | Key Features..... | 32 |
| 2.8 | EATOP Analyzer..... | 35 |
| 2.8.1 | Objectives | 35 |
| 2.8.2 | Related Project Requirements | 35 |
| 2.8.3 | Background and Version History | 35 |
| 2.8.4 | Key Features..... | 35 |
| 2.9 | Variability Resolution | 36 |
| 2.9.1 | Objectives | 36 |
| 2.9.2 | Related Project Requirements | 36 |
| 2.9.3 | Background and Version History | 36 |
| 2.9.4 | Key Features..... | 36 |
| 2.10 | MODELISAR Functional Mock-up Unit Import | 37 |
| 2.10.1 | Objectives | 37 |
| 2.10.2 | Related Project Requirements..... | 37 |
| 2.10.3 | Background and Version History | 37 |
| 2.10.4 | Key Features..... | 37 |
| 3 | References..... | 38 |

1 Introduction

Deliverable D5.2.1 consists of the tool implementations that make up the MAENAD analysis workbench (MAW), which extends the MAENAD Modeling Workbench (D5.1.1) for engineering tasks related to architecture refinement and quality assessment. The main objective of this work is to validate the analysis concepts developed in WP3. The MAW is available as a project dissemination in order to make the analysis concepts and tool support more accessible and understandable for public audience.

This document serves as an overall introduction of the available tools in the MAW. For each tool implementation, there are the following subsections: 1. *Objectives*, 2. *Background and Version History*, and 3. *Implementation*. The first section describes the key features to be supported by a tool implementation as well as the related language and project requirements being fulfilled. The second section provides information about the technical context and evolution. The third section introduces the expected EAST-ADL models and tool principles.

2 The components of the MAENAD Analysis Platform

The MAENAD analysis workbench (MAW) includes tool plugins that are newly developed to address the method and tool concepts being developed through the MAENAD project. It also contains tool plugins that have their roots in the ATESSST [1], ATESSST2 [1] and EDONA [2] projects, but are evolved for new releases of EAST-ADL, new tool environments, or new technical solutions. Please see the subsequent subsections for further details. The key features of these tool plugins are listed below.

| Tool plugin | Main developer | Key feature |
|---------------------------------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AUTOSAR Gateway | CEA | Providing an enhanced transformation from EAST-ADL design architecture description to AUTOSAR compliant software architecture, based on the ARTOP framework. |
| Qompass Timing Analysis Gateway | CEA | Providing support for early-stage timing analysis of EAST-ADL models. |
| Simulink Gateway | KTH | Providing support for automatic creation Simulink/Matlab models based on EAST-ADL architecture models and behaviour constraints. |
| HiP-HOPS Safety Analysis Gateway | KTH | Providing support for static safety analysis with HiP-HOPS based on EAST-ADL error models and error behaviour descriptions. |
| Architecture optimization and configuration | TUB | Providing support for multi-objective optimization with the prototype tool OptiPAL/EPM. |
| ASIL allocation with HiP-HOPS | UoH/TUB | Providing support for ASIL allocation with HiP-HOPS |
| UPPAAL/SPIN Model-checking Gateway | KTH | Providing support for formal behavioural analysis with UPAAL/SPIN based on EAST-ADL behaviour constraints. |
| Analyzer | VTEC | Analyze EAST-ADL Models based on non-functional annotations |
| Variability Resolution | OHM | Resolve variability in EAST-ADL model |
| FMU Import | VTEC | Providing support for generating EAST-ADL FAA models from Functional Mock-up Units. |

2.1 AUTOSAR Gateway ---

2.1.1 Objectives ---

The AUTOSAR gateway aims to provide an enhanced transformation from the EAST-ADL design architecture to AUTOSAR vehicle architecture design and initial system configuration for a more detailed architecture specification and analysis. The gateway is based on the ARTOP framework, which is an implementation of common base functionality for AUTOSAR development tools, available free of charge for AUTOSAR members [3]. The transformation takes as input the EAST-ADL Design Level with functional description, hardware description and allocation information and generates a tentative AUTOSAR software architecture, a hardware topology and mapping constraints (coming from EAST-ADL allocation information). The generation of the tentative software architecture is based on mappings between EAST-ADL functions and AUTOSAR software components/runnables that fit in the EAST-ADL Implementation Level.

2.1.2 Related Project Requirements ---

The AUTOSAR gateway is mentioned in the following requirement:

DOW#0111: The SWC Synthesis (FDA-IL) shall be modelled in MAW-AR Gateway plugin [4].

2.1.3 Background and Version History ---

The AUTOSAR gateway was built on results from EDONA and ATESS2 projects. The first version has been released at M12 and then it had been ported to the new Papyrus MDT platform and 2.1.9 EAST-ADL profile version.

A second version of the AUTOSAR gateway has been released at M24. In this version the transformation towards the AUTOSAR implementation architecture relies on an AUTOSAR UML profile (subset of AUTOSAR centred on relevant templates: software component, system and ECU resource namely). As a result, the transformation from EAST-ADL design architecture to AUTOSAR vehicle design architecture/system configuration produces UML profiled models.

AUTOSAR gateway implements also export functionality from AUTOSAR-UML profiled models to AUTOSAR XML.

2.1.4 Key Features ---

As specified before, AUTOSAR Gateway servers to generate AUTOSAR model (UML model profiled with AUTOSAR concepts) and out of the last, AUTOSAR Gateway can generate arxml file. Therefore this subsection will be divided accordingly to this two-step generation process.

2.1.4.1 EAST-ADL to AUTOSAR model ---

Generation of AUTOSAR model consists of three phases. These are generation of the *Application View*, the *Topology View* and the *Mapping View*. The first one contains specification of the software components types and their prototypes. Topology View relates to the hardware topology. Mapping View encompasses the definition of a mapping, i.e. software components to ECUs allocation. Proper and complete generation of each view depends on the information included in the EAST-ADL model. Let us consider the EAST-ADL model as composed out of three packages representing the functional architecture (FDA – Functional Design Architecture), hardware topology

(HDA – Hardware Design Architecture) and Allocation, i.e. mapping of functional entities onto the hardware topology. Figure 1 represents general dependencies of the generation results from the information included in the EAST-ADL model. Namely, in order to generate complete and correct Application View it is necessary to provide FDA, HDA and Allocation. This is a consequence of the approach used to generate software architecture. Namely, all the atomic functions of the same composite function and allocated on the same ECU are transferred to one atomic software component type. The last contains the runnable entities, where each runnable entity is generated from one atomic function. In general the way in which software architecture is generated, i.e. software component types, their prototypes and runnable entities, is influenced by the compositional structure of the functional model (FDA) but also the way in which atomic functions are allocated on the nodes. Therefore all three models, i.e. FDA, HDA and Allocation are necessary to generate Application View model.

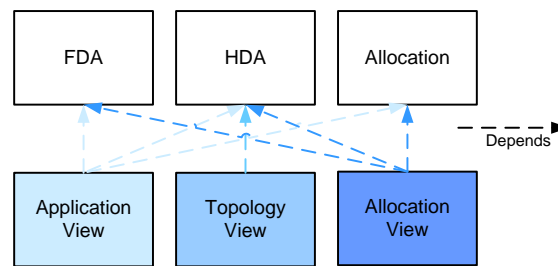


Figure 1. Dependencies in the Generation of the AUTOSAR Model from the EAST-ADL Model

Generation of the Topology View depends only on the information included in the HDA. Accordingly, the absence or presence of the FDA model or Allocation model has no influence on the Topology View.

The last Allocation View depends on FDA, HDA and Allocation model. This dependency is rather obvious. If the specification of FDA, HDA and Allocation is not complete, as explained before, complete and correct Application View cannot be generated, hence there are no entities to allocate. In addition if an HDA specification does not exist, there are no hardware entities on which the software components can be allocated. Lastly, as EAST-ADL Allocation model specifies functional allocation, the allocations of software components can be inferred by tracing the realization dependencies between functions and the produced software components.

2.1.4.2 AUTOSAR model to arxml

Generation of the arxml file is simpler than the generation of the AUTOSAR model from EAST-ADL. This is a consequence of using one-to-one transformation, i.e. each entity from the AUTOSAR model has one, corresponding entity in the arxml file. This was not the case for the AUTOSAR model generation, where the relation between the EAST-ADL model and AUTOSAR elements is not that obvious, due to the different concepts present in the different languages.

In the context of the implementation, similarly to the generation of AUTOSAR model, generation of arxml file is divided into a few phases, five in this case. They are as follows:

- Phase 1: packages generation: all the packages and sub-packages, present in the AUTOSAR model are reflected in the arxml file.
- Phase 2: components generation: all the software component types and software component prototypes has their counterparts in the arxml file. Also in this phase, ports and their interfaces are generated.
- Phase 3: connectors generation: this phase generates connectors communicating ports of different software components.
- Phase 4: hardware platform generation: this phase can be divided into three sub-phases:

- Phase 4.1: generation of ECUs
- Phase 4.2: generation of Sensors and Actuators
- Phase 4.3: generation of hardware pins
- Phase 5: allocation generation: this phase produces the specification of software components to ECUs mapping.

All of these phases are fully independent and hence failure in the generation of one of them does not influence other phases. If the generation of arxml file, directly follows the generation of the AUTOSAR model using the AUTOSAR Gateway, the arxml file will be produced without any problems. However the idea of the two-fold process (i.e. EAST-ADL model to AUTOSAR model and AUTOSAR model to arxml) was to enable the designer to change the generated implementation model at the modelling level, not at the level of the arxml file. Hence, while doing the specific changes at the AUTOSAR model level, the designer may have the risk of violating important overall system architectural rules.

The entire input model needs to be structured in a way presented in Figure 2. Arxml file generator searches first for the package called Technical View, and then depending on the generation phase it searches for the corresponding elements either in the Application View or the Topology View or the Mapping View package. For instance if this is “Components generation” phase it will look for the software component types and prototypes in the Application View package. Otherwise if there is a software component type or prototype specified outside of this package, no corresponding entity in the arxml file will be generated. Other phases are run analogously. Below is a set of general rules for the appropriate containment of AUTOSAR model elements within the three mentioned packages, so their corresponding arxml file entities will be generated.

Application View: this package should contain all software component types, such as ApplicationSwComponentType, SensorActuatorSwComponentType or CompositionSwComponentType. Then all the prototypes, i.e. SwComponentPrototypes should be specified as owned attributes of the composition software component (see Figure 3). InternalBehaviour should be specified as an owned behaviour of a software component type. Ports of SwComponentTypes should be specified as owned attributes (see Figure 5). Lastly, port connectors are specified as owned connectors of the software composition type (see Figure 6). In general, all the information needed for the phase 2 and 3 should be included in the Application View.

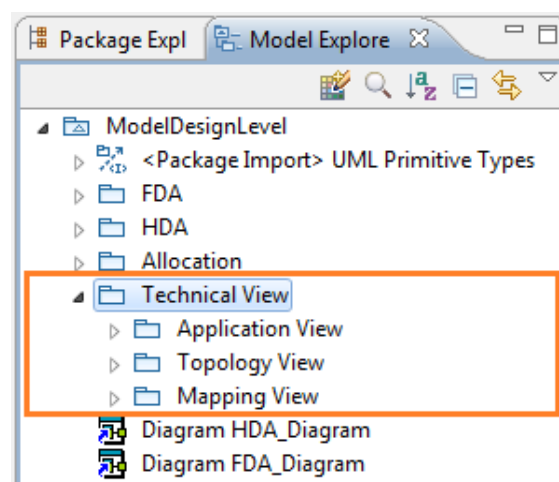


Figure 2. Structure of the AUTOSAR model recognized by the arxml file Generator

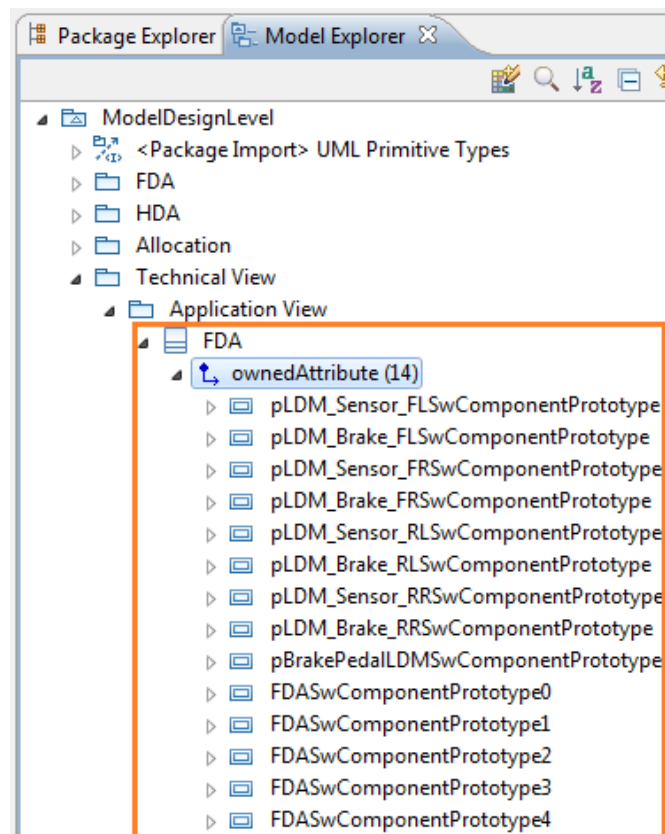


Figure 3. Specification of Software Component Prototypes as Owned Attributes of a CompositionSwComponentType

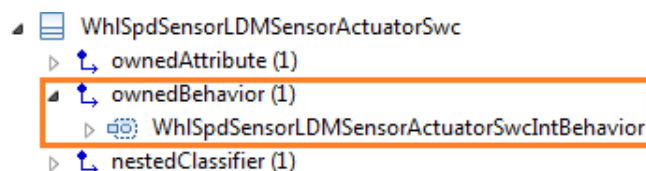


Figure 4. InternalBehavior of SwComponentType specified as an ownedBehavior

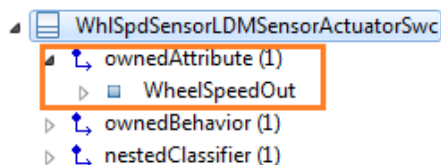


Figure 5. Port specified as Owned Attribute of Software Component Type

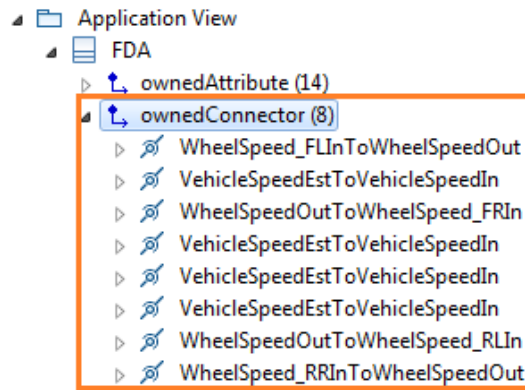


Figure 6. Port connectors specified as Owned Connectors of Composition Software Type

Phase 4 relates to the Topology View and hence all the elements that refer to hardware should be specified within this package. These are sensors and actuators (SensorHw, ActuatorHw), ECUs and their occurrences, communication connectors, hardware pins, communication clusters (e.g. FlexrayCluster) and physical channels (e.g. EthernetPhysicalChannel). Few important things should be kept in mind concerning the Topology View. First, AUTOSAR stereotypes SensorHw and ActuatorHw are applicable on Class and represent the sensor/actuator type. The occurrence of specific sensor/actuator is specified as a Property, but no stereotype is applied. This comes from the fact that no specific element exists in AUTOSAR to model the instance of either sensor or actuator, while EAST-ADL has the HardwareComponentPrototype. The Property representing the occurrence of a specific sensor/actuator should be present as an owned attribute of the sensor/actuator type. Secondly occurrences of ECUs, i.e. ECUInstance are specified as owned attributes of their ECU type. Next, communication connectors of ECU instances are specified as owned attributes of their ECU type. Lastly, concerning the global communication, i.e. communication buses, they are specified using the AUTOSAR stereotypes such as CanCluster (for the CAN bus), FlexRayCluster (for the FlexRay bus), etc. and their corresponding physical channels, such as CanPhysicalChannel (for the CanCluster), etc. The physical channel is specified as the UML Connector stereotyped with the PhysicalChannel stereotype. The physical channel should be within the owned connectors of the Cluster, present in the model as the UML Class (see Figure 7).

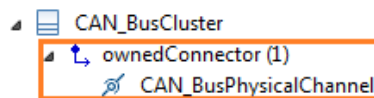


Figure 7. Specification of a Cluster and corresponding Physical Channel as an Owned Connector.

The last, Phase 5 requires only that the specification of software allocation is contained in the Mapping View package. See example on the Figure 8.

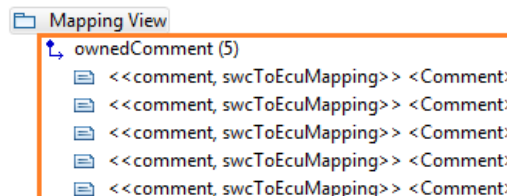


Figure 8. Specification of a Mapping View with the Owned Comments modeling allocation of Software Components

2.2 Qompass Timing Analysis Gateway

2.2.1 Objectives

Through MANEAD, the following timing analyses have been identified as best-suited to support EAST-ADL Design level models [6]:

1. *Early Stage Schedulability Analysis*. The allocation model of EAST-ADL defines on which ECUs, functions will be executed and on which buses, communication between functions will take place. Based on this information, the following two interesting metrics, relevant from a schedulability point of view, can be considered:
 - Resource Utilization. Resource utilization is a function of (i) the function's activation rate and (ii) a time budget representing the time an execution/communication will take. Based on utilization of single resources, other related interesting metrics can also be extracted, as load distribution and function/signals extensibility (function of processors/bus slacks).
 - Interference Time: represents the waiting time to access shared resources (CPU/Bus). This delay is caused by concurrent functions/signals that are allocated to the same execution/communication node. Small interference is desirable to minimize end-to-end latency.
2. *Schedulability analysis*. Schedulability analysis is applied for the special case of linear chains of activations running on a mono-processor system and when chain rates are harmonic. The task model is generated automatically. This generation is transparent to the user. Once the task model is obtained, a response time will be computed for each end-to-end chain (thread) through Rate Monotonic Analysis [6].

2.2.2 Related Project Requirements

The Timing Analysis plugin is mentioned in the following requirement:

DOW#0110: The Timing analysis (DL) shall be modelled in MAW-Timing plugin [4].

2.2.3 Background and Version History

Let us be reminded that at the beginning of the MAENAD project the analysis engine should have been provided as a third-party tool. On the other hand, after EDONA and ATESS2, CEA worked on the implementation of schedulability analysis algorithms as part of its research activities (not included in the MAENAD project) in its own MARTE-based plug-in called Qompass. This timing analysis support was not completely compliant with timing analyses identified as best-suited for EAST-ADL design models (Section 2.2). Moreover, in order to directly analyse EAST-ADL models, a transformation between EAST-ADL profile models and entry models for Qompass was needed.

During year 2 the Qompass timing support has been adapted/enhanced in order to support EAST-ADL design-level timing analyses. An EAST-ADL/Qompass transformation has been developed as well. A new version of the Timing Analysis plug-in, has been released at M24 embedding these new features. At the end of year 2, the input model for the Qompass tool assumed only linear event chains, where each chain contained a linear sequence of functions and where neither functions nor stimuli could belong to more than one event chain. In order to support the analysis of the brake-by-wire (BBW) system, an extension was needed, as the BBW owns functions belonging to multiple event chains. With the latest release, this extension has been implemented and now Qompass can run on the BBW model.

2.2.4 Key Features

In this section we present the EAST-ADL methodology needed to build a well-formed model for timing analysis. Timing analysis needs a minimal amount of information that must be specified. The goal of timing analysis is to give estimation of the quality of an allocation of functions to hardware nodes. Indeed, the usage of concrete resources for the execution of functions and communication among functions has a huge impact on the ‘timing’ aspect of the application, i.e. the delay for an expected response to be produced. The main issue here is that in the general case resources will be shared among multiple functions. The way in which resources are shared determines the time for a given function to complete. From these considerations it is clear that relevant information for the timing analysis is: the chains of function activations that compose a system response and subject to a deadline; the allocation of functions to nodes; the estimation of the resource demand of each function, the maximal utilization capacity of resources.

In the reminder of this section we describe how to specify this information in EAST-ADL using a running example, the BBW model. In Figure 9 an excerpt of the functional design architecture of the BBW is shown. All these functions belong to system responses subject to a hard deadline.

In order to define a system response, i.e. a path of function activations, from a stimulus to a response, we need to specify an EventChain. Figure 10 shows the event chain whose stimulus is the ‘BrakePedalSensorInputPortEvent’ EventFunctionFlowPort, related to the ‘PositionIn’ FunctionPort of the ‘pBrakePedalSensor’ FunctionPrototype (see Figure 11) and whose response is the ‘BrakeActuatorFLOutputPortEvent’ EventFunctionFlowPort, related to the ‘BrakeTorq’ FunctionPort of the ‘pBrakeActuator_FL’ FunctionPrototype (see Figure 12). Please note the introduction of the following modelling rules:

- An EventFunctionFlowPort must have both the port and port_path attributes specified.
- An event chain must have exactly one stimulus and exactly one response.
- Each stimulus must be generated by a unique source function

Once an event chain has been specified like that, the entire path can now be unambiguously derived only if the graph of functions is a directed acyclic graph without cycles. In this particular case this means that all ports are unidirectional, with the ‘PositionIn’ FunctionPort with direction=‘in’ and the BrakeTorq FunctionPort with direction= ‘out’. Note that it is *mandatory* to remove all the bidirectional function ports to run the transformation and the analysis.

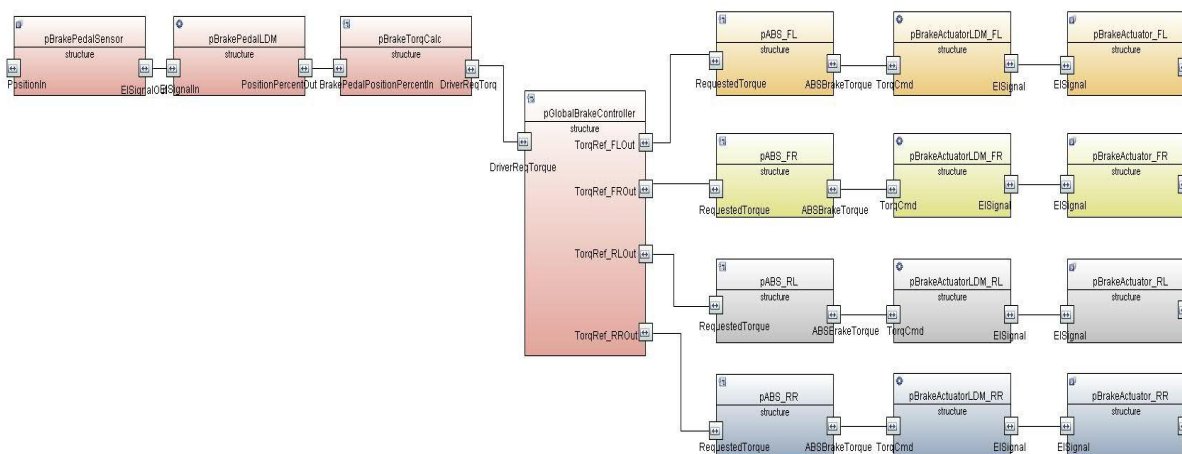


Figure 9. Excerpt of the Functional Design Architecture

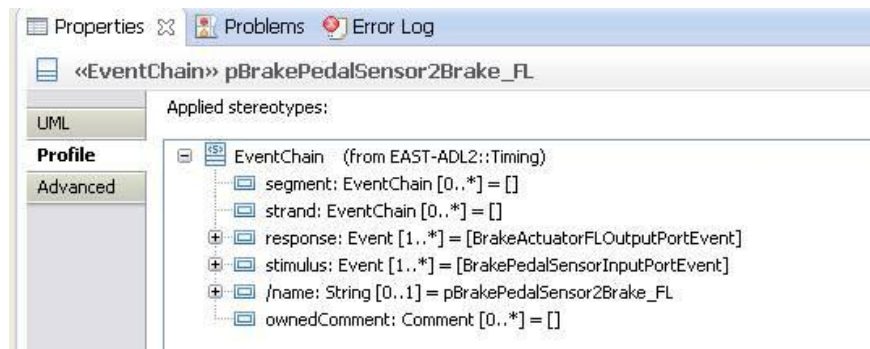


Figure 10. Example of Event Chain

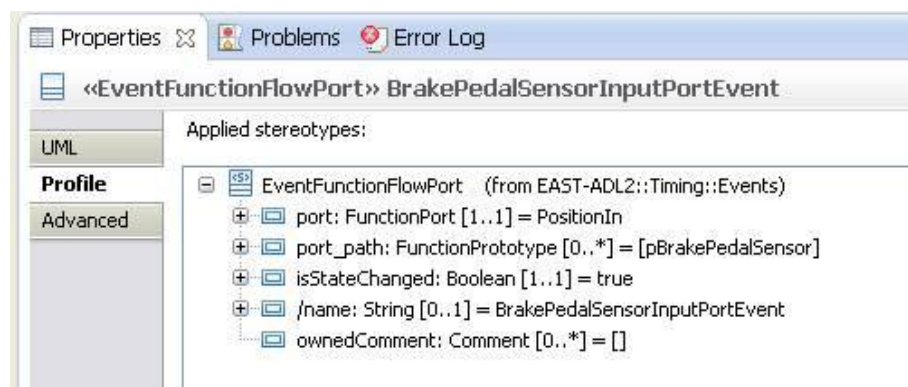


Figure 11. Stimulus Specification

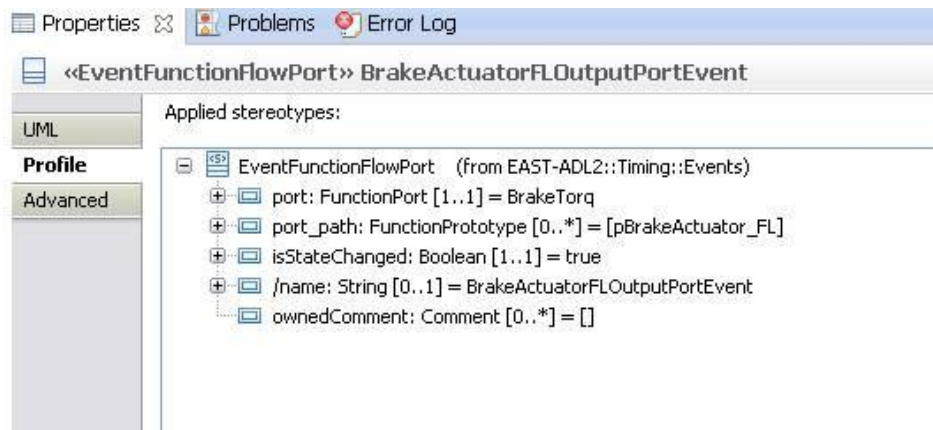


Figure 12. Response Specification

In order to characterize an event chain in terms of timing properties, a number of constraints must be specified, as follows:

- Periodic Constraint, for the stimulus of each event chain; it specifies the arrival period for the stimulus;
- Reaction Constraint for the entire event chain; it specifies the relative deadline for the execution of all functions belonging to the path identified by the event chain;
- Execution Time Constraint for each function in the path of the event chain; it specifies the worst case execution time for the function to be executed as if it were executed in isolation on the resource;

Note that the specification of these properties is not mandatory, as the Qompass tool allows for specifying this information directly in the MARTE model, right after the transformation EAST-ADL/Qompass.

The specification of a periodic constraint is shown in Figure 13. An element must be created while specifying two attributes: *period* and *event*.

- In order to set the period, a TimeDuration element must be previously defined. The TimeDuration element must have the attribute *value* specified (see Figure 14). This value is the actual period for the constraint.
- In order to set the event the constraint refers to, the event attribute must be set. Figure 13 shows that the constraints refers to the event 'BrakePedalSensorInputPortEvent', which is the stimulus of the event chain shown in Figure 10.

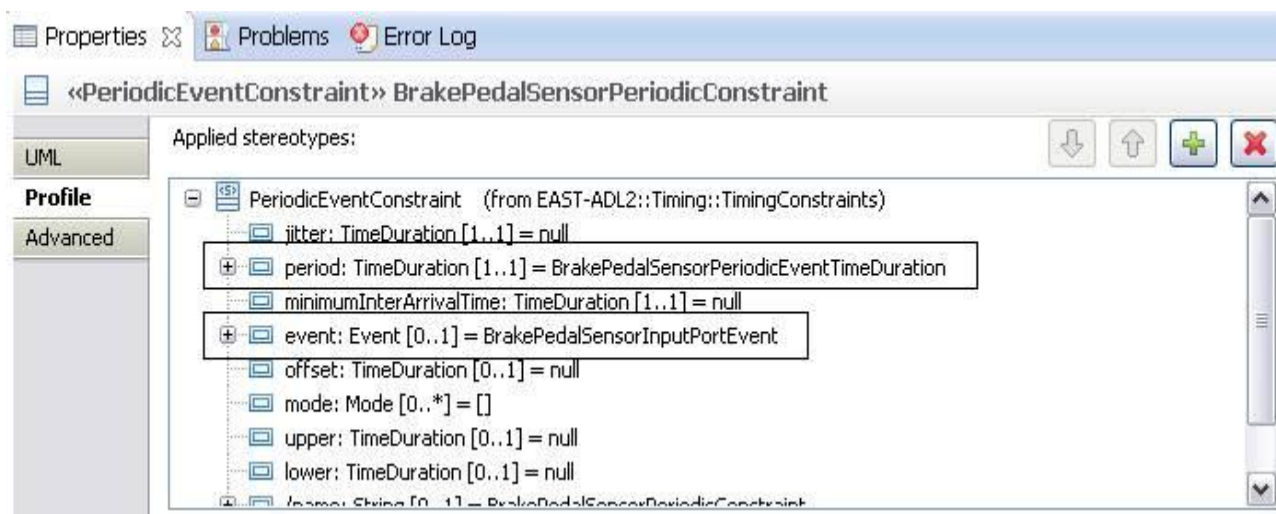


Figure 13. Periodic Event Constraint Specification

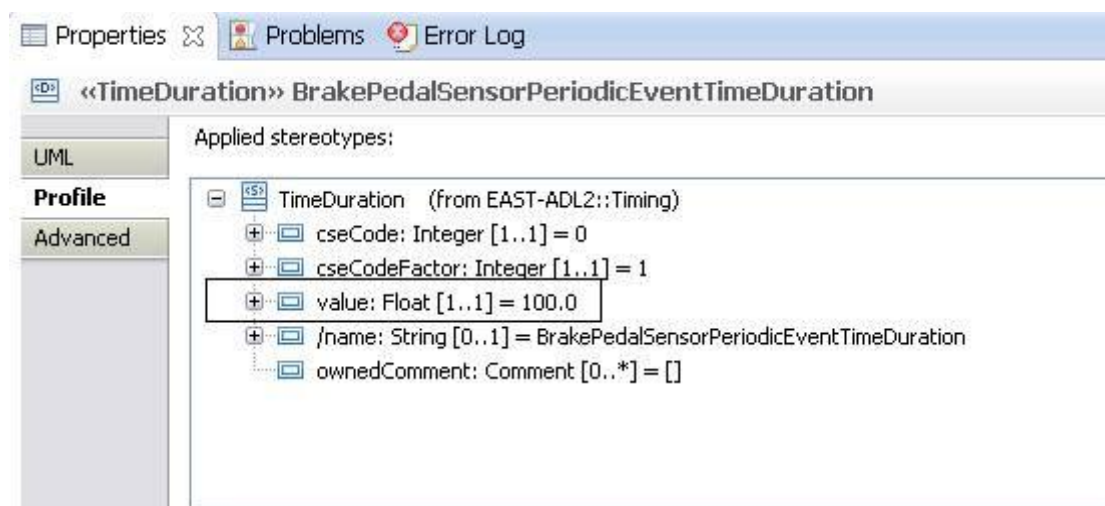


Figure 14. Time Duration Specification

The specification of a Reaction Constraint is shown in Figure 15. Two attributes have to be specified here, *scope* and *upper*. Scope refers to the event chain subject to the constraint, in our case the event chain shown in Figure 10. Upper is again a TimeDuration element, which must specify the value of the deadline.

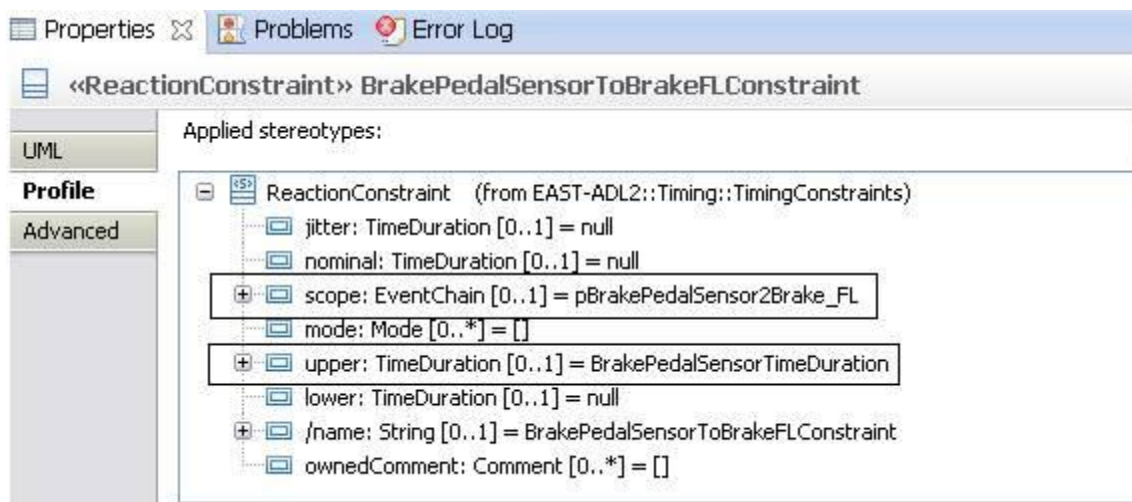


Figure 15. Reaction Constraint Specification

The specification of an Execution Time Constraint is shown in Figure 16. Two attributes have to be specified here, *targetDesignFunctionPrototype* and *upper*. The *targetDesignFunctionPrototype* attribute specifies the function (prototype) the execution time constraint refers to. Upper is again a TimeDuration element, which must specify the value of the worst case execution time of the function prototype.

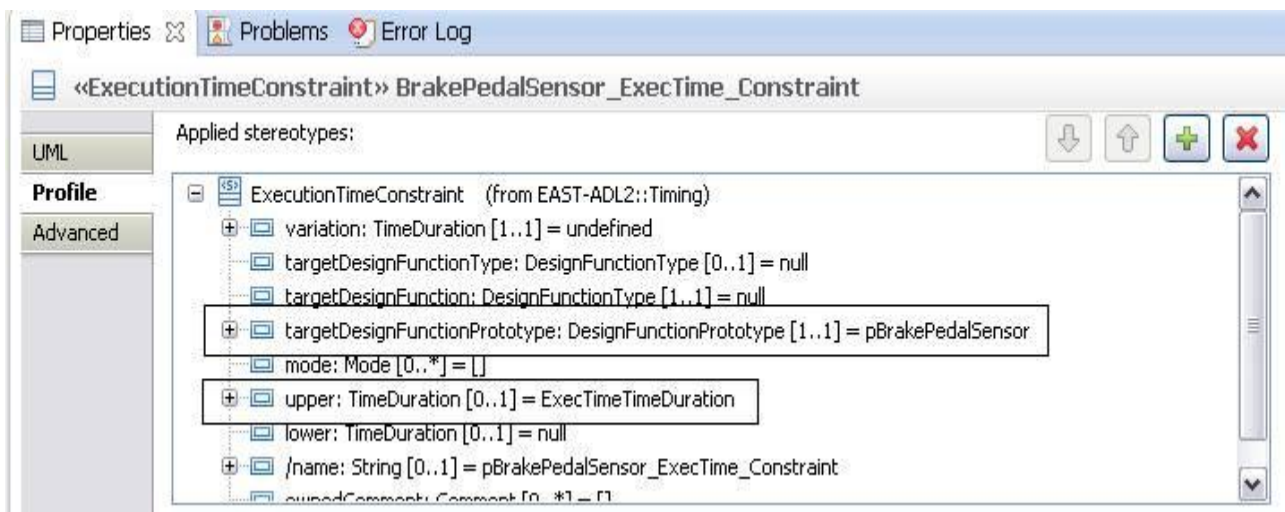


Figure 16. Execution Time Constraint Specification

In order to specify the concrete resources for the execution of functions, allocations must be specified. Once again allocation information is optional, in the sense that allocations can be alternatively specified right after the transformation EAST-ADL/Qompass. Figure 17 shows an allocation of all the functions belonging to the functional design architecture of Figure 9.

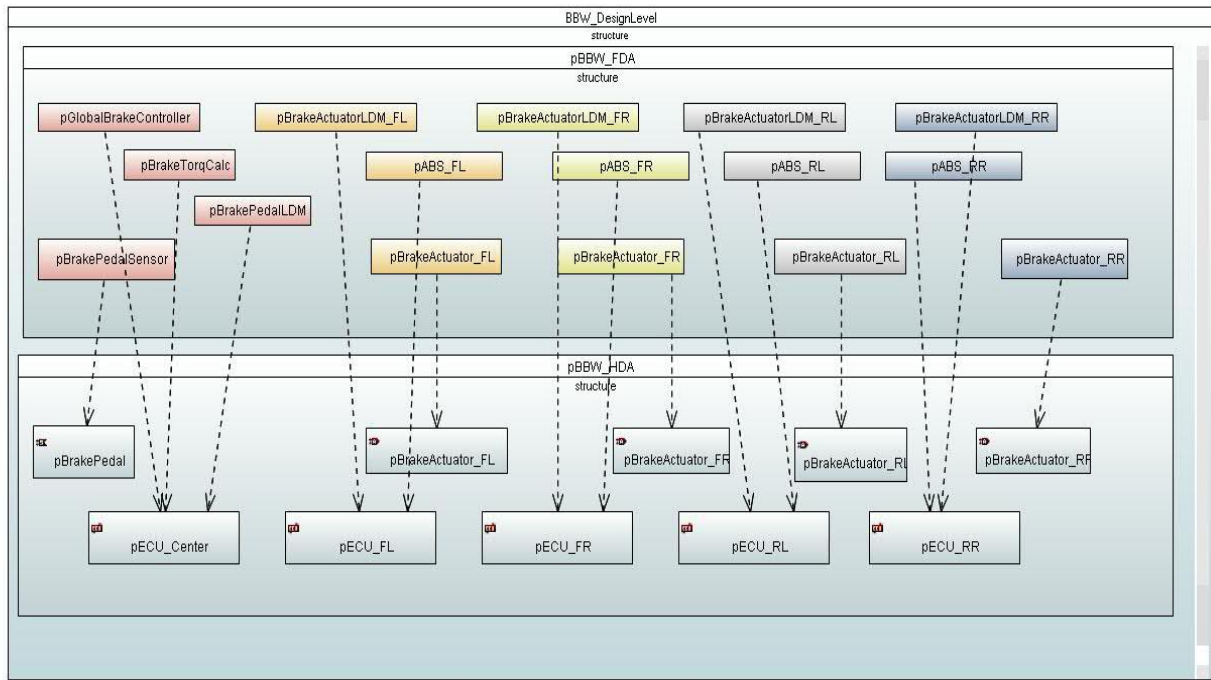


Figure 17. Allocation

In this UML diagram, the FunctionAllocation is represented as a dependency (more technically the FunctionAllocation concept extends the UML Dependency metaclass), through a dotted arrow from the client to the supplier. The two attributes */target* (supplier) and */allocatedElement* (client) are derived, i.e. automatically set when the dependency is established.

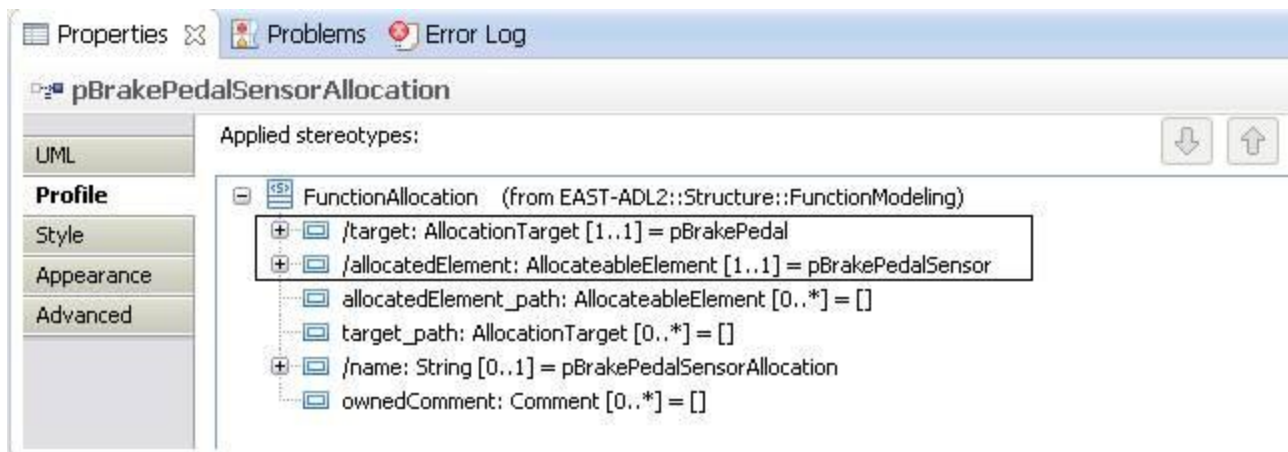


Figure 18. Allocation attributes.

In the case the ensemble of functions is allocated on a distributed platform, the topology of the platform must be specified. In particular the Qompass tool needs to know how nodes are connected through buses. This information is retrieved using the LogicalBus concept. A logical bus must be defined, and in particular all the connectors connecting nodes that can communicate through the bus must be specified in the *wire* attribute, as shown in Figure 19.

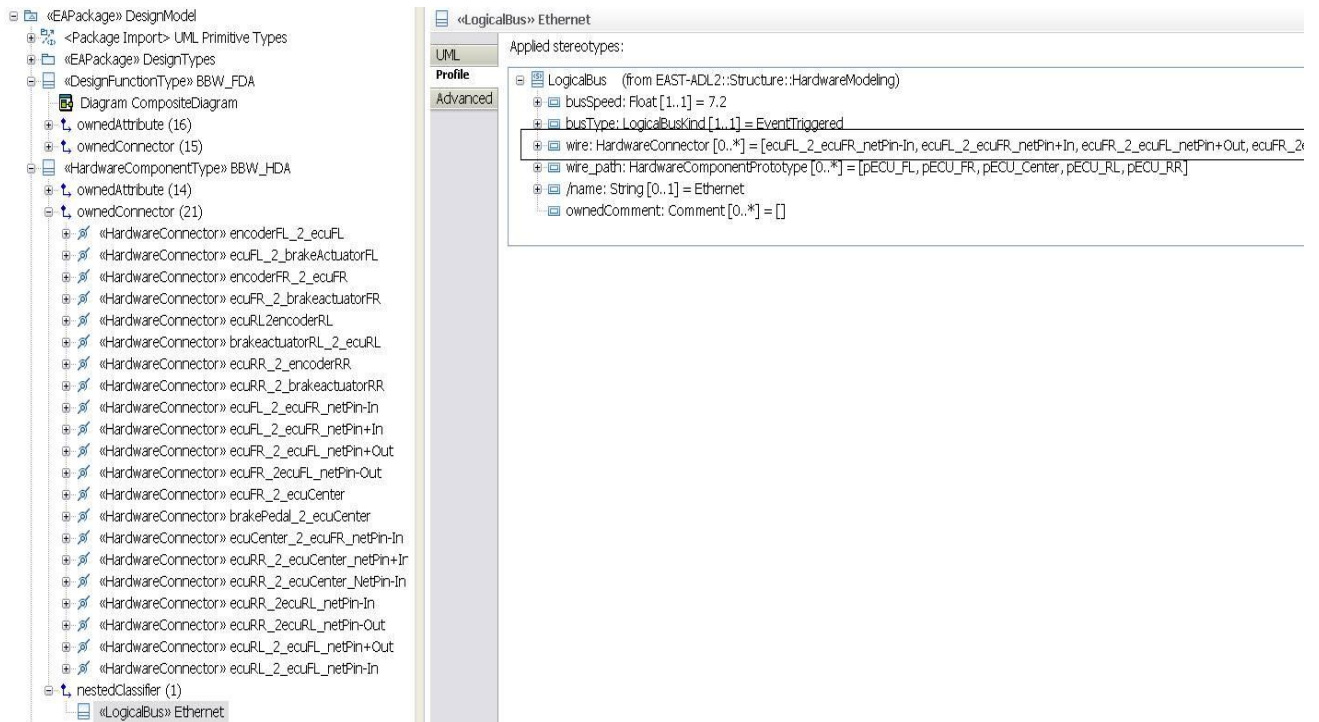


Figure 19. Logical Bus Specification

2.3 Simulink Gateway

2.3.1 Objectives

The Simulink gateway aims to allow a transformation of architecture specification in EAST-ADL, together with the annotated behaviour constraints, to Simulink/Matlab for detailed control design and behaviour analysis through simulation. This makes it possible for the control engineers and the system architects to have a common view about the system specification for traceability management and early verification&validation.

2.3.2 Related Project Requirements

DOW#0112: The Simulink import-export (FAA, FDA) shall be modelled in MAW-Simulink plugin.

2.3.3 Background and Version History

Within the ATESS2 project, a gateway was developed to provide input/output facilities of models in Simulink and Papyrus/Eclipse. The gateway has two parts: 1. a GUI plugin to the MATLAB/Simulink environment, which aids the user in creating models that conform to the format that is needed to being able to convert it into an EAST-ADL model, and 2. an Eclipse plugin, which can convert between the intermediate format of Simulink models and EAST-ADL models. In particular, the GUI plugin for Simulink converts standard Simulink subsystems to system reference blocks, and tags them for conversion to EAST-ADL by putting them in a “FunctionTypes”-library, and assigning a unique ID, to allow bi-directional exchange and updates. Import works the other way around, the function types and prototypes of an EAST-ADL architecture model are imported to empty library blocks in the “FunctionTypes”-library, and instances of them respectively. For the model exchange, the plugin also converts the MATLAB/Simulink models into a custom Ecore-based format.

In MEANAD, two Simulink model transformation plugins have been developed for the MetaEdit+ environment, for a more effective EAST-ADL language verification&validation. The first plugin, developed by MetaCase, parses an EAST-ADL architecture model and creates the .mdl-files of corresponding Simulink models directly. The second plugin, developed by KTH, relies on Matlab API for automatic creation of Simulink and Stateflow models. For an EAST-ADL model, this plugin considers both the architecture specifications and the behaviour constraint annotations.

2.3.4 Key Features

To allow the creation of a Simulink model, the first step is the establishment of an architecture description in EAST-ADL, in terms of FAA (Functional Analysis Architecture) or FDA (Functional Design Architecture). Behaviours can be annotated to the system blocks through the newly defined EAST-ADL Behaviour Annex, for capturing the expected internal state machine behaviours and computations. During the model transformation, a Simulink plugin searches through the EAST-Model for identifying the data, function composition and communication definitions and then setting up the .mdl files or Matlab API instructions for the creation of corresponding Simulink blocks and links. This is shown in Figure 20, where the exported file defines the commands to Matlab API for model creation. When behaviour transformation is also of concern, a Simulink plugin also searches the behaviour constraint annotation for each functional block and thereby creates the Matlab API instructions for the setting up of corresponding StateFlow model. This is shown in Figure 21, where the exported file defines the commands to Matlab API for StateFlow model creation.

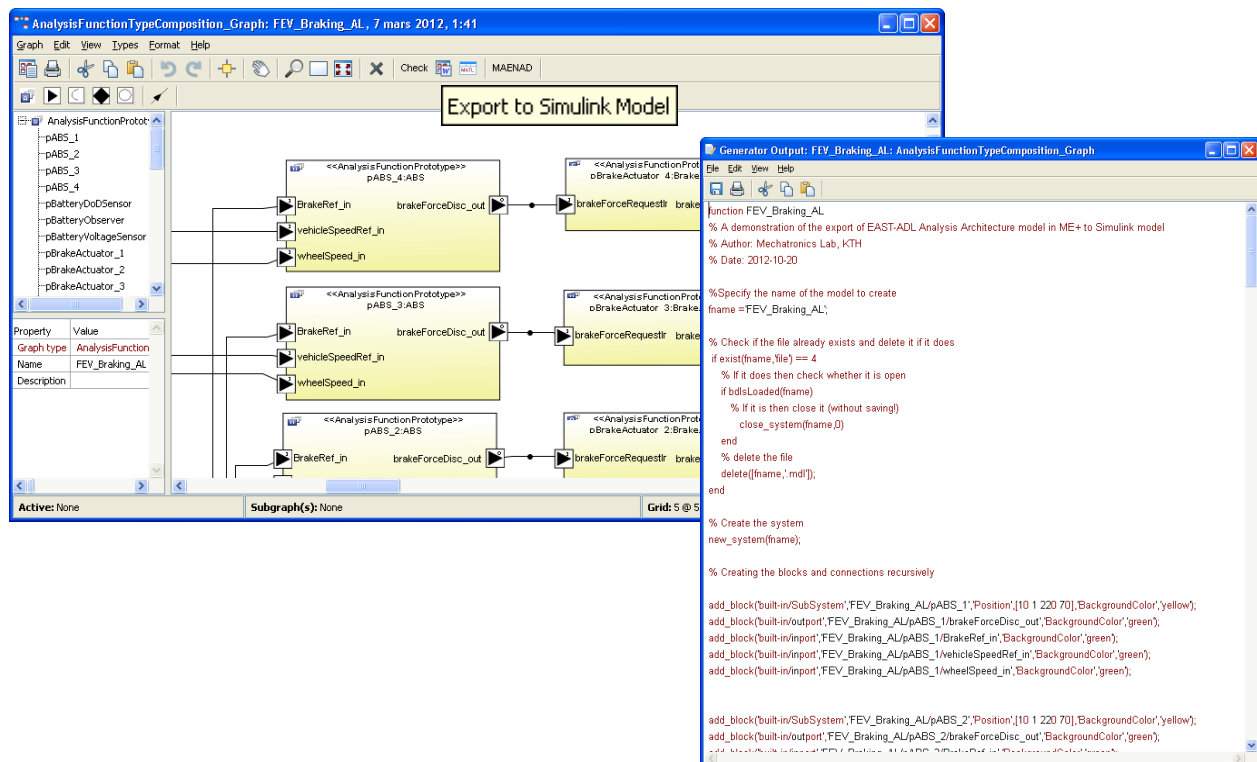


Figure 20. Screenshot of exporting an EAST-ADL architecture model to Simulink.

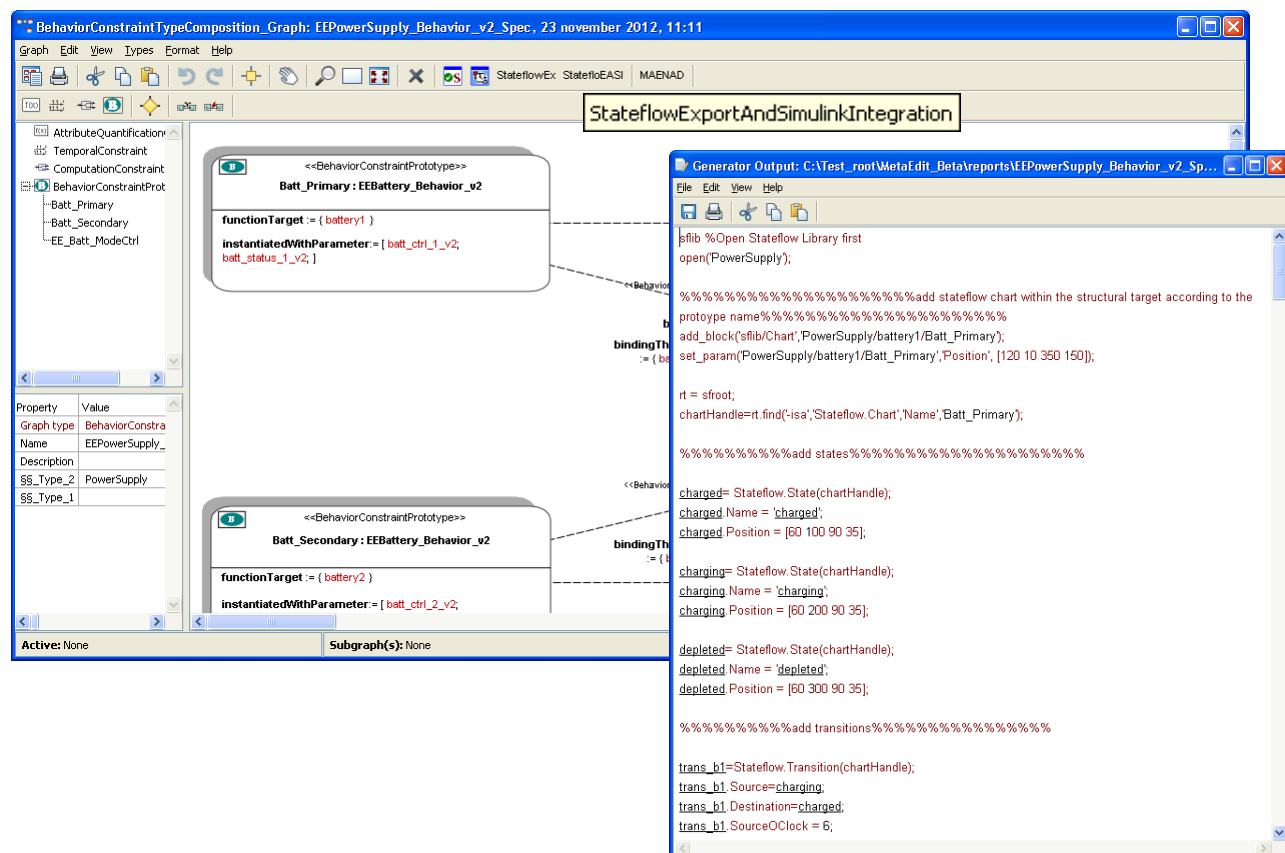


Figure 21. Screenshot of exporting an EAST-ADL behavior constraint annotation to Stateflow.

2.4 HiP-HOPS Safety Analysis Gateway

2.4.1 Objectives

Integrating safety analysis into the development of automotive embedded systems requires translating concepts of the automotive domain to the generic safety and error analysis domain. We assume a model-based development process where automotive concepts are represented by the EAST-ADL architecture description language, which supports system design and safety concepts on multiple levels of abstraction. With the traceability across architecture models and dependability models well managed by EAST-ADL, the HiPHOPS gateway ensures the analytical leverage in regard to FTA/FMEA by transforming the EAST-ADL error descriptions into HiP-HOPS models. With the resulting tight integration of the safety analysis tool and the model-based development environment, the safety engineers can perform the safety analysis incrementally with a seamless integration with requirements and system specification.

2.4.2 Related Project Requirements

UOH#0003: The HiP-HOPS analysis tool should support any ISO 26262 or related concepts (such as ASIL decomposition) necessary to allow ISO-compatible dependability analysis of EAST-ADL models.

UOH#0004: EAST-ADL and HiP-HOPS should be able to intercommunicate by means of model transformations provided by a dependability plugin in the MAENAD Analysis Workbench (MAW). Furthermore it should be possible to import or store the results from HiP-HOPS in the Workbench and/or the EAST-ADL model, which will require establishing some form of (perhaps XML based) interchange format.

2.4.3 Background and Version History

Within the ATESSST and ATESSST2 project, a gateway was developed to transform EAST-ADL error models in and Papyrus/Eclipse to HiPHOPS. Within the MAENAD project, this Papyrus plugin is updated to support EAST-ADL models built using the UML profile version 2.1.10. It uses the concepts defined in the EAST-ADL error-model, but also other language constructs from the FDA level.

Within the MAENAD project, two new gateways to HiP-HOPS that have been developed: 1. a plugin for MetaEdit+, 2. a plugin for EPM. The first plugin is mainly used for investigating a full-fledged coverage of the EAST-ADL dependability modelling concepts. The second plugin serves as the basis of ASIL decomposition export to HiP-HOPS.

2.4.4 Key Features

Along with the EAST-ADL dependability modeling support for the elicitation of safety requirements, EAST-ADL also allows safety engineers to precisely defining the related error behaviors for the purposes of safety analysis through explicit error models. See Figure 22 for an example, where the connection links represent the error propagations due to communication links or allocation relations in the design. To facilitate safety engineering tasks, the MetaEdit+ EAST-ADL tool can automatically create an initial setup of such error models according to a nominal architecture model. Within each error model block, there is a declaration of error behavior (ErrorBehavior) for relating the declared output failures to the declared faults. Currently, this step has to be performed by the engineers. The exact formalism could be based on Boolean logic expression as given in HiP-HOPS or a state-machine (SM) based definition by using the Behavior Constraint Annex.

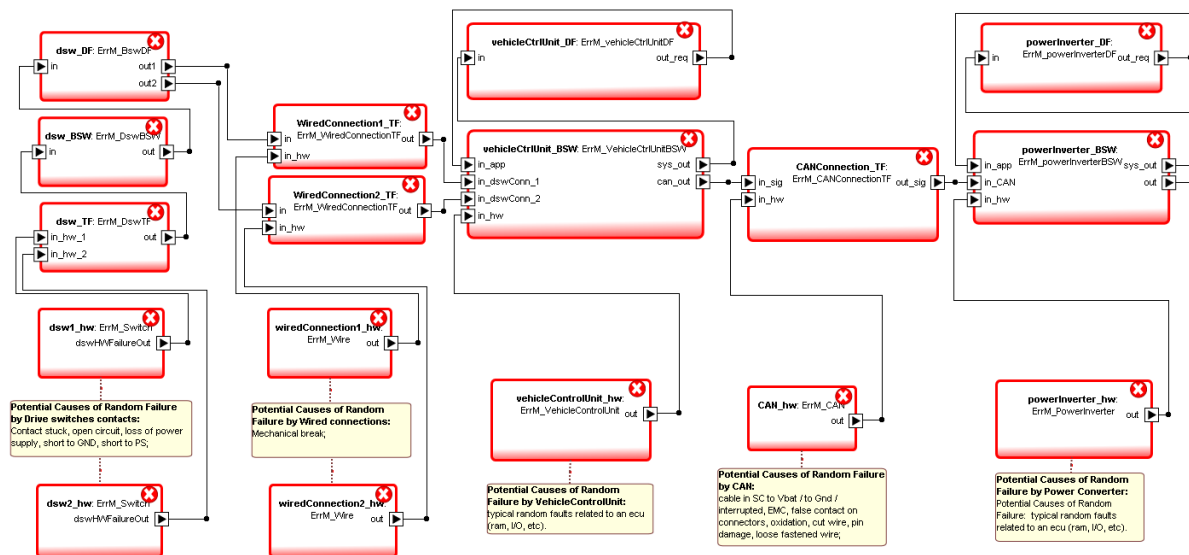


Figure 22. Error model in MetaEdit+ defining the faults and error propagations of target system hardware and functions.

The model transformation plugin parses the EAST-ADL error model for the definitions of failure modes, error behaviors and propagations and thereby synthesizes the corresponding HiPHOPS file for FTA/FMEA. This is shown in Figure 23.

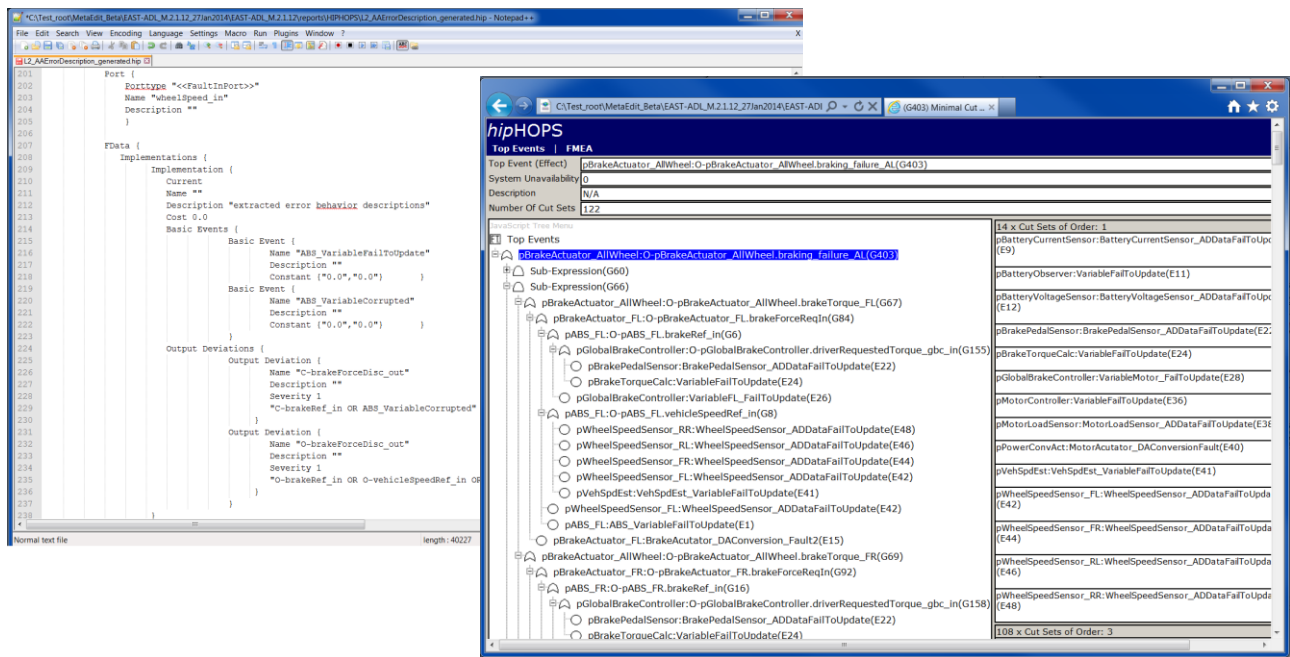


Figure 23. The exported HiP-HOPS file (.hip) and the analysis result in fault-tree.

The second EAST-ADL gateway to HiP-HOPS is an interface developed for the EPM tool by TUB. Unlike the Papyrus interface, which exports to HiP-HOPS via a model transformation, the EPM plugin performs a simpler export of only the relevant error model information to construct the HiP-HOPS input XML file. Because this does not rely so much on the EAST-ADL metamodel, it is less prone to becoming out of date due to version changes in the metamodel (whereas the Papyrus plugin would potentially need updating with each change). See also section 2.6 below for more information.

2.5 Architecture optimization and configuration

2.5.1 Objectives

The architectural optimization&configuration capabilities developed in MAENAD build upon several tools and plugins, including CVM (and the corresponding variability management plugin from ATESS2), EPM (an EAST-ADL component modelling tool with variability management support, built on CVM) and HiP-HOPS. These tools had to be interfaced with an optimization engine for fully multi-objective optimization of EAST-ADL models to be possible. The prototypical implementation of such an optimization engine that was developed during MAENAD is called OptiPAL.

2.5.2 Related Project Requirements

VTEC#UC006: A model of the validator with timing, dependability and cost annotations as well as design space, variability and take rate annotations is defined and exported to EAXML. An optimization tool computes the optimal design for the defined product line. The resulting optimized model is recorded in the model (design space variability removed) and exported in the EAXML file.

UOH#0002: The EAST-ADL error model should fully support automatic optimisation, e.g. through rules that specify a 1:1 mapping from nominal to error models.

UOH#0005: To support multi-objective optimisation, there must be a standardised way of passing design candidates to analysis tools/plugins and receiving results in a given format.

2.5.3 Background and Version History

A proposal for an EAST-ADL optimization architecture was developed over the course of the second year of MAENAD, building upon initial ideas from the meeting in York May 2011. In the third project year, the architecture was then further refined to support product line optimization and the distinction of product line versus design space variability, mainly by introducing the concepts of optimization representatives in addition to optimization candidates. This led to the architecture shown below:

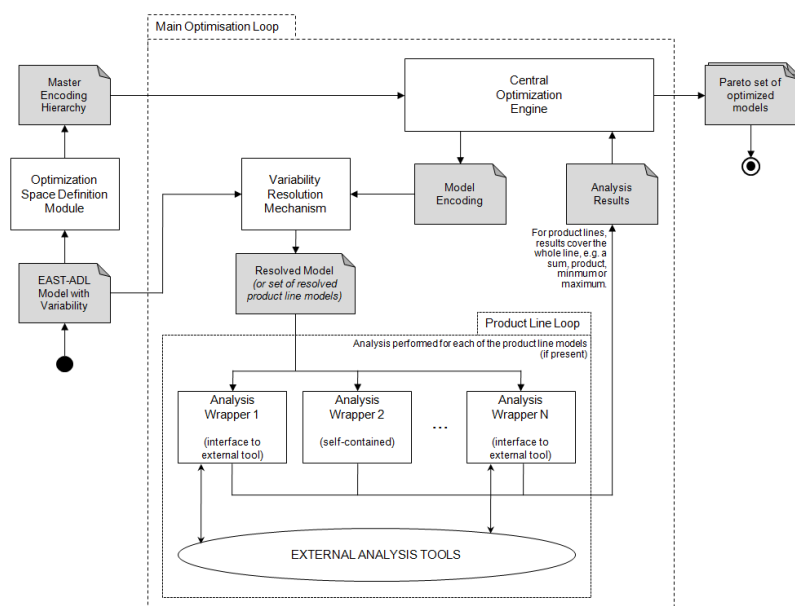


Figure 24. Optimization Architecture (version 1 as of late 2012).

The above optimization architecture was used as a basis for implementation of the MAENAD optimization prototype, called OptiPAL, in 2012 and the first quarter of 2013. The experience with this implementation and with early example modelling led to a number of refinements and improvements of the optimization architecture. The figure below shows the current optimization architecture including these changes:

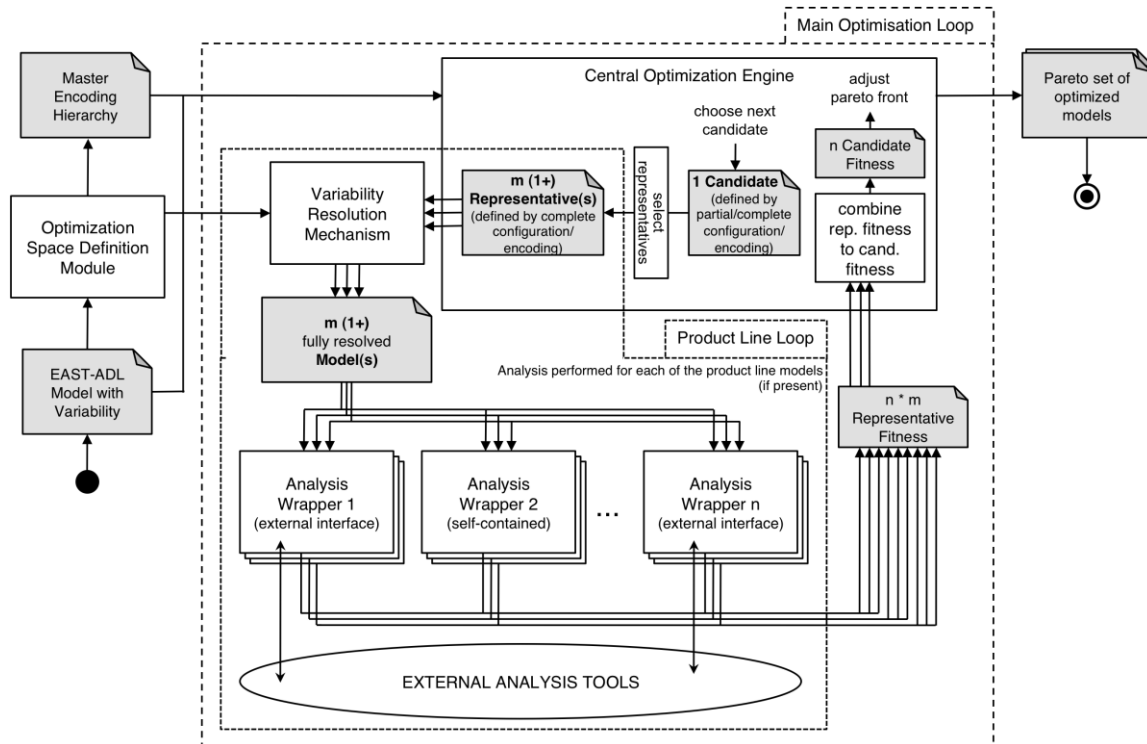


Figure 25. Optimization Architecture (version 2 as of May 2013)

The optimization architecture is described further in D3.2.1, but is intended to serve as a blueprint for the implementation of tool support for the optimisation process. It consists of several major elements, briefly described below:

- **Optimization Space Definition Module (OSDM)**

This module takes a variability-rich EAST-ADL model and generates a 'master encoding hierarchy', which is a hierarchical definition of the design space represented by the model variability. In practice, this takes the form of a feature tree (with slight modifications), and can therefore be generated by variability tools like CVM.

- **Central Optimization Engine (COE)**

The COE is the driver of the optimization process. It is responsible for exploring the optimization design space on the basis of heuristic algorithms such as genetic algorithms. It generates an encoding for a particular design candidate, which is then resolved by the VRM (see below) and evaluated by analysis plugins to determine its relative score in each of the objectives being optimized (e.g. reliability, performance, cost, energy consumption etc.). When these analysis results are returned back to the COE in the form of a candidate fitness value per objective/analysis wrapper, optimal candidates are preserved by the COE while sub-optimal dominated designs are discarded. Once the process is complete, the COE will generate a report containing the set of pareto-optimal design candidates.

- **Variability Resolution Mechanism (VRM)**

The COE does not manipulate the EAST-ADL model directly. Instead, it modifies encodings (essentially feature trees), and then passes each encoding to the VRM, which is responsible for

resolving the variability in the original model according to the encoding in order to produce a new model — a design candidate — that can then be analysed and evaluated.

- Analysis modules

For each objective being analysed, there needs to be a corresponding analysis module. The intention is that these analysis modules can be either external tools (such as HiP-HOPS or timing analysis tools like Qompass) or plugins written for the modelling/analysis environment (e.g. Papyrus, MetaEdit+, EPM etc.). The optimisation architecture does not necessarily interact with them directly; instead there should be a common approach, implemented by 'wrapper' objects if necessary, to present a consistent interface to the analysis modules. The hope is that this will allow new analysis modules and thus new objective types to be added (or removed) from the optimization process without requiring modification of the main optimization elements (i.e., the OSDM, COE and the VRM).

In the first version of the optimisation prototype, support for product line variability had been left to be added at a future date. This has been done in early 2013 and led to the second version of the optimisation architecture. We now summarize the main changes in this second version:

- Representatives

A candidate is a certain design variant to be evaluated during optimization. In the non-product line case when all variability in the system model is design space variability, each candidate is defined by a complete configuration and corresponds to exactly one fully-resolved system model. On the other hand, when some variability in the system model is product line variability, then each candidate is defined by a partial configuration and corresponds to a partially-resolved model (or, in other terms, a set(!) of fully-resolved models). As analysis must always take place on a fully-resolved system model, the candidate itself is not sufficient for evaluation of its fitness; instead, one or more so-called representatives have to be selected for the candidate. A full explanation of the concept of representatives with detailed examples can be found in deliverable D3.2.1.

- Selection of Representatives

The representatives for a candidate are obtained by completing the partial configuration that defines this candidate. "Completing" a partial configuration means configuring all those variation points in the configuration that are still undecided. There are different strategies for selecting the representatives to be evaluated for a given candidate, for example full evaluation (all representatives are selected), random selection of a certain number of representatives or selection of the same representatives for all candidates. This selection of representatives is done by the COE, because it is very similar to the selection of a candidate and therefore algorithms and implementations of candidate selection can be reused within the COE.

- Combining Representative Fitness into Candidate Fitness

Assuming the COE has selected m representatives $R_1 \dots R_m$ for a given candidate C , then m fully-resolved models (one per representative) will be provided by the VRM, these will each be sent to the n analysis wrappers (assuming we have n objectives/analysis wrappers) leading to $m \cdot n$ individual fitness values for each representative and for each objective. In the end, the optimization is not about representative but about candidates, and therefore for each of the n objectives the m representative fitness values have to be combined into a single candidate fitness value, leading to n candidate fitness values (one per objective/analysis wrapper). Again, there are different strategies for combining the fitness of m representatives into the overall fitness of the candidate; a typical example might be a weighted arithmetic mean. In the non-product line case, m equals 1 and version 2 of the architecture becomes very similar to version 1 (i.e. there is exactly 1 representative per candidate which is defined by the same complete configuration as the candidate).

- Instantiation of Analysis Wrappers

In addition to introducing representatives, version 2 of the architecture has a number of other improvements and clarification. The most important of these is the notion of instantiation of analysis wrappers. This means that a single analysis wrapper may be used more than once within a concrete optimisation scenario, for example if the analysis implements some generic abstract functionality that can be configured to realize several distinct, concrete objectives. An example might be a summation analysis that sums up the value of a particular user attribute; such a generic summation analysis could then be instantiated twice within a single optimization using one user attribute for cost in the first case and another for energy consumption in the second case. This is only possible if analysis wrappers are not simply selected during optimisation but if they are instantiated and each instance can be configured differently from the other instantiations of the same analysis wrapper.

An initial prototype tool, OptiPAL, was developed by TUB in 2011/2012 (version 1) and late 2012 and 2013 (version 2) and builds upon the CVM/EPM platform. It implements both the OSDM and VRM as part of the existing CVM plugin and also includes a new prototype COE. The COE is presently only a simple experimental version and does not implement the full genetic algorithm for optimization yet, but it does allow for generation of different encodings and thus allows the main optimization loop to take place. Analysis is provided by an OptiPAL cost analysis plugin (which currently only does simple cost summations) and a bridge to the HiP-HOPS safety analysis tool for dependability analysis via FTA.

OptiPAL is primarily intended as a proof of concept and as a way of testing out the optimization concepts on test models, to provide feedback to facilitate further development of the optimization architecture concept. Should it prove successful, however, it may evolve into a more fully functional optimization tool.

Current OptiPAL may be upgraded and extended further. The first step might be to implement full genetic algorithms in the COE module, to allow a more efficient application of the implementation to real-world models. As tool support for other analyses matures, it may also be possible to create new interfaces to other tools or additional OptiPAL-based analysis plugins, e.g. for other FEV-related objectives such as cable length, battery life, or energy consumption etc.

Finally, it would be worthwhile to further investigate the feasibility of the basic notion of product line optimization. The result of the experiments in WT3.2 and WT3.3 provides a solid basis and clearly defined concepts for dealing with product line variability in an optimization context (mainly the distinction between representatives vs. candidates).

2.5.4 Key Features

As a basis for the implementation of the optimization prototype called OptiPAL, the EPM component editor has been chosen. While this editor implements only a subset of EAST-ADL, it provides a good basis for the optimization prototype because it already contains full support for variability management and configuration (making use of the CVM framework) and the optimization architecture relies on variability modelling concepts for defining the optimization space, as detailed above. In addition, EPM provides flexible extension mechanisms that allows for a tight integration of the OptiPAL prototype with the basic modelling capabilities of EPM.

The input for an optimization with OptiPAL can be summarized as follows:

1. A variant-rich EPM model. System variations defined in this model comprise (a) product line variability and (b) design space variability. For the purpose of optimization, the second form of variability defines the optimization space while the first form requires special treatment during optimization. For more details refer to deliverable D3.2.1. For the purpose of the OptiPAL prototype and EAST-ADL validation within the MAENAD project, this EPM model can be perceived as an EAST-ADL model, because the EPM meta-model is sufficiently close to the core package of the EAST-ADL domain model.

2. An optimization scenario specification. This defines precisely how to conduct the optimization, for example how many optimization cycles to perform and which external tools to use for candidate evaluation. The following information is part of such an optimization scenario specification:
 - a. Selection of one optimization engine. List of available engines depends on what engines are installed.
 - b. Selection of one or more optimization objectives. Each such optimization objective is an instance of one of the installed analysis wrappers (cf. optimization architecture above). Here, “an instance” means that a single analysis wrapper may be used twice or more within a single optimization scenario in order to realize several distinct objectives that can be evaluated with the same analysis wrapper but with different configurations.
 - c. For each engine and objective: an assignment of values to customization parameters of the corresponding engine / analysis wrapper.
In other words: each OptiPAL-compatible engine and each OptiPAL analysis wrapper declares a set of customization parameters allowing to customize the precise behaviour of the engine/analysis during optimization; the optimization scenario specification then provides value assignments for the parameters of the selected engine and each selected objective’s analysis wrapper. If a single analysis wrapper is used more than once for multiple objectives, then distinct parameter assignments can be provided for each objective, i.e. each instance of the analysis wrapper.

Figure 26 shows a screenshot of the optimization scenario specification editor in OptiPAL.

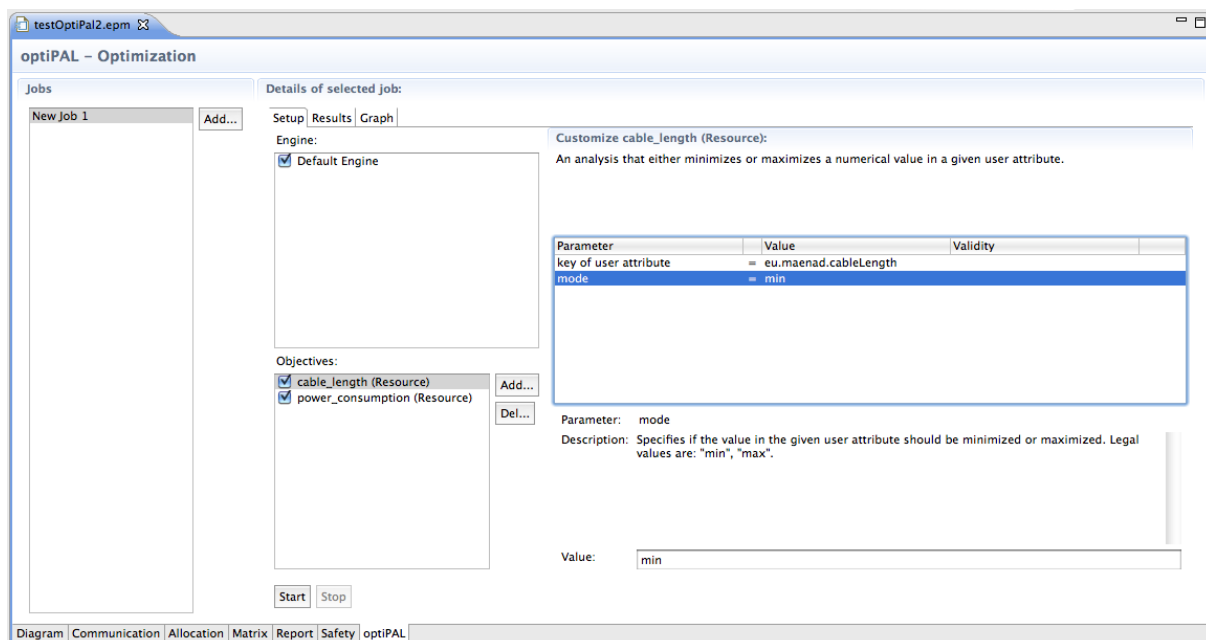


Figure 26. Screenshot of an optimization scenario specification in OptiPAL.

Given an optimization scenario specification as described in the previous section, OptiPAL can automatically conduct the entire optimization, i.e. start external tools through analysis wrappers for each objective, sending candidates to these external tools for evaluation, receiving fitness values, etc. The result of such an optimization is a set of pareto-optimal candidates. The presentation of these results in the OptiPAL prototype has been kept fairly simple, because presentation and visualization was not within the research focus of the MAENAD project. OptiPAL provides a very simple graphical presentation of the pareto front (see following figure).

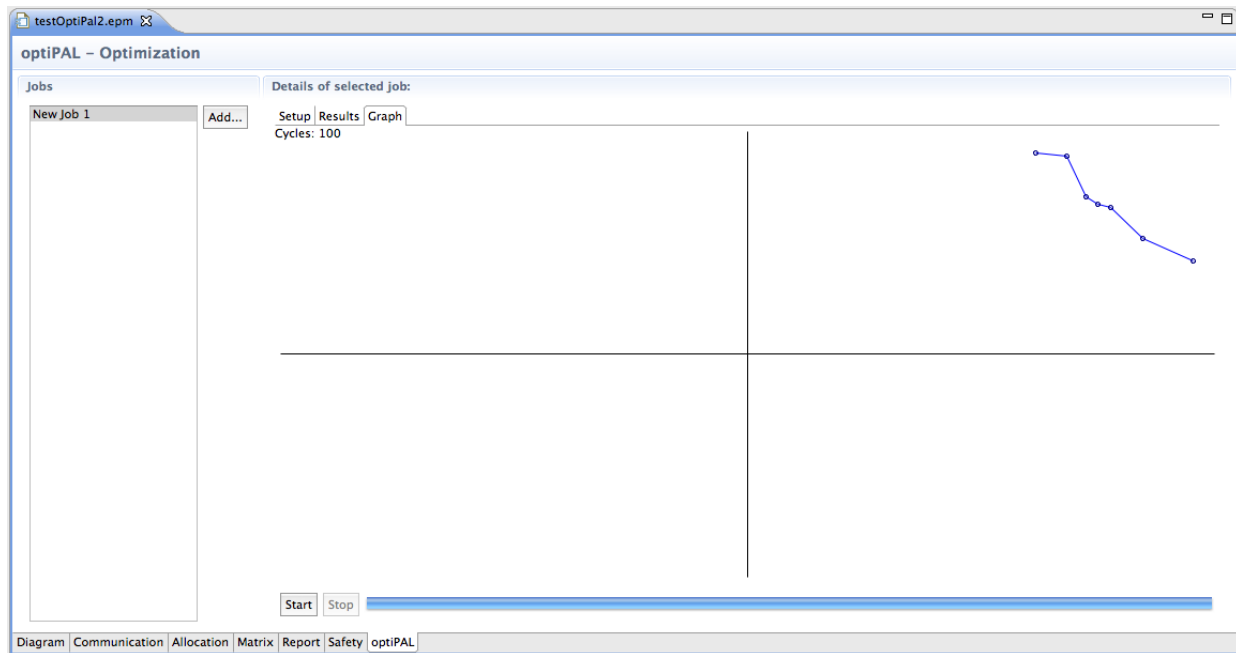


Figure 27. Graphical representation of a pareto-front in OptiPAL.

And the full details of each pareto-optimal candidate is presented in a very simple textual form, as shown in the below figure. This textual output was not tailored to readability but to ease of implementation.

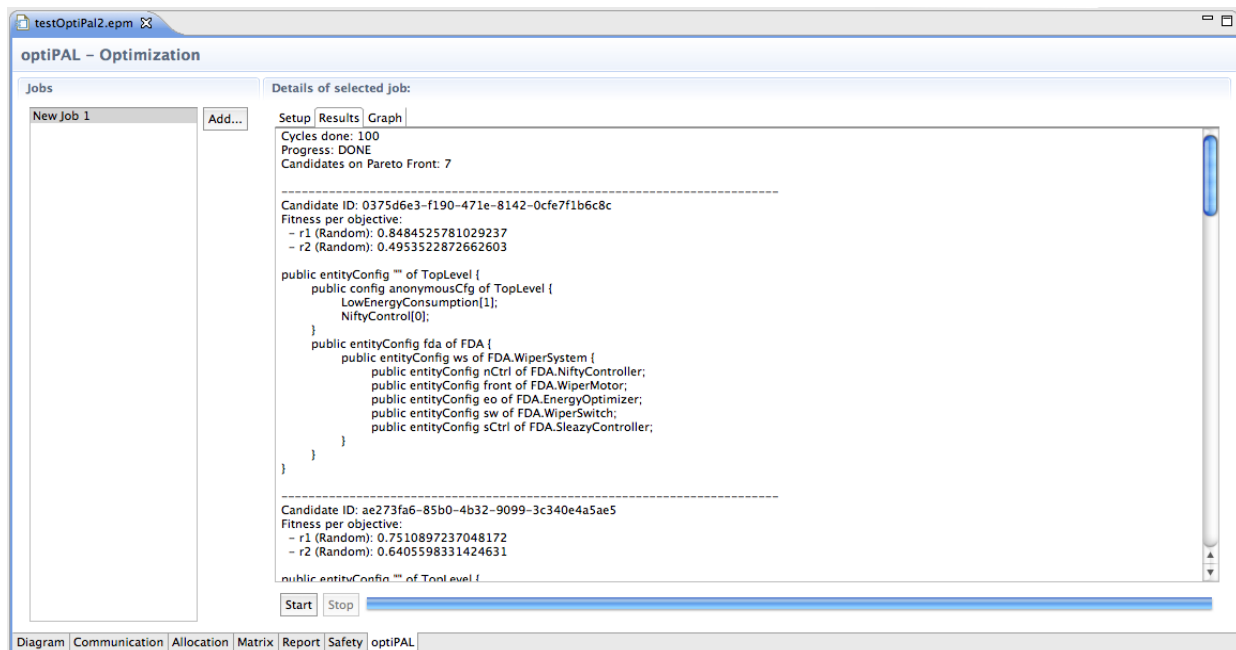


Figure 28. Textual representation of optimization results in OptiPAL.

2.6 ASIL allocation with EPM/HiP-HOPS

2.6.1 Objectives

The overall implementation of the ASIL allocation consists of these two parts:

1. Implementation of the actual ASIL allocation algorithm in HiP-HOPS.
2. Extension of EPM and its HiP-HOPS export to provide modeling and editing support for the additional information required in the model specifically for ASIL allocation.

The first part in HiP-HOPS is the main implementation while the part in EPM mainly provides a front-end to conveniently feed the ASIL allocation algorithm in HiP-HOPS with input data. The algorithm for ASIL allocation has been described in full detail in deliverable D3.2.1, so please refer to that document for an explanation of the ideas behind the related concepts. For evaluation and demonstration purposes, a combination of HiP-HOPS and the EPM component modeling tool has been chosen as a basis. EPM covers the relevant parts of the EAST-ADL domain model in sufficient detail and had the advantage of already supporting a model export to the HiP-HOPS tool as a means for external model analysis.

2.6.2 Requirements from WT2.1: Identification of needs

The work on ASIL decomposition covers MAENAD objective O1-2 “Automatic allocation of safety requirements (ASILs)”.

2.6.3 Background and Version History

As of MAENAD milestone MS7, the ASIL allocation has been implemented in HiP-HOPS and related modelling and editing support has been provided in EPM. The implementation has been tested on smaller models and a larger model immediately taken from industrial practice (from Continental). The support of ASIL decomposition at the end of project year 2 was already on a level of what was aimed for in the MAENAD DoW, and it was possible to perform decomposition of ASILs via HiP-HOPS. Apart from minor fixes and corrections, no additional work has been conducted during year 3 (in accordance to planning). Longer term development work by UOH on improved ASIL decomposition algorithms is still underway, but lies beyond the MAENAD project objectives.

2.6.4 Key Features of ASIL allocation

Several additional modelling elements and attributes had to be added to the data model, together with editing support in the user interface:

- Hazards, with:
 - Name
 - Safety Requirement (the ASIL)
 - Severity
 - Logic (i.e. an expression defining what failure will cause the hazard).
- RiskTime
- Unavailability Formula

The following two screenshots show how this has been realized in the tool. In addition to extending the meta-model and providing editing support, the HiP-HOPS export in EPM had to be extended to take care of this information and to properly provide it to EPM.

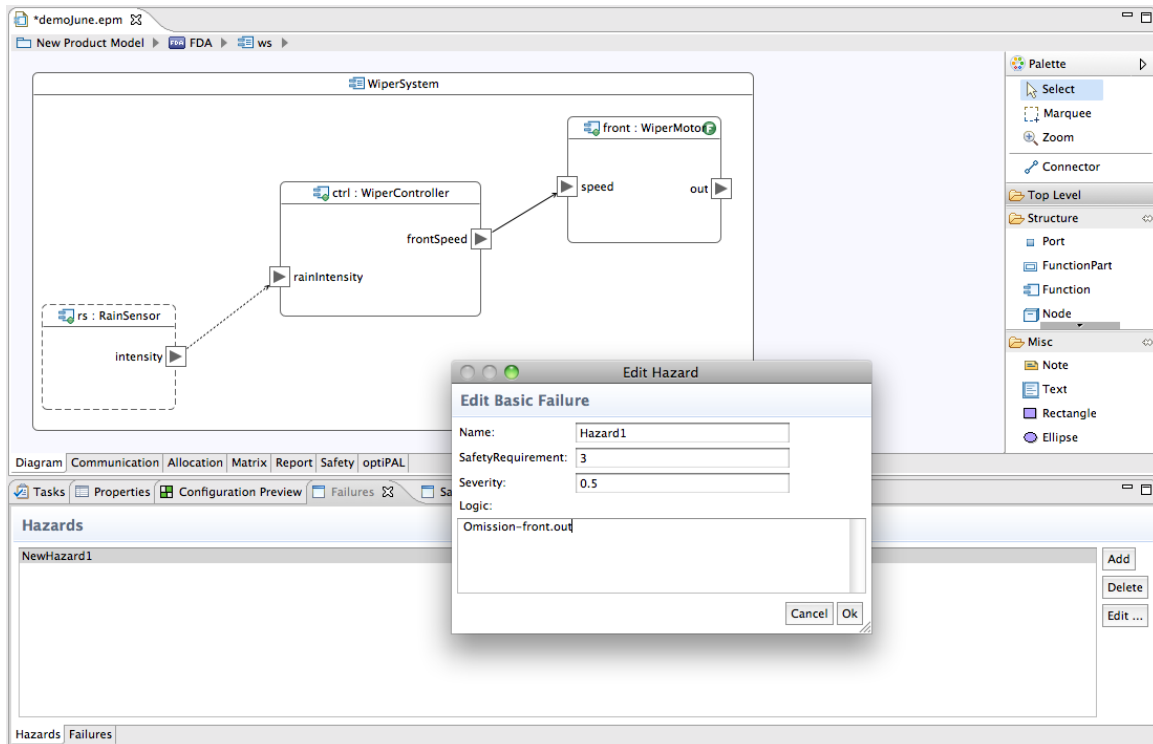


Figure 29. Editing Hazards in EPM.

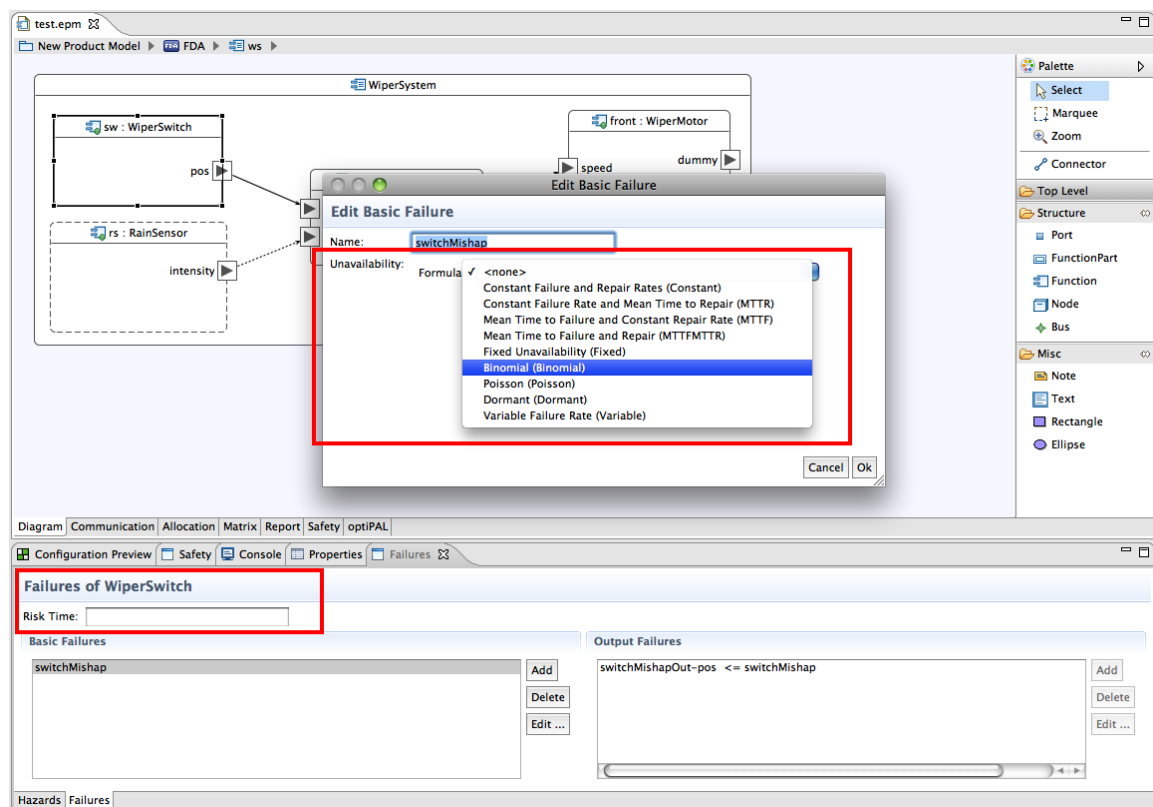


Figure 30. Editing Risk Time (bottom left) and Unavailability Formular (center) in EPM.

2.7 UPPAAL/SPIN Model-Checking Gateway

2.7.1 Objectives

For formal analysis of EAST-ADL behaviour constraint specification, the model transformations to two external well-known model checkers are supported. Through these external tools, the users of EAST-ADL can exhaustively verify an EAST-ADL behaviour constraint specification in regard to temporal properties of concern, including reachability (i.e. some condition can possibly be satisfied), safety (i.e. some condition will never occur), and liveness (i.e. some condition will eventually become true), for the purposes of requirements engineering, compositionality control, and cross-level conformance check, etc.

The primary target model checker of this MAENAD work is UPPAAL. The basic building blocks of UPPAAL models are asynchronous processes in terms of timed-automata. UPPAAL uses the concept of broadcast channels for synchronizing more than two processes. UPPAAL distinguishes the types of process definitions (referred to as templates) from their instantiations (referred to as process) inside a system. A subset of CTL (computation tree logic) is used as the query language in UPPAAL for verification. This means, each template definition can be instantiated multiple times with different parameters. Within MAENAD, similar transformation concept has been studied in regard to another model-checker, SPIN. The basic building blocks of SPIN models are asynchronous processes in terms of finite state automata. SPIN use buffered and rendezvous message channels, as well as synchronizing statements, for synchronizing more than two processes.

2.7.2 Related Project Requirements

This analysis support addresses the following requirements:

- DOW#0012 O2-2: Behavioral Simulation of EAST-ADL models
- DOW#0017 O4-2: Evaluation of dependability & performance analyses

2.7.3 Background and Version History

Within the MAENAD project, an EAST-ADL language package, referred to as Behavior Constraint Description Annex, is developed to allow an explicit description of various behavioral concerns in EAST-ADL. On the basis of a formal semantics, several transformations from EAST-ADL to behavior analysis tools have been developed, including UPPAAL, SPIN and Stateflow. The tool demonstration is based on UPPAAL transformation mainly due to its rich semantics and user friendliness. A preliminary tool prototype for the SPIN transformation has however also been developed.

2.7.4 Key Features

With the EAST-ADL Behavior Description Annex, the developers capture and formalize various behaviour concerns during the phases of requirements engineering, architecture design, verification and validation, safety engineering. In Figure 31, the behaviour model for a battery management subsystem is shown. There are two blocks for the behaviours of individual battery functions and one block for the battery controller. These behaviour blocks are connected by four behaviour binding channels with shared variable semantics. Figure 32 shows the declaration of temporal constraint of battery in EAST-ADL state-machine description, which is done within the behaviour constraint block of ABS function. Within the same behaviour constraint block, there are

also attribute-quantification constraints specifying state invariants and guard conditions as well as computation constraint specifying the effects of transitions.

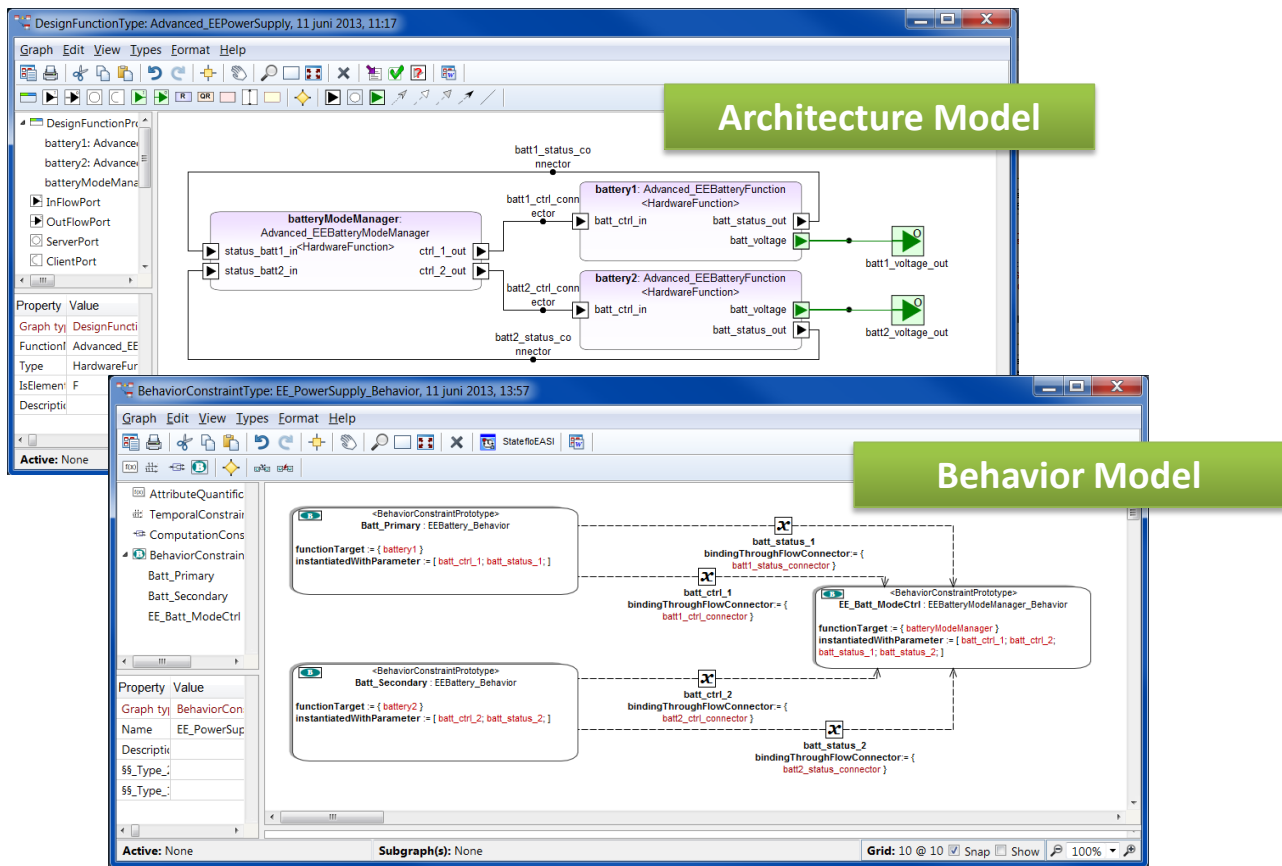


Figure 31. Screenshot of a top-level behaviour description for an architecture model.

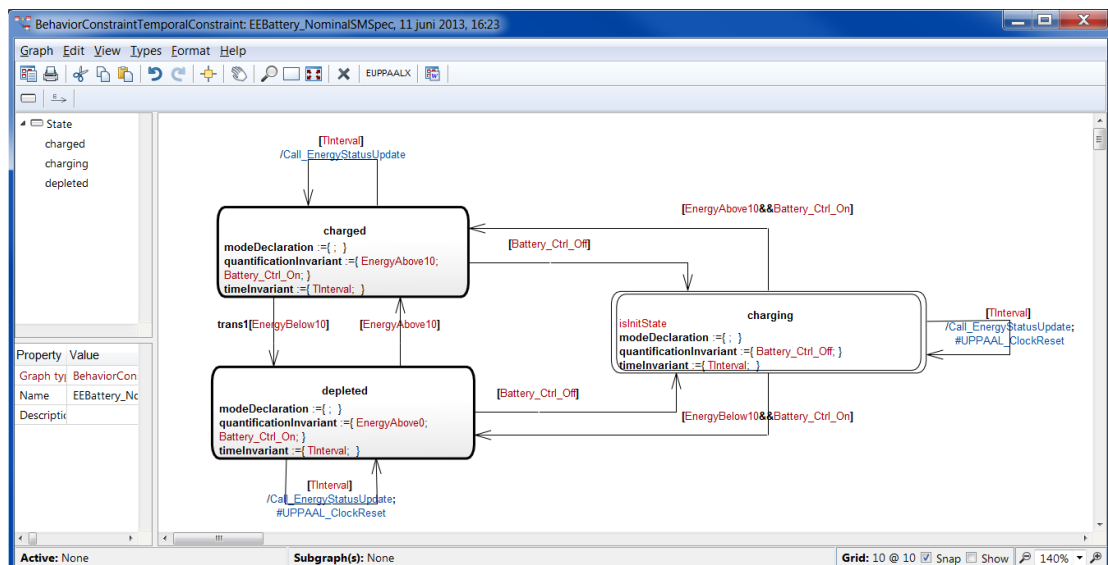


Figure 32. Screenshot of a temporal constraint declaration.

The model-checking gateway parses a top-level behaviour description in EAST-ADL by identifying the declarations of state-machines and binding channels and thereby producing expected external models. The mapping rules are implemented as an algorithm in MetaEdit+ script language. See Figure 33.

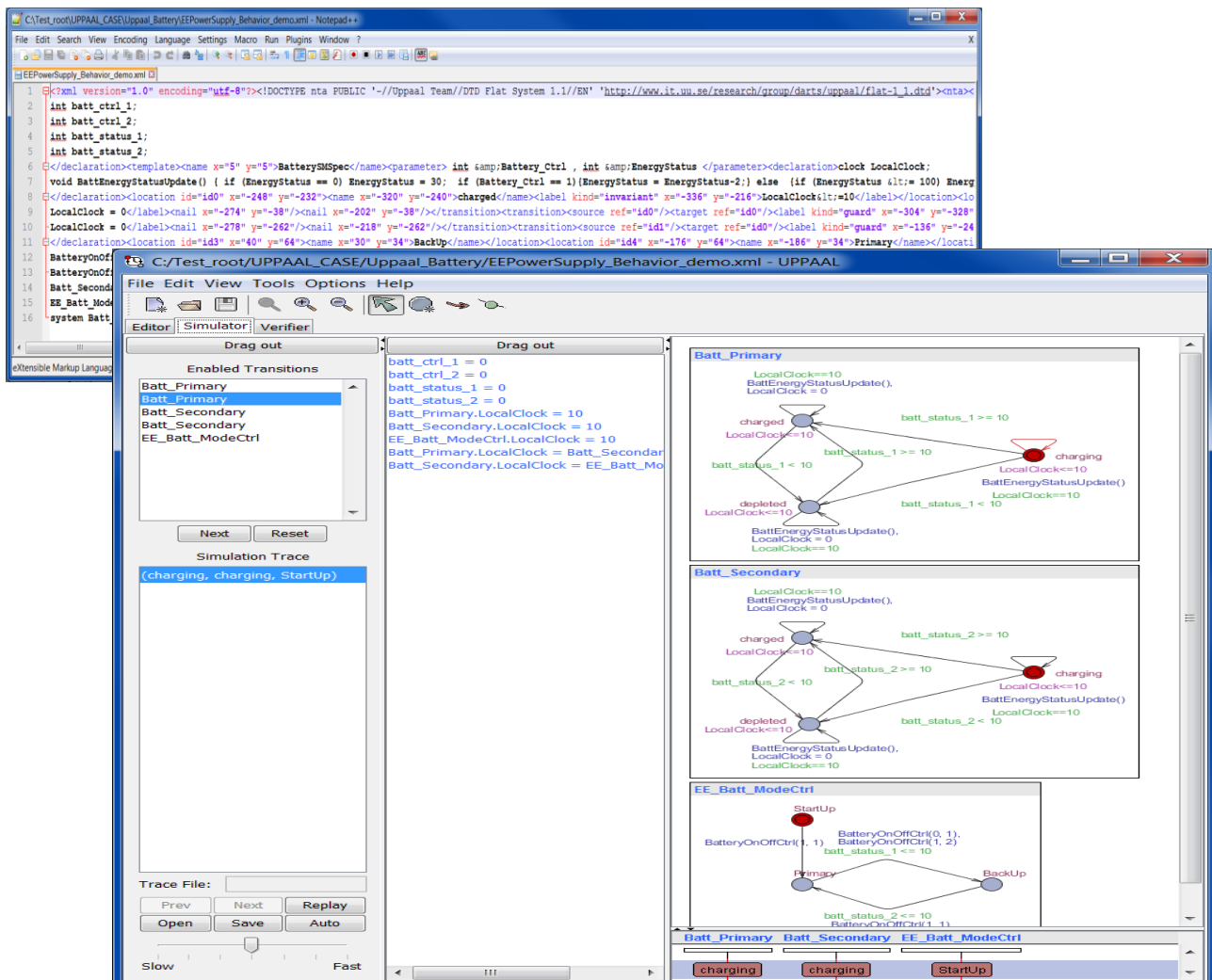


Figure 33. Screenshot of the generated UPPAAL file and model.

2.8 EATOP Analyzer

2.8.1 Objectives

EAST-ADL supports annotation of non-functional properties such as cost, power consumption or cable length. The EATOP Analyzer allows such annotations to be analysed.

2.8.2 Related Project Requirements

This analysis support addresses the following requirements:

- DOW#0017 O4-2: Evaluation of dependability & performance analyses
- VTEC#UC006 Model optimization

2.8.3 Background and Version History

The Analyser was initially defined by VTEC and implemented in a project with Chalmers University of Technology. It was subsequently refined and ported to EATOP for EAST-ADL 2.1.11 and 2.1.12. The EATOP Analyzer will evolve beyond MAENAD to support additional analysis concerns and to refine the presentation of results.

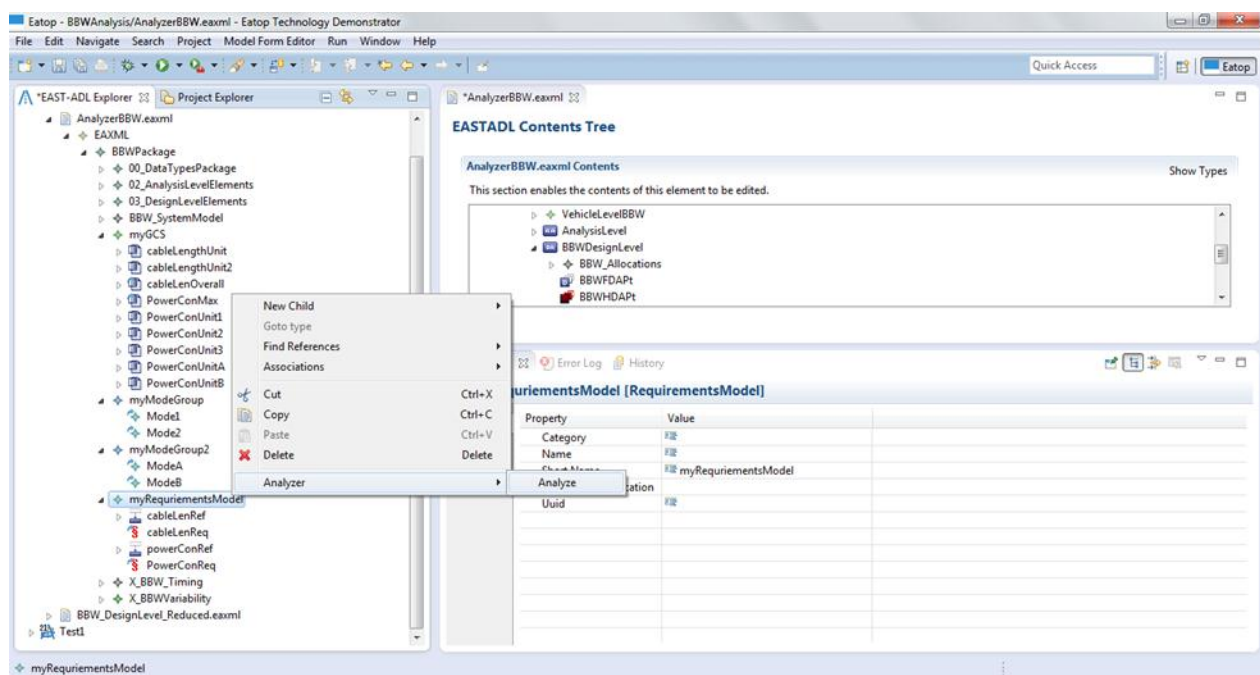


Figure 34. Screen shot on the invocation of the Analyzer plugin.

2.8.4 Key Features

The Analyzer plugin is able to compute analysis result for relevant GenericConstraintTypes of EAST-ADL. The Analyzer expects requirements, with a GenericConstraint as Refinement. These GenericConstraints are taken as the required maximum sum for each affected composition.

GenericConstraints annotating the types used in the composition are summed to form a total value for the analysis. GenericConstraints as well as the Requirements can be linked to modes, and modes and mode groups are respected during analysis. The output is presented both as a value value, and with an VVActualOutcome element in VerificationValidation.

2.9 Variability Resolution

2.9.1 Objectives

The purpose of the variability resolution plug-in is to provide an artifact-level Variability Resolution Mechanism (VRM) for variant-rich EAST-ADL models. It is intended as an artifact-level complement to the already existing feature-level variability resolution tooling.

2.9.2 Related Project Requirements

This variability resolution plugin addresses the following requirements:

- 4SG#0039: Variability of EV architectures
- TUB#0001 Variability Validation
- VTEC#UC006 Model optimization

2.9.3 Background and Version History

The plug-in is currently still in an early development stage; while most of the core variability resolution functionality has been implemented and successfully tested against test models, several development concerns are currently still ongoing.

Once all core concerns and secondary functionality has been implemented and the plug-in has been exhaustively tested, it is intended to be ported to the EAST-ADL Tool Platform EATOP. Once this step has been achieved, the plug-in shall be extended by additional functionality in order to access CVM based feature-link variability support, thus providing widespread support for the entire EAST-ADL variability vocabulary. This step will then provide a basis for extensive artifact-level optimization efforts, possibly including representative-based artifact-level optimization of not-yet-configured product line models.

2.9.4 Key Features

The plug-in essentially resolves all variability logic within one specific variability context of a model, for one context at a time. For that purpose, the contents of all variation groups of all configurable containers of the current context are read into an internal data model and are then solved by applying suitable resolution algorithms for the respective variability dependency kinds. The resulting configuration matrix then also constitutes the proper Master Encoding Hierarchy (MEH) of the respective variability context and can in the following be used in order to generate all actually viable system configurations.

The plug-in is using the standardized EAST-ADL interchange format EAXML for the variant-rich input models. EAST-ADL Variability constructs are used to define variability. Feature models and SelectionCriteria are currently not considered. The output models are captured as one EAXML for each resolved model.

2.10 MODELISAR Functional Mock-up Unit Import

2.10.1 Objectives

The Functional Mock-up Interface (FMI) [5] is a standard that many simulation tools use for model exchange and co-simulation. Through the notion of Function Mock-up Units (FMUs), it defines the input and output variables of each unit and also the data types of these variables. The aim of this tool implementation is to support the creation of architectural models in EAST-ADL from external behaviour models.

2.10.2 Related Project Requirements

MODELISAR FMU import is not explicitly mentioned in the requirements, although it relates to behavioural simulation in general.

- DOW#0012 O2-2: Behavioural Simulation of EAST-ADL models
- 4SG#0003: Perform behavioural Simulation of EAST-ADL models according to performance evaluation standards
- 4SG#0004: Perform behavioural Simulation of EAST-ADL models according to standards covering communication with infrastructures
- 4SG#0019: The project shall enable to perform behavioural simulation according to ISO 8715: Electric road vehicles - Road operating characteristics

There are more similar requirements on behaviour simulations, see e.g.

4SG#0020, 4SG#0021, 4SG#0022, 4SG#0023, 4SG#0024, 4SG#0025, 4SG#0026

2.10.3 Background and Version History

This is a new tool implementation developed in MAENAD. It focuses on the model import aspect based on the FMI. Export from EAST-ADL models in the form of FMU generation is out of scope currently for the tool implementation. However, export of the FMU linked to FunctionBehaviors to a simulation engine would be useful. This could concern creating S-functions in Simulink according to the connected FunctionPrototypes or to configure a Simulation manager to run the executables according to execution and connection information defined in the EAST-ADL model.

2.10.4 Key Features

This tool implementation uses a plugin for EATOP to import the Function Mock-up Unit specification, called Function Mock-up Interface (FMI). Based on this, the tool then defines EAST-ADL functional blocks in terms of AnalysisFunctionType with the corresponding port interface. For each functional block, one sub-package is created for the data types typing the ports.

The input model for FMU import is the FMI XML file, which is a part of the ZIP archive constituting an FMU. This archive also contains an executable file for the intended execution platform(s), for example a Windows DLL. The FMI XML file contains sufficient information to create an EAST-ADL Analysis Function, i.e. function name, ports and datatypes.

The current plugin creates functional units in EAST-ADL in terms of AnalysisFunction. However, the traceability from the created EAST-ADL functional units to the source external behaviour descriptions is currently not created and populated with FMU information, such as the path to the FMU file.

3 References

- [1] www.atesst.org, accessed 2011-05-26
- [2] www.edona.fr, accessed 2011-05-26
- [3] www.artop.org, accessed 2011-05-26
- [4] MAENAD: Deliverable D2.1.1, draft version 0.5
- [5] www.fmi-standard.org, accessed 2013-03-19
- [6] MAENAD: Deliverable D3.1.1
- [7] MAENAD: Deliverable D3.2.1