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	**Intermediate**
Version number	**3.0**
Date of preparation	**2013-03-28**

Authors

Editor	E-mail
Carl-Johan Sjöstedt	carlj@md.kth.se

Authors	E-mail
Carl-Johan Sjöstedt	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
De-Jiu Chen	chen@md.kth.se
Henrik Lönn	henrik.lonn@volvo.com

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)

© 2013 The MAENAD Consortium

Revision chart and history log

Version	Date	Reason
1.0	2011-05-31	First release
2.0	2012-08-31	Second release
3.0	2013-03-20	Third release

© 2013 The MAENAD Consortium

Table of contents

© 2013 The MAENAD Consortium

1 Introduction

Deliverable D5.2.1 consists of:

- The plugins and tools that make up the MAENAD analysis workbench (MAW)
- This document is providing references and short descriptions of the tools in the MAW.

The MAW consists of software developed to provide support for modelling and analysis, based on specifications of WT3.x. One objective is to validate the analysis concepts. The MAW will be made public in order to make the analysis concepts more accessible and understandable.

For each plugin/tool, the sections *Current status*, *Input Models, Future plans* and *Requirements* are available. The *Input model* sections describe how a model needs to be set up to work with the tool, which could also be seen as a test case for the tool. The *Requirements* sections describe what project and language requirements that the plugin/tool fulfils.

The main target of MAW is the MAENAD Modeling Workbench, D5.1.1, but as it will support exchange to other tools using the EAXML format, some adaptors are based on other tools, mainly MetaEdit+.

© 2013 The MAENAD Consortium

2 The components of the MAENAD Analysis Platform

Some of the plugins were developed throughout the ATESST [1], ATESST2 [1] and EDONA [2] projects, and will be updated during the MAENAD project, because of new releases of the MAENAD modelling workbench, and new releases of the profile. There will also be new plugins, for the interchange of EAST-ADL models using an exchange format, and for exchange with Modelica and MODELISAR FMU:s.

	Main developer	Overview
AUTOSAR Gateway	CEA	Provide an enhanced transformation from EAST-ADL2 design architecture to AUTOSAR compliant software architecture, based on the ARTOP framework.
Timing Analysis – Qompass Tool	CEA	Provides support for early-stage timing analysis of EAST-ADL models
Simulink Gateway	KTH	Provides input/output facilities with Simulink to enable simulation.
HiP-HOPS Gateway	KTH	Builds on previous results, expanding analysis capability and optimization engine of HiP-HOPS, and enhancing the feedback of FMEA/FTA in the design process.
Architecture optimization and configuration	TUB	To achieve multi-objective optimization, the prototype tool OptiPAL has been developed, based on the EPM platform.
ASIL allocation with EPM/HiP-HOPS	UoH	ASIL allocation with EPM/HiP-HOPS
UPPAAL&SPIN Gateway	KTH	Analysis of behavioural constraints using the model checkers UPAAL and SPIN.
Modelica Exchange	KTH	Using Modelica together with EAST-ADL for behavior analysis.
FMU Import	VTEC	Generate EAST-ADL FAA models from Functional Mock-up Units.

© 2013 The MAENAD Consortium

2.1 AUTOSAR Gateway

The AUTOSAR gateway builds on results from EDONA and ATESST2 projects to provide an enhanced transformation from the EAST-ADL design architecture to AUTOSAR vehicle architecture design and initial system configuration. 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 [5]. 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.

2.1.1 Current status

The first version of the AUTOSAR gateway has been released at M12. This version was based on the one developed in the EDONA project and then it had been ported to the new Papyrus MDT platform and 2.1.9 EAST-ADL profile version.

As previously planned, a new 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.

2.1.2 Input models for the AUTOSAR Gateway

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.

2.1.2.1 EAST-ADL2 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-ADL2 model. Let us consider the EAST-ADL2 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-ADL2 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

© 2013 The MAENAD Consortium

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.

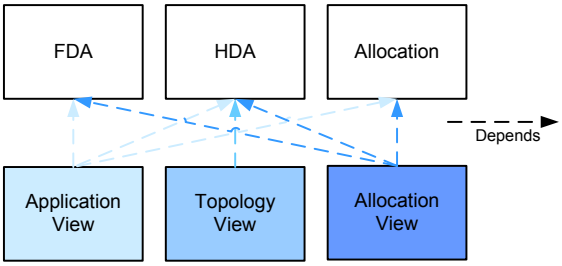


Figure 1. Dependencies in the Generation of the AUTOSAR Model from the EAST-ADL2 Model

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, EAST-ADL2 Allocation model specifies functional allocation and by tracing the relation between functions and the produced software components, using the EAST-ADL2 Allocation model (described by UML dependencies), allocation of software components can be inferred.

2.1.2.2 AUTOSAR model to arxml

Generation of the arxml file is relatively simpler than the generation of AUTOSAR model from EAST-ADL2. 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, in which the relation between the EAST-ADL2 model and AUTOSAR elements is not that obvious, due to the different concepts present in different languages.

In the context of the implementation, similarly to the generation of AUTOSAR model, generation of arxml file is divided on 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

© 2013 The MAENAD Consortium

- 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-ADL2 model to AUTOSAR model and AUTOSAR model to arxml) was to enable the designer to change 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, designer can violate few important rules that have to be respected.

The entire input model needs to be structured into a way presented on the 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 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.

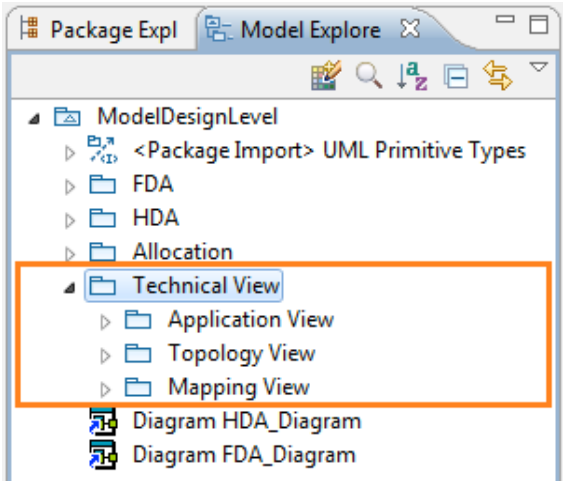


Figure 2. Structure of the AUTOSAR model recognized by the arxml file Generator

© 2013 The MAENAD Consortium

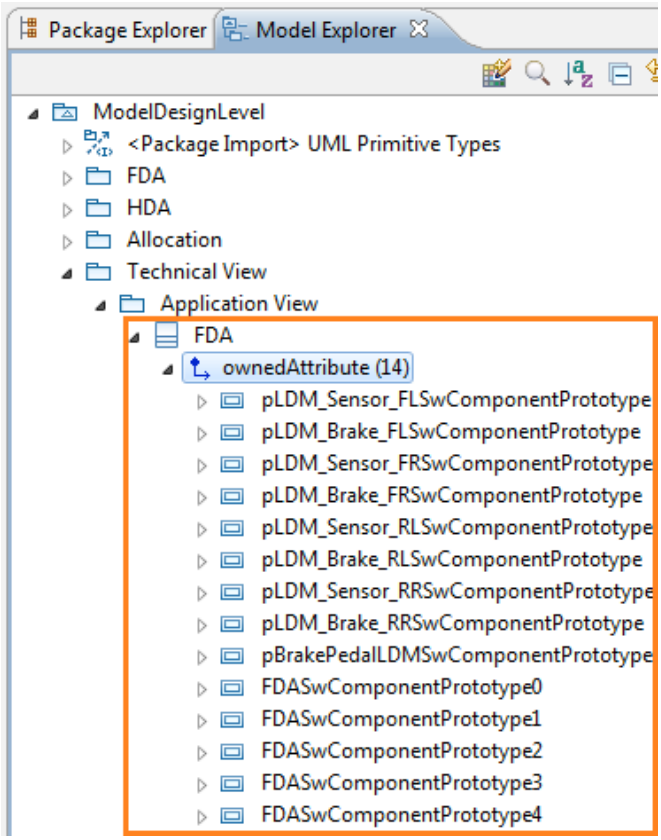


Figure 3. Specification of Software Component Prototypes as Owned Attributes of a CompositionSwComponentType

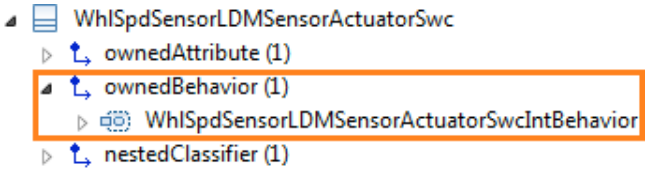


Figure 4. InternalBehavior of SwComponentType specified as an ownedBehavior

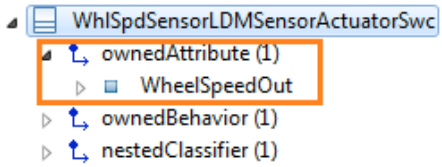


Figure 5. Port specified as Owned Attribute of Software Component Type

© 2013 The MAENAD Consortium

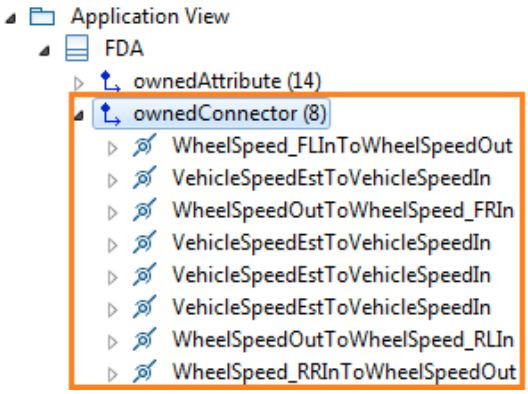


Figure 6. Port connectors specified as Owned Connectors of Composition Software Type

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 occurence 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-ADL2 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).

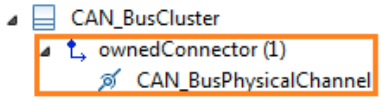


Figure 7. Specification of a Cluster and corresponding Physical Channel as an Owned Connector.

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.

© 2013 The MAENAD Consortium

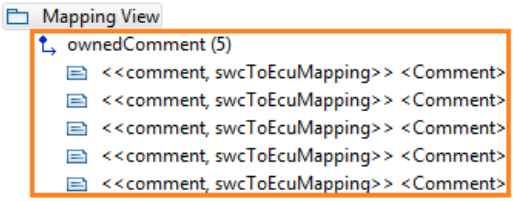


Figure 8. Specification of a Mapping View with the Owned Comments modeling allocation of Software Components

2.1.3 Future plans

Beyond M30 we will proceed to the AUTOSAR gateway assessment considering feedback from MAENAD users.

2.1.4 Requirements from WT2.1: Identifications of needs

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].

© 2013 The MAENAD Consortium

2.2 Timing Analysis

At the beginning of the project the purpose of the Timing Analysis plug-in was centred on the idea of providing schedulability analysis of EAST-ADL models. Beyond M12, Timing Analysis for EAST-ADL models has been refined [6]. The following timing analyses have been identified as best-suited to support EAST-ADL Design level models (as documented in D3.1.1):

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.

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) trough Rate Monotonic Analysis [6].

2.2.1 Current status

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 ATESST2, 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. At the current state this extension has been implemented and now Qompass can run on the BBW model.

2.2.2 Input models for the Timing Analysis plug-in

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

© 2013 The MAENAD Consortium

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).

Note that it is *mandatory* for an EventFunctionFlowPort that both the port and port_path attributes are specified.

Note that it is mandatory for the event chain to have exactly one stimulus and exactly one response.

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.

It is also mandatory that each stimulus is generated by a unique source function.

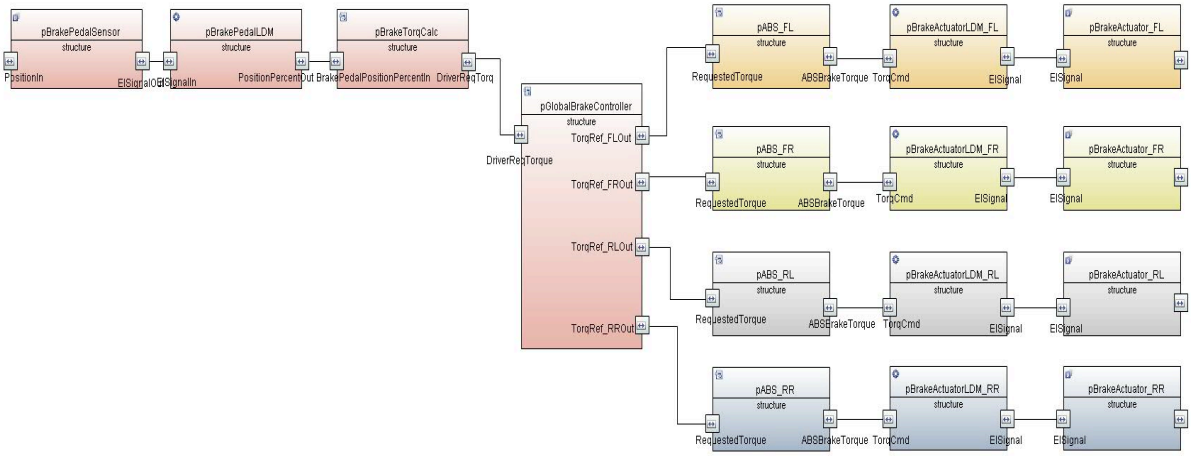


Figure 9. Excerpt of the Functional Design Architecture

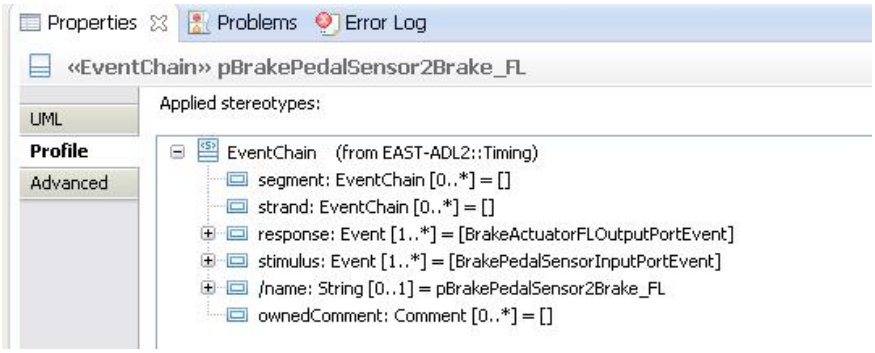


Figure 10. Example of Event Chain

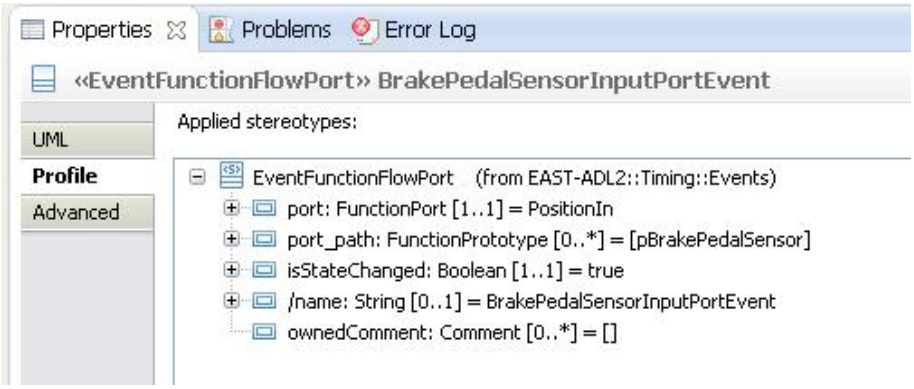


Figure 11. Stimulus Specification

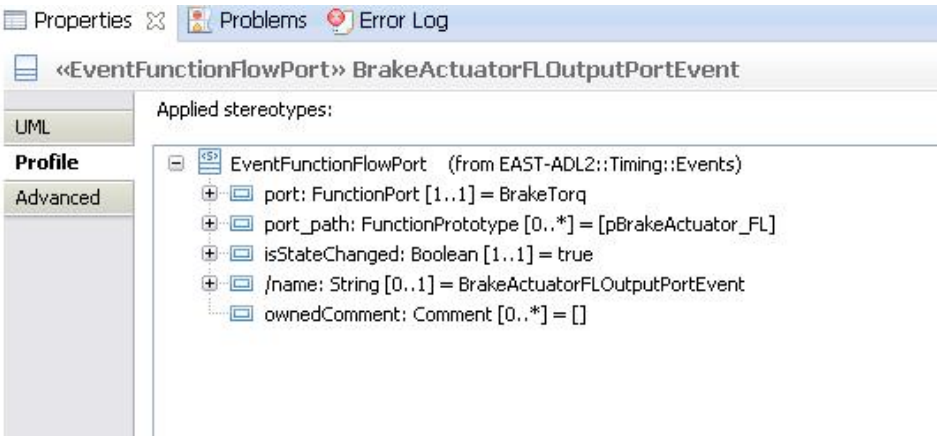


Figure 12. Response Specification

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;

© 2013 The MAENAD Consortium

- 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.

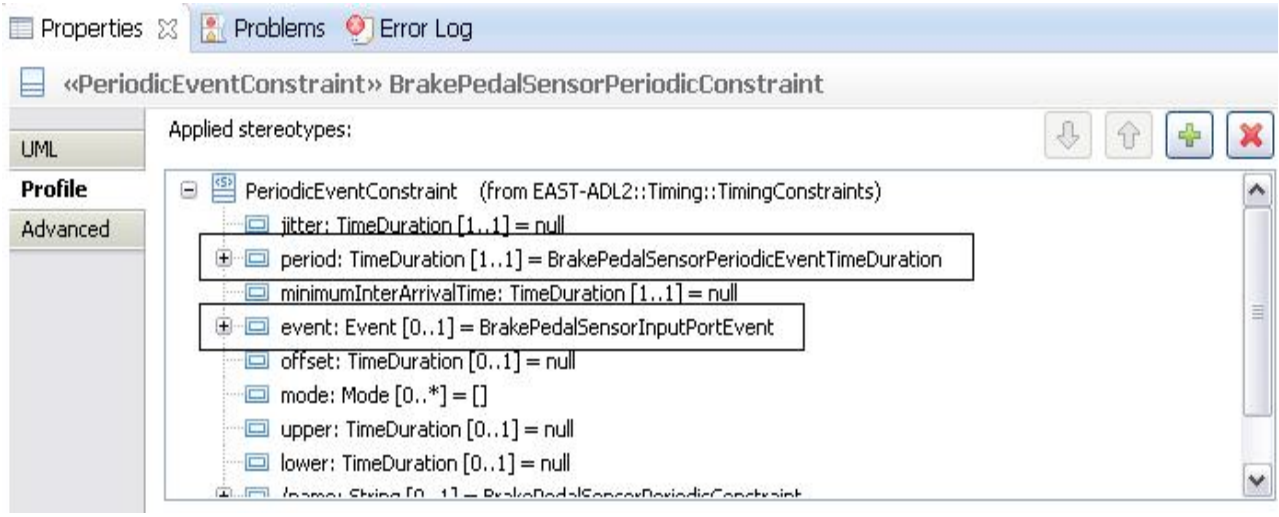


Figure 13. Periodic Event Constraint Specification

© 2013 The MAENAD Consortium

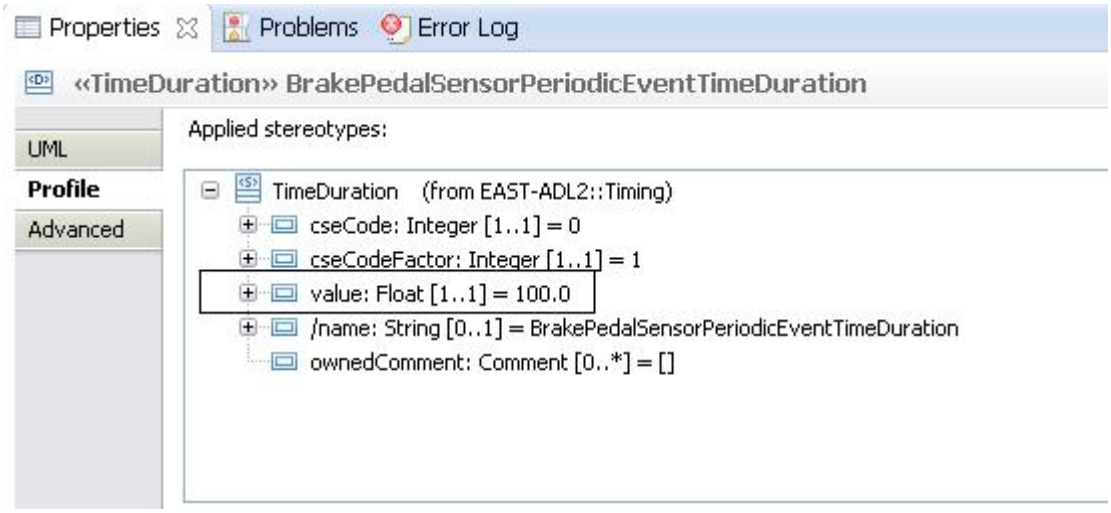


Figure 14. Time Duration Specification

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.

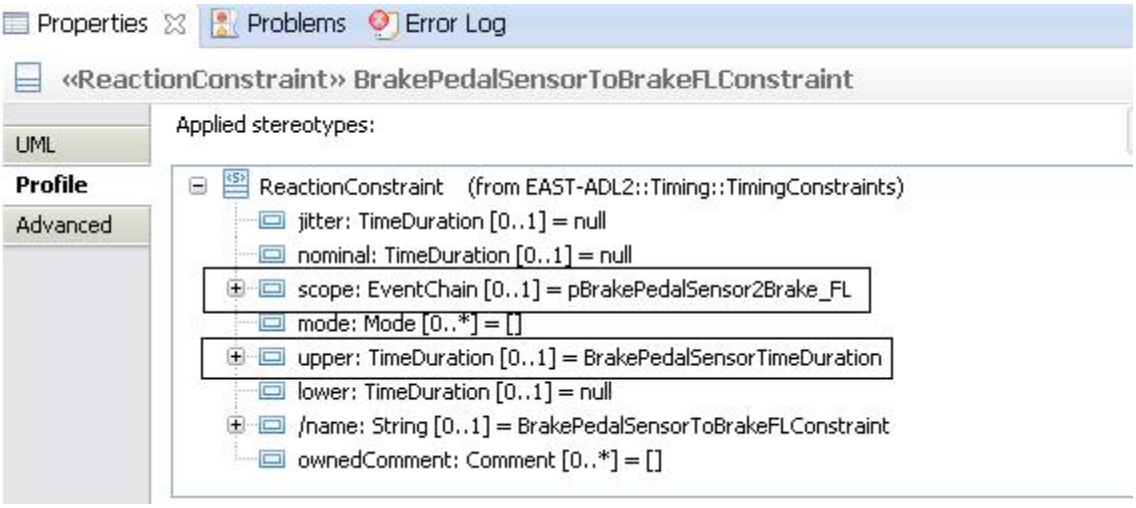


Figure 15. Reaction Constraint Specification

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.

© 2013 The MAENAD Consortium

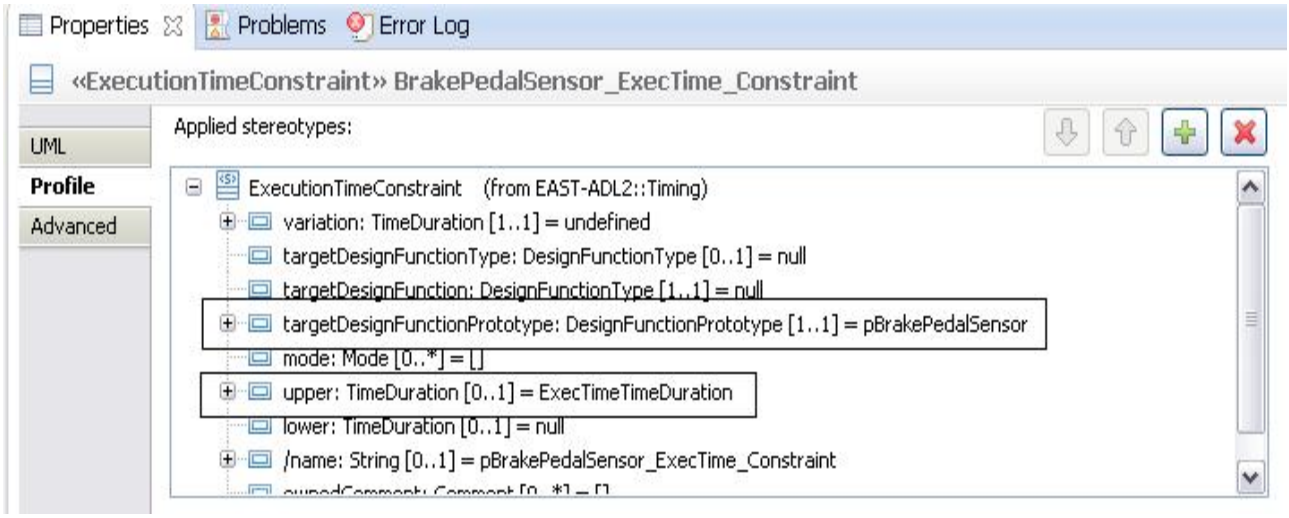


Figure 16. Execution Time Constraint Specification

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.

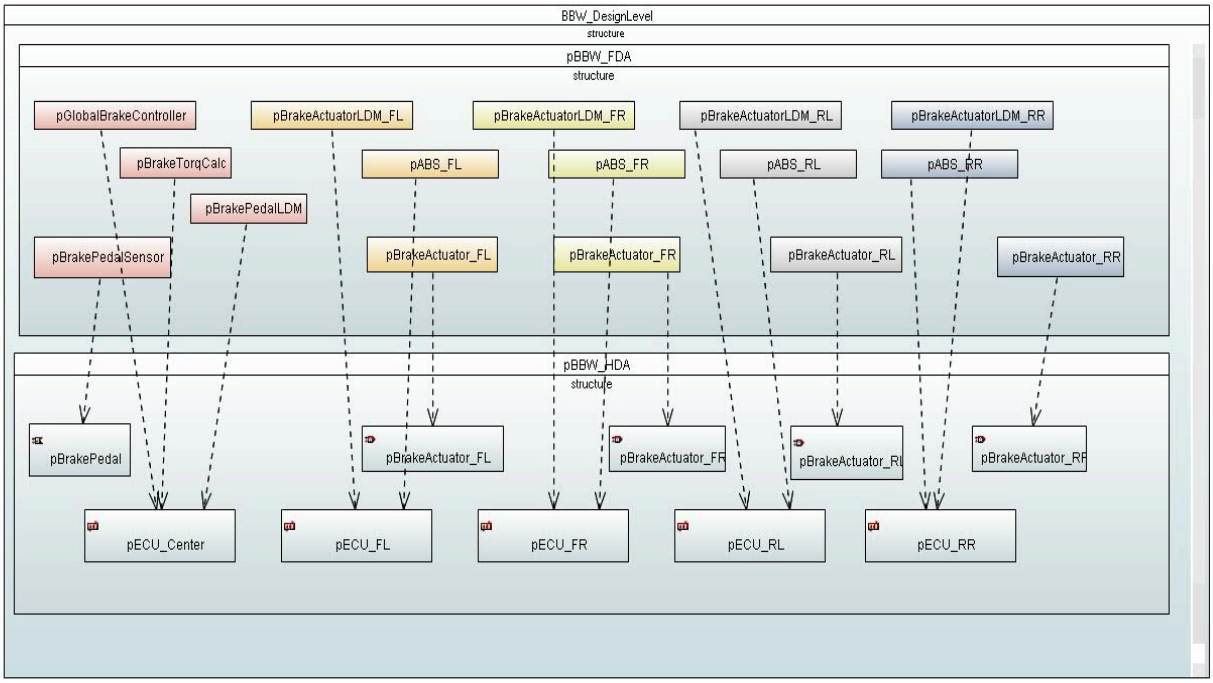
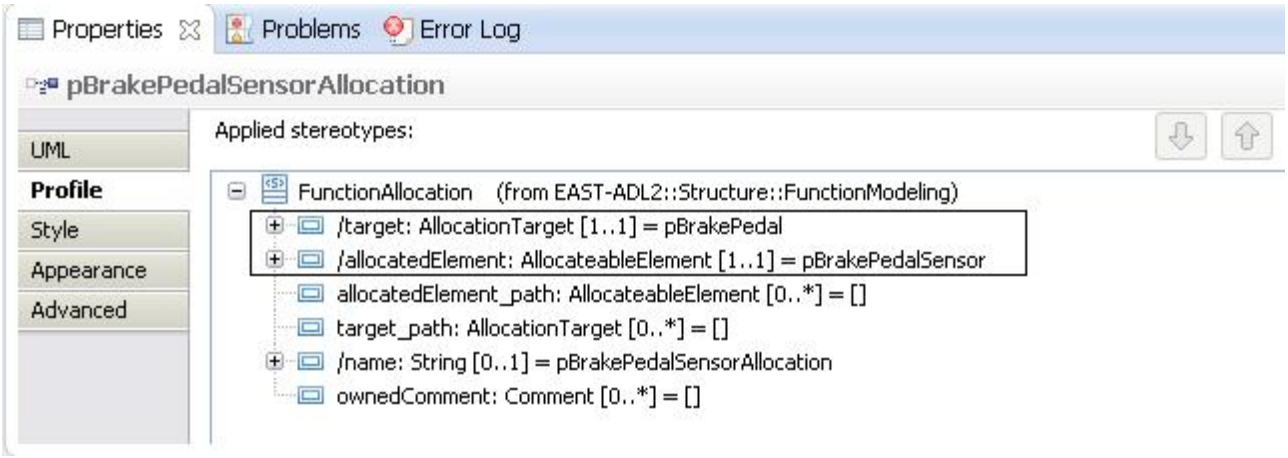


Figure 17. Allocation

In this UML diagram, the FunctionAllocation is represented as a dependency (more technically the FunctionAllocation concepts 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.

© 2013 The MAENAD Consortium



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 18.

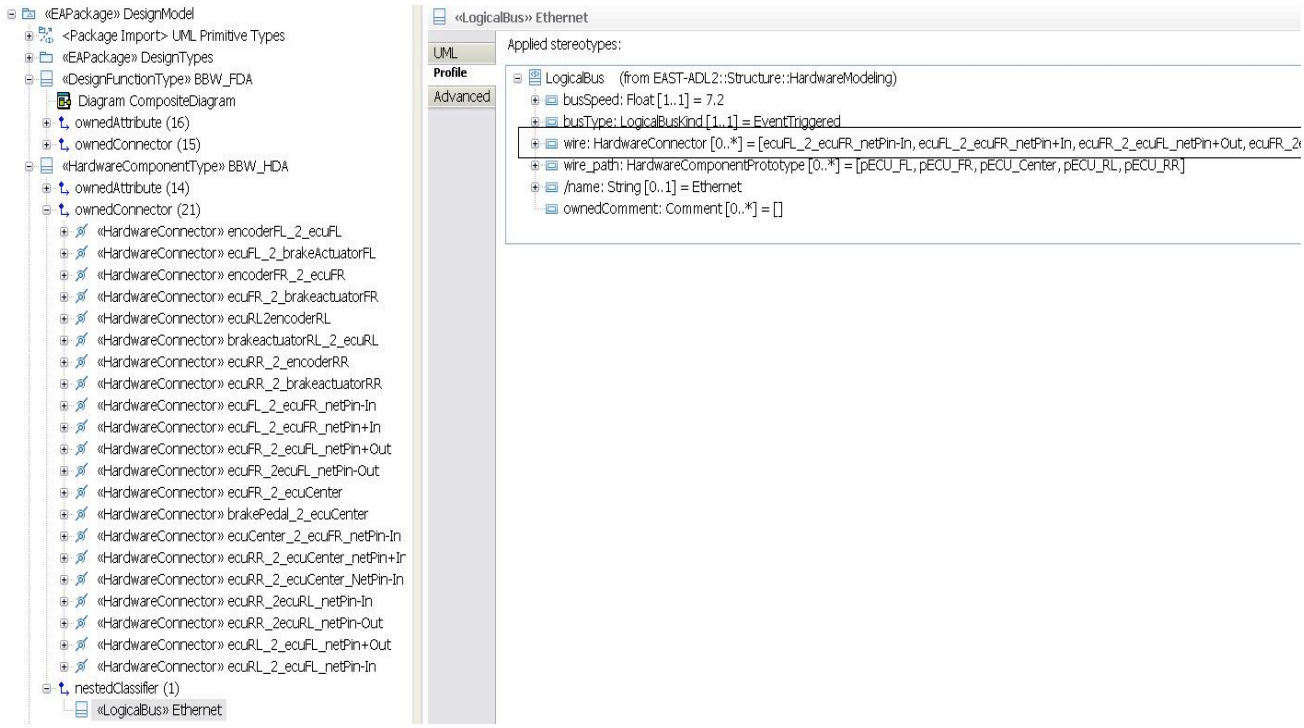


Figure 18**. Logical Bus Specification**

2.2.3 Future plans

M30 was planned to be the last release for the Qompass plug-in: i in compliance with what has been planned, the timing analysis can now be applied on the BBW model. This analysis performs an early stage evaluation of timing properties hold by design level models (see [7]), but response time analysis is not provided. The Qompass plug-in, in fact, computes response times only for the mono-processor case (as an implementation model is automatically and transparently generated from the design model). An update of the plug-in has been scheduled for the end of June providing

© 2013 The MAENAD Consortium

response time analysis for the distributed case and possible improvements required by MAENAD users. Requirements from WT2.1: Identifications of needs

The Timing Analysis plugin is mentioned in the following requirement:

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

2.3 Simulink Gateway

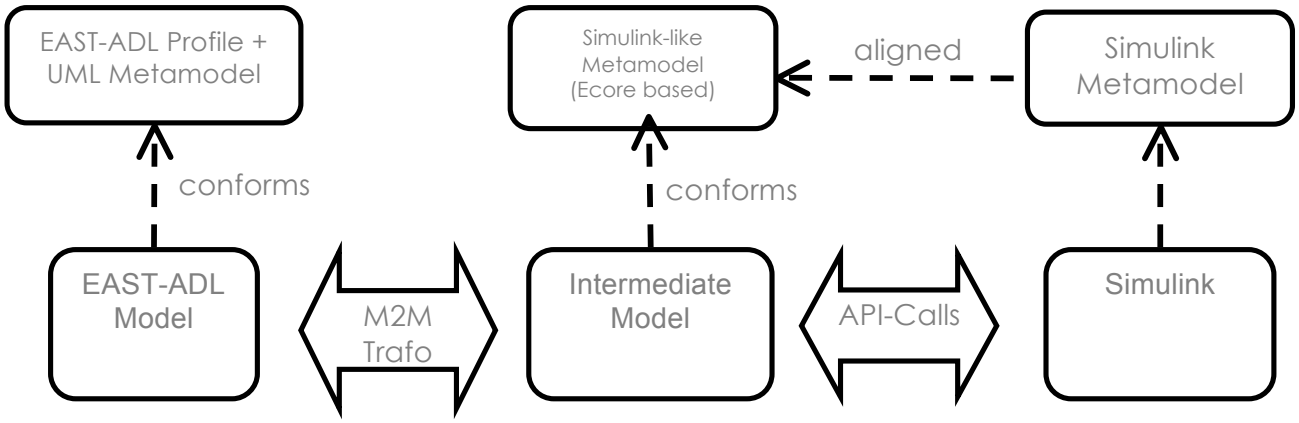


Figure 19. Overall design

The Simulink gateway builds on results from the ATESST2 project, a Simulink gateway was developed, and provides input/output facilities of models with Simulink to enable simulation. The plugin is divided in two parts (Figure 19):

- 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.
- An Eclipse plugin, which can convert between the intermediate format of Simulink models and EAST-ADL models

The MATLAB plugin exports the MATLAB/Simulink models into a custom Ecore-based format. A subset of Simulink functions is used; only library blocks of subsystems are considered. However, any Simulink model could be converted into a structure of system reference blocks, without affecting the model's simulation behaviour. The GUI plugin for Simulink mentioned above 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. To include the internal structure of a subsystem, the same pattern is repeated.

Import works the other way around, FAA FunctionTypes and FunctionPrototypes are imported to empty library blocks in the "FunctionTypes"-library, and instances of them respectively.

In addition to the above tool MetaCase has developed a Simulink-exchange mechanism for MetaEdit+, creating .mdl-files. A similar mapping is used as the above mentioned plugin.

With MetaEdit+, KTH has developed another exchange from MetaEdit+ to Simulink and StateFlow, as a validator for the behavioural annex. Instead of .mdl-files, this plug-in relies on Matlab API for the creation of Simulink/Stateflow models.

2.3.1 Current status

An effort has been made to port the plugin developed in the ATESST2 project to the new Papyrus MDT environment. It is possible to run the plugin, but the ATL transformation eventually crashes, for unknown reasons.

© 2013 The MAENAD Consortium

Due to lack of resources from KTH, no further development has taken place, there are no interest in further developing this plugin.

Simulink exchange has been implemented in MetaEdit+, which could better serve as a demonstrator for proof-of-concept exchange between Simulink and EAST-ADL.

2.3.2 Input models for the Simulink Gateway

The ME+ plugin by MetaCase for Simulink transformation takes a FAA or a FDA architecture description as the input model.

The ME+ plugin by KTH for Simulink&Stateflow transformation takes a FAA or a FDA architecture description, together with the corresponding behaviour constraint annotations, as the input model.

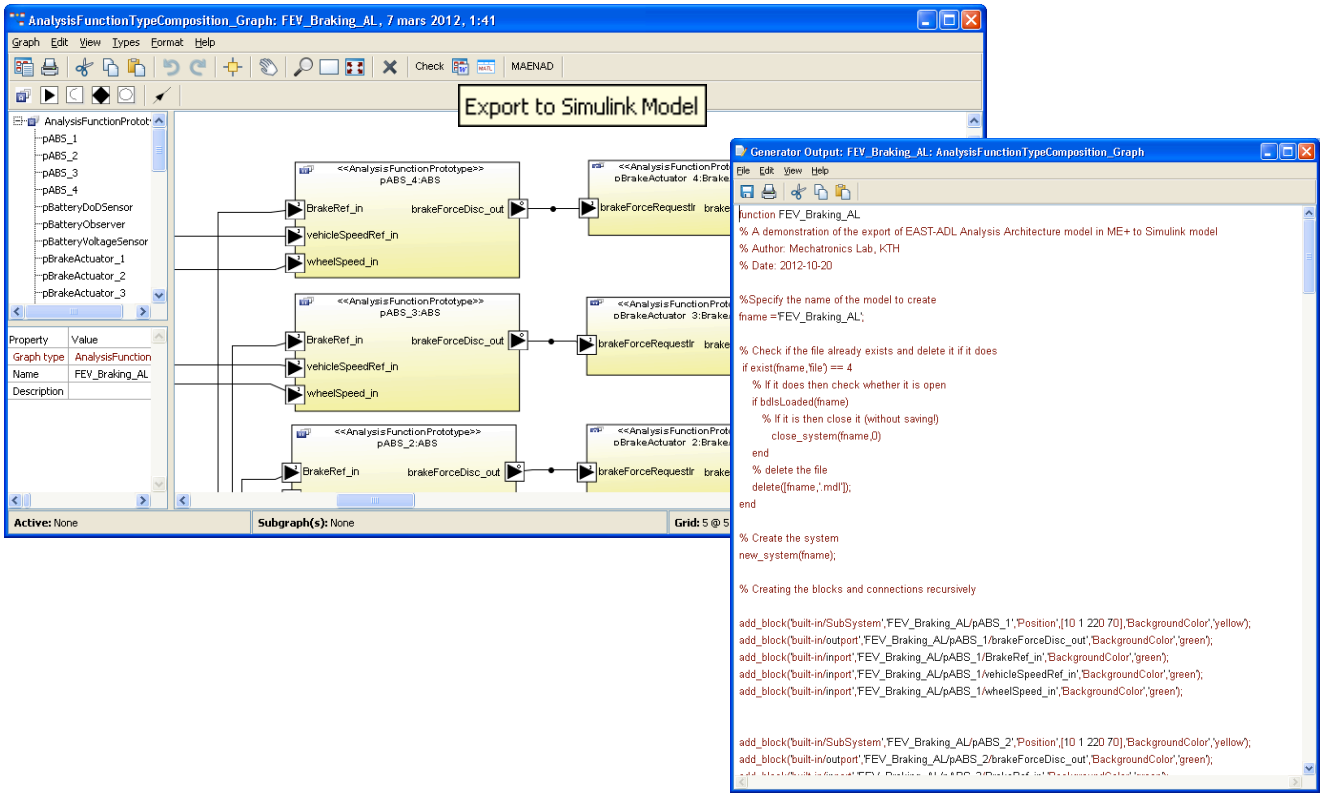


Figure 20. Screenshot of a Simulink export scenario. The exported file defines the commands to Matlab API for model creation.

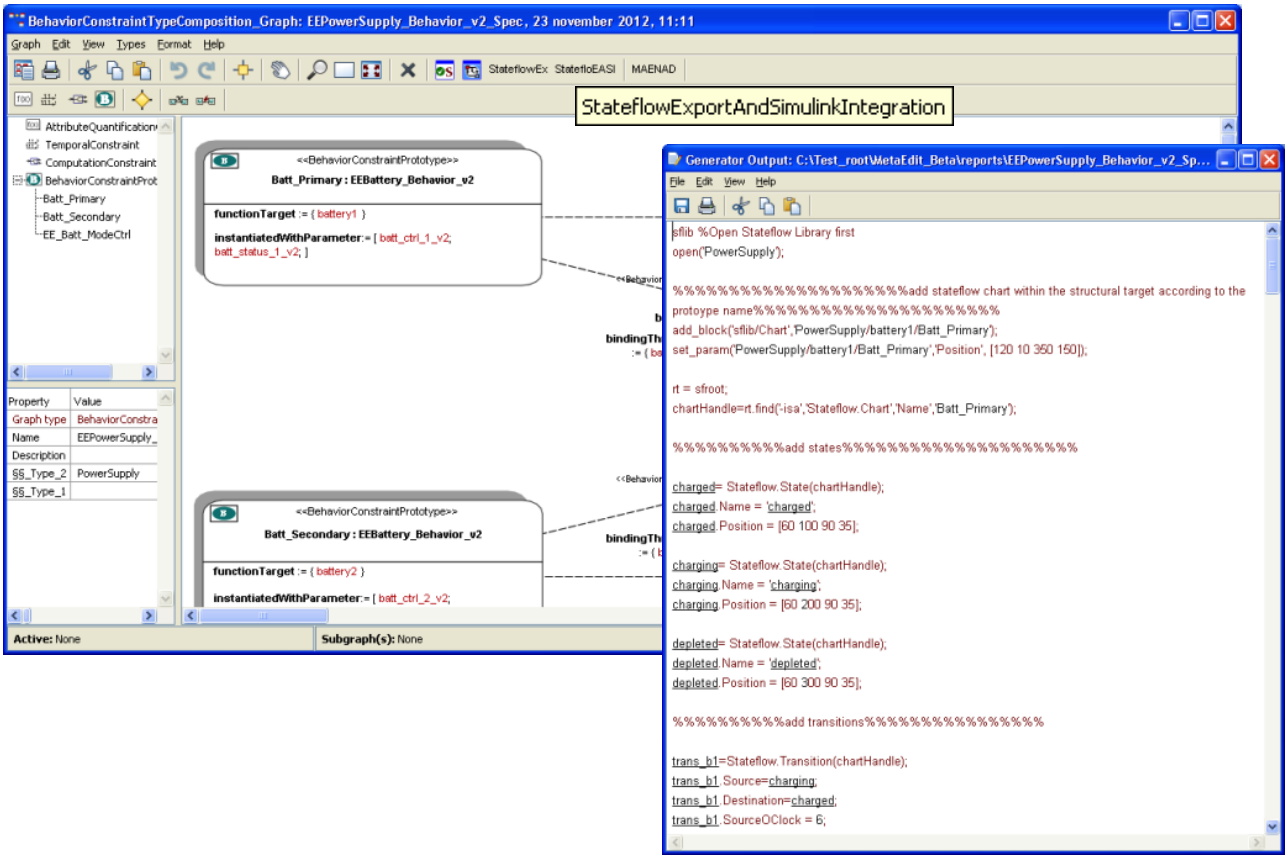


Figure 21. Screenshot of a Simulink export scenario, where the behavioural content of blocks in Stateflow is added. The exported file defines the commands to Matlab API for behaviour model creation.

2.3.3 Future plans

KTH is prototyping of behaviour analysis with MATLAB/Simulink using the MetaEdit+ editor, and exchange mechanism.

2.3.4 Requirements from WT2.1: Identifications of needs

A2#11: A formalized meta-model of the architecture description language shall be developed. (including structural elements, behavioural description means, models of computations, and transformation rules to prototype tools and Simulink.)

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

© 2013 The MAENAD Consortium

2.4 HiP-HOPS Gateway

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-ADL2 architecture description language, which supports system design on multiple levels of abstraction. The concepts of the error analysis domain are represented by the safety analysis tool HiP-HOPS.

There are two separate interfaces to HiP-HOPS that have been developed. The first is a plugin for Papyrus, and the second is a plugin for EPM. Both will be briefly described below.

2.4.1 Papyrus Plugin

The first of the two interfaces to HiP-HOPS is a model transformation plugin for the Papyrus tool, which was primarily developed in ATESST2 by Matthias Biehl from KTH. It has since been updated again during MAENAD by Nataliya Yakymets of CEA, with additional assistance from KTH. It allows EAST-ADL models that have been developed in Papyrus to be exported to HiP-HOPS for analysis.

It is assumed that EAST-ADL models are built using the UML profile. The HiP-HOPS plugin extensively uses the concepts defined in the EAST-ADL error model, but also other language constructs from the FDA level.

We automate the translation from EAST-ADL to HiP-HOPS by using model transformations. We leverage the advantages of different model transformation techniques by decomposing the translation into two distinct phases, and using an appropriate technique for each phase: A phase for conceptual mapping between the domains followed by a phase for representing the output in the desired concrete syntax.

With the resulting tight integration of the safety analysis tool and the model-based development environment, the automotive safety engineer can perform the safety analysis repeatedly on refined models with minimal effort. This is compliant with the iterative design activities, which require starting the analysis after each change in the system design.

The HiP-HOPS Gateway builds on previously developed Model Transformations and Eclipse Plugin knowledge developed during ATESST2. The process involved some harmonization between the EAST-ADL error model and the HiP-HOPS metamodel as well, and led to the development of multi-perspective analysis capabilities in HiP-HOPS.

2.4.2 EPM Plugin

The second EAST-ADL gateway to HiP-HOPS is an interface developed for the EPM tool by Mark-Oliver Reiser at TUB. This also originally arose from near the end of the ATESST2 project, where it was needed to be able to perform a HiP-HOPS safety analysis on a demonstrator model that only existed in EPM. It has since been extended further during MAENAD in parallel to the Papyrus plugin.

Unlike the Papyrus interface, which exports to HiP-HOPS via a model transformation engine, 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).

© 2013 The MAENAD Consortium

The range of information EPM can export to HiP-HOPS has been considerably expanded during MAENAD, also serving as the basis for the ASIL decomposition tool support, and further extensions are planned for the future.

2.4.3 Current status

The 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. Core functionality is working, including HW-SW allocation.

- Update of the plugin to work with the current version of Eclipse Indigo
- Update of the plugin to work with the current version of Papyrus MDT 0.8.2
- Update of the plugin to work with the current version of the EAST-ADL Profile 2.1.10
- Update of the test model for the current version of Papyrus

The EPM plugin is also being updated to serve as the basis of ASIL decomposition export to HiP-HOPS.

2.4.4 Input models for the HiP-HOPS Gateway

An example input model for the Papyrus HiP-HOPS Gateway is available on the repository below. Example EPM models will also become available in due course.

2.4.5 Future plans

The Papyrus HiP-HOPS gateway has been made public on the link below. It is released under the Eclipse Public License. No further development or maintenance is planned by KTH.

https://code.google.com/p/east-adl-safety-analysis/

As stated above, the EPM plugin is still being updated and developed further to increase the range of information it can export to HiP-HOPS, and thus increase the range of analyses available to users. See also section 2.6 below for more information.

© 2013 The MAENAD Consortium

2.4.6 Requirements from WT2.1: Identifications of needs

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.

© 2013 The MAENAD Consortium

2.5 Architecture optimization and configuration

The architectural optimization & configuration capabilities to be developed in MAENAD build upon several tools and plugins, including CVM (and the corresponding variability management plugin from ATESST2), HiP-HOPS (and its associated plugin), and timing analysis tools like MAST (and associated plugin(s)). These tools need to be interfaced with an optimization engine for fully multi-objective optimization of EAST-ADL models to be possible.

2.5.1 Current status

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. This architecture is shown below:

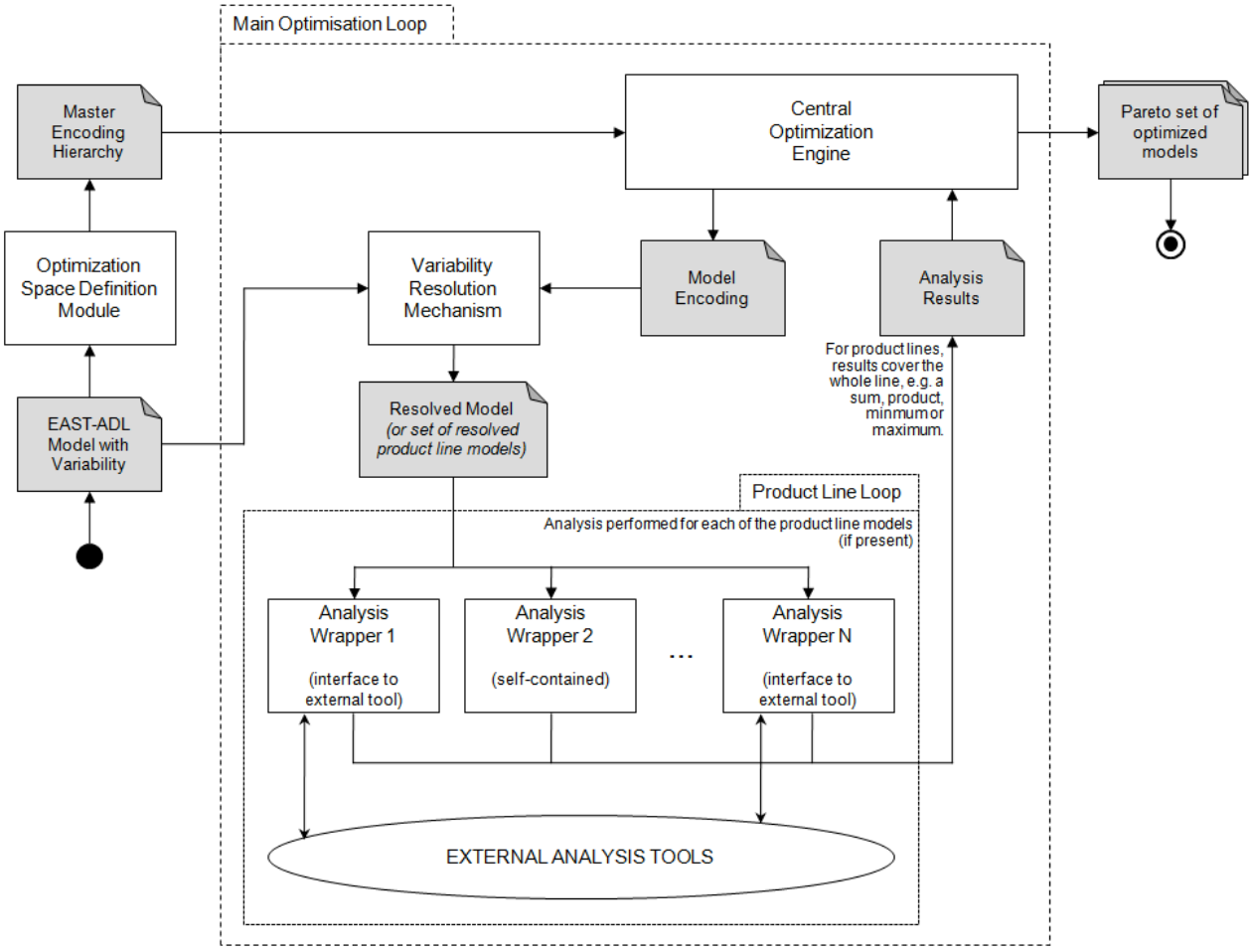


Figure 22. Optimisation Architecture

The optimisation 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 key to 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.). Optimal candidates are preserved while sub-optimal designs are discarded. Once the process is complete, the COE will generate a report containing the set of optimized 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).

Scope has been left for product line optimisation to be added at a future date, although this will require additional complexity in the analysis wrappers, as each analysis type may have to function differently to achieve optimisation of product lines (e.g. the results of an unavailability analysis on a product line may just be the maximum probability, while the results of a cost analysis are likely to be a sum of weighted costs; the result is therefore heavily dependent on the type of analysis).

An initial prototype tool, OptiPAL, was developed by TUB and builds upon the 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.

2.5.2 Input models for the OptiPAL tool

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

© 2013 The MAENAD Consortium

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

© 2013 The MAENAD Consortium

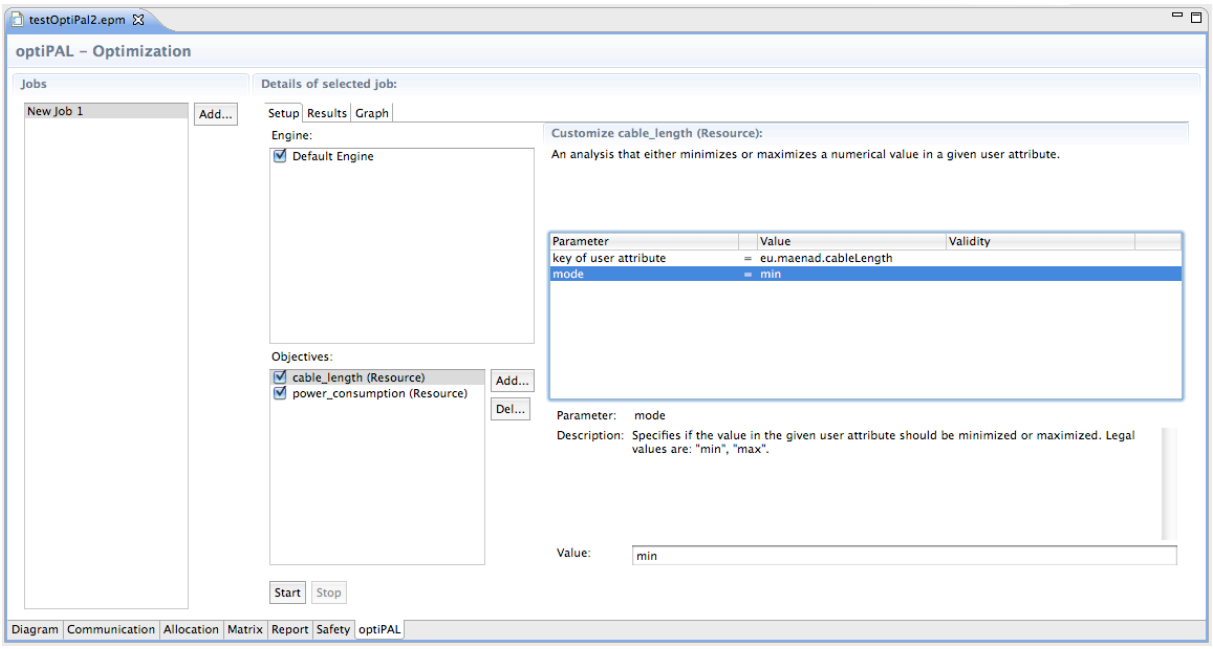


Figure 23. Screenshot of an optimization scenario specification in OptiPAL.

2.5.3 Presentation of Optimization Results 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).

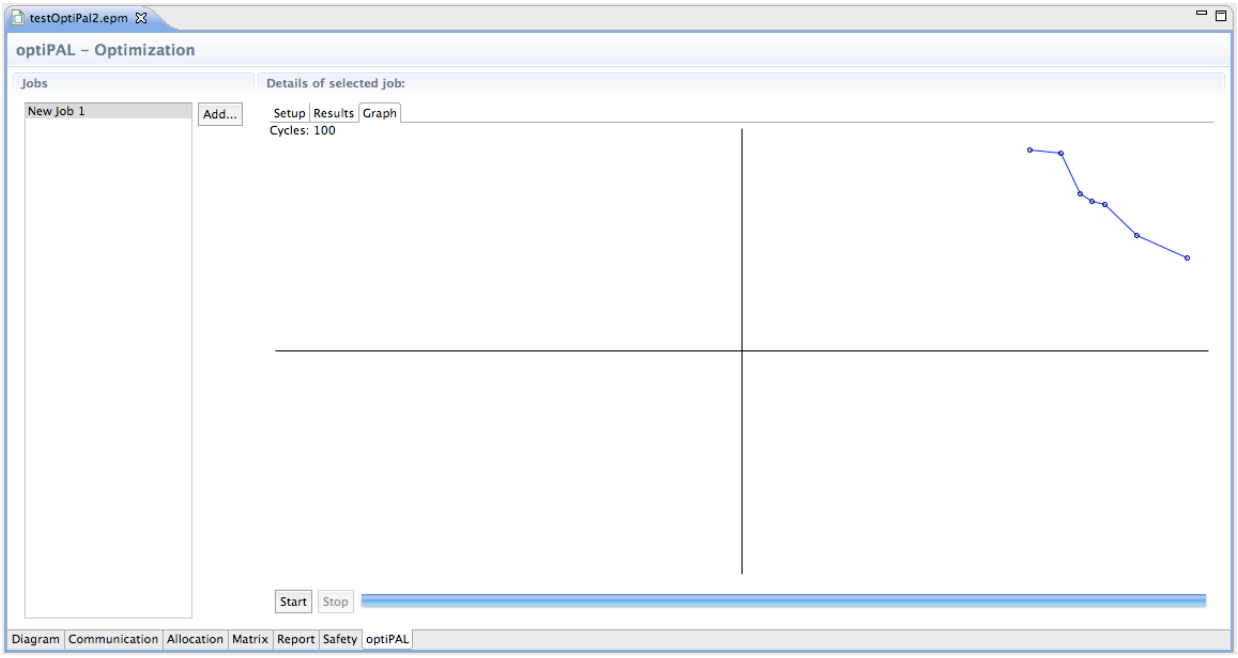


Figure 24. Graphical representation of a pareto-front in OptiPAL.

© 2013 The MAENAD Consortium

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.

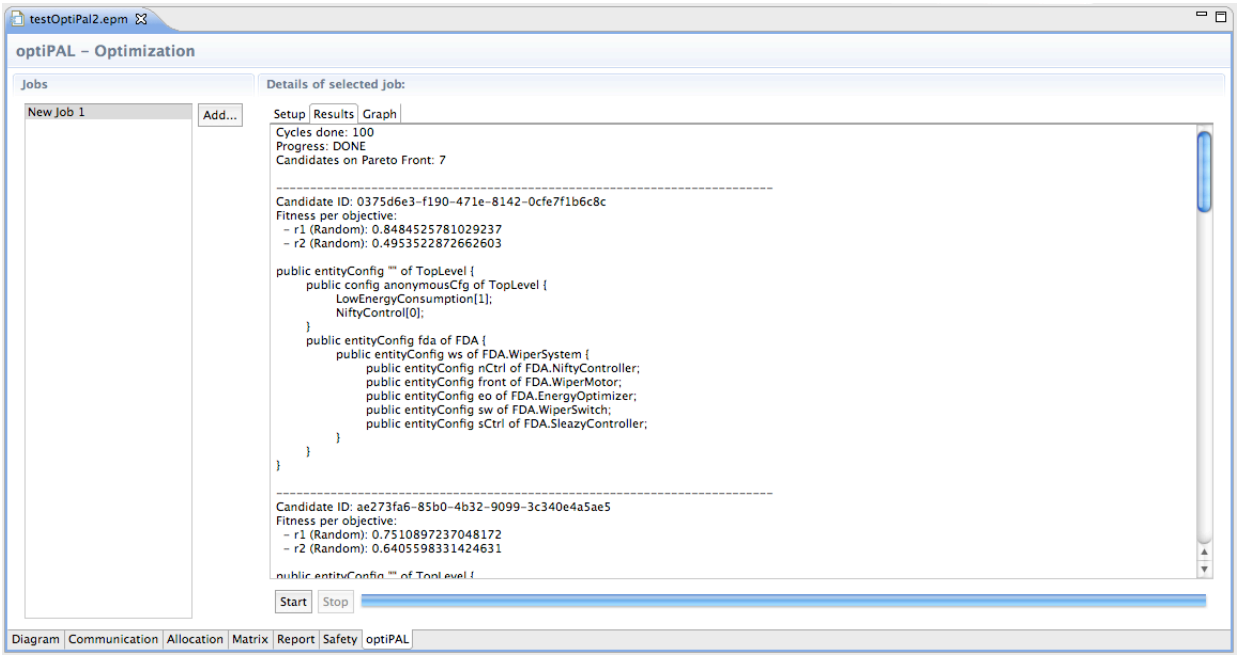


Figure 25. Textual representation of optimization results in OptiPAL.

2.5.4 Future plans

The development of OptiPAL should greatly facilitate XGO work in the third year of MAENAD, enabling us to test the optimization concept on test models and use the feedback received to develop the concepts further. One of the main tasks ahead is the generation of those test models, both smaller, simpler models to test individual ideas and problems, and larger models such as the brake-by-wire model, which is intended to be extended with optimization variability. This has been a goal for some time but was hampered by a lack of tool support until now.

Should the experiments prove fruitful, the aim is to upgrade and extend OptiPAL further. The first step will be to implement full genetic algorithms in the COE module, to allow a better assessment of the practicality of the optimization process and produce more meaningful results in the form of a Pareto set of optimized designs. 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, the goal is to investigate the feasibility of product line optimization further. This would take the form of an 'inner optimization loop' and would mean leaving some variability unresolved in the encoding from the COE to allow the different parts of the product line to be iterated through exhaustively and analysed individually. The results of these analyses would then be combined in an analysis-specific manner by the appropriate analysis wrapper and returned as the results of the evaluation for that product line. Product line optimization would likely be a topic investigated towards the latter half of the third year, as it depends upon the standard optimization process being developed successfully.

© 2013 The MAENAD Consortium

2.5.5 Requirements from WT2.1: Identifications of needs

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.

© 2013 The MAENAD Consortium

2.6 ASIL allocation with EPM/HiP-HOPS

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.

Therefore, 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.

2.6.1 Current status

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).

2.6.2 Input Models for 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.

© 2013 The MAENAD Consortium

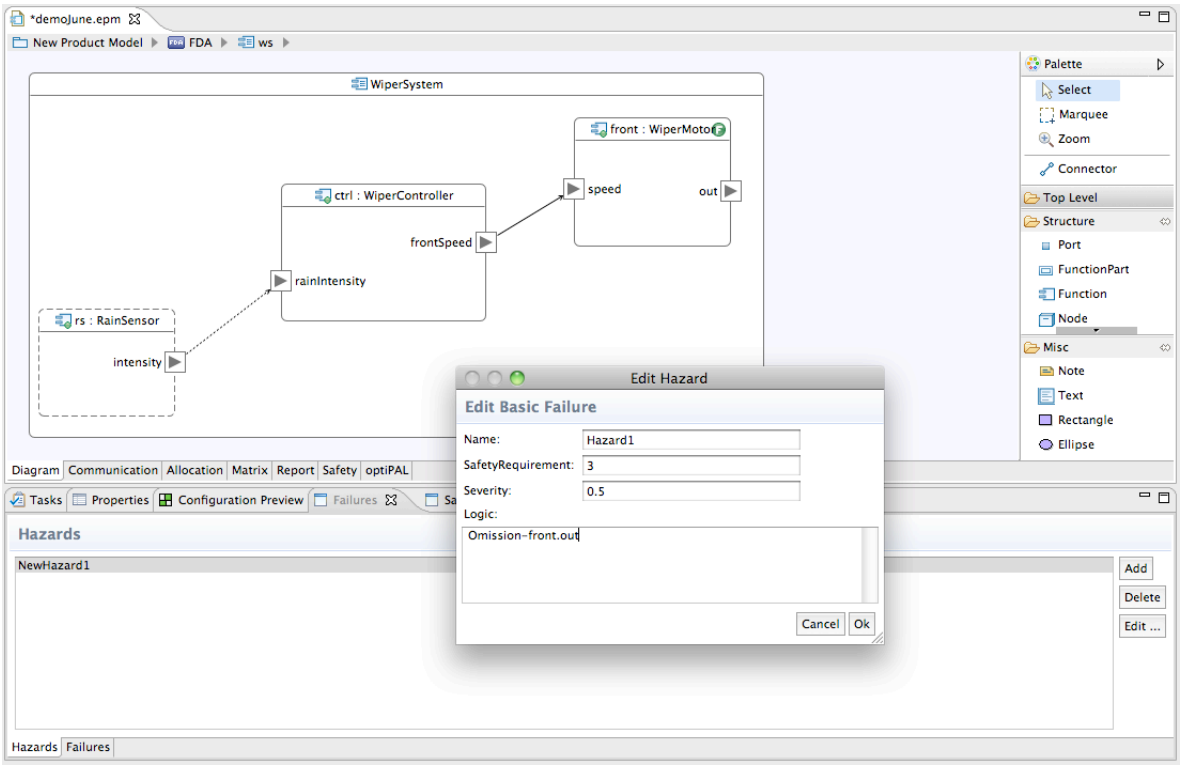


Figure 26. Editing Hazards in EPM.

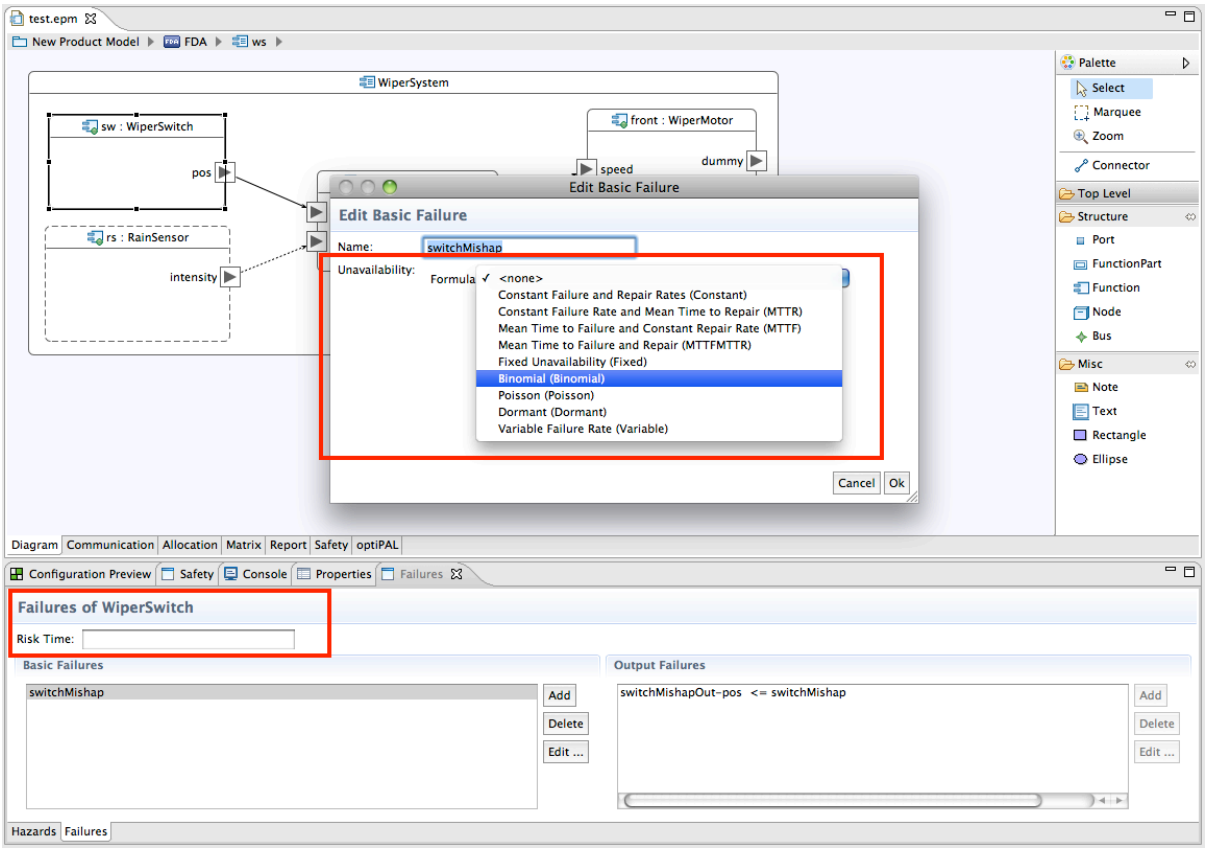


Figure 27. Editing Risk Time (bottom left) and Unavailability Formular (center) in EPM.

2.6.3 Future Plans

The current support of ASIL decomposition is on a level of what was aimed for in the MAENAD DoW, and it is possible to perform decomposition of ASILs via HiP-HOPS. Therefore, only minor fixes and refinements are expected until the end of the project. However, longer term development work by UOH on improved ASIL decomposition algorithms is still underway, and this may yet feed into the project if sufficient progress is made.

2.6.4 Requirements from WT2.1: Identification of needs

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

© 2013 The MAENAD Consortium

2.7 UPPAAL&Spin Gateways

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.

1. **UPPAAL** – a well-known timed model checker. 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.
2. **SPIN** – a well-known logic model checker. 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.

2.7.1 Current Status

The mapping schemes from EAST-ADL Behavior Description Annex to these two external tools are available. Work has been conducted to define the plug-in prototypes in MetaEdit+. A demonstrator for the UPPAAL transformation has been developed.

2.7.2 Input Models for the UPPAAL/Spin Gateways

The ME+ plugin by KTH for UPPAAL&SPIN transformation takes a behaviour constraint specification as the input model.

© 2013 The MAENAD Consortium

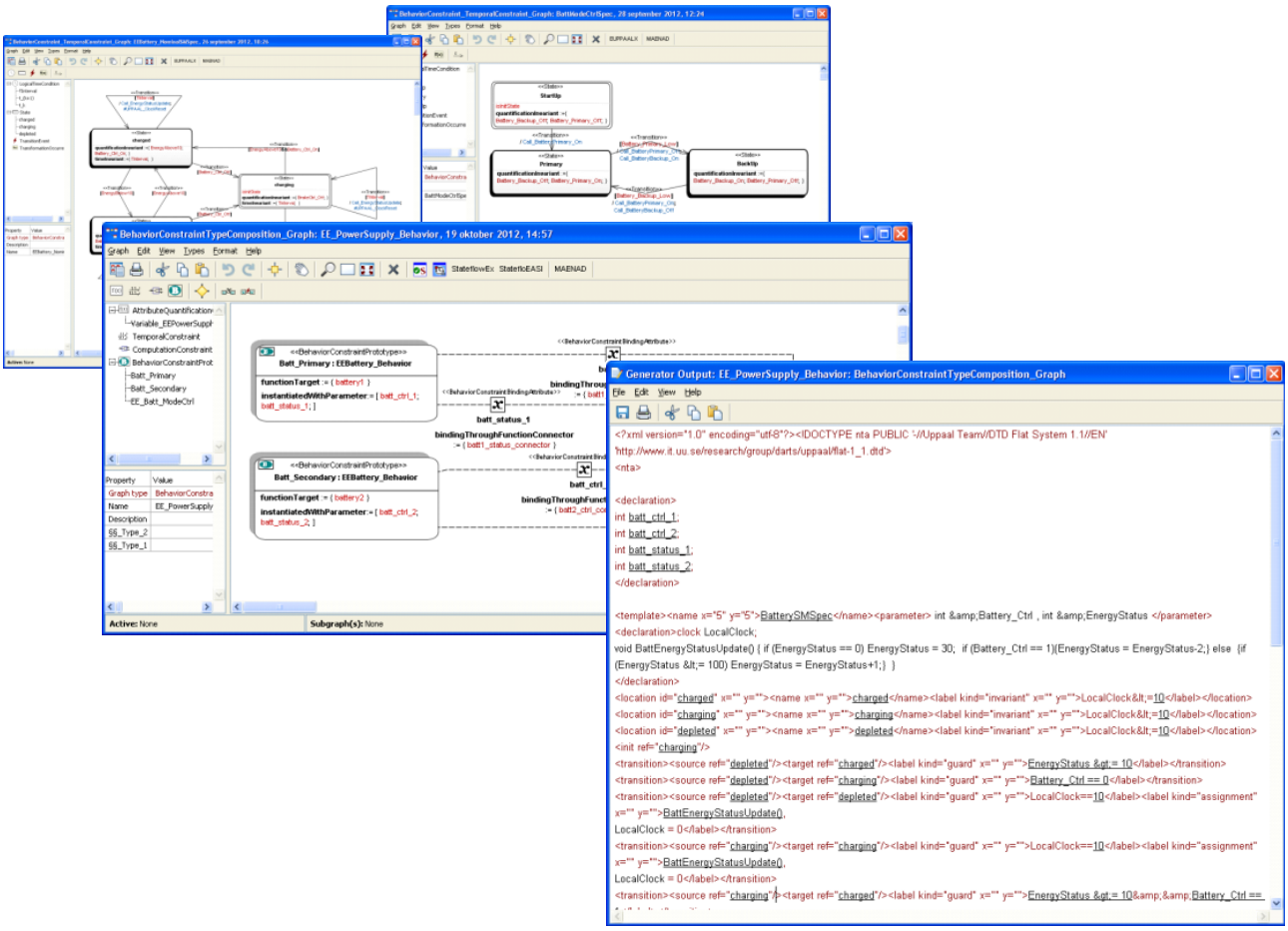


Figure 28. Screenshot of a UPPAAL/SPIN export scenario. The exported file defines the behaviour model in UPPAAL XMI.

2.7.3 Future plans

Further investigation will be centred on the following issues: 1. To have a more complete coverage of the regenerative braking case for demonstration purposes. 2. To finish the SPIN transformation plug-in prototype in MetaEdit+.

2.7.4 Requirements from WT2.1: Identification of needs

Behavior simulation is mentioned in:

DOW#0012 O2-2: Behavioral Simulation of EAST-ADL2 models

DOW#0017 O4-2: Evaluation of dependability & performance analyses

2.8 Modelica Exchange

Modelica is a language for modelling and simulation of dynamical systems, and could be used for various analyses of EAST-ADL models, e.g. modelling of plant models or timing constraints.

2.8.1 Current status

There is a plugin for Papyrus MDT for ModelicaML, using the UML stereotype mechanism. By assigning both an EAST-ADL FunctionType and a ModelicaML stereotype to a class, Modelica behaviour can be assigned to EAST-ADL FunctionTypes.

The current idea is to develop Modelica exchange for evaluation of behavioural contraints. It could also be used for modelling of Hardware Design architecture. There are many FEV-related requirements that assume that there is a possibility to model and simulate electrical networks.

2.8.2 Input models for Modelica Exchange

The ME+ plugin by KTH for Modelica transformation takes a FAA or a FDA architecture description, together with the corresponding attribute quantification constraint annotations, as the input model. The modelling support is given as a part of the EAST-ADL Behaviour Description Annex.

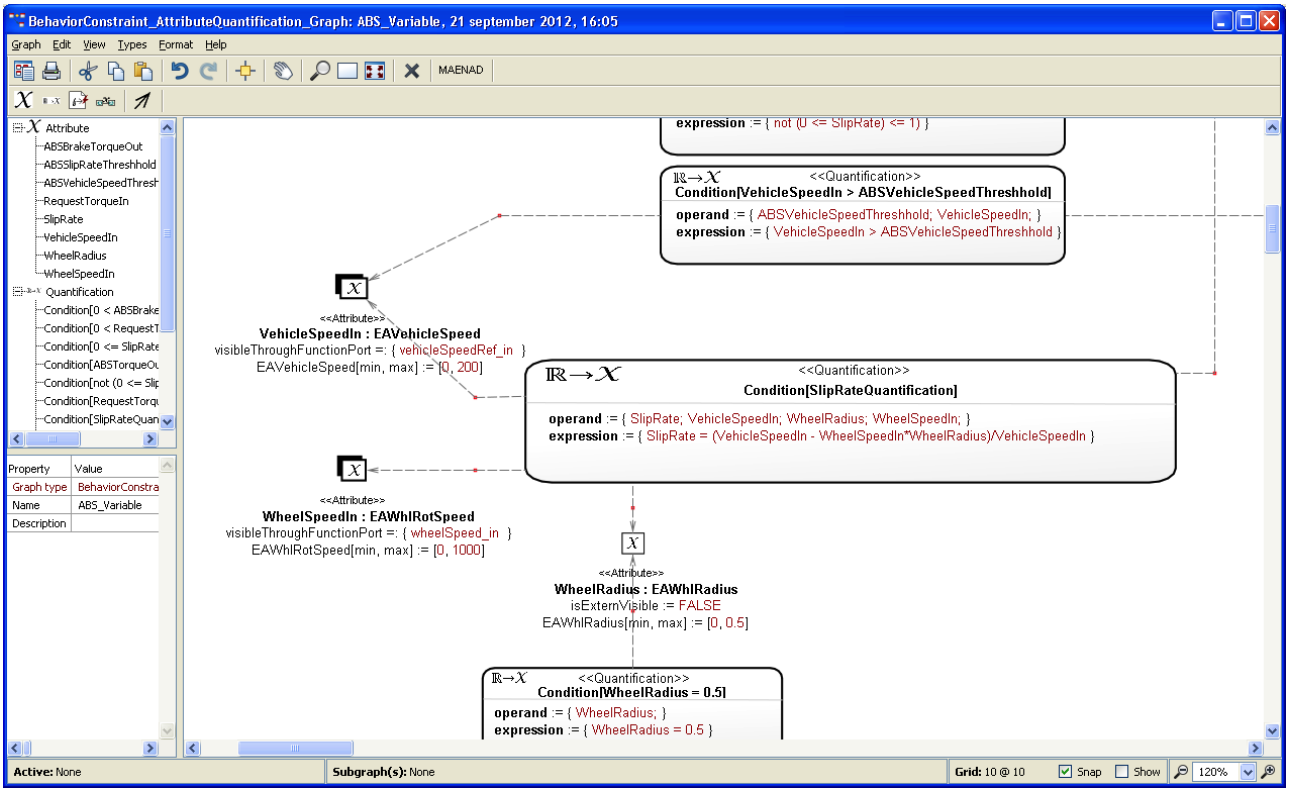


Figure 29. Screenshot of an example EAST-ADL Attribute Quantification Constraint Annotation for Modelica export.

2.8.3 Future plans

One way of co-use of EAST-ADL and ModelicaML is using the EAST-ADL structural model in combination with the ModelicaML dynamic model. The structural parts could be mapped more or less directly 1:1 into ModelicaML, the dynamic parts can be modelled by ModelicaML as the behavioural parts of a FunctionType. The FunctionType in this case should be seen as an AUTOSAR runnable.

Using the OpenModelica environment, equations could be modelled and simulated. Since this is a free software, it is attractive.

2.8.4 Requirements from WT2.1: Identifications of needs

Modelica exchange is mentioned in e.g.:

CON#0009: Annotate SysML/Modelica models with EAST-ADL stereotypes. On base of a defined mapping between SysML and EAST-ADL, the SysML model of the profile and mode selection logic shall be annotated with EAST-ADL stereotypes. Structural as well as behavioural elements shall be annotated with EAST-ADL stereotypes

CON#0012: EAST-ADL supports the definition of timing of constraints by the inclusion of the TADL language. A verification of the TADL constraints shall be possible. It is an option to verify TADL constraints either by the use of timing analysis techniques as provided by languages as MARTE or AADL or model simulation techniques as provided by Modelica. Within the scope of the project, it has to be evaluated, if the verification of timing constraints on base of these techniques is possible and samples shall be given.

CON#0014: VV Case Development, including fault injection and verification of model constraints in a Modelica simulation environment.

CON#0021: Virtual integration is an important use case during development within the ID4EV project. It is obvious that the physical demonstrator will not be available for a long time and the SW modules must be integrated in a virtual environment. The Modelica simulation environment is very well suited for integration C-code into the simulation environment.

© 2013 The MAENAD Consortium

2.9 Functional Mock-up Unit Import

The Functional Mock-up Interface (FMI) [5], defines Function Mock-up Units (FMUs) to exchange and integrate simulation blocks from different modelling tools. This is a standard that many simulation tools use for interchange of models, and co-simulation. The Function Mock-up Interface defines the input and output variables of each unit and also the data types of these variables. This is similar to EAST-ADL AnalysisFunctionTypes, so a transformation should be feasible, to make a modelling shortcut to create architectural models from simulation models.

2.9.1 Current status

There is an Eclipse plugin that imports the Function Mock-up Unit specification, called Function Mock-up Interface (FMI). Based on this information, an AnalysisFunctionType with the corresponding interface is defined in EAXML.

2.9.2 Input models for FMU import

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.

2.9.3 Future plans

The current plugin has flaws concerning the formatting of the EAXML file, which needs to be corrected. Further, additional options for the import can be foreseen. Currently, only AnalysisFunction is supported, but any specialization of the FunctionType is a candidate for import. Further, the FunctionBehavior construct is currently not created and populated with FMU information, such as the path to the FMU file.

A more extensive potential addition concerns export. Export in the form of FMU generation is probably not appropriate since EAST-ADL is not primarily a behavioural definition language. However, export of the FMU:s linked to FunctionBehaviors to a simulation engine would be useful. This would 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.

2.9.4 Requirements from WT2.1: Identifications of needs

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-ADL2 models

4SG#0003: Perform behavioural Simulation of EAST-ADL2 models according to performance evaluation standards

4SG#0004: Perform behavioural Simulation of EAST-ADL2 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.

© 2013 The MAENAD Consortium

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

© 2013 The MAENAD Consortium

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

© 2013 The MAENAD Consortium