Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2)

<table>
<thead>
<tr>
<th>Report type</th>
<th>Deliverable D5.2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report name</td>
<td>Guideline for Analysis-driven modeling and architecture evaluation method</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU (Public)</td>
</tr>
<tr>
<td>Status</td>
<td>Final release</td>
</tr>
<tr>
<td>Version number</td>
<td>1.0</td>
</tr>
<tr>
<td>Date of preparation</td>
<td>2010-05-03</td>
</tr>
</tbody>
</table>
###Authors

<table>
<thead>
<tr>
<th>Editor</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>David Servat</td>
<td><a href="mailto:david.servat@cea.fr">david.servat@cea.fr</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huascar Espinoza</td>
<td><a href="mailto:huascar.espinoza@cea.fr">huascar.espinoza@cea.fr</a></td>
</tr>
<tr>
<td>Yiannis Papadopoulos</td>
<td><a href="mailto:y.i.papadopoulos@hull.ac.uk">y.i.papadopoulos@hull.ac.uk</a></td>
</tr>
<tr>
<td>Henrik Lönn</td>
<td><a href="mailto:henrik.lonn@volvo.com">henrik.lonn@volvo.com</a></td>
</tr>
<tr>
<td>Martin Walker</td>
<td><a href="mailto:Martin.Walker@hull.ac.uk">Martin.Walker@hull.ac.uk</a></td>
</tr>
<tr>
<td>David Parker</td>
<td><a href="mailto:D.J.Parker@hull.ac.uk">D.J.Parker@hull.ac.uk</a></td>
</tr>
<tr>
<td>Rolf Johansson</td>
<td><a href="mailto:rolf_johansson@mentor.com">rolf_johansson@mentor.com</a></td>
</tr>
<tr>
<td>Matthias Biehl</td>
<td><a href="mailto:biehl@md.kth.se">biehl@md.kth.se</a></td>
</tr>
<tr>
<td>Mark-Oliver Reiser</td>
<td><a href="mailto:Mark-Oliver.Reiser@web.de">Mark-Oliver.Reiser@web.de</a></td>
</tr>
</tbody>
</table>

###The Consortium

- Volvo Technology Corporation (S)
- Volkswagen/Carmeq (D)
- Centro Ricerche Fiat (I)
- Continental Automotive (D)
- Delphi/Mecel (S)
- Mentor Graphics Hungary (H)
- CEA LIST (F)
- Kungliga Tekniska Högskolan (S)
- Technische Universität Berlin (D)
- University of Hull (GB)
Revision chart and history log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2010-05-03</td>
<td>First release based on existing material</td>
</tr>
</tbody>
</table>
List of abbreviations

Table of terms and abbreviations used in this document

<table>
<thead>
<tr>
<th>Term / Abbreviation</th>
<th>Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATESST2</td>
<td>Advancing Traffic Efficiency and Safety through Software Technology phase 2</td>
</tr>
<tr>
<td>MARTE</td>
<td>UML Profile for Modeling and Analysis of Real-Time Embedded systems</td>
</tr>
<tr>
<td>HiP-HOPS</td>
<td>Hierarchically Performed Hazard Origin and Propagation Studies</td>
</tr>
</tbody>
</table>
## Table of contents

Authors .......................................................................................................................... 2
Revision chart and history log ..................................................................................... 3
List of abbreviations .................................................................................................... 4
Table of contents ........................................................................................................ 5
List of Figures .................................................................................................................. 7
1 Introduction .................................................................................................................. 8
2 Scenarios ..................................................................................................................... 10
3 Concepts ..................................................................................................................... 12
   3.1 Architecture Analysis Concepts ............................................................................. 12
      3.1.1 Safety Analysis Concepts ............................................................................. 13
      3.1.2 Timing Analysis Concepts ......................................................................... 14
      3.1.3 Evaluation Context Concept ..................................................................... 18
   3.2 Architecture Evaluation and Optimization Concepts ............................................. 18
      3.2.1 Design Space Description: Architecture Options, Parameters and Substitutions ...................................................................................... 18
      3.2.2 Optimization Problem Description ............................................................... 19
   3.3 Summary ............................................................................................................. 19
4 Tool Infrastructure Specification ................................................................................ 20
   4.1 Functionality Specification ................................................................................... 20
   4.2 Technical Specification ....................................................................................... 25
      4.2.1 Preconditions .............................................................................................. 26
      4.2.2 Technological Requirements ..................................................................... 27
   4.3 Available plugin description ................................................................................ 27
   4.4 HiP-HOPS Plugin ................................................................................................. 28
      4.4.1 Technology ................................................................................................. 28
      4.4.2 Tool Integration .......................................................................................... 29
      4.4.3 Translation from EAST-ADL to HiP-HOPS .................................................. 30
      4.4.4 Involved Models and Metamodels ................................................................. 31
      4.4.5 Semantic Mapping Transformation ............................................................. 33
      4.4.6 Representation Transformation ................................................................. 34
      4.4.7 Benefits of the Chosen Decomposition ....................................................... 35
5 Optimisation Example ............................................................................................... 36
   5.1 York Model .......................................................................................................... 36
      5.1.1 Initial Design ............................................................................................... 36
      5.1.2 Improved York Model .................................................................................. 40
   5.2 Optimisation Process ........................................................................................... 41
      5.2.1 Enumeration & Provision of Optimisation Data ........................................... 42
      5.2.2 Timing Analysis .......................................................................................... 43
   5.3 Results of the HiP-HOPS Multi-objective Optimisation ....................................... 46
6 Method for Analysis-Driven Architecture Evaluation and Optimization.............................................53
7 Result evaluation and conclusions......................................................................................................55
8 Annex A: Terms and Definitions .........................................................................................................56
9 References...........................................................................................................................................60
List of Figures

Figure 1: ATESTST2 WT’s involved in this deliverable ............................................................... 8
Figure 2: Organization of analysis-specific model elements in MARTE ........................................ 15
Figure 3: Simplified canonical model for scheduling analysis .................................................. 16
Figure 4: Selecting between non-dominated solutions in a Pareto Optimal Set ......................... 23
Figure 5: Generalised optimization process .............................................................................. 24
Figure 6: Possible tool architecture for architecture evaluation ............................................... 25
Figure 7: Possible tool architecture for optimization ................................................................. 26
Figure 8: Initial York Model ........................................................................................................ 37
Figure 9: Alternative implementations/variants for Function1 & Function2 ............................ 37
Figure 10: Improved York Model ................................................................................................. 41
Figure 11: New implementations for the Software Functions ................................................... 41
Figure 12: Nominal End-To-End flow ......................................................................................... 44
Figure 13: Reconfiguration end-to-end flow ............................................................................. 44
Figure 14: Analysis Context ......................................................................................................... 45
Figure 15: Encoding Visualisation for solution #1 ..................................................................... 47
Figure 16: Graph for the System Slack optimization ................................................................. 48
Figure 17: Graph for End-to-end WCET .................................................................................... 49
Figure 18: End-to-End Slack ....................................................................................................... 50
Figure 19: End-to-End Best Case Execution Time ................................................................. 51
Figure 20: Possible workflow ...................................................................................................... 53
This document merges the following internal deliverables in one unique deliverable:

<table>
<thead>
<tr>
<th>#</th>
<th>Deliverable Name</th>
<th>WT</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>I3.5.1</td>
<td>EAST-ADL Update Suggestions for Multi-Objective Design</td>
<td>3.5</td>
<td>T15</td>
</tr>
<tr>
<td>I4.3.1</td>
<td>The EAST-ADL analysis platform</td>
<td>4.3</td>
<td>T9</td>
</tr>
<tr>
<td>I5.2.1</td>
<td>Guideline for Analysis-driven modelling and architecture evaluation</td>
<td>5.2</td>
<td>T12</td>
</tr>
</tbody>
</table>

As we can see from this table, these deliverables are concerned with WTs 3.5, 4.3, and 5.2 of ATESST2. The main reason of merging the three deliverables is that they are closely related to the same topic, that we call “Analysis-Driven Architecture Evaluation and Optimization”. Having these deliverables in the same document makes the whole deliverable fully comprehensible and self-consistent.

Figure 1 shows the involved ATESST2 tasks. WT 3.5 concerns the provisions to support multi-objective analysis and evaluation in EAST-ADL. WT 4.3 relates to the specification and tool implementation to support architecture analysis and evaluation. WT 5.2 articulates the previous two tasks by providing the guidelines for analysis-driven design decision making.

![Figure 1: ATESST2 WT's involved in this deliverable](image-url)
The initial objectives of these WTs in the initial proposal (DoW) can be summarized as follow (see the EAST-ADL DoW for full descriptions):

**WT 3.5 Multi-objective analysis and model-based architecture evaluation (Leader: CEA):**

- To extend the domain model of EAST-ADL with concepts required for multi-objective evaluation and optimization. They will include concepts to describe design spaces (i.e., architecture candidates, design constraints) in order to enable, after architecture evaluation, substitution, reassignment and patterns of replication of software and hardware components.
- To specify a multi-objective evaluation framework that will be backed up by an overall design method in WP5 and supported by specific plug-in and enhancement of the tool infrastructure in task 4.3.
- To develop the automatic optimization algorithm, assuming that such optimization is deemed to be feasible in the context of cooperative systems.

**WT 4.3 Support for analysis-driven modelling (Leader: KTH):**

- This task is dedicated to providing all necessary tool infrastructures to allow for architecture modelling driven by analysis results
- Integrate simulation monitoring capabilities to the Papyrus EAST-ADL modelling environment.
- Analysis feedback in terms of architecture evaluation and design hints to the user based on the results from WP 5.

**WT 5.2 Analysis Techniques Based on the EAST-ADL (Leader: CEA):**

- Define a method intended to predict, evaluate and explore architectural alternatives by analyzing and testing the accomplishment of the envisaged non-functional aspects (i.e., timeliness, performance and variability).
- Define methods to (1) facilitate the selection of specific non-functional requirements for candidate system architectures, (2) identify sensitivity and trade-off points in the architecture that, if changed, affect the desired non-functional properties, and (3) help make design decisions supported on an analytical understanding of the impact on non-functional properties.
- Deliver a method for analysis-aided design decision making in the form of a handbook.

Plugins developed within this set of tasks make up for the analysis platform of ATESST2 and are described in a separate wrapper document, entitled “The EAST-ADL analysis platform” (D4.3.1).

The rest of the document is structured as follows:

- Chapter 2 reflects the scenarios that we tare in scope
- Chapter 3 introduces the concepts that are needed to support multi-objective evaluation
- Chapter 4 provides a tentative specification for tool support
- Chapter 5 gives a report on an experiment conducted on an example
- Chapter 6 gives insight on methodological aspects
- Chapter 7 wraps results and draws conclusion
2 Scenarios

The scope of analysis-driven architecture evaluation and optimization is in the ATESST2 project limited by the scope of the architectures to be described by EAST-ADL. This means that the architectures to be analysed and optimized are the ones that appear on the analysis level and on the design level respectively. On Vehicle level, there is no architecture, but a number of feature models, and on implementation level, the architecture description language is defined by AUTOSAR.

While the initial proposal provides some key specifications on the technical objectives, there is a need to identify precise areas of interest to focus the related project tasks. The ATESST2 deliverable D2.1 “State of practice and State of the art” [5], Section 3.1.5, provides a good survey on “Analysis-Driven Architecture Evaluation and Optimization” that outlines technical issues of interest for the automotive industry, and which can be efficiently tackled with model-based approaches. Note, however, that in the context of the ATESST2 project it may be necessary to limit the scope in certain areas, e.g. limit optimization only to certain quality attributes such as safety or timeliness.

From that document, three different scenarios can be distinguished: analysis, evaluation, and optimization (for detailed definitions, see Annex 1).

(a) Architecture Analysis

In our context, architecture analysis is related to specific analysis reasoning frameworks used to predict different quality attributes. Quality attributes are specific criteria that can be used to judge the operation of a system (e.g., performance, safety, availability, resource usage). The terms “performance analysis” and “safety analysis” are commonly used to describe analysis of those quality attributes. Note that one architecture may include variability. Performing analysis of an architecture includes analysis of all its valid variants. Given assumptions on the distributions of binding decisions on the vehicle level, and the configuration decision information, the resulting analysis reflects the value of all the variants that can be derived for a given attribute.

(b) Architecture Evaluation (Assessment)

Evaluation is the process of examining options and assessing their relative merits. Architecture evaluation is often used to describe assessment either before or after architecture analysis. It is also often used to describe the process of proving whether the quality attributes of an architecture option meet system requirements. Evaluation can be also used to select one out of many architecture options or candidates. Architectural options might be architecture patterns, scheduling policies, fault tolerant schemes, alternative subsystems, or anything else about which a decision is needed. When the space of possible solutions is large, evaluation can be done by “design space exploration”. Different trade-off strategies can apply to make “right” architectural selections. One strategy is optimization.

(c) Architecture Optimization

Optimization is the process of finding the best solution from all feasible solutions regarding certain objectives and subject to certain constraints. Maximizing performance and minimizing fuel consumption of a vehicle, and minimizing weight while maximizing the...
strength of a particular component are examples of optimization problems. A (multi-objective) optimization problem can be formalized as the searching of a solution \( x \) – i.e. an optimal design - (element of solution space \( X \) of possible designs), which optimizes a vector of objective functions \( f(x) = [f_1(x), f_2(x), f_3(x), \ldots, f_n(x)] \). For example, one optimization problem is to search for Pareto optimal (i.e. non-dominated) solutions. A solution \( x_1 \) dominates another solution \( x_2 \) if \( x_1 \) matches or exceeds \( x_2 \) in all objectives.

All these three activities can be either manual, semi-automatic (under guidance), or automatic (according to some algorithm). This means that, independently of tool support aspects (WT 4.3), conceptual (WT 3.5) and methodological (WT 5.2) aspects should cover capabilities generic enough to be useful for different analysis, evaluation, and optimization concerns.

Particularly, in the automotive industry, some aspects are of primary importance from the validation and verification viewpoint. Among dependability quality attributes, safety and timeliness are at the core of model-based approaches, and of EAST-ADL in particular. Safety concerns have been extensively included in EAST-ADL since the ATEST phase 1. Timeliness concerns have been delegated to TIMMO and its TADL language, which has then been integrated into the EAST-ADL language during ATEST2.

At first glance, safety and timeliness may seem the best quality attributes for focusing the work in the involved WTs. It must be noted that when we talk about the involved WTs, the scope is exclusively the above-mentioned activities (b) and (c). Architecture analysis, as the (language, tool, and method) provisions for supporting specific analysis reasoning frameworks, is not a core part of the involved WTs. However, the identification and integration of available analysis technologies needs to be done.

Closer examination reveals, however, that the envisaged approach for analysis-driven architecture evaluation and optimization should be general enough for supporting arbitrary kinds of quality attributes. For instance, we should be able to evaluate architecture candidates regarding any quality criteria by generalizing the evaluation/optimization problem. This is indeed the case in optimization approaches, where an optimization engine (e.g., HiP-HOPS optimization engine) could include cost and any quality attribute (such as safety, reliability, availability, performance etc), albeit with two constraints [5]. Firstly, there has to be a technique capable of assessing the quality of the model in terms of the chosen attribute (e.g. in the case of safety, it must be possible to evaluate the safety of the system). Secondly, there must be the necessary support to alter the chosen attribute in the model (e.g. in the case of reliability, one option would be the possibility of replicating a component).

For the first constraint, WT 3.2 provides the necessary support for model-based safety analysis, whereas MARTE [7] and TIMMO [2] provide inputs for model-based timing analysis. For the second constraint, a “post-analysis engine” should be developed to annotate back, or, in a more sophisticated scenario, to modify the EAST-ADL model. In a stepwise refinement of the ATEST2 approach, one can identify six levels or steps:

1. Means to model the system with enough information to do analysis
2. Means to automatically analyse EAST-ADL model
3. Back-annotation of analysis results in to EAST-ADL model
4. Means to define the design space
5. Assessment of architecture alternatives
6. Optimization, i.e. automatic modification of EAST-ADL model

In the remaining of this deliverable, we describe how these aspects can be supported from three different perspectives: the required concepts (chapter 3), the tool infrastructure specification (chapter 4), and the methodological support (chapter 6). As a proof of concept and interest for the approach we provide a report on a small optimization example where safety and timeliness concerns are evaluated (see chapter 5).
3 Concepts

In this section, we specify the concepts that will be taken into account in WT 3.5. This includes the kind of aspects that may be analyzed, evaluated, or optimized in a system architecture model, and how these aspects should be captured in the language (i.e., EAST-ADL).

Given the cross-conceptual nature of the work involved here, it was not intended to provide language extensions out of our activities here – as is the case for other 3.x tasks - but rather to take a step back and check how ready the language is to support such a cross-concern approach. This paves the way for future language extensions in subsequent, focused project.

3.1 Architecture Analysis Concepts

The scope for architecture analysis is partly dependent on the abstraction level of concern. As mentioned before, the scope here is limited to the analysis and the design levels, respectively. On analysis level the architecture problem in EAST-ADL consists of identifying the FunctionPrototypes and interconnect these in such a way that all features are realized. On the design levels there are three different architectures in parallel, and the architecture problem is extended to both the allocation decisions, and decision on introduction of redundancy.

Common for both these abstraction levels are that there is no architect having detailed information on all components. Different companies want to protect their IP, thus hiding much detailed information. This means that architecture analysis has to be performed on the limited information that is shared to the company doing the actual architecture. To guarantee that enough information is shared among companies there are dedicated activities to assure this.

One example is the TIMMO project in which a timing model (called TADL) is defined for sharing timing information without giving away other design IP. As TADL is now an integrated part of EAST-ADL, we hence have possibilities to express all inputs given from other companies within the EAST-ADL model. Furthermore, all timing requirements on the system can also be expressed in the EAST-ADL model. The problem of doing an architectural analysis w.r.t. timing can hence be performed with an EAST-ADL model as input, without requiring any further “analysis model” – provided a tool support is available. Given the fact that timeliness analysis tooling was not part of the ATEST2 planning, an imperfect solution is provided for the moment. It relies on using MARTE timing annotations and not directly EAST-ADL concepts – we go back to this later in the document. However mapping strategies are under work in the ADAMS project.

As concerns safety analysis WT3.2 has provided the necessary extensions to add architectural information like error models of the functions, failure behaviour of the system, error propagations, etc. Tool support has been developed constantly during the project – see D4.3.1 and later in this document.

For other qualities further investigation is needed.

One assumption here is that we will analyse one architecture candidate, which may itself include several variants. For example there might be two different architecture candidates, where one is to have the same FunctionPrototypes including both mandatory and optional functionality, and the other is to have two different FunctionPrototypes for the mandatory and the optional functionality, respectively. Depending on the assumption of the take rate of the optional functionality, the analysis may give different answers which one of these two architecture alternatives is the best one.

In this context, the interest is to ensure that we can extract, from EAST-ADL, a canonical analysis model, both for safety and timing, with enough information to feed analysis tools. In other words, the goal of this section (3.1) is to determine (or point out to the right ATEST2 WT or external source) the information completeness regarding analysis information needs.
3.1.1 Safety Analysis Concepts

The modelling support for this kind of analysis is developed in WT 3.2 (not a part of this document). A summary of the kind of information required for safety analysis is provided here.

In order to be used as the basis of a safety-driven design process, EAST-ADL provides native support for representing safety concepts such as hazards, safety requirements, and error propagation. In EAST-ADL, this support is provided by the error modelling part. The existing error model developed in ATESST1 has been extensively updated with greater support for error propagation modelling (vital to support safety analysis techniques like FTA and FMEA), decomposition of safety requirements (involving hazard analysis and allocation of required ASILs), and multi-perspective analysis by means of error propagation via the allocation relationship between a software and an hardware architecture.

At the centre of the error model are the ErrorModelType and ErrorModelPrototype classes, analogous to the FunctionType and FunctionPrototype respectively. This allows an ErrorModelPrototype to represent a unique occurrence of the ErrorModelType that types it, facilitating reuse of a single ErrorModelType across many functions. These error model entities are intended to represent the failure behaviour of a system, subsystem, or function in the EAST-ADL nominal model. The error model is hierarchical, and as such, any system, subsystem, or function may be described by its own error model.

The actual behaviour of the target architectural entity is described by an ErrorBehavior. The ErrorBehavior is the primary provider of the interface to external safety analysis tools such as HiP-HOPS; by means of the ExternalErrorBehavior specialisation, the description of the failure behaviour of a function or other entity can be described in a tool-specific way. For HiP-HOPS, the ExternalErrorBehaviour contains a logical expression that describes how a combination of input or internal failure events can lead to output failures being propagated from the outputs of the function. The internal failure errors (also known as basic events in a HiP-HOPS or FTA context) are described explicitly in the error model by means of the InternalFault, which possesses quantitative data – the failure and repair rates of that event – as well as a failure type to identify the event.

Propagation of failures is modelled by means of FaultFailurePorts and associated ErrorPropagationLinks. The FaultFailurePort represents error events that traverse ErrorModelTypes (i.e. that have external consequences and thus propagate to other functions); the ErrorPropagationLink connects FaultFailurePorts of different ErrorModelTypes. FaultFailurePorts can describe different error types. Finally, FaultFailurePorts are linked to a HazardCause, which in turn is linked to a Hazard and thus provides a means of obtaining a safety requirement (specifically, an ASIL) and enabling ASIL decomposition to take place throughout the error model hierarchy.

The error model is not specifically intended to have a 1:1 mapping between nominal architecture (e.g. FunctionTypes) and the error model entities (e.g. ErrorModelTypes). This is in order to provide the maximum degree of freedom when modelling the error behaviour of a system. However, in order to be able to perform analysis and optimization with HiP-HOPS a 1:1 mapping has been imposed between nominal and error propagation architectures. The reader can find additional information in deliverable D3.2.2 on various alternative modeling styles.

Error model support for multi-perspective analysis (e.g. between software and hardware) is provided by the 'allocationConstraint'. This links entities in one perspective (e.g. design) to entities in another (e.g. hardware). In addition, variability constructs can be applied to these allocation constraints, raising the possibility of performing a form of optimisation simply by means of changing the allocation of functions to hardware.

To meet the need for dependability analysis and to assess and evolve system designs with respect to safety, reliability, performance, and cost, we use the HiP-HOPS analysis tool. To support the new requirements of the ATESST2 project, the tool has been extended with a number of new concepts, including the capability to model multiple perspectives (e.g.
Hardware/Software/Middleware), new concepts to support the ISO 26262 safety workflow, particularly a novel algorithm for decomposing safety requirements in the form of ASILs, and also improved architectural optimisation capabilities. To facilitate the interface between HiP-HOPS and EAST-ADL, a new plugin was created that makes use of model transformation technology.

Finally, HiP-HOPS also offers automatic design optimisation capabilities that have been greatly enhanced during ATESST2. The previous PESA-II algorithm was replaced by a newly developed variant of the NSGA-II algorithm, which is superior both in performance and in terms of results obtained. Together with the new analysis extensions described above, this introduces the capability to perform multi-objective architectural optimisation of EAST-ADL models with respect to safety, reliability, and cost. The design space to be explored can be defined by means of the variability constructs in EAST-ADL. Other optimisation objectives, e.g. timing, were also investigated.

### 3.1.2 Timing Analysis Concepts

Modelling for timing analysis is supported by TIMMO’s TADL. In addition, MARTE provides a good basis to provide timing annotations into EAST-ADL models. Research and tool support for this alternative is under way in the EDONA French project. The results of the EDONA project to connect EAST-ADL models augmented with MARTE timing annotations are under consideration within the ADAMS project, which involves partners from both projects. The following section provides a summary of the main elements in both languages, EAST-ADL (and TIMMO) and MARTE.

#### 3.1.2.1 EAST-ADL/TADL

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

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

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

In EAST-ADL, timing requirements are divided into various kinds of delays (or latencies) for single time-consuming modeling entities as well as specific requirements for temporal synchronization of input or output data. The delays are either end-to-end delays, which are subject to segmentation along the functional decomposition track (e.g. end-to-end delay for a top-level function), or the delays form part of an end-to-end timing chain, and thus constitute segments of such an end-to-end timing chain. Furthermore, delays can be sub-classified as being induced either by data...
transformation performed by a “FunctionPrototype” or data transmission via a “FunctionConnector”.

More precisely, EAST-ADL Timing concepts is based on

- Events: relate to EAST-ADL and AUTOSAR structural entities and depict observables: e.g. data arriving on a port, triggering of function execution, etc.
- EventChains: bind together events to establish sequences/relations between events e.g. to capture a complete end-to-end flow requirement between data sent by a sensor to output by an actuator.
- Constraints: put temporal constraints on sets of events or on event chains, e.g. deadlines to be met, expected delays, patterns of data arrival, synchronization of outputs on a set of ports, etc.

3.1.2.2 MARTE

The domain model for non-functional analysis in MARTE is organized around the notion of Analysis Context (see Figure 2). An analysis context is the root concept used to collect relevant quantitative information for performing a specific analysis scenario. Starting with the analysis context and its elements, a tool can follow the links of the model to extract the information that it needs to perform the model analysis.

**Figure 2: Organization of analysis-specific model elements in MARTE**

Within the application model, “workload behavior” describe how often events will arrive and how much data are provided as inputs to the system, and how response events are generated in return by the system. In addition, workload behavior models provide information about the processing behavior that is used to execute the various tasks.

On the other hand, “resource platform” models provide information about the properties of the computing and communication resources that are available within a system, such as processor speed and communication bus bandwidth. The system model thereby captures information about the applications and the available resource platform of the system, and it also defines the mapping of application tasks to computation or communication resources.

The main analysis techniques in the scope of MARTE are scheduling-aware timing analysis. This kind of techniques offers a mathematically-sound way to determine whether a system will meet its timing requirements and how it could be balanced (e.g., task allocation, priority assignment, component deployment) by still respecting timeliness. To this purpose, scheduling analysis tools are a key component in this chain. Implementation of scheduling analysis tools depends a lot on the kind of scheduling analysis supported. In particular, the domain of scheduling analysis for automotive applications has received special attention in recent real-time systems literature. A comprehensive summary of analysis techniques for automotive applications is provided in [8]. Among relevant advances in this field, the holistic approach [9] extends the classical single-
processor scheduling theory and applies it for specific combinations of input event models, resource sharing and communication policies. The global view on the system allows taking global dependencies into account (offsets between the activation instants of tasks), thus providing tightly calculated analysis bounds. A different approach for distributed architectures is the compositional scheduling analysis [10] [11]. The basic idea of this approach is to break down the analysis calculation into separate local component analyses (e.g., mono-processor analysis with RMA) and to integrate them for system-level analysis by evaluating the propagation of event stream models.

In our experiment (see chapter 6), we worked with the MAST tool from the University of Cantabria, which supports classical RMA analysis and holistic analysis [12][13].

Figure 3: Simplified canonical model for scheduling analysis

Figure 3 shows a simplified canonical model of the modeling features required for scheduling analysis, which is discussed in this section.

Timing constraints

We can distinguish at least three abstraction layers of time constructs:

- Time-related abstraction layer, level 1: time is modeled as a set of fundamental notions such as time instants, durations, time bases, or clocks. They can be specified by relative/absolute durations (maximum time in intervals) or instants (occurrence of a timeout event).

- Time-related abstraction layer, level 2: MARTE provides mechanisms to annotate timing requirements and constraints in models. Basic timing constraints include deadlines and maximum jitters. One key-modeling feature is the concept of observation. Observations provide marking points in models to specify real-time assertions. Some typical assertions have been predefined in ready-to-use patterns, such as jitters or conditional time constraints.

- Time-related abstraction layer, level 3: time concepts are defined as part of the behavior, not mere annotations. This set of constructs cover both physical and logical time. Most of the current scheduling analyses use the notion of physical time as measured by a unique time base. However, distributed applications often experience problems for agreement on consistent time
reading due to clock synchronization. This means that scheduling them depends on different time bases, and therefore constraints must refer to specific clocks.

End-to-end flows
In MARTE, end-to-end flows describe logical units of processing work in the system, which contend for the use of processing resources (e.g., processors and buses). Let us also notice that firstly data and control can be part of the processing, and secondly, different kind of timing constraints can be attached to end-to-end flows (e.g., deadlines or output jitters).

One important feature in MARTE is that end-to-end flows can be represented in behavioral views (e.g., Sequence or Activity diagrams) complementing component models. This approach allows modelers to specify multiple end-to-end flow configurations that could be likely related to (a) specific operational modes, (b) alternative execution chains, or (c) different quantitative scenarios of activation parameters or other non-functional annotations.

Activation events
Both event-triggered and time-triggered paradigms are often involved in automotive applications. Event-triggered means that tasks are started, or messages are transmitted, following the occurrence of one (or a conjunction of) significant event (e.g., "a door has been opened"). Time-triggered consists of tasks started, or messages transmitted, at predetermined points in time, usually periodically.

In MARTE, activation models are denoted by means of workload events. Workload events can be modeled under different forms: by known patterns (e.g., periodic, aperiodic, sporadic or burst), by irregular time intervals, by trace files, or by workload generator models (e.g., state machine models). Workload events also enable to specify additional parameters for periodic and aperiodic patterns such as jitters, burst parameters, and distribution probabilities.

SW and HW resources
What is needed for scheduling analyses is to take into account the impact of the OS and hardware resources on applications (e.g., overheads due to the OS and the stack of communication layers or throughputs and bandwidths of underlying networks). Among these aspects, access protocols to mutual exclusive resources are of paramount importance in scheduling analysis of modern multiprocessor architectures.

The MARTE analysis model distinguishes two kinds of processing resources: execution hosts, which include for example processors, coprocessors and controllers, and communication hosts, which include networks and buses.

Processing resources can be characterized by throughput properties as for example processing rate, efficiency properties such as utilization, and overhead properties such as blocking times and clock overheads.

The system model shown in Figure 3 thereby captures information about the applications and the available resource platform of the system, and it defines the mapping of application functions to OS resources, ECUs and buses.

3.1.2.3 Summary

Both languages are complementary. In general MARTE is more focused on a behavioral description of a system in which timing information is a first-class input, whereas EAST-ADL assumes a fixed execution semantics for the functions, resulting in a more structured and static nature of models. Both approaches rely on end-to-end flow descriptions. The main difference is the level of detail in which both languages regard allocation. On this point MARTE requires more: an initial mapping from functions to tasks is needed, which is explained by the fact that MARTE
considers the whole design life-cycle, where EAST-ADL delegates to AUTOSAR the details of the implementation levels.

Tooling for timing analysis is provided as a result from CEA LIST work in the EDONA project. It is for the moment MARTE-centric and takes a MARTE-annotated model as input and interacts with the MAST schedulability analysis tool (from Univ. of Cantabria, Spain).

This concludes the part dedicated to architecture analysis concepts. We believe the same kind of canonical view can be built for various quality-centered description, and language extensions defined to integrate them in a general framework. The latest revision of the EAST-ADL language (with the introduction of a language core and satellite extensions) goes in this direction.

### 3.1.3 Evaluation Context Concept

Due to the specific tools used for safety and timing analysis in ATESSST2 (scenario based techniques and NOT state based techniques), it is important to bound system model elements to a particular **analysis or evaluation scope**. As automotive applications become more complex, there is often a need to represent a system by multiple analysis models, corresponding to different application-platform allocations, abstraction levels, operational modes, or different quantitative values of non-functional parameters. Unfortunately, EAST-ADL does not have such evaluation scope construct. Starting with the evaluation context and its parameters (e.g., quality properties), a tool can follow the links of the model to extract the information that it needs to perform analysis. Analysis results will be then able to be annotated back in EAST-ADL models to take them into account for architecture refinement.

One alternative is to define explicitly such concept in EAST-ADL as a kind of packaging or composite construct. Another alternative is to let the evaluation scope out of EAST-ADL and implemented as part of the tool infrastructure. In the latter case, the traceability and historical results should be maintained by the tool.

### 3.2 Architecture Evaluation and Optimization Concepts

A typical scenario of tool usage for evaluation would take the results of analysis and use it to make decisions about a design model, e.g. to drive further development, or to determine whether or not the design meets system requirements. In this context, however, it can also be used to select one (or more) model options.

This model options can be manually selected or can be automated from an optimization tool. In any case, evaluation should explore the design space to suggest different alternatives, each with optimal (or at least improved) characteristics in one or more objective attributes. While optimization is not the only possible approach for generating these different options, and evaluation is not limited to choosing between them, these activities will also be expected to be conducted in a joint manner to suggest possible improvements to design models.

### 3.2.1 Design Space Description: Architecture Options, Parameters and Substitutions

For optimization to take place, it must be possible to define a design space. This can be done by providing different parameters for various model attributes, some of which may be conflicting, e.g. end-to-end response times vs. deadlines for timing analysis or cost vs. unavailability for safety analysis.

This may be achieved by means of the **variability mechanisms** in EAST-ADL (or an extension thereof) or may be achieved in another way. In some cases it may also be desirable to define or extend the design space by means of different architectural implementations, not just parametric information; this will likely be more complex but would allow a greater degree of flexibility during
optimization, e.g. to consider different fault tolerant subsystem architectures or perhaps even different H/W-S/W allocation strategies.

These variants and/or alternative parameters are necessary for optimization to take place, as they provide the scope for different model possibilities to be explored and evaluated. However, for evaluation to be meaningful, it is important that these variants and parameters are substitutable, i.e. that one parameter or variant can be substituted with another.

The most important factor to take into account here is the notion of substitutability, i.e. for optimization to be valid it must be possible to replace set of parameters or variants with another set. In practice, this most often means that if an entity performs a certain function and possesses a certain interface, then substituting it for another variant or changing its parameters should not result in an incompatibility within the system being modelled. As a crude example, the steering wheel of an automobile is unlikely to be a valid substitution for one its tyres.

EAST-ADL is well defined to allow both design space description and exploration as variability and feature modelling played a central role in its definition right from the very first project (EAST-EA).

Tool support is available – see results from WT3.3 – which can be used to deal partly with the scenario described here. However dedicated support would be needed to produce design space presentation to an end-user – see later sections.

### 3.2.2 Optimization Problem Description

The description of an optimization problem can vary according to the kind of quality attribute considered, and the description of the design space (e.g., architecture alternatives, parameters).

Some typical optimization aspects in automotive applications for safety and timing analyses are the following:

**Optimization Objectives.** For both timing and safety analysis, a typical objective can be to maximize the margins on the constraints. Given a distributed development with several companies involved. For timing analysis, these may include end-to-end response times vs. deadlines, resources reservation vs. resources utilization, maximum jitters vs. jitter constraints. For safety analysis, objectives may include competing parameters like cost and unavailability (i.e. the probability that the system will be inoperable at any given time).

**Search Space.** For both timing and safety analysis the way to express the search space is depending on the abstraction level of the architecture to optimize. The allocation decisions of functions on hardware resources may have a large influence on both timing and safety. Hence, searching through different allocation alternatives may be important when being on design level. For safety, different redundancy strategies are also candidates on the design level. On analysis level, it is only about to find a Functional Analysis Architecture, where neither allocation nor redundancy is an issue. On this level there is more a focus of relating the constraints for different component candidates with the constraints of the features they are realizing. Note, that things like allocation of applicative logic into tasks (e.g., find the best runnable-to-task allocation configuration), task priority assignation, etc is not within the scope for ATESST2 as these are related to the implementation level defined by AUTOSAR.

### 3.3 Summary

Concepts are available in EAST-ADL to cover some particular quality-centered analysis, especially safety and timeliness. Furthermore the variability concepts enable to lay the ground for a real design space description and exploration, provided tool support is available. This is the object of the next chapter.

We have spent quite some time in describing the concepts involved in timing and safety analysis, to help readers grasp the experiments we have made on a joint timing and safety optimization scenario – see chapter 5.
4 Tool Infrastructure Specification

4.1 Functionality Specification

It is important to note that this tool infrastructure merely identifies the kinds of functionality and
information required for this process and is not intended to prescribe how, or even whether, that
functionality is to be partitioned among plugins or tools. At one extreme, a single complex
plugin/tool may realize all of these functions or, at the other extreme, a different plugin/tool may
realize each one.

The general functionality that is required for optimization is a product of the process. The three
main activities defined above are analysis, evaluation, and optimization. Analysis is the activity
used to produce estimates about certain qualities of a system, e.g., performance in the case of
timing analysis or safety & reliability in the case of dependability analysis. There may be multiple
analysis functions, each tailored for a different system quality, or there may be functions capable of
analysing multiple qualities or attributes at once. In general, however, an analysis function requires
a model – in this case an EAST-ADL system design model, produced using Papyrus UML – which
has been annotated or enriched in some manner with the necessary information required to
perform that analysis. Such an analysis may also require certain constraints to be imposed on the
model. This leads to the requirement for a model validator function:

• Model Validator

The model validator function is used to constrain and validate EAST-ADL models for specific
usage in architecture evaluation and optimization. In principle, the model validator does not
have to be aware of the semantics of the different analysis frameworks (e.g., safety analysis).
However, it should be able to enforce the rules and constraints defined in WT 3.5, such as
restrictions on how to model the design space (variants or parameters), objective functions,
and design constraints (this capability should be standard on any analysis framework that
supports EAST-ADL). In the case of safety analysis, for example, it is likely to be necessary to
constrain the error model to a 1:1 mapping between ErrorModelTypes and FunctionTypes. Of
course, the semantics of specific analysis frameworks can be supported elsewhere (WT 3.2 for
safety analysis and TIMMO/MARTE for timing analysis).

Assuming the model is valid, it can then be exported or passed in some other manner to the
relevant analysis functions. However, because there will likely be many such analysis functions,
each with their own separate (or collective) requirements for annotated data and their own set of
possible results, there is a requirement for some mechanism to effectively organise this
information. This can be described as the evaluation or analysis scope. In a typical scenario of
EAST-ADL profile usage, the system design model (in Papyrus UML) is annotated using the
appropriate EAST-ADL stereotypes. Eventually, additional model views may need to be defined to
represent specific quality attribute analysis information (e.g., a representation of error models for
safety analysis). Once the annotation is complete, the model is passed from the model editing tool
to the analysis tool where it is analyzed and the results feed back to the editing tool (this allows
viewing of the results in the same environment and form in which the model was produced). As
system design models become more and more complex, there is often a need to represent a
system by multiple evaluation models, likely corresponding to different application-hardware
allocations, abstraction levels, operational modes, or different quantitative values of quality
parameters.

Unfortunately, EAST-ADL does not have such constructs. What it is required is an evaluation context
notion, which enables EAST-ADL model elements to be bound to a particular evaluation scope. Starting with the evaluation context and its parameters (e.g., quality properties), a tool can
follow the links of the model to extract the information that it needs to perform analysis. Analysis
results will be then able to be annotated back in EAST-ADL models to take them into account for architecture refinement. In this way, pre-processing and post-processing tools will filter a parameterized EAST-ADL model in order to produce a “view” suitable for model transformation, and then for taking analysis results and substituting the initial parameters. This pre- and post-processing gives rise to the requirement for additional functions:

- **Analysis Pre-processing and Post-processing**

As described above, analysis functions may require their own particular views on the EAST-ADL model. There is the need for a function to translate the information in the model, subject to a particular evaluation context, and export it or pass it to the analysis function in a particular format. This may entail a certain amount of pre-processing and model transformation.

Secondly, once the analysis function has completed its work, the results of the analysis need to be imported back into the model editing tool to allow them to be viewed in the same environment as the original model. This may also involve annotating the original model with the results, e.g. by means of V&V constructs in EAST-ADL. This would provide a means of documenting the model and ensuring that the results of an analysis are not ‘lost’. Similarly, some form of post-processing function may be necessary to incorporate the results into the EAST-ADL model (which may be described as 'back-annotation').

In practice, these functions are likely to be provided by Papyrus plugins.

Analysis is however only the first of the activities mentioned above. The second and third activities are more closely linked: *evaluation* and *optimization*. Evaluation takes the results of analysis and uses it to make decisions about a design model, e.g. to drive further development, or to determine whether or not the design meets system requirements. In this context, however, it can also be used to select one (or more) model options produced by the optimization activity, which explores the design space to suggest different alternatives, each with optimal (or at least improved) characteristics in one or more objective attributes. While optimization is not the only possible approach for generating these different options, and evaluation is not limited to choosing between them, these activities will also be used in conjunction to suggest possible improvements to design models. However, before this can take place, it must be possible to define a design space:

- **Design space handler(s)**

As noted above, a model produced by a model validator may have variants or parameters included within it. These variants and parameters define the design space and allow a single model to be analyzed for different variants or parameter values without having to produce an entirely new model every time the architecture or values change. However, this creates a need for a design space handling function, a function that is similar in purpose and form to the C pre-processor. It takes a design space model and, by substituting an appropriate set of variants or parameter values, it produces a different model.

These variants and/or alternative parameters are necessary for optimization to take place, as they provide the scope for different model possibilities to be explored and evaluated. However, for evaluation to be meaningful, it is important that these variants and parameters are *substitutable*, that one parameter or variant can be substituted by another. This is a constraint that may need to be included in the model validation function, if not enshrined in the EAST-ADL model itself. These variants and parameters may also need to be tailored to different qualities that may be analysed, e.g. the requirements for a safety analysis may not be the same as the requirements for a timing analysis. These differences will also need to be handled by the design space functionality.
It is possible that such functionality will require some form of custom-built graphical user interface (GUI), particularly as the module may be required both before and after evaluation/optimization. It can be used at the beginning of the evaluation/optimization process to define and pass the right set of variants, to describe the design space, and also at the end of the evaluation/optimization process to select one of the optimal architectures (e.g. from amongst the Pareto Optimal solutions) to present to users the corresponding EAST-ADL model. Thus, the EAST-ADL user should simply have to choose the solution that is of interest for the architecture at hand, rather than take the information in the optimization results and redefine the model manually to suit its characteristics.

Assuming there are the requisite analysis functions and the design space can be defined and exported successfully (as ensured by the model validator), then there needs to be a way of generating different possible architectures. This functionality must be provided by an optimization function, which also implies that there is a corresponding evaluation function for the various optimization objectives being used:

- **Optimization function**

  As described earlier, a number of different approaches to optimization exist, but in general they all generate new alternative models by exploring the design space that has been defined. This is done by taking different parameters or choosing different variants and reconfiguring the model accordingly. Because the design space can quickly become very large, it is frequently necessary to have some form of automated algorithm in place to explore this design space in an efficient manner. Examples of such algorithms include genetic algorithms, Tabu search, and simulated annealing. These all focus their explorations to try to locate superior alternatives and thus eventually lead to optimal solutions, producing better results without needing to explore the entire design space (which will likely be infeasible). This process is iterative and requires one or more optimization objectives – the criteria by which the generated alternatives will be judged.

  Once different alternative options have been generated, however, they need to be evaluated in order to determine whether they are better or worse than the current option(s). This requires an evaluation function:

- **Evaluation function**

  Evaluation in this context means comparing the various solutions proposed by an optimization algorithm and comparing them to determine which are better and which are worse. This will involve applying an analysis function for each of the optimization criteria being used. In the case of single-objective optimization, it is likely that only a single analysis function will be required, but for multi-objective optimization, more than one analysis function may be required.

  Traditional meta-heuristic optimization strategies operate on a single objective. Individual solutions can be evaluated and compared based on the analysed value of that objective. One can be systematically chosen as better over another, eg. Solution X is more reliable than solution Y. One approach for achieving multi-objective optimization is to combine the multiple objectives (perhaps with a weighted sum or product of the objectives, or the mechanism of treating some of the objectives as constraints and penalising constraints violations) into a single objective. This is generally not desirable however. The combination of multi-objectives is often unintuitive and the selection of the weights is prone to error. How does one combine a reliability probability with a cost? Multiply them together? How should they be weighted? Choosing the incorrect parameters in these cases will leave an ineffective optimization.
The use of Pareto-optimality in multiple-objective optimization has established itself as an alternative. As there is no systematic way to choose between each of the objectives a different fitness evaluation is required. The goal is no longer find the most reliable or find the cheapest solution. The goal is find the Pareto-optimal set. Furthermore the goal is to have a Pareto-frontier that is evenly spread across the search space. This has a beneficial by-product in that each of the Pareto-optimal solutions is an non-dominated trade-off for each of the objectives. Thus for example there may be a more reliable solution but it will cost more, or conversely a cheaper solution may be available but it will be less reliable. The even spread of the Pareto-frontier ensures that the resultant set of non-dominated solutions will present as diverse a choice as possible for the designer when the optimization process is completed.

Once estimates of the system qualities have been produced through these analyses, the evaluation function must be able to compare them in some way in order for the next iteration of the optimization to continue. In general, this means discarding poor options and preserving good alternatives, but this is dependent on the optimization algorithm being used.

These functions are all required to provide support for a generalised optimization process along the following lines:
Figure 5: Generalised optimization process
4.2 Technical Specification

From a tool implementation perspective, it seems that a modular approach based on building blocks may provide different levels of assistance and degrees of automation. Figure 6 shows a proposal of tool architecture for WT 4.3.

In this approach, two kinds of building block are required in addition to analysis tools. First, a tool that allows for transforming the EAST-ADL model into the analysis tool-specific model and then back into an annotated EAST-ADL model. This may impose some requirements in the EAST-ADL metamodel. For timing the back annotation requires updating models e.g., adding a response time to an execution chain. For safety, the failure rate of a composite component may be back annotated from the analysis of its parts.

Second, a design space exploration tool chain needs to take analysis results and to evaluate them according to a set of criteria. Then a modification engine may iteratively search other variants until it finds an optimal solution (or a set of optimal solutions if there are multiple objectives). For defining the design space, there are two options. Either we use a kind of variability mechanism in EAST-ADL to account for different architectural variants, or alternatively we can use a parameterization mechanism in EAST-ADL to account for different architecture "variables". The first option is time consuming but allows capturing structural dependencies in the model. The second option may be used for modifications without dependencies e.g., finding optimal quality parameters or assessing all combinations of allocation.

For the second kind of building block, automatic optimization needs to be considered. Automating optimization has the main benefits of performing exhaustive exploration of complex architectures in a reasonable time. Figure 7 shows one proposal for model-based optimization support.

Figure 6: Possible tool architecture for architecture evaluation
In addition to search space modeling, this tool chain requires developing ways to specify objectives of optimization and constraints on objectives. A building block should be able to import all Pareto Optimal solutions back in the EAST-ADL model. Then, the user should be able to select among those designs based on the values of objectives functions and to access to the corresponding architectures within Papyrus.

Following the previous ideas, the following sections describe the tool infrastructure selected for WT 4.3.

### 4.2.1 Preconditions

- Any information required during the transformation phase as part of the pre- and post-processing of the model. In practice, this means that knowledge of the model formats of any analysis tools must be taken into account in order to be capable of a valid transformation to (and from) that format. For example, an EAST-ADL error model may need to be exported according to the format used by HiP-HOPS analysis tool for a safety analysis, and the output format needs to be understood in order for the results to be integrated back into the EAST-ADL model in Papyrus. This may include any tool-specific options or commands and thus may require some form of graphical user interface as part of any plugin/export mechanism.

- There is likely also a requirement for the information produced by the optimization – namely, a set of possible optimal designs – to be handled properly. In practice, this will likely involve some form of interface that allows the user to select one of the optimums from within Papyrus and allow the EAST-ADL model to be reconfigured accordingly. For example, the results of a multi-objective optimization may be a set of Pareto optimal designs, meaning the designs are all optimums representing different trade-offs between the various objectives. See Technological Requirements below for more on this.

In addition to the information requirements described above, there are also a number of technological aspects. Again, this depends to a large extent on the analysis and optimization tools chosen.
4.2.2 Technological Requirements

- **Environment:** Eclipse-based support for Papyrus.

- **Model validator:** Wizards to perform transformations (including model validation and enabling/disabling constraints and rules) based on ATL model transformation language.

- **Design space handler:** Cheat-sheets, process black board. If variability mechanisms are chosen as a means to describe the design space, then this may include any variability plugins or other external tools.

- **Export of model:** Plugins will be required to export EAST-ADL models to analysis tools (such as SymTA/S, HiP-HOPS) or optimization tools (e.g. HiP-HOPS, Matlab etc). This may include model validation mentioned above and may involve model transformation.

- **Import of results:** The user should be able to select one of a number of optimal models/model configurations produced by the optimization tools, if more than one is generated (which is likely in the case of multi-objective optimization). These could be displayed according to a graph (e.g. the Pareto Front) and the relevant evaluation attributes (e.g. cost vs safety, or performance metrics) displayable—perhaps by tooltip or pop-up window—in order to inform the user's choice of optimal design. Selecting that option would then reconfigure the EAST-ADL model in Papyrus to reflect the optimum chosen.

  - **Optimization parameters:** Any optimization tools will likely require their own set of parameters to define the scope of the optimization process, e.g. how many alternatives to generate, what constitutes an optimum, any bounds or constraints on the optimization objectives etc. These parameters may need to be entered by means of some form of interface when the optimization process is initiated.

More detailed technical and information requirements would be needed to support any development of the optimization process as part of EAST-ADL and Papyrus, but as mentioned at the start of this section, the intention here is not to prescribe how or what functionality is to be implemented by which plugins and tools.

4.3 Available plugin description

Currently the set of plugins available for analysis-based modelling are: safety-analysis is offered by a bridge to the HiP-HOPs engine, timing analysis is offered by a bridge to the MAST schedulability analysis tool—although this relies on a MARTE description of timing attached to an EAST-ADL model. Variability and feature modelling are supported by a dedicated tool. A description and reference is available in a separate document D4.3.1.

Thus, tool pieces are there but the general framework is at this moment too crude to support the scenarios described in this document. Based on this work we believe a subsequent project, more focused on this challenge, should achieve to upgrade the tool support to another level.

However despite these current limitations we wanted to conduct experiments with tools to make a proof of concept and serve as a basis for future works. This is the subject of the next chapter.
4.4 HiP-HOPS Plugin

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 on multiple levels of abstraction. The concepts of the error analysis domain are represented by the safety analysis tool HiP-HOPS.

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 requiring to invoke the analysis after each change in the system design.

4.4.1 Technology

In this section we introduce the technology we depend on when integrating tool-based safety analysis into an automotive model-based development process.

4.4.1.1 Model Transformations

Model transformations play a key role in model-based software development. Model transformations describe the relationship between models, more specifically the mapping of information from one model to another one. These model transformation descriptions are interpreted by a model transformation engine. The model transformation engine produces the output model based on the transformation description and information from the input model. A model transformation involves two models: a source model and a target model, where source and target model can have the same or different metamodels. Model transformations use concepts of the metamodels in their descriptions. Thus they are general enough to describe the mapping for any model specified with the same metamodel. Model transformations are also used in generative programming [13], where they are called generators.

In the following we introduce a classification scheme for model transformations.

We can distinguish model transformations with respect to the creation of the target: Model-to-model transformations directly create elements of the target model. Each element in the source model maps to a specific element in the target model. Model-to-text transformations on the other hand create arbitrary, unstructured text. Each source element maps to an arbitrary fragment of text. This kind of transformation is also called model-to-code transformation or code transformation.

Model transformations can change the amount of detail presented in the model. They either introduce new details, reduce the amount of detail or leave it unchanged: Refinement transformations (vertical transformations) produce the target model by adding details to the source model. A change in the metamodel might be necessary for this step. This kind of transformation is the most common form. Abstraction transformations (vertical transformations) produce the target model by reducing the amount of detail. Translation transformations (horizontal transformations) produce the target model by expressing the same information found in the source model in a different language. The degree of detail stays the same. Translations are also called horizontal transformations.
We can differentiate model transformations according to the metamodels used in the transformation. *Endogenous* transformations map between the same metamodel. *Exogenous* transformations map between different metamodels.

Model transformations can be written using different languages. We can classify transformations according to the type of model transformation languages they use. *Declarative transformation languages* describe preconditions of the transformation and the according change with a postcondition. *Graph transformations* are described in this way, where a left-hand-side is the precondition and the right-hand-side is the postcondition. *Operational transformation languages* describe the transformation as a sequence of actions.

### 4.4.1.1.1 Model Transformation Engine openArchitectureWare (OAW)

OAW integrates a number of tools for model transformations into a coherent framework [15]. Among other tools, the OAW provides a workflow specification language and the transformation language Xpand. The workflow language is used to control the transformation process and to specify the sequence of transformations between the different models. The Xpand transformation language is a template-based, imperative language for model-to-text transformations. OAW is distributed as a plugin of the Eclipse platform and is able to process models that are conform to the EMF (Eclipse Modeling Framework).

### 4.4.1.1.2 Model Transformation Engine ATL

The ATLAS Transformation Language (ATL) is a hybrid model transformation language [16]. It includes both declarative and imperative constructs and supports both programming styles. However, the preferred style is declarative, which allows a cleaner and simpler implementation for simple mappings. However, imperative constructs are provided so that some mappings that are too complex to be handled declaratively can still be specified. An ATL transformation program is composed of rules that describe how to create and initialize the elements of the target models. The language is specified both as a metamodel and as a textual concrete syntax.

ATL is integrated in the Eclipse development environment and can handle models based on EMF. In this project we chose ATL for its ability to process UML models which are annotated with the UML extension mechanism for profiles, so we are able to process models that are conform to the EAST-ADL profile.

### 4.4.2 Tool Integration

To establish the link between EAST-ADL system modeling tools and safety analysis tools, they have to be integrated. In his seminal work, Wasserman identifies five different aspects of tool integration [18]:

- control integration: tools can interoperate
- data integration: tools can exchange data
- presentation integration: tools have a unified GUI
- platform integration: a common platform provides services as a basis for integration
- process integration: the SW development processes can be integrated

In the following we evaluate how these five aspects can be realized for the integration of safety analysis into model-based development.

Process integration cannot be done by software itself, but depends on personal preferences, company culture and development organization. Automation of safety analysis has several advantages: It makes safety analysis easy, it is readily available and allows the engineers to obtain a thorough and quick analysis of their design. This fast feedback based on analysis results allows
engineers to perform more micro iterations in the development process, where each iteration refines and improves the previously built model. The safety analysis is integrated in the development and more specifically in the safety analysis process. This process is aligned to upcoming ISO-CD-26262 standard and described in an EPF (Eclipse Process Framework) model for EAST-ADL.

The Eclipse platform provides a framework for platform and presentation integration. We use it by implementing our tool as an Eclipse plugin. We extend the graphical user interface of Eclipse by adding menus to invoke safety analysis for a given EAST-ADL model. This ensures seamless integration in the UML modeling environment and keeps the overhead for safety analysis experienced by the user as low as possible and thus allows for an iterative safety development process.

Control integration is realized by parameterizing and executing the model transformation engines and the safety analysis tool from within the developed plugin.

Data integration in this context is concerned with the transformation of modeling data. We transform from an EAST-ADL representation to a HiP-HOPS representation, while preserving the semantics. State of the art data integration for model-based development is supported by powerful model transformation engines and languages. Different transformation languages and engines are available, each of them solving a particular problem especially well. This is why the next section is dedicated to choosing the right model transformation language.

### 4.4.3 Translation from EAST-ADL to HiP-HOPS

Integrating safety analysis into the development of automotive embedded systems requires data integration. This can be achieved by translating concepts of the automotive domain to the error analysis domain. In the context of this work the automotive concepts are represented by the architecture description language EAST-ADL including its dependability model and the concepts of the error analysis domain are represented by the safety analysis tool HiP-HOPS. We need to expose the information of the EAST-ADL error models to HiP-HOPS in its native input format.

#### 4.4.3.1 Model Transformation

We automate the translation between EAST-ADL and HiP-HOPS using model transformations. Model transformation languages are domain specific languages for extracting information from models, for building and for manipulating models. Model transformation languages, paradigms and engines have been classified in [14] and [17]. Different model transformation languages have their strengths and weaknesses in solving particular types of tasks [14]. A challenge is choosing the right tool for the model transformation task at hand.

We have identified the following fundamental requirements for the model transformation engine used in our solution.

- Needs to be able to process UML models which have a UML profile applied, in our case this is the EAST-ADL profile
- Needs to produce text output, not a model
- Needs to be maintainable, the source code needs to be compact and reusable, since both EAST-ADL and HiP-HOPS evolve
- Needs to integrate as a plugin into the modeling environment

The model transformations we have evaluated do not fulfill all requirements at once. For instance we could not find an engine that allows us to produce text output and process the EAST-ADL profile. For this reason, we decompose the model transformation into two specialized transformations. Each of the two transformations fulfills the requirements partially, but the two transformations together fulfill all requirements.
4.4.3.2 Transformation Design

We leverage the advantages of different model transformation techniques by splitting the translation into two distinct phases and using an appropriate model transformation technique for each phase. Each phase has a distinct purpose and tackles a different concern.

![Diagram of Transformation Design]

Figure 1: Transformation Design

(1) Semantic Mapping Transformation: The first transformation step is a model-to-model transformation and is called M2M Trafo in figure 1. It transforms an EAST-ADL model that was created in the Papyrus UML modeling environment into an intermediate model. The structure of the intermediate model resembles the HiP-HOPS grammar, so it is close to the structure of the desired output. This stage performs the semantic mapping between the domains of EAST-ADL and that of HiP-HOPS. However, this stage is not concerned with the actual representation of the data.

(2) Representation Transformation: The second transformation step, called M2T Trafo in figure 1, takes the intermediate model and creates the input file for the HiP-HOPS program. This step is mainly concerned with the representation of the information according to the concrete syntax required by HiP-HOPS.

We will discuss both transformations in more detail in sections 4.4.5 and 4.4.6. There we use the scheme for classifying model transformation introduced in section 4.4.1.1 to determine the type of each of the two transformations and to choose a transformation engine fitting the properties of that particular transformation.

4.4.4 Involved Models and Metamodels

Three different models are involved in this model transformation. An EAST-ADL model, an intermediate HiP-HOPS model and the final HiP-HOPS file. The EAST-ADL model serves as the initial source model, and is conform to the EAST-ADL metamodel. The HIP-HOPS file is the final outcome of the transformation and conforms to the HiP-HOPS grammar. We discuss the metamodels separately in the following sections.

4.4.4.1 EAST-ADL Error Model

EAST-ADL models created in the Eclipse-based Papyrus UML tool have a metamodel that is a composition of several separate metamodels. This metamodel consists of the UML metamodel and the EAST-ADL profile definition. These artifacts are combined by the Eclipse framework to the EAST-ADL metamodel. However, this combined metamodel is not an autonomous entity or file. This complicates the model transformation and limits the choice of model transformation engines.

The EAST-ADL domain model contains concepts for modeling the anomalies of a system in a so called error model, which describes the failure semantics of a system by relating the occurrences of internal errors and the propagations of such errors. These error modeling constructs are separated from the constructs used for the nominal system definition, to clearly separate their...
different natures: error models are purely descriptive while nominal models are prescriptive and may be used for code generation.

Figure 2: Domain Model of the EAST-ADL Error Modeling Concepts

The domain model of the EAST-ADL error modeling concepts is illustrated in figure 2. In the following we introduce the core concepts. The ErrorModelType metaclass represents the container for maintaining the information relating to the anomalies of a system, function, software component, or hardware device. The ErrorModelPrototype metaclass describes an instance of an ErrorModelType. Even though these concepts are similar to the concepts for nominal behavior, the decomposition of the system into ErrorModelTypes is kept separate from the nominal decomposition into FunctionTypes. This makes it possible to have either totally aligned or separate topologies in error modeling than in the targeted nominal architecture, depending on the needs for error analysis. An ErrorPropagationLink describes how failures in one component can propagate to other components.

4.4.4.2 HiP-HOPS Ecore Metamodel

Due to the decomposition into two separate transformations we introduced an intermediate model which connects the two transformations (see figure 1). The intermediate model is conform to a HiP-HOPS Ecore metamodel that is aligned to the HiP-HOPS grammar. It is conform to the Ecore metamodel. The HiP-HOPS Ecore metamodel is depicted in figure 3.

At its core, the HiP-HOPS Ecore metamodel is a hierarchical decomposition into systems and components, where a system can contain components, which contain an implementation, which can contain another system. Thus hierarchical systems of any depths can be built recursively. Components and systems can be annotated with failure data, i.e. how failures propagate through the systems and where they originate from.
Figure 3: HiP-HOPS Ecore metamodel

4.4.5 Semantic Mapping Transformation

The purpose of the semantic mapping transformation is to map concepts from EAST-ADL to HiP-HOPS in a way that preserves the semantics of the original model, even though the structure of the model must be changed heavily. EAST-ADL models and HiP-HOPS models are structurally different. This can be demonstrated by the following example. EAST-ADL follows the concepts of declaring types first and referencing to the declaration from each point of use. In HiP-HOPS on the other hand, the declaration and usage of a type is coupled, types are declared at the same point as they are used. Thus the declarations have to be inlined into every point of usage, when
transforming from EAST-ADL to HiP-HOPS. Table 1 lists the detailed mapping between EAST-ADL concepts and HiP-HOPS concepts. In Figure 4 we show the part of this transformation that maps ErrorModelPrototypes of EAST-ADL to Components of the HiP-HOPS Ecore Metamodel.

According to the classification scheme introduced in section **Erreur ! Source du renvoi introuvable**, the representation transformation can be classified as an endogeneous, horizontal, model-to-text transformation. Model-to-text transformations are well suited for our semantic mapping transformation, because both input and output are models. Mapping patterns can be described by relational and declarative transformation languages in a concise manner. Our solution leads to relatively short source code for the solution. We selected the ATLAS Transformation Language (ATL), a language that allows a choice of relational and imperative constructs. It furthermore allows processing of models that have a profiled metamodel, i.e. a metamodel that consists of a metamodel and a profile description. In our case the EAST-ADL metamodel consists of the UML metamodel and the EAST-ADL profile.

Figure 4: Example: part of the semantic mapping transformation in ATL

<table>
<thead>
<tr>
<th>EAST-ADL Pattern (Source)</th>
<th>EAST-ADL Type</th>
<th>HiP-HOPS Pattern (Target)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorModelType</td>
<td>ErrorModelType</td>
<td>System</td>
</tr>
<tr>
<td>ErrorModelType.errorConnector</td>
<td>ErrorPropagationLink</td>
<td>System.Lines</td>
</tr>
<tr>
<td>ErrorModelType.parts</td>
<td>ErrorModelPrototype</td>
<td>System.Component</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorPort</td>
<td>ErrorPort</td>
<td>System.Component.Ports</td>
</tr>
<tr>
<td>ErrorModelPrototype</td>
<td>ErrorModelPrototype</td>
<td>System.Component.Implementation</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorBehaviorDescription.internalErrorEvent</td>
<td>ErrorEvent</td>
<td>System.Component. Implementation.FData. basicEvent</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorBehaviorDescription.failureLogic</td>
<td>String</td>
<td>System.Component. Implementation.FData.outputDeviation</td>
</tr>
</tbody>
</table>

Table 1: Semantic mapping between EAST-ADL and HiP-HOPS

4.4.6 Representation Transformation

The purpose of the representation transformation is the generation of a textual description based on the intermediate model. According to section **Erreur ! Source du renvoi introuvable**, the representation transformation can be classified as an endogeneous, horizontal, model-to-text transformation.

Textual representations can be generated particularly well with model-to-text transformation languages. We choose the Xpand language of OpenArchitectureWare. Xpand is a template-based model transformation language, which incorporates the output in the form of templates into the
control structure. Figure 5 shows a part of this transformation that creates a textual representation of the intermediate model that can serve as input to HiP-HOPS.

The intermediate model is designed to have structure which is aligned to HiP-HOPS. No structural changes are required in this transformation. The focus is on serializing the model as text. When serializing a graph structure to text, as done here, the choice of serialization strategy is important, as it dictates the order of the output. We explore the intermediate model using a depth first exploration strategy.

![Example: part of the representation transformation in OAW Xpand](image)

### 4.4.7 Benefits of the Chosen Decomposition

In this section we discuss the benefits of this solution.

- Our solution separates two different concerns of the transformation from EAST-ADL to HiP-HOPS: (1) the semantic mapping between the domains of EAST-ADL and that of HiP-HOPS and the (2) details of the concrete syntax of the HiP-HOPS input file.

- Each transformation is a separate, self-contained module, which can be developed, changed and tested independently. This decomposition into two separate transformations allows us to parallelize the work on the two transformations and reduce development time. It also allows the two transformations to evolve independently without affecting each other, e.g. a change in the HiP-HOPS grammar will only affect the representation transformation.

- As discussed in the section on data integration, different transformation engines have different strengths which can be played out for different concerns. The solution allows us to select an appropriate tool for each concern.

- Since we chose appropriate tools for each step, the resulting model transformation source code is very concise, resulting in a maintainable codebase.
5 Optimisation Example

Although it is not possible in the scope of ATESST2 to achieve a full technical implementation of a multi-objective optimisation tool capable of optimising EAST-ADL models with respect to multiple non-functional characteristics like safety, timing, and cost, it was deemed important to show that the concept was sound and that significant benefits could be obtained by such a tool. As a result, to demonstrate the concepts, a simple example model was developed during a meeting in York (the so-called ‘York Model’) that would be the basis of a multi-objective optimisation process involving both safety, cost, and timing analyses. Although the different analysis tools remain separate, the example was small enough that some exhaustive analysis of each enumerated design candidate could be performed, enabling that information to be used as an objective in the HiP-HOPS optimisation tool.

In this section, the example model, the optimisation process, and the results obtained will all be discussed.

5.1 York Model

The intention for the example model was to be able to combine many disparate elements of the EAST-ADL language and perform some kind of optimisation on them. In particular, it was deemed important to have a model that:

- Featured multiple perspectives, e.g. software and hardware, and allocation relationships between the different perspectives. This was to test the multi-perspective dependability analysis capabilities of HiP-HOPS and also to add a new dimension to the design space by allowing variable allocations, so that in one design candidate, function X may be allocated to hardware component Y, and in another design candidate, it would be allocated to component Z.

- Featured variability to help define the size and scope of the design space. Variability in this regard came in three forms: the ability to have different implementations of a component/function with different characteristics, the ability to have different hierarchical architectures, i.e. different subsystems/subcomponents, and as mentioned above, variable allocation relationships. These variability constructs would then define and control the scope of the optimisation.

- Featured timing information to allow a timing analysis to take place and enable optimisation with schedulability as one objective.

The result was a generic multi-perspective model involving sensors, processors/processing functions, and actuators that could be an example of a braking system, cruise control system, or many other automotive systems. It would take input from sensors, perform some calculations based on that input, and deliver some output instructions to an actuator as a result.

5.1.1 Initial Design

The initial design of the model is shown below. The top shows the software perspective, containing six software functions (including software functions describing the actuator and sensor functions and local device managers for these). The bottom shows the hardware perspective with four components – a sensor, an actuator, and two ECUs in series configuration. The software actuator function (“ActuatorFn”) delivers the system output.
In the software view, the system contains abstract sensor and actuator functions ("SensorFn" and "ActuatorFn") that represent the functions/behaviour of their respective hardware elements; they are also allocated to those hardware elements to make this link more explicit. It is necessary to include at least an abstract 'ActuatorFunction' to serve as the output function of the system, otherwise a failure of the H/W actuator component has no way to propagate to the output of the system in the S/W view. This is therefore a constraint resulting from the modelling paradigm that uses the software perspective as the primary perspective, the one that delivers system output, and the hardware perspective only as a source for fault propagations, not as a destination.

There are also two 'local device manager' (LDM) functions, one for the sensor and one for the actuator. These handle I/O for those devices, taking input and converting it into the appropriate output. These are software functions that are allocated to ECUs. However, the allocation for the LDMs is fixed, as the sensor is connected only to ECU1 (so the LDM for the sensor must be allocated there) and the actuator is connected only to ECU2 (so the actuator LDM must be allocated only to ECU2).

The main 'work' of the system is performed by the two software functions, Function1 and Function2. These are both more complex software functions that perform some calculation based on the sensor input (from the SensorLDM) and then send some appropriate commands to the actuator as a result (via the ActuatorLDM). The functions are in series, so the output of Function1 is directed to Function2. Both functions can be allocated to either ECU.

Each function can have one of three different implementations or variants, represented as different subsystems. The first implementation is a single function that offers no redundancy and performs all the work itself; the second implementation uses two functions, a standby and a primary, so that if the primary fails, the standby is able to take over operation, adding some redundancy to the function. A monitor function serves to activate the standby in the case of primary failure. The third implementation/variant is a triple redundant subsystem with three (different) functions all connected to a voter; the system can continue to operate if one of the three functions fails, and may also continue to function (but potentially with increased likelihood of value failures) if two of the three functions fail. However, we ignore value failures to keep things simple.
Furthermore, the intention is that each subfunction could be allocated to either ECU, so for example the Primary function of the Primary/Standby could be allocated to ECU1 and the Standby to ECU2.

5.1.1.2 Hardware Perspective

In the hardware perspective, there are four components. The first is the Sensor, which serves as the input for the system. The Sensor is then connected to the first of two ECUs, upon which the various software functions can run. The first ECU is connected in series to a second ECU, with the intention of providing some measure of redundancy. Then the second ECU is connected to the Actuator.

Although SensorFunction and ActuatorFunction are allocated to Sensor and Actuator respectively, these are not processing components and their S/W functions are merely abstractions to allow correct failure propagation to take place. The ECUs provide the computing capabilities for the system and can execute any of the other S/W functions (i.e., LDMs, Function1, Function2), although the LDMs have fixed allocations.

5.1.1.3 Fault Propagation modelling

There are two main channels for fault propagation in the model. The first is through the H/W, so a failure of the Sensor will propagate to ECU1, any failure of ECU1 (i.e. internal or a failure propagated from the Sensor) will in turn propagate to ECU2, and again any failure from ECU2 will propagate to the Actuator. All of these have a possible internal failure mode that will cause an omission of output. Furthermore, a failure of any of these components will lead to the failure (omission) of any S/W functions allocated to them; in the case of the Actuator, this means that any propagated H/W failure will in turn propagate to the ActuatorFunction and thus cause system failure.

The hardware failure logic – in HiP-HOPS style failure expression format – is shown below:

• Omission-Sensor.Out = SensorFailure
• Omission-Sensor.Allocated = SensorFailure
• Omission-ECU1.Out = Omission-ECU1.In OR ECU1Failure
• Omission-ECU1.Allocated = Omission-ECU1.In OR ECU1Failure
• Omission-ECU2.Out = Omission-ECU2.In OR ECU2Failure
• Omission-ECU2.Allocated = Omission-ECU2.In OR ECU2Failure
• Omission-Actuator.Allocated=Omission-Actuator.In OR ActuatorFailure

Thus for example an omission of the output from the sensor is caused only by an internal failure of the sensor ("SensorFailure"), and similarly a fault propagated along the allocation link (allocation propagations are indicated by the "Allocated" suffix) is also caused by a SensorFailure. For the ECUs and Actuator, the logic is more complex, as either type of output failure (propagation via H/W and via allocation) can be caused by either the same class of failure received at the input or by an internal failure mode.

In the software perspective, we can either assume that no internal failures occur or alternatively that internal failures are possible, but with fairly abstract failure rates (e.g. determined solely by ASIL). As with the H/W, failures propagate from SensorFunction through SensorLDM through Function1 & Function2 to ActuatorLDM and eventually ActuatorFunction (the system output). In addition, failures propagated from H/W will also enter this channel of propagation, e.g. a failure of the H/W Sensor will cause an omission of output from the S/W SensorFunction.
The S/W failure logic is shown below:

- Omission-SensorFunction.Out = FromAllocation(Omission) OR SensorFunctionFailure
- Omission-SensorLDM.Out = FromAllocation(Omission) OR SensorLDMFailure OR Omission-SensorLDM.In
- Omission-ActuatorLDM.Out = FromAllocation(Omission) OR ActuatorLDMFailure OR Omission-ActuatorLDM.In

The “FromAllocation(<failure class>)” construct indicates faults propagated via the allocation links to H/W components. Thus an omission failure of the Actuator Function’s output is caused by either an internal failure mode, an omission of input to the Actuator Function, or a failure propagated from H/W.

The failure logic for the single Function implementation is simply:

- Omission-Function1/2.Out = FromAllocation(Omission) OR SingleFailure OR Omission-Function1/2.In

In other words, the single software Function can fail because of internal fault, propagated H/W failure, or omission of input in the S/W layer.

For the primary/standby, it is:

- Omission-Primary.Out = Omission-Primary.In OR FromAllocation(Omission) OR PrimaryFailure

Thus an omission of the whole Function is caused by an omission of both primary and standby subfunctions, which in turn can be caused by combinations of failures propagated from H/W or S/W or internal failure modes.

For the triple voter, it is:

- Omission-Function1/2.Out = Omission-Voter.Out
- Omission-Sub1/2/3 = Omission-Sub1/2/3.In OR Sub1/2/3Failure OR FromAllocation(Omission)

Thus failure of the triple voter Functions requires an omission of two out of three inputs, or an internal failure of the voter, or some failure propagated from H/W.
5.1.1.4 Problems with the initial model

During preparatory modelling for failure analysis & optimisation, it was discovered that this model is not ideal for optimisation for a number of reasons.

- One problem is related to the nature of the H/W failure propagation. Any failure of any H/W component will always be propagated to the system output, meaning that any H/W component failure is a single point of failure for the whole system, regardless of the allocation strategy or implementation/variants chosen. This is due primarily to the way the ECUs are connected in series rather than in parallel.

- The two LDM functions are also single points of failure for the entire system and prevent any allocation/implementation strategy from providing fault tolerance. Since each LDM has a fixed allocation (SensorLDM→ECU1, ActuatorLDM→ECU2), a failure of either ECU will cause a failure of at least one LDM, thus leading to a failure of the system regardless of how the Functions are allocated.

- The alternative implementations/variants for the two Functions are very dependent on the allocation strategy. For example, the monitor and the primary must be on separate ECUs otherwise the monitor fails to activate the standby. Similarly, with the triple voter, the voter itself is a single point of failure and prevents the triple functions from being fault tolerant. Since so few possible allocation combinations provide any fault tolerance, there is little difference between using the single and more fault tolerant implementations, which will affect the optimisation.

- If S/W failures are not included, then only H/W failures have any effect, but since each H/W failure is a single point of failure unaffected by any variability in the model, there is no scope for optimisation (at least on the basis of reliability/safety; there may still be scope for cost vs timing, for example).

- There are too many possible combinations – around 600 – for the different variants to be enumerated and individually subjected to timing analysis.

To help correct these problems and hopefully produce a more workable example, an evolved version of the model was developed.

5.1.2 Improved York Model

The new design is similar to the old one, but with four main changes. Firstly, the ECUs are now connected in parallel rather than in series, and bi-directional communication is possible to ensure the S/W functions can communicate in any allocation combination. This should mean that the ECUs are no longer always single points of failure in the H/W perspective, i.e. if the functions are allocated correctly, the system should survive the failure of an ECU.

The second change is the duplication of both LDMs. This is necessary to ensure that the LDMs are not single points of failure in the software perspective. Each LDM still has a fixed allocation, but now e.g. one SensorLDM is allocated to ECU1 and the other is allocated to ECU2. This also means that the software functions can correctly receive/send data whichever ECU they are allocated to.

The third change is a simplification of the alternative implementations/variants for the two main S/W functions. The triple voter has been removed, partly to reduce the number of combinations and partly to avoid the problems with the voter itself being a single point of failure. The primary/standby implementation has also been simplified by combining the monitor with the standby; in practice this has little effect on the analysis as the failure data remains mostly the same, but it also helps to reduce the number of combinations.

Partly to compensate for the loss of so many combinations, and partly to try to add interesting new characteristics to the model, we have also added an alternative substitute for each ECU. This would simply be another ECU with different failure/timing/cost attributes; alternatively it could be considered to be the same ECU but with a different operating system/firmware.
The new model is shown below.

![Improved York Model](image)

**Figure 10: Improved York Model**

![New implementations for the Software Functions](image)

**Figure 11: New implementations for the Software Functions**

The failure data for this system remains much the same, except that the Standby is now triggered directly by an omission of output from the Primary. The only other changes are necessitated by the parallelization: for example, Function1 can take its input from either SensorLDM, so both SensorLDMs must fail to cause an omission of input to Function1. This is true regardless of which ECU Function1 is allocated to; it is assumed that the Function is able to obtain its input from the LDM on the other ECU if the LDM on the same ECU fails. Similarly, ActuatorFunction takes its input from both ActuatorLDMs (both must fail to cause an omission of input) and the H/W Actuator only fails if both ECUs fail. This also helps enable fault tolerance throughout the system, without which the various allocation/implementation strategies would have little effect.

### 5.2 Optimisation Process

The optimisation process involved three main steps. The first step was to enumerate the different possibilities to be able to link the different analyses together (since timing analysis was to be run for each possible configuration) and to provide data for the optimisation, e.g. cost and failure rates etc. The second step was to run the timing analyses on each possibility (or set of possibilities, in cases where the configuration made no difference). The results of the analyses could then be embedded in the model as an additional optimisation objective. The final stage was to run the full multi-objective optimisation using HiP-HOPS to see what results it can provide.
5.2.1 Enumeration & Provision of Optimisation Data

For the improved York Model, there are 144 different possible configurations for the system. These covered the full range of possibilities resulting from different allocation strategies, the use of different component implementations (each ECU had two possible implementations), and the use of different fault-tolerance architectures in the software functions:

- Functions 1 & 2 could each be implemented as either a single function or as a primary/standby architecture.
- Each Function / Subfunction could be allocated to either ECU. Thus the single function implementation could be allocated to ECU1 or ECU2, and in the primary/standby version, the primary and standby could each be allocated to ECU1 or ECU2 (thus providing four options – Primary → ECU1 and Standby → ECU1, Primary → ECU1 and Standby → ECU2, Primary → ECU2 and Standby → ECU1, or both to ECU2).
- Each ECU had two possible implementations with different non-functional characteristics. Version 1 was cheaper but had less processing power, while Version 2 was more expensive but had greater processing power.

The different enumerations were all listed in an associated Excel file (see optimisation-global.xls), which shows all variants in the example model enumerated, together with some simple pseudo-analysis results. There are 144 variants, broken down as follows:

- 16 variants for the single function implementation for both Function1 and Function2;
- 64 variants for a combination of one single function and one primary/standby;
- 64 variants where both Functions are using primary/standby implementations.

The columns provide the following information:

1. This is a simple numeric ID.
2. Indicates whether Function1 is using a single or primary/standby implementation.
3. Indicates whether Function2 is using a single or primary/standby implementation.
4. Indicates whether Function1’s primary (or single function, if that is the case) is allocated to either ECU1 or ECU2.
5. Indicates whether Function2’s primary (or single function, if that is the case) is allocated to either ECU1 or ECU2.
6. Indicates whether Function1’s standby is allocated to ECU1 or ECU2 (or if Function1 is using a single function, this is N/A).
7. Indicates whether Function2’s standby is allocated to ECU1 or ECU2 (or if Function2 is using a single function, this is N/A).
8. Shows the version/variant ECU1 is using.
9. Shows the version/variant ECU2 is using.
10. This is an attempt at some simplistic measure of how hard ECU1 is working, based solely on a calculation of how many functions are allocated to it (excluding LDMs). The first implementation of ECU1 (“Version1”) is cheaper but assumed to have a max of 4 functions, while the second implementation (“Version2”) is assumed to have a max of 5 functions, but costs more. The ‘load’ is then expressed as a percentage of maximum.
11. As above, but for ECU2.
12. This is an average of the 'load' for ECU1 and ECU2, i.e. (ECU1load + ECU2load) / 2.
13. This is a simple cost calculation. Each function/subfunction for Function1 & 2 is assumed to cost 10 points, so the first implementation (single function) costs 10 while the second (primary/standby) costs 20. The ECUs also have a cost value: Version1 costs 10, while Version2 costs 15. These costs are then summed.
14. This shows a simplistic view of the failure behaviour for the ECUs and Functions. It ignores the failures of the other H/W components and the other S/W functions (LDMs, SensorFunction &
ActuatorFunction) as those are the same every time. This can also be used as a crude qualitative indicator of reliability, i.e. the more single points of failure there are, the less reliable the system is, whereas the more conjunctions of failures there are, the more robust the system is. Thus the most reliable combination is one with three conjunctions and no single points of failure, and the least reliable is one with four single points of failure (i.e. ECU1, ECU2, Primary1, Primary2).

15. In addition, the final column shows the encoding string that HiP-HOPS uses to uniquely identify each configuration.

Cost information was relatively abstract: each software function had a cost of 10. The combined primary/standby architecture therefore cost 20 (as it had two functions), while the single function architecture cost 10. All other functions had a constant cost of 10. Each hardware component also had a cost of 10, except for the ECUs: Version 1 cost 10 as normal, Version 2 cost 15. Thus choosing to use Version 2 for both ECUs was equivalent in cost to having a third ECU.

The cost for the whole system therefore ranged from 120 (Version 1 for both ECUs, single functions for both Function1 and Function2) to 150 (Version 2 for both ECUs, primary/standby for both Function1 and Function2).

All software functions were given a constant failure rate of 1e-6. All hardware components had a constant failure rate of 1e-3. System unavailability therefore ranged from 0.002001 to 0.003996.

5.2.2 Timing Analysis

The next step was to run timing analyses for the 144 different system configurations. The results could then be added to the model and used as an extra objective by the full optimisation carried out by HiP-HOPS. Timing analysis is provided via the CEA LIST bridge from MARTE models to MAST schedulability analysis (see refs). The current version of the bridge and of EAST-ADL to MARTE mapping is not capable of performing automatic analysis on a set of models. Thus the models with variants are manually edited, then converted to MAST format by the bridge. Afterwards, analysis is performed on the transformed model.

There are significantly fewer configurations for timing than the 144 different configurations described in the Excel file mentioned above: only 64 configurations need to be tested. This is because in many configurations, only cost or safety/failure information changes, not the timing characteristics. For instance, when allocating all functions (primary and/or standby) on the same ECU, say ECU1, the second ECU, say ECU2 is not used for the end-to-end flow, hence the situation is the same if we allocate everything on ECU2. Because of symmetry, situations where P1 (F1.Primary) is allocated on ECU2 are identical to an allocation to ECU1, with other elements (standbys and other primary, speeds) permuted.

Results of the timing analyses were stored in a separate Excel sheet (“optimization-timing”) and index mapping is provided to the original Excel file (by means of the #global index field; for example, timing configuration #1 applies to global configurations #1, #2, #13, and #15).

Version 1 implementations for the ECUs are assumed to be identical with a speed factor of 1.0, whereas the Version 2 implementations of the ECUs have a speed factor of 4.0. This was chosen to show sufficient change in the results.

As far as timing is concerned, the example is modelled as follows:

- A nominal end-to-end flow is provided. It accounts for the execution flow from sensor to actuator. This is depicted as an activity diagram (see Figure 12 below). Each step is either a communication step (attached to an OS channel) or an execution step (attached to an OS task). The assumption is made here that all steps have a distinct task or channel available on the OS level. Communication steps occur when two functions allocated on different
ECUs need to exchange data: this is the case between Sensor and SensorLDM, F1 and F2 when these are on different ECUs. When F1 and F2 are allocated on the same ECU, the communication step function1.send() is removed. An end-to-end flow is fed with a triggering event, in our case a PeriodicEvent of period 200ms. Each step is given an execution time, in the form of a (min,max,value) value. Function1 is (10,20,ms), Function2 (20,30,ms), their replicas have the same properties. All other steps are (5,10,ms).

Figure 12: Nominal End-To-End flow

- Whenever a primary/standby scheme is used, we introduce additional end-to-end flows, for instance for Function1 see the figure below. Such additional end-to-end flows show the communication step between the primary function and its replica and the execution of the replica itself. Period for this end-to-end flow is set to 100ms (after several tests). The communication step may be removed when the primary and standby are on the same ECUs.

Figure 13: Reconfiguration end-to-end flow

- When we replicate not only F1 but also F2, then 3 end-to-end flows are considered by the timing analysis (nominal, reconfiguration1 and reconfiguration2). The tool tries to evaluate whether the requirements for all flows can be met, taking into account the fact that end-to-end flows concurrently load the ECUs.

- As said above, steps are allocated to tasks or channels. These are modelled as properties of an OS Layer on top of the HDA layer (see Figure 14). More precisely the analysis is performed on an analysis context which summarizes the system features: an FDA on top of a Platform, the Platform being an OS layer on top of an HDA. The HDA features all the hardware elements. Each of this hardware element is provided with a speed factor. The OS
layer regroups all tasks and channels that are needed for the various configurations. Each one of them is allocated to a hardware element and features a priority (all have priority 10 in our example). The steps of the end-to-end flows are allocated to tasks and channels. Hence from the end-to-end flows it is possible to follow each step across the architecture.

![CombinedOptimization](image)

**Figure 14: Analysis Context**

The timing analysis methods suitable for the example are limited because of the distributed nature of the communication. We chose the offset-based optimized analysis provided by MAST (see references). This technique takes into account the presence of multiple tasks involved in different end-to-end flows (called transactions in MAST) on the same ECU by adding offsets (extra delay time) to the worst case execution time (WCET). The optimized version takes into account the order of the tasks, that is the WCET of steps are increased by the WCET of all concurrent steps except for those which are known to occur before.

Example: f1.primary, f1.standby, f2.primary are all on ECU1. We consider f1.primary.compute()→f2.primary.compute() in the nominal flow and f1.standby.compute() in the reconfiguration end-to-end flow.

- For f1.primary.compute() the WCET is increased by the WCET of f1.standby.compute().
- For f2.primary.compute() the WCET is increased by the WCET of f1.standby.compute().
- For f1.standby.compute() the WCET is increased by the WCET of both f1.primary.compute() and f2.primary.compute().

Results are given as slacks and response times. Response times are both best case execution time (BCET) and worst case execution time (WCET) in ms for all end-to-end flows involved in the configuration. Slacks are percentages which show how much more the system can be loaded compared to the current timing requirements: for instance 100% would mean you can double the timing requirement and the system should still be schedulable. Slacks provided by MAST can be very high and are indeed in some configurations of our example; this comes from the fact that we chose large periods (200ms for the nominal and 100ms for each of the reconfiguration) so that all configurations would be schedulable. The worst case is of course when everything is allocated on
the same ECU and a “slow” version is chosen, in which case slack drops to 16.41%, whereas when using fast ECUs you can reach 1000%.

This is a limit of our approach here. The combined analysis will act as a first filter on the configurations. One would later need to further enhance the analysis on some of the chosen configurations.

### 5.3 Results of the HiP-HOPS Multi-objective Optimisation

For the full optimisation, three objectives were used:

- Cost (a simple sum of the costs of each component & function)
- Unavailability (calculated from the failure rates of individual functions & components by HiP-HOPS)
- System slack (calculated from the results of the timing analyses above)

The goal was to obtain the Pareto frontier, i.e. the set of solutions (design candidates) which have superior attributes that are not dominated by another other solutions. As explained earlier, a dominated solution is one for which some other solution has better characteristics for each of the objectives being evaluated. In the case of the York Model, there are 28 Pareto optimal solutions (out of the full 144).

<table>
<thead>
<tr>
<th>Cost</th>
<th>Unavailability</th>
<th>System Slack</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>0.002999</td>
<td>183.59</td>
<td>1</td>
</tr>
<tr>
<td>125</td>
<td>0.002999</td>
<td>1036.7</td>
<td>3</td>
</tr>
<tr>
<td>120</td>
<td>0.003996</td>
<td>396.88</td>
<td>5</td>
</tr>
<tr>
<td>130</td>
<td>0.003996</td>
<td>1889.8</td>
<td>8</td>
</tr>
<tr>
<td>120</td>
<td>0.003996</td>
<td>396.88</td>
<td>9</td>
</tr>
<tr>
<td>130</td>
<td>0.003996</td>
<td>1889.8</td>
<td>12</td>
</tr>
<tr>
<td>120</td>
<td>0.002999</td>
<td>183.59</td>
<td>13</td>
</tr>
<tr>
<td>125</td>
<td>0.002999</td>
<td>1036.7</td>
<td>14</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>183.59</td>
<td>21</td>
</tr>
<tr>
<td>140</td>
<td>0.002999</td>
<td>894.53</td>
<td>24</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>183.59</td>
<td>41</td>
</tr>
<tr>
<td>140</td>
<td>0.002999</td>
<td>894.53</td>
<td>44</td>
</tr>
<tr>
<td>135</td>
<td>0.002999</td>
<td>623.44</td>
<td>51</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>183.59</td>
<td>53</td>
</tr>
<tr>
<td>140</td>
<td>0.002999</td>
<td>894.53</td>
<td>56</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>183.59</td>
<td>73</td>
</tr>
<tr>
<td>140</td>
<td>0.002999</td>
<td>894.53</td>
<td>76</td>
</tr>
<tr>
<td>135</td>
<td>0.002999</td>
<td>623.44</td>
<td>78</td>
</tr>
<tr>
<td>150</td>
<td>0.002998</td>
<td>695.31</td>
<td>88</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>183.59</td>
<td>94</td>
</tr>
<tr>
<td>150</td>
<td>0.002001</td>
<td>396.88</td>
<td>96</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>121.09</td>
<td>105</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>467.97</td>
<td>110</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>467.97</td>
<td>115</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>121.09</td>
<td>117</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>183.59</td>
<td>131</td>
</tr>
<tr>
<td>150</td>
<td>0.002001</td>
<td>396.88</td>
<td>132</td>
</tr>
<tr>
<td>150</td>
<td>0.002998</td>
<td>695.31</td>
<td>140</td>
</tr>
</tbody>
</table>
HiP-HOPS found the complete optimal Pareto frontier after 62 generations, generating 10 new solutions each generation. An item of note with this problem is that it takes more than the 144 evaluations required to process every absolute combination. However, more than 50% of the solutions have identical objective values for cost, unavailability, and system slack. This search criteria focuses on gaps in the frontier and strictly speaking there are no gaps when looking at solutions with identical results. These solutions are still permitted as long as the configuration is not identical as there may be features of a particular solution that are not modelled in the objective function but can nevertheless differentiate the solutions in the eyes of the designer.

The 28 solutions are listed below:

Each of these solutions has an ‘encoding’ that uniquely identifies it. The encoding for each of the solutions in the results can be viewed graphically, e.g. the encoding for solution #1 is shown below. This encoding can also be used to ‘configure’ the original model, by setting the variability, thus potentially enabling the results to be re-imported back into the original modelling environment.

Figure 15: Encoding Visualisation for solution #1

From the results, it is also possible to generate a number of 3D graphs, showing the Pareto front in each dimension (one for each of the objectives). For example:
Figure 16: Graph for the System Slack optimization

The optimisation was also run with different timing analysis results as the third optimisation objective. For example, using End-To-End R2 Worst Case Execution Time (EndToEndR2WCET):

<table>
<thead>
<tr>
<th>Cost</th>
<th>Unavailability</th>
<th>EndToEndR2WCET</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>0.002999</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>120</td>
<td>0.002999</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>73</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>50</td>
<td>85</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>20</td>
<td>86</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>70</td>
<td>93</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>32.5</td>
<td>94</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>70</td>
<td>129</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>32.5</td>
<td>131</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>50</td>
<td>137</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>20</td>
<td>139</td>
</tr>
</tbody>
</table>
In this case, there were only 16 Pareto optimal solutions.

Using End-To-End R2 Slack as the third objective:

<table>
<thead>
<tr>
<th>Cost</th>
<th>Unavailability</th>
<th>EndToEndR2Slack</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>0.002999</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>120</td>
<td>0.002999</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>135</td>
<td>0.002999</td>
<td>1109.4</td>
<td>19</td>
</tr>
<tr>
<td>135</td>
<td>0.003995</td>
<td>1159.4</td>
<td>30</td>
</tr>
<tr>
<td>135</td>
<td>0.003995</td>
<td>1159.4</td>
<td>35</td>
</tr>
<tr>
<td>135</td>
<td>0.003995</td>
<td>231.25</td>
<td>41</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>1109.4</td>
<td>46</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>214.84</td>
<td>85</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>1109.4</td>
<td>91</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>794.53</td>
<td>94</td>
</tr>
<tr>
<td>145</td>
<td>0.003994</td>
<td>1159.4</td>
<td>102</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>181.25</td>
<td>105</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>181.25</td>
<td>117</td>
</tr>
<tr>
<td>145</td>
<td>0.003994</td>
<td>1159.4</td>
<td>123</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>794.53</td>
<td>131</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>1109.4</td>
<td>134</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>214.84</td>
<td>137</td>
</tr>
</tbody>
</table>
Figure 18: End-to-End Slack

Using End-To-End R2 Best Case Execution Time:

<table>
<thead>
<tr>
<th>Cost</th>
<th>Unavailability</th>
<th>EndToR2BCET</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>0.002999</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>120</td>
<td>0.002999</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>73</td>
</tr>
<tr>
<td>130</td>
<td>0.002999</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>20</td>
<td>81</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>5</td>
<td>83</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>20</td>
<td>89</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>5</td>
<td>91</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>25</td>
<td>93</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>10</td>
<td>94</td>
</tr>
<tr>
<td>140</td>
<td>0.002001</td>
<td>25</td>
<td>105</td>
</tr>
<tr>
<td>145</td>
<td>0.002001</td>
<td>10</td>
<td>107</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>20</td>
<td>109</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>5</td>
<td>110</td>
</tr>
<tr>
<td>140</td>
<td>0.002998</td>
<td>20</td>
<td>113</td>
</tr>
<tr>
<td>145</td>
<td>0.002998</td>
<td>5</td>
<td>115</td>
</tr>
</tbody>
</table>
It is clear that the optimisation process results in a set of candidate solutions, each representing a different optimal or near optimal trade-off among the parameters of the optimisation. Choosing a single solution out of this set for further development and implementation is essential and would require analysts to examine the results closely in order to assess which solution best meets requirements. To facilitate this process, HiP-HOPS allows the sorting of solutions in terms of their performance in individual objectives. By sorting results in terms of unavailability, for instance, it will be possible to exclude solutions that do not perform adequately well in terms of unavailability.

Figure 19: End-to-End Best Case Execution Time

Other Pareto solutions with the remaining timing analysis results as the objectives can be found in the Pareto.docx results file and the associated optimization-paretos.xls file, which also lists all of the solutions using the same IDs from the original enumeration. The latter file also includes a case where all timing analysis results were used as objectives, i.e. 10 timing objectives, cost, and unavailability, for a total of 12 objectives. In this case there were 107 Pareto optimal solutions, since with so many objectives, it is difficult for any solution to be dominated in every dimension.
Iterating this selection process for other objectives will make it possible to further reduce the set of candidate solutions and ultimately locate a single solution for further development.

In general, the above process is structured and very systematic and helps to arrive at sound decisions about the architecture which are informed by a set of essential analyses that can be considered in a single framework of evaluation and optimisation.
One argument for avoiding automation of all the optimization chain in WT 4.3 is that optimization is hard to deploy in all industrial cases. Many unsystematic and non-representable constraints and criteria would be involved in the process of architecting. Whatever the decision, it still may be possible to define a method that deals with architecture trade-offs from a process viewpoint and having EAST-ADL in the loop. Figure 20 illustrates a possible flow to these activities. The process aspects will be enriched in future versions of this document.

The process starts with the whole set of requirements and one specific architecture description (e.g., EAST-ADL analysis architecture or design architecture) and then determines the concrete subset of requirements and system components that are in the scope of the evaluation context. Three general activities are of interest:

(a) **Prioritize requirements.** This activity determines a subset of quality parameters that will point to one or likely more than one analysis frameworks (e.g., safety analysis, scheduling analysis). This subset represents the design constraints under evaluation and comprises independent and dependent parameters and their relations. Some of the independent parameters are directly determined by requirements and others may be quantified in this activity. For instance, in timing analysis, the model of requirements may establish response deadlines but event activation parameters may be fixed only just before evaluation. These parameters become input parameters for the evaluation. Other parameters are variable in the sense that they have not yet been assigned values. Variable parameters (i.e., trade-off parameters) become the focus of design decisions.

(b) **Evaluate architecture.** This activity helps proving whether the quality attributes of an architecture option meet system requirements and thereby includes the analytic machinery for describing and predicting specific quality attributes. It starts with determining the functionality scenarios and system components (including variants) participating in the evaluation context. These are then annotated, and new views are created if required (e.g., timing chain models for

---

**Figure 20: Possible workflow**

The process starts with the whole set of requirements and one specific architecture description (e.g., EAST-ADL analysis architecture or design architecture) and then determines the concrete subset of requirements and system components that are in the scope of the evaluation context. Three general activities are of interest:

(a) **Prioritize requirements.** This activity determines a subset of quality parameters that will point to one or likely more than one analysis frameworks (e.g., safety analysis, scheduling analysis). This subset represents the design constraints under evaluation and comprises independent and dependent parameters and their relations. Some of the independent parameters are directly determined by requirements and others may be quantified in this activity. For instance, in timing analysis, the model of requirements may establish response deadlines but event activation parameters may be fixed only just before evaluation. These parameters become input parameters for the evaluation. Other parameters are variable in the sense that they have not yet been assigned values. Variable parameters (i.e., trade-off parameters) become the focus of design decisions.

(b) **Evaluate architecture.** This activity helps proving whether the quality attributes of an architecture option meet system requirements and thereby includes the analytic machinery for describing and predicting specific quality attributes. It starts with determining the functionality scenarios and system components (including variants) participating in the evaluation context. These are then annotated, and new views are created if required (e.g., timing chain models for...
timing analysis), with analysis-specific information, upon existing architecture models. Analysis model fragments of interest are assembled in an evaluation context that uses a set of composition rules defined in the related analysis frameworks. Given the evaluation context, values for the unbound independent parameters are annotated (e.g., event activation parameters used in the evaluation context). When all the independent quality parameters have values, the value of the dependent parameters can be computed (mostly by using analysis tools). The dependent parameters are then compared with the quality requirements. Rules can be used to select between architecture variants or to adjust the value of independent parameters in a manner that will satisfy the concrete requirements. As an alternative architecture adjustment, optimization can be used.

(c) Optimize architecture. This activity helps finding the best solution from all feasible solutions (search space) regarding certain objectives and subject to certain constraints. Objectives and constraints are determined in activity (a). The search space is defined by the architecture variants and analysis models built in activity (b). When a set of valid solutions satisfy requirements, optimisation can then be used to determine the best architectural modifications that are consistent with the optimisation problem description.

These are initial hints for a general workflow. We believe that with tools and conducting experiments such a methodology can be further enhanced and backed up by dedicated tool support to help end-users in their tasks. To this end, it is rather interesting to consider the definition of helpers and wizards out of a process-description made for instance in the EPF framework provided by Eclipse (see WT5.1 results).
7 Result evaluation and conclusions

During M3.5 it was agreed to combine deliverables from 3.5, 4.3 and 5.2 tasks in one document updated in the course of the project, reflecting the following aspects: concepts and overall approach to analysis-driven modeling, tool framework specifications, method guidelines and tool experiments.

At the same time, it was also agreed to follow two branches: one dedicated to defining an “ideal” framework specifications and guidelines which would concentrate on the document, another focused on developing plugins to make investigations and tool experiments. The first steps of this experimental branch were to develop a plugin for safety-analysis connecting Papyrus to HipHops, and experimenting with a plugin on Timing analysis, developed in the frame of the EDONA project. Plugins developed within this set of tasks make up for the analysis platform of ATESST2 and are described in a separate wrapper document, entitled “The EAST-ADL analysis platform” (D4.3.1).

We found out that at the core of architecture evaluation is the definition of a canonical view for each of the quality to be assessed, i.e. timeliness, safety, etc. The latest revision of the EAST-ADL language (with the introduction of a language core and satellite extensions) goes in this direction.

Another important element is to be able to bound system model elements to a particular analysis or evaluation scope. As automotive applications become more complex, there is often a need to represent a system by multiple analysis models, corresponding to different application-platform allocations, abstraction levels, operational modes, or different quantitative values of non-functional parameters. Unfortunately, EAST-ADL does not have such evaluation scope construct. Starting with the evaluation context and its parameters (e.g., quality properties), a tool can follow the links of the model to extract the information that it needs to perform analysis.

We saw that tool pieces are there but the general framework is at this moment too crude to support the scenarios described in this document. Particularly central to this framework is the design space definition and exploration. We identified variability constructs and tools as promising candidates for this. However the feature modelling constructs were not primarily meant for this in the language. This needs to be further investigated so as not to misuse concepts.

The initial experiments presented in this document showed both the benefits that one can take from a joint analysis of a model with variability and also the feasibility of the approach.

All in all and to conclude with, the present document presented the results of what turned out to be a highly enjoyable collaborative work inside the project. It is rather relevant to see that the core concern of this work served as inputs for the definition of a sequel project, MAENAD – which was elected to be financed. We believe promising results and tools are ahead of us.
8 Annex A: Terms and Definitions

For the purposes of this specification, the following terms and definitions apply. These terms and definitions, including relevant notes and examples, are taken from other standards [3] and [4]. Some additional considerations or explanations for the ATESSST2 context are given in italics. Before the term [3] or [4], clause number is given in parentheses.

**architecting (1.1) [ISO/IEC42010]**

Architecting activities of conceiving, defining, describing, documenting, certifying proper implementation of, maintaining and improving an architecture throughout a system’s life cycle.

NOTE 1: Architecting takes place within the context of an organization (“person or a group of people and facilities with an arrangement of responsibilities, authorities and relationships”) or of a project (“endeavour with defined start and finish criteria undertaken to create a product or service in accordance with specified resources and requirements”).

**architecture (1.2) [ISO/IEC42010]**

Fundamental conception of a system in its environment embodied in elements, their relationships to each other and to the environment, and principles guiding system design and evolution.

**architecture decision (1.3) [ISO/IEC42010]**

Choice made from among possible options that addresses one or more architecture-related concerns.

**architecture description (1.4) [ISO/IEC42010]**

Collection of work products used to describe an architecture.

**architecture element (1.5) [ISO/IEC42010]**

Construct in an architecture description.

NOTE 1: Architecture elements include: constructs in views and models, entities in models, relations in models, identified stakeholders, identified concerns, views, viewpoints, and models. Each architecture viewpoint defines its own vocabulary of elements—not defined by this International Standard.

**architecture framework (1.6) [ISO/IEC42010]**

Conventions and common practices for architecture description established within a specific domain or stakeholder community.

EXAMPLE: Reference Model of Open Distributed Processing (RM-ODP) is an architecture framework.

**architecture model (1.7) [ISO/IEC42010]**

Work product from which architecture views are composed.

**architecture rationale (1.8) [ISO/IEC42010]**

Explanation or justification for an architecture decision.

**architecture view (1.9) [ISO/IEC42010]**

Work product representing a system from the perspective of architecture-related concerns.

**architecture viewpoint (1.10) [ISO/IEC42010]**

Work product establishing the conventions for the construction, interpretation and use of architecture views and associated architecture models.

**environment (of a system) (1.11) [ISO/IEC42010]**
Context determining the setting and circumstances of developmental, technological, business, operational, organizational, political, regulatory, social and any other influences upon a system

NOTE 1: A system of interest may have many distinct "operating environments", "development environments", etc. As defined herein, the term environment (of a system) is intended to cover all of these. The environment is the sum total of all influences on the system and influences of the system.

**life cycle (1.12) [ISO/IEC42010]**

Evolution of a system, product, service, project or other human-made entity from conception through retirement

**stakeholder (1.13) [ISO/IEC42010]**

Individual, team, organization, or classes thereof, having concerns with respect to a system

**required inputs (1.14) [IEEE1012]**

The set of items necessary to perform the minimum V&V tasks mandated within any life cycle activity.

**required outputs (1.15) [IEEE1012]**

The set of items produced as a result of performing the minimum V&V tasks mandated within any life cycle activity.

**reusable software product (1.16) [IEEE1012]**

A software product developed for one use but having other uses, or one developed specifically to be usable on multiple projects or in multiple roles on one project.

EXAMPLE: Examples include, but are not limited to, COTS software products, acquirer-furnished software products, software products in reuse libraries, and pre-existing developer software products. Each use may include all or part of the software product and may involve its modification. This term can be applied to any software product (for example, requirements, architectures), not just to software itself.

**risk (1.17) [IEEE1012]**

(A) The combination of the probability of occurrence and the consequences of a given future undesirable event. Risk can be associated with products and/or projects. (B) The combination of the probability of an abnormal event or failure and the consequence(s) of that event or failure to a system's components, operators, users, or environment.

**validation (1.18) [IEEE1012]**

(A) The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. (B) The process of providing evidence that the software [or system] and its associated products satisfy system requirements allocated to software at the end of each life cycle activity, solve the right problem (e.g., correctly model physical laws, implement business rules, use the proper system assumptions), and satisfy intended use and user needs.

**verification (1.19) [IEEE1012]**

(A) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (B) The process of providing objective evidence that the software [or system] and its associated products conform to requirements (e.g., for correctness, completeness, consistency, accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance); satisfy standards, practices, and conventions during life cycle processes; and successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (e.g., building the software correctly).
validation and verification (V&V) effort (1.20) [IEEE1012]
The work associated with performing the V&V processes, activities, and tasks. The following framework illustrates how V&V processes are subdivided into activities, which in turn have associated tasks.

cost-effectiveness analysis (1.21) [MultiCriteria1]
A term used to describe analysis which examines options which provide the same, or similar, benefits, and which assesses their relative merits by quantifying and comparing the costs of providing them.

criterion (1.22) [MultiCriteria1]
one of a number of measures against which options are assessed and compared in a multi-criteria analysis for the degree to which they achieve objectives.

decision analysis/decision theory (1.23) [MultiCriteria1]
Decision analysis and decision theory refer to any decision aiding approach that is based on expected utility theory and its later extension to decisions with multiple objectives. The theory assumes only that the decision maker wishes to be consistent in his or her preferences and decisions (see expected utility theory). Decision analysis is the applied technology that was developed from decision theory in the 1960s by Professor Howard Raiffa and his colleagues at Harvard University and by Professor Ronald Howard and his colleagues at Stanford University. The theory was extended in 1976 by Ralph Keeney and Howard Raiffa to include decisions with multiple objectives. This latter approach is commonly referred to as multi-criteria decision analysis (MCDA).

decision tree (1.24) [MultiCriteria1]
A diagram that shows the outcomes that may occur for a series of interdependent decisions sequenced over time. The actual outcome of each of the individual decisions at each stage is not known with certainty. Appropriate analysis of the tree allows the decision maker to develop, from the outset of the decision process, a contingent decision strategy. This indicates what is the best choice to make at each stage in the decision sequence, contingent upon the pattern of earlier decisions and outcomes.

evaluation (1.25) [MultiCriteria1]
The process of examining options and assessing their relative merits. In this document, it used to describe analysis after architecting. The terms ‘timeliness evaluation’ and ‘safety evaluation’ are often used to describe evaluation in those two areas. In general usage the word evaluation is often used to describe either before or after analysis. It is also often used to describe the process of deciding where the quality attributes of an architecture option with regard to a particular criterion met system requirements.

multi-criteria analysis (1.26) [MultiCriteria1]
Multi-criteria analysis can be used to describe any structured approach to determine overall preferences among alternative options, where the options accomplish several objectives.

objectives (1.27) [MultiCriteria1]
The purposes which a system life cycle wishes to achieve in areas of concern. Broad overall objectives, or ultimate objectives, are broken into lower level or intermediate objectives which are more concrete, and these may be further detailed as sub-objectives, immediate objectives, or criteria which are more operational.

variants (or options) (1.28) [MultiCriteria1]
Ways of achieving objectives. Variants might be architecture patterns, scheduling policies, fault tolerant schemes, subsystems, or anything else about which a decision is needed.
prescriptive decision models (1.29) [MultiCriteria1]

Prescriptive decision models are practical tools designed to help decision makers make more rational choices. They recognize the limited effectiveness of unaided, intuitive decision making processes, and seek to develop methods that will take decision makers closer to some rational ideal.

optimization (1.30) [wikipedia]

Optimization is the process of finding the best solution from all feasible solutions regarding certain objectives and subject to certain constraints. Maximizing performance and minimizing fuel consumption of a vehicle, and minimizing weight while maximizing the strength of a particular component are examples of objective optimization problems.

multi-objective optimization (multi-criteria or multi-attribute optimization) (1.31) [wikipedia]

The process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. Multi-objective optimization problems can be found in various fields: product and process design, finance, aircraft design, the oil and gas industry, automobile design, or wherever optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives.

dominance (1.32) [MultiCriteria1]

The situation where, in a pairwise comparison of two options, the first scores higher than the second on at least one criterion and no lower on any of the others. In this case the first option is said to dominate the second.

multi-objective optimization problem (1.33) [slides Hull]

An optimization problem is to find a solution x – i.e. an optimal design - (element of solution space $X$ of possible designs), which optimizes a vector of objective functions $f(x) = [f_1(x), f_2(x), f_3(x), ..., f_n(x)]$.

EXAMPLE: Search for Pareto Optimal (i.e. non-dominated) solutions. A solution $x_1$ dominates another solution $x_2$ if $x_1$ matches or exceeds $x_2$ in all objectives.

architecture analysis (1.34)

Architecture analysis is related to specific analysis reasoning frameworks used to predict different quality attributes. The terms “performance analysis” and “safety analysis” are commonly used to describe analysis of those quality attributes.

quality attributes (1.35)

Quality attributes are specific criteria that can be used to judge the operation of a system (e.g., performance, safety, availability, resource usage).
References


