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

<table>
<thead>
<tr>
<th>Report type</th>
<th>D3.1 Appendix A3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report name</td>
<td>EAST-ADL update suggestions for Safety Analysis support</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Version number</td>
<td>1.1.1</td>
</tr>
</tbody>
</table>
### Authors

<table>
<thead>
<tr>
<th>Editor</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin Walker</td>
<td><a href="mailto:Martin.Walker@hull.ac.uk">Martin.Walker@hull.ac.uk</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin Walker</td>
<td><a href="mailto:Martin.Walker@hull.ac.uk">Martin.Walker@hull.ac.uk</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>Matthias Biehl</td>
<td><a href="mailto:Biehl@md.kth.se">Biehl@md.kth.se</a></td>
</tr>
<tr>
<td>Schierano Walter</td>
<td><a href="mailto:walter.schierano@crf.it">walter.schierano@crf.it</a></td>
</tr>
<tr>
<td>Tagliabò Fulvio</td>
<td><a href="mailto:fulvio.tagliabo@crf.it">fulvio.tagliabo@crf.it</a></td>
</tr>
<tr>
<td>Torchiaro Sandra</td>
<td><a href="mailto:sandra.torchiaro@crf.it">sandra.torchiaro@crf.it</a></td>
</tr>
<tr>
<td>DeJiu Chen</td>
<td><a href="mailto:chen@md.kth.se">chen@md.kth.se</a></td>
</tr>
</tbody>
</table>

### Reviewers

<table>
<thead>
<tr>
<th>E-mail</th>
</tr>
</thead>
</table>

### The Consortium

- Volvo Technology Corporation (Sweden)
- VW/Carmeq (Germany)
- Centro Ricerche Fiat (Italy)
- Continental Automotive (Germany)
- Delphi/Mecel (Sweden)
- Mentor Graphics Hungary (Hungary)
- CEA LIST (France)
- Kungliga Tekniska Högskolan (Sweden)
- Technische Universität Berlin (Germany)
- University of Hull (UK)
### Revision chart and history log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>15/09/09</td>
<td>First version, synchronised with input for D3.1</td>
</tr>
<tr>
<td>0.2</td>
<td>30/09/09</td>
<td>Second version, incorporating further input on tools from KTH &amp; UOH</td>
</tr>
<tr>
<td>0.3</td>
<td>02/09/09</td>
<td>Third version, incorporating methodology section from CRF</td>
</tr>
<tr>
<td>0.4</td>
<td>11/09</td>
<td>Error modelling information added by KTH</td>
</tr>
<tr>
<td>1.0</td>
<td>20/11/09</td>
<td>Finalised</td>
</tr>
<tr>
<td>1.1</td>
<td>1/6/10</td>
<td>Final adaptations for use as appendix in D3.1</td>
</tr>
<tr>
<td>1.1.1</td>
<td>14/6/10</td>
<td>Updated again to resolve some inconsistencies with latest MM</td>
</tr>
</tbody>
</table>
### List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AADL</td>
<td>Architecture Analysis &amp; Design Language</td>
</tr>
<tr>
<td>CCF</td>
<td>Common Cause Failure</td>
</tr>
<tr>
<td>FAA</td>
<td>Functional Analysis Architecture</td>
</tr>
<tr>
<td>FDA</td>
<td>Functional Design Architecture</td>
</tr>
<tr>
<td>FMEA</td>
<td>Failure Modes &amp; Effects Analysis</td>
</tr>
<tr>
<td>FMECA</td>
<td>Failure Modes Effects &amp; Criticality Analysis</td>
</tr>
<tr>
<td>FTA</td>
<td>Fault Tree Analysis</td>
</tr>
<tr>
<td>HDA</td>
<td>Hardware Design Architecture</td>
</tr>
<tr>
<td>HiP-HOPS</td>
<td>Hierarchically Performed Hazard Origin and Propagation Studies</td>
</tr>
<tr>
<td>H/W</td>
<td>Hardware</td>
</tr>
<tr>
<td>S/W</td>
<td>Software</td>
</tr>
<tr>
<td>VFM</td>
<td>Vehicle Feature Model</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</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 EAST-ADL Metamodel Updates ............................................................................................................. 10
  2.1 Key Concepts and Rationale ............................................................................................................... 10
  2.1.1 Expected Modelling Features ..................................................................................................... 10
  2.1.2 Current Error Modelling Support ................................................................................................. 11
  2.2 Overview of the Updates ................................................................................................................... 13
  2.3 Updated Definitions .......................................................................................................................... 14
    2.3.1 ErrorModelType and ErrorModelPrototype .............................................................................. 15
    2.3.2 ErrorBehavior and ErrorBehaviorKind ...................................................................................... 17
    2.3.3 InternalFault, FaultFailure, CustomFaultFailure, GenericFaultFailure, and GenericFaultFailureKind .................................................................................................................. 19
    2.3.4 FaultFailureType, FaultFailurePort, FaultInPort, and FailureOutPort ........................................ 22
    2.3.5 FaultFailurePropagationLink and HazardCause ..................................................................... 24
    2.3.6 SafetyConstraint and ASILKind .................................................................................................. 26
    2.3.7 Item, Malfunction, Hazard, and Related Constructs ................................................................... 28
  2.4 Discussion on relation between error model and nominal model .................................................... 32
  2.5 Discussion on Datatype of FaultFailures ......................................................................................... 33
3 Methodology Updates ............................................................................................................................. 36
  3.1 Safety Design Flow ISO\(^2\)6262 compliance: main tasks .................................................................... 36
  3.2 EAST-ADL methodology support for the Safety Design Flow .......................................................... 38
4 External tools and plugins ....................................................................................................................... 44
  4.1 EAST-ADL Safety Analysis Plugin ...................................................................................................... 44
    4.1.1 Tool Integration .......................................................................................................................... 44
    4.1.2 Semantic Mapping Transformation ............................................................................................ 46
    4.1.3 Representation Transformation .................................................................................................. 47
  4.2 Extensions to the HiP-HOPS Safety Analysis Engine ........................................................................ 49
    4.2.1 Harmonisation of EAST-ADL and HiP-HOPS .......................................................................... 49
    4.2.2 Extensions to HiP-HOPS ......................................................................................................... 50
5 Conclusions .............................................................................................................................................. 58
  5.1 EAST-ADL Language support for Safety .......................................................................................... 58
List of Figures

Figure 1 – A general pattern of supporting non-functional constraints in EAST-ADL ......................... 11
Figure 2 – An overview of existing EAST-ADL error modelling package. ........................................... 12
Figure 3 – The required pattern of supporting safety constraints in EAST-ADL ..................................... 12
Figure 4 – An overview of the proposed update of EAST-ADL error modelling package ..................... 13
Figure 5 – ErrorModelType, ErrorModelPrototype, and related constructs ........................................ 15
Figure 6 – ErrorBehavior, ErrorBehaviorKind, and related constructs ............................................... 17
Figure 7 – InternalFault, FaultFailure, CustomFaultFailure, GenericFaultFailure, GenericFaultFailureKind, and their related constructs ................................................................. 19
Figure 8 – FaultFailurePort and its related constructs ........................................................................... 22
Figure 9 – FaultFailurePropagationLink and its related constructs ..................................................... 24
Figure 10 – HazardCause and its related constructs ............................................................................. 25
Figure 11 – SafetyConstraint, ASILKind, and related constructs ......................................................... 26
Figure 12 – Item, Malfunction, Hazard, and related Constructs ........................................................... 28
Figure 13. ErrorModel and related elements ......................................................................................... 32
Figure 14. EAST-ADL Datatypes ......................................................................................................... 34
Figure 15. EAST-ADL ErrorModel ....................................................................................................... 34
Figure 16 - Safety Design Flow linked with EAST-ADL architecture .................................................. 38
Figure 17 - EAST-ADL Item description ............................................................................................... 39
Figure 18 - Safety analysis meta-model ................................................................................................. 40
Figure 19 - Safety goal in the EAST-ADL metamodel .......................................................................... 41
Figure 20 - Safety analysis top-down approach .................................................................................. 42
Figure 21 - Safety requirements in EAST-ADL meta-model ................................................................ 44
Figure 22 - Integration between EAST-ADL and HiP-HOPS ............................................................... 45
Figure 23 - Model transformation process ........................................................................................... 46
Figure 24 - Mapping of concepts from EAST-ADL to HiP-HOPS ....................................................... 47
Figure 25 - Intermediate HiP-HOPS model ......................................................................................... 48
Figure 26 - Multi-Perspective Modelling in HiP-HOPS ....................................................................... 51
Figure 27 - ASIL decomposition .......................................................................................................... 54
Figure 28 - A single possible ASIL allocation ...................................................................................... 55
1 Introduction

EAST-ADL is an architectural description language (ADL) intended to support a model-based design process for automotive electronic engineering systems. However, one aspect of system design that is seldom fully integrated into the design process is safety – a quality that is particularly important for automotive systems, as they have a direct capacity to inflict harm and thus a responsibility to prevent such harm from occurring. By including safety as an integral requirement of the system throughout the design process, it is hoped that the result will be a safer and more robust system. Since the upcoming international safety standard, ISO 26262, defines good practice for safety-driven system design in the automotive domain, one of the objectives of EAST-ADL is to provide native, language-level support for ISO 26262 concepts, allowing EAST-ADL models to directly represent safety-related information. By also ensuring compatibility and cooperation with model-based safety analysis techniques and tools, such as HiP-HOPS, EAST-ADL will provide comprehensive support for a safety-driven, model-based design process.

One of the goals of WT3.2 is therefore to determine what EAST-ADL language and tool support is necessary to enable a safety-driven design process, as set out in ISO 26262. In doing so, we have an opportunity to develop a concept in which safety considerations are included from the start of the design process, rather than simply emerging at the end of the process and forcing the design to be adapted to take them into account. The aim, therefore, is to enable ISO 26262 safety requirements and constraints to be captured in an EAST-ADL system model and evolved throughout the design process with the aid of compatible safety analysis tools such as HiP-HOPS. In this document, we hope to present the modelling elements, methodology, and tool support necessary to achieve this aim.

The first step necessary to support safety-driven design is to support safety concepts within the EAST-ADL language itself. This means extending or updating the language's metamodel with the necessary constructs to represent safety concepts such as hazards, failures, error propagation, and safety requirements. In some cases, the language already supports such concepts but needs extending further; in other cases, new constructs have to be introduced. The language support for safety concepts is described in Section 2.

Being able to represent such concepts within a language is only useful if there is meaningful guidance on how such concepts are to be used and what they mean. This guidance can be provided by a methodology. If the methodology is based on ISO 26262, then the result should be support for a process that ensures compatibility with the sound principles of safety-driven design. The safety methodology for EAST-ADL is summarised in Section 3.

Related to this process of introducing language-level support for safety is ensuring compatibility between EAST-ADL and safety analysis tools such as HiP-HOPS. Enabling representation of safety concepts in a modelling language such as EAST-ADL is only the first step of supporting safety-driven design; the next step is to provide more active support by means of analysis tools that can be applied to system models and determine their safety and reliability characteristics. In order to achieve this, EAST-ADL must be in a form, or convertible to such a form, that is understandable to external tools. This can be accomplished by means of model transformation technology. In addition, external analysis tools – in this particular case, HiP-HOPS – may also need extending or modifying both to ensure compatibility with EAST-ADL and also to introduce support for new safety analysis techniques necessary to support the ISO 26262 safety process. The link between EAST-ADL and supporting analysis tools such as HiP-HOPS is covered in Section 4.

The additional capabilities provided by introducing support for safety requirements and analysis make it possible for a number of new safety modelling and safety analysis processes to take place. One example is ASIL decomposition, a key component of safety-driven design described in ISO 26262. During ASIL decomposition, safety requirements (represented by required ASILs) are distributed to system components or subcomponents on the basis of their contribution to the potential hazards of the system: those components that have a large (or direct) contribution to
hazards receive higher ASILs and thus more stringent safety requirements, while those components that have a smaller (or no) contribution receive lower ASILs and thus more lenient safety requirements. The design can then later be verified to see if it meets the decomposed safety requirements, e.g. by means of a fault tree analysis.

Such a process can have significant benefits during a safety-driven design process and can be repeated at different levels of abstraction as the system design evolves. However, support for ASIL decomposition relies on a number of prerequisites: the modelling language must contain constructs to represent safety requirements and error propagation, there must be a sound methodology for performing hazard analysis and assigning safety requirements, and external tools are needed to analyse the propagation of failures through the model and determine each component's contribution to the potential hazards (and thus each component's share of the responsibility for meeting the system safety requirements). A novel algorithm for the automatic decomposition of safety requirements (in the form of ASILs) to components of a design with the aid of HiP-HOPS is outlined in Section 4. The algorithm was conceived in ATESST2 and, to the best of our knowledge, is the first automatic ASIL decomposition algorithm to be developed.

Including safety-related capabilities in EAST-ADL introduces other new possibilities, such as multi-perspective safety analysis. Typically, safety analysis of hardware and software is performed separately; however, EAST-ADL's updated error modelling capabilities allow an error model to describe both the hardware, software, and even middleware of a system. The combination of multiple aspects or perspectives of a system in a single model presents unique challenges for any potential safety analysis: how can we determine the propagation of failures through a system architecture that is represented by multiple (possibly contradictory) perspectives? One of the advances put forward in this document describes a concept where failures are captured primarily in the functional or software model, while contributions from other perspectives, such as hardware, are represented by means of propagation via allocation of software functions to hardware components. In this way, for example, the effect of a power failure on a number of software functions can now be correctly taken into account.

These new capabilities are described in more detail in Section 5.

Ultimately, with the unique new capabilities introduced by extending EAST-ADL with additional safety concepts, providing ISO26262 based methodology guidelines, and ensuring compatible analysis tools support by means of the HiP-HOPS safety analysis engine, we hope to produce an effective means of designing safety-critical automotive systems.
2 EAST-ADL Metamodel Updates

The error modelling package of EAST-ADL aims to facilitate the analysis, V&V, and design of functional safety for EE systems. It provides a modelling basis for capturing the anomalies of ADLEntity, which is the abstract notion for a system, a function, a software component, or a hardware device, as well as the consequences in terms of system malfunction and hazards. This section presents the recent updates of the metamodel by describing the key concepts and updated language definitions.

2.1 Key Concepts and Rationale

The updates have been performed by assessing the current support with regard to the expected features of error modelling. The concerns are discussed in the following parts of this section.

2.1.1 Expected Modelling Features

Error modelling is concerned with the description of likely behaviors that a component/system may exhibit when it deviates from the supposed functions or behaviors. In ISO26262, the function, component, or system that is of particular concern in regards to functional safety and the related safety engineering is referred to as the item.

Error behaviors are denoted fundamentally in terms of faults, errors, and failures. According to ISO26262, a fault is the “abnormal condition that can cause an element or an item to fail”, while a failure is the “termination of the ability of an element or an item to perform a function as required”. An error is the “discrepancy between a computed, observed or measured value or condition and the true, specified, or theoretically correct value or condition”. For a system/component, errors represent the part of total system/component states that can result in the system/component failures, while being caused by the activations of internal or external faults (e.g., internal malfunctions and environmental disruptions). This is stated in ISO26262 as “a fault can manifest itself as an error within the considered element and, at the end of its latency, the error can cause a failure” and “an error can arise as a result of unforeseen operating conditions or due to a fault within the system, subsystem or component being considered”.

EAST-ADL aims to enable multi-leveled error modelling with multiple formalisms for the development of automotive EE systems. It provides explicit language support for capturing the anomalies of various system entities and for managing such information and its integration with system requirements, non-functional constraints, nominal behaviors, and V&V needs. Moreover, it will allow the integration of existing external formalisms for the representation of error behaviors (e.g., HiP-HOPS, AADL and Altarica), while providing support for model transformation and tool interoperability with the related external safety analysis techniques.

Error modelling with multiple levels of abstraction is often desired to cope with the complexity and the different needs in different development lifecycle stages.

- Structured fault-models for example provide information about the component/system failure-modes, internal and external faults, and the cause-effect relationship of faults and failure-modes. In system development, such structured models often constitute the primary means for hazard assessment and for the derivation of safety requirements and V&V needs. The modelling requires domain knowledge and general system understanding, e.g., for the identification of failure-modes and faults as well as for defining their cause-effect relationships.
- In order to simulate and analyse error behaviors and safety solutions (e.g., through simulation and fault-injection), or to validate the structured fault-models, more detailed behavior models are normally needed. Such models extend nominal behavior models with
information about the occurrences of faults and failures, errors and error transitions, and possible error handling behaviors. They differ from the structured ones in providing well-defined behavior semantics (e.g., FSM). Formal error behavior modelling in general requires more detailed information about the target systems but useful for detailed safety analysis, safety design, and test planning.

To express non-functional system constraints, EAST-ADL now adopts a general pattern by associating quality constraints with quality models, as shown in Figure 1.

![Figure 1 – A general pattern of supporting non-functional constraints in EAST-ADL.](image)

### 2.1.2 Current Error Modelling Support

Figure 2 provides an overview of the existing EAST-ADL error modelling package. This version is light-weight in syntax as it does not provide a specific construct for each of the above-mentioned concepts. The intention was to have a concise structure baseline for enabling the definitions of errors and the error propagation channels within a system, while allowing a rather high user-level modelling expressiveness in regards to the behavior semantics and thereby the modelling efficiency and flexibility in regards to external modelling formalisms. Associated with the modelling constructs, the informal semantic descriptions provide a general definition of the supported notions of error behaviors.

However, considering the needs of detailed error modelling, fidelity control, as well as the allocation of non-functional constraints, such a baseline approach needs to be further refined. For example, the *ErrorBehavior* metaclass allows the definition of failure logics/semantics of a system function or component. However, no dedicated language constructs are provided for faults and failures. According to the existing domain model, the possible faults and failures of concern are classified and declared by means of the *ErrorType* of *ErrorPort*, while being instantiated in the *failureLogic* expression for the definition of particular failure logics/semantics. Although an extension of the language has been planned, there is currently no explicit support for ensuring the consistency of declared faults/failures in the error ports and their instantiations in the logic expressions. The lack of dedicated language constructs for faults and failures also makes it difficult to support error behavior models (e.g., with UML state machines) and to allocated non-functional constraints (e.g., ASILs).
The current EAST-ADL support for error modelling is not fully in line with this general pattern for non-functional system constraints. A direct alignment implies a pattern as illustrated in Figure 3.

---

**Figure 2** – An overview of existing EAST-ADL error modelling package.

**Figure 3** – The required pattern of supporting safety constraints in EAST-ADL.
Further refinement of the error modelling support indicates the following major changes:

- Renaming `ErrorPort` to `FaultFailurePort`, `ErrorEvent` to `InternalFault`, and `ErrorPropagationLink` to `FaultFailurePropagationLink` to be better in line with the notions of fault and failure.
- Adding `FaultInPort` and `FailureOutPort`, which specialise the `FaultFailurePort`, to allow explicit reasoning of faults and failures as well as the direction of error propagation.
- Adding `FaultFailure` to allow explicit description of fault and failure instances and to facilitate integration of error description with other behavior modelling support of EAST-ADL.
- Adding `FaultFailureType`, which specialises the `ADLDesignDataType`, for a more devoted support for the declarations of fault and failure types.
- Removing the distinction of `NativeErrorBehavior` and `ExternalErrorBehavior` for simplifying the specification of multiple modelling formalisms.
- Adding `SafetyConstraint` for allowing explicit safety constraints on faults and failures.
- Redirecting the link ends of `HazardCause` to `FaultFailure` and `FeatureFlaw` for a refined support of hazard semantics.

In Figure 4, an updated version of the EAST-ADL error modelling package is presented.
The error modelling package provides specific meta-level constructs for capturing and managing the related structural and behavioral concerns.

Each ErrorModelType defines what system entity in the nominal model it targets by having an association to a core element either of EAST-ADL or of AUTOSAR. This means that the target system entity can be a function, an application program, an IO or communication software, or a hardware device. In particular, for an ErrorModelType, there is always one targetType, specifying target system entity the error model refers to. An ErrorModelType can also have variants to cope with the variability of minimal architecture.

Following the same type-prototype pattern for hierarchical models as the ADLFunctonType and ADLFunctionPrototype for nominal structure, an error model is given by both ErrorModelType and ErrorModelPrototype. This allows that a hierarchical setup of error models along with the nominal system composition hierarchy. In particular, this means that the instances of an ErrorModelType in the form of ErrorModelPrototype can be included as parts in a particular analysis context given by another ErrorModelType.

In each ErrorModelType there are a set of metaclasses introduced for an explicit representation of structure for the propagation of failures/faults across components/systems. Such a structure for error propagation can differ from the targeted architecture and thereby provides an analysis view of the target architecture for specifying error behaviors and safety constraints. This separate view allows effective error modelling when IP protection prevents the disclosure of details within a system part (e.g., the involved nominal ports, transfer functions, and the connectivity), or when nominal ports and connections are not significant or sufficient for error propagations. In particular, the latter case happens in the following situations:

- For common cause hardware failures, nominal communication connectors or composition relationship will not be sufficient for error propagate from memory and CPU to software components sharing the same hardware resources.
- Certain nominal ports and connections are dedicated for certain system modes, such as relating only to system start-up/shut-down, particular application modes, QoS (quality of service) adaptation, and therefore could not be significant for a particular error propagation or hazard perspective under consideration.

An ErrorModelType has fault/failure ports (FaultFailurePort) for declaring the access point to and from the error model. Each fault/failure port specifies the allowed fault/failure types (e.g., logical, software specific, or physical faults) as well as the actual fault/failure instances being received/transmitted. While the fault/failure propagations between error models are described by the links of fault/failure ports (FaultFailurePropagationLinks), the fault activations and error propagations inside one error model are described by the ErrorBehavior. The faults of an error model can be either internal faults (InternalFault) of itself, or external faults (FaultFailure) received via ports (FaultInPort) for failures propagated from other error models.

A SafetyConstraint either targets a external fault (cause), an internal fault (cause), or a failure (consequence). The constraint says how severe it is, i.e. what ASIL level it has. Furthermore it specifies how large deviation (toleratedDeviation) from the nominal behavior that is considered as the safety-critical fault or the failure having the associated safety constraint with this ASIL.

### 2.3 Updated Definitions

This section provides the definitions of key constructs that have been revised or added in the updated version of EAST-ADL error modelling package.
2.3.1 ErrorModelType and ErrorModelPrototype

ErrorModelType and ErrorModelPrototype are introduced to support the hierarchical composition of error models based on the type-prototype pattern also adopted for the nominal architecture composition. See Figure 5.

Figure 5 – ErrorModelType, ErrorModelPrototype, and related constructs

ErrorModelType

The ErrorModelType metaclass represents the containers for maintaining the information relating to the anomalies of a system element.

The ErrorModelType inherits the abstract metaclass ADLTraceableSpecification, allowing this kind of analytical information to be explicitly described in a similar way as requirements, test cases and other specifications.

An ErrorModelType provides the following attributes and association properties:

- **targetType:** Identifiable[1]
  - the target element (i.e., a system, a function, a component, or hardware device) owning the anomalies.

- **faultFailurePort:** FaultFailurePort[*]:
  - the owned ports for propagations of faults/failures to/from the targetType.

- **ErrorBehaviorDescription:** ErrorBehavior[1..*]
  - the specification of failure behaviors of the targetType.

- **part:** ErrorModelPrototype[*]
  - the contained subordinate error models.

- **faultFailureConnector:** ErrorPropagationLink[*]
  - the contained links for internal propagations of faults/failures between the subordinate error models.

- **genericDescription:** String
  - the generic statement of the error model

Semantics:
The ErrorModelType provides a specification of information related to the anomalies of a system, a function, a software component, or a hardware device.
Extension:
(see ADLTraceableSpecification)

**ErrorModelPrototype**

The *ErrorModelPrototype* metaclass represents subordinate error models in an error model according to type-prototype pattern. It is used to support hierarchical error modelling, where an *ErrorModelType* have other *ErrorModelType* as its parts. The subordinate error models together support a more detailed error description.

An *ErrorModelPrototype* provides the following association property:

- **type:** *ErrorModelType* [1] the ErrorModelType that types the ErrorModelPrototype.

**Semantics:**

An *ErrorModelPrototype* represents a unique compositional occurrence of the *ErrorModelType* that types it in the containing *ErrorModelType*.

Extension:
(See ADLFunctionPrototype)
2.3.2 ErrorBehavior and ErrorBehaviorKind

**ErrorBehavior**

The *ErrorBehavior* metaclass represents the descriptions of error behaviors that a system, a function, a software component, or a hardware device can have. It provides information about the fault activations and error propagations of the target system element indicated by the containing error model. See Figure 6.

An *ErrorBehavior* provides the following attributes and association properties:

- **owner: ErrorModelType [1]** the container *ErrorModelType* for the error behavior description.
- **internalFault: InternalFault [*]** the occurrences of internal faults.
- **faultFailure: FaultFailure [1..*]** the occurrences of external faults or the derived failures.
- **failureLogic: String [0..1]** the specification of error behavior based on an external formalism or the path to the file containing the external specification.
- **type: ErrorBehaviorKind [1]** the type of formalism applied for the error behavior description.

**Semantics:**

Each *ErrorBehavior* description captures the likely fault activations and error propagations that the target system element of the container *ErrorModelType* can have. For the target system element, the description relates the occurrences of external and internal faults with the errors and also provides information about the propagation of such errors to failures.
An `ErrorBehavior` description can be based on different formalisms, depending on the analysis techniques and tools of interest and available. This is indicated by its `type:ErrorBehaviorKind` attribute. The attribute `failureLogic:String` is only used for external formalisms other than the EAST-ADL native support.

**Extension:**

UML:Behavior

**ErrorBehaviorKind**

The `ErrorBehaviorKind` metaclass represents an enumeration of literals describing various types of formalisms used for specifying error behavior.

The included enumeration literals are:

- **EAST_NATIVE**
  
  a specification of error behavior with the native EAST-ADL support.

- **HIP_HOPS**
  
  a specification of error behavior according to the external formalism HiP-HOPS.

- **ALTARICA**
  
  a specification of error behavior according to the external formalism ALTARICA.

- **AADL**
  
  a specification of error behavior according to the external formalism AADL.

- **OTHER**
  
  a specification of error behavior according to other user defined formalism.
2.3.3 InternalFault, FaultFailure, CustomFaultFailure, GenericFaultFailure, and GenericFaultFailureKind

The InternalFault, FaultFailure, CustomFaultFailure, GenericFaultFailure, and GenericFaultFailureKind provide support for specifying the faults and failures of a system element. See Figure 7.

**InternalFault**

The InternalFault metaclass represents the particular internal anomalies that a system, a function, a software component, or a hardware device can have. It provides information about the internal abnormal conditions of the target system element in its error behavior description. Internal faults are defined in the context of an error behavior definition.

An InternalFault provides the following attributes:

- **description**: String  
  a statement of the internal fault.

- **failure_type**: String  
  a definition of internal failure type.

- **failure_rate**: int [0..1]  
  a specification of the density of probability of failure divided by probability of survival for a hardware element (ISO26262).

- **repair_rate**: int [0..1]  
  a specification related to the mean time to/between failure.

**Semantics:**

The system anomaly represented by an InternalFault, which when activated, can cause errors and failures of the target element.

**Extension:**

UML::DataType / UML::Event
FaultFailure

The *FaultFailure* metaclass represents the particular faults/failures that propagate across systems, functions, software components, or hardware devices. It is an abstract class further specialised by *CustomFaultFailure* and *GenericFaultFailure*.

External faults and failures are defined in the context of fault and failure propagation definition (i.e., *FaultFailurePort*), while their occurrences are related to the internal faults and errors in error behaviors.

Semantics: *FaultFailure* provides information about anomalies of a system element, manifested either in terms of external faults or in terms of failures. An external fault is concerned with the abnormal environmental conditions of a system element, which when activated, can cause errors of the target system element. Such errors, when propagated internally in the target system element, will case the failures of the element.

Extension: UML::DataType / UML::Event

CustomFaultFailure

The *CustomFaultFailure* metaclass specialises the *FaultFailure* for costumed faults/failures. A *CustomFaultFailure* provides the following attribute:

- **value**: String
  
  The value of the custom external fault/failure type.

Semantics: (See *FaultFailure*)

Extension: (See *FaultFailure*)

GenericFaultFailure

The *GenericFaultFailure* metaclass specialises the *FaultFailure* for generic faults/failures. A *GenericFaultFailure* provides the following attribute:

- **value**: GenericFaultFailure
  
  The value of the generic external fault/failure.

Semantics: (See *FaultFailure*)

Extension: (See *FaultFailure*)

GenericFaultFailureKind

The *GenericFaultFailureKind* metaclass represents an enumeration of literals describing generic types of faults/failures external to/from a system element.

The included enumeration literals are

- **omission**
  
  an omission fault/failure.

- **commission**
  
  a commission fault/failure.
- **tooLargeValue**
  a fault/failure when a value has too large value.
- **tooSmallValue**
  a fault/failure when a value has too small value.
- **tooLate**
  a fault/failure when a value is generated too late.
2.3.4 FaultFailureType, FaultFailurePort, FaultInPort, and FailureOutPort

The metaclasses FaultFailureType, FaultFailurePort, FaultInPort, and FailureOutPort provide support for specifying the structure of failure/fault propagations across components/systems. See Figure 8.

**Figure 8 – FaultFailurePort and its related constructs**

**FaultFailureType**

The FaultFailureType metaclass is a specialised ADLDesignDataType for specifying the set of allowed faults/failures to be received/transmitted through fault/failure ports. It provides information about the dimension (e.g., functional, software specific, and physical fault/failure) and resolutions (e.g., omission, commission, value, and timing fault/failure) of faults/failures in a fault/failure port. The actual faults/failures received/transmitted by a Fault/Failure port are given by the faultFailure inside the port.

**Semantics**

FaultFailureType is a specialisation of ADLDesignDataType.

**Extension:** See ADLDesignDataType

**FaultFailurePort**

The FaultFailurePort metaclass represents the access points through which faults/failures propagate to/from an ErrorModelType. Each FaultFailurePort also specifies the types of faults/failures that are allowed to traverse through it to/from the containing ErrorModelType by its type declaration. The actual fault/failure instances that are received/transmitted through it to/from the containing ErrorModelType are given by its contained faults/failures.

A FaultFailurePort provides the following attributes:
• **type: FaultFailureType [1]** the declaration of port type.

• **faultFailure:FaultFailure [1..*]:** the fault/failure instances received/transmitted through the port.

The *FaultFailurePort* is an abstract class further specialised by *FaultInPort* and *FailureOutPort*.

Two fault/failure ports are compatible if

- Their types have the error data of the same dimension (i.e., both in logical dimension or both in physical dimension).
- Their types have the error data of the same resolution (i.e., both allowing the same set of failure modes).

Semantics:
The *FaultFailurePort* declares a transmission point of faults/failures to/from its containing ErrorModelType.

Extension:
UML::Port

**FaultInPort**
The *FaultInPort* metaclass specialises the *FaultFailurePort* for the reception of external faults to the containing ErrorModelType.

Semantics:
The faults of a *FailureInPort* indicate the receiving external faults, which will be treated in the ErrorBehavior specification of the containing ErrorModelType of the port.

Extension:
UML::Port

**FailureOutPort**
The *FailureOutPort* metaclass represents a description of error events that can propagate and traverse out from the containing ErrorModelType as failures.

Semantics:
The failures of a *FailureOutPort* indicate the transmission of failures, which can propagate to other system elements or to the environment.

Extension:
UML::Port
### 2.3.5 FaultFailurePropagationLink and HazardCause

#### FaultFailurePropagationLink

The `FaultFailurePropagationLink` metaclass represents the links for the propagations of faults/failures across system elements. In particular, it defines that one error model provides the faults/failures that another error model receives. See Figure 9.

A fault/failure link can only be applied to compatible `FaultFailurePorts`, either for fault/failure delegation within an error model or for fault/failure transmission across two error models. A `FaultFailurePropagationLink` can only connect fault/failure ports that have compatible types (see also `FaultFailurePort`).

A `FaultFailurePropagationLink` provides the following association properties:

- **port**: `FaultFailurePort` [2] the connected `FaultInPort` and `FailureOutPort`.

**Semantics:**

The `FaultFailurePropagationLink`, when applied for a nominal connector (e.g., a communication connector) or a design relationship (e.g., an allocation relationship), signifies that the connected faults/failures traverse along such a nominal connection.

**Extension:**

UML::Connector

![Figure 9 – FaultFailurePropagationLink and its related constructs.](image)

#### HazardCause

The `HazardCause` metaclass represents the dependency relationship between the failures of system elements and the malfunctions of an item. In particular, it signifies that the definition of a failure of system element is necessary for the definitions of malfunctions of items. In other words, the definitions of item malfunctions are semantically dependent on the definitions of system element failures. See Figure 10.
A HazardCause provides the following association properties:

- **faultFailure: FaultFailure [1]** The failure of system element used for the definitions of malfunction(s) (i.e., the supplier of dependency relationship).

- **malfunction: FeatureFlaw [1..*]** The malfunction(s) dependent on the associated failures (i.e., the client of dependency relationship).

Semantics:
A HazardCause dependency signifies an information supplier/client relationship between failure and malfunction definitions. A change on a failure definition may have impact on the malfunction definitions.

Extension:
UML::Dependency
2.3.6 SafetyConstraint and ASILKind

SafetyConstraint and ASILKind provide information about the constraints on faults and failures due to safety design decisions. See Figure 11.

**Figure 11 – SafetyConstraint, ASILKind, and related constructs.**

**QuantitativeSafetyConstraint**

The QuantitativeSafetyConstraint metaclass represents the quantitative integrity constraints on a fault or failure. Thus, the system has the same or better performance with respect to the constrained fault or failure, and depending on the role this is either a requirement or a property.

A QuantitativeSafetyConstraint has two attributes representing the failure and repair rate respectively of the constrained fault or failure. Both represent the number of failures/repairs per unit time.

Semantics:

A QuantitativeSafetyConstraint provides information about the probabilistic estimates of target faults/failures, further specified by the failureRate and repairRate attribute.

Extension:

(see ADLTraceableSpecification)

**SafetyConstraint**

The SafetyConstraint metaclass represents the qualitative integrity constraints on a fault or failure (via the association constrainedFaultFailure). Thus, the system has the same or better performance with respect to the constrained fault or failure, and depending on the role this is either a requirement or a property.

A SafetyConstraint can have several different origins. One is to realize a safety requirement (that is only textual). By pointing to a certain (set of) failure and with what ASIL it should be avoided, this safety requirement has then come into the model. Another way is to use it as is as a kind of a data sheet, in which it is promised that a certain failure is promised to be constrained by a certain ASIL value. But also assumptions can be modelled this way. By setting a SafetyConstraint on a fault
(cause) rather than a failure (consequence), such an assumption is also modelled in a well-defined way.

Semantics:
A SafetyConstraint defines qualitative bounds on the constrainedFaultFailure in terms of safety integrity level, asILValue.

Depending on role, the SafetyConstraint may define a required or an actual safety integrity level.

Extension:
(see ADLTraceableSpecification)

ASILKind
The ASILKind is an enumeration metaclass with enumeration literals indicating the level of safety integrity in accordance with ISO26262.
2.3.7 Item, FeatureFlaw, Hazard, and Related Constructs

Item, FeatureFlaw, and Hazard provide support for hazard analysis and risk assessment. See Figure 12.

Figure 12 – Item, FeatureFlaw, Hazard, and related Constructs.

**Item**

The Item metaclass is one means to represent the objective for defining and describing the notation of item and to develop an adequate understanding of it, so that each activity defined in the safety lifecycle of ISO26262 can be performed. Safety analyses are carried out on the basis of an item definition and the safety concepts is derived from it.

The Item is intended for annotation and to add the specified attributes on the item under safety analysis. The ADLContext sets the boundary and interfaces for the item under safety analysis.

An Item provides the following attributes and association properties:

- **vehicleFeature : VehicleFeature [1..*]**
- **developmentCategory : DevelopmentCategoryKind**

It shall be determined whether the item is a modification of an existing item or if it is a new development.
Semantics:
Item represents the scope of safety information and the safety assessment through its reference to one or several Features.

Extension:
UML::Class

**FeatureFlaw**
FeatureFlaw denotes an abstract failure of a set of items, i.e. an inability to fulfill one or several of its requirements.

It provides the following attributes and association properties:

- **nonFulfilledRequirement : Requirement [**]
  Identifies the requirements that are not fulfilled.

- **item : Item [1..*]
  The item for which the FeatureFlaw is identified

Semantics:
A *item* is the function, component, or system that is of particular concern in regards to functional safety and the related safety engineering.

Extension:
UML::Class

**Hazard**
The *Hazard* metaclass represents a condition or state in the system that may contribute to accidents. It is usually a failure of some kind, but may also be a result of nominal operation.

The *Hazard* does not address hazards as electric shock, fire, smoke, heat, radiation, toxicity, flammability, reactivity, corrosion, release of energy, and similar hazards unless directly caused by malfunctioning behavior of safety related EE-systems.

A *Hazard* provides the following attributes and association properties:

- **malfunction : FeatureFlaw [1..*]
  The deviation of the item’s operation compared to specified behavior.

- **item : Item [1..*]
  The item for which the FeatureFlaw is identified

Semantics:
No additional remarks.

Notation:
The Hazard is shown as a solid-outline rectangle with "Haz" at the top right. It contains the name of the Hazard and optionally the name of the source entity.

Extension:
UML::Class
HazardousEvent

The HazardousEvent metaclass represents a combination of a Hazard and a specific situation, the latter being characterized by operating mode and operational situation in terms of a particular use case, environment and traffic.

A HazardousEvent provides the following attributes and association properties:

- **controllability**: ControllabilityClassKind
  
  The controllability by the driver or other traffic participants shall be estimated. The controllability shall be assigned to one of the controllability attributes C0, C1, C2 or C3 in accordance with ISO26262.

- **exposure**: ExposureClassKind
  
  The probability of exposure of the operational situations shall be estimated. The probability of exposure shall be assigned to one of the probability attributes E1, E2, E3 or E4 in accordance with ISO26262.

- **Severity**: SeverityClassKind
  
  The severity of potential harm shall be estimated. The severity shall be assigned to one of the severity attributes S0, S1, S2 or S3 in accordance with ISO26262.

- **hazardClassification**: ASILKind
  
  The ASIL-Level shall be determined for each hazardous event using the estimation parameters in accordance with ISO26262.

- **classificationScheme**: String
  
  The classificationScheme attribute denotes identification of applicable ISO26262.

- **classificationAssumptions**: String[0..1]
  
  The classificationAssumptions attribute denotes assumptions concerning the classification of the Hazard.

- **operatingMode**: Mode[*]
  
  OperatingMode denotes the Operating mode of the item

- **externalMeasures**: RequirementsRelationship[*]

- **environment**: OperationalSituation[*]
  
  A definition of the road environment in terms of road conditions, lanes, geometry, etc. Represents the external and static aspects of the vehicle operating situation.

- **traffic**: OperationalSituation[*]
  
  A definition of the traffic situation in terms of adjacent vehicles, pedestrians and other dynamic aspects. Represents the external and dynamic aspects of the vehicle operating situation.

- **operationalSituationUseCase**: UseCase[1..*]
  
  Operational situation with respect to the activities of actors, typically the driver.

- **hazard**: Hazard[1..*]
  
  The Hazard that together with the operational situation constitutes the HazardousEvent
Semantics:
The HazardousEvent denotes a combination of a Hazard and an operational situation. The controllability and severity attributes shall be consistent with the operational situation and operational scenario, and the Exposure shall reflect the likelihood of the operational situation and scenario.

**DevelopmentCategoryKind**

The DevelopmentCategoryKind metaclass in an enumeration with enumeration literals indicating whether the item is a modification of an existing item or if it is a new development.

**SeverityClassKind**

The SeverityClassKind is an enumeration metaclass with enumeration literals indicating the severity attributes S0, S1, S2 or S3 in accordance with ISO26262

**ControllabilityClassKind**

The ControllabilityClassKind is an enumeration metaclass with enumeration literals indicating controllability attributes C0, C1, C2 or C3 in accordance with ISO26262

**ExposureClassKind**

The ExposureClassKind is an enumeration metaclass with enumeration literals indicating the probability attributes E1, E2, E3 or E4 in accordance with ISO26262.
2.4 Discussion on relation between error model and nominal model

The ErrorModelType defines the error behavior for a nominal model. The nominal model is referenced by a target association. There are several cases to consider:

- **One nominal type:**
  The error model represents the identified nominal type wherever this nominal type it is instantiated.

- **Several nominal types:**
  The error model represents the identified nominal types individually, i.e. the same error model is reused.

- **One nominal prototype with instanceref:**
  The error model represents the identified nominal prototype whenever its context, i.e. its top-level composition is instantiated.

- **Several nominal prototypes with instanceref:**
  The error model represents the identified set of nominal prototypes whenever their context, i.e. their top-level composition is instantiated.

In general, ports on the errormodelType may be associated with ports on the nominal model. Connectors between nominal ports can then be used as a pattern for connecting error ports when this error model is instantiated as an ErrorModelPrototype.
The ErrorModelPrototype also defines the error behavior for a nominal model. Because it is a prototype, it is not reusable in itself. Identifying nominal components therefore only serves as a declaration of which part of the nominal system is covered by the respective ErrorModelPrototype.

The nominal elements are referenced by a target association. There are several cases to consider:

- **One nominal prototype with instanceref:**
  
  The error model prototype represents the identified nominal prototype every time the nominal prototype’s context, i.e. its top-level composition is instantiated.

  To be consistent, the corresponding errormodelPrototype should be associated with the nominal prototype’s type

  *This is redundant information if the nominal type is referenced by a single errorModelType.*

- **Several nominal prototypes with instanceref:**
  
  The error model represents the identified set of nominal prototypes every time their context, i.e. their top-level composition is instantiated.

  *This is redundant information if the set of nominal prototypes is referenced by a single errorModelType.*

### 2.5 Discussion on Datatype of FaultFailures

The regular datatype has an attribute "Semantics". Semantics defines the semantic meaning of the EnumerationLiteral.

Example: For enumerationLiteral Open, the semantics may be "the door angle exceeds 1 degree"

FaultFailures, i.e. faults on input ports, failures on outputfaults and internal faults are typed by the regular Datatype.

Faults are typically enumeration types, however, as they convey the error propagation viewpoint. A typical enumeration type may thus be

BrakeFaultFailureType

- **BrakePressuretooLow**
  Semantics="brake pressure is below 20% of requested value"

- **Omission**
  Semantics="brake pressure is below 10% of maximal brake pressre"

- **Comission**
  Semantics="brake pressure exceeds requested value with more than 10% of maximal brake pressure"

Semantics may also a more formal expression defining in the type of the nominal datatype what value range is considered a fault. This depends on the user and tooling available.
In the error model, this means that each port has a set of well-defined values typed by a Datatype. In most cases this is an enumeration type. Each literal has a semantics which defines the meaning of the fault failure’s value. It is also possible to refer the fault failure port to the set of nominal ports it relates.

**Figure 14. EAST-ADL Datatypes**

In the example below, the error model has its datatype defined as omission and commission. The error model port identifies which nominal port it corresponds to.

**Figure 15. EAST-ADL ErrorModel**
BrakeFaultType
(omission ("value below 10% of request"), omission ("value above request with more than 10% maxrange"))

Nominal Model

Func B

Func A

Nominal Model

ErrorModelB

ErrorModelA

Prototype a

Prototype b

isOfType

isOfType

isOfType

isOfType

BrakeFaultType
(omission ("value below 10% of request"), omission ("value above request with more than 10% maxrange"))

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

BrakeFaultType
(omission ("value below 10% of request"), omission ("value above request with more than 10% maxrange"))

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType

isOfType
3 Methodology Updates

The ISO° 26262 requires conducting safety activities, beginning from the preliminary development phase of automotive systems and including the whole product life-cycle, in order to design a safer automotive system. Therefore it is necessary to integrate safety activities with the common development activities, a process that EAST-ADL supports by means of its Safety Design Flow methodology.

3.1 Safety Design Flow ISO° 26262 compliance: main tasks

All the necessary safety activities are encompassed in the safety design flow [1] which includes the tasks necessary to perform a safety analysis compliant with ISO°26262 [2]. The safety design flow activities are focused on the concept and design phases of the safety lifecycle.

The main tasks of the safety design flow can be briefly summarised as follows:

✓ System Functional Analysis and Boundary definition

The first step of safety lifecycle consists of identifying and describing the Item under safety analysis, and to develop an adequate understanding of it. This is a necessary step, since the subsequent phases of the safety design flow are based on the item definition and the safety concept is derived from it.

To have a satisfactory understanding of the Item, it is essential to properly analyse the item itself in terms of input(s)/output(s), functionality, interfaces, and how the item interacts with the vehicle and/or with the environment.

Moreover it is necessary to carry out an impact analysis by analysing the development category (new development or modification) of the item:

a) In the case of a new development, the entire safety lifecycle shall be examined. It is possible to define the item target feature (the feature description in terms of vehicle output(s) behavior).

b) In the case of a modification to an existing item, then an impact analysis shall be applied and a tailored safety lifecycle is required. In this case, all the information and documentations are collected. The VFM and the FAA, inherited from the original item, shall be used to evaluate the item and to directly determine the target function (the function description in terms of its output(s) behavior), which will be the base for the subsequent malfunctions definition phase.

The boundary of the item and the item’s interfaces with other elements shall also be determined; functions of the item can be classified according to whether they are safety critical or not.

✓ Hazard identification & Hazardous events definition

To evaluate the risk associated with the item under safety analysis, a risk assessment should be carried out, starting from the scenarios definition (all variables and/or states that characterise the functions or affect them, including both operative conditions and environmental conditions), in respect of the item functionality under analysis.

To identify the hazards it is essential to define the malfunction(s), the misuse(s) and eventual maintenance condition(s) related to the item. If the item target feature(s) / target function(s) has been correctly identified and described, the malfunction can be always defined in terms of anomalies of function activation. Therefore the hazards have to be identified. A hazard is describable in terms of potentially dangerous vehicle behavior (i.e. "the vehicle is not braking on driver's demand").
Each identified hazard, when applied to the scenarios, triggers the so-called "hazardous events", which are combinations of a hazard and an operational situation.

✓ Risk assessment

Once the hazardous events have been defined, they each need to be classified according to their ASIL (Automotive Safety Integrity Level), which specifies the safety requirements, necessary to the item, for achieving an acceptable residual risk. The ASIL is determined on the basis of the controllability, severity, and exposure time estimated for each hazardous event. The result is an ASIL class from A (lowest) to D (highest), or QM (quality management - only the functions that are considered to be not safety relevant can belong to QM class).

✓ Safety Goal and Safe State Definition.

If the item under safety analysis has only hazardous events classified as QM, then the safety analysis can be stopped. If, instead, the item cannot be classified as Quality Management, then the Safety Goal (top-level safety requirements) and, if applicable, the Safe State for each hazardous event should be defined. ASILs are assigned to safety goals to ensure that the item will meet those goals.

✓ Risk Assessment Approval.

The estimated controllability values assigned to the several situations should be validated. Therefore, specific testing on road shall be carried out for the final value assessment. The idea consists of performing a “Test Drive” by using a significant vehicle to inject the faults that determine the failures, and monitoring the drivers’ opinions and reactions.

✓ Functional safety requirements definition.

For each safety goal and safe state (if applicable) that are the results of the risk assessment, at least one safety requirement shall be specified – the defined safety goals are the “Top-level functional safety requirements”. To achieve this result, it is necessary to perform qualitative and quantitative safety analyses such as FMEA and FTA to determine any common causes or single points of failure. If severity information is also provided, then more detailed criticality estimation can also be produced, e.g. in an FMECA. This can also be accomplished in EAST-ADL by means of external tools such as HiP-HOPS. At this stage, the process may also involve a multi-perspective analysis (see Section 4.2.2.1).

To satisfy the safety goals for each identified failure mode the diagnosis and recovery actions shall be specified.

As specified in the ISO 26262 (Part 3), ASIL, operating modes, fault tolerant time spans (if applicable), safe states (if applicable), emergency operation times (if applicable) and functional redundancies (if applicable) should all be attributes of the functional safety requirements.

✓ Technical safety requirements definition.

Once the functional safety requirements have been defined, the item can be developed further by specifying the technical safety requirements. These describe how to implement the safety measures described by the functional safety requirements, or in other words, what the safety requirements must be for each element of the detailed technical architecture if the design is to fulfil the safety requirements for the system as a whole. The process of defining the technical safety requirements is an iterative process that involves both ASIL decomposition and criticality analysis.
3.2 EAST-ADL methodology support for the Safety Design Flow

Thanks to the integration of EAST-ADL with the most important safety concepts, all ISO\textsuperscript{26262} safety design flow tasks can be modelled at different abstraction levels. The following figure shows how the safety design flow could be linked to EAST-ADL architecture.

![Figure 16 - Safety Design Flow linked with EAST-ADL architecture](image-url)
Item(s) description

The system functional analysis starts from the Vehicle level, but the identification and description of the Item spans all EAST-ADL abstraction levels. In fact, since the item is a system or array of systems or functions/features that are subject to safety assessment, it is not possible to have a strict description of what the Item is in terms of modelling elements. So, the Item description can be performed at each abstraction level, starting from the vehicle level, down to the operational architecture level, as highlighted in the following figure.

![EAST-ADL Item description diagram](image-url)

*Figure 17 - EAST-ADL Item description*
Safety analysis metamodel

Since the Item is modelled on each EAST-ADL abstraction level, the Item boundary also lacks a strict definition in EAST-ADL. The Item boundary can be defined on FAA and FDA level in terms of ADLFunctions, on HDA and MWA level in terms of H/W entities, and on implementation level in terms of S/W, H/W and BSW components.

Following a top-down approach, the safety analysis can start from the VFM level, beginning from the “target feature” definition. Based on the target feature, the hazards are identifiable independently of the architectural solution. Therefore, on VFM level, it is already possible to perform a hazard analysis and Risk assessment to evaluate, in a preliminary manner, the “safety relevance” of the Item under safety analysis. For this purpose, the hazards should be evaluated in different scenarios for assessing Severity, Controllability and Exposure. The relevant scenarios could be derived from the safety-oriented Use Cases [1]. The hazard under analysis, when applied to the various operational situations (operative & environmental conditions), triggers the so called “hazardous events” (HE).

The introduction of “hazardous event” forces the modification of the metamodel specification, defining a relation among Hazard, Malfunction, and Use Case with the HE, as described in the following figure.

![Figure 18 - Safety analysis meta-model](image)

Each Hazardous Event (HE) has to be classified in terms of associated risk (ASIL). Since the identified hazardous events are related to a target feature, it makes sense to define (for each hazardous event that appears safety relevant) the Safety Goals (SG) on VFM level too. In EAST-ADL, the Safety Goal artefact is modelled as a specialisation of ADLRequiremnt. The ASIL determined for the hazardous event should be assigned to the corresponding safety goal.

NB:

Summarising, a Hazard has one or more HE associated with it, and for each HE, a SG is assigned. In most cases, typically with “single function” items, the different SG (related to each HE) may be compatible, so the safety goal can be formulated in terms of the negation of the Hazard, and therefore the Safety Goal should refer to the Hazard that it addresses.
ASILs and Safe States (if defined) are attributes of Safety Goals.

Figure 19 - Safety goal in the EAST-ADL metamodel
Risk Assessment check and completion

To verify the correctness and completeness of the preliminary Hazard analysis and risk assessment performed on VFM level, a deeper analysis has to be performed, by looking at the architectural level. Therefore the target function on FAA abstraction level should be defined by deriving it from the target feature introduced at the upper abstraction level. At this point it is possible to define the malfunction as anomalies of the item's outputs. By analysing the malfunctioning behaviors of the item, it is possible to perform a complete risk assessment and so to verify the completeness of the hazard list and the correctness of the risk analysis performed at vehicle level. In any case, the process is iterative: it is necessary to provide the possibility to make a back annotation on the safety analysis results, for updating the hazards, the Safety Goals and the Safe State.

The top-down approach above described is intended to be applicable whether or not the item/function is a new development.

In the case of a modification of an already existing item, then an impact analysis is required and a tailored safety lifecycle is advisable. Therefore, with the hypothesis that the safety analysis on VFM level is already available (inherited from original item), the most convenient approach is the bottom-up one, i.e. by entering directly on the FAA level and by verifying the impact in terms of differences in hazard list and risk assessment outcomes.

For each safety goal and safe state (if applicable), which are the results of the risk assessment, at least one functional safety requirement should be specified. The definition of functional safety requirements only makes sense in EAST-ADL at the FAA level. At this level, the goal is to verify...
that the functional safety concept realises all safety goals defined at VFM level. Each safety requirement could be associated with one or more ADLFunctions.

**Safety requirements architectural allocation**

Once the functional safety concept is specified, the item can be developed at the system perspective, starting from the technical safety requirements specification.

The safety requirements should be allocated to architectural elements, as early as those are available, starting from preliminary architectural assumptions and, finally, to hardware and software elements. These activities can be performed only at FDA level, when the item is realised in a more implementation-oriented manner.
4 External tools and plugins

This section describes the safety analysis tool support for EAST-ADL. Tool support is provided primarily by the HiP-HOPS tool with an EAST-ADL plugin to serve as the interface to HiP-HOPS. Both the plugin and the integration of HiP-HOPS with EAST-ADL concepts will be described below.

4.1 EAST-ADL Safety Analysis Plugin

The EAST-ADL Safety Analysis Plugin is a tool providing the means to perform safety analysis on systems modelled using the EAST-ADL language. In EAST-ADL, the user explicitly models the propagation of errors in a so called error model. The error model can be created easily because it is built from local error information. For system safety, however, the global effects of the error propagations are of interest. Tools such as HiP-HOPS can calculate the global effects through safety analyses like FMEAs and FTAs by combining the local error information. To provide a view of the global failure effects to EAST-ADL users without additional effort on the users' side, we provide an automated link between EAST-ADL and HiP-HOPS.

4.1.1 Tool Integration

To establish the link between EAST-ADL design tools such as Papyrus and safety analysis tools such as HiP-HOPS, they have to be integrated. Integration is described by Wassermann [3] to have five different aspects:

- **control integration**: programs can interoperable;
- **data integration**: programs can use each others data;
- **presentation integration**: programs have a unified GUI;
- **platform integration**: services provided by platform;
- **process integration**: the software development processes are integrated.

The plugin mainly performs three forms of integration: data, control and presentation integration. All of these forms are realised in the form of a plugin for the Eclipse framework.
4.1.1.1 Control and Presentation Integration

This link is provided in the form of a plugin. This ensures seamless integration in the modelling environment (Papyrus/Eclipse) and keeps the safety analysis overhead experienced by the user as low as possible and thus allows for an iterative development process.

Without tool support, safety analysis requires tedious, manual work which is frequently seen as an obstacle by engineers, often reducing the scope of this task or limiting it to a single safety analysis. Automation of safety analysis has several advantages: it makes safety analysis easier, it is readily available, and it allows the engineers to obtain a thorough and quick analysis of their design. This rapid 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.

4.1.1.2 Data Integration

Data integration in this context is concerned with the transformation of error modelling 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. Identifying the right model transformation language/engine for the task at hand is a fundamental part of the solution.

4.1.1.3 Transformation Design

We partitioned the model transformation process from EAST-ADL to HiP-HOPS into two steps. Each step has a separate purpose and concern.
Figure 22 - Model transformation process

(1) Semantic Mapping Transformation: The first transformation step transforms an EAST-ADL model that is created in the Papyrus UML modelling tool and creates 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 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.

In the following 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 parallelise 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 the best tool for each concern.

4.1.2 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. The mapping between EAST concepts and HiP-HOPS concepts is explained in the table below.
Table 2 - Mapping of concepts from EAST-ADL to HiP-HOPS

<table>
<thead>
<tr>
<th>EAST-ADL</th>
<th>HIP-HOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorModelType</td>
<td>System</td>
</tr>
<tr>
<td>ErrorModelType.errorConnector of type ErrorPropagationLink</td>
<td>System Lines</td>
</tr>
<tr>
<td>ErrorModelType.parts of type ErrorModelPrototype</td>
<td>System Component</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorPort of type ErrorPort</td>
<td>System Component.Ports</td>
</tr>
<tr>
<td>ErrorModelPrototype</td>
<td>System ComponentImplementation</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorBehaviorDescription.internalErrorEvent of type ErrorEvent</td>
<td>System.Component.Implementation.FData.basicEvent</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.genericDescription of type String</td>
<td>System.Component.Implementation.FData.outputDeviation</td>
</tr>
</tbody>
</table>

Figure 23 - Mapping of concepts from EAST-ADL to HiP-HOPS

Model to model transformations are well suited for a semantic mapping transformation. Both input and output of a model to model transformation are models themselves. Mapping patterns can be described by relational and declarative transformation languages in a concise manner. We chose 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 the case of EAST-ADL the metamodel consists of the UML metamodel and the EAST-ADL profile.

4.1.3 Representation Transformation

The purpose of the Representation Transformation is the generation of a textual description based on the intermediate model. 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 serialising the model as text.

Textual representations can be generated particularly well with model to text transformation languages. We chose the Xpand language from OpenArchitectureWare. Xpand is a template-based model transformation language, which incorporates the output in the form of templates into the control structure. The intermediate model is explored using a depth-first strategy.

EAST-ADL models created in Papyrus have a metamodel that is a composition of several separate metamodels. In the case of EAST-ADL, the metamodel consists of the UML metamodel and the EAST-ADL profile. These artefacts are composed by the Eclipse Framework to an EAST-ADL metamodel at runtime. The EAST-ADL metamodel corresponding to a Papyrus EAST-ADL model is not a separate artefact, and this complicates the model transformation and limits the choice of model transformation engines.
Figure 24 - Intermediate HiP-HOPS model

The intermediate model conforms to the HiP-HOPS ecore metamodel. It is aligned to the HiP-HOPS grammar. It also conforms to Ecore, and thus processable with the Eclipse Modelling Framework (EMF).
4.2 Extensions to the HiP-HOPS Safety Analysis Engine

The EAST-ADL safety analysis plugin creates an interface between EAST-ADL and HiP-HOPS that allows an EAST-ADL model to be analysed by HiP-HOPS. However, this interface relies upon a certain commonality of concepts between EAST-ADL and HiP-HOPS, which requires a degree of harmonisation. In addition, the extra modelling capabilities of EAST-ADL makes it possible to represent a wider range of safety-related information than traditionally represented in HiP-HOPS, and thus to take fuller advantage of this information, HiP-HOPS also needs extending with new capabilities.

These processes are described below.

4.2.1 Harmonisation of EAST-ADL and HiP-HOPS

As EAST-ADL has evolved, it has incorporated information necessary to support a safety driven model-based design process. In particular, it has been designed to be able to support the upcoming ISO26262 automotive standard, which sets out a methodology for the incorporation of safety in the design of an automotive system. But EAST-ADL is only a modelling language – to use the safety information contained within an EAST-ADL model, an external tool is needed. HiP-HOPS is one such tool – a recently developed safety analysis & optimisation engine [4] designed to support model-based safety analysis techniques. However, HiP-HOPS was not originally designed to be compatible with EAST-ADL or ISO26262, and assumes a somewhat different modelling approach. To ensure that EAST-ADL and HiP-HOPS can work together, the safety-related concepts in both need to be harmonised.

As explained earlier, EAST-ADL’s primary means of representing safety information in a model is via its Error Model. The Error Model is designed to be a separate architecture, parallel to – but not necessarily with a one-to-one mapping to – the nominal architecture. By contrast, HiP-HOPS uses a hierarchical model that represents both nominal architecture and failure behavior together. This is the first major difference with HiP-HOPS. EAST-ADL's Error Model allows a great deal of freedom when modelling the failure behavior of a system, whereas HiP-HOPS uses a much more constrained model; although the unconstrained Error Model can in theory be converted to an equivalent HiP-HOPS model, HiP-HOPS expects that model to represent the structure of the nominal system architecture, not just a ‘virtual’ error architecture. Though this does not necessarily preclude a successful analysis, it does require a certain degree of careful interpretation of the results, since the components that HiP-HOPS refers to are really entities in the EAST-ADL error model, and not nominal entities. This issue takes on much greater importance when optimisation is being applied, in which case the EAST-ADL Error Model must be constrained to a one-to-one mapping with the corresponding nominal model for the optimisation to make sense. This has required the development of methodological guidelines and the provision of superimposed stereotypes to allow nominal and error models to share the same architecture for the purposes of generating models that can be optimised by HiP-HOPS.

The modelling concepts of HiP-HOPS have also informed the design and development of the EAST-ADL Error Model itself. HiP-HOPS was designed with a compositional model that allows hierarchies of subsystems and subcomponents to be built up, all with ports (interfaces to the rest of the system) and interconnecting lines (connections between ports). Each component is defined with data describing its local failure behavior; in particular, ‘failure logic’ is defined that relates failures at a component's outputs ('output deviations') to a combination of internal failure modes ('basic events') and failures received at the component's inputs ('input deviations').

As the plugin's concept mapping table shown earlier indicates, many of these concepts (or their equivalents) have found their way into EAST-ADL. The failure logic is contained in EAST-ADL's ErrorBehavior entities, while ErrorModelTypes and ErrorModelPrototypes build up a hierarchy in the same way HiP-HOPS uses subsystems and components. FaultFailurePorts behave similarly to
HiP-HOPS ports and their associated output/input deviations, and are connected by error propagation links analogous to HiP-HOPS lines. InternalFaults represent internal failure modes.

Thus the development of the Error Model in EAST-ADL has evolved with an improved level of support for HiP-HOPS concepts in mind, particularly with regard to system failure propagation. However, HiP-HOPS is only one possible tool and so care has been taken to ensure that EAST-ADL safety concepts are not limited to HiP-HOPS; indeed, in theory the Error Model should also be transformable to other compositional safety analysis techniques or even more formal techniques. In some cases this has led to complications in the translation to HiP-HOPS (e.g. in the requirement for a one-to-one mapping in optimisation), but the benefit is a greater degree of flexibility and potentially a wider range of supported tools.

### 4.2.2 Extensions to HiP-HOPS

Although the safety analysis concepts from HiP-HOPS have influenced the development of the Error Model, resulting in a closer harmonisation between the two, there are some concepts found in EAST-ADL – particularly in terms of new support for ISO26262 processes – that have no equivalent in HiP-HOPS. Thus to support these concepts, HiP-HOPS has also needed extending with new capabilities. The two major extensions are to support multi-perspective analysis and the decomposition of safety requirements.

#### 4.2.2.1 Multi-Perspective Analysis

As explained above, EAST-ADL does not limit the modeller to a single architecture, as HiP-HOPS does. Not only are the Error Model and nominal model separate, but EAST-ADL provides multiple layers or levels of potential modelling as well as different views or perspectives of a model. HiP-HOPS is expected to be applied to EAST-ADL models at the analysis/artefact and design levels, and at these levels EAST-ADL offers a number of different perspectives. At the analysis level, the primary perspective is the Functional Analysis Architecture (FAA), which is a relatively abstract view of the functions of the system. At the design level, the model is more concrete and is separated into the Functional Design Architecture (FDA) or software perspective and the Hardware Design Architecture (HDA) or hardware perspective. There is also the possibility of a middleware perspective at the design level.

Although HiP-HOPS is designed to be able to support iterative analyses at different levels of abstraction, and thus can cope equally well with an initial qualitative analysis of an abstract functional model and a later quantitative analysis of a more detailed component model, it was only designed to support a single perspective. The concept of separating software and hardware perspectives of the same system is without analogue in HiP-HOPS. This is because HiP-HOPS assumes a compositional model that combines hardware and software, with individual software functions (or subsystems of functions) contained within the hardware processors that run them. Therefore, although HiP-HOPS is capable of analysing both software and hardware, it requires them to be in the same model, with the hardware the primary means of failure propagation: it is not possible for failures in one software function to propagate directly to another remote, physically separated software function; instead software failures must propagate up to their parent hardware component and then propagate along hardware connections to other hardware components that carry the affected functions.

This can be emulated in EAST-ADL by combining everything into a single perspective, but the advantages of having multiple perspectives are then lost. Instead, HiP-HOPS is being extended to provide native support for multi-perspective analysis: the analysis of failure behavior across two or more separate perspectives of the same system (e.g. H/W and S/W). This requires the introduction of the concept of a 'perspective' to the HiP-HOPS model hierarchy.
In EAST-ADL, hardware and software entities are in separate architectures (HDA and FDA) and communication from one perspective to the other is accomplished by means of allocation relationships: software functions are allocated to hardware components that execute them. Failures of the hardware components will propagate to the software functions allocated to them. Failures can also propagate from H/W to H/W and also from S/W to S/W, but this form of connection is restricted to only H/W → S/W propagations. This avoids problems inherent in the multi-perspective modelling paradigm where failures can initiate intractable circular logic loops in when propagating through the system (e.g. a S/W fault could cause a H/W failure that in turn is caused by the same S/W fault occurring).

HiP-HOPS has been extended with a similar approach, as shown in Figure 25. Although the original single-perspective approach is still possible, when analysing FDA/HDA models from EAST-ADL, the new multi-perspective representation can be used. To accomplish this, a new ‘Perspective’ layer in the model hierarchy was added:

- **Model** (top-level entity representing an entire system under analysis)
- **Perspective** (contains a different perspective of the entire system, e.g. S/W, H/W)
- **Subsystem** (subsystem containing one or more components)
- **Component** (component representing some H/W or S/W entity)
- **Implementation** (the failure behavior of a component; may also contain a subsystem)

Any number of perspectives can be used, but one perspective must be the ‘primary’ perspective. Since EAST-ADL assumes that the FDA is the primary perspective, and restricts failures to propagating from H/W to S/W but not from S/W to H/W, it is assumed that an exported EAST-ADL model will use the software perspective as the primary perspective in HiP-HOPS too.

As in EAST-ADL, HiP-HOPS also provides means for failures to propagate from one perspective to another. As in EAST-ADL, it is possible to define ‘allocations’ of components in one perspective.
to components in another perspective. Failures can propagate along these allocations by means of 'External Propagation' constructs. In fact, in both EAST-ADL and in HiP-HOPS, it is possible to define more than one possible allocation, e.g. using variability constructs in EAST-ADL. This allows scope for potential optimisation of the model on the basis of changing the allocation of software functions to different hardware components, and indeed the optimisation capabilities of HiP-HOPS are also being extended in that direction.

HiP-HOPS provides for certain types of 'local failure data'. Traditionally this has included definitions for Basic Events (representing internal failure modes), Output Deviations (representing the propagation of failures at component outputs), and Common Cause Failures (linked to common model-level failures). Each, once declared, is visible to other definitions in that component, e.g. a basic event can be used in an output deviation of the same component.

Due to multi-perspective analysis, HiP-HOPS now also supports the definition of 'External Propagation' failure data. These are equivalent to Output Deviations, but rather than representing the propagation of failure from the component outputs, they represent the propagation of failure across allocation relationships. Also, unlike other failure data types, External Propagations are visible to allocated components as well as the local component. This means that they can be used in the definition Output Deviations of allocated components.

For example, consider CPU2 and F2 in Figure 25 above. We may define the local failure data of CPU2 to include:

- Basic event "CPU Failure", optionally with appropriate failure rates etc.
- Output Deviation "Omission-output", which is caused by either "Omission-input" or "CPU Failure" (defined above) or "EMI" (defined below).
- Potential Common Cause "EMI", which is caused by electromagnetic interference affecting the entire system.

In addition, because we know that CPU2 can have software functions allocated to it, we define an External Propagation like so:

- External Propagation "Omission", which is caused by "Omission-input", "CPU Failure", or "EMI".

Then when we define the local failure data for F2, which is a software function allocated to CPU2, we can reference this external propagation in F2's output deviation:

- Basic event "Undetected bug"
- Output Deviation "Omission-fout", caused by "Omission-fin" or "Omission". The external propagation is referred to more explicitly as "FromAllocation(Omission)".

Because any component can only be allocated to one component at a time (it has at most one 'current allocation'), we know that "Omission" must refer to an external propagation found in CPU2. HiP-HOPS can therefore link this "Omission" in F2 to the failure logic for "Omission" in CPU2 during its fault tree synthesis process, connecting the perspectives together and allowing for propagation of failure from hardware to software.
Note also that the semantics of common cause failures have been changed too; whereas before common cause failures (CCFs) were globally visible throughout the model, now CCFs are defined per perspective, thus a CCF defined for the hardware perspective (e.g. flooding, fire) would not be accessible from a component in the software perspective, for instance.

HiP-HOPS also allows a more flexible form of connection by means of the 'LocalGoto' and 'GlobalGoto' declarations. These support a more abstract type of connection for situations where there is no form of allocation relationship between two components in different perspectives, but propagation is nevertheless necessary. In these cases, the Gotos act like special forms of input deviations that can connect directly to another output deviation elsewhere in the model. Local gotos only connect to output deviations in the same subsystem (hence 'local') whereas Global gotos can connect to any fully-qualified output deviation in any perspective. For example:

- "O-F2.out = LocalGoto(O-F1A.out)" is valid and states that O-F2.out is caused by O-F1A.out.
- "O-F2.out = LocalGoto(O-CPU1.out)" is invalid as the output deviations are not in the same subsystem.
- "O-F2.out = GlobalGoto(O-CPU1.out)" is valid and states that O-F2.out is caused by O-CPU1.out.

However, these gotos are considered potentially harmful to a coherent analysis and should only be used sparingly, because they break the connection between the propagation of failure and the architecture of the system.

Because these new connections (allocations and gotos) connect existing HiP-HOPS constructs (namely, output deviations), they fit relatively seamlessly into the HiP-HOPS fault tree synthesis process. Once the fault trees are generated, then they can be analysed as normal without any further distinction between one perspective and another.

It is hoped that, once complete, these extensions to HiP-HOPS will enable better support for the analysis of EAST-ADL-style multi-perspective models involving separate representation of hardware and software.

### 4.2.2.2 Decomposition of Safety Requirements

One area where EAST-ADL is being developed to offer support for an ISO 26262 process is in the area of safety requirements. A hazard analysis on vehicle feature level is able to determine the possible hazards for a system and assign corresponding safety goals, each of which includes the requirement to reach a certain level of safety known as an Automotive Safety Integrity Level or ASIL. In ISO 26262, these safety requirement (i.e. ASILs) can be decomposed through the system to obtain safety requirements for those parts of the system that cause any given hazard, rather than making the entire system meet the requirement (which may not even be possible).

EAST-ADL provides the means to support the hazard analysis on vehicle level and also allows the results (in the form of ASIL requirements) to be passed down to the analysis and design levels, where they can be associated with causes in the Error Model, thus linking the propagation of individual failure modes through the model with the hazards they cause and thus the ASIL required to prevent that hazard.

However, once again, although EAST-ADL can model the relevant information, it cannot actually perform decomposition of ASILs itself. Nor can HiP-HOPS, which is was not designed with that sort of analysis. In fact there are no tools which can perform this activity, and perhaps no other language is designed to represent it. ASIL decomposition is a new concept and requires new modelling entities and tool algorithms to support it.
As a result, HiP-HOPS is being extended with new algorithms to support ASIL decomposition. The goal is to be able to decompose the safety requirement (represented by the ASIL) associated with a particular hazard and determine which safety requirements are necessary for individual failure modes to fulfil the safety requirements of the system as a whole. For example:

![Figure 26 - ASIL decomposition](image)

In this example, the modelled system as a whole has ASIL C, indicating there is a relatively severe hazard that must be avoided. Parts of the system that contribute directly to the system failure (such as the Dedicated CAN and the outputs) receive the ASIL directly too. When there is a disjunction, e.g. two or more components can all cause the system failure individually, they all receive the full ASIL. Only when components must fail together in conjunction to cause the system failure is the ASIL diluted. This can be seen in the simple case for Macro-block3 in the diagram, which collectively receives an ASIL C; the subcomponents within (ultimately SWS1-4) may not have the full ASIL C, however, and there are a number of different possible combinations of ASILs that would fulfil the ASIL C for the subsystem as a whole (e.g. A + A + A + A).

A more complex case arises with Macro-block1 and Macro-block2. Both must fail in conjunction to cause system failure, thus both together receive a share of the ASIL C. In the figure, Macro-block2 is assigned ASIL B whereas Macro-block1 is assigned ASIL A. However, both blocks have two outputs; in Macro-block1, a failure of either output is all that is necessary, whereas in Macro-block2, a failure of both outputs is needed. In Macro-block1, all components would require ASIL A since any individual failure is sufficient, but in Macro-block2, both originating contributors (NO and NC in the EPB Button subsystem) could receive ASIL A as both need to fail for Macro-block2 to fail.

In this example, there are many alternative ASIL allocation strategies that could be used. However, when there is more than one hazard and thus more than one system safety requirement, the competing demands of these multiple requirements can be used to reduce the number of possible allocations. In the more abstract example below, there is only one optimum allocation possible:
Figure 27 - A single possible ASIL allocation

Here, there are four possible hazards linked to four possible system failures, each of which has been given a different safety goal (and associated ASIL):

- **F1(O)** - Omission of output from F1, with ASIL D
- **F1(C)** - Commission of output from F1, with ASIL A
- **F2(O)** - Omission of output from F2, with ASIL C
- **F2(C)** - Commission of output from F2, with ASIL A

Due to the connecting failure logic amongst the six subcomponents in the system, it transpires that there is only one possible allocation. To see why, it is necessary to go into more detail. First, the causes of each hazard must be explored. As an example, take F1(O). Its initial cause is an omission from E1, which can be caused by an internal failure mode or by an omission of input. Since E1 is directly capable of causing the hazard, it receives ASIL D. Omission of input to E1 is caused solely by an omission from E2, which in turn is caused either by an internal failure mode or by an omission of input at both inputs. E2 therefore receives ASIL D as well. However, omission of input to E2 is caused by a conjunction of omissions from both E3 and E4, so they each contribute some share of the ASIL D. There are five possible allocations for these two components:

- **E3 = QM (0), E4 = D (4)**  Total = 4 (D)
- **E3 = QM (1), E4 = D (3)**  Total = 4 (D)
- **E3 = QM (2), E4 = D (2)**  Total = 4 (D)
- **E3 = QM (3), E4 = D (1)**  Total = 4 (D)
- **E3 = QM (4), E4 = D (0)**  Total = 4 (D)
Allocations where both E3 and E4 are more than sufficient to meet the safety requirement, e.g. E3 = ASIL C and E4 = ASIL C, are also possible, but not considered unless necessary (i.e. unless both E3 and E4 directly contribute to an ASIL C hazard somewhere).

To reduce the number of combinations, it is necessary to look at the other ASILs. It is determined through decomposition that F1(C), with ASIL A, is caused by any internal failure mode of E1, E2, E3, and E4; thus each of those failure modes must be at least ASIL A. That eliminates two possibilities for E3 and E4 (namely, where one is ASIL D and the other is QM). Next, decomposition of F2(O) – with ASIL C – shows that there are four possible causes: internal failures of E2, E4, E5, or E6. Each therefore requires a minimum of ASIL C. This removes two more possibilities from the list, leaving only one possible allocation: E3 = ASIL A, E4 = ASIL C.

In this way, ASIL decomposition allows us to determine which failure modes contribute to which hazards, and by process of elimination, determine what ASIL each failure mode should have (or, more frequently, determine a set of many possible ASIL allocations for those failure modes). The new algorithms in HiP-HOPS support this.

The first step is to ensure that the failure behavior of the system is also modelled in an Error Model. This allows HiP-HOPS to understand the routes of propagation through the system necessary to cause each hazard, each represented by a different fault tree. Fault Tree Analysis then yields the combinations of basic failure modes necessary to cause each hazard, and these are assigned ASILs accordingly. For example, if we have a fault tree with the following cut sets (combinations of failure mode causes):

- \{A, B, C\}
- \{D\}
- \{E, F\}

and the fault tree leads to a hazard with associated ASIL C, then each set receives ASIL C as well. In cases where more than one failure mode is involved (e.g. A.B.C and E.F) then multiple possible allocations to those failure modes are possible. For example, all of A, B, and C may receive ASIL A, or one of them may receive ASIL C and the others only QM. It is up to the designer to decide which possible allocation is best.

However, as previously explained, HiP-HOPS can use multiple system safety requirements to reduce the number of possible combinations by eliminating impossible ones (i.e. allocation combinations in which one or more safety requirements are unfulfilled) and ignoring ones that are unnecessarily excessive (i.e. ones that assign higher ASILs to failure modes than are required). This helps the designer to make an informed choice, although there may still be many possible allocations to choose from.

This ASIL decomposition process is designed to be applied to the FAA on the analysis level. At this stage the model is still quite abstract, and true internal failure modes may not be known, but it can be assumed that each function has at least one internal failure mode that causes any given output failure. Thus for example function X may have an omission failure mode, a commission failure mode, a value failure mode, and so forth. Since these failure modes will often directly be responsible for causing output failures of their functions, assigning an ASIL to the failure mode is synonymous with assigning an ASIL to the output failure of the function too. This is not what matters, however: what is important is that the ASIL is assigned on the basis of that component's local contribution to the system failure, and only includes a consideration of input to that component if it mitigates the failure in some way (e.g. it requires all inputs to fail in conjunction, or fails silent in response to error).

The result should be an allocation of potential ASILs to the failure modes (or more generally, the functions) of the FAA. Then on the next level of development, when the design is more mature and
an FDA has been modelled, these preliminary ASILs can be tested by conducting a full quantitative analysis on the FDA (and optionally the HDA etc too in a multi-perspective analysis). This will yield probabilistic values for failures that can be used to determine whether the originally assigned ASILs have been met; for example, guidelines may state that an unavailability of $1 \times 10^{-5}$ or less is sufficient to meet ASIL B, whereas $1 \times 10^{-6}$ is needed to meet ASIL C. If safety requirements are not being met, then the system design can be changed accordingly until they are met. Alternatively the original allocation of ASILs may be questioned, and another allocation strategy chosen instead.

This ASIL decomposition capability in both EAST-ADL and HiP-HOPS is a particularly novel and potentially very valuable capability that is thus far unique. It could prove to be a very effective means to support a safety-driven design process.
5 Conclusions

In automotive systems, safety is a critical issue: any failure of the system could result in harm being inflicted on people inside the vehicle, in another vehicle, or outside the vehicle. However, safety is often relegated to the end of the design process, at which point the system design has to be changed to ensure it meets the safety requirements imposed upon it. A more efficient way of improving the safety and reliability of a system is to ensure that it is designed with safety in mind from the start. This is known as a safety-driven design process. At all stages of the process, as the design evolves over time and becomes more technical and less abstract, safety requirements are applied and the system verified to ensure that it meets those requirements. The result is that safety becomes an integral part of the design and the eventual system is consequently much safer.

The need for safety-driven design processes is recognised internationally, and nowhere is this more evident than in the upcoming international safety standard, ISO/26262. ISO/26262 introduces a solid safety-conscious methodology into automotive design by defining how safety requirements should be introduced and applied to a system design. EAST-ADL, which supports a model-based design process, is being developed with ISO/26262 in mind and offers an excellent opportunity: if it can be extended with the appropriate safety concepts, provided with methodological guidelines based on ISO/26262, and provided with compatible tool support for analysis and verification purposes, then EAST-ADL could offer automotive designers a compelling and comprehensive solution for safety-driven model based design.

However, achieving this goal – providing comprehensive support for safety-driven design in EAST-ADL – means bringing a number of different elements together and ensuring they all work in harmony. Firstly, EAST-ADL itself must be able to provide language-level support for the type of safety concepts found in ISO/26262. In practice, this means extending the metamodel with appropriate language entities, such as hazards, errors, and safety requirements. Secondly, a methodology has to be developed to ensure that these language entities are used correctly and that the design process fulfils the standard defined by ISO/26262. Thirdly, EAST-ADL needs compatible tool support to provide analysis and verification capabilities. Safety analysis tool support in ATESTST2 is provided by the HiP-HOPS safety analysis engine, and so it is necessary to ensure that EAST-ADL models can be converted into a format that can be understood HiP-HOPS (and potentially other external safety tools) and also means extending HiP-HOPS itself to support the new safety analysis techniques necessary to fulfil ISO/26262.

Together, these elements make it possible for EAST-ADL to provide support for safety analysis techniques such as hazard analysis, fault trees, and FMEAs, as well as more advanced new forms of safety processes, such as the decomposition of safety requirements, multi-perspective error propagation analysis, and even safety-driven design optimisation.

5.1 EAST-ADL Language support for Safety

In order to be used as the basis of a safety-driven design process, EAST-ADL itself must provide native support for representing safety concepts such as hazards, safety requirements, and error propagation. In EAST-ADL, this support is provided by the error model. The existing error model developed in ATESTST1 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 propagation via the allocation relationship.

At the centre of the error model are the ErrorModelType and ErrorModelPrototype classes, analogous to the ADLFunctionType and ADLFunctionPrototype 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 behavior of a system, subsystem, or function in the EAST-ADL nominal model. The error model is hierarchical, and as such, any system, subsystem, function, or any ADLEntity may be described by its own error model.

The actual behavior 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 behavior of a function or other entity can be described in a tool-specific way. For HiP-HOPS, the ExternalErrorBehavior 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, represented in EAST-ADL by an ADLDesignDataType. 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.

It is important to note that the EAST-ADL error model is designed to be flexible and not limited to any particular external tool (e.g. HiP-HOPS). As a result, the error model is not specifically intended to have a 1:1 mapping between nominal architecture (e.g. ADLFunctionTypes) and the error model entities (e.g. ErrorModelTypes). This is in order to provide the maximum degree of freedom when modelling the error behavior of a system. However, it also poses problems for the translation of an EAST-ADL model to external tools – like HiP-HOPS – which assume that the nominal architecture and the error model have a 1:1 mapping. In the case of HiP-HOPS, the tool assumes that failures occur in components which have their own error model and propagate through connections in the nominal architecture; this makes a compositional error modelling approach possible in which architectural transformations that involve substitution of components and subsystems together with their error models can lead to valid and analysable system error models. In HiP-HOPS this feature enables architectural optimisation to be performed automatically by a genetic algorithm. EAST-ADL does not impose such mapping restrictions at the language level.

After a series of investigations on how best to resolve this difference in a way that the benefits of both approaches could be achieved, three different modelling styles were identified, each of which could be imposed through methodological guidelines:

- **Error Modelling with Self-Contained Monolithic Error Descriptions.** This is the standard style, which does not assume any 1:1 mapping; instead the error model is self contained. In this case, any HiP-HOPS analysis would only follow the error model and no correspondence with the underlying architectural model could be assumed. In this case, any form of automatic optimisation would not be possible as any changes would be relevant only to the error model, not the nominal model, and it may be difficult to accurately decompose ASILs.

- **Integrated Error Modelling with Superimposed Stereotypes but Separated Topologies.** This is a hybrid style which does assume a 1:1 mapping. The mapping is achieved by extending the definition of the ADLEntities with an error definition (ErrorModelType). In this way, the error model follows the hierarchy of the nominal structural architecture and there is better alignment between any nominal variability and variability in the error model. However, propagations between entities still differ between error model and nominal model and therefore still poses a significant obstacle to any optimisation or even propagation analysis.
Integrated Error Modelling with Superimposed Stereotypes with Integrated Topologies. This is the style closest to the compositional structural hierarchy used in tools like HiP-HOPS. In this modelling style, not only do nominal and error entities have a 1:1 mapping, but so do the connections between them: failures are assumed to propagate by means of nominal connections. In this case, nominal structural ports also serve as FaultFailurePorts. Although this approach can create additional modelling complexities, it harmonises the error model and nominal model and results in much better compatibility with external tools like HiP-HOPS. Due to the full 1:1 mapping, it also allows external tools to make architectural changes (e.g. for optimisation purposes) on the basis of an exported error model and resultant safety analysis.

Although standard safety analysis should be possible using any approach, the first two approaches make it harder to relate any such results to the nominal architectural model, and more severely limit the potential to apply more advanced techniques such as safety-driven optimisation, multi-perspective analysis, and ASIL decomposition. As a result, it has been agreed that the third style should be used when any HiP-HOPS analysis is required.

So far it has been assumed that external safety analyses will be applied primarily at the analysis (i.e. FAA) and design (i.e. FDA/HDA) levels, while the preliminary hazard identification and assignment of safety requirements would take place internally at the vehicle level (i.e. VFM). However, at the design level, there can be more than one view of the system, reflecting the hardware, software, or even middleware aspects of the system. Each will typically have their own error model.

Clearly, this presents a substantial problem – how do we capture the full failure behavior of a system, and thus perform a complete analysis, when the description of that behavior is distributed across multiple error models? The answer is to perform a multi-perspective analysis, in which propagations from one perspective to another are possible by means of allocation relationships. After a lengthy investigation, it was decided that the software/functional model should be the ‘primary’ model – all failures should ultimately propagate to this model. Contributions from other perspectives, e.g. hardware errors such as power failures etc, should propagate to the functional model by means of the allocations, which indicate how the functions are distributed across hardware. For example, a H/W processor may be host to a number of allocated S/W functions, each of which is ultimately dependent on the power supply to the processor on the H/W layer. This cross-perspective propagation makes it possible to take into account failures arising in any perspective.

However, to limit the potential for any ambiguities or contradictions to arise, propagations are unidirectional, i.e. failures only propagate from a secondary perspective to the primary perspective: H/W failures may lead to S/W failures, but any S/W failures that may cause H/W failures are not modelled in this approach.

Error model support for multi-perspective analysis is provided by the ‘allocationConstraint’. This links entities in one perspective (e.g. FDA) to entities in another (e.g. HDA). 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.

5.2 EAST-ADL Methodology support for Safety

It is not enough for EAST-ADL to provide means of representing safety concepts in the language; it must also provide guidance for how it is to be used correctly. Because the goal is for EAST-ADL to support ISO26262, much of the guidance will originate from that standard, which defines a safety workflow designed to ensure that safety is fully incorporated as part of an automotive system design. In particular, ISO26262:
• provides an automotive safety lifecycle (management, development, production, operation, service, decommissioning) and support for tailoring the necessary activities during these lifecycle phases;
• provides an automotive specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs);
• uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk; and
• provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety being achieved.

The safety workflow and how it relates to EAST-ADL and its tool support can be briefly summarised as follows:

1. **System Functional Analysis.** The first step of the safety lifecycle is to identify and describe the 'Item' being considered for safety analysis, and to develop an adequate understanding of it. This is an essential step, since the subsequent phases of safety design flow are based on the item definition and the safety concept is derived from it. In practice, this involves generating a number of use cases and data flow diagrams to determine the item's purpose and interaction with the rest of the system or system environment.

2. **System-Vehicle Interaction Analysis.** The second step consists of analysis how the item interacts with the vehicle. This necessitates the definition of behavioral models (e.g. state machines, data flow diagrams) for behavioral and impact analysis purposes. This helps ensure that the system behavior is captured from different perspectives.

3. **System boundary definition.** It is important for the boundary of the system to be properly defined if we are to be able to understand how it interacts with its environment. Functions of the item can be classified according to whether they are safety critical or not; the safety critical functions then comprise the Subsystem Under Safety Analysis (SUSA).

4. **Scenarios Definition.** In order to perform a correct hazard analysis, a proper set of realistic operative scenarios must be defined. A complete description of a scenario includes all variables and/or states that characterise the functions or affect them. This includes both operative conditions and environmental conditions.

5. **Hazard Identification.** Next, it is necessary to identify the hazards. To do this, we must define the malfunctions, the misuse(s) and eventual maintenance condition(s) related to the item. Malfunctions can be enumerated for each target function, e.g. unwanted activation, missed activation etc; misuse and maintenance conditions are then defined for periods when the item is in use or undergoing maintenance, though deliberate misuse is typically excluded. A hazard is then the combination of a malfunction occurring in a given scenario, i.e. the effect of a malfunction in specific operating and environmental conditions. Hazard identification is performed on the vehicle feature level.

6. **Risk Analysis.** Once the hazards have been identified, they each need to be classified according to their Automotive Safety Integrity Level (ASIL), which estimates the risk level associated with the item. The ASIL is determined on the basis of the controllability, severity, and exposure of each hazard. The result is an ASIL class from A (lowest) to D (highest), or QM if no special safety requirements are needed (i.e. the given function is not safety relevant).

7. **Safety Goal and Safe State Definition.** Unless the item is not safety relevant (i.e. it has no ASIL above QM level), then the safety goals for each hazardous event should be defined. These are the top level safety requirements for the item, and aids in the definition of item characteristics necessary to avert hazards and reduce the risk associated with the item down to an acceptable level. ASILs are assigned to safety goals to ensure that the item will meet those goals. Furthermore, for each safety goal, a safe state is defined that specifies what state the system should be in (or should enter) in the event of a failure, to allow any mitigation or other corrective action to take place and ensure the safety goal is not violated.
8. **Risk Assessment Approval.** The estimated controllability values assigned to the various situations also needs to be validated. Therefore, specific testing should be carried out for the final value assessment – in other words, test drives should be performed by injecting faults into a vehicle and determining the results of the failures and their effect on the drivers' opinions and reactions.

9. **Functional safety requirements definition.** At this stage, for each safety goal and safe state, safety requirements should be specified. These are the top-level functional safety requirements. To achieve this, it is necessary to perform qualitative safety analyses such as FMEA and FTA to determine the causes of failures (particularly any common causes or single points of failure). This helps us understand which parts of the system are safety critical and which do not contribute to system failure. In an EAST-ADL context, the FTA and FMEA would be conducted by external tools such as HiP-HOPS and thus requires an error model to have been constructed for the system FAA design at the analysis level. Such an error model need not necessarily include specific quantitative failure information; it may instead assume more abstract failures, e.g. a generic failure of each function of each generic failure type (omission, commission etc).

10. **Technical safety requirements definition.** Once the functional safety requirements have been defined, the item can be developed further by specifying the technical safety requirements. These describe how to implement the safety measures described by the functional safety requirements, or in other words, what the safety requirements must be for each element of the detailed technical architecture if design is fulfill the safety requirements for the system as a whole. The process of defining the technical safety requirements is an iterative process that involves both ASIL decomposition (which can be accomplished in EAST-ADL through its interface with HiP-HOPS) and criticality analysis (including further assignment of ASILs). ASIL decomposition determines what contribution each element of a system has to a given system-level safety requirement and determines an ASIL on that basis; an element that is solely responsible for contributing to a functional safety requirement will inherit the full ASIL from that requirement, whereas a set of elements that together deliver a functional safety requirement may have the ASIL of the functional safety requirement divided amongst them. During criticality analysis, it is possible to reduce the ASIL inherited by some internal system element by determining the potential of that element to violate a safety goal.

11. **Detailed Safety Analysis.** Once the safety requirements (both technical and functional) have been assigned, it is valuable to determine whether the system design is likely to meet those goals. This can be accomplished by means of a more detailed quantitative safety analysis, e.g. FTA, which provides estimates for the probability of system failures. If severity information is also provided, then more detailed criticality estimates can also be produced, e.g. in an FMECA. This can also be accomplished in EAST-ADL by means of external tools such as HiP-HOPS. At this stage, the process may also involve a multi-perspective analysis.

12. **Potential safety-based optimisation.** If the safety requirements have not been met, as indicated by the detailed safety analysis, then the design likely needs revising. One option is to perform some kind of automated safety-drive optimisation. Although optimisation is not within the scope of this document, it would inevitably involve external safety analysis tools (like HiP-HOPS) and thus require error modelling and potentially multi-perspective analysis etc.

The above is only a summary of the full safety workflow methodology, but serves to demonstrate how it is important that language support for safety concepts is accompanied by a well-defined process for employing those concepts if best use of them is to be made. By following the above stages, it is hoped that a system design will be compatible with the standards set out in ISO°26262 and ultimately result in a safer, more reliable system.
5.3 EAST-ADL Tool support for Safety

EAST-ADL is only a language; it can provide means of representing safety-related concepts, and can even provide guidance on their usage during the design process, but without tool support, it cannot provide any way of using those concepts as part of any safety analyses. The primary tool chosen to provide analysis support to EAST-ADL is HiP-HOPS. Connection to HiP-HOPS is achieved by means of an Eclipse-based plugin to Papyrus, EAST-ADL's modelling environment, enabling Papyrus to be able to communicate with the external HiP-HOPS executable by translating an EAST-ADL model into a HiP-HOPS-compatible format. This then makes it possible for HiP-HOPS to perform a variety of analyses on the model, from standard FTA and FMEA to automated ASIL decomposition, multi-perspective FTA, or even automated safety-driven optimisation.

The conversion from EAST-ADL to HiP-HOPS is accomplished by means of model transformation technology. This ensures a separation of concerns between model transformation and model representation, facilitates easy parsing and traceability, and results in improved maintainability over a custom-built converter utility (which was the solution used in ATESST1). Grammars for external file formats (HiP-HOPS, in this case) can be defined using ecore and then the mapping of EAST-ADL concepts to HiP-HOPS concepts, and the transformation of one model into the other, is defined using the ATL model transformation language. The result is a robust and readily extensible interface between EAST-ADL's error model and the HiP-HOPS safety analysis engine.

However, HiP-HOPS itself was not designed with EAST-ADL or ISO26262 in mind; HiP-HOPS is based on the annotation of compositional system models with failure behavior, allowing for the automatic synthesis and analysis of fault trees and FMEAs. It was later extended with new capabilities for automatic optimisation of system models with respect to safety, reliability, and cost (and optionally other attributes too).

To support the advanced concepts found in EAST-ADL and ISO26262, HiP-HOPS has been extended with new algorithms for multi-perspective analysis and ASIL decomposition. Taking the multi-perspective analysis a step further, its optimisation capabilities are also being extended to support concepts introduced by EAST-ADL, e.g. the possibility of optimising not only on the basis of fault-tolerance schemes involving component replication or replacement of components with more reliable versions, but also on the basis of different allocation schemes.

Multi-perspective analysis support in HiP-HOPS is provided by means of the new 'Perspective' level in its system model hierarchy. The Perspective level is immediately below the Model level, meaning that a system model can be composed of a number of different perspectives, each of which may contain its own components and subsystems and thus its own failure behavior descriptions. Connections between perspectives are achieved, as in EAST-ADL, by means of allocations of components from one perspective to components in another perspective. For maximum flexibility, HiP-HOPS also provides for the possibility of direct propagations between perspectives without requiring allocation relationships; this may be useful for modelling failure transfers that occur as a result of other propagation vectors, e.g. proximity or some other form of common cause. HiP-HOPS's fault tree synthesis algorithm has consequently been extended to construct fault trees that cross perspective boundaries, thus modelling the propagation of failures throughout the entire system and not merely within the architecture modelled by each perspective. It is hoped that this will provide much better support for the multiple perspective modelling found in ADLs such as EAST-ADL.

ASIL decomposition is another new addition to HiP-HOPS. Unlike multi-perspective analysis, which is primarily an extension of existing algorithms, the ASIL decomposition algorithm involves introducing entirely new capabilities and concepts to HiP-HOPS. To support ASIL decomposition, HiP-HOPS has been extended to represent the concepts of system-level Hazards (which are caused by some combination of system-level failures – approximately equivalent to the 'HazardCause' of the EAST-ADL error model) as well as Safety Requirements, i.e. ASILs, that can be assigned to Hazards and then inherited by those subsystems, components, or even failure modes that contribute to causing those Hazards.
ASIL decomposition in HiP-HOPS relies upon the hierarchical fault propagation model built up during the fault tree synthesis process (and thus can only be performed on a correctly annotated system model). This tells HiP-HOPS which failure modes (and in turn, which components or systems) contribute to which hazards. In practice this is achieved by performing a fault tree analysis; those failure modes that collectively contribute to a given hazard share the safety requirement of preventing that hazard; those failure modes that directly contribute to a given hazard receive the full burden of preventing that hazard. In practice, this results in a large set of possible required ASIL assignments for every failure mode or component in the system. It is then up to the designer to evaluate these choices and determine which is the best (and most efficient) choice for the system design. However, the advantage is that each of the possible component ASIL assignments will already have been calculated to meet the system-level safety requirements as defined by the system hazards. If necessary, the design's ability to meet these ASIL assignments could later be verified by performing a standard quantitative FTA/FMEA. This will produce estimates of the probability of system failure; if those probabilities are within the limits set by the safety requirements, then it helps ensure that the system design will meet its safety requirements.

HiP-HOPS therefore provides EAST-ADL with a set of valuable capabilities for performing various different analyses appropriate to different stages of the design process. At early functional stages of the design, e.g. an FAA, HiP-HOPS can perform simple qualitative analyses (FTA, FMEA) to help provide an understanding of the global failure behavior of the system; it can also perform an ASIL decomposition at this stage to help determine what the safety requirements for the system's constituent functions should be. At the design stage, e.g. for FDA and HDA models, HiP-HOPS can also perform more detailed quantitative and multi-perspective analyses, which can help verify that previously assignment safety requirements are still being met. If necessary, additional ASIL decomposition could also be performed at this stage too.

The capability for automatic ASIL allocation is a particularly novel result of ATESST2. It effectively establishes a way of allocating the overall safety requirements of a system to components of the design in the course of the evolution of that design. In practice, this ability helps to enable a mode of design in which safety becomes a controlled property as opposed to a property that is allowed to emerge (or not) at the end. The allocation algorithm was conceived in ATESST2 and, to the best of our knowledge, is the first automatic ASIL allocation algorithm to be proposed. EAST-ADL and HiP-HOPS will therefore respectively be the first language and tool to support this capability.

The parallel development of HiP-HOPS and EAST-ADL during ATESST2 has meant that the two have been increasingly harmonised at the conceptual level. This ensures that each is compatible with the other and can work together so that EAST-ADL models can be analysed by HiP-HOPS (thanks in no small part to the Papyrus plugin). It has also been the case that concepts from each one have led to valuable updates and improvements in the other, e.g. improved support for error propagation in EAST-ADL and multi-perspective analysis support in HiP-HOPS. This kind of comprehensive safety analysis tool support helps ensure that EAST-ADL is capable of being used as part of a safety-driven model-based design process.

5.4 Safety-driven design support using EAST-ADL

The work described in this document – the extensions to the EAST-ADL metamodel, the inclusion of an ISO26262 compatible methodology, the interface with HiP-HOPS's safety analysis capabilities – has meant that EAST-ADL now has comprehensive support for safety analysis and safety-based design in general. In its error model, EAST-ADL provides the means to model the occurrence and propagation of failures from one function to another as well as from one perspective or architecture to another; the updates to HiP-HOPS and the creation of a Papyrus plugin ensures that this error model is not simply an isolated representation but can also be used as the basis of a number of safety analysis techniques throughout the design process, ranging
from simple qualitative FTA or FMEA of the FAA at the analysis level of EAST-ADL to full quantitative FTA and FMEA with multi-perspective support for the FDA/HDA models at the design level of EAST-ADL.

Because the concepts and methodology of EAST-ADL’s support for safety has been heavily influenced by ISO26262, it also ensures that EAST-ADL design model should be compatible with the requirements, processes, and ideals set out in that standard. Support for new processes introduced by ISO26262, including vehicle-level hazard analysis, the definition of safety goals and safety requirements, and ASIL decomposition through the error model, is now present in EAST-ADL. Through its interface with HiP-HOPS, it is also possible to automate some of these processes, e.g. ASIL decomposition.

Furthermore, by ensuring that EAST-ADL has a solid foundation of safety-based representation and analysis, it is in a good position to be extended in the future to support new, even more advanced safety analysis and design techniques whenever they arise.

Ultimately, with the unique new capabilities introduced by extending EAST-ADL with additional safety concepts, providing ISO26262 based methodology guidelines, and ensuring compatible analysis tools support by means of the HiP-HOPS safety analysis engine, it is hoped that EAST-ADL can serve as a comprehensive solution for the design safety-critical automotive systems.
6 Acknowledgements

We would like to thank David Parker, Septavera Sharvia and Nidhal Mahmud for their contribution to the implementation of the extensions to HiP-HOPS described in section 4.2.
7 References


