Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems

Matthias Biehl  Chen Dejiu  Martin Törngren
Embedded Control Systems, Royal Institute of Technology (KTH), Stockholm, Sweden
{biehl,chen,martin}@md.kth.se

Abstract
The automotive industry has a growing demand for the seamless integration of safety analysis tools into the model-based development toolchain for embedded systems. This requires translating concepts of the automotive domain to the safety domain. We automate such a translation between the automotive architecture description language EAST-ADL2 and the safety analysis tool HiP-HOPS by using model transformations and by leveraging the advantages of different model transformation techniques. Through this integration, the analysis can be conducted early in the development process, when the system can be redesigned to fulfill safety goals with relatively low effort and cost.

Categories and Subject Descriptors D.2.6 [Programming Environments]: Integrated Environments

General Terms Design, Reliability, Languages

Keywords Safety Analysis, Model-based Development, Architecture Description Language, Tool Integration

1. Introduction
An increasing number of industrial systems are provided with new functionalities and enhanced performance enabled by embedded systems. In many cases, the corresponding applications are increasingly safety related, exemplified by automotive driver assistance systems and industrial robots operating without safety barriers. Facing the challenge of developing increasingly advanced safety-critical systems, the automotive industry has a growing demand for the seamless integration of safety analysis tools into the model-based development toolchain for embedded systems. Such an integrated solution will allow iterative and incremental development of safety critical systems and is a step towards fulfilling the demands of the upcoming standard for safety-critical road vehicles, ISO-CD-26262.

Safety is a cross-cutting system property that has to be considered from the start and throughout the development of the system. Safety engineering is an iterative process. It starts with determining safety-critical aspects, proceeds with identifying the causes of failures and deriving the safety requirements and concludes with developing safety solutions.

Integrating safety analysis into the development of automotive embedded systems requires translating concepts of the automotive domain to the generic safety and error analysis domain. We assume a model based development process where automotive concepts are represented by the EAST-ADL2 architecture description language, which supports system design on multiple levels of abstraction. The concepts of the error analysis domain are represented by the safety analysis tool HiP-HOPS.

We automate the translation from EAST-ADL2 to HiP-HOPS by using model transformations. We leverage the advantages of different model transformation techniques by decomposing the translation into two distinct phases, and using an appropriate technique for each phase: A phase for conceptual mapping between the domains followed by a phase for representing the output in the desired concrete syntax.

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

The remainder of this work is organized as follows. Section 2 introduces the technology we use in our approach. Section 3 explains our approach for safety analysis tool integration. Section 4 focuses on the model transformation of the approach. By means of a comprehensible case study we demonstrate in section 5 how the integrated safety analysis can be used. We present related work in section 6 before we conclude in section 7.

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

2.1 Model Transformations
Model transformations play a key role in model-based software development. Model transformations describe the relationship between models, more specifically the mapping of information from one model to another one. These model transformation descriptions are interpreted by a model transformation engine. The model transformation engine produces the output model based on the transformation description and information from the input model. A model transformation involves two models: a source model and a target model, where source and target model can have the same or different metamodels. Model transformations use concepts of the metamodels in their descriptions. Thus they are general enough to describe the mapping for any model specified with the same metamodel. Model transformations are also used in generative programming [3], where they are called generators.
In the following we introduce a classification scheme for model transformations.

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

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

We can differentiate model transformations according to the metamodels used in the transformation. **Endogenous transformations** map between the same metamodel. **Exogenous transformations** map between different metamodels.

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

### 2.1.1 Model Transformation Engine openArchitectureWare (OAW)

OAW integrates a number of tools for model transformations into a coherent framework [6]. Among other tools, the OAW provides a workflow specification language and the transformation language Xpand. The workflow language is used to control the transformation process and to specify the sequence of transformations between the different models. The Xpand transformation language is a template-based, imperative language for model-to-text transformations. OAW is distributed as a plugin of the Eclipse platform and is able to process models that are conform to the EMF (Eclipse Modeling Framework).

### 2.1.2 Model Transformation Engine ATL

The ATLAS Transformation Language (ATL) is a hybrid model transformation language [7]. It includes both declarative and imperative constructs and supports both programming styles. However, the preferred style is declarative, which allows a cleaner and simpler implementation for simple mappings. However, imperative constructs are provided so that some mappings that are too complex to be handled declaratively can still be specified. An ATL transformation program is composed of rules that describe how to create and initialize the elements of the target models. The language is specified both as a metamodel and as a textual concrete syntax.

ATL is integrated in the Eclipse development environment and can handle models based on EMF. In this project we chose ATL for its ability to process UML models which are annotated with the UML extension mechanism for profiles, so we are able to process models that are conform to the EAST-ADL2 profile.

### 2.2 EAST-ADL2

EAST-ADL2 is an architecture description language for the development of automotive embedded systems [2]. It can be used to describe hardware (electronics), software and the environment (mechanics) of an embedded system. The goals of modeling with EAST-ADL2 are to handle complexity and improve safety, reliability, cost, and development efficiency through model-based development. A primary feature of EAST-ADL2 is its capability to structure a model into different abstraction levels. All these levels describe the same system, but on different levels of abstraction and from different viewpoints. Each level is associated with a different stage of the development process. EAST-ADL2 is an information model, connecting different views of the system. The views are influenced by the different engineering traditions and backgrounds in automotive industry. Control engineers focus on the functional view, the design view is often preferred by software engineers. This concept allows functional decomposition, supports analysis activities, design activities, implementation of software and hardware components and variability management.

EAST-ADL2 specifies a domain model, which is implemented as a UML profile depending on UML and SysML. Using the EAST-ADL2 profile for UML, it is possible to create EAST-ADL2 models in any UML design tool, i.e. the Eclipse-based Papyrus UML tool.

### 2.3 HiP-HOPS

Creating the FMEA (Failure Modes and Effects Analysis) and the FTA (Fault Tree Analysis) by hand is a very laborious and error-prone task, hindering the safety design process. However, safety considerations should be built into the design right from the start and an iterative safety analysis needs to be performed during the design. HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) can support such an iterative safety design by automating FTA and FMEA and even combining the results. This analysis data can also be the basis for an optimization of the security and reliability of the system. HiP-HOPS expects a model describing the topology of the system (components and their subcomponents) including information about how individual components can fail as well as how failures are propagated. Among other functionalities, HiP-HOPS creates local fault trees, combines them to a system fault tree and calculates a minimum cutset [9].

### 3. Tool Integration

To establish the link between EAST-ADL2 system modeling tools and safety analysis tools, they have to be integrated. In his seminal work, Wasserman identifies five different aspects of tool integration [14]:

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

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

Process integration cannot be done by software itself, but depends on personal preferences, company culture and development organization. Automation of safety analysis has several advantages: It makes safety analysis easy, it is readily available and allows the engineers to obtain a thorough and quick analysis of their design. This fast feedback based on analysis results allows engineers to...
perform more micro iterations in the development process, where each iteration refines and improves the previously built model. The safety analysis is integrated in a development and more specifically in the safety analysis process. This process is aligned to upcoming ISO-CD-26262 standard and described in an EPF (Eclipse Process Framework) model for EAST-ADL2.

The Eclipse platform provides a framework for platform and presentation integration. We use it by implementing our tool as an Eclipse plugin. We extend the graphical user interface of Eclipse by adding menus to invoke safety analysis for a given EAST-ADL2 model. This ensures seamless integration in the UML modeling environment and keeps the overhead for safety analysis experienced by the user as low as possible and thus allows for an iterative safety development process.

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

Data integration in this context is concerned with the transformation of modeling data. We transform from an EAST-ADL2 representation to a HiP-HOPS representation, while preserving the semantics. State of the art data integration for model-based development is supported by powerful model transformation engines and languages. Different transformation languages and engines are available, each of them solving a particular problem especially well. This is why the next section is dedicated to choosing the right model transformation language.

4. Translation from EAST-ADL2 to HiP-HOPS

Integrating safety analysis into the development of automotive embedded systems requires data integration. This can be achieved by translating concepts of the automotive domain to the error analysis domain. In the context of this work the automotive concepts are represented by the architecture description language EAST-ADL2 including its dependability model and the concepts of the error analysis domain are represented by the safety analysis tool HiP-HOPS. We need to expose the information of the EAST-ADL2 error models to HiP-HOPS in its native input format.

4.1 Model Transformation

We automate the translation between EAST-ADL2 and HiP-HOPS using model transformations. Model transformation languages are domain specific languages for extracting information from models, for building and for manipulating models. Model transformation languages, paradigms and engines have been classified in [4] and [8]. Different model transformation languages have their strengths and weaknesses in solving particular types of tasks [4]. A challenge is choosing the right tool for the model transformation task at hand.

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

- Needs to be able to process UML models which have a UML profile applied, in our case this is the EAST-ADL2 profile
- Needs to produce text output, not a model
- Needs to be maintainable, the source code needs to be compact and reusable, since both EAST-ADL2 and HiP-HOPS evolve
- Needs to integrate as a plugin into the modeling environment

The model transformations we have looked at, do not fulfill all requirements at once. For instance we could not find an engine that allows us to produce text output and process the EAST-ADL2 profile. For this reason, we decompose the model transformation into two specialized transformations. Each of the two transformations fulfills the requirements partially, but the two transformations together fulfill all requirements.

4.2 Transformation Design

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

Figure 1. Transformation Design

1. Semantic Mapping Transformation: The first transformation step is a model-to-model transformation and is called M2M Trafo in figure 1. It transforms an EAST-ADL2 model that was created in the Papyrus UML modeling environment into an intermediate model. The structure of the intermediate model resembles the HiP-HOPS grammar, so it is close to the structure of the desired output. This stage performs the semantic mapping between the domains of EAST-ADL2 and that of HiP-HOPS. However, this stage is not concerned with the actual representation of the data.

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

We will discuss both transformations in more detail in sections 4.4 and 4.5. There we use the scheme for classifying model transformation introduced in section 2.1 to determine the type of each of the two transformations and to choose a transformation engine fitting the properties of that particular transformation.

4.3 Involved Models and Metamodels

Three different models are involved in this model transformation. An EAST-ADL2 model, an intermediate HiP-HOPS model and the final HiP-HOPS file. The EAST-ADL2 model serves as the initial source model, and is conform to the EAST-ADL2 metamodel. The HiP-HOPS file is the final outcome of the transformation and conforms to the HiP-HOPS grammar. We discuss the metamodels separately in the following sections.

4.3.1 EAST-ADL2 Error Model

EAST-ADL2 models created in the Eclipse-based Papyrus UML tool have a metamodel that is a composition of several separate metamodels. This metamodel consists of the UML metamodel and the EAST-ADL2 profile definition. These artifacts are combined by the Eclipse framework to the EAST-ADL2 metamodel. However, this combined metamodel is not an autonomous entity or file. This complicates the model transformation and limits the choice of model transformation engines.

The EAST-ADL2 domain model contains concepts for modeling the anomalies of a system in a so called error model, which describes the failure semantics of a system by relating the occurrences of internal errors and the propagations of such errors [2]. These error modeling constructs are separated from the constructs used for the nominal system definition, to clearly separate their different natures: error models are purely descriptive while nominal models are prescriptive and may be used for code generation.

The domain model of the EAST-ADL2 error modeling concepts is illustrated in figure 2. In the following we introduce the core con-
cepts. The ErrorModelType metaclass represents the container for maintaining the information relating to the anomalies of a system, function, software component, or hardware device. The ErrorModelPrototype metaclass describes an instance of an ErrorModelType. Even though these concepts are similar to the concepts for nominal behavior, the decomposition of the system into ErrorModelTypes is kept separate from the nominal decomposition into FunctionTypes. This makes it possible to have either totally aligned or separate topologies in error modeling than in the targeted nominal architecture, depending on the needs for error analysis. An ErrorPropagationLink describes how failures in one component can propagate to other components.

4.3.2 HiP-HOPS Ecore Metamodel

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

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

4.4 Semantic Mapping Transformation

The purpose of the semantic mapping transformation is to map concepts from EAST-ADL2 to HiP-HOPS in a way that preserves the semantics of the original model, even though the structure of the model must be changed heavily. EAST-ADL2 models and HiP-HOPS models are structurally different. This can be demonstrated by the following example. EAST-ADL2 follows the concepts of declaring types first and referencing to the declaration from each point of use. In HiP-HOPS on the other hand, the declaration and usage of a type is coupled, types are declared at the same point as they are used. Thus the declarations have to be inlined into every point of usage, when transforming from EAST-ADL2 to HiP-HOPS. Table 1 lists the detailed mapping between EAST-ADL2 concepts and HiP-HOPS concepts. In Figure 4 we show the part of this transformation that maps ErrorModelPrototypes of EAST-ADL2 to Components of the HiP-HOPS Ecore Metamodel.

According to the classification scheme introduced in section 2.1 the representation transformation can be classified as an exogenous, horizontal, model-to-model transformation. Model-to-model transformations are well suited for our semantic mapping transformation, because both input and output are models. Mapping patterns can be described by relational and declarative transformation languages in a concise manner. Our solution leads to relatively short source code for the solution. We selected the ATLAS Transformation Language (ATL), a language that allows a choice of relational and imperative constructs. It furthermore allows processing of models that have a profiled metamodel, i.e. a metamodel that consists of a metamodel and a profile description. In our case the EAST-ADL2 metamodel consists of the UML metamodel and the EAST-ADL2 profile.

4.5 Representation Transformation

The purpose of the representation transformation is the generation of a textual description based on the intermediate model. According to section 2.1 the representation transformation can be classified as an endogeneous, horizontal, model-to-text transformation.
Textual representations can be generated particularly well with model-to-text transformation languages. We choose the Xpand language of OpenArchitectureWare. Xpand is a template-based model transformation language, which incorporates the output in the form of templates into the control structure. Figure 5 shows a part of this transformation, that creates a textual representation of the intermediate model, that can serve as input to HiP-HOPS.

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

4.6 Benefits of the Chosen Decomposition

In this section we discuss the benefits of this solution.

• Our solution separates two different concerns of the transformation from EAST-ADL2 to HiP-HOPS: (1) the semantic mapping between the domains of EAST-ADL2 and that of HiP-HOPS and the (2) details of the concrete syntax of the HiP-HOPS input file.

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

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

• Since we chose appropriate tools for each steps, the resulting model transformation source code is very concise, resulting in a maintainable codebase.

5. Case Study

To demonstrate how the safety plugin works, we have created a hierarchical system model in EAST-ADL2 with a focus on the system’s comprehensibility. The model is depicted in figure 6 and contains three functions. While F1 and F3 are atomic functions, F2 contains two subcomponents, a primary component F21 and a standby component F22, where the later takes over if the primary fails.

Table 1. Semantic mapping between EAST-ADL2 and HiP-HOPS

<table>
<thead>
<tr>
<th>EAST-ADL2 Pattern (Source)</th>
<th>EAST-ADL2 Type</th>
<th>HiP-HOPS Pattern (Target)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorModelType</td>
<td>ErrorModelType</td>
<td>System</td>
</tr>
<tr>
<td>ErrorModelType.errorConnector</td>
<td>ErrorPropagationLink</td>
<td>System.Lines</td>
</tr>
<tr>
<td>ErrorModelType.parts</td>
<td>ErrorModelPrototype</td>
<td>System_Component</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorPort</td>
<td>ErrorPort</td>
<td>System_Component.Ports</td>
</tr>
<tr>
<td>ErrorModelPrototype</td>
<td>ErrorModelPrototype</td>
<td>System_Component.Implementation</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorBehaviorDescription.internalErrorEvent</td>
<td>ErrorEvent</td>
<td>System_Component. Implementation.FData.basicEvent</td>
</tr>
<tr>
<td>ErrorModelPrototype.type.errorBehaviorDescription.failureLogic</td>
<td>String</td>
<td>System_Component. Implementation.FData.outputDeviation</td>
</tr>
<tr>
<td>ErrorModelPrototype.type</td>
<td>ErrorModelType</td>
<td>System_Component. Implementation.System (recursion)</td>
</tr>
</tbody>
</table>

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

```
rule recursiveErrorModelPrototypeRule{
    from
        emp : ErrorModelPrototype
    to
        _components : bishops | Component {
            implementation <- _implementation,
            ports <- _ports,
            name <- emp.base_Property.name
        }.
}
```

Figure 5. Example: part of the representation transformation in OAW Xpand

```
<DEFINE ComponentRule FOR Component->
    Component {
        Component type "componentType="
        Name "name="
        Description "description="
        <EXAND PortRule FOR ports->
        <EXAND ImplementationRule FOR implementation->
    }
</DEFINE->
```

Figure 7. Fault tree of the hot standby system of figure 6
Figure 2. Domain Model of the EAST-ADL2 Error Modeling Concepts

Figure 6. EAST-ADL2 model describing the failure propagation in a hot standby system
This pattern is called hot standby. It consists of a primary component and a standby component, that is ready to take over if the primary fails. The pattern is applied in reliability engineering as a failover mechanism to improve the reliability and safety of a system. The result of the safety analysis shows, how this pattern affects the outcome of the safety analysis.

We run our automated model transformation for the model depicted in figure 6. The EAST-ADL2 model is transformed into the HiP-HOPS language, as described in section 4.2. HiP-HOPS subsequently performs the analysis based on the transformed data and presents the results in various ways. The results can be represented as minimal cutsets, FMEA tables or fault trees. A fault tree of the system in figure 6 is depicted in figure 7.

The results of the safety analysis need to be interpreted by a safety engineer. Depending on the current stage of development, the engineer can use this information either to create and refine safety requirements or to adjust his design. He can do this e.g. by increasing redundancy, if a higher level of reliability is desired, or reducing the cost by removing unnecessary redundancy.

6. Related Work

The need to bridge the gap between the safety and system design disciplines has been identified in several domains. Integration in general covers the need to align processes, tools and the competences of the developers. The focus here is on model and tool integration between system design models (which may refer to structure and/or behavior at different abstraction levels) and safety analysis models such as failure modes and effects analysis and fault tree analysis models. Earlier work that has addressed this gap includes:

- As part of the SETTA project, Papadopoulos et al. annotate Matlab/Simulink models with FMEA information and provide an export for fault-tree generation [10].
- In the ESACS project and its follow up project ISAAC this gap was identified and addressed by providing tools, such as Statemate Magnum and Scade, to perform automated safety analysis starting from a system design model.
- Dumas et al. [5] perform model transformations between AADL, the Architecture and Analysis Description Language, and the analysis tool AltaRica [1].
- Price et al. [12] focus on safety analysis of electrical systems for cars. They explain the tradeoff between numerical and qualitative analysis and point out the importance of continuous safety analysis as opposed to snapshot analysis. While this work focuses mainly on electrical system, our approach can be used for electrical systems, software systems, or a combination of the two.
- The integration of HiP-HOPS and EAST-ADL2 has been attempted in the ATESST research project [13]. This resulted in a monolithic Java program where the code for data integration, control integration and presentation integration was mixed, resulting in low maintainability. In this work we specifically addressed these shortcomings by separating the different integration issues and by designing a transformation that leverages state-of-the-art model transformation technology and at the same time separates mapping concerns from representation concerns.
- A recent project that is based on all these advances and that has the ambition to provide multi-domain solutions, is the CESAR project [11].

7. Conclusion and Future Work

In this work we have shown how we integrated the safety analysis tool HiP-HOPS into the automotive model-based development toolchain based on EAST-ADL2. We used different model transformation techniques to translate the relevant information from the automotive domain to the safety analysis domain. This link enables early safety analysis.

Currently our analysis supports fault tree analysis for models of either hardware or software. We work on supporting fault tree analysis that considers the propagation of failures between hardware and software. This will allow us to analyze e.g. the effects of hardware failures on the software and how the software can handle them.

Acknowledgments

The research presented in this article was supported by funding from the 7th Framework Programme of the European Community under grant agreement no. 224442.

References