Integrated tool-chain for improving traceability during the development of automotive systems

E. Armengaud[1], M. Biehl[2], Q. Bourrouilh[1], M. Breunig[3], S. Farfeleder[4], C. Hein[5], M. Oertel[6], A. Wallner[1], M. Zoier[7]
[1]AVL List GmbH, {eric.armengaud, quentin.bourrouilh, alfred.wallner}@avl.com
[2]KTH, biehl@md.kth.se,
[3]Infineon, michael.breunig@infineon.com,
[4]TU Vienna, Stefan.farfeleder@tuwien.ac.at,
[5]Fraunhofer Fokus, christian.hein@fokus.fraunhofer.de,
[6]OFFIS, markus.oertel@offis.de
[7]VIF, markus.zoier@v2c2.at

***Abstract*— Tool integration is a key factor for improving development efficiency and product quality during the development of safety-relevant embedded systems. We present in this work a demonstrator based on the most recent outcomes of the CESAR project. The proposed integrated tool-chain aims at better linking development activities together, thus improving traceability during requirements engineering, system design, safety analysis and V&V activities using a model-based development approach. We analyze the proposed tool-chain from three different points of view: (1) tool integrator, (2) technology provider, and (3) end-user. These different points of view enable the description of the different technologies used at the different levels and the analysis of the benefits for the end-user.**

***Index Terms*—CESAR, integrated tool-chains, functional safety, ISO 26262**

I. Introduction

Embedded systems are widely used in transportation domains (e.g., automotive, avionics, aerospace, rail) in order to reduce the costs, improve the existing functionalities or even enable completely new functionalities. Electronic control units are involved in safety-critical functionalities, and proper development and analysis of the electrics, electronics and software parts are strongly required in order to ensure the required product safety. To that purpose, different functional safety standards have been developed for a more rigorous approach to the system modeling, design, analysis, verification and validation such as DO178B [8] for the avionic domain, EN50128 [9] for the railway domain, ISO 26262 [7] for the automotive domain or IEC61508 [6].

A major challenge in the development of safety-relevant embedded systems is the integration of the different disciplines and the management of the traces between the different activities. The CESAR[1] project [5] has been started in this context to improve the processes and methods for safety-critical embedded systems development. In addition to significantly improving tools and methods of the System Engineering, this R&D project focuses on the development of an interoperability platform, which is called the CESAR Reference Technology Platform (RTP). The CESAR RTP represents a conglomerate of entities which facilitate the creation of integrated development environments for the development of safety-relevant embedded systems for various domains. These tool chains are called instances of the RTP.

The contribution of this work is to present an integrated tool-chain for improving the development of automotive embedded systems. A similar work for the aeronautics domain is available in [18]. This paper represents a further enhancement of the work presented in [1] and is based on the latest developments within the CESAR project. In this work, we discuss how the disciplines of requirements engineering (requirement formalization, management, analysis) have been integrated to system engineering disciplines (model-based development, safety modeling) and to V&V disciplines (simulation, evaluation) and what the benefits for the end-users are. From a tool integration point of view, different strategies have been used to integrate the different disciplines. The proposed platform enables the evaluation of the different integration approaches. The resulting tool-chain including eight tools is illustrated in Figure 1.

The contribution is organized as follows: Section II focuses on the tool-chain from the integrator point of view. In this section, the integration platform and different integration strategies (ModelBus platform, CMM API, model transformation with QVT, ATL, or direct with Java) are discussed. In Section III, the tool-chain from the technology provider point of view is discussed and the innovations regarding requirements engineering, system and safety modeling and verification and validation are presented. In Section IV the benefits of the proposed tool-chain from an end-user point of view are evaluated. Finally, Section V concludes this work.

[1] http://www.cesarproject.eu/

II. TOOL-CHAIN FROM INTEGRATOR POINT OF VIEW

In terms of realizing an integrated tool-chain an engineer has to deal with several problem domains. In the late 80's Wassermann [12] already identified the five dimensions for tool integration: platform, presentation, data, control and process. This approach was extended by [13]. However, in today's business of tool integration, the main focus is on data and process integration, in particular the creation of a seamless integrated tool-chain in which data can be exchanged between different tools according to a defined development process in order to reduce gaps between tools and mainly to bring as much automation as possible into the development process.

But how can tools be integrated at all? Unfortunately, there is no step-by-step approach available, due to the complexity of the problem. Nevertheless, several guidelines exist which provide useful information for integration engineer. Accordingly, an integrator has to answer different questions like which tools should be integrated, what kind of data should be shared with others or what functions can be separated. However, all these guidelines and useful questions don't answer the most difficult question how to achieve the best interoperability in a tool-chain. Therefore, we have analyzed and implemented different integration approaches in our tool-chain.

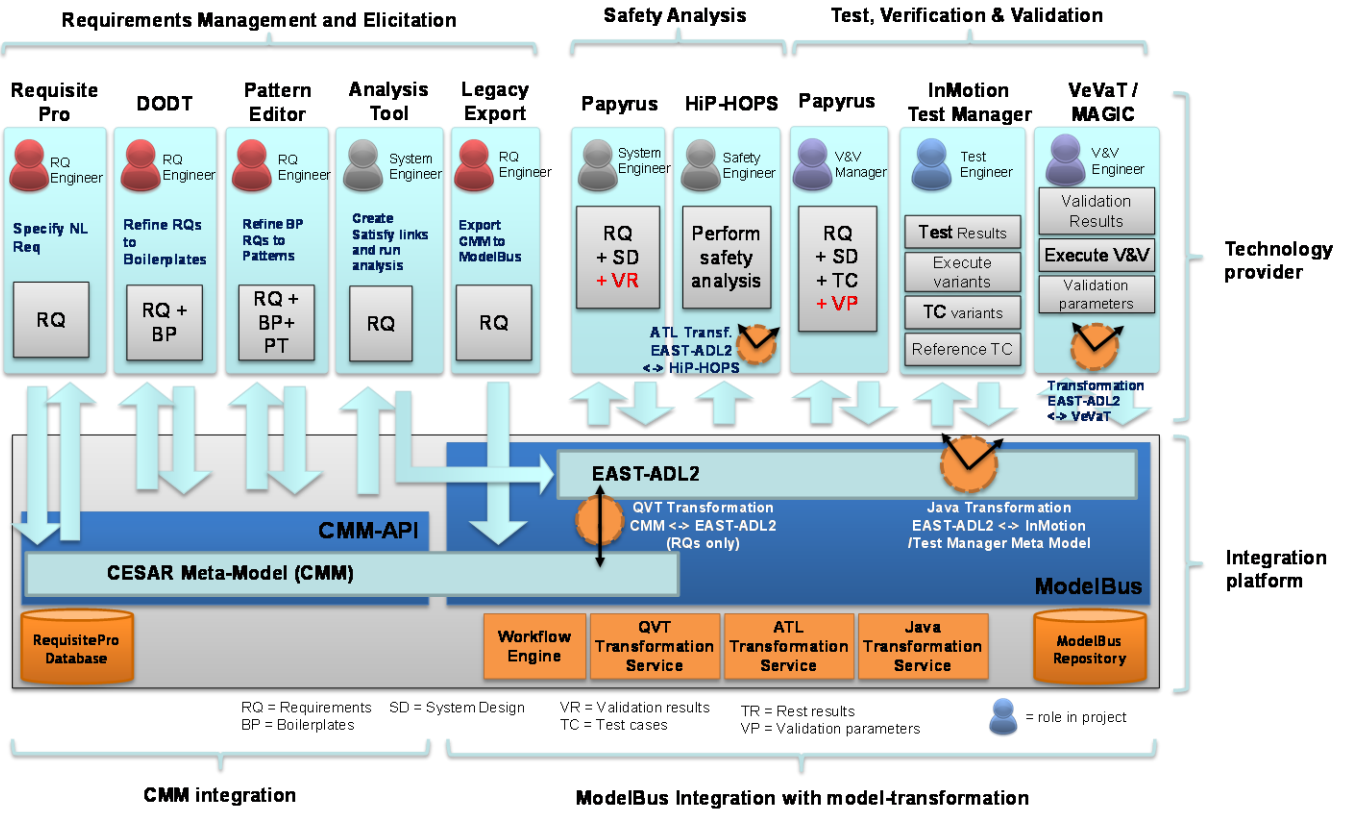

Figure 1: Proposed tool-chain

Figure 1 illustrates the proposed approach. In this platform, direct tool integration using the CMM-API has been provided for the requirements engineering part. This approach enables tight integration from the semantic point of view since all tools use the same meta-model (same structure of the exchanged data). Further, the ModelBus technology has been used as SOA platform for the integration of the system design, safety analysis and V&V parts. In this integration platform, the provision of a common semantic understanding has been achieved relying on EAST-ADL2 and providing model-transformation using different technologies such as QVT, ATL or direct Java code. The model transformation service is executed either as platform service (e.g., QVT transformation) or integrated in the tool adapter.

A. *ModelBus and QVT engine*

ModelBus [15] is the underlying integration framework for the described tool-chain. ModelBus is based on SOA principle and in particular developed for model-driven tool-chains in which models are the central artifacts. According to SOA principles, functions and methods are provided as services which are useful for other stakeholders of the tool-chain. In a model-driven tool-chain typical modeling services are transformation, verification or report generation.

In the proposed tool-chain several model transformation services are used. One of these transformations is the QVT transformation service. QVT stands for Query, View, Transformation which is a specification provided by the OMG [14]. The benefit of using a QVT based transformation is the possibility to transform models in an incremental way. Incremental means that only updates or changes are transformed, instead of transforming the complete model whenever a change happens. In addition to that, it supports also bi-directional transformation by using transformation

rules which are formulated as relations between models and model elements.

In order to provide these features the QVT engine creates a traceability model for each transformation according to the defined transformation rules. The traceability model will also be stored in the model repository together with all other models which will be produced during development time within the tool-chain. The traceability model can also be used by other services like coverage analysis or report generation services.

In the proposed tool-chain the model-to-model transformation from requirements in CMM format to EAST-ADL2 is realized by QVT. Transformation rules mapping the CMM model elements to the corresponding EAST-ADL2 elements have been designed, also taking into account that not all elements are available in both meta-models (e.g., EAST-ADL2 has no dedicated object for boilerplates). The QVT engine takes source- and target meta-model, the transformation rules and an initial predefined trace model containing the information, where to insert the requirements model into the overall EAST-ADL system model. The execution of the transformation is triggered by the workflow engine of the ModelBus platform, the trace model stores the relations between source and target model elements for each incremental transformation.

B. CMM API

The CMM-API[16] shall support the user with two different aspects of interoperability: Ease the integration of new tools and repositories in a tool landscape by providing access and editing methods for data and provide a semantically unified view on the different data formats.

In this use-case CMM-APIs establish the interoperability between the requirements engineering tools. As an example it enables the DODT and the PatternEditor to connect to RequisitePro to exchange requirements. Being a library for system engineering data, it provides methods to handle requirements and other system artifacts regardless of the target data repository and the native data format. This means that a client tool does not need to know any specifics of RequisitePro. If the requirements repository would be changed to IBM DOORS there is no need to change a single line of code in the tools using the CMM API.

The main benefit of the CMM-API is the support of multiple engineering data repositories in a transparent way. Requirements can be seamlessly and simultaneously retrieved from tools such as Requisite Pro or DOORS as well as from files being stored in a file-based repository like the ModelBus repository. This provides a unified homogeneous and traceable view on all distributed and heterogeneous engineering data that is involved in a system engineering process. The CESAR Meta-Model (CMM) is used for this representation. Figure 2 illustrates this view vividly in form of the CMM glasses. Different tools use different representations to store and present their data. Simulink, EAST-ADL and AUTOSAR are some common design technologies, RequisitePro and DOORS are typical tools for Requirements engineering. The CMM glasses provide a view that allows getting all elements as CMM objects, creating and viewing links across tool borders and creating a unified look and feel for the whole system. The CMM-APIs are exactly these glasses for the tool that wants to interact in a Reference Technology Platform (RTP).

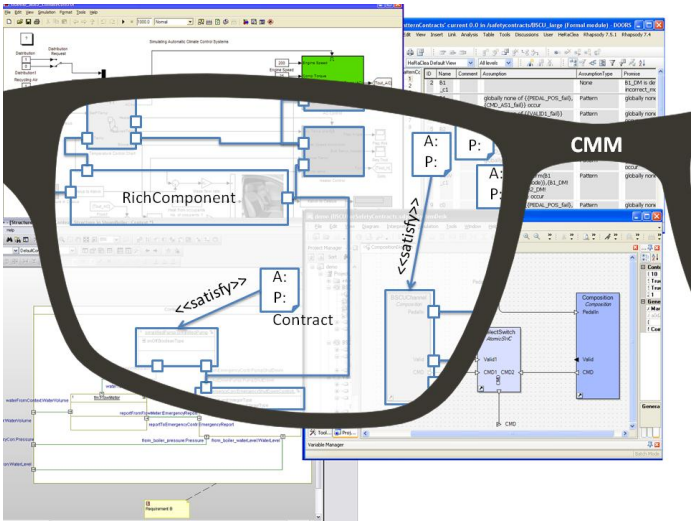

Figure 2: The CMM API provides a unified view on the whole system

Concluding: The CMM-APIs enable the designer to relate and trace the different elements across the whole system. This is not limited to requirements only. A further benefit is the fact that data is not duplicated. The API interacts “live” with the target repository. It is still possible though to modify and add requirements in RequisitePro manually. Changes are automatically available in all CMM-API connected tools.

The unified view on system models and the tool connectivity are an important part of the use case. Requirements needed to be linked to model elements stored in an EAST-ADL file. Through the CMM-API the EAST-ADL file looks like CMM and could be processed by the satisfy-link tool. Storing and processing requirement in RequisitePro, DODT and the PatternEditor was made easy. The problems of different file formats and tool interactions vanish, the engineering activity itself gains the focus.

C. Integration HiP-HOPS, ATL

To integrate the safety analysis tool HiP-HOPS into the CESAR RTP, we create a ModelBus-conform tool adapter. The tool adapter handles several tool integration aspects [4], both control integration and data integration. The tool adapter implements the ModelBus tool adapter interface, ensuring that it can be integrated into the ModelBus and register for ModelBus events.

To realize control integration, the tool adapter listens to checkin-events in the ModelBus repository, specifically changes on Papyrus UML files. If such a change is detected, the Papyrus model is fetched from the ModelBus repository, transformed into a HiP-HOPS representation, the HiP-HOPS analysis for FMEA and FTA is started and finally the results

are displayed graphically.

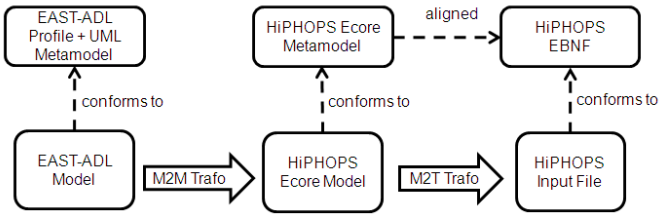

Figure 3: Model Transformation Chain for the Integration of the HiP-HOPS Tool

To realize data integration, the tool adapter implements model transformations. The model transformations automate the translation between EAST-ADL2 and HiP-HOPS. The model transformation is decomposed into two parts, (1) a semantic mapping transformation from EAST-ADL2 to an intermediate representation and (2) a representation transformation from the intermediate representation to HiPHOPS.

Semantic Mapping Transformation: The first transformation step is a model-to-model transformation written in ATL [17]. 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.

EAST-ADL	HiPHOPS
ErrorModelType	System
ErrorModelType.errorConnector of type ErrorPropagationLink	System.Lines
ErrorModelType.parts of type ErrorModelPrototype	System.Component
ErrorModelPrototype.type.errorPort of type ErrorPort	System.Component.Ports
ErrorModelPrototype	System.Component.Implementation
ErrorModelPrototype.type.errorBehaviorDescription.internalErrorEvent of type ErrorEvent	System.Component.Implementation.FData. basicEvent
ErrorModelPrototype.type.genericDescription of type String	System.Component.Implementation.FData. outputDeviation
ErrorModelPrototype.type of type ErrorModelType	System.Component.Implementation.System **(recursion)**

Figure 4: meta-model integration

Representation Transformation: The second transformation step, is a model-to-text transformation, it 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.

Discussion: 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 (2) the 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 a result, the robustness and maintainability of the transformation is improved.

III. TOOL-CHAIN FROM TECHNOLOGY PROVIDER POINT OF VIEW

The proposed development process is illustrated in Figure 5 and covers different development activities such as requirements engineering, system design, safety analysis and V&V activities.

A. Requirements engineering

Through safety standards like the ISO26262, emerging complexity in automotive embedded systems and a highly competitive market the amount and variety of requirements on a product and the development process itself increased rapidly. Technologies for ensuring full traceability, high quality of requirements and easy analyzability are demanded.

In the requirements engineering part of this use case requirements get formalized from natural language text to semi-formal boilerplates using DODT und finally to formal patterns using the PatternEditor. The requirements are stored in the RequisitePro tool. Traceability between all artifacts is established through the CMM API described in section II.B.

Rational RequisitePro is a requirements management tool marketed by IBM. A wrapper library translates CMM API commands into invocations of the RequisitePro Extensibility API and vice versa. As a result this allows storing and retrieving natural language, boilerplate and pattern requirements without loss of information. Requirements in RequisitePro are grouped into different types; this implementation uses three different types: one for the requirements themselves, one for the boilerplate templates and the last one for the CMM data which is shared between several requirements, e.g., stakeholders. The adapter also allows storing links to entities stored outside of RequisitePro, e.g., a model component stored In the ModelBus repository.

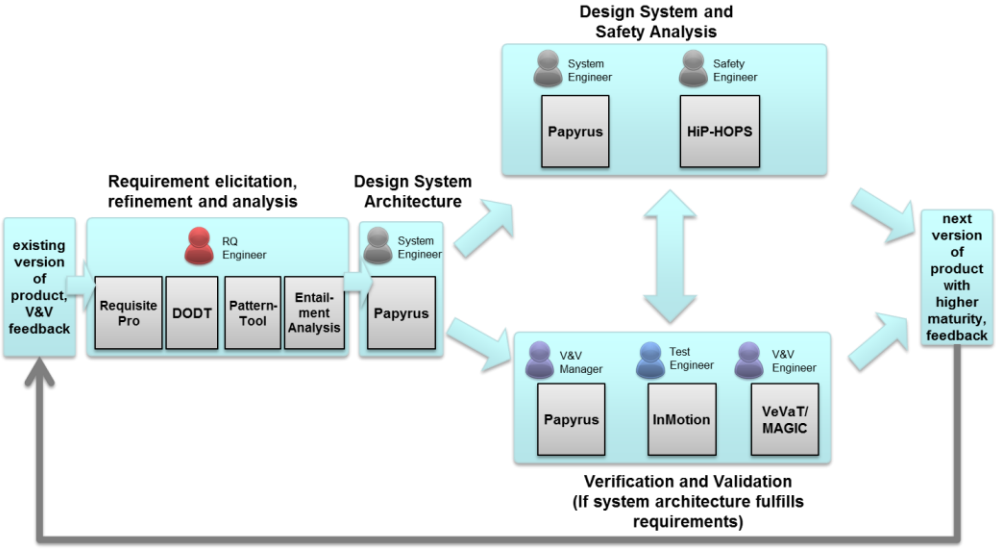

Figure 5: Resulting development process

DODT (domain ontology design tool) is a requirements elicitation tool. Its main purpose is to support the requirements engineer in specifying and analyzing requirements. DODT supports natural language and boilerplate requirements. A boilerplate is a template for a natural language requirement, consisting of fixed syntax elements and variable parts, the attributes, that need to be filled by the requirements engineer. The following is an example for a boilerplate requirement (the fixed syntax elements are highlighted with bold font):

The Hybrid system **shall** keep the temperature of HV battery **within** 10 **and** 30 °C.

A suggestion functionality is available to help refining existing natural language requirements into boilerplate requirements [10].

A domain ontology contains the concepts and relations of the problem domain. Using this knowledge, DODT is able to make proposals during the specification of requirements [11] and to analyze existing requirements with regards to completeness, consistency and ambiguity. The tool also gives suggestions on how to improve those requirements. The domain concepts in a requirement are highlighted which helps seeing important parts of a requirement. An ontology editor is included that allows creating and editing the domain ontology information used by DODT; alternatively an external editor like Protégé can be used.

After having stated the requirements as structured text using DODT a further formalization to patterns can be performed through the PatternEditor. Patterns consist of static text elements and attributes being filled in by the requirements engineer. Each pattern has a well defined semantic in order to ensure a consistent interpretation of the written system specification across all project participants. On the one hand this limits the possibilities of writing a requirement on the other hand it prevents misunderstandings regarding the interpretation of the sentences. To gain a set of quality requirements this limitation is necessary especially to allow a guided and automated verification of the system against the requirements (generating observers, automatic test case generation). However writing requirements shall still be possible in an intuitive way. Patterns allow the writing of natural sounding requirements with defined semantics while being expressive enough to formalize complex requirements.

DODT stores additional information from the ontology with the requirements itself giving the PatternEditor the possibility to identify elements like events or conditions. Based on this information the selection process for the best fitting pattern can be guided. Also the filling of the attribute values can be assisted. A pattern that is used in the hybrid vehicle use case expressing the braking delay of the e-drive looks like this:

```
whenever recuperate occurs
(brakeForceDrive == 1) holds during
[0ms,20ms]
```

Having this kind of specification of the components of the electric braking system it is possible to formally check the requirements against higher level requirements in the requirements structure. Most safety standards require

traceability between requirements and demand checks to ensure that requirements are split up correctly. With the entailment analysis this check can be automated. In this use case we have a high level safety requirement related to the Anti-lock braking system (ABS) functionality stating that there shall be a timeframe of 20ms within the next 40ms after slip is detected on the wheels, where steering should be possible.

Taking the functional requirements of the system into account the entailment analysis is capable of determining if the safety goal can be fulfilled. If there are cases where the safety goal can be violated the analysis provides a possible failure scenario. In this use case it has not been considered in the specification that the road conditions may change very quick (e.g. driving on cobblestones) and this fact would lead to permanent brake force activation and a deactivation of the ABS.

The proposed two stage approach using structured text in combination with formal representations and ensuring traceability through the CMM-API allows reducing the effort for consistent and complete specification. Furthermore mistakes in the specification can be detected very early in the design process, by either running the ontology based analyzes or formal requirement checks at selected parts of the system.

B. System design and safety analysis

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 tool-chain 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-26262. This highlights the need for (1) the rigorous modeling of automotive functions including safety related-aspect and (2) the seamless integration of safety analysis into the development process.

EAST-ADL2 is an architecture description language for modeling automotive embedded systems [3]. 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 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 a UML design tool, i.e. the Eclipse-based Papyrus UML tool. 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. 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. We will make use of the EAST-ADL2 error model in the following tool integration.

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 the risks, proceeds with identifying the causes of failures and deriving the safety requirements and concludes with developing safety solutions. One of the first steps in safety analysis is a hazard analysis, i.e. an analysis of the risks exposed by the system under study. This is typically accomplished by FMEA (Failure Modes and Effects Analysis) or the FTA (Fault Tree Analysis). However, creating the FMEA and the FTA 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 [2] (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.

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

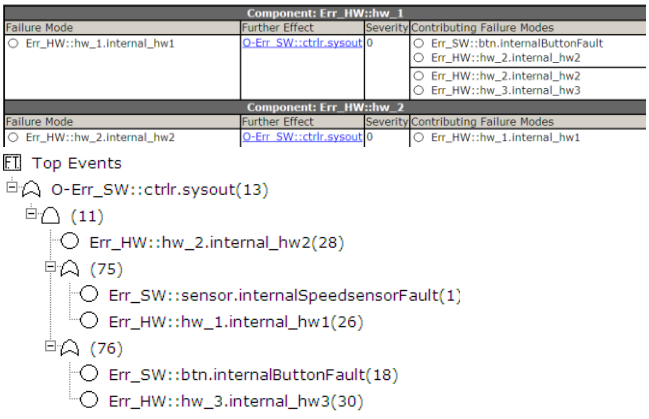

Figure 6: FMEA and FTA generation with HiP-HOPS

C. *Verification and validation*

One important challenge while testing automotive embedded systems is the complexity of the components and the complex interactions with its environment. Hence, the electronic control unit directly interacts with mechanical counterparts (e.g., E-motor, engine), that are further integrated in a vehicle driving on a road and controlled by a driver. This complex multi-physics simulation and the complex interactions with the electronics require the use of advanced simulation and evaluation tools. The proposed approach enables more complete evaluation according to complex validation rules, reduces the risk of human error and speed-up the processing time.

As explained in previous chapters, at the beginning of the workflow via the "Automotive tool-chain" in hand, requirements are elicited, refined and formalized. After doing this, a further challenge in this workflow is an automated verification & validation of these requirements, i.e. a check, whether requirements are fulfilled by the system – or not. Thereby the traceability from requirements definition via testing and test result validation must be ensured.

In a rough overview, the Verification & Validation sub-workflow comprises three activities:

1. Modeling the test cases and corresponding validation cases which are needed for validating the requirements (by means of Papyrus).
2. Performing automated test runs (by means of the simulation tool AVL InMotion), which yield characteristic test results of the system to be validated.
3. Automated checking of the generated test results against previously defined requirements (by means of the V&V tool AVL VEVAT).

These steps (incl. involved tools and rough data flows) are shown in the graphics below:

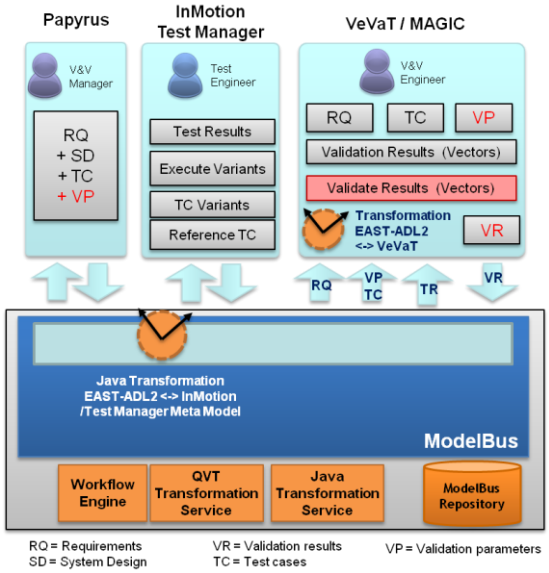

Figure 7: V&V - Workflow survey

The first activity (modelling the test cases and corresponding validation cases) is the enhancement of the EAST-ADL2 model with description of the test cases, test campaigns and validation cases in Papyrus by the V&V manager. Thereby, in order to ensure traceability, links are created between requirements, InMotion test cases (simulation runs with parameter variations) generated test results and VEVAT validation cases and validation results. The following graphics shows a section of the enhanced EAST-ADL model (incl. links between VEVAT validation procedures, InMotion test results and VEVAT validation results.

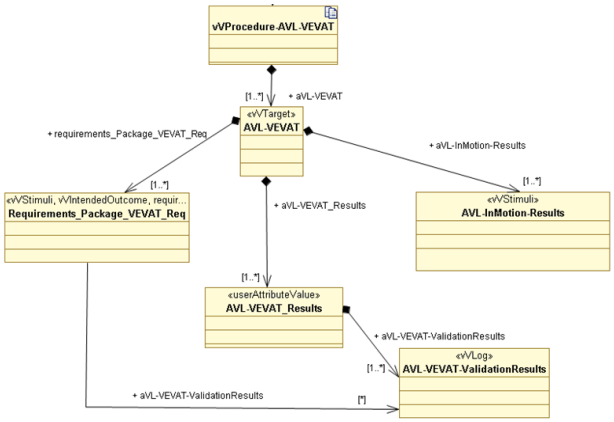

Figure 8: V&V – Model enhancement

Based on this information, the test scripts for the simulation tool AVL InMotion are generated by the test engineer and the validation procedures for the V&V tool AVL VEVAT are generated by the V&V engineer.

Now the scripts for the simulation can be automatically executed by AVL InMotion. Thereby the links to the generated InMotion test results (result-IDs of simulated waveforms) are stored in the model – to be transferred afterwards (together with the system requirements) into a so-called "Requirements file" to the VEVAT tool.

Currently this "VEVAT Requirements file" is an Excel file, which is generated automatically by the VEVAT RTP-adapter,

i.e. the requirements are fetched from the model bus and a transformation from EAST-ADL to Excel (current VEVAT input format) is performed.

After completion of InMotion simulation runs, the V&V engineer gets notified that new test results are available for validation. He starts the V&V tool AVL VEVAT, which automatically performs evaluation of the simulation results according to predefined test- and validation criteria. Thereby VEVAT detects resp. calculates significant system behavior properties from simulated data (e.g. vehicle deceleration during energy recuperation, battery temperature and SOC etc.) and compares them to predefined system requirements.

During evaluation, the analyzed signals optionally might be visualized – as shown in the picture below. Thereby the areas of interest (e.g. recuperation phases of a hybrid vehicle) appear highlighted in the graphics.

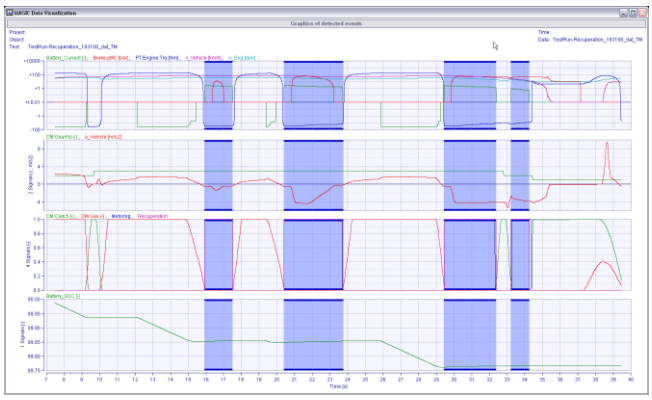

Figure 9: V&V – Graphics of analyzed data

After validation, VEVAT inserts the validation results (Passed/Failed flags, comments, detected signal ranges) of each requirement into the Requirements file.

The following picture shows an example of a VEVAT Requirements file after performing the requirements validation, i.e. it contains not only the requirement definitions but additionally the validation results.

After performing the requirements validation, the VEVAT requirements file contains not only the requirement definitions, but shows evidently the validation status of each checked requirement (PASSED (green), doubtful (yellow), **FAILED** (red) in addition. Furthermore it provides reason for failures, detected and demanded values etc.

Requirement definitions					Validation results				
RequirementID	Description	Lower limit	Upper limit	Unit	Status	Comment	Detected minimum	Detected maximum	Unit
VV_RECUP_General_01-BP	The Hybrid system shall be able to check the HV SOC.				unchecked	No value supplied			
VV_RECUP_General_07-BP	If clutch pedal signal = 0 , the Hybrid system shall enable the recuperation.		0,00		PASSED		0	0	
VV_RECUP_General_08-BP	The Hybrid system shall enable the recuperation when the Driver activates the service brake.	1,00			FAILED	Lower error limit violated	0	1	
VV_RECUP_General_08-BP_	The Hybrid system shall enable the recuperation when vehicle in motoring mode.	1,00			PASSED		1	1	
VV_RECUP_General_10-BP	The Hybrid system shall keep the temperature of HV battery within 10 and 30 °C	10,00	30,00	°C	PASSED		20,017292	20,1153238	°C
VV_RECUP_General_11-BP	The e-motor shall be able to generate the brake torque.				unchecked	No value supplied			
VV_RECUP_General_12-BP	The Hybrid system shall be able to conform to the NVH criteria.				unchecked	No value supplied			
VV_RECUP_General_14-BP	The Hybrid system shall enable the recuperation when vehicle speed greater than 60 km/h.	60,00		km/h	FAILED	Lower error limit violated	13,55733756	73,2275136	km/h
VV_RECUP_General_15-BP	While motoring_recuperation , the Hybrid system shall decelerate the vehicle less than 10 m/s².		10,00	m/s²	PASSED		-6,661513303	-1,364381317	m/s²
VV_RECUP_General_16-BP	The charging power of Hybrid system shall depend on the HV SOC.				unchecked	No value supplied			
VV_RECUP_General_18-BP	If temperature of e-motor greater than 70 °C the Hybrid system shall not charge the HV battery.		70,00	°C	PASSED		20,017292	20,1153238	°C
VV_RECUP_General_18-BP_	If temperature of HV battery greater than 50 °C the Hybrid system shall not charge the HV battery.		50,00	°C	PASSED		20,5443213	22,872681	°C
VV_RECUP_General_17-BP	The the Hybrid system shall maintain the HV SOC less than 95 percent.		95,00	percent	PASSED		89,052839	89,8554265	percent

Figure 10: V&V – Requirements list incl. validation results

Finally the results of this automated validation are stored (individually for each requirement) back to the model bus resp. RTP-repository (via transformation of a subset of Excel-data to EAST-ADL). Now the requirements manager might track the status of all requirements directly in the model (via the Papyrus tool).

IV. End-user point of view / evaluation

The evaluation has been performed according to four different perspectives: Efficiency (%), Quality (%), Complexity (%), and Cost / Effort (%). A list of 22 metrics has been developed for the first three perspectives for the automotive domain, see Figure 11. Notice that the fourth perspective (cost / effort) is directly dependent from the efficiency, quality and complexity goals.

For the evaluation, a recuperation function for an electric / hybrid vehicle as pilot application has been used [1]. This function is a key feature in order to achieve the targeted fuel economy and emissions reduction. The basic operation of this function is following: when a negative torque is requested, the electric motor is then working as generator and the kinetic energy of the vehicle is transformed into electrical energy. This function has safety critical aspects: indeed, it has an impact on the dynamic of the vehicle (e.g., unintended recuperation and therefore braking while driving at a high speed). The function has also an effect on the storage of the electrical energy with for example the possibility of high voltage battery overloading (leading to fire or explosion). The development of such a recuperation function for an electric / hybrid vehicle provides some rooms for improvement. The engineering activities of the development process, in particular for the definition of the system requirements and the design of the system architecture, are not harmonized regarding to the methods and tools used. Therefore the transfer of information is still often based on Office documents and human data transformations, which is the source of ambiguity and misunderstanding in larger teams. Another problem is the lack of traceability between the exchanged data through the development, e.g. between requirements, system components, test cases and test results.

The proposed tool-chain demonstrator provides the following key features

- Formalized requirements enabling better quality and early validation of the system specification
- Semi-formal architecture specification enabling higher quality in the system description and mapping between the different system viewpoints.
- Automated generation of safety analysis (based on semi-formal architecture description) enabling earlier system improvements
- Automated test case execution and evaluation, thus enabling regression test and saving time during system evaluation and improving test and validation coverage.
- .Traceability: A major benefit is routed in the

traceability between requirements, system components, test cases and test results. This has been enabled by the meta-model covering the entire development process.

- Automated notification facilitating coordination within the development team.

Model transformation enabling automated data transfers between the tools and thus reducing development time and human error at tool boundaries.

Goal 1: Efficiency			
Sub-Goal ID	**Sub-Goal**	**Measure Recurrence**	**Metrics**
1.1	Reduce development cycle time	General	Efforts spent in the phase Elapsed time for the phase
1.2	Find defects as early as possible	For each detected failure	Total number of failure in the phase Total effort spent to fix a failure Average failure distance for each phase
1.3	Improve requirements stability	General	Opened defects/failures in a phase Closed defects/failures in a phase
1.4	Improve system model maintainability	For each detected failure	Total number of failure in the phase Total effort spent to fix a failure Average failure distance for each phase
Goal 2: Quality			
Sub-Goal ID	**Sub-Goal**	**Measure Recurrence**	**Metrics**
2.1	Improve reliability	For each failure	Severity
2.2	Improve system's document quality	For each document type For each document revision For each document defect	Number of revision Number of page Severity level of defects
2.3	Improve requirements traceability	General	Number of traced requirements / total number of requirements
Goal 3: Complexity			
Sub-Goal ID	**Sub-Goal**	**Measure Recurrence**	**Metrics**
3.1	Improve configurability	General	Configurable items that were handled by software change versus total number of configurable items
3.2	Improve requirements management	General	Number of requirements Efforts to manage the requirements
3.3	Improve verification ability	General	Number of verified requirements / total number of requirements
3.4	Improve handling of change	General	Number of changes Efforts to manage the changes
3.5	Improve system modeling capabilities	General	Number of aspects that can be captured

Figure 11: Evaluation criteria

The efficiency goal is mainly supported by the requirement formalization and component-based modeling according to the EAST-ADL methodology. The meta-model provides a specification of the data structure of the different assets (e.g., requirements, functions), as well as of the traceability links between the assets. This structure guides the designer to provide complete information and also supports tests of the data organization. This supports finding defects as soon as possible. The availability of explicit traceability links between the system assets (e.g., requirements, functions, test cases) further supports the understanding of the system by the engineers, thus simplifying system development. System maintainability is also improved using this semi-formal approach. Additionally, the automated generation of safety analysis (FTA, FMEA) enables fast assessment of the model being developed, thus enabling iterative development of safety-critical systems and finding defects early in the development process.

Efficiency is also improved thanks to the requirement formalization part. Hence, the resulting requirements are more complete, comprehensible and correct, thus improving requirement stability and shortening validation. A further aspect to manage the efficiency is the support of notifications. Hence, after the completion of a development activity, a notification is sent to the next user to trigger the following activity. This notification service is useful to coordinate development within large teams and thus reduce coordination efforts.

The quality goal is improved thanks (1) to the enhanced traceability of the requirements within the development process and above the tool boundaries, and (2) to the formalized requirements. Traceability during architecture description according to EAST-ADL with the Papyrus tool is provided by the methodology itself. The model transformations between requirements, system modeling (Papyrus), safety analysis (HiP-HOPS) and V&V (InMotion, VEVAT) provide traceability of selected assets between the tools. Hence, the information is automatically transformed from a tool to another thus (1) minimizing human error while transmitting the information and (2) enhancing the traceability above the tool boundaries. With this integrated tool-chain, the requirements are linked with the architecture description, test cases and test results information. Further, the improved quality of the requirements provides a better description / specification of the system.

The complexity management goal is supported by the traceability within the development system as well as automatic generated safety analysis and the model-transformation implementing the generation of test cases. Hence, test campaigns can be specified in EAST-ADL with different variation parameters (different possible values for inputs parameters). The transformation engine automatically generates one test case for each possible value of the input parameters. Additionally, the test results are automatically inserted in the system description during the second transformation. This eases the management of test campaigns. Furthermore, the ability to automatically generate FTA and FMEA improves the verification ability of the system.

Regarding the cost / effort goal, an additional effort is required while introducing the semi-formal approach from EAST-ADL. Hence, performing requirement specification as well as system architecture definition with an UML tool

according to the EAST-ADL meta-model is more complex than entering free text requirements and performing architecture specification with a text editor or graphical tool. It is the same case for formalization of requirements – additional efforts are required for the formalization activity. However, the capability of formalizing requirement and system model directly improves the quality of the product. Further, we expect that the improved description quality will have a positive aspect for the later development activities in minimizing the number of defects and enabling detection and correction of the defects earlier in the development phase.

V. Conclusion

Tool integration is a key factor to reduce development costs and time of safety–relevant embedded systems while improving the product quality. However, different challenges are arising and the support of different experts is required in order to design, deploy and use such an integrated tool-chain. We proposed in this work a tool-chain demonstrator that highlights several main results of the CESAR project. The tool-chain was presented first from a tool integrator point of view in order to discuss the challenges of control flow (tool integration platform) and data flow (semantic understanding). Second, the tool-chain was presented from the technology provider point of view in order to illustrate main technical achievement of the CESAR project in the domain of requirements engineering and component-based design. Finally, the tool-chain was presented from the end-user point of view in order to analyze the expected benefits of such tool-chains. The main outcomes of the evaluation was (1) the additional efforts at the beginning of the process for formalizing the system, leading however to (2) improved product quality and expected savings during the later development phase (due to test frontloading). Further, (3) the improved traceability along the development process as well as model transformation were also a key factor to improve system quality and development efficiency.

Acknowledgment

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement Nr 100016, the Austrian BMVIT under the program "Forschung, Innovation und Technologie für Informationstechnologien" and from specific national programs and / or funding authorities.

References

[1] E. Armengaud, et al. (2011b). ‘Model-Based Toolchain for the Efficient Development of safety-relevant Automotive Embedded Systems’. In SAE2011 (2011-01-0056).

[2] Y. Papadopoulos and J. A. McDermid. Hierarchically performed hazard origin and propagation studies. In M. Felici, K. Kanoun, and A. Pasquini, editors, SAFECOMP, volume 1698 of Lecture Notes in Computer Science, pages 139–152. Springer, 1999.

[3] D. Chen, R. Johansson, H. Lönn, Y. Papadopoulos, A. Sandberg, F. Törner, and M. Törngren. Modelling support for design of safety critical automotive embedded systems. In Computer Safety, Reliability, and Security, Lecture Notes in Computer Science SAFECOMP2008, 2008.

[4] A. I. Wasserman. Tool integration in software engineering environments. In F. Long, editor, Software Engineering Environments, International Workshop on Environments Proceedings, number 467 in Lecture Notes in Computer Science, pages 137–149. Springer-Verlag, September 1989.

[5] G. Griessnig, et al. (2010). “CESAR: Cost-Efficient Methods and Processes for Safety Relevant Embedded Systems”. Embedded World 2010- ARTEMIS Session.

[6] IEC 61508 Edition 2.0, Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety-Related Systems, part 1-7’, 2010.

[7] ISO/FDIS 26262, Road vehicles – Functional safety, part 1-10,

[8] DO178B---RCTA/DO-178B/ED-12B, "Software Considerations in Airborne Systems and Equipment," Federal Aviation Administration software standard, RTCA Inc., December 1992.

[9] CENELEC (2001): Railway Applications - Software for Railway Control and Protection Systems. EN50128

[10] Farfeleder, Moser, Krall, Stålhane, Zojer and Panis, DODT: Increasing Requirements Formalism using Domain Ontologies for Improved Embedded Systems Development. In DDECS 2011, pp . 271-274. 2011.

[11] Farfeleder, Moser, Krall, Stålhane, Omoronyia and Zojer, Ontology-Driven Guidance for Requirements Elicitation. In ESWC 2011, pp. 212-226. 2011.

[12] Wassermann, A.: Tool Integration in software engineering environments. In The International Workshop on Environments Software Engineering Environments), volume 647 of Lecture Notes in Computer Sciences, pages 137-149, Springer-Verlag, Berlin, September 1989, Chinon, France

[13] Thomas, I., Nejmeh, B.: Definitions of Tool Integration for Environments. IEEE Software, 9(2):29-35, March 1992

[14] OMG Specification: Query/View/Transformation, v1.1, formal/2011-01-01, http://www.omg.org/spec/QVT/1.1/PDF/

[15] Aldazabal, A ., Baily, T., Nanclares, F., Sadovykh, A., Hein, C., Esser, M., Ritter, T. :Automated Model Driven Development Processes; Proceedings of the ECMDA workshop on Model Driven Tool and Process Integration, Fraunhofer IRB Verlag, Stuttgart, 2008, ISBN: 978-3-8167-7645-1

[16] Baumgart, A., Ellen, C., Oertel, M., Rehkop, P.: A reference technology platform with common interfaces for distributed heterogeneous data models; Proceedings of the EmbeddedWorld Conference, Nürnberg, (in press) 2012.

[17] F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev: A Model Transformation Tool, Science of Computer Programming, vol. 72, pp. 31-39, Jun. 2008.

[18] O. Laurent, I. Viglietti, F. Paganelli; S. Bonnet , G. Cristau, N. Priggouris, P. Baufreton : Customization principles of an aeronautics SLM environment and an illustration on aeronautics use cases, the doors management system and the flight control system, ERTS² 2012