Dipl.-Ing. Harald Sporer, BSc.

Mechatronic System Development: an Automotive Industry Approach for Small Teams

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Eugen Brenner

Institute of Technical Informatics
Head: Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Römer

Graz, March 2016

AFFIDAVIT

Abstract

Nowadays 90% of all product innovations in the automotive industry are driven by electronics and software. Up to 40% of a vehicle's development costs are determined by these components. Due to recent major trends in this industry sector, such as autonomous driving and green propulsion systems, there is further scope for this proportion to grow even further. Even a modern low-end car has dozens of electronic control units integrated which are connected to each other through fast communication techniques. Moreover, new driver assistant functions and multimedia applications demand that the car is connected to its environment. Away from the electronics and software view, functional safety has been a major concern for several years now. With the trend of connecting a vehicle to its environment comes the increased importance of security, requiring the development of so-called dependable systems. The definition of sound engineering processes, which is one of the major aspects of this thesis, is a vital basis for the production of these systems.

To cope with the rising overall system complexity, many research projects have been carried out to create adequate methodologies and tools to support the development of electronic systems. Most of them agree on model-driven strategies as best practice when designing the different development artefacts. One of the key challenges is to keep these artefacts consistent and to ensure traceability between them, which is often approached by a virtually automated model-to-model transformation and code generation. Usually the focus of these methodologies is purely on solving the engineering task without considering non-technical aspects such as the size of the development team, even though many of the companies involved in the product development cycle of a modern car can be categorized as small and micro sized enterprises.

The research work described in this thesis aims to remedy this issue and proposes an approach for the facilitation of mechatronic system development within small entities. Thus, the target group includes the previously mentioned small enterprises, but is also suitable for small and micro sized teams within a larger company too. The assumed experience of the team that this pertains to is an existing high level of technical expertise related to their particular product, but little experience with state-of-the-art engineering processes. The support mentioned is the top level goal of the research work, which is divided into the two sub-goals *(i) support by providing a domain-specific modelling* *approach for an easy system design*, and *(ii) support by providing a feasible engineering process reference model.*

Most of the established embedded automotive system design approaches, utilize some kind of UML-based modelling within multi-purpose development environments such as Eclipse. An industrial project revealed that for domain experts, who are not familiar with the UML notation, creating a system architectural design is an awkward task to perform. To enhance this situation, a meta-model for a domain-specific modelling of mechatronic systems has been defined and exemplary implemented for custom-made tool support. Moreover, an integration of this mechatronic system modelling methodology into existing approaches has been shown. This enables, for example, software engineers to design the particular components in a more detailed way using techniques that have already been established.

The presented embedded mechatronic system design methodology also supports the proposed engineering process reference model, which is composed of selected process definitions from the de facto standard Automotive SPICE and of newly defined processes for the hardware related development phase. Aside from the mentioned definitions, sociological aspects of introducing engineering processes to small development entities are discussed and a pattern for establishing the processes is outlined.

Kurzfassung

Produktinnovationen in der automotiven Industrie werden mit einem Anteil von 90% mittlerweile überwiegend von Hardware- und Software-basierten Applikationen getrieben. 40% der Entwicklungskosten eines modernen Fahrzeuges sind diesen Komponenten zuzuordnen. Ein weiterer Anstieg ist auf Grund sich abzeichnender großer Trends, wie etwa autonomes Fahren, absehbar. Bereits Kleinwagen sind mit Dutzenden untereinander kommunizierenden Steuergeräten ausgestattet. Diese Kommunikation findet aber nicht mehr ausschließlich innerhalb des Fahrzeuges statt; die Systemgrenze Fahrzeug verwischt durch zahlreiche neue Funktionen immer mehr mit ihrer Umwelt. Durch die Realisierung von sicherheitskritischen Funktionen mittels elektronischer Komponenten, ist die funktionale Sicherheit seit vielen Jahren ein wesentlicher Faktor in der Automobilindustrie. Die Vernetzung des Fahrzeugs bringt zudem das Thema Datensicherheit mit sich und verlangt die Entwicklung einer neuen Generation von zuverlässigen Systemen. Für deren Entwicklung ist die Implementierung von wohldefinierten Engineering Prozessen unabdingbar, was eines der zentralen Themen dieser Arbeit darstellt.

Um mit der weiter ansteigenden Komplexität in der Entwicklung umgehen zu können, wurden zahlreiche Forschungsprojekte hinsichtlich neuer Methoden und Werkzeuge durchgeführt. Im Bereich der elektrischen und elektronischen Systeme werden Modell-basierte Entwicklungsmethoden weitestgehend als bester Ansatz gesehen. Typischerweise fokussieren die in den Forschungsarbeiten dargestellten Ansätze auf rein technische Aspekte, ohne weitere Einflussfaktoren wie etwa die Größe und Zusammensetzung des Entwicklungsteams, welches die Methoden und Werkzeuge verwenden soll, in Betracht zu ziehen.

Die vorliegende Arbeit versucht hier Abhilfe zu schaffen und zeigt mögliche Ansätze, welche die Entwicklung von mechatronischen Systemen speziell für kleine Entwicklungsteams erleichtern soll. Die Zielgruppe dieser Forschungsarbeit stellen jene Unternehmen bzw. Teams dar, die ein hohes Maß an technischer Expertise hinsichtlich ihres Produktes, jedoch wenig Erfahrung mit Engineering Prozessen aufweisen. Die Unterstützung dieser Kategorie von Entwicklungseinheiten ist gleichzeitig das vorangestellte Ziel dieser Arbeit, welches in die beiden Teilziele *(i) Vereinfachung des System Designs durch einen Domänen-spezifischen Modellierungsansatz*, und *(ii) Spezifikation eines Referenzmodells für erforderliche Engineering Prozesse*, aufgeteilt wird.

Existierende Ansätze für das Design von eingebetteten automotiven Systemen basieren meist auf einer Art UML-Notation. Ein durchgeführtes Industrieprojekt hat gezeigt, dass das Erstellen eines Designs innerhalb solcher Umgebungen, für Domänen-Experten ohne vertieftes Wissen im Bereich der UML-Modellierung eine meist mühsame Aufgabe darstellt. Zur Verbesserung dieser Situation definiert die vorliegende Arbeit ein Meta-Modell für die Domänen-spezifische Modellierung von mechatronischen Systemen und stellt dieses anhand einer beispielhaften Implementierung vor. Darüber hinaus wird eine mögliche Integration der aufgezeigten Methoden in bereits bestehende Ansätze präsentiert. Dies ermöglicht unter anderem eine Detailbeschreibung der mittels Domänen-spezifischer Sprache beschriebenen Komponenten in einer bereits etablierten Umgebung.

Die vorgestellte Methodik zum Erstellen des Designs von eingebetteten mechatronischen Systemen unterstützt das vorgeschlagene Engineering Prozess Referenz Modell, welches sich aus ausgewählten Prozessdefinitionen des Industriestandards Automotive SPICE, sowie innerhalb dieser Arbeit neu definierten Hardware-Entwicklungsprozessen zusammensetzt. Neben den zuvor genannten Definitionen werden soziologisch relevante Aspekte hinsichtlich der Entwicklungstätigkeiten kleiner Einheiten diskutiert und unter Berücksichtigung aller Faktoren ein Pattern zur Einführung der aufgezeigten Prozesse vorgeschlagen.

Danksagung

Im Laufe der letzten Jahre, in denen die vorliegende Dissertation entstand, gab es viele Höhen und eine ausreichende Anzahl an Tiefen zu verzeichnen, was vermutlich dem natürlichen Verlauf einer wissenschaftlichen Arbeit entspricht. Besonders in den anspruchsvolleren Zeiten hatte ich das große Glück, mich auf meine Familie, Freunde und Kollegen stützen zu dürfen. Vielen Dank für all die Geduld und die unzähligen motivierenden Worte über viele Jahre hinweg an meine Familie - Friederike, Johann, Johannes und Nina - und meine lieben Freunde.

Besonderer Dank gilt meinem Betreuer und Mentor Prof. Dr. Eugen Brenner für seine tatkräftige Unterstützung, sowohl in fachlichen als auch administrativen Belangen. Mit viel Freiraum zur Entfaltung eigener Ideen und Ansätze hat er mir auf großartige Art und Weise neue Blickwinkel auf die unterschiedlichsten Herausforderungen nähergebracht. Des Weiteren möchte ich mich bei Dr. Christian Kreiner bedanken, der mir Einblicke in höchstinteressante Arbeitsgruppen ermöglichte und mich stets zum richtigen Zeitpunkt aus meiner Komfortzone holte.

Vielen Dank für all die Unterstützung!

Extended Abstract

Modern passenger cars can be seen as highly complex compositions of powerful microprocessor-based embedded systems on wheels. Many engineering challenges which are drawn from a wide range of domains are combined in the automotive industry. That is, electronic components in cars are often exposed to adverse environmental conditions such as moisture or heat, and electromagnetic interference produced by high power inductive loads. Their reliability and availability have to be high, they are expected to be both safe and secure, they must take up little space and obviously these systems have to be cheap. To establish these desired properties, huge efforts along the embedded system development are needed. According to [3], up to 40% of a vehicle's development costs are actually determined by these hardware and software components. Due to major trends, such as autonomous driving and green propulsion systems, a further increase in this proportion in the next few years is most likely.

The range of numbers of electronic control units (ECUs) integrated in modern vehicles starts at a few dozen in the case of low-end cars and tot up to about 100 pieces in the luxury class category, with millions of lines of code implemented. Alfred Katzenbach, former director of Information Technology Management at Daimler, mentioned that in the fifth generation of the S-class Mercedes-Benz the radio and navigation system requires more than 20 million lines of code [12], which is about 3 times the amount of code used to operate the avionics and onboard support systems of the actual Boeing 787 Dreamliner.

Due to the shift of safety-relevant mechanics-based functionality to mechatronic-based systems, functional safety has been one of the major concerns over the last few years. New functionality, such as that needed for novel advanced driver assistant systems (ADAS) and for linking smart devices to a car's infrastructure, requires the engineers to examine security threats carefully. The combination of the attributes of availability, safety, and security leads to a new generation of systems to be developed, often referred to as *dependable systems*. As a basis for the development of mechatronic products that fulfil the mentioned properties, the implementation of sound engineering processes at the particular company or team, which is one of the major aspects of this thesis, is absolutely vital.

Many research projects have been carried out to create adequate methodologies and

tools to cope with the overall system complexity of the development of modern passenger cars. Usually the focus of these projects is purely on solving the engineering task without considering non-technical aspects such as the size of the development team, even though many of the companies involved in the product development cycle of a modern car can be categorized as small and micro sized enterprises. This thesis attempts to remedy this issue by setting its top level goal as *facilitating the embedded mechatronic system development carried out by small entities.* Thus, the target group includes the previously mentioned small enterprises, but is supposed to be suitable for small and micro sized teams within a larger company too. The assumed maturity of the team that is supported is an existing high level of technical expertise related to their particular product, but little experience with state-of-the-art engineering processes. In relation to the mentioned problem statement and the top-level goal, two major topics have been identified and defined as sub-goals: *(i) providing support by specifying a feasible engineering process reference model*, and *(ii) providing support by creating a domain-specific modelling approach for easy system design.*

The proposed engineering process reference model (PRM) is composed of selected process definitions from the de facto standard Automotive SPICE [58] and of newly defined processes, which mainly represent a hardware development related extension to the established definitions. In Figure 0.1 an overview of the new PRM is illustrated. The green coloured elements depict the novel components of the process definitions that are introduced in the course of this work. In addition to the hardware related topics, processes for *Mechanical Requirements Analysis* and *System Validation* have been introduced. Aside from the PRM definition, sociological aspects of introducing engineering processes to small development entities are discussed and a pattern for establishing the processes is outlined.

For an appropriated system design, most of the existing approaches agree on model-driven strategies as best practice. Within the group of model-based design methodologies, typically UML or a form of UML derived modelling language is utilized. Extending UML-based diagrams through profiles, which are defined by using stereotypes, constraints, and tag definitions, is a widely used technique. New model elements are derived from existing ones and equipped with the specified attributes. An industrial project revealed that for domain experts, who are not familiar with the UML notation, creating a system architectural design most likely results in an awkward task. To enhance this situation, a meta model for a domain-specific modelling of mechatronic systems has been defined, which is illustrated in Figure 0.2. The meta model of the embedded mechatronic system domain-specific modelling (EMS-DSM) provides different components that are required for a system architectural design. Additionally, basis and application software

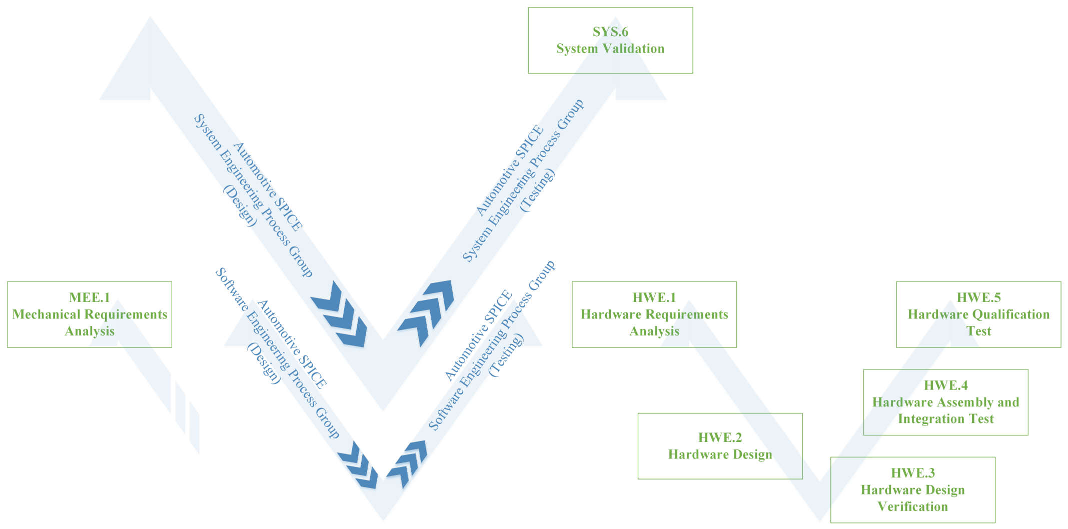

SYS.6
System Validation
Automotive SPICE
System Engineering Process Group
(Design)
Automotive SPICE
System Engineering Process Group
(Testing)
MEE.1
Mechanical Requirements Analysis
Automotive SPICE
Software Engineering Process Group
(Design)
Automotive SPICE
Software Engineering Process Group
(Testing)
HWE.1
Hardware Requirements Analysis
HWE.5
Hardware Qualification Test
HWE.4
Hardware Assembly and Integration Test
HWE.2
Hardware Design
HWE.3
Hardware Design Verification

Figure 0.1: Novel engineering process definitions aligned to Automotive SPICE [58]

components are defined to enable the design of a software architecture. The properties of each component are derived from the meta model's top node *EMS-DSM Component*. Further properties are added for the particular components as required.

Special attention has been paid to the integration of the proposed methodology into existing approaches. Consistency and bidirectional traceability as key aspects of the previously described PRM, are fully supported by the EMS-DSM definition. Exemplary implementations of the meta model are presented in the course of this research work, which showcases the claimed traceability and also requirements management approaches that are feasible for small entities. Further methodologies are proposed that allow the transformation of the system and software architectural design created with EMS-DSM, into the previously mentioned UML-based representation or directly into models belonging to the software detailed design, such as Simulink models.

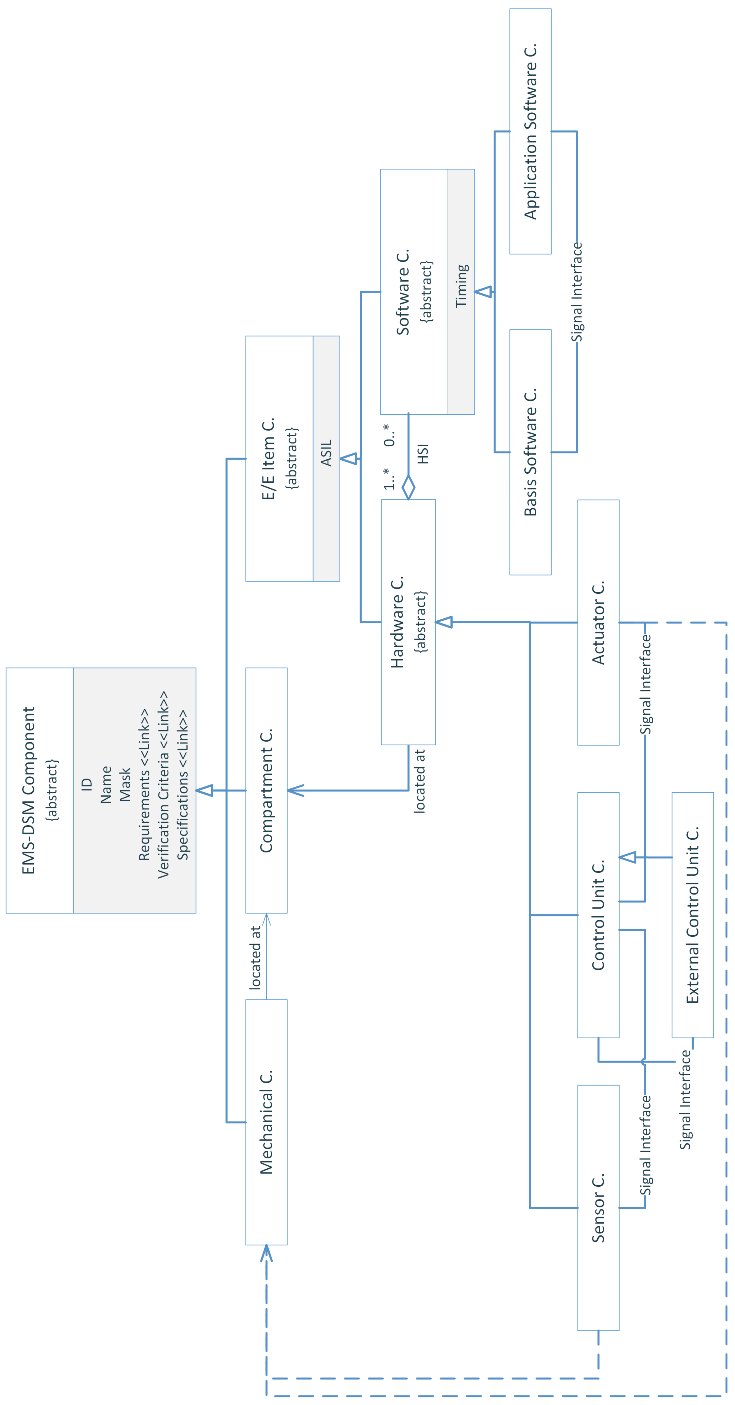


Figure 0.2: Definition of the *Embedded Mechatronic System Domain-Specific Modelling (EMS-DSM)* Meta Model

Contents

List of Figures

List of Tables

Acronyms

ADAS Advanced Driver Assistance System

AQUA Knowledge Alliance for Training Quality and Excellence in Automotive

ARIS Architecture of Integrated Information Systems

ASPICE Automotive Software Process Improvement and Capability Determination

AUTOSAR Automotive Open System Architecture

BP Base Practice

BPMN Business Process Model and Notation

CAD Computer-Aided Design

CAM Computer-Aided Manufacturing

CASE Computer-Aided Software Engineering

CMMI Capability Maturity Model Integration

DSL Domain-Specific Language

DSM Domain-Specific Modelling

E/E System Electrical and/or Electronic System

EAST-ADL Electronics Architecture and Software Technology - Architecture Description Language

EEA Electric/Electronic Architecture

ECU Electronic Control Unit

EMF Eclipse Modeling Framework

EMS Embedded Mechatronic System

EU	European Union
GEF	Graphical Editor Framework
GME	Generic Modeling Environment
GMF	Graphical Modeling Framework
GMP	Graphical Modeling Project
GOPRR	Graph, Object, Property, Relationship, and Role
GP	Generic Practice
GPL	General Purpose Language
HIS	Herstellerinitiative Software
HSI	Hardware-Software Interface
MARTE	Modeling and Analysis of Real-time Embedded Systems
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MEE	Mechanical Engineering
MSDN	Microsoft Developer Network
OCL	Object Constraint Language
OEM	Original Equipment Manufacturer
OMG	Open Management Group
PES	Programmable Electronic System
PRM	Process Reference Model
R&D	Research and Development
RSA	Rational Software Architect
SBVR	Semantics of Business Vocabulary and Rules
SDK	Software Development Kit

SME	Small and Medium Sized Enterprise
SMiE	Small and Micro Sized Enterprise
SOA	Service Oriented Architecture
SysML	Systems Modeling Language
UML	Unified Modeling Language
V&V	Verification and Validation
VSM	Viewpoint Specification Model

1 Introduction

1.1 Motivation

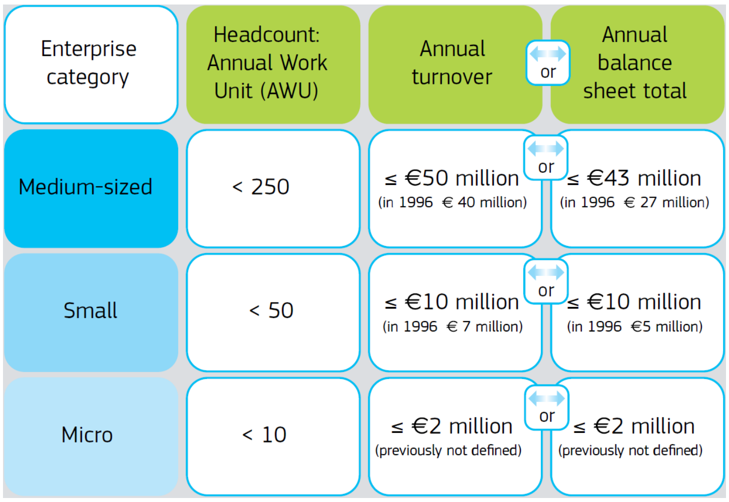
On a sunny day in July 2012 a meeting took place between the Aston Martin's Special Projects department, Alset GmbH, and the Institute of Technical Informatics from Graz University of Technology, at Aston Martin's headquarters in Gaydon, UK. The appointment led to a very unique project named *AML-24HR*, which was kicked-off only a few weeks later. The overall goal of this project was defined as *developing a hydrogen bi-fuel race car and participating with this car at the 24-hour race at the Nürburgring in May 2013*. Within 10 months the new propulsion system had been developed, installed and tested. *New propulsion system* in this context means a conventional internal combustion engine running on gasoline, retrofitted to an engine which is able to operate with both kinds of fuel, gasoline and hydrogen. Due to the very short project duration it has to be admitted that the longest test run was the 24 hour race itself. Nevertheless, the project was a full success and for the first time in history a hydrogen racing car finished the 24 hour race at the famous Green Hell.

Figure 1.1: 2013 Aston Martin Hydrogen Race Car

The way the AML-24HR project was carried out is a very good example for many other development processes within small teams. Due to the characteristic of such small-scale teams there are a lot of assets, but also many drawbacks during an automotive prototype or product development. The intention of the research that led to this thesis has been to analyse these advantages and disadvantages and to provide an approach for a more sustainable development for small project teams or companies.

1.2 Small and Micro Sized Entities

According to the *SME User Guide* from the European Commission [20] small and medium sized enterprises (SMEs) are the engine of the European economy. Nine out of every ten enterprises is an SME and they generate two out of every three jobs. In 2013 the total number of this type of enterprise aggregated to over 21 million companies and they provided close to 90 million jobs within the EU. The SME definition takes the criteria *Staff Headcount*, *Annual Turnover*, and *Annual Balance Sheet Total* into account. To be classified as an SME it must employ less than 250 persons and have either an annual turnover not exceeding 50 million euro or an annual balance sheet total not exceeding 43 million euro. The gradation within the category SME related to the outlined criteria is illustrated in Figure 1.2.



Enterprise category	Headcount: Annual Work Unit (AWU)	Annual turnover		Annual balance sheet total
Medium-sized	< 250	≤ €50 million (in 1996 € 40 million)	or	≤ €43 million (in 1996 € 27 million)
Small	< 50	≤ €10 million (in 1996 € 7 million)	or	≤ €10 million (in 1996 €5 million)
Micro	< 10	≤ €2 million (previously not defined)	or	≤ €2 million (previously not defined)

Figure 1.2: SME Definition Thresholds [20]

A rigorous definition of this thesis' target group, in terms of the enterprise categories introduced by the European Commission, is not reasonable. This research work proposes an approach for the facilitation of mechatronic system development within small *entities*. Thus, the target group includes small and micro sized enterprises as defined by the *SME User Guide*, but the approach is supposed to be suitable for small and micro sized teams within a larger company too. More important than the exact number of employees or the annual turnover, is the current maturity level of the entity in terms of engineering processes. The methodologies and techniques described in this work focus on teams that have a high level of technical expertise related to their particular product, but little experience with state-of-the-art engineering processes. Typically start-up companies show these characteristics, but also enterprises which have been on the market for a long period of time and want to capture new business fields related to mechatronic products, may be supported by the ideas proposed in this thesis.

1.3 Challenges for Small Entities in the Automotive Industry

Nowadays 90% of all product innovation in the automotive industry is driven by electronics and software. Up to 40% of a vehicle's development costs are determined by these components [3]. Due to recent major trends in this industry sector, such as autonomous driving and green propulsion systems, there is still scope for this percentage to increase. Even a modern low-end car has dozens of integrated electronic control units (ECUs) that are connected to each other using fast communication techniques. Furthermore, premium cars can run with around 100 million lines of code, executed on about 100 microprocessor-based electronic control units, that's 15 times the amount of software implemented in the ECUs of a Boeing 787 Dreamliner. The current S-class Mercedes-Benz requires over 20 million lines of code for the radio and navigation system alone. Experts from the automotive industry predict that the software in cars will continue to grow in both amount and complexity [12]. Moreover, new driver assistant functions and multimedia applications demand that the car is connected to its environment. Away from the electronics and software view, functional safety has been a major concern for several years now, but with the trend of connecting the vehicle with its environment, security also becomes more important and requires the development of so-called dependable systems.

The ever growing amount of software and complexity in modern passenger cars is obviously a challenge for all parties involved in the development cycle of an embedded automotive system, irrespective of a particular company's size. But typically small and micro sized entities are not supposed to develop a complete car on their own. Instead, these small teams most likely have a high level of expertise in developing a certain component, which may be delivered to a tier-1 supplier or an original equipment manufacturer

(OEM).

In the author's opinion, the small entities involved in the vehicle development cycle should be aware of the mentioned complexity but do not have to cope with it directly. Usually their customer (tier-1 or OEM) is responsible for integrating the system and has to breakdown the complexity through well-defined requirements and interfaces for the particular product. The definition of sound engineering processes, which is one of the major aspects of this thesis, is crucial for being capable of delivering the mentioned components and the related embedded automotive electrical and/or electronic systems (E/E systems). Moreover, it is a vital basis for the development of the previously mentioned dependable systems. Hence, one of the major challenges for small entities, as they are characterized in 1.2, is to acquire a strategy to quickly enter into the world of engineering processes. Unfortunately, this is not done with selecting an appropriated process reference model. A for all parties involved practicable support, e.g. by means of tools, has to be established.

Many research projects have been carried out to propose such adequate methodologies and tools to support the development of electronic systems. Most of them agree on model-driven strategies as best practice when designing the different development artefacts. One of the key challenges is to keep these artefacts consistent and to assure traceability between them, which is often approached by a virtually automated model-to-model transformation and code generation. Usually the focus of these methodologies is purely on solving the engineering task without considering non-technical aspects such as the size of the development team. The research work described in this thesis aims to remedy this issue and proposes an approach for the facilitation of mechatronic system development within small entities.

1.4 The Goals of this Thesis

The major goal of this thesis and the work that led to it can be stated as: small and micro-sized embedded system development entities, which are entering into the engineering process world, shall be equipped with some ideas that make their challenging life within a fast, complicated and sometimes complex domain easier. The author does not claim to provide a number of methodologies for the embedded automotive industry, which will solve the day to day development problems for teams or companies involved in this business domain, but hopes to facilitate the production of one or another development artefacts through the proposed approach. The published methodologies and techniques reflect the conclusion the author draws from the last few years that he spent in various R&D projects, most of the time in very small teams. The range of items that were developed at these projects is a broad one, from small stand-alone control units, just processing a handful of sensor data, to complete novel propulsion systems. The latter type of projects led to great results as illustrated in Figure 1.1.

1.5 Thesis Organization

The structure of this thesis is as follows. Chapter 2 discusses the work related to this thesis' key aspects *Standards and Process Reference Models*, *Model-Driven Design Methodologies*, and *Domain-Specific Modelling*, all from the view of automotive E/E system development. In Chapter 3 an approach to the *Embedded Mechatronic System Development*, with a special focus on small entities, is described. Thoughts on sociological and engineering aspects are outlined before engineering processes, as an extension to an established process reference model, are defined. The chapter concludes with a definition of a pattern for establishing engineering processes in small development entities. Chapters 4 and 5 briefly introduce the definition of the domain-specific modelling meta model and exemplary applications of the meta model. Moreover, both chapters establish references to detailed information in the particular author's publications related to the outlined methodologies and techniques. These publications are presented in Chapter 7 and their allocation to the thesis' goals is outlined in Figure 1.3. In Chapter 6 the thesis concludes with a brief summary of the presented approach and a description of potential future work.

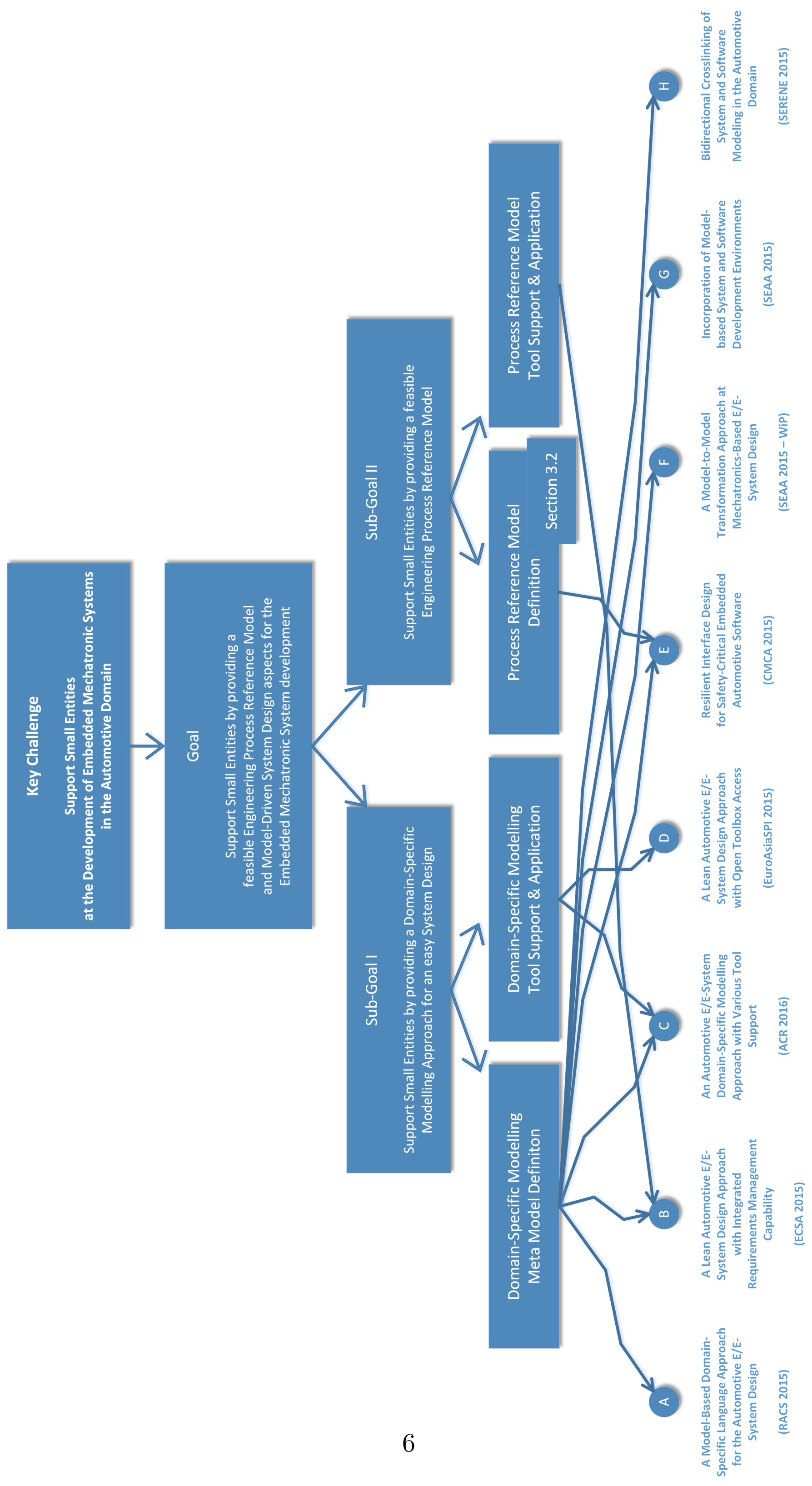

Key Challenge
Support Small Entities
at the Development of Embedded Mechatronic Systems
in the Automotive Domain
Goal
Support Small Entities by providing a
feasible Engineering Process Reference Model
and Model-Driven System Design aspects for the
Embedded Mechatronic System development
Sub-Goal I
Support Small Entities by providing a Domain-Specific
Modelling Approach for an easy System Design
Sub-Goal II
Support Small Entities by providing a feasible
Engineering Process Reference Model
Domain-Specific Modelling
Meta Model Definiton
Domain-Specific Modelling
Tool Support & Application
Process Reference Model
Definition
Section 3.2
Process Reference Model
Tool Support & Application
A
A Model-Based Domain-
Specific Language Approach
for the Automotive E/E-
System Design
(RACS 2015)
B
A Lean Automotive E/E-
System Design Approach
with Integrated
Requirements Management
Capability
(ECSA 2015)
C
An Automotive E/E-System
Domain-Specific Modelling
Approach with Various Tool
Support
(ACR 2016)
D
A Lean Automotive E/E-
System Design Approach
with Open Toolbox Access
(EuroAsiaSPI 2015)
E
Resilient Interface Design
for Safety-Critical Embedded
Automotive Software
(CMCA 2015)
F
A Model-to-Model
Transformation Approach at
Mechatronics-Based E/E-
System Design
(SEAA 2015 – WiP)
G
Incorporation of Model-
based System and Software
Development Environments
(SEAA 2015)
H
Bidirectional Crosslinking of
System and Software
Modeling in the Automotive
Domain
(SERENE 2015)

Figure 1.3: Allocation of the Publications to the Thesis' Goals

2 Related Work

In this chapter, an analysis of work related to the thesis' key topics *Process Reference Models*, *Automotive E/E-System Design Methodologies*, and *Domain-Specific Modelling* is conducted. Additionally, these topics are considered from the view of small development teams and companies in the business area of automotive mechatronic systems.

2.1 Standards & De Facto Standards

Small entities, in particular start-up companies, do have specific needs regarding the introduction of quality management and engineering processes. Often the need for this kind of *overhead* is not visible for them and the clarification of technical aspects, e.g. how the implementation of a prototype could look in detail, are classified with the highest priority. It is by their nature that engineers most likely try to solve technical problems as quickly as possible, but with the development of complex systems, design methodologies are required in order to be capable of delivering quality products.

2.1.1 Process Reference Models (PRMs)

On the basis of their broad dissemination, the two most important reference models in the automotive industry are *Capability Maturity Model Integration (CMMI)* [53] and *Automotive Software Process Improvement and Capability Determination (ASPICE)* [58]. Both pursue similar targets, which can be stated as

(1) determining the process capability & maturity, and

(2) to achieve continuous process improvement in the particular development entity.

Instead of specifying how the described processes have to be implemented, the reference models define desired *Process Outcomes* (ASPICE) or *Goals* (CMMI). These definitions are characterized in more detail by best practices (*Base* or *Generic Practices* at ASPICE, and *Specific* or *Generic Practices* at CMMI).

ASPICE is widely used in Europe, especially by the German car manufacturers, as well as in some parts of Eastern Asia. ASPICE is based on the international standard

ISO 15504 [1], which consists of ten[1] parts and provides a framework for the assessment of processes in the field of information technology.

The CMMI reference model has been developed by the Software Engineering Institute (SEI) at the Carnegie Mellon University. Different parts of CMMI are available for the topics *Acquisition*, *Development* and *Services*. Owing to the fact that CMMI is not wide spread in the European automotive industry, this work focuses on Automotive SPICE (latest version 3.0, released in July 2015) as the relevant process reference model.

The Automotive SPICE process reference model contains on one hand the definition of the processes including the related *Base Practices (BPs)*, and on the other hand the definition of *Generic Practices (GPs)* which are valid for all processes. If the goals of the BPs of a particular process are fulfilled and the required work products are created, capability level 1 can be achieved through an assessment of that single process (the relationship between the process and capability dimensions of ASPICE is illustrated in Figure 2.1). To achieve higher capability levels for the particular process, the GPs of the certain level have to be implemented.

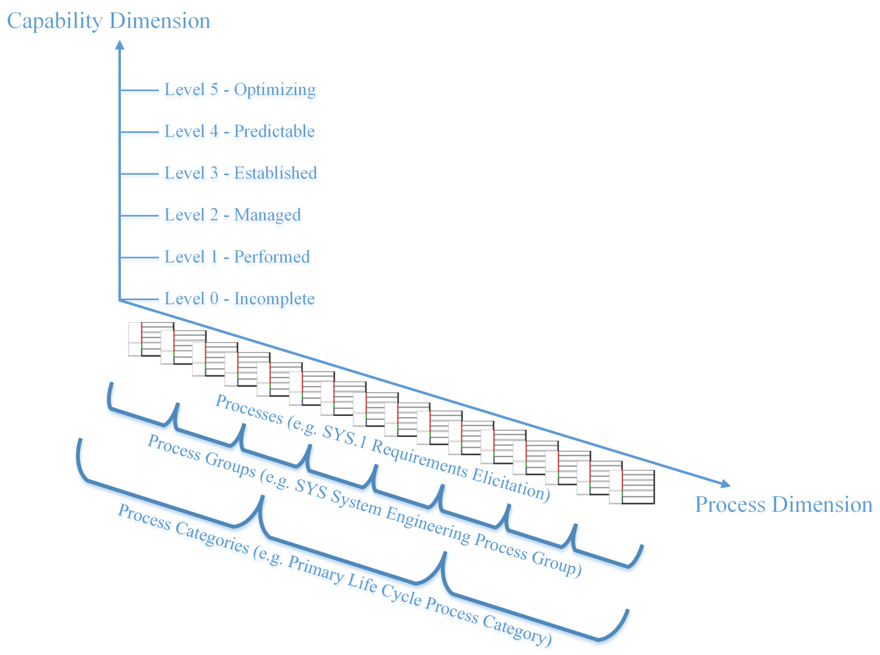

Capability Dimension
Level 5 - Optimizing
Level 4 - Predictable
Level 3 - Established
Level 2 - Managed
Level 1 - Performed
Level 0 - Incomplete
Processes (e.g. SYS.1 Requirements Elicitation)
Process Groups (e.g. SYS System Engineering Process Group)
Process Categories (e.g. Primary Life Cycle Process Category)
Process Dimension

Figure 2.1: Process and Capability Dimensions in Automotive SPICE [7]

[1]The first part ISO/IEC 15504-1:2004 has been revised by ISO/IEC 33001:2015 [4]

Though the *Herstellerinitiative Software (HIS)*, a joint effort of Audi, BMW, Daimler, Porsche and Volkswagen for consistent standards in the field of automotive software, states that the required capability level is not defined, typically an average level of 2 to 3 is demanded if a company wants to act as a supplier and deliver their products. It has to be noted that this demand is not related to all processes in the reference model. HIS provides a list of processes to which their supplier should pay special attention. The selection of these processes is called *HIS Scope* and can be found at [21]. Due to the fact that this list is several years old it references the previous version of Automotive SPICE. The information has been mapped to the processes of the new ASPICE version 3.0 by the author of this thesis and is shown in Table 2.1.

Table 2.1: HIS Scope mapped to ASPICE 3.0, inspired by [21]

Management ProcessGroup (MAN)		*System Engineering Process Group (SYS)*	
MAN.3	Project Management	SYS.2	System Requirements Analysis
Supporting Process Group (SUP)		SYS.3	System Architectural Design
SUP.1	Quality Assurance	SYS.4	System Integration and Integration Test
SUP.8	Configuration Management	SYS.5	System Qualification Test
SUP.9	Problem Resolution Management	*Software Engineering Process Group (SWE)*	
SUP.10	Change Request Management	SWE.1	Software Requirements Analysis
Acquisition ProcessGroup (ACQ)		SWE.2	Software Architectural Design
ACQ.4	Supplier Monitoring	SWE.3	Software Detailed Design and Unit Construction
		SWE.4	Software Unit Verification
		SWE.5	Software Integration and Integration Test
		SWE.6	Software Qualification Test

From the view of the various process groups in Automotive SPICE, this thesis has an emphasis on the engineering process groups which include the *System Engineering Process Group (SYS)* and the *Software Engineering Process Group (SWE)*. Other process groups and the particular processes may be considered implicitly, e.g. the *Change Request Management* process in the *Supporting Process Group (SUP)*, but no special attention will be paid. Due to this thesis' focus on small- and micro-sized development teams, a context where low process model implementation levels are assumed, the goal is to support the achievement of ASPICE level 1 for the considered processes through

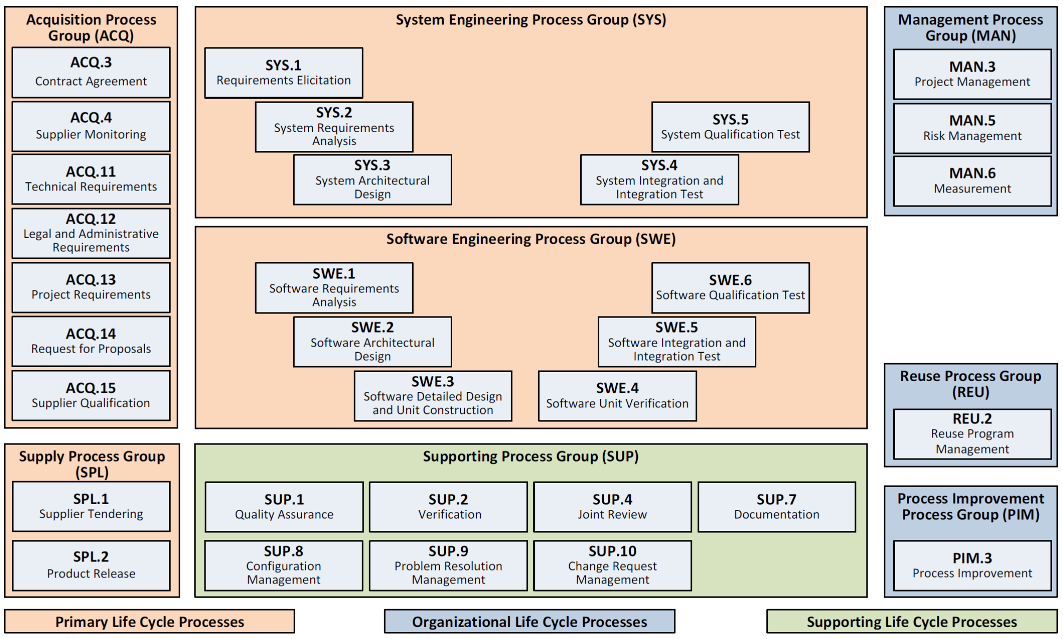

Acquisition Process Group (ACQ)
ACQ.3 Contract Agreement
ACQ.4 Supplier Monitoring
ACQ.11 Technical Requirements
ACQ.12 Legal and Administrative Requirements
ACQ.13 Project Requirements
ACQ.14 Request for Proposals
ACQ.15 Supplier Qualification
System Engineering Process Group (SYS)
SYS.1 Requirements Elicitation
SYS.2 System Requirements Analysis
SYS.3 System Architectural Design
SYS.4 System Integration and Integration Test
SYS.5 System Qualification Test
Software Engineering Process Group (SWE)
SWE.1 Software Requirements Analysis
SWE.2 Software Architectural Design
SWE.3 Software Detailed Design and Unit Construction
SWE.4 Software Unit Verification
SWE.5 Software Integration and Integration Test
SWE.6 Software Qualification Test
Management Process Group (MAN)
MAN.3 Project Management
MAN.5 Risk Management
MAN.6 Measurement
Reuse Process Group (REU)
REU.2 Reuse Program Management
Supply Process Group (SPL)
SPL.1 Supplier Tendering
SPL.2 Product Release
Supporting Process Group (SUP)
SUP.1 Quality Assurance
SUP.2 Verification
SUP.4 Joint Review
SUP.7 Documentation
SUP.8 Configuration Management
SUP.9 Problem Resolution Management
SUP.10 Change Request Management
Process Improvement Process Group (PIM)
PIM.3 Process Improvement
Primary Life Cycle Processes
Organizational Life Cycle Processes
Supporting Life Cycle Processes

Figure 2.2: Automotive SPICE PRM [58]

the presented methodologies and techniques.

2.1.2 Functional Safety

In the late 1990s the *International Electrotechnical Commission* released the first version of the IEC 61508. The revised second edition was published in April 2010. This standard sets out a generic approach for all activities related to the safety lifecycle of systems comprised of *electrical and/or electronic and/or programmable electronic components (electrical/electronic/programmable electronic systems (E/E/PESs))* that are used to perform safety functions [2]. Owing to the fact that the IEC 61508 is too generic for many applications, it has been adopted for particular domains, such as the automotive sector, and released as independent standards for these domains. For the application sector of *electrical and/or electronic (E/E) systems* within road vehicles up to 3.5 tons, the ISO 26262 was published in 2011 [26]. Part 2 of the ISO 26262 (*Management of functional safety*) [25] requires the organisation that is involved in the execution of the safety life cycle to have an operational quality management system complying with a relevant standard like ISO 9001 or ISO/TS 16949. This *safety life cycle* not only comprises of the concept and development phases but also the product life time after the release for production until decommissioning. Figure 2.3 illustrates the product life cycle according to the safety standard.

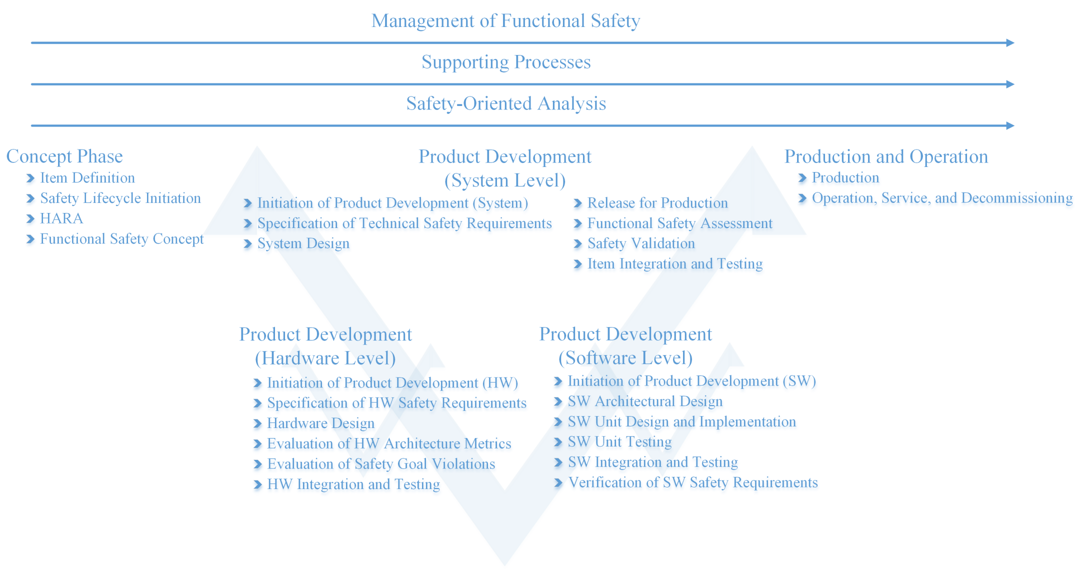

Management of Functional Safety
Supporting Processes
Safety-Oriented Analysis
Concept Phase
Item Definition
Safety Lifecycle Initiation
HARA
Functional Safety Concept
Product Development (System Level)
Initiation of Product Development (System)
Specification of Technical Safety Requirements
System Design
Release for Production
Functional Safety Assessment
Safety Validation
Item Integration and Testing
Production and Operation
Production
Operation, Service, and Decommissioning
Product Development (Hardware Level)
Initiation of Product Development (HW)
Specification of HW Safety Requirements
Hardware Design
Evaluation of HW Architecture Metrics
Evaluation of Safety Goal Violations
HW Integration and Testing
Product Development (Software Level)
Initiation of Product Development (SW)
SW Architectural Design
SW Unit Design and Implementation
SW Unit Testing
SW Integration and Testing
Verification of SW Safety Requirements

Figure 2.3: E/E System Product Safety Life Cycle [25]

Functional Safety and ISO 26262 only have an indirect relevance for this thesis. The focus is clearly to support the initial phase of engineering process implementation at small entities, which is the basis for safety relevant product development. Functional safety (ISO 26262 & IEC 61508) related publications, training and other material may be obtained from other projects, such as *SafEUr* [38]. All methodologies and patterns presented in this work are aligned to safety product life cycle development artefacts wherever possible, but as mentioned previously, functional safety is not a fundamental aspect of this thesis.

2.1.3 Integrated View of Related Standards

Kreiner et al. [33] mention that *Automotive SPICE*, *Functional Safety*, and *Lean Six Sigma* form the quality backbone in the automotive industry. They present an integrated view of these three standards/de facto standards, which is gained by extracting the essence of specific topics, such as *Capability* and *Product Life Cycle*, from each of them and by grouping the information into modules, so called *Base Layer Modules*. Further, an interaction layer was developed within the research project, which is titled *Knowledge Alliance for Training Quality and Excellence in Automotive (AQUA)* and funded with support from the European Commission. This layer acts as a linking module between the base layer modules and the established knowledge related to the particular module, which can be found in standards and other materials. As indicated by the research project's title, one of the major aspects is *Training*. Therefore, the acquired knowledge has been prepared as learning material and courses has been conducted with participants from the European automotive industry.

An integrated view of the intertwined methodologies of *Automotive SPICE*, *Functional Safety*, and *Lean Six Sigma* is a good approach to coping with the complexity of introducing these standards/de facto standards. From the view of this work, relevance is given in a way that the herein proposed methodologies and techniques can be seen as the basis for safety related development and for integrating advanced tools from *Lean Six Sigma* such as the *Design Failure Mode and Effect Analysis (DFMEA)*. As mentioned in Section 2.1.2, the approach described in this thesis is aligned to *Functional Safety* in terms of facilitating the extension of the methodologies and techniques towards safety related development artefacts (incorporation of different levels of safety requirements, defining a hardware-software interface with integrated safety attributes, etc.).

2.2 Model-Driven Architecture

Before digging into the design methodologies it has to be stated that this research work focuses on model-driven architecture wherever applicable. That does not imply that the approaches described in Chapters 3 and 4 cannot be applied to non-model-driven development, but to gain the most benefit from the ideas in this contribution, modelling is highly recommended.

Effective arguments for model-driven development (MDD) can be found in many sources. Selic [52] states that models help us in understanding complex problems and their possible solutions through abstraction. He sees MDD as potentially being the first true generational leap in software development since the introduction of compilers. A key characteristic of the MDD related methods is their reliance on automation and the advantages that it brings. He further mentions that models are usually by far less bound to the underlying implementation technology compared to the most popular programming language. Therefore models are much closer to the problem domain and this makes them easier to specify so that in some cases it is possible for domain experts rather than computing technology experts to create systems. The latter is especially important for the research work presented in this thesis. In small development entities that want to create some kind of embedded mechatronic system, computing technology experts with extensive knowledge of e.g. UML are often not available. Therefore the system modelling environment should be prepared to such an extent that domain experts without special modelling language knowledge are also capable of creating models. Besides the described benefits from Selic, he points out that model-driven development methods are only as good as the models they help us construct. Moreover, he states that the full benefits of MDDs can only be achieved, if their potential for automation is fully exploited (e.g. automatically generating complete programs from models, and automatically verifying the models on the computer).

Broy et al. [11] provide further benefits of the model-based design with a focus on the field of embedded automotive software systems. In a survey, experts reported a simplification in communication because of the use of function models in the software design. The communication within as well as outside the software development department, including also colleagues who are not familiar with software engineering, was experienced as much simpler due to the use of models. One interesting reason, which has nothing to do with a primarily advantage, for using model-based development is according to 83% of the individuals surveyed the trend in the industry to develop model-based. Suppliers in particular mentioned that the *Original Equipment Manufacturers (OEMs)* demand them to develop model-based, otherwise someone else would get the contract. Besides the development aspects the authors also analysed the influence of model-based software design on maintenance costs. Broy et al. surveyed reduced maintenance costs of 15% on average compared to a classical development.

Attempts has been made to stick to the *Model-Driven Architecture (MDA) Guide* [45] of the *Object Management Group (OMG)* during the creation of the herein presented approach wherever possible. The guide explains the major concepts of MDA in a compact but nonetheless comprehensive way. By defining the most important terms and their role within the model-driven architecture methodology, a framework for preparing modelling languages and models for nearly every domain, for different architectural layers (e.g. logical system models), and for various viewpoints is provided. The main feature of *Model-Driven Architecture* is the definition of the model's notation, structure and semantics using industry standards. Models that comply with these definitions are called *MDA Models* and can be used for the creation of documentation, specifications and other development artefacts (e.g. source code). The OMG claims that MDA models facilitate the handling of the complexity of large systems as well as the cooperation between companies and people. One of the most important properties of MDA models is that information can be (automatically) extracted and utilized to prepare other development artefacts, including different representations for various stakeholders (*Model Transformation and Execution, Viewpoints).* In terms of modelling languages OMG tends to favour UML and its derivations but also does not exclude other modelling standards. As described later in this work, UML is only used for the platform-independent modelling language definition. The model for e.g. the system architectural design is based on a completely domain-specific approach (see also Figure 2.7, respectively [60] for the gradation of domain-specific languages).

2.3 Automotive E/E System Design Methodologies

The research field of electrical and/or electronic system design is many years old but still an interesting and by far unexhausted topic. Specifically for embedded automotive systems this longevity is mostly based on the increasing proportion of electronics and software in the overall system *Vehicle*. To cope with the challenges emerging from the increasing number of functions realized through hardware and software components, and with the growing complexity of the integrated overall system, many methodologies and techniques to support the various tasks along the product life cycle have been introduced. In the present state of scientific knowledge, model-driven approaches have been proposed to support the majority of these tasks. The benefits of establishing methodologies based on modelling were discussed in 2.2. In this section, related work regarding system design methodologies, with a focus on the automotive sector, is presented.

Macher [34] carried out a research project in the field of development of automotive embedded systems and summarized the results in his PhD thesis titled *Framework for the Integrated Model-Based Development of Dependable Automotive Systems and Software*. The three major challenges his thesis focus on are (1) multi-core system development, (2) extra-functional system attributes development, and (3) life-cycle wide model-based development. As *extra-functional system attributes* he categorises system and product aspects such as *Functional Safety* and *IT Security*. Consistent with Chemuturi [13], in this thesis these kinds of attributes belong to the category of *Ancillary Functions*. In the author's opinion it is not that important if either the prefix *Extra* or *Ancillary* is selected. Both are less misleading than the widespread term *Non-Functional*, which is avoided throughout this work. Macher conducted his research work at the Graz University of Technology during almost the same period of time as the author of this thesis. Therefore, several intersections can be found when comparing both projects. In some parts, such as the incorporation of system and software architectural modelling, a development cooperation has been established between the authors. The work of Macher is directly related to this thesis and may be seen as the subsequent step, in terms of adding methods and techniques to a company's repertoire, after the herein described approach is implemented as a sound basis. Like others in the field of model-driven embedded system development ([6], [19], [46], [48]), the research work of Macher is based on methods utilizing a subset of SysML. In his related work [34], he presents an overview of the existing UML/SysML based approaches, such as *MARTE* and *EAST ADL*, and categorizes them as *General Purpose Meta-Model MBD*. Macher focuses on this category throughout his work, whereas this thesis concentrates on the second methodology category *Domain Specific Language MBD*. Nevertheless, the decision for one of the mentioned methodology categories does not have to be exclusive. That is, within the herein presented research work, integration endeavours have taken place to connect the two approaches aided by an adapter. The incorporation of both methodologies is outlined in Figure 2.4, whereby the approach proposed by Macher is labelled with *Existing Approach*, and the methodology presented in this thesis is labelled with *New DSM Approach*.

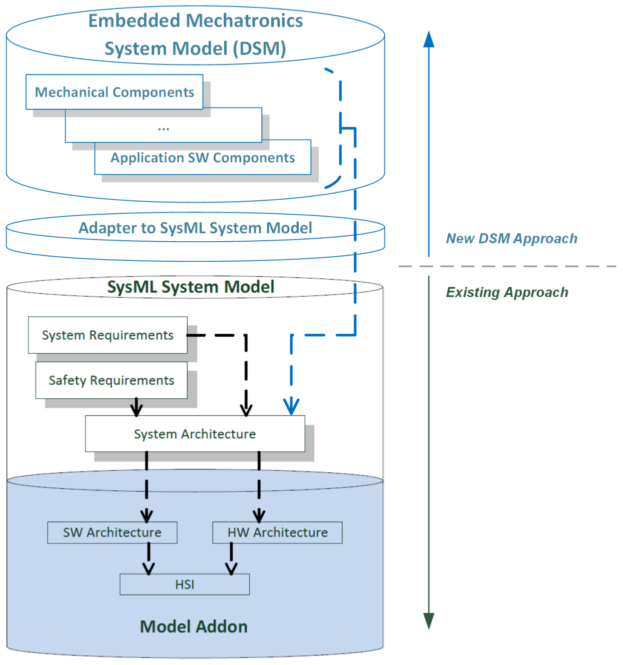


Figure 2.4: Incorporation of the Presented Approach and [34]

Wolf [61] presents a good overview of basic design methodologies, which in general can be utilized for most embedded system domains. As a rationale for design methodologies he states that without them products cannot be reliably delivered. Sometimes thinking about the sequence of steps which are necessary to build up a system may seem superfluous, but the fact is that everyone has their own idea about the design process. Since many embedded systems are too complex to be built by one person it is necessary that several people work together on a project. Therefore establishing a common understanding of how an embedded system shall be developed is crucial. He further states that a good methodology is critical to building properly working systems and that delivering buggy systems always causes dissatisfaction. Moreover, in some applications such as automotive systems, bugs can lead to safety problems that can harm the user or people around the system.

To agree with Wolf, high quality products and therefore an adequate development including good design methodologies are vital for every company. Denying this fact most likely leads either to disappointed customers and thus to a shrinking business or in the case of safety-relevant applications to financial loss due to product liability, quite apart from ethical matters.

As mentioned in the above paragraph Wolf also presents a good overview of design methodologies for embedded system development. The major steps in the embedded system design process as described by Wolf are reflected in Figure 2.5. The process starting at the *Requirements* level and ending at the *System integration* level is called *Top-down design*. The other way round, illustrated by dashed lines, he refers to as *Bottom-up design*. Along the embedded system design the major goals *Manufacturing Cost*, *Performance* (from overall speed as well as a deadline point of view), and *Power Consumption* have to be considered continuously. Moreover, at each step the tasks

a) *Analyze* the design to decide how the systems specification can be met

b) *Refine* the design to add detail, and

c) *Verify* the design for fulfilment of the functional and non-functional goals

has to be carried through.

The main reason for introducing the work of Wolf as related work in this thesis, is the simplicity and simultaneously the comprehensiveness of the presented general *Embedded System Design Process*. As it will be apparent through the subsequent sections and chapters, the established design process models at the various domains can be abstracted towards this model.

For the architectural system design Wolf utilizes both, simple block diagrams and *Unified Modeling Language (UML)* diagrams for a more formal description. Moreover, he distinguishes between a structural description (UML class diagram), a behavioural description (UML state diagram with optional sub states), and a temporal sequence description (UML sequence diagram). In the thesis author's view the UML diagrams, in particular the UML class diagram, are not the best choice for designing an embedded automotive system. The language's elements are too versatile and thus the probability that two system designers model the same system artefacts with different elements is

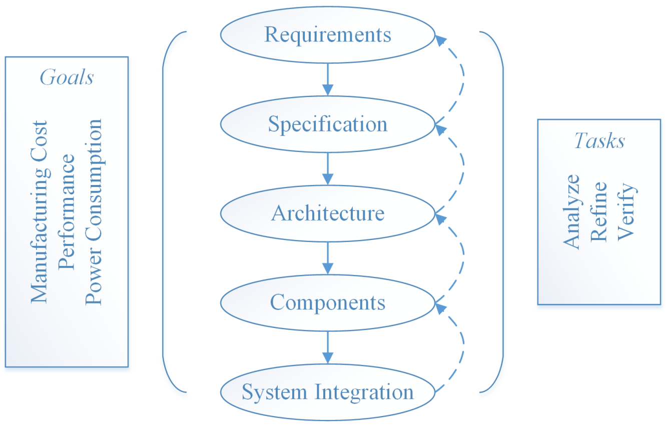


Figure 2.5: Major Steps in the Embedded System Design Process from [61]

high. Considering the demand for a more formal way to design the system, this situation is not optimal.

Reke [49] conducted a PhD thesis with the translated topic *Model-based Development of Automotive Control Systems in Small and Medium-sized Enterprises*. He mentions that small and medium enterprises (SMEs) are the industry's backbone in term of innovation. However, the cooperation between SMEs and large companies is hampered by the comprehensive and strongly regulated processes which are necessary considering the large number of employees and the distributed development scenario at large enterprises. In contrast, SMEs usually have implemented a simple process landscape and are able to act more flexibly in their daily business. Reke presents an approach that mitigates the challenges resulting from these differences. His strategy is to use the development process techniques utilized by large companies as a basis and to tailor the processes to the special needs of SMEs. That is, to eliminate all aspects from the processes that were introduced to facilitate distributed development and to keep those that contribute to the achievement of high quality software. Moreover he appraises model-based development as the best choice regarding the support of the described approach and presents a tool chain for the model-based software development. Additionally, Reke claims that the tailored processes fulfil the outcomes and practice demanded by *Automotive SPICE*.

To agree with Reke, implementing the complete set of a state-of-the-art process model like *Automotive SPICE* overextends most small enterprises, at least if they are new to the development process world. In the context of medium enterprises, the author of this thesis takes a different view. The existing process models are not exclusively made for large enterprises and groups. According to [20] the main criteria for an enterprise to be assigned to the category of SMEs are the number of employees, the annual turnover, the balance sheet total, and resource attributes such as ownership and similar. To reflect the detailed definition would go beyond the scope of this work, but solely from the number of people a medium sized enterprise has between 50 and 249 employees. In the opinion of the author a company with more than 50 employees has usually been in business for several years and should have established the necessary processes. Thus, the focus of this work is on the support of small- and micro-sized enterprises.

Aside from the concept phase, Reke places particular emphasis on the software development. An overview of his tailored process is illustrated in the form of the *Business Process Model and Notation (BPMN)*[2]) in Figure 2.6. He mentions that the design and implementation of hardware and system is not described in his thesis, but takes place temporally in parallel and out of the process' view analogue to the software design and implementation. In the author's opinion, at the very least the system design and integration work cannot take place in parallel with those related to hardware and software. However, the strategy presented by Reke is a good basis for small enterprises to start from. The present thesis shall give more insights into the system design aspects at embedded mechatronics-based system development.

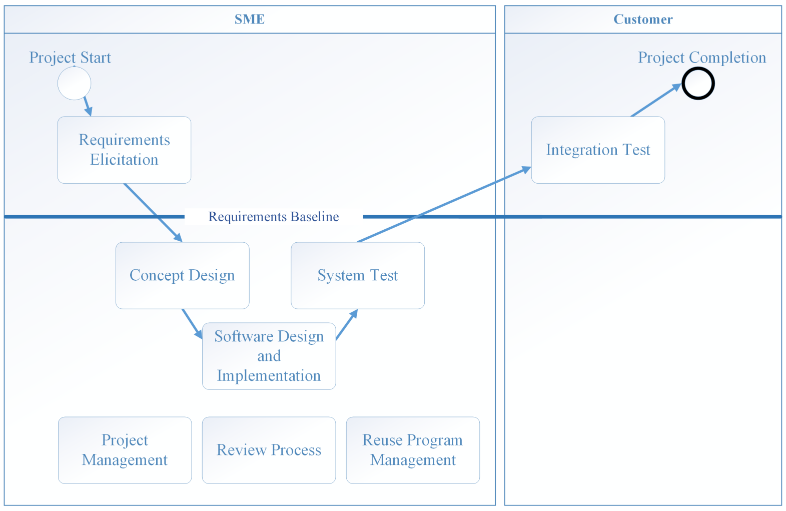


Figure 2.6: Software Development Process for SMEs from [49]

Meyer [40] conducted a PhD study in the field of consistent model-based development methodologies for the development of automotive control systems incorporating

[2]`http://www.bpmn.org/`

the AUTOSAR[3] standard. He states that AUTOSAR does not give any advice about the description of dynamic system aspects and also that there is no relation between requirements engineering and AUTOSAR. He claims to close this gap by combining SysML/UML with the AUTOSAR standard and thus establishing a consistent automotive control system development process, comprising of all artefacts from the architecture to the implementation.

Meyer does not consider the verification and validation (V&V) of any artefact along the product development life cycle. Furthermore he strongly focuses on the de facto standard AUTOSAR, which is a widespread software reference architecture. By February 2016 there are nine core partners (*Original Equipment Manufacturer* and *Tier-1 Suppliers*) and 198 other companies supporting the AUTOSAR project [8]. Nevertheless there are still many other software architectures around and will remain to be over the next few years. Martínez-Fernández et al. [36] conducted a survey on the benefits and drawbacks of AUTOSAR. Their target group were practitioners from the automotive industry with AUTOSAR experience. The survey showed *Standardization* and *Reuse* as the biggest advantages named by the experts. As experienced drawbacks they stated the *Complexity*, the high *Initial Investment*, and the adverse *Learning Curve*. AUTOSAR is out of scope of this thesis because the main focus lies on the system design, not on a detailed software architecture. Moreover, V&V aspects are considered in contrast to [40].

Hillenbrand [22] pursues the incorporation of ISO 26262 and the model-driven architectural design, in his words the *Electric/Electronic Architecture (EEA)*, of automotive electric/electronic systems in his PhD thesis. As described previously, ISO 26262 is the international standard for functional safety in the domain of road vehicles. Hillenbrand strongly focuses on the modelling of the hardware components including the corresponding safety aspects according to the ISO 26262. He defines the goals of his work as creating a common understanding, respectively a common language of ISO 26262 and EEA modelling, developing methods for the support of an optimized functional safety view within the EEAs, facilitating the vehicle development by deployment of context-related data from the EEA model for concurrent or subsequent development activities, and proposing an approach for the specification and execution of queries related to the EEA models.

The methods presented by Hillebrand, which deal with the incorporation of ISO 26262 related artefacts and the EEA, are premised on model-based development techniques. He describes the required features of potentially usable tools, such as the possibility of domain-specific tailoring and execution of various algorithms on the created model. Considering these constraints he comes to the conclusion that the tool *PREEvision* from

[3]`http://www.autosar.org/`

Aquintos, which is now fully owned by *Vector Informatik GmbH*[4], is the only tool in the domain of EEA modelling which fulfils the stated requirements. PREEvision is based on Eclipse and implements the *Electric/Electronic Architecture - Architecture Description Language (EEA-ADL)*. This language provides different abstraction layers, e.g. *Requirements*, *Logical Architecture*, *Electric Circuit*, etc. for the EEA modelling. Thus, EEA-ADL represents a meta model and defines about 1200 classes, 400 associations, and 400 attributes in the version 3.0.

The topics of this thesis partly overlap with those from Hillebrand, but in general the herein presented methodologies and techniques takes care of the design on a higher system level including also mechanical items. Moreover the target group is completely different. While Hillebrand's methods tends to support development teams as they are usually composed at medium and large companies as a complete vehicle development, i.e. experts for the various domains such as electronics, basis software, application software, mechanical engineering, system architecture, etc. are available, this work tries to support small-sized development entities by providing a lightweight and methodical approach for a continuous embedded system development.

2.4 Domain-Specific Modelling

A rigorous separation of the previous Chapter 2.3, where the related work concerning *Automotive E/E System Design Methodologies* was presented, and this chapter, which outlines the existing contributions in the field of *Domain-Specific Modelling*, is not possible. Many aspects of these two topics overlap, in particular when it comes to specifying the model-based artefacts along the development lifecycle. Thus, the following part may be also seen as a specialisation of the previous topic.

Within this research work, a *Domain-Specific Language (DSL)* that is combined with *Model-Driven Development (MDD)* techniques is called *Domain-Specific Modelling (DSM)*. When dealing with DSLs or their model-driven counterpart DSM, sometimes the question of whether a language may be called domain-specific or not arises. Andrew Watson [60], Technical Director of the *Object Management Group*[5], gives an answer to this question. He mentions that there is a spectrum of standardisation from bespoke domain-specific languages for particular applications, to standard language dialects, to the strict use of general purpose languages (GPLs). He presents the levels as being more or less domain-specific with focus on UML and UML profiles, but in this classification, which is portrayed in Figure 2.7, UML may be substituted by any general purpose

[4]http://vector.com/
[5]`http://www.omg.org/`

language.

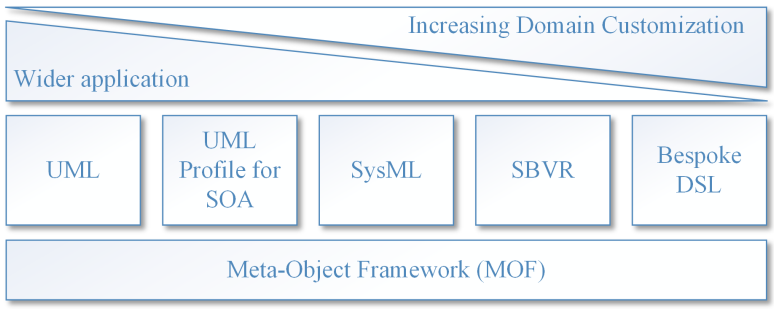


Figure 2.7: Modelling Languages from General-Purpose to Completely Domain-Specific from [60]

According to Mernik et al. [37], substantial gains in expressiveness and ease of use are obtained for languages that are tailored to the needs of a specific domain, compared to general-purpose languages. The application of DSLs facilitates analysis, verification, optimization, parallelization, and transformation of the various work products related to the DSL. Hudak [24] argues that programs written in a DSL are more concise, can be written more quickly, and are easier to maintain. Additionally, they can often be written, or in the case of a DSM modelled, by non-programmers. That is, DSLs and DSMs can enable domain experts to develop programs and designs which helps to bridge the gap between developers and users. Furthermore, Hudak defines basic steps for creating a domain-specific language as

1) definition of the domain,
2) design of the language that captures the domain semantics,
3) creation of tool support for the DSL, and
4) application of the new DSL infrastructure.

These steps are fulfilled by the presented approach and described in Chapters 4 & 5, and in the related publications which are referenced in these chapters.

Also Preschern et al. [47] present advantages of DSLs and state that the domain-specific languages help to decrease system development cost by providing an effective way of constructing systems for a particular domain. However, the benefit of developing in a more effective way compared to using general purpose languages, has to be higher than the investment of creating and establishing a DSL. Because of new tools which better support a domain-specific language creation, Preschern et al. claim that the amount of this investment will decrease significantly over the next few years.

Tools for creating model-based domain-specific languages, as mentioned by [47], have been available for several years now. Both, Kern et al. [30] and Kouhen et al. [32] evaluate the benefits and drawbacks of the state-of-the-art *Meta Meta Models.* Interestingly, the ranking of the tools resulting from their particular evaluation, with regard to which is the most powerful expressive tool [30] and to which one is the most capable of fast language development [32], is similar.

Kern et al. state that the definition of modelling languages is a cornerstone in *Domain-Specific Modelling* and that meta modelling is a wide-spread approach for formalizing DSM. In [30] they compare a set of meta modelling languages. The selection is based on criteria such as the kind of language definition (*Lightweight* or *Heavyweight*) and the concrete syntax (*Graphical* or *Textual*). In their work they focus on heavyweight types of DSM meta meta models that enable graphical language definition with textual annotation, and for which a tool support is available. *Heavyweight* in this context means that the particular language is created by the definition of a meta model from scratch. A well-known example of a lightweight approach is the stereotype mechanism of UML. The approach presented in Chapter 4 complies to the term *Heavyweight.*

Based on their requirements towards the meta modelling definition, Kern et al. choose the tools and meta meta models *ARIS* [50], *Ecore* [54], *GME* [17], *GOPPRR* [29], *MS DSL Tools* [15], and *MS Visio* [10] for their evaluation. Kouhen et al. additionally selected the tools and meta meta models IBM's *Rational Software Architect (RSA)* [55] and the *Obeo Designer* [27]. In their study they do not take *ARIS*, *MS DSL Tools*, and *MS Visio* into account. Generally, for their evaluation they chose only tools which provide *Meta-CASE (Computer-Aided Software Engineering)* characteristics. Tools belonging to this category can generate CASE tools in a very short time, compared to conventional development time scales, which support software development methods [5]. A description of the named tools and meta meta models including their particular relevance for the herein presented work, can be found at the end of this section.

The evaluation of the meta meta models and the corresponding tools proposed by Kouhen et al. is based on observing the process of customization for a sample DSM. For the presented case study they prepared a simplified version of the *Business Process Model and Notation* [44] meta model. Based on the work of Mohagheghi and Haugen[42], they proposed evaluation metrics and criteria with regard to *Customization Level*, *Graphical Expressiveness and Completeness*, *Tool Openness*, *Tool Usability*, *Required Resources*, *License Nature*, and *Produced Artefacts.* To describe the selected criteria in detail goes beyond the scope of this work and can be found in [32]. For the majority of criteria

Eclipse GMF and *Obeo Designer* are evaluated as the best solution, but regarding the required resources *MetaEdit+* is clearly ahead with only 0.5 man-days followed by *Obeo Designer* with 5 man-days.

As mentioned previously, Kern et al. propose an evaluation of the meta meta models and tools too. In terms of the most powerful expressiveness they rate *MetaEdit+* and *GME* highest and *MS Visio* lowest. The evaluation results presented by Kern et al. and Kouhen et al. are well prepared and informative, but every company or development team has to find the environment which suits them best on their own, taking their particular product they are to develop and available resources into account. Nevertheless, the comparison of the meta meta model's attributes, which are included at the presented evaluations, may help to come to a decision regarding which model or tool should be selected.

As mentioned previously, an overview of the available meta meta models and corresponding tools is given in the remaining part of this section.

Architecture of Integrated Information Systems (ARIS)

According to Scheer and Nüttgens [51] the *Architecture of Integrated Information Systems (ARIS)* can be used as a cornerstone for both, business process re-engineering and business process management. Together with the *ARIS-House of Business Engineering* architecture it provides a framework for managing business processes, incorporating the four levels *Process Engineering*, *Process Planning and Control*, *Workflow Control*, and *Application System*.

Kern and Kühne [31] mention that the DSM definition relies on the ARIS method, which can be configured according to the predefined instances of *ModelType*, *ObjectType*, *ConnectionType*, *AttributeType*, and *Symbol*. The existing types can be renamed and new ones can be derived from them.

ARIS strongly focusses on business processes and cannot be directly utilized to support the presented research work.

Ecore

The *Eclipse Modeling Framework (EMF)* project is a modelling framework and code generation facility for building tools and applications, based on a structured data model. The framework provides a basic editor and other tools, including a runtime support, to produce Java classes for the model. Aided by a set of adapter classes, viewing and command-based editing of the model is enabled. *Ecore* is one of the three fundamental pieces of EMF and represents a meta model for describing models and runtime support for the models. The other two main pieces of EMF are the *EMF.Edit* framework for building editors for the models through generic classes, and *EMF.Codegen* for completing the framework and the created editors with code generation capabilities [54].

While model-driven development approaches based on EMF are widespread in the academic world, it has been chosen not to use it for this research work. Though sound experience with software development and general purpose languages such as Java exists, the effort for familiarization with the EMF concept to achieve a proper support of this work's approach has been classified as too high. This justification has been strengthened through the feedback from mainly mechanical engineers, who were asked to build a sample architectural design utilizing SysML within an Eclipse editor. Most of them complained that the modelling environment was too complex and therefore too confusing. Thus, other possibilities have been selected for the support of the approach described in the subsequent chapters.

Generic Modeling Environment (GME)

Davis [17] describes the *Generic Modeling Environment (GME)* as an architecture for creating domain-specific design environments. GME supports a variety of general modelling principles which can be utilized to create a domain-specific language and a corresponding domain-specific modelling environment. Meta models in UML notation are used to capture the syntax, semantics, and presentation and thus to specify the modelling language for the particular target domain. In detail, the syntax is specified through UML class diagrams, the semantics are described aided by the *Object Constraint Language (OCL)*, and the presentation and visualization information is captured using stereotypes and predefined attributes of the UML classes and associations.

While the *Generic Modeling Environment* seems suitable for supporting the approach in Chapter 3, it has not been applied yet. A difficult constraint to the utilization of a meta meta model for the approach is the capability to provide a multi-level modelling environment for the target domain, which is automotive mechatronics system design in the case of this work. Multi-level means that a class defined in the meta model is capable of containing itself and or other classes. Sometimes this feature is also titled *Subgraph.* The modelling of hardware and software components described in Chapter 5 is an example of such a multi-level characteristic. It has not yet been clarified whether GME is capable of producing a domain specific design environment with the required features. This belongs to future work.

IBM Rational Software Architect (RSA)

Swithinbank et al. [55] describe the *Rational Software Architect* as an integrated design and development tool that leverages model-driven development with UML. The main features of RSA with respect to MDD are a UML 2.0 editor, the support for UML 2.0 profiles, a pattern library, and a transformation infrastructure. With the combination of these features they claim to being capable of providing the required level of customization at RSA to support the automation of IT development processes according to the MDD approach. To customize UML for use in particular domains and applications, UML profiles are used. They are defined by introducing a set of stereotypes, which extend the existing elements of UML. RSA is capable of generating editors for such profiles and can be seen as representative for many other tools that utilize profiles as an UML extension mechanism [32].

A utilization of one of these *other tools* is presented by Macher [34]. For the system and software architectural design he uses *Enterprise Architect*[6], based on the mentioned UML customization mechanism through profiles. He conducted his research work at the Graz University of Technology during the same period of time that this work was carried out. Some topics were prepared and published together and points of intersection are presented in Chapters 3, 4, and 5.

MetaEdit+ (GOPRR)

Starting in the early 1990s, the products from *MetaCase* are one of the oldest and most widely used in the field of producing domain-specific languages.

MetaEdit stands for *Metamodeling Editor* and is based on the conceptual data model *GOPRR*, which is an acronym for *Graph, Object, Property, Relationship, and Role. Objects* typically appear as shapes in diagrams and can have properties such as *ID* or *Name*. *Properties* are attributes of objects and typically appear as text labels on items in diagrams. *Relationships* are associations between objects and typically appears as lines in diagrams. Like the objects, they also may have properties. *Roles* define how objects participate in specific relationships. Typically, roles appear as the endpoints of relationships in diagrams (e.g. as an arrowhead describing a particular data flow). The *Graph* aggregates a certain set of objects and their relationships with specific roles. In diagrams, graphs typically appear as a window containing objects, their relationships, and roles including the particular properties. The concept of graphs facilitate the illustration of one conceptual graph as multiple representational graphs. This also enables a deconstruction of a parent graph into subgraphs [28].

[6]`http://www.sparxsystems.com/`

Many of today's E/E System design methodologies and techniques in the automotive industry have been implemented into the MetaEdit environment. *EAST-ADL*, as an representative of these methodologies, has been integrated with its different abstraction levels. The models are verified according to the correctness, consistency, and completeness rules and can be transformed to targets such as *AUTOSAR Software Platforms* [39].

Baetens [9] describes some potential drawbacks. The meta model in MetaEdit is spread among different tools and a complete display of the whole model is not available. Furthermore the licenses are pretty expensive and the question of whether the investment pays off in the particular application has to be considered properly.

Even if there is a lot of good marketing in the background, this meta meta model may be a good choice for many DSL approaches. Nevertheless, the drawbacks presented by Baetens should be taken into account. The approach presented in this thesis can be supported by this tool without doubt. The main reasons for following a different strategy in this work are on the one hand the tool's price and on the other hand simply the desire to show an alternative way of implementing a DSM.

Microsoft Modeling SDK for Visual Studio - Domain-Specific Languages (MSDK-DSL)

The software development kit (SDK) for domain-specific languages is provided by the *Microsoft Developer Network (MSDN)* [41]. Generally, the definition of the domain-specific language is separated into two parts. Syntax and semantics are specified by modelling *Domain Classes* and their *Relationships* in a UML notation. The relationships can either be of type *Embedding* or *Reference*. Typically, a top node domain class, representing a kind of container class for all other classes in the language definition, is specified. Each domain class which shall be available in the DSM is connected to the top node domain class by an embedding relationship. Domain classes which shall have relationships to each other in the DSM are connected by a reference relationship. Domain classes can also be derived from a parent class. *Domain Properties* can be added for each domain class and also for the relationships. The second part of the DSM definition relates to the visualization. The modelling of this part typically starts with a top node of type *Diagram*. This node is directly connected to the top domain class in the syntax and semantics definition. Further elements of the visualization definition are typically *Connectors* and *GeometryShape*. The allocation of domain classes and geometry shapes, and relationships and connectors takes place through associations. From the DSM definition MSDK generates the model implementation, a graphical editor, a tree based explorer, and facilities for generating code or other artefacts by text templating. Further information related to *Microsoft Modeling SDK - Domain Specific Languages* can be found in [15].

A big advantage of the MSDK-DSL meta meta model and tool is the easy definition of the DSM meta model elements, their relationship and properties they belong to. Further customization of the DSM meta model can be realized by extensions of the generated environment, written in a GPL such as C#, which is a potential drawback. A further disadvantage is the need to purchase the Visual Studio software for each user of the developed DSM due to the fact that VS is simultaneously the runtime environment.

Without the extensions, the MSDK-DSL is not suitable to support the approach described in the subsequent chapters. Nevertheless, this meta meta model has been utilized and the application is presented in Chapter 5.

Microsoft Visio

At first glance, *Microsoft Visio* has little to do with the meta meta models described so far. However, a closer look reveals that there is potential to produce a DSM meta model. This can be done by creating a new *Stencil* and adding a new *Master* for every element in the DSM meta model. The master contains the visualization and the properties of the element. Relationships between the elements can be implemented as master too. With the command *Create New Subprocess* multi-level models can be introduced. More effort is required to establish diagram validation rules. As described by Hopkins [23], a specific validation API can be utilized to create and manage rule sets, using GPLs such as VBA. Aided by the available data connections to external sources such as databases, other development artefacts can be linked to the DSM. Diagram data can be exported into spreadsheets or databases and prepared with an external generator for subsequent data processing. Due to the fact that Visio has been developed for creating diagrams, the editor is one of the major strengths. Further information related to *Microsoft Visio* can be found in [10].

As mentioned before, Microsoft Visio may not be the first thing which comes to ones mind if talking about DSM meta meta models and tools, but for certain applications it may be a good choice. For features such as connection to databases, the professional version is required. The costs for purchasing the tool are comparatively low and a viewer is available for free. A drawback of this tool is the missing or at least hard to implement feature of modelling constraints. This makes it less suitable for the approach described in the remaining part of this thesis.

Sirius/Obeo Designer

Vujović et al. [59] state that *Sirius* is a framework to create, visualize and edit models using interactive editors called *Modelers.* The framework is based on the *Graphical Modeling Framework (GMF)*, which is a component of the *Eclipse Graphical Modeling Project (GMP)* [56]. Vujović et al. mentions that producing a new DSM modelling workbench graphically is one of the big advantages of *Sirius.* Skills in developing software in Java and knowledge of Eclipse's *Graphical Editor Framework (GEF)* and *Graphical Modeling Framework (GMF)* is not required. The structure, appearance, and behaviour of the DSM is described by a *Viewpoint Specification Model (VSM)*, which stores the five main concepts of Sirius *Viewpoint*, *Representation*, *Mapping*, *Style*, and *Tool.* The creation of viewpoints using *Obeo Designer* is described by Juliot and Benois [27]. The designer hides the complexity of GMF and utilizes *MARTE* [43], which is a UML profile geared for usage in the field of real-time systems and embedded systems, for building an embedded modelling tool. They claim that *Obeo Designer* is powered by Sirius, which means that the viewpoint created by the designer is utilized by *Sirius* to create a DSM modelling environment.

Even if the *Obeo Designer* in combination with *Sirius* provides an abstraction of the GMP components there is still, from the author's point of view, expert knowledge in Eclipse environment development required. The development can become a time consuming task very quickly, especially for certain DSM meta model customizations. As mentioned in the paragraph above describing *Ecore*, for this research work the decision has been made not to use methodologies based on *Eclipse GMP.* For someone who is familiar with the Eclipse development environment, it is maybe a less time consuming task to integrate the approach described in this work into Sirius, which is theoretically feasible.

3 Embedded Mechatronic System Development in Small Entities

The heading of this chapter and the challenge which comes along with it, is simultaneously the key issue of this thesis and was also one of the ulterior motives to starting the research work in this field back in 2012. In every project conducted with only a small number of developers - where the author was involved - the same kinds of challenges always resurfaced. Some of these matters were usually of a non-technical nature, which makes this thesis' top level research question *"How to facilitate the interdisciplinary development of embedded mechatronic systems within small development entities"* into a blend of engineering, management, and social aspects.

3.1 Sociological Aspects

The most important requirement, which affects all described steps towards a sound basis for high quality development, is non-technical. DeMarco and Lister [18] claim that the most serious problems, which harm the success of projects, are not of a technical nature, but based on sociological issues. Over almost 20 years they analysed industrial development projects that failed. 15 percent of small and medium projects were cancelled, postponed or delivered results which have never been used. At large projects with 25 man-years of development effort or more, the failure rate is even higher and tot up to 25 percent. DeMarco and Lister conducted interviews with the parties concerned and came to the result that in the overwhelming majority of failed projects no technological issues could be found. As a reason for the failure of the particular project their interview partners most frequently named *Politics*. This may be rather unfortunately worded since the problems behind this term typically belong to issues such as *communication problems*, *personnel problems*, *difficulties with the direct supervisor or with the customer*, *lack of motivation* or *high turnover*.

The described major problems above are not limited to small entities, but have a relatively greater impact compared to large companies due to their limited resource's flexibility. In the author's opinion, the main difference is the worst case scenario which can arise from these problems. For medium-sized or large enterprises the result may be a project carried through without success and the loss of money and reputation, which is a serious challenge but still manageable with adequate resources available. For small- and micro-sized companies, e.g. start-ups, ignoring sociological issues may be a complete knock out criteria for their business.

In one of the last projects the author was involved with, the situation had been as described by DeMarco and Lister. A small development team composed of two mechanical and two electronics/software engineers were supposed to build a prototype of an automotive mechatronic system (project details and company name shall remain anonymous). The management board included the founder and CEO, the CFO, and two vice presidents who were supposed to lead the engineering team. For what reason four persons at the management level are necessary at a total head count of nine (including the assistant to the CEO), should not be discussed here. Most likely it was due to the fact that neither the CEO nor the CFO had any experience in the automotive industry. To compensate these deficits two people were hired as vice presidents for product development and operations. The major problem in this constellation was that the vice presidents had no clue about product development either, but had been relatively clever in terms of self marketing. To hide their missing skills, they prevented the flow of information between the engineers and the CEO. One of the side effects of their strategy was a big delay in every development step. The engineers who had experience in the automotive domain and developed various mechatronic systems before, talked to the CEO about the recognized problem. He was grateful for the honest feedback and promised to take action, which never happened. Unfortunately, this sociological problem ended in the bankruptcy of a company, which, out of the technological view, would have been capable of developing the desired products.

More than 35 years ago, Crosby [16] concluded that if the developers are allowed to define their own quality standard, in return the company receives an increase in productivity which can absorb the costs of additional efforts to obtain a higher quality product.

Typically, an engineer is connected to the developed product in multiple ways. One of them may be described as an emotional attachment which leads to the fact that usually it is not necessary to urge an engineer to increase the quality of the developed product. This emotional connection is one of the key factors for high quality and has to be supported by the utilized methodologies and tools.

3.2 Engineering Processes

Unfortunately - at least sometimes from the view of a small development entity which has to deal with the additional cost - the safety standards such as the ISO 26262 for the automotive domain, represents the state of the art. This means that in the case of a physical injury or damage to the health of a person caused by a malfunctioning behaviour of the developed product, there is no room for excuses for the development team if they did not follow the directives of these standards. Apart from the ethical issues it is also a matter of product liability if a human suffers harm. Not least since the discovery of the defective ignition switches installed in millions of General Motors vehicles, it has become common currency that flaws arising during development can tragically lead to many lives being lost, as well as to a financial loss of millions or even billions of euros. Hence, independent of the respective manpower, every development team shall foster high quality in all aspects of the product's life cycle. Even if the item being developed does not perform a safety-related function, high quality introduced by establishing well-defined and practised engineering processes contributes to the product's reliability and therefore to customer satisfaction, which is vital for an economical survival.

An additional argument for having well-defined engineering processes implemented is to avoid increased costs of debugging due to a late error recognition in the product's life cycle. Although these statistics are several years old, they are still valid. Most of these well-known analysis results presenting 100 times higher relative costs to fix software defects when the product has been already released for production compared to debugging at the design phase. Altogether, it can be stated that well-defined development processes are necessary for every organisation involved in the product's development cycle, irrelevant of whether it is a micro-sized or a large company. However, an implementation of e.g. a requirements management process for a large entity will most likely not be suitable for a small one. In the following subsections within this section a possible engineering process model for small development entities is presented, which should ease the entry into the engineering processes for them. Nevertheless, it has to be clearly stated that in this work, as well as in all other serious contributions in the field of embedded mechatronic system development, a *From Zero to Hero* strategy is not, and cannot be provided. That is, teams and companies that are relatively new to the field of performing engineering processes and functional safety, cannot be enabled to being capable of developing a highly safety relevant mechatronic system, such as a steer-by-wire system, within a short period of time. With this in mind, this work should be understood as assistance to get into the relevant topics quickly and to avoid losing precious time looking for the answer to the question *Where should I start?* Though functional safety is not a key aspect of this thesis, the methodologies and techniques of ISO 26262 have been taken into account when preparing the herein described approach. Hence, the proposed models and patterns have been developed with the ulterior motive to be a basis for a safety relevant development capability.

3.2.1 Process Reference Model for the Embedded Mechatronic System Development (EMS-PRM)

As mentioned in Chapter 2, *Automotive SPICE* is used as the underlying process reference model throughout this research work. At the EU project *AQUA* [33], an integrated view of *Automotive SPICE*, *Functional Safety*, and *Six Sigma* is proposed. Inspired by that approach, the process reference model described in this chapter, the system architectural design methodology (see Chapter 4), and the tool support (see Chapter 5) of the presented approach, are aligned to Automotive SPICE and set up for functional safety relevant development artefacts whenever applicable.

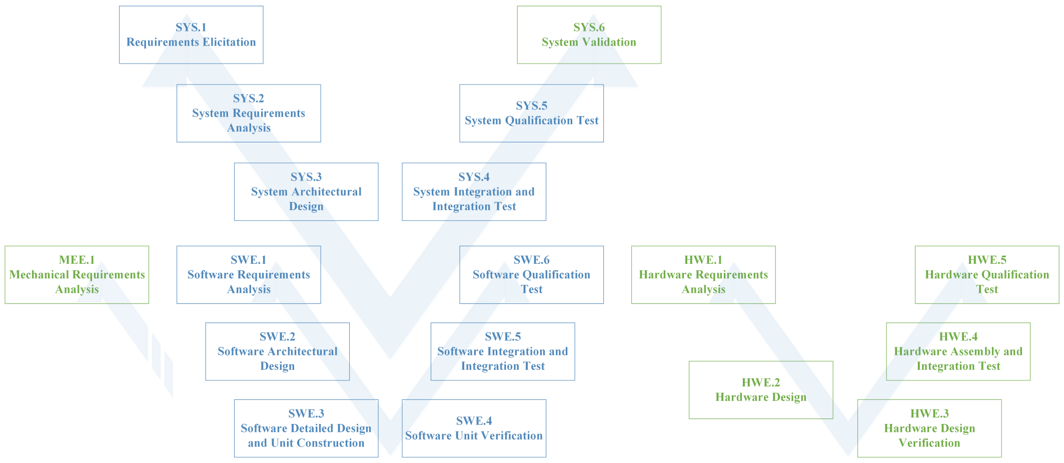

SYS.1
Requirements Elicitation
SYS.6
System Validation
SYS.2
System Requirements Analysis
SYS.5
System Qualification Test
SYS.3
System Architectural Design
SYS.4
System Integration and Integration Test
MEE.1
Mechanical Requirements Analysis
SWE.1
Software Requirements Analysis
SWE.6
Software Qualification Test
HWE.1
Hardware Requirements Analysis
HWE.5
Hardware Qualification Test
SWE.2
Software Architectural Design
SWE.5
Software Integration and Integration Test
HWE.4
Hardware Assembly and Integration Test
HWE.2
Hardware Design
SWE.3
Software Detailed Design and Unit Construction
SWE.4
Software Unit Verification
HWE.3
Hardware Design Verification

Figure 3.1: EMS-PRM aligned to Automotive SPICE [58]

In Figure 3.1 the process reference model for the embedded mechatronic system development is shown. The blue coloured elements are references to the Automotive SPICE processes and can be used one to one. The green coloured elements represent new processes proposed by the author, which are aligned to the Automotive SPICE engineering processes. The methodology is briefly described as *The Plug-in Concept* in [58]. For a seamless integration into the existing process reference model, it has been attempted to describe the new processes in an identical way. The proposed process definitions to extend the PRM:

- HWE.1 - Hardware Requirements Analysis (see Table 3.1)
- HWE.2 - Hardware Design (see Table 3.2)
- HWE.3 - Hardware Design Verification (see Table 3.3)
- HWE.4 - Hardware Assembly and Integration Test (see Table 3.4)
- HWE.5 - Hardware Qualification Test (see Table 3.5)
- MEE.1 - Mechanical Requirements Analysis (see Table 3.6)
- SYS.6 - System Validation (see Table 3.7)

The tables in the remaining part of this subsection present the detailed definition of the new process reference model elements. The number given in front of the particular items in the section *Output work products* of the process definition, represent the *Work product identifier* and references to the more detailed information in *Annex B Work product characteristic* in [58].

Table 3.1: EMS-PRM HWE.1 - Hardware Requirements Analysis - Process Definition

Process ID	**HWE.1**
Process name	**Hardware Requirements Analysis**
Process purpose	The purpose of the Hardware Requirements Analysis Process is to transform the hardware related parts of the system requirements into a set of hardware requirements.

Process outcomes	As a result of successful implementation of this process: 1) the hardware requirements to be allocated to the hardware elements of the system and their interfaces are defined; 2) hardware requirements are categorized and analysed for correctness and verifiability; 3) consistency between the impact of the software requirements on the hardware related parts of the operating environment and the hardware requirements is examined; 4) prioritization for implementing the hardware requirements is defined; 5) the hardware requirements are updated as needed; 6) consistency and bidirectional traceability are established between system requirements and hardware requirements; and consistency and bidirectional traceability are established between system architectural design and hardware requirements; 7) the hardware requirements are evaluated for cost, schedule and technical impact; and 8) the hardware requirements are agreed and communicated to all affected parties.
Base practices	**HWE.1.BP1: Specify hardware requirements.** Use the system requirements and the system architecture, and changes to system requirements and architecture to identify the required functions and capabilities of the hardware. Specify functional and non-functional hardware requirements in a hardware requirements specification. [Process Outcome 1, 5, 7] **HWE.1.BP2: Structure hardware requirements.** Structure the hardware requirements in the hardware requirements specification by e.g. grouping to project relevant clusters, sorting in a logical order for the project, categorizing based on relevant criteria for the project, prioritizing according to stakeholder needs. [Process Outcome 2, 4]

	HWE.1.BP3: Analyse hardware requirements. Analyse the specified hardware requirements including their interdependencies to ensure correctness, technical feasibility and verifiability, and to support risk identification. Analyse the impact on cost, schedule and the technical impact. [Process Outcome 2, 7] **HWE.1.BP4: Analyse the impact of the software development artefacts on the hardware requirements.** Analyse the impact that the software requirements and the software architectural design will have on the hardware requirements to enable the availability of necessary resources. [Process Outcome 3, 7] **HWE.1.BP5: Develop verification criteria.** Develop the verification criteria for each hardware requirement that define the qualitative and quantitative measures for the verification of a requirement. [Process Outcome 2, 7] **HWE.1.BP6: Establish bidirectional traceability.** Establish bidirectional traceability between system requirements and hardware requirements. Establish bidirectional traceability between the system architecture and hardware requirements. [Process Outcome 6] **HWE.1.BP7: Ensure consistency.** Ensure consistency between system requirements and hardware requirements. Ensure consistency between the system architecture and hardware requirements. [Process Outcome 6] **HWE.1.BP8: Communicate agreed hardware requirements.** Communicate the agreed hardware requirements and updates to hardware requirements to all relevant parties. [Process Outcome 8]
Output work products	13-04 Communication record [Process Outcome 8] 13-19 Review record [Process Outcome 6] 13-21 Change control record [Process Outcome 5, 7] 13-22 Traceability record [Process Outcome 1, 6] 15-01 Analysis report [Process Outcome 2, 3, 4, 7] 17-08 Interface requirements specification [Process Outcome 1] 17-12 System requirements specification [Process Outcome 1, 3] 17-50 Verification criteria [Process Outcome 2]

The *Hardware Design* process definition in the following Table 3.2 is comparable with the process *SWE.3 Software Detailed Design and Unit Construction.* An architectural design as it shall be created during the software development cycle, is not necessary for the hardware because an architectural view of the hardware components is already given by the system architectural design.

Table 3.2: EMS-PRM HWE.2 - Hardware Design - Process Definition

Process ID	**HWE.2**
Process name	**Hardware Design**
Process purpose	The purpose of the Hardware Design Process is to establish a detailed design for the hardware, and to identify which hardware requirements are to be allocated to which hardware components.
Process outcomes	As a result of successful implementation of this process: 1) a detailed design is developed that describes the composition of all required hardware components; 2) the hardware requirements are allocated to the hardware components; 3) a detailed description of each hardware component and its interface is available; 4) the dynamic behaviour and resource consumption objectives of the software elements related to the particular hardware component are analysed; and 5) consistency and bidirectional traceability are established between hardware requirements and hardware design.
Base practices	**HWE.2.BP1: Develop hardware design.** Develop a hardware design, typically established through schematics, PCB design, integrated circuit design, and similar artefacts, that represents a composition of all required hardware components, and that enables the fulfilment of the established functional and non-functional hardware requirements. [Process Outcome 1] **HWE.2.BP2: Allocate hardware requirements.** Allocate the hardware requirements to the components of the hardware design. [Process Outcome 2] **HWE.2.BP3: Define interfaces of hardware units.** Identify, specify and document the interfaces of each hardware component. [Process Outcome 3]

	HWE.2.BP4: Describe dynamic behaviour. Evaluate and document the timing and dynamic interaction of hardware components to meet the required dynamic behaviour of the system. [Process Outcome 3, 4] **HWE.2.BP5: Analyse resource consumption objectives.** Determine and document that the resource consumption objectives for the software elements related to the particular hardware component can be met. [Process Outcome 4] **HWE.2.BP6: Establish bidirectional traceability.** Establish bidirectional traceability between hardware requirements and components of the hardware design. [Process Outcome 5] **HWE.2.BP7: Ensure consistency.** Ensure consistency between hardware requirements and the hardware design. [Process Outcome 1, 2, 5]
Output work products	04-00 Design - Hardware Design [Process Outcome 1, 2, 3, 4, 5] 13-19 Review record [Process Outcome 4, 5] 13-22 Traceability record [Process Outcome 5] 17-08 Interface requirement specification [Process Outcome 3]

Table 3.3: EMS-PRM HWE.3 - Hardware Design Verification - Process Definition

Process ID	**HWE.3**
Process name	**Hardware Design Verification**
Process purpose	The purpose of the Hardware Design Verification Process is to verify the hardware design to provide evidence for compliance of the design with the hardware requirements, and to review the hardware design for early design failure detection.

Process outcomes	As a result of successful implementation of this process: 1) a hardware design verification strategy is developed to verify the hardware design; 2) criteria for hardware design verification are developed according to the hardware design verification strategy that are suitable to provide evidence for compliance of the hardware design with the hardware requirements; 3) the hardware design is verified according to the hardware design verification strategy and the defined criteria for hardware design verification and the results are recorded; 4) consistency and bidirectional traceability are established between hardware design, criteria for verification and verification results; 5) the hardware design is agreed and communicated to all affected parties; and 6) intermediate hardware development artefacts according to the verified hardware design are produced.
Base practices	**HWE.3.BP1: Develop hardware design verification strategy.** Develop a strategy for verification of the hardware design including a regression strategy for re-verification if a component in the hardware design is changed. The verification strategy shall define how to provide evidence for compliance of the hardware design with the hardware requirements. [Process Outcome 1] **HWE.3.BP2: Develop criteria for hardware design verification.** Develop criteria for the hardware design verification that are suitable to provide evidence for compliance of the hardware design with the hardware requirements according to the verification strategy. Evaluate the hardware design in terms of interoperability, interaction, criticality, technical complexity, risks and testability. [Process Outcome 2]

	HWE.3.BP3: Perform static verification of the hardware design. Verify the hardware design for correctness using the defined criteria for verification. Record the results of the static verification. [Process Outcome 3] **HWE.3.BP4: Establish bidirectional traceability.** Establish bidirectional traceability between the hardware design and static verification results. [Process Outcome 4] **HWE.3.BP5: Ensure consistency.** Ensure consistency between the hardware design and the static verification results. [Process Outcome 4] **HWE.3.BP6: Communicate agreed hardware design.** Communicate the agreed hardware design and updates to the hardware design to all relevant parties. [Process Outcome 5] **HWE.3.BP7: Produce intermediate hardware development artefacts.** Produce intermediate hardware development artefacts, such as printed circuit-boards, according to the hardware design. [Process Outcome 6]
Output work products	08-50 Test specification [Process Outcome 2] 08-52 Test plan [Process Outcome 1] 11-00 Product [Process Outcome 6] 13-04 Communication record [Process Outcome 5] 13-19 Review record [Process Outcome 3, 4] 13-22 Traceability record [Process Outcome 4] 13-25 Verification results [Process Outcome 3] 13-50 Test result [Process Outcome 3] 15-01 Analysis report [Process Outcome 3]

Table 3.4: EMS-PRM HWE.4 - Hardware Assembly and Integration Test - Process Definition

Process ID	**HWE.4**
Process name	**Hardware Assembly and Integration Test**

Process purpose	The purpose of the Hardware Assembly and Integration Test Process is to assemble the individual hardware components and the intermediate hardware development artefacts (e.g. printed circuit boards) into the complete integrated hardware (hardware device), consistent with the hardware design and to ensure that the hardware device is tested to provide evidence for compliance with the hardware design, including the interfaces between the hardware components within the hardware device.
Process outcomes	As a result of successful implementation of this process: 1) hardware components and hardware intermediate development artefacts are assembled into a complete hardware device according to the hardware design; 2) a hardware device test strategy including a regression test strategy is developed to test the hardware device and the interaction between the hardware components; 3) a specification for the hardware device test according to the hardware device test strategy is developed that is suitable to provide evidence for compliance of the hardware device with the hardware design, including the interfaces between the hardware components; 4) Test cases included in the hardware device test specification are selected according to the hardware device test strategy, and the release plan; 5) the hardware device is tested using the selected test cases and the results of the hardware device test are recorded; 6) consistency and bidirectional traceability are established between the elements of the hardware design and the test cases included in the hardware device test specification and between test cases and test results; and 7) results of the hardware device test are summarized and communicated to all affected parties.

Base practices	**HWE.4.BP1: Assemble hardware components and intermediate hardware development artefacts.** Assemble the hardware components and the intermediate hardware development artefacts according to the hardware design. [Process Outcome 1] **HWE.4.BP2: Develop hardware device test strategy including regression test strategy.** Develop a strategy for testing the hardware device. This includes a regression test strategy for retesting the hardware device if components are changed. [Process Outcome 2] **HWE.4.BP3: Develop specification for hardware device test.** Develop the test specification for a hardware device test including the test cases according to the hardware device test strategy. The test specification shall be suitable to provide evidence for compliance of the hardware device with the hardware design. [Process Outcome 3] **HWE.4.BP4: Select test cases.** Select test cases from the hardware device test specification. The selection of test cases shall have sufficient coverage according to the hardware device test strategy and the release plan. [Process Outcome 4] **HWE.4.BP5: Perform hardware device test.** Perform the hardware device test using the selected test cases. Record the test results and logs. [Process Outcome 5] **HWE.4.BP6: Establish bidirectional traceability.** Establish bidirectional traceability between elements of the hardware design and test cases included in the hardware device test specification. Establish bidirectional traceability between test cases included in the hardware device test specification and hardware device test results. [Process Outcome 6] **HWE.4.BP7: Ensure consistency.** Ensure consistency between elements of the hardware design and the test cases included in the hardware device test specification. [Process Outcome 6] **HWE.4.BP8: Summarize and communicate results.** Summarize the hardware device test results and communicate them to all affected parties. [Process Outcome 7]
Output work products	08-50 Test specification [Process Outcome 3, 4] 08-52 Test plan [Process Outcome 2]

	11-00 Product [Process Outcome 1] 13-04 Communication record [Process Outcome 7] 13-19 Review record [Process Outcome 6] 13-22 Traceability record [Process Outcome 6] 13-50 Test result [Process Outcome 5, 7]

Table 3.5: EMS-PRM HWE.5 - Hardware Qualification Test - Process Definition

Process ID	**HWE.5**
Process name	**Hardware Qualification Test**
Process purpose	The purpose of the Hardware Qualification Test Process is to ensure that the hardware device, which has been assembled from the individual hardware components and intermediate hardware development artefacts such as printed circuit boards, is tested to provide evidence for compliance with the hardware requirements.

Process outcomes	As a result of successful implementation of this process: 1) a hardware qualification test strategy including regression test strategy consistent with the project plan and release plan is developed to test the hardware device; 2) a specification for a hardware qualification test of the hardware device according to the hardware qualification test strategy is developed that is suitable to provide evidence for compliance with the hardware requirements; 3) test cases included in the hardware qualification test specification are selected according to the hardware qualification test strategy and the release plan; 4) the hardware device is tested using the selected test cases and the results of the hardware qualification test are recorded; 5) consistency and bidirectional traceability are established between hardware requirements and hardware qualification test specification including test cases and between test cases and test results; and 6) results of the hardware qualification test are summarized and communicated to all affected parties.
Base practices	**HWE.5.BP1: Develop hardware qualification test strategy including regression test strategy.** Develop a strategy for hardware qualification testing consistent with the project plan and the release plan. This includes a regression test strategy for re-testing the hardware device if a hardware component is changed. [Process Outcome 1] **HWE.5.BP2: Develop specification for hardware qualification test.** Develop the specification for hardware qualification test including test cases based on the verification criteria, according to the hardware qualification test strategy. The test specification shall be suitable to provide evidence for compliance of the hardware device with the hardware requirements. [Process Outcome 2]

	HWE.5.BP3: Select test cases. Select test cases from the hardware qualification test specification. The selection of test cases shall have sufficient coverage according to the hardware qualification test strategy and the release plan. [Process Outcome 3] **HWE.5.BP4: Test hardware device.** Test the hardware device using the selected test cases. Record the hardware qualification test results and logs. [Process Outcome 4] **HWE.5.BP5: Establish bidirectional traceability.** Establish bidirectional traceability between hardware requirements and test cases included in the hardware qualification test specification. Establish bidirectional traceability between test cases included in the hardware qualification test specification and hardware qualification test results. [Process Outcome 5] **HWE.5.BP6: Ensure consistency.** Ensure consistency between hardware requirements and test cases included in the hardware qualification test specification. [Process Outcome 5] **HWE.5.BP7: Summarize and communicate results.** Summarize the hardware qualification test results and communicate them to all affected parties. [Process Outcome 6]
Output work products	08-50 Test specification [Process Outcome 2, 3] 08-52 Test plan [Process Outcome 1] 13-04 Communication record [Process Outcome 6] 13-19 Review record [Process Outcome 5] 13-22 Traceability record [Process Outcome 5] 13-50 Test result [Process Outcome 4, 6] 19-00 Strategy [Process Outcome 1]

Table 3.6: EMS-PRM MEE.1 - Mechanical Requirements Analysis - Process Definition

Process ID	**MEE.1**
Process name	**Mechanical Requirements Analysis**
Process purpose	The purpose of the Mechanical Requirements Analysis Process is to transform the mechanical related parts of the system requirements into a set of mechanical requirements.

Process outcomes	As a result of successful implementation of this process: 1) the mechanical requirements to be allocated to the mechanical elements of the system are defined; 2) mechanical requirements are categorized and analysed for correctness; 3) prioritization for implementing the mechanical requirements is defined; 4) the mechanical requirements are updated as needed; 5) consistency and bidirectional traceability are established between system requirements and mechanical requirements; and consistency and bidirectional traceability are established between system architectural design and mechanical requirements; 6) the mechanical requirements are evaluated for cost, schedule and technical impact; and 7) the mechanical requirements are agreed and communicated to all affected parties.
Base practices	**MEE.1.BP1: Specify mechanical requirements.** Use the system requirements and the system architecture, and changes to system requirements and architecture to identify the required functions and capabilities of the mechanical components. Specify functional and non-functional mechanical requirements in a mechanical requirements specification. [Process Outcome 1, 4, 6] **MEE.1.BP2: Structure mechanical requirements.** Structure the mechanical requirements in the mechanical requirements specification by e.g. grouping to project relevant clusters, sorting in a logical order for the project, categorizing based on relevant criteria for the project, prioritizing according to stakeholder needs. [Process Outcome 2, 3]

	MEE.1.BP3: Analyse mechanical requirements. Analyse the specified mechanical requirements including their interdependencies to ensure correctness, technical feasibility and verifiability, and to support risk identification. Analyse the impact on cost, schedule and the technical impact. [Process Outcome 2, 6] **MEE.1.BP4: Establish bidirectional traceability.** Establish bidirectional traceability between system requirements and mechanical requirements. Establish bidirectional traceability between the system architecture and mechanical requirements. [Process Outcome 5] **MEE.1.BP5: Ensure consistency.** Ensure consistency between system requirements and mechanical requirements. Ensure consistency between the system architecture and mechanical requirements. [Process Outcome 5] **MEE.1.BP6: Communicate agreed mechanical requirements.** Communicate the agreed mechanical requirements and updates to mechanical requirements to all relevant parties. [Process Outcome 7]
Output work products	13-04 Communication record [Process Outcome 7] 13-19 Review record [Process Outcome 5] 13-21 Change control record [Process Outcome 4, 6] 13-22 Traceability record [Process Outcome 1, 5] 15-01 Analysis report [Process Outcome 2, 3, 6] 17-12 System requirements specification [Process Outcome 1]

From the author's experience of previous mechatronic projects, it is not expedient to define additional *Mechanical Engineering (MEE)* processes in the way they are described for the system, software, and hardware processes. Typically the design approach for mechanical components differs significantly from the E/E system design approach. The mechanical engineering design methodology is innately *model-driven* by utilizing a *Computer-Aided Design (CAD)* software. Usually, a workshop drawing is automatically generated from the CAD design, or the necessary data for the manufacturing process is directly generated aided by a CAD/CAM interface (*Computer-Aided Manufacturing (CAM)*. Safety relevant requirements are given by e.g. the *Directive on Machinery* [57], and domain specific standards. In the field of mechanical engineering, the verification of mechanical components is also carried out in a quite different way compared to E/E system verification. Hence, the development of verification criteria etc. is out of scope of the EMS-PRM and subject to the established verification processes of mechanical engineering. The interface between the E/E system and the mechanical world are the defined mechanical requirements, which are derived from the system requirements. Consistency and bilateral traceability between these artefacts have to be assured throughout the whole product life cycle.

Table 3.7: EMS-PRM SYS.6 - System Validation - Process Definition

Process ID	**SYS.6**
Process name	**System Validation**
Process purpose	The purpose of the System Validation is to ensure that the system, which has been successfully verified according to the system integration and qualification tests, is tested to provide evidence for compliance with the customer requirements, and that the system is ready for delivery.
Process outcomes	As a result of successful implementation of this process: 1) a system validation strategy including regression test strategy consistent with the project plan and release plan is developed; 2) a specification for system validation according to the system validation strategy is developed that is suitable to provide evidence for compliance with the customer requirements; 3) test cases included in the system validation specification are selected according to the system validation strategy and the release plan; 4) the system is tested using the selected test cases and the results of the system validation are recorded; 5) consistency and bidirectional traceability are established between customer requirements and test cases included in the system validation specification and between test cases and test results; and 6) results of the system validation are summarized and communicated to all affected parties.

Base practices	**SYS.6.BP1: Develop system validation strategy including regression test strategy.** Develop a strategy for system validation consistent with the project plan and the release plan. This includes a regression test strategy for re-testing the system if a system item is changed. [Process Outcome 1] **SYS.6.BP2: Develop specification for system validation.** Develop the specification for system validation including test cases according to the system validation strategy. The test specification shall be suitable to provide evidence for compliance of the system with the customer requirements. [Process Outcome 2] **SYS.6.BP3: Select test cases.** Select test cases from the system validation specification. The selection of test cases shall have sufficient coverage according to the system validation strategy and the release plan. [Process Outcome 3] **SYS.6.BP4: Test system.** Test the system using the selected test cases. Record the system validation results and logs. [Process Outcome 4] **SYS.6.BP5: Establish bidirectional traceability.** Establish bidirectional traceability between customer requirements and test cases included in the system validation specification. Establish bidirectional traceability between test cases included in the system validation specification and system validation results. [Process Outcome 5] **SYS.6.BP6: Ensure consistency.** Ensure consistency between customer requirements and test cases included in the system validation specification. [Process Outcome 5] **SYS.6.BP7: Summarize and communicate results.** Summarize the system validation results and communicate them to all affected parties. [Process Outcome 6]
Output work products	08-50 Test specification [Process Outcome 2, 3] 08-52 Test plan [Process Outcome 1] 13-04 Communication record [Process Outcome 6] 13-19 Review record [Process Outcome 5] 13-22 Traceability record [Process Outcome 5] 13-50 Test result [Process Outcome 4, 6]

3.2.2 Pattern for Establishing Engineering Processes in Small Entities

When establishing an implementation of the proposed engineering process model, the sociological aspect described in 3.1 is best integrated through an open mindset towards the cross-domain issues. Hardware and software engineers do have a similar language when talking about their development and also the processes are typically established in an analogous manner. The working practices of mechanical engineers and hardware/software engineers are usually quite different. Often the same wording is used in a completely different context (e.g. *Design*). Thus, the most important point for establishing engineering processes successfully, is to integrate the whole team in this operation. The communication channels in small entities are rather short and therefore the participation of the whole team is typically manageable. In medium-sized and large companies, which are out of scope of this thesis, this is certainly not the case, but concepts such as *Key Users* can be utilized to achieve the same results.

The major steps for small and micro-sized teams for establishing a basis for high quality product development can be stated as follows:

(a) Select an engineering process model such as *EMS-PRM*

(b) Customize the process model to the specific needs of the domain, company, product, etc.

(c) Communicate the customized process model to the whole team

(d) Get feedback related to (c) from the whole team

(e) Integrate feedback from (d) and define the engineering processes

(f) Get proposals for the tools to be used from the parties concerned

(g) Create a toolchain proposal aligned to (e) and (f)

(h) Communicate the toolchain concept to the whole team

(i) Get feedback related to (h) from the whole team

(j) Integrate feedback from (i) and specify the toolchain (see also related process *SUP.8 Configuration Management* in [58])

(k) Determine possible improvements regarding (e) and (j) after each project completion

In the author's opinion it is important to try not to define a bunch of processes and introduce them all at once, in a worst case scenario without coordinating the subjects with the parties concerned. This most likely leads directly to a *dead process landscape* which is not adhered to.

Successfully implementing an engineering process can sometimes be a laborious task, but (1) as mentioned previously it is necessary to create a basis for an appropriate product quality, and (2) the persons in charge of this task may keep in mind that doing something is always better than doing nothing in this context. That is, if there are, at the present state at the small entity, e.g. no resources left for creating a seamless and fully automatic tool support, simple strategies may be chosen to create as many work products of the defined processes as possible. Independent of the level of the established processes, in 99% of cases there is always room for improvement. Hence, in the author's opinion it is better to establish a low level of engineering processes than to do nothing.

4 Domain-Specific Modelling of Embedded Mechatronic Systems

In Chapter 2 related work in the field of meta models and meta meta models including the particular supporting tools, which are available on the market, were presented. In Chapter 3 various aspects related to the embedded mechatronic system (EMS) development in small entities were highlighted, a hardware development extension to an established process reference model was introduced, and a pattern for the EMS development in small development entities was proposed. In this chapter, the definition of a meta model for domain-specific modelling (DSM) in the field of automotive embedded mechatronic systems is introduced. The meta model and its different aspects were presented at several conferences and published in the particular proceeding. Hence, this chapter is prepared as a road map to the detailed information in the publications, which can be found in Chapter 7.

The meta model of the EMS-DSM has been defined in UML notation and is depicted in Figure 4.1. As mentioned previously, the main focus of the defined language is to support engineers at creating designs of the different architectural design levels related to a embedded automotive system development life cycle.The goal has been to enable all parties concerned in the system development to contribute to the system architecture design process. That is, independent of the engineer's particular software development skills, such as more or less experience in creating structural UML diagrams, developing a component model consisting of the most important mechanical, hardware, and software parts including their basic properties, shall be made possible. A detailed description of the defined model-based language and its features can be found in **Paper A**.

Another key aspect of the presented system modelling approach in this thesis is that established design methodologies, such as the proposed system and software level modelling methods of Macher [34], should not be replaced. Instead, a bidirectional transformation between the design models in EMS-DSM and SysML notation shall enhance the possibilities of the approaches. A proposal related to this bidirectional transformation can be found in **Paper F**. Hence, other important system properties, such as the dynamic behaviour, can be modelled by an expert within the classical development environments outlined in Chapter 2.

As the generation of further development artefacts is one of the major aspects of a domain-specific modelling, not only a methodology of the transformation between the two architectural design notations is proposed. Moreover, in **Paper G** and **H** an approach for fostering the transformation between a software architectural design and a framework for the software detailed design in Simulink[1]/TargetLink[2] is presented. The methodology has been developed in cooperation with Macher [35], who integrated the transformation strategy into the software architectural design notated in SysML. In the course of this research work, the transformation between the software architectural design and the system detailed design, has also been enabled in the EMS-DSM development environment.

In **Paper B** the integration of requirements management capabilities is described. Out of the view of the EMS-DSM, an interface to development artefacts such as requirements, verification criteria, and test case specifications, is provided by the adopted project management tool (see Chapter 5), which establishes a direct access to the artefact's database.

Special attention has been paid to the support of the *Hardware-Software Interface (HSI)* definition. In the author's opinion, this interface is a fundamental artefact and central point throughout the product development life cycle. The HSI is the connection between the hardware and software components in the meta model (see Figure 4.1) and details about this interface, also with a view to safety-critical applications, can be found in **Paper E**.

[1]`http://www.mathworks.com/`
[2]`https://www.dspace.com/`

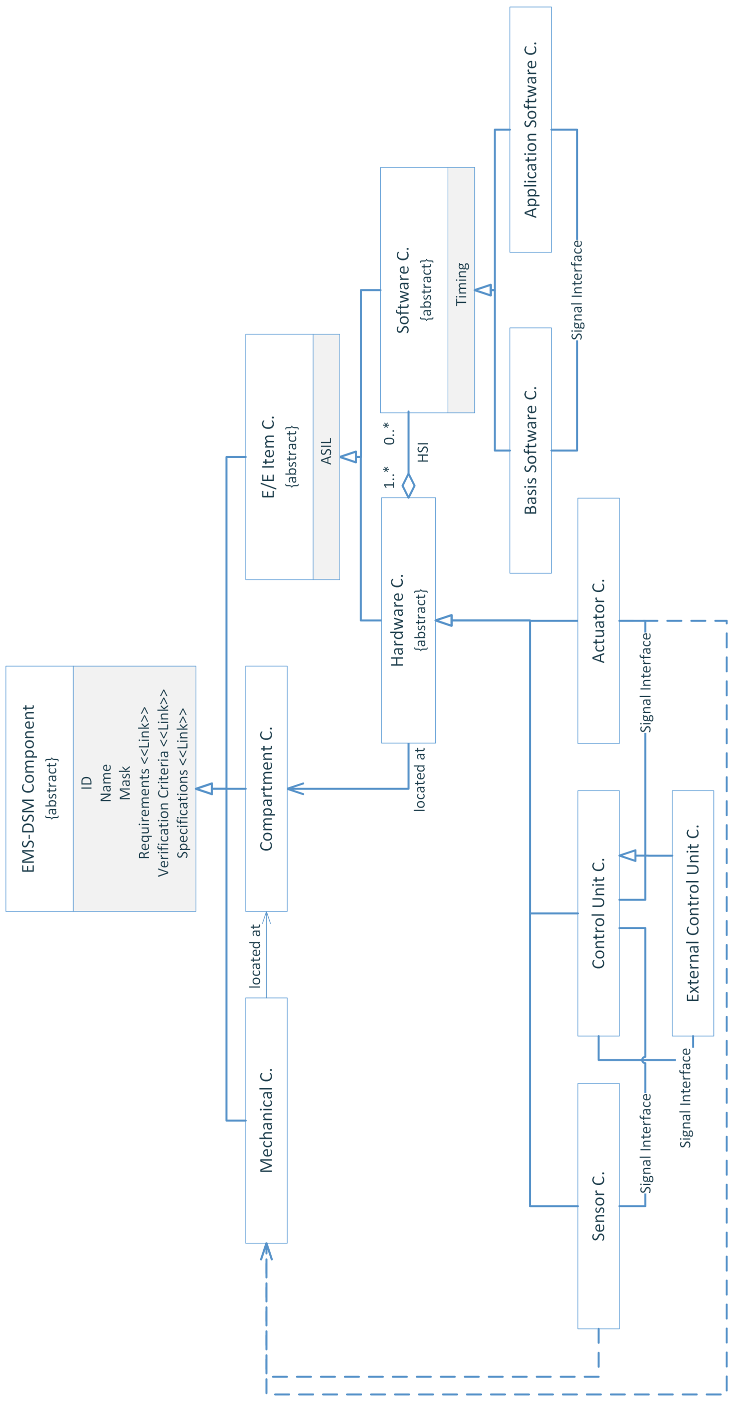


Figure 4.1: Definition of the *Embedded Mechatronic System Domain-Specific Modelling (EMS-DSM)* meta model

5 Application of the PRM and DSM

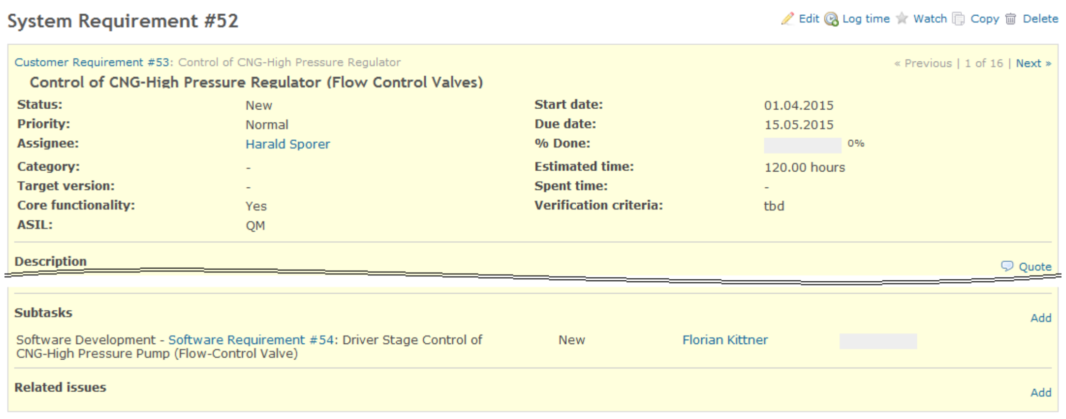
In Chapter 3 the hardware and mechanical extensions for a complete mechatronic system development to a state-of-the-art engineering process reference model (*Automotive SPICE*) were presented. In Chapter 4 the meta model to facilitate the process of creating the system and software architectural design was introduced. To enhance an application of the outlined methodologies and techniques, an adequate tool support has been created in the course of this research work and was presented at different conferences. Hence, this chapter is prepared as a road map to the detailed information in the proceedings of these conferences, which can be found in Chapter 7.

The application of the methodologies introduced in Chapter 3 and 4 may be separated into the two categories (1) document-centric development artefact support and (2) model-based development artefact support. The first category comprising of the management of all artefacts that are textually notated, such as the various types of requirements and specifications along the product life cycle. The second category supports the management of the model-based development artefacts, such as the architectural designs at system and software level. For an application of the methodologies related to the first category, the open source and web based project management tool *Redmine*[1] (see Figure 5.1 has been used. The required customization of this tool and the interface towards the model-based toolchain is described in **Paper B**. For the application of the methodologies related to the second category, an open source diagram editor [14] has been chosen as a basis to start the tool development from. The EMS-DSM meta model described in Chapter 4 has been implemented in the custom-made tool, which was named *EASy Design* (see Figure 5.2). **Paper C** presents the implementation of the meta model in EASy Design. Moreover the utilization of the *Microsoft Modeling SDK for Visual Studio - Domain-Specific Languages (MSDK-DSL)* (see Section 2.4) for the herein described approach is presented in this publication. **Paper D** outlines a concept called *Open Toolbox Access*, which proposes a tool support approach that enables the small development entity to easily customize the object library comprising of the instantiated EMS-DSM meta model components.

In addition to the general description of the approach's application in the publications

[1]`http://www.redmine.org/`

mentioned in the previous paragraph, for every aspect of the approach presented in this thesis, a use-case has been prepared to demonstrate the application of the particular methodology.



System Requirement #52

Edit Log time Watch Copy Delete

Customer Requirement #53: Control of CNG-High Pressure Regulator

« Previous | 1 of 16 | Next »

Control of CNG-High Pressure Regulator (Flow Control Valves)

Status:	New	**Start date:**	01.04.2015
Priority:	Normal	**Due date:**	15.05.2015
Assignee:	Harald Sporer	**% Done:**	0%
Category:	-	**Estimated time:**	120.00 hours
Target version:	-	**Spent time:**	-
Core functionality:	Yes	**Verification criteria:**	tbd
ASIL:	QM		

Description

Quote

Subtasks

Add

Software Development - Software Requirement #54: Driver Stage Control of CNG-High Pressure Pump (Flow-Control Valve)	New	Florian Kittner	

Related issues

Add

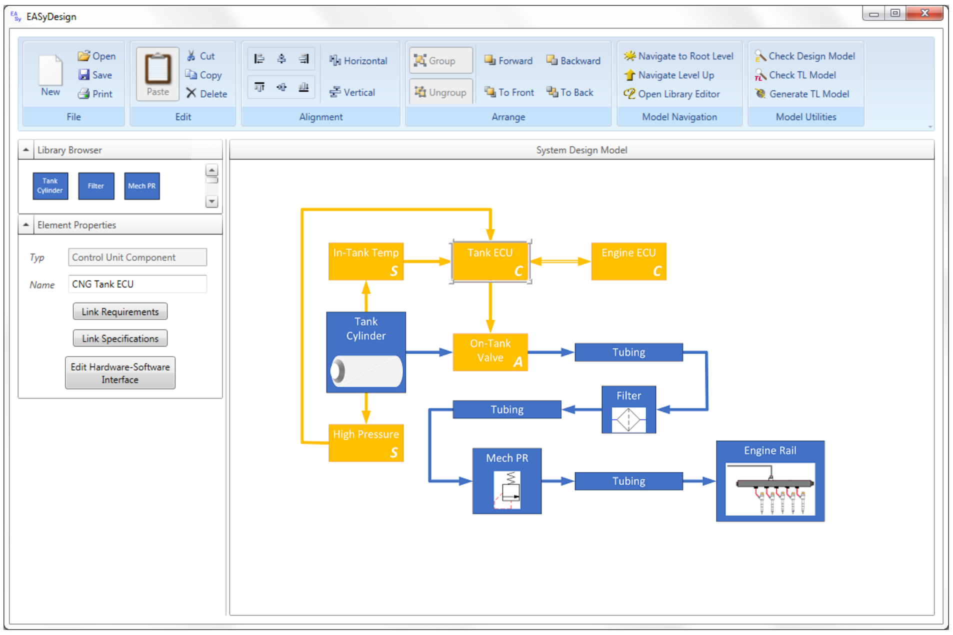
Figure 5.1: System Requirement at *Redmine*

So far, the methodologies presented in this thesis have been applied at two companies, both complying to the category of small development entities. The first one is a small-sized enterprise with less than ten employees (*Company A*). The second one is a large enterprise, but with no experience at hardware and software development so far (*Company B*). A small-sized team was set up a few weeks ago to develop mechatronic systems for a new business segment of the company. Therefore well-defined processes in terms of quality management, such as ISO 9001, are established at the latter company, but out of the E/E system development view, the situations of the two companies are similar.

At *Company A* approximately half of the processes of the PRM have been introduced. For the non-model-based part of the work-products, such as the various kinds of requirements, *Redmine* was installed on the company's server.

At *Company B* the process reference model has been tailored to the specific needs and is under review now. For the management of requirements, test case specifications, reports, etc. the software tool *JIRA* including the document management application *Confluence* has been selected and recently installed. Both, *Redmine* and *JIRA* provide a web-based interface that allow the developers easy access to these tools. The tools have been adapted to the PRM needs as described in **Paper B**.

At both companies the tool support for the presented DSM meta model is in progress. Due to the time an establishment of the proposed methodologies and techniques in a



Typ: Control Unit Component

Name: CNG Tank ECU

Figure 5.2: CNG Tank System Use-Case in EASy Design

productive environment takes, the evaluation through an industrial case study is out of scope of this thesis.

6 Conclusion and Future Work

This chapter concludes this thesis by briefly summarizing the presented methodologies and techniques and outlines potential future work.

6.1 Summary and Conclusion

The presented approach tackles the major objective - *facilitating the embedded mechatronic system development carried out by small entities* - from two starting points. First, a complete process reference model for embedded systems has been presented. Combined with non-technical aspects related to small teams, a pattern for establishing these engineering process definitions has been specified. Second, special attention has been paid to enhancing the system architectural design for domain-experts who are not very familiar with UML-based modelling, which represents the backbone of the majority of the existing approaches in this field. This has been achieved by the definition of a meta model for domain-specific modelling in the context of mechatronic system development, with specific attention to automotive needs. Moreover, an integration of the research work into existing approaches has been described and shown by various use cases. For instance, the mentioned UML-based methodologies and techniques for embedded automotive system design are not ignored or decried by this work. Instead, potential transformation strategies between the domain-specific models of this work and UML-based models of other contributions has been proposed.

As a counterpart to the system architectural design, special focus has also been laid upon the management of the different types of requirements along the product development cycle. There is a big gap between managing the requirements at micro/small and medium/large sized development entities. In terms of *Return of Investment* it is often hard for small teams or companies to argue for the utilization of state of the art requirements management tools, which are available on the market but are in most cases too expensive by far for small entities. Feasible alternatives has been integrated to the overall approach and shown in this work.

Implementing engineering processes and appropriated tools from scratch is never an easy task. From the author's experience there is no such thing as *the* for everyone most suitable procedure. The development team or company where the processes shall be established may be seen as a dynamic framework and thus the detailed strategy for implementing the particular processes has to be flexible too. As already mentioned in Chapter 3, doing something is always better than doing nothing in this context. For example, if there are in the short term no resources left for creating a seamless and fully automatic tool support, simple strategies may be chosen to create as many work products from the defined processes as possible.

6.2 Future Work

While initially focussed on the automotive sector, further industrial projects have shown that the proposed ideas in this research work are compatible with other industry branches too. Some of the aspects seems to be even more suitable for the mechatronic system development in non-automotive engineering domains, simply due to the lower dissemination of the presented kinds of methodologies and techniques. This may be seen as an opportunity for sound research and subsequently provide support in other domains.

As mentioned previously, so far the methodologies presented in this thesis have been applied in two companies, both complying to the category of small development entities, but with a different history related to their time in business. *Company A* is a small-sized enterprise with less than ten employees, whereas *Company B* represents a large enterprise, but with no experience of hardware and software development so far. The latter has slight advantages in terms of the established ISO 9001 quality management system, but out of the E/E system development view, the situations of the two companies are similar.

At *Company A* approximately half of the processes of the PRM have been introduced. For the non-model-based part of the work-products, such as the various kinds of requirements, *Redmine* is utilized. At *Company B* the process reference model has been tailored to the specific needs and is under review now. For the management of requirements, test case specifications, reports, etc. the software tool *JIRA* including the document management application *Confluence* has been selected and recently installed. At both companies the tool support for the presented DSM meta model is in progress. Due to the time an establishment of the proposed methodologies and techniques in a productive environment takes, the evaluation through an industrial case study is out of scope of this thesis and shall be presented in future publications.

7 Publications

This chapter quotes publications that have been peer reviewed and presented by the author at international conferences. They are sorted according to their content starting from a more general towards a specific view on the approaches introduced mainly in the Chapters 4 and 5.

Paper A: H. Sporer. A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design. In *International Conference on Research in Adaptive and Convergent Systems*, pages 357-362, RACS '15, Prague, Czech Republic, ACM New York, 2015.

Paper B: H. Sporer, G. Macher, C. Kreiner, and E. Brenner. A Lean Automotive E/E-System Design Approach with Integrated Requirements Management Capability. In D. Weyns, et al., editors, *Software Architecture*, volume 9278 of *Lecture Notes in Computer Science*, pages 251-258. 9th European Conference on Software Architecture, ECSA '15, Dubrovnik/Cavtat, Croatia, Springer International Publishing, 2015.

Paper C: H. Sporer and E. Brenner. An Automotive E/E System Domain-Specific Modelling Approach with Various Tool Support. *Applied Computing Review (ACR), 16(1)*, ACM Digital Library, 2016. in press.

Paper D: H. Sporer. A Lean Automotive E/E-System Design Approach with Open Toolbox Access. In R. V. O'Connor et al., editors, *Proceedings of the 22nd European Conference EuroSPI 2015, Ankara, Turkey, September 30 - October 2, 2015*, volume 543 of *Communications in Computer and Information Science*, pages 41-50. EuroAsiaSPI '15, Ankara, Turkey, Springer International Publishing, 2015.

Paper E: H. Sporer, G. Macher, C. Kreiner, and E. Brenner. Resilient Interface Design for Safety-Critical Embedded Automotive Software. In J. Zizka et al., editors, *Sixth International Conference on Computer Science and Information Technology*, CCSIT '16, Zurich, Switzerland, pages 183-199. Academy & Industry Research Collaboration Center (AIRCC), 2016.

Paper F: H. Sporer, G. Macher, C. Kreiner, and E. Brenner. A Model-to-Model Transformation Approach at Mechatronics-Based E/E-System Design. In *41st EUROMICRO Conference on Software Engineering and Advanced Applications*, Session on "Work in Progress", SEAA '15, Funchal, Madeira, Portugal, pages 21-22. EUROMICRO, 2015.

Paper G: H. Sporer, G. Macher, E. Armengaud, and C. Kreiner. Incorporation of Model-based System and Software Development Environments. In *41st EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA '15, Funchal, Madeira, Portugal, pages 177-180. IEEE, 2015.

Paper H: H. Sporer, G. Macher, A. Höller, and C. Kreiner. Bidirectional Crosslinking of System and Software Modeling in the Automotive Domain. In A. Fantechi and P. Pelliccione, editors, *Software Engineering for Resilient Systems*, volume 9274 of *Lecture Notes in Computer Science*, pages 99-113. SERENE '15, Paris, France, Springer International Publishing, 2015.

A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design

Harald Sporer
Institute of Technical Informatics
Graz University of Technology
Inffeldgasse 16/1
Graz, Austria
sporer@tugraz.at

ABSTRACT

The electrical and electronic systems (E/E-Systems) in the automotive world have been getting more and more complex over the past decades. New functionality, which is mainly realized through embedded E/E-Systems, as well as the growing connectivity (*Car2X-Communication*), will keep this trend alive in the upcoming years. Additionally, new standards and regulations have been released during the last years (e.g. ISO 26262), which leads to an even higher system complexity. Therefore, well-defined development processes are crucial to manage this complexity and achieve high quality products. To accomplish an appropriated guidance through these processes, a tool chain has to be established, which supports each phase of the E/E-System development. However, it isn't enough to provide a stand-alone solution for the assistance at each phase. A smooth transition of the development artefacts between the different levels as well as their bilateral traceability is crucial. Common approaches utilize tools like Enterprise Architect or Artisan Studio to model the E/E-System design in SysML or a kind of UML2 profile. Usually, several abstraction layers are designed with these tools, starting from the system architectural design down to the software architectural design. Although, in the majority of cases the design should represent a mechatronics-based system, the hardware as well as the mechanics view is not considered. The aim of this work is to remedy the deficiencies regarding the missing representation of hardware and mechanics artefacts within the E/E-System design. Therefore, a model-based domain-specific language was developed that describes the system in a more comprehensive way. Additionally, it makes it easier for domain experts, who are not that familiar with UML or SysML, to create an architectural design. Furthermore, the already existing SysML models are not ignored at the presented methodology, but supported through a translator, which converts the DSL model into a SysML representation.

RACS '15, October 09-12, 2015, Prague, Czech Republic
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3738-0/15/10...$15.00
DOI: http://dx.doi.org/10.1145/2811411.2811533

CCS Concepts

•Computer systems organization → Embedded systems; •Software and its engineering → Software design engineering;

Keywords

automotive embedded systems; system architectural design; domain-specific modeling; E/E-Systems

1. INTRODUCTION

The number of functionalities realized through E/E-Systems at modern cars, and therefore the overall complexity, will keep increasing over the next years. Connecting the cars with their environment, as well as new propulsion technologies will foster this trend. The potential concerning product differentiation between competing companies as well as the possibility to optimize existing E/E-System functionalities is enormous.

High quality standards along the whole product life cycle are crucial to cope with the upcoming challenges. To achieve this, methods and techniques from concepts like Automotive SPICE [2] are strongly recommended. Some of the key aspects of these concepts are bilateral traceability, as well as consistency between the different design abstraction levels, starting from a system design down to a detailed software component design.

In the embedded automotive sector, system design models are usually created with techniques based on the *Unified Modeling Language (UML)*. Either the meta-model is extended, or a profile is created to enable the UML-based approach for the embedded automotive system design. A wide-spread example of an UML2 profile is SysML, which reuses many of the original diagram types (*State Machine Diagram*, *Use Case Diagram*, etc.), uses modified diagram types (*Activity Diagram*, *Block Definition Diagram*, etc.), and adds new ones (*Requirement Diagram*, *Parametric Diagram*) [6].

Even if the UML-based methodologies are valuable for projects with emphasis on software, for the embedded automotive system design, sometimes they are too powerful due to the numerous representation options. In particular for domain experts who have no or limited knowledge about software development, the high number of elements available for modeling, turns the system architectural design into an awkward task. However, it is not the intention of this work to decry the SysML approaches created so far. They are a good choice for a multitude of tasks.

Instead, this paper showcases an extension to these SysML approaches, to ease the architectural design of embedded mechatronics system designs for UML-non-natives. Therefore, a model-based domain-specific language, respectively a domain-specific modeling (DSM) for the specific needs of embedded automotive mechatronics systems, has been developed. Additionally, a software tool has been created to support the new DSM.

By linking development artefacts like requirements (e.g. technical system requirements, software requirements, etc.), and verification criteria to the design model, the earlier mentioned traceability is assured.

In the course of this document, section 2 presents an overview of the related approaches, as well as of domain-specific modeling and integrated tool chains. In section 3, a description of the proposed DSM approach for the model-based system engineering is provided. An application of the described methodology is presented in section 4. Finally, this work is concluded, with an overview of the presented work, in section 5.

2. RELATED WORK

In recent years, a lot of effort has been made to improve the automotive model-based E/E-System design methods and techniques. Traceability, as well as consistency, between the development artefacts has always been an important topic. However, due to the increasing number of electronic- and electric-based functionality, these properties have become vital.

If it comes to safety-critical functionalities, according to the 2011 released international standard ISO 26262, traceability between the relevant artefacts is mandatory [9]. A description of the common deliverables along an automotive E/E-System development, and a corresponding process reference model is presented by the de-facto standard Automotive SPICE [2]. Neither the functional safety standard nor the process reference model enforces a specific methodology, how the development artefacts have to be linked to each other. However, connecting the various work products manually is a tedious and error-prone task.

In [11] the authors describe a seamless model-based tool chain orchestration for the automotive system and software engineering domain. As at other contributions in this field ([3], [1], [7], [10], [13]), SysML is utilized for the system architectural design.

To agree with Broy et al. [4], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities of how to customize the UML diagrams, to get a language for embedded system design, drive these drawbacks. On the one hand, the meta model can be extended, and on the other hand, a profile can be defined [13]. Even if there is an agreement to utilize a common UML profile like SysML, a plenty of design artefact variations are feasible.

Another long term research project in the field of system design is the *Ptolemy Project* [15]. Together with the related open source and simulation tool *Ptolemy II*, the project provides an environment for the modeling of heterogeneous cyber-physical systems. With the four integrated syntax classes *block diagrams*, *bubble-and-arc diagrams*, *imperative programs*, and *arithmetic expressions* various design domains can be addressed. This general purpose aspect is the major strength, but simultaneously also a weakness as it is at the UML approaches. Moreover, the bidirectional traceability to requirements and other development artefacts is missing.

The scenario described in this section so far, doesn't provide an optimal base for the engineer who has to design the embedded automotive system from a mechatronics point of view. Ideally, the tool should be intuitive and easily operated also without specific UML knowledge. These findings led the authors to the idea to create a more tailored model-based language for the stated domain.

Mernik et al. [12] describe a domain-specific language as a language that is tailored to the specific application domain. Enhanced by this tailoring, substantial gains in expressiveness and ease of use, compared to general-purpose languages, should be given. Even if a gain regarding the expressiveness is achieved by the utilization of SysML-based modeling techniques, the ease of use regarding an embedded automotive mechatronics system design is out of sight.

Preschern et al. [14] claim that DSLs help to decrease system development costs by providing developers with an effective way to construct systems for a specific domain. The benefit in terms of a more effective development has to be higher than the investment for creating or establishing a DSL at a company or department. Supplementary, the authors argue that in the next years the mentioned DSL development cost will decrease significantly, due to new tools supporting the language creation like the Eclipse-based *Sirius*[1].

Vujović et al. [16] present a model-driven engineering approach to create a domain-specific modeling (DSM). Sirius is the framework for developing a new DSM, respectively the DSM graphical modeling workbench. The big advantage of this tool is that the workbench for the DSM is developed graphically. Therefore, knowledge about software development with Java, the graphical editor framework (GEF) or the graphical modeling framework (GMF) is not needed.

According to Hudak [8], programs written in a DSL are more concise, can be written more quickly, are easier to maintain and reason about. In the authors opinion, this list of advantages is also valid for domain-specific modeling. Furthermore, Hudak determines the basic steps for developing a own domain-specific language as *(a) Definition of the domain, (b) Design of the DSL capturing the domain semantics, (c) Provide support through software tools,* and *(d) Create use-cases for the new DSL infrastructure.* The approach described in this paper is highlighted, according to theses steps, in section 3 and 4.

3. APPROACH

The main goal of this contribution is to provide a lean approach for engineers to facilitate an embedded automotive mechatronics system modeling on a high abstraction level. The focus of the described approach is on the model-based structural description of the E/E-System under development. Additionally, the signals and interfaces are essential parts of the modeling. The described methods and techniques at section 2 are well-defined basis for this work.

The existing SysML-based design method is extended by the newly developed *Embedded Mechatronics System Domain-*

[1]https://eclipse.org/sirius/

Specific Modeling (EMS-DSM) for the automotive embedded system design. It is not intended to replace the SysML-based solution created so far. Instead, the EMS-DSM is integrated into the existing approach, and the whole tool-chain, starting from the SysML-based system architectural design tool, down to the software / hardware architectural design, can be utilized if desired. An overview of the tool integration is shown in figure 1.

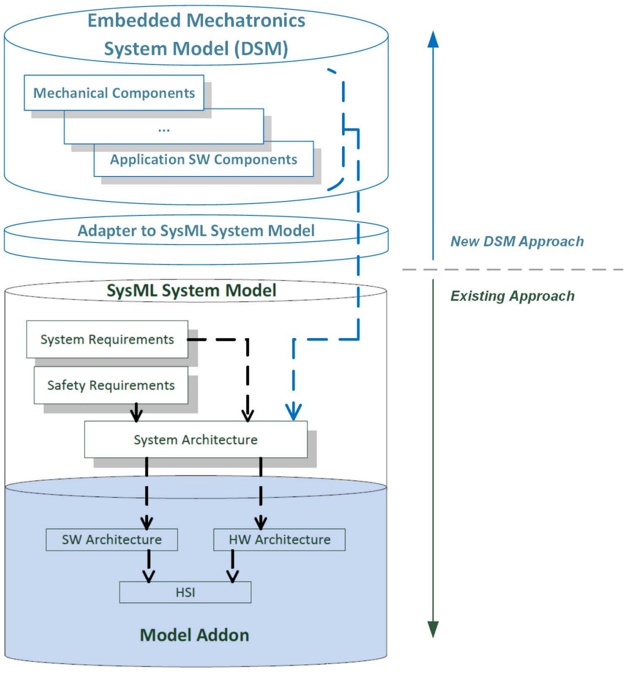


Figure 1: Tool-Chain Integration of DSM and Existing SysML Model Approach (based on [11])

For topics like project management and requirements management, the web-based open source application *Redmine*[2] is used at this approach. Owing to its high flexibility through configuration, new trackers, which also reflect the engineering process, are added:

- Functional System Requirement
- Technical System Requirement
- Hardware Requirement
- Software Requirement
- System Test Case
- Hardware Test Case
- Software Test Case

The test case and requirement items are connected to each other by their unique identifier. The relationship between them is shown in figure 2. For a safety-critical development according to ISO26262, additional issue types like *Functional Safety Requirements* are added.

[2]http://www.redmine.org/

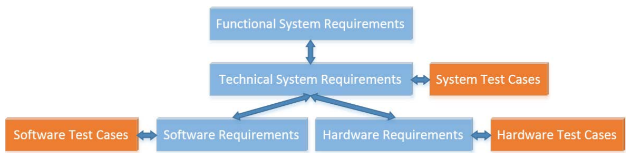


Figure 2: Requirements Hierarchy and Test Case Relationships

3.1 Definition of the Domain

This work focus on the *Embedded Mechatronics E/E-System Design* in the automotive field. This can be seen as the meta-domain of the model-based language. The EMS-DSM itself is tailored to the needs of the domain at the particular project or company. E.g. the domain of the presented application in section 4 is *Embedded Mechatronics E/E-System Design for Compressed Natural Gas (CNG) Fuel Tank Systems.*

3.2 Design of the EMS-DSM

The definition of the newly developed model-based domain-specific language is divided into four levels as shown in figure 3. These levels were introduced for a categorization of the derivations only. They do not depict any order of the component instances at the design model.

EMS-DSM Level 1

The *EMS-DSM Component* at level 1 is the origin of all other classes at the language definition. The five properties of this class are

- *ID:* unique identifier of the particular instance at the design model, set automatically
- *Name:* name or short description of the particular instance, chosen by the design engineer
- *Requirement:* in this approach, a link to the Redmine requirements database is set by the designer
- *Verification Criteria:* similar to the Requirement, a link to the Redmine verification criteria artefact is set by the designer
- *Specification:* link to further information about the actual component, e.g. a CAD drawing or a data sheet

The abstract *EMS-DSM Component* serves as the base node of the EMS-DSM definition, and declares the common properties of the derived classes at the lower levels. Therefore this component is not instanced.

EMS-DSM Level 2

At the second level, the following component classes are available:

- *Mechanical Components:* used by all mechanical, domain-specific components, e.g. the *Mechanical Pressure Regulator* class at the use-case shown in section 4
- *Compartment Components:* gives the opportunity to specify areas or compartments, where mechanical and hardware components are installed

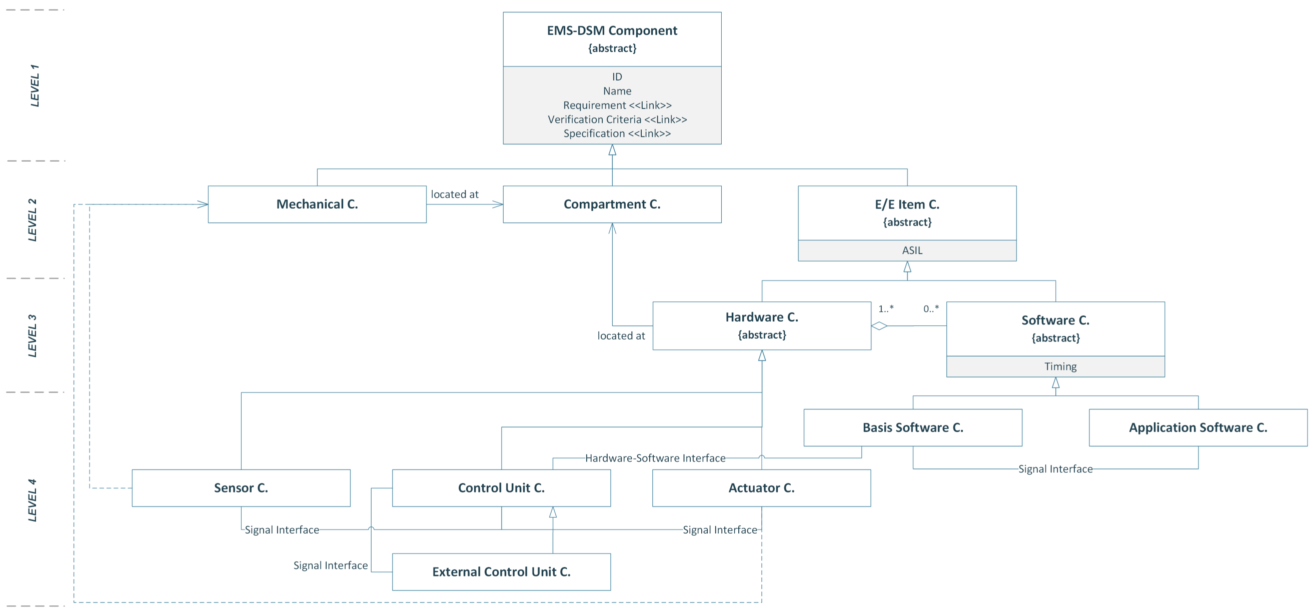


Figure 3: *EMS-DSM* Definition (UML)

- *E/E Item Components:* an abstract component class definition, which serves as basis for the hard- and software components at the lower levels. Additionally, the property *ASIL*, corresponding to the ISO 26262, is declared

EMS-DSM Level 3

At this level, the abstract classes *Hardware Component* and *Software Component* are defined as basis for the derived classes at level 4. The property *Timing* is added at the software component class, which defines the scheduling of the software components at the subsequent level.

EMS-DSM Level 4

The majority of the non-abstract component classes are defined at this level. From the hardware component derived classes are:

- *Sensor Component:* used for all domain-specific sensor components
- *Control Unit Component:* used for all domain-specific control unit components
- *Actuator Component:* used for all domain-specific actuator components
- *External Control Unit Component:* special class, to make signals from an external system available at the considered system

All hardware components, respectively their instances at the system design model, exempt the *External Control Unit Component*, are capable of creating a software design model. That is, any kind of software component instance is only allowed to be implemented at a software design model which belongs to an instance of a hardware component. In view of smart sensors / actuators, not only the *Control Unit Component*, but also the *Sensor* and *Actuator Components* were enabled for this procedure.

To conclude the EMS-DSM level description, from the software component derived classes are:

- *Basis Software Component:* used for all low-level, hardware-dependent software components
- *Application Software Component:* used for all functional software components

3.3 EMS-DSM Features

In this paper an overview of the novel domain-specific modeling definition is presented. The features, as well as the benefits compared to other approaches in this field, are described in detail in further publications. Nevertheless, a short overview over the main features shall be given:

- *Integrated Requirements Management Capability* - Core Functionality Requirements and Ancillary Functionality Requirements are specified and stored at a MySQL database. By utilizing the ADO.net driver for MySQL, one or more requirements can be linked to each component at the architectural design.
- *Incorporation of System and Software Development Environments* - The system as well as the software architectural design can be created in the provided environment. Supported by both, an export and an import functionality, the software architectural design at the EMS-DSM model can be transferred to an Simulink framework model, and can also be created from an Simulink model.
- *Model-to-Model Transformation at the System Design Level* - As mentioned in the previous section, the EMS-DSM shall not replace the established SysML approach. Instead, it can be seen as an extension and the models can be bidirectionally transformed into the different representations.

- *Open Modeling Toolbox Access* - Utilizing a library editor, the modeling toolbox can be adapted to the needs of the respective company or project easily.

3.4 EMS-DSM Tool Support

Generally speaking, the EMS-DSM can be supported by a various number of tools, but at the time when the research project was initiated, a highest possible flexibility, as well as full access to the tools source code was desired.

To avoid an application development from scratch, the open source project *WPF Diagram Designer* has been chosen as a basis to start the tool development from [5]. The corresponding documentation has close to 500.000 views and the source code has been downloaded more than 20.000 times. Therefore, it's a quite good reviewed source which provides standard functionality like file handling and basic graphical modeling. The source code is written in C# and provides good expandability. New functionalities have been implemented at the diagram designer, named *EASy-Design (**E**mbedded **A**utomotive **Sy**stem-Design)*, to facilitate the EMS-DSM engineering.

However, EASy-Design is just one possibility for an EMS-DSM tool support. The methodology, respectively its C# implementation can be ported to e.g. Enterprise Architect[3] by the provided *Add-in* mechanism. Another alternative is the already mentioned Eclipse[4] project *Sirius*, which enables the creation of an graphical modeling workbench, by facilitating the Eclipse modeling technologies, without writing code. Instead, the DSM is implemented by graphical modeling.

4. APPLICATION

At section 3, the first three steps towards developing a DSL / DSM, as defined by Hudak, are shown. Below, the last step *Create use-cases for the new DSL infrastructure* is described. Therefore, the EMS-DSM approach is applied to the development of an automotive fuel tank system for compressed natural gas (CNG). For an appropriate scale of the use-case, only a small part of the real-world system is utilized. The application should be recognized as an illustrative material, reduced for internal training purpose for students. Therefore, the disclosed and commercially non-sensitivity use-case is not intended to be exhaustive or representing leading-edge technology.

In figure 4 the EMS-DSM tool *EASy-Design* including the *System Design Model* is shown. The CNG fuel tank system consists of seven mechanical components (coloured blue):

- Tank Cylinder
- Filter
- Mechanical Pressure Regulator
- Gaseous Injector Rail
- 3 x Tubing

The medium flow between mechanical components, which is CNG in this use-case, is displayed by blue lines with an arrow at the end.

Furthermore, five hardware components (coloured yellow) are placed at the *System Design Model* level:

[3]http://www.sparxsystems.com/

[4]http://eclipse.org/

- In-Tank Temperature (Sensor Component)
- CNG High Pressure (Sensor Component)
- On-Tank Valve (Actuator Component)
- Tank ECU (Control Unit Component)
- Engine ECU (External Control Unit Component)

The signal flow between the components is displayed by yellow lines, ending with an arrow. Between the Control Unit and the External Control Unit component, a communication bus is inserted, characterized by the double compound line type and arrows on both ends.

In figure 3, dependencies are defined between *Mechanical Components* and *Sensor Components*, respectively *Hardware Components*. These relationships enable the direct connection between e.g. the *On-Tank Valve Acutator* (as a hardware component) and the *Tank Cylinder* (as a mechanical component).

Software Components can not be placed on the System Design Level. With a double-click on a Hardware Component, the next modeling level is opened (named *E/E Item Design Level*). Here, the green coloured *Basis Software Components* and *Application Software Components* are put in place.

By double-clicking a connection between two components, a dialogue is opened and the signal, or in case of a communication bus, the signals can be specified. The properties of each component, like the name or the link to the corresponding requirement, are easily set by selecting the particular with a single click, and entering the data at the *Element Properties* toolbox.

5. CONCLUSIONS

The described model-to-model transformation approach combines the advantages of the lean domain specific modeling technique, which is best suitable for a rough system design by automotive domain experts with limited SysML skills, and the established SysML system modeling methodology.

The overall cost efficiency is an further advantage of this approach, which especially makes it interesting for a lean development at small companies or project teams. Redmine, as the project and requirements management tool, and EASy-Design are open source tools. For the SysML system model design, Enterprise Architect is used, which is also reasonable priced.

Although the adapter is currently in a first trial phase for industrial project applicability, evidences of the approaches benefits are already present. Moreover, important topics for future work, like the transformation of the SysML model into the DSM representation (also for consistency check features), and the transformation of a system of systems model in one step, have been identified.

6. REFERENCES

[1] E. Andrianarison and J.-D. Piques. SysML for embedded automotive Systems: a practical approach. In *Conference on Embedded Real Time Software and Systems*. IEEE, 2010.

[2] Automotive SIG. Automotive SPICE®Process Assessment Model. Technical report, The SPICE User Group, May 2010. Version 2.5.

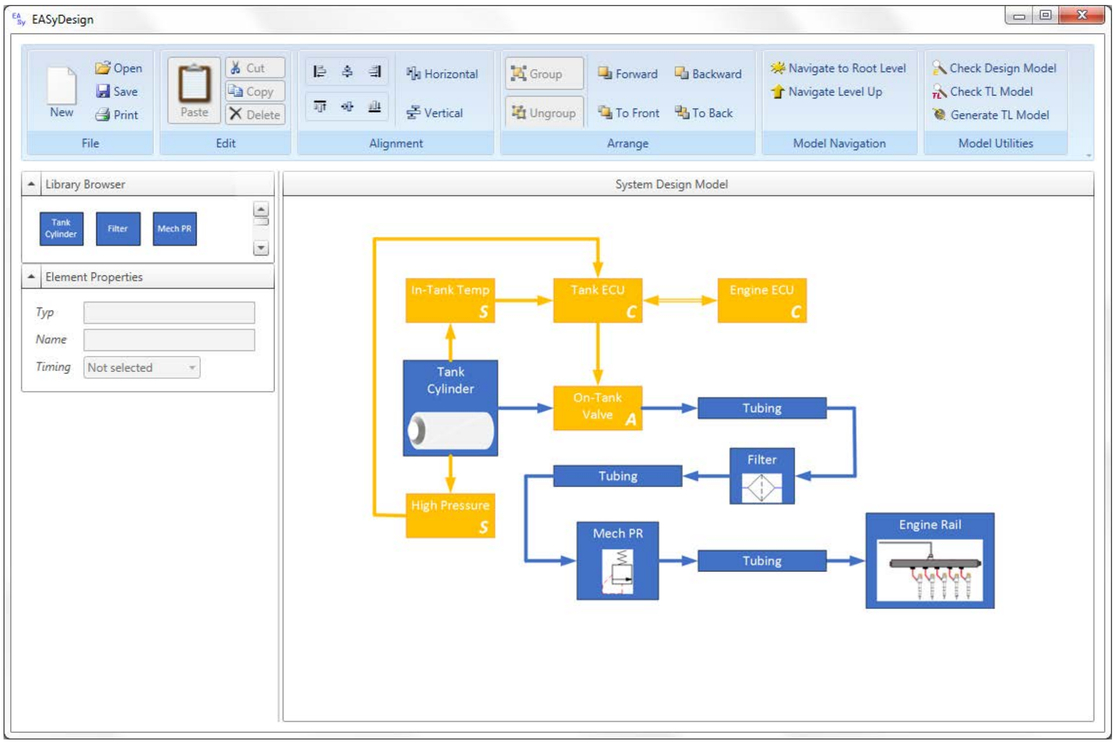


Figure 4: *EASy-Design* System Model

[3] R. Boldt. Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group, July 2009.

[4] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526–545, 2010.

[5] Code Project. WPF Diagram Designer - Part 4. Online Resource, March 2008. http://www.codeproject.com/Articles/24681/WPF-Diagram-Designer-Part, accessed Mar 2015.

[6] S. Friedenthal, A. Moore, and R. Steiner. OMG Systems Modeling Language (OMG SysML^{TM}) Tutorial. In *INCOSE International Symposium*, 2006.

[7] H. Giese, S. Hildebrandt, and S. Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages 555 –579, 2010.

[8] P. Hudak. Domain-specific languages. *Handbook of Programming Languages*, 3:39–60, 1997.

[9] ISO 26262, Road vehicles - Functional safety. International standard, International Organization for Standardization, Geneva, CH, November 2011.

[10] R. Kawahara, H. Nakamura, D. Dotan, A. Kirshin, T. Sakairi, S. Hirose, K. Ono, and H. Ishikawa. Verification of embedded system's specification using collaborative simulation of SysML and simulink models. In *International Conference on Model Based Systems Engineering (MBSE'09)*, pages 21–28. IEEE, 2009.

[11] G. Macher, E. Armengaud, and C. Kreiner. Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In *7th European Congress Embedded Real Time Software and Systems Proceedings*, pages 256–263, 2014.

[12] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[13] J. Meyer. *Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards*. PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Germany, July 2014.

[14] C. Preschern, N. Kajtazovic, and C. Kreiner. Efficient development and reuse of domain-specific languages for automation systems. *International Journal of Metadata, Semantics and Ontologies*, 9(3):215–226, 2014.

[15] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[16] V. Vujović, M. Maksimović, and B. Perišić. Sirius: A rapid development of DSM graphical editor. In *18th International Conference on Intelligent Engineering Systems (INES)*, pages 233–238. IEEE, 2014.

A Lean Automotive E/E-System Design Approach with Integrated Requirements Management Capability

Harald Sporer( ), Georg Macher, Christian Kreiner, and Eugen Brenner

Institute of Technical Informatics, Graz University of Technology,
Inffeldgasse 16/1, 8010 Graz, Austria
{sporer,georg.macher,christian.kreiner,brenner}@tugraz.at
http://www.iti.tugraz.at/

Abstract. Replacing former pure mechanical functionalities by mechatronics-based solutions, introducing new propulsion technologies, and connecting cars to their environment are only a few reasons for the still growing E/E-System complexity at modern passenger cars. Smart methodologies and processes are necessary during the development life cycle to master the related challenges successfully. In this paper, a lean approach for a model-based domain-specific E/E-System architectural design is presented. Furthermore, an integrated requirements management methodology is shown, satisfying the needs for a full traceability between the requirements and design artifacts. The novel model-based language allows domain experts, with limited knowledge of the de-facto system design standard SysML, to describe the mechatronics-based system easily and unambiguously. The lean tool chain orchestration makes the presented approach, especially but not limited to, interesting for small project teams.

Keywords: Automotive embedded E/E-systems · System architectural design · Domain-specific modeling · Requirements management

1 Introduction

The number of functionalities realized through electrical and/or electronic systems (E/E-Systems) at modern cars, and therefore the overall complexity, will keep increasing over the next years. Connecting the cars with their environment, as well as new propulsion technologies will foster this trend. The potential concerning product differentiation between competing companies as well as the possibility to optimize existing E/E-System functionalities is enormous.

High quality standards along the whole product life cycle are crucial to cope with the upcoming challenges. To achieve this, methods and techniques from concepts like Automotive SPICE [1] are strongly recommended. Some of the key aspects of these concepts are bidirectional traceability, as well as consistency between the different development artifacts. Regardless what kind of tool chain

D. Weyns et al. (Eds.): ECSA 2015, LNCS 9278, pp. 251–258, 2015.
DOI: 10.1007/978-3-319-23727-5_20

is chosen to facilitate the product development life cycle, these key concepts must be supported.

In the automotive industry, the E/E-System design models are usually created with techniques based on the *Unified Modeling Language (UML)*. To enable this de facto standard for the embedded automotive system design, either the meta-model is extended or a profile is created. A wide-spread example of an UML2 profile is the *Systems Modeling Language (SysML)*, which reuses many of the original diagram types (*State Machine Diagram*, *Use Case Diagram*, etc.), uses modified diagram types (*Activity Diagram*, *Block Definition Diagram*, etc.), and adds new ones (*Requirement Diagram*, *Parametric Diagram*) [2].

Even if the UML-based methodologies are valuable for projects with emphasis on software, for the embedded automotive system design, sometimes they are too powerful due to the numerous representation options. In particular for domain experts who have no or limited knowledge about software development, the high number of elements available for modeling, turns the system architectural design into an awkward task. However, it is not the intention of this work to decry the SysML approaches created so far. They are a good choice for a multitude of tasks. Instead, this paper showcases an extension to these SysML approaches, which eases the architectural design of embedded mechatronics system designs for UML-non-natives, and provides a comfortable integration of the requirements management processes at the different design abstraction levels. To achieve these goals, a domain-specific modeling (DSM) for the particular needs at the embedded automotive mechatronics-based system development has been created. Moreover, the described design approach has been complemented by a lean requirements management strategy.

In the course of this document, Section 2 presents an overview of the related approaches, as well as of domain-specific modeling and requirements management. In Section 3, a description of the proposed modeling approach with integrated requirements capability is provided. An application of the described methodology is presented in Section 4. Finally, this paper is concluded with an overview of the presented work in Section 5.

2 Related Work

In recent years, a lot of effort has been made to improve the automotive model-based E/E-System design methods and techniques. Traceability, as well as consistency, between the development artifacts has always been an important topic. However, due to the increasing number of electronic- and electric-based functionality, these properties have become vital.

If it comes to safety-critical functionalities, according to the 2011 released international standard ISO 26262, traceability between the relevant artifacts is mandatory [9]. A description of the common deliverables along an automotive E/E-System development, and a corresponding process reference model is presented by the de facto standard *Automotive SPICE* [1]. Neither the functional safety standard nor the process reference model enforces a specific methodology, how the development artifacts have to be linked to each other. However, connecting the various work products manually is a tedious and error-prone task.

In [4] a seamless model-based tool chain orchestration for the automotive system and software engineering domain is described by the authors. As in other contributions in this field ([5], [6], [7]), SysML is utilized for the system architectural design.

To agree with Broy et al. [8], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities of how to customize the UML diagrams, to get a language for embedded system design, drive these drawbacks. This scenario does not provide an optimal base for the engineer who has to design the embedded automotive system from a mechatronics point of view. Ideally, the tool should be intuitive and easily operated also without specific UML knowledge. These findings led the authors to the idea to create a more tailored model-based language for the stated domain. In [3] a detailed description of this domain specific modeling approach can be found.

Regarding the needs for an appropriate requirement handling, Mäder et al. [10] provide evidence that standards like the ISO 26262, which demand full traceability, are not the only argument for implementing a proper requirements management strategy. They conducted an experiment with more than 50 subjects performing maintenance tasks on two projects. Half of the tasks with and the other half without traceability. The result was unambiguous: the subjects with requirements traceability performed on average 21% faster and delivered 60% more correct solutions.

Based on functionality classification, Chemuturi [11] primarily categorize requirements into *Core Functionality Requirements* and *Ancillary Functionality Requirements*, instead of the in the automotive field wide-spread types *Functional Requirements* and *Non-Functional Requirements*. In his opinion, the term *Non-Functional* connotes that the corresponding requirements do not function or do not serve any function. However, even if they may not serve a business process function directly, they are serving a useful purpose in the product. Therefore, he labels requirements corresponding to topics like *Safety*, *Response Time*, and *Memory Constraints* as ancillary functionality requirements. At this approach, the requirements classification of Chemuturi is utilized and adapted to the needs of mechatronics-based systems.

Herrmann et al. [12] depict requirement attributes for different phases during the product development cycle. Additionally, recommendations on their usage are given, supported by the categorization of the attributes into *mandatory*, *reflective*, *optional*, and *not required*. Most of the presented attributes are also used at comprehensive requirements management (RM) tools like *IBM Rational DOORS*[1] and *PTC Integrity Lifecycle Manager*[2]. In this work, the recommen-

[1] http://www.ibm.com/
[2] http://www.ptc.com/

dations of Hermann et al. are taken into account and necessary adjustments, evoked by the automotive E/E-System development domain, are made.

3 Approach

In this section, the domain specific modeling methodology for automotive mechatronics-based system development, with a focus on the integrated requirements management capability, is presented. As mentioned in Section 2, details on the domain specific modeling can be found in [3]. Therefore, just a quick overview is given in the following subsection.

3.1 Domain Specific Modeling Approach

The established SysML-based design method from [4] is extended by the newly developed *Embedded Mechatronics System Domain-Specific Modeling (EMS-DSM)* for the automotive embedded system design. The main goal of this methodology is to provide a lean approach for engineers to facilitate an embedded automotive mechatronics system modeling on a high abstraction level. The focus of the approach is on the model-based structural description of the E/E-System under development. Additionally, the signals and interfaces are an essential part of the modeling.

The definition of the newly developed model-based domain specific language is shown in Figure 1. The top node *EMS-DSM Component* is the origin of all other classes at the language definition. Therefore, each of the derived classes inherits the five properties (*ID*, *Name*, *Requirement*, *Verification Criteria*, and *Specification*) from the base class.

The language definition in Figure 1 represents the meta-domain of the model-based language. Subsequently, the EMS-DSM is tailored to the needs of the domain at the particular project or company. That is, design elements of possible types *Mechnical*, *Compartment*, *Sensor*, *Control Unit*, *Actuator*, *External Control Unit*, *Basis Software*, and *Application Software*, are specified for the particular field of application. E.g. the domain of the presented application in Section 4 is *Embedded Mechatronics E/E-System Design for Compressed Natural Gas (CNG) Fuel Tank Systems.*

The EMS-DSM can be supported by a various number of tools, but at the time when the research project was initiated, a highest possible flexibility, as well as full access to the tools source code was desired. To achieve this, an own model editor (***E****mbedded* ***A****utomotive* ***Sy****stem Design*) has been developed, based on the open source project *WPF Diagram Designer* [13].

3.2 Requirements Classification and Attributes

As mentioned in Section 2, the requirements are primarily categorized into *Core Functionality Requirements* and *Ancillary Functionality Requirements.* Typical

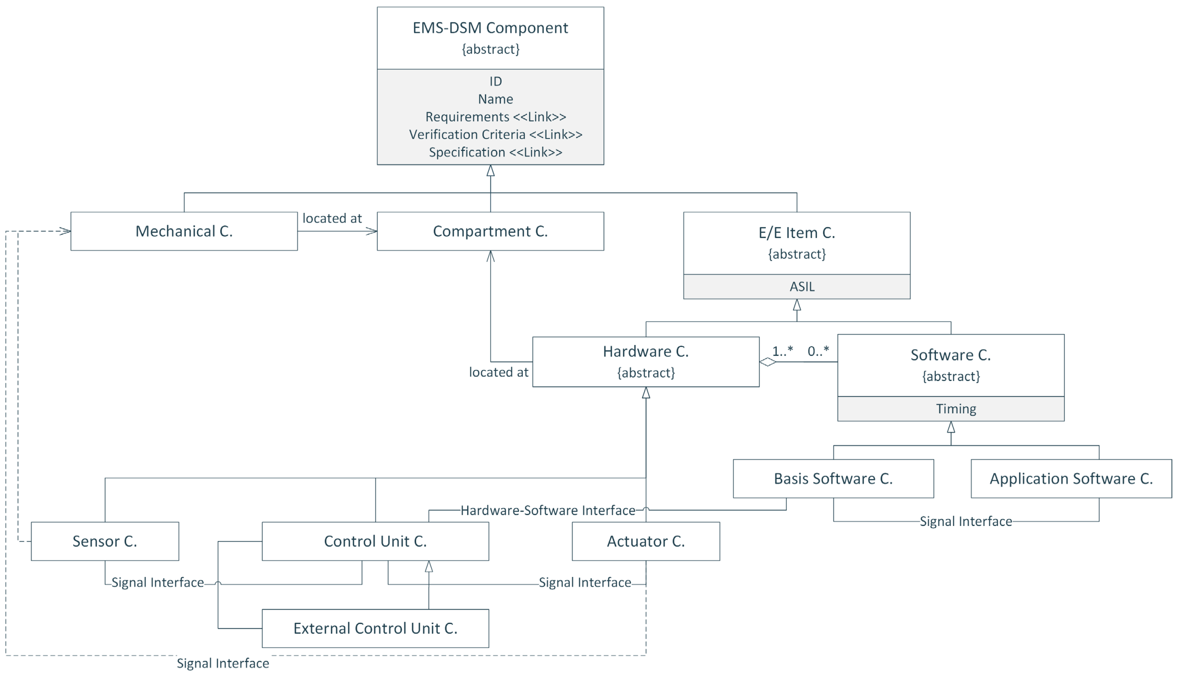


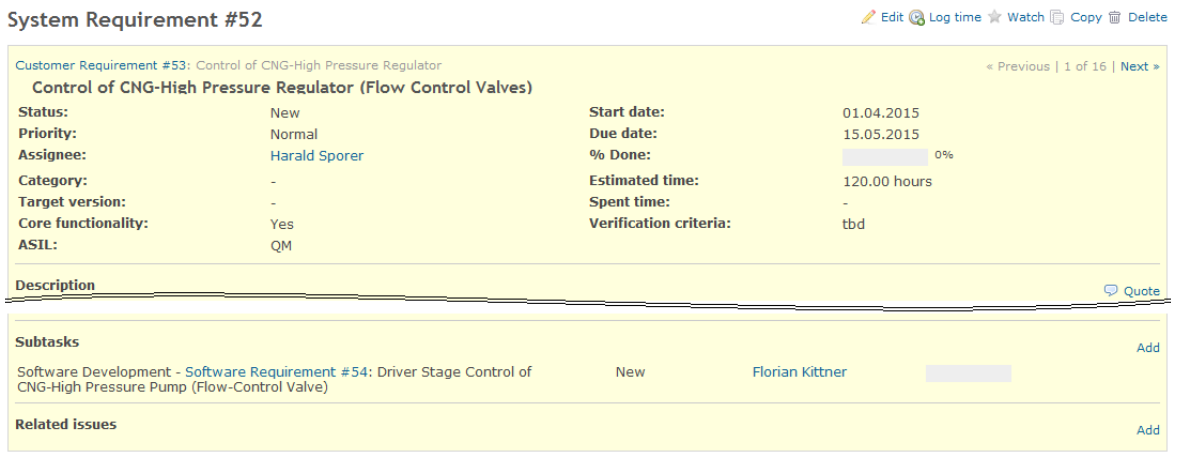
Fig. 1. *EMS-DSM* Definition (UML)

examples for ancillary functionality topics are *Software Footprint*, *Memory Constraints*, *Response Time*, *Reliability*, and *Safety* [11]. By introducing new requirement (issue) attributes, the utilized web-based tool *Redmine*[3] can be adapted to these needs for requirements categorization easily.

The de facto standard Automotive SPICE [1] defines three different types of requirements at the engineering process group: *Customer Requirements*, *System Requirements*, and *Software Requirements*. Out of the embedded E/E-System view, at least the hardware focus is missing. Additionally, requirements and design items regarding the mechanical components, have to be introduced for the design of an embedded mechatronics-based E/E-System. Similar to the Automotive SPICE methodology on system and software level, engineering processes has been defined for these missing artifacts. Summing up, the available requirement and test case types at this work are: *Customer Req*, *System Req*, *System TC*, *System Integration TC*, *Software Req*, *Software TC*, *Software Integration TC*, *Hardware Req*, *Hardware TC*, *Mechanics Req*, and *Mechanics TC*.

By reconfiguring the project management tool Redmine, all mentioned requirement types have been implemented. The most important attributes which have been added are *Core Functionality* (artifact can be marked as contributing to the products core functionality), *ASIL* (shows the automotive safety integrity level of the artifact), and *Verification criteria*. In Figure 2 a system requirement at Redmine is shown. The link to the corresponding costumer requirement is located at the top of the definition. At *Subtasks* the subsequent requirements, e.g. software requirements are listed and to satisfy the demand for full traceability, a link to the corresponding test cases can be added at *Related issues*.

[3] http://www.redmine.org/



System Requirement #52

Customer Requirement #53: Control of CNG-High Pressure Regulator

Control of CNG-High Pressure Regulator (Flow Control Valves)

Status:	New	Start date:	01.04.2015
Priority:	Normal	Due date:	15.05.2015
Assignee:	Harald Sporer	% Done:	0%
Category:	-	Estimated time:	120.00 hours
Target version:	-	Spent time:	-
Core functionality:	Yes	Verification criteria:	tbd
ASIL:	QM		

Description

Subtasks

Software Development - Software Requirement #54: Driver Stage Control of CNG-High Pressure Pump (Flow-Control Valve)	New	Florian Kittner	

Related issues

Fig. 2. System Requirement at *Redmine*

3.3 Bridging the Gap between Design and Requirements

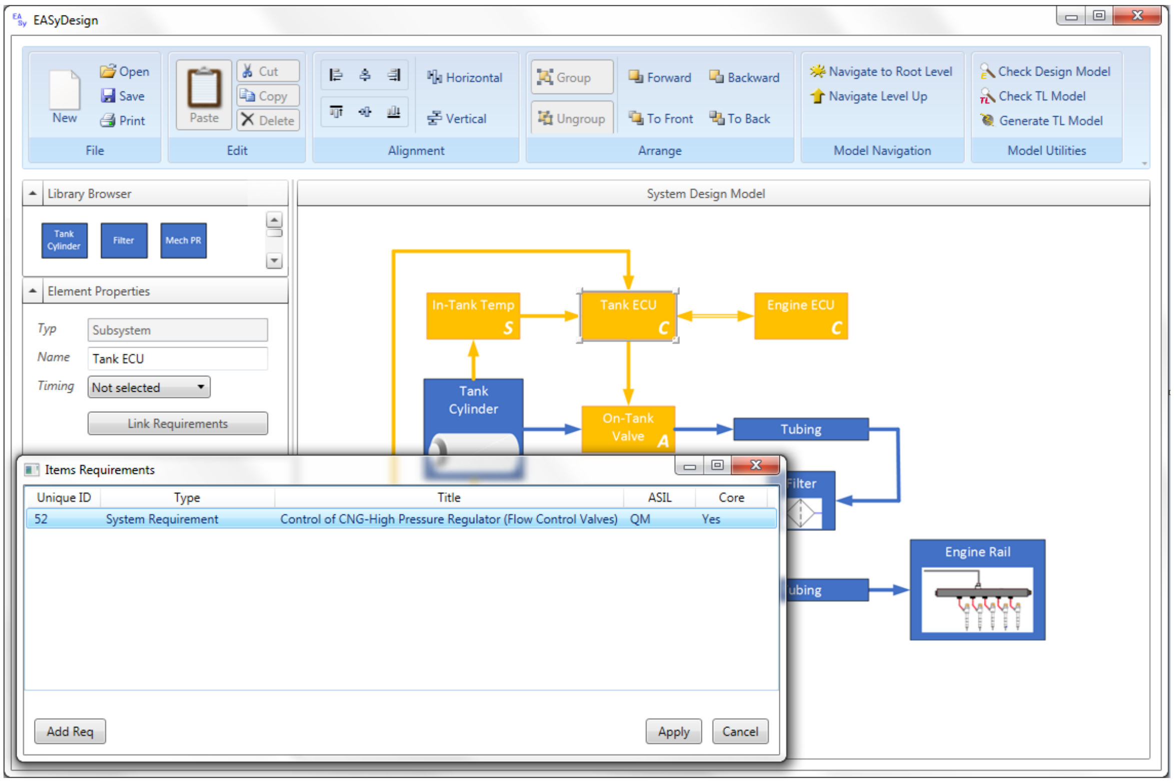
Section 3.1 contains the description of how the different types of designs (system level, software level, etc.) are created corresponding to the novel domain specific modeling. To achieve full traceability, these designs, respectively the various components at the designs, have to be linked to the corresponding requirements. This can be done by the *Requirements Linker* at EASy Design, which establishes a connection to the MySQL database, and therefore has full access to the requirements data at Redmine. By utilizing the *ADO.Net driver for MySQL*[4], the Requirements Linker can easily execute all kinds of MySQL commands on the database.

4 Application

In this section, the EMS-DSM approach with integrated requirements management capability, is applied to the development of an automotive fuel tank system for compressed natural gas (CNG). For an appropriate scale of the use-case, only a small part of the real-world system is utilized. The application should be recognized as an illustrative material, reduced for internal training purpose for students. Therefore, the disclosed and commercially non-sensitivity use-case is not intended to be exhaustive or representing leading-edge technology.

In figure 3 the EMS-DSM tool *EASy-Design* including the *System Design Model*, as well as the *Requirements Linker* dialogue is shown. The CNG fuel tank system consists of seven mechanical components, which are blue coloured (Tank Cylinder, Filter, etc.) The medium flow between mechanical components, which is CNG in this use case, is displayed by blue lines with an arrow at the end. Furthermore, five hardware components are placed at the *System Design Model* level, which are yellow coloured (In-Tank Temperature Sensor, Tank ECU, etc.) The signal flow between the components is displayed by yellow lines, ending with an arrow. Between the Control Unit and the External Control Unit component,

[4] https://dev.mysql.com/



Typ	Subsystem
Name	Tank ECU
Timing	Not selected

Unique ID	Type	Title	ASIL	Core
52	System Requirement	Control of CNG-High Pressure Regulator (Flow Control Valves)	QM	Yes

Fig. 3. Self-developed tool *EASy Design* with Integrated Requirements Management Capability

a communication bus is inserted, characterized by the double compound line type and arrows on both ends.

Software Components can not be placed on the System Design Level. With a double-click on a Hardware Component, the next modeling level is opened (named *E/E Item Design Level*). Here, the green coloured *Basis Software Components* and *Application Software Components* are put in place.

By double-clicking a connection between two components, a dialogue is opened and the signal, or in case of a communication bus, the signals can be specified. By selecting a model element and a click on the button *Link Requirements*, the elements requirements dialogue is opened (shown in Figure 3). Already linked requirements from the Redmine database are listed with their ID, Type, Title, ASIL, and Core functionality attribute. By a click on the button *Add Req*, a connection to the database is established as described in Section 3.3 and a new requirement from the database can be added.

5 Conclusions

In the previous sections, a lean method for the design of embedded automotive mechatronics-based E/E-Systems, with full requirements traceability characteristic, was presented. This approach has the potential to bring together the different engineering disciplines along the E/E-System development. Moreover, it's feasible for automotive domain experts with limited knowledge of UML/SysML.

First use case implementations show promising results. However, there are at least two important functionalities which has to be implemented in a next step. On the one hand, the M2M-Transformator between the EMS-DSM and the SysML model has to be developed. On the other hand, the so far hard coded tool box at EASy Design has to be transferred to a library file that can be adapted also during run time.

References

1. Automotive SIG: Automotive SPICE®Process Assessment Model. Technical report, Version 2.5, The SPICE User Group (2010)
2. Friedenthal, S., Moore, A., Steiner, R.: OMG systems modeling language (OMG SysML^{TM}) tutorial. In: INCOSE International Symposium. INCOSE, Orlando (2006)
3. Sporer, H., Macher, G., Kreiner, C., Brenner, E.: A model-based domain-specific language approach for the automotive E/E-System design. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) (2015) (under review)
4. Macher, G., Armengaud, E., Kreiner, C.: Bridging automotive systems, safety and software engineering by a seamless tool chain. In: 7th European Congress Embedded Real Time Software and Systems Proceedings, pp. 256–263, Toulouse, France (2014)
5. Boldt, R.: Modeling AUTOSAR systems with a UML/SysML profile. IBM Software Group (2009)
6. Andrianarison, E., Piques, J.: SysML for embedded automotive Systems: a practical approach. In: Conference on Embedded Real Time Software and Systems, Toulouse, France (2010)
7. Giese, H., Hildebrandt, S., Neumann, S.: Model synchronization at work: keeping SysML and AUTOSAR models consistent. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Graph Transformations and Model-Driven Engineering. LNCS, vol. 5765, pp. 555–579. Springer, Heidelberg (2010)
8. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: from isolated tools to integrated model engineering environments. Proceedings of the IEEE **98**(4), 526–545 (2010)
9. International Organization for Standardization: ISO 26262. Road vehicles - Functional safety. International Standard, Geneva, Switzerland (2011)
10. Mäder, P., Egyed, A.: Assessing the effect of requirements traceability for software maintenance. In: 28th IEEE International Conference on Software Maintenance (ICSM), pp. 171–180. IEEE (2012)
11. Chemuturi, M.: Requirements Engineering and Management for Software Development Projects. Springer Science & Business Media (2012)
12. Herrmann, A., Knauss, E.: Requirements Engineering und Projektmanagement. Xpert.press, Springer (2013)
13. Code Project - WPF Diagram Designer - Part 4. http://www.codeproject.com/Articles/24681/WPF-Diagram-Designer-Part

An Automotive E/E System Domain-Specific Modelling Approach with Various Tool Support

Harald Sporer
Institute of Technical Informatics
Graz University of Technology
Inffeldgasse 16/1
Graz, Austria
sporer@tugraz.at

Eugen Brenner
Institute of Technical Informatics
Graz University of Technology
Inffeldgasse 16/1
Graz, Austria
brenner@tugraz.at

ABSTRACT

The electrical and electronic systems (E/E Systems) in the automotive world have been getting increasingly complex over the past decades. New functionality, which is mainly realized through embedded E/E Systems, as well as the growing connectivity (*Car2X-Communication*), will keep this trend alive in the upcoming years. Additionally, new standards and regulations have been released during the last few years (e.g. ISO 26262), which improve system properties such as dependability, but also lead to an even higher system complexity. Therefore, well-defined development processes are crucial to manage this complexity and achieve high quality products. To accomplish an appropriated guidance through these processes, a tool chain has to be established, which supports each phase of the E/E System development. However, it is not enough to provide a stand-alone solution for the assistance at each phase. A smooth transition of the development artefacts between the different levels as well as their bilateral traceability is crucial. Common approaches utilize tools such as Enterprise Architect or Artisan Studio to model the E/E System design in SysML or a kind of UML2 profile. Usually, several abstraction layers are designed with these tools, starting from the system architectural design down to the software architectural design. Although, in the majority of cases the design should represent a mechatronics-based system, the hardware and the mechanics view are not considered. The aim of this work is to remedy the deficiencies regarding the missing representation of hardware and mechanics artefacts within E/E System design. Therefore, a model-based domain-specific language was developed that describes the system in a more comprehensive way. It makes it easier for domain experts, who are not that familiar with UML or SysML, to create an architectural design. The methodology presented does not ignore existing SysML models, but rather supports them by means of a translator, which converts the DSL model into a SysML representation. As well as the domain-specific language definition itself, a feasible tool support is presented. To showcase that the language definition can be implemented easily in different ways, a custom-made tool written in C# as well as a tool generated from a UML definition is shown.

CCS Concepts

•Computer systems organization → Embedded systems; •Software and its engineering → *Software design engineering;*

Keywords

automotive embedded systems, system architectural design, domain-specific modelling, E/E Systems

1. INTRODUCTION

The number of functionalities realized through E/E Systems in modern cars, and therefore the overall complexity, will keep increasing over the next few years. The connection of cars with their environment and new propulsion technologies will foster this trend. There is enormous potential for product differentiation between competing companies and for optimizing existing E/E System functionalities.

High quality standards along the whole product life cycle are crucial for coping with the upcoming challenges. To achieve this, methods and techniques from concepts such as Automotive SPICE [2] are strongly recommended. Some of the key aspects of these concepts are bilateral traceability, as well as consistency between the different design abstraction levels, starting from a system design down to a detailed software component design.

In the automotive sector, embedded system design models are usually created with techniques based on the *Unified Modelling Language (UML)*. Either the meta-model is extended, or a profile is created to make it possible to use the UML-based approach for the embedded automotive system design. A wide-spread example of an UML2 profile is SysML, which reuses many of the original diagram types (*State Machine Diagram*, *Use Case Diagram*, etc.), uses modified diagram types (*Activity Diagram*, *Block Definition Diagram*, etc.), and adds new ones (*Requirement Diagram*, *Parametric Diagram*) [6].

Even if the UML-based methodologies are valuable for projects with an emphasis on software, they are sometimes too powerful for embedded automotive system design due to the numerous representation options. In particular for domain experts who have no or limited knowledge about software development, the high number of elements available for modelling, turns system architectural design into an awkward task. However, it is not the intention of this work to criticize the SysML approaches created so far. They are a good choice for a multitude of tasks.

Instead, this paper showcases an extension to these SysML approaches, to make the architectural design of embedded mechatronics systems easier for UML non-natives. Therefore, a model-based domain-specific language and domain-specific modeling have been developed to meet the specific needs of embedded automotive mechatronics systems. Therefore, a model-based domain-specific language, respectively a domain-specific modelling (DSM), has been developed to meet the specific needs of embedded automotive mechatronics systems. Additionally, this contribution demonstrates that a tool support of the defined language can be established in different ways. In Section 3 and Section 4 two variants for a EMS-DSM tool support are described. The first one is a self developed application which has been written in C#. To speed up the application development process, an open source diagram editor project has been used as a basis for the self-written parts of the implementation. The second presented tool utilizes the *Modeling SDK for Visual Studio - Domain-Specific Languages* [14], which contains development kits for domain-specific languages and architecture tools. With the aid of this SDK, the DSL definition is implemented by modelling the language specification in UML notation. In both cases, the traceability mentioned earlier is assured by linking development artefacts such as requirements (e.g. technical system requirements, software requirements, etc.), and verification criteria to the automotive E/E System architectural design model.

In the course of this document, Section 2 presents an overview of the related approaches, as well as of domain-specific modelling and integrated tool chains. In Section 3, a description of the proposed DSM approach for the model-based system engineering is provided. An application of the described methodology is presented in Section 4. Section 5 concludes the work with an overview of the work presented.

2. RELATED WORK

In recent years, a lot of effort has been made to improve the automotive model-based E/E System design methods and techniques. Traceability and consistency between the development artefacts has always been an important topic. However, due to an increase in electronic and electric-based functionality, these properties have become vital.

According to international standard ISO 26262, which was released in 2011, traceability between the relevant artefacts is mandatory for safety-critical functionalities [9]. A description of the common deliverables along an automotive E/E System development, and a corresponding process reference model is presented by the de-facto standard Automotive SPICE [2]. Neither the functional safety standard nor the process reference model enforces a specific methodology of how the development artefacts have to be linked to each other. However, connecting the various work products manually is a tedious and error-prone task.

In [11] the authors describe a seamless model-based tool chain orchestration for the automotive system and software engineering domain. As in other contributions in this field ([3], [1], [7], [10], [13]), SysML is utilized for the system architectural design.

To agree with Broy et al. [4], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities for customizing the UML diagrams to get a language for embedded system design, drive these drawbacks. On the one hand, the meta model can be extended, and on the other hand, a profile can be defined [13]. Even if there is an agreement to utilize a common UML profile such as SysML, plenty of design artefact variations are feasible.

Another long term research project in the field of system design is the *Ptolemy Project* [16]. Together with the related open source and simulation tool *Ptolemy II*, the project provides an environment for the modelling of heterogeneous cyber-physical systems. With the four integrated syntax classes *block diagrams*, *bubble-and-arc diagrams*, *imperative programs*, and *arithmetic expressions*, various design domains can be addressed. It is general purpose, something which is both its major strength and its weakness, as is the case with UML approaches. Moreover, the bidirectional traceability to requirements and other development artefacts is missing.

The scenario described in this section so far, does not provide an optimal base for engineers who have to design the embedded automotive system from a mechatronics point of view. Ideally, the tool should be intuitive and easy to operate, even without specific UML knowledge. These findings led the authors to the idea to create a more tailored model-based language for the stated domain.

Mernik et al. [12] describe a domain-specific language as a language that is tailored to the specific application domain. Enhanced by this tailoring, there should be substantial gains in expressiveness and ease of use, compared to general-purpose languages. Even if SysML-based modelling techniques do increase expressiveness, there is no improvement in ease of use for embedded automotive mechatronics system design.

Preschern et al. [15] claim that DSLs help to decrease system development costs by providing developers with an effective way to construct systems for a specific domain. The benefit in terms of a more effective development has to be higher than the investment for creating or establishing a DSL at a company or department. In addition, the authors argue that the DSL development cost mentioned will decrease significantly over the next few years due to new tools supporting language creation such as Eclipse-based *Sirius*[1].

Vujović et al. [21] present a model-driven engineering approach to creating a domain-specific modelling (DSM). Sirius is the framework for developing a new DSM and the DSM graphical modelling workbench. The big advantage of this tool is that the workbench for the DSM is developed graphically. Therefore, knowledge about software development with Java, the graphical editor framework (GEF) or the graphical modelling framework (GMF) is not needed.

According to Hudak [8], programs written in a DSL are more concise, can be written more quickly, are easier to maintain

[1]https://eclipse.org/sirius/

and reason about. In the author's opinion, this list of advantages is also valid for domain-specific modelling. Furthermore, Hudak determines the basic steps for developing a domain-specific language as *(a) Definition of the domain, (b) Design of the DSL capturing the domain semantics, (c) Provide support through software tools,* and *(d) Create use-cases for the new DSL infrastructure.* The approach described in this paper is highlighted, according to these steps, in Section 3 and 4.

3. APPROACH

The main goal of this contribution is to provide a lean approach for engineers to facilitate embedded automotive mechatronics system modelling on a high abstraction level. The focus of the described approach is on the model-based structural description of the E/E System under development. Additionally, the signals and interfaces are essential parts of the modelling. The described methods and techniques in Section 2 form a well-defined basis for this work.

The existing SysML-based design method is extended by the newly developed *Embedded Mechatronics System Domain-Specific Modelling (EMS-DSM)* for the automotive embedded system design. It is not intended to replace the SysML-based solution created so far. Instead, the EMS-DSM is integrated into the existing approach, and the whole tool-chain, starting with the SysML-based system architectural design tool, down to the software / hardware architectural design, can be utilized if desired. An overview of the tool integration is shown in Figure 1.

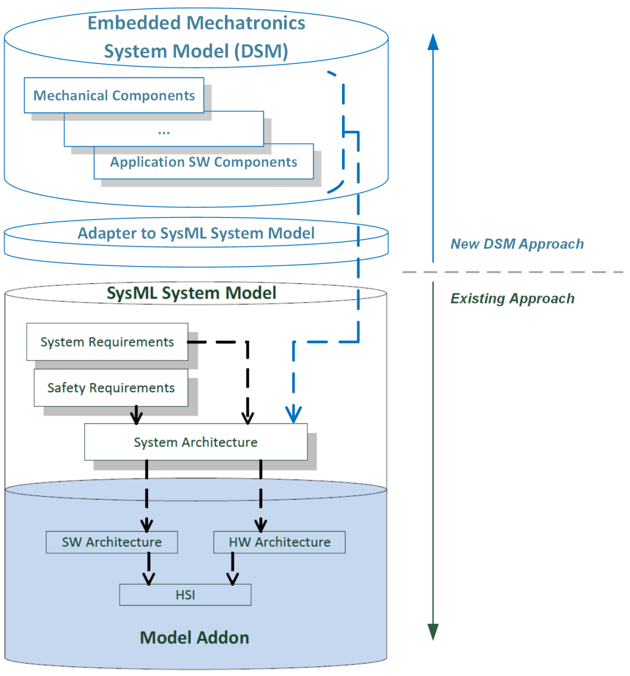


Figure 1: Tool-Chain Integration of DSM and Existing SysML Model Approach (based on [11])

For topics such as project management and requirements management, the web-based open source application *Redmine*[2] is used. Owing to its high flexibility through configuration, new trackers, which also reflect the engineering process, are added:

- Functional System Requirement
- Technical System Requirement
- Hardware Requirement
- Software Requirement
- System Test Case
- Hardware Test Case
- Software Test Case

The test case and requirement items are connected to each other by their unique identifier. The relationship between them is shown in Figure 2. For a safety-critical development according to ISO 26262, additional types of issues such as *Functional Safety Requirements* are added.

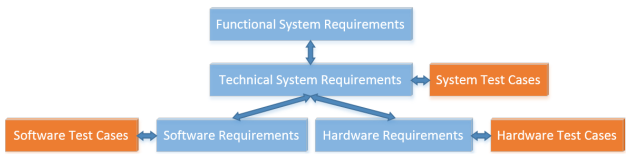


Figure 2: Requirements Hierarchy and Test Case Relationships

3.1 Definition of the Domain

This work focusses on the *Embedded Mechatronics E/E System Design* in the automotive field. This can be seen as the meta-domain of the model-based language. The EMS-DSM itself is tailored to the needs of the domain at the particular project or company. E.g. the domain of the presented application in Section 4 is *Embedded Mechatronics E/E System Design for Compressed Natural Gas (CNG) Fuel Tank Systems.*

3.2 Design of the EMS-DSM

The definition of the newly developed model-based domain-specific language is divided into four levels as shown in Figure 3. These levels were only introduced to categorize the deviations. They do not depict any order of the component instances in the design model.

EMS-DSM Level 1

The *EMS-DSM Component* at level 1 is the origin of all other classes in the language definition. The five properties of this class are

- *ID:* unique identifier of the particular instance in the design model, set automatically

[2]http://www.redmine.org/

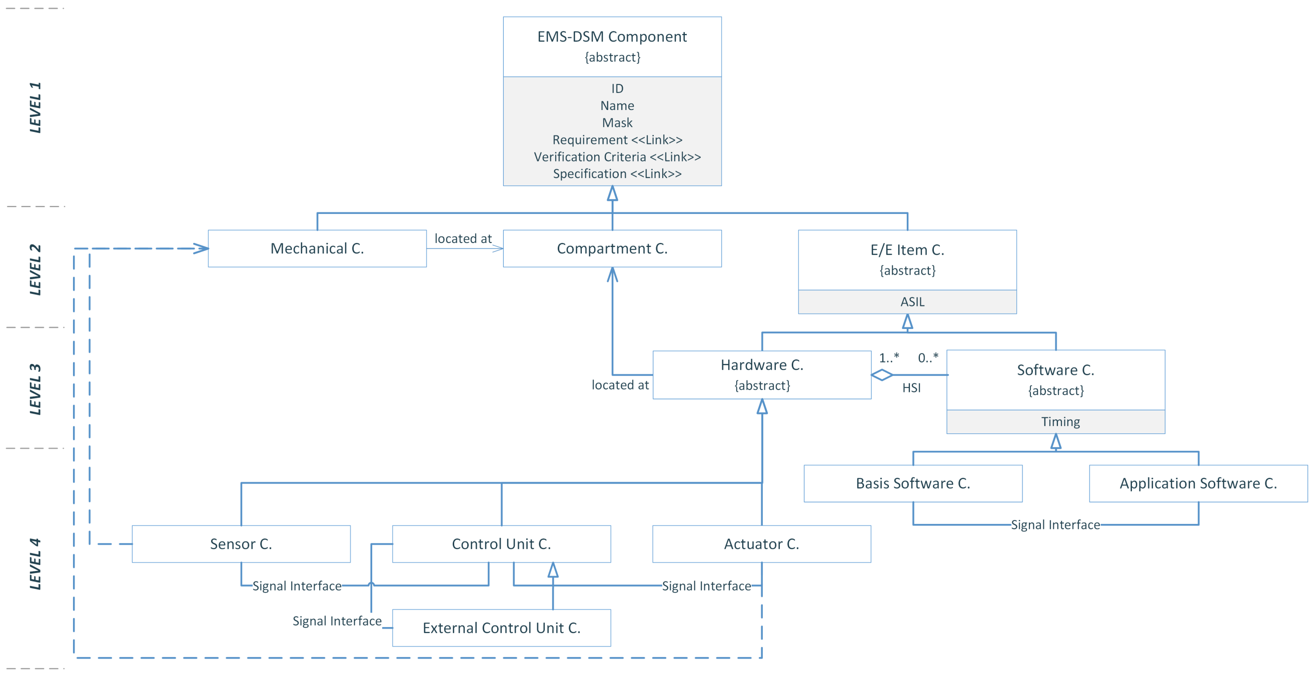


Figure 3: *EMS-DSM* Definition (UML)

- *Name:* name or short description of the particular instance, chosen by the design engineer
- *Requirement:* in this approach, a link to the Redmine requirements database, set by the designer
- *Verification Criteria:* similar to the Requirement, a link to the Redmine verification criteria artefact, set by the designer
- *Specification:* link to further information about the actual component, e.g. a CAD drawing or a data sheet

The abstract *EMS-DSM Component* serves as the base node of the EMS-DSM definition, and declares the common properties of the derived classes at the lower levels, meaning that this component is not instanced.

EMS-DSM Level 2

At the second level, the following component classes are available:

- *Mechanical Components:* used by all mechanical, domain-specific components, e.g. the *Mechanical Pressure Regulator* class at the use case presented in Section 4
- *Compartment Components:* gives the opportunity to specify areas or compartments, where mechanical and hardware components are installed
- *E/E Item Components:* an abstract component class definition, which serves as a basis for the hardware and software components at the lower levels. Additionally, the property *ASIL*, corresponding to the ISO 26262, is declared

EMS-DSM Level 3

At this level, the abstract classes *Hardware Component* and *Software Component* are defined as a basis for the derived classes at level 4. The property *Timing* is added at the software component class, which defines the scheduling of the software components at the subsequent level.

EMS-DSM Level 4

The majority of the non-abstract component classes are defined at this level. Classes derived from the hardware component are:

- *Sensor Component:* used for all domain-specific sensor components
- *Control Unit Component:* used for all domain-specific control unit components
- *Actuator Component:* used for all domain-specific actuator components
- *External Control Unit Component:* special class, to make signals from an external component available at the considered system without modelling the complete control unit in the actual design

With the exception of the External Control Unit Component, all hardware components, and their instances in the system design model, are capable of creating a software design model. Any kind of software component instance is only allowed to be implemented at a software design model which belongs to an instance of a hardware component. Regarding smart sensors / actuators, not only the *Control Unit Component*, but also the *Sensor* and *Actuator Components* were enabled for this procedure.

To conclude the EMS-DSM level description, the classes derived from the software component are:

- *Basis Software Component:* used for all low-level, hardware-dependent software components
- *Application Software Component:* used for all functional software components

3.3 EMS-DSM Features

In this paper, an overview of the novel domain-specific modelling definition is presented. The features, as well as the benefits compared to other approaches in this field, are described in detail in further publications. Nevertheless, a short overview of the main features shall be given:

- *Integrated Requirements Management Capability* - Core Functionality Requirements and Ancillary Functionality Requirements are specified and stored in a MySQL database. By utilizing the ADO.net driver for MySQL, one or more requirements can be linked to each component in the architectural design [19].
- *Incorporation of System and Software Development Environments* - The system and the software architectural design can be created in the environment provided. Supported by both an export and an import functionality, software architectural design in the EMS-DSM model can be transferred to a Simulink framework model, and can also be created from a Simulink model [18].
- *Model-to-Model Transformation at the System Design Level* - As mentioned in the previous section, the EMS-DSM is not a replacement for the established SysML approach. Instead, it can be seen as an extension and the models can be bidirectionally transformed into the different representations [20].
- *Open Modelling Toolbox Access* - Utilizing a library editor, the modelling toolbox can be easily adapted to the needs of the respective company or project [17].

3.4 EMS-DSM Tool Support

At the time when the research project was initiated, the aims were the highest possible flexibility and full access to the tool's source code. Therefore, the decision was taken to write a custom-made tool which implements the modelling language definition. Nevertheless, in different publications the authors mentioned that the EMS-DSM language definition can also be implemented using other technologies. To deliver a proof of concept of this statement, the modelling language definition has been integrated into the Visual Studio environment, aided by Microsoft's *Modeling SDK*. In the following subsections, as well as in Section 4, the initial custom-made tool support as well as the approach based on Visual Studio is described.

3.4.1 Custom-made Tool

To avoid having to develop an application from scratch, the open source project *WPF Diagram Designer*[5] has been chosen as the basis of the start of tool development. The corresponding documentation has more than 500.000 views and the source code has been downloaded more than 25.000 times. It is therefore a relatively well reviewed source, which provides standard functionality such as file handling and basic graphical modelling. The source code is written in C# and provides good expandability. To facilitate EMS-DSM engineering, new functionalities have been implemented in the diagram designer. To express the initially strong relation to the automotive industry, the newly developed tool was named *EASy-Design (**E**mbedded **A**utomotive **Sy**stem-Design).*

The advantages of the custom-made tool implementation for the domain-specific modelling language are (a) (a) that it provides a stand-alone application without any need for purchasing an IDE (*Integrated Development Environment*) which serves as a runtime environment, (b)that it is easier to use due to a reduction of the number of user interface elements compared to off-the-shelf software approaches, and (c) that it provides full access to all parts of the source code (e.g. implementation of an architectural design multi-level view is much easier compared to other approaches). The disadvantages are (a) the huge tool development efforts, and (b) the missing (external) tool maintenance.

3.4.2 Modeling SDK-based Tool

Model-based development tools, which are integrated into Visual Studio, can be created aided by the *Modeling SDK for Visual Studio*. The software development kit (SDK) is provided by the *Microsoft Developer Network* [14], and is available for different versions of Visual Studio, whereby the domain-specific modelling presented here is based on Version 2013, which was published in November 2013.

The language definition illustrated in Figure 3 has been implemented in the *DSL Definition* environment at Visual Studio. All defined components (*EMS-DSM Component, Mechanical Component, etc.*) are represented by *Domain-Class* elements. The component properties have been added as *Domain Properties* and are inherited downwards starting from the abstract top-level class *EMS-DSM Component*. The modelling of *Level 1* and *Level 2* (see Subsection 3.2) of the defined language is shown in Figure 4. It has to be stated that the illustrated DSL definition belongs to the EMS-DSM language as presented in Figure 3. For a more specific domain such as *Embedded Mechatronics E/E System Design for Compressed Natural Gas (CNG) Fuel Tank Systems*, which is presented as a use case in Section 3, additional classes are inherited (e.g. *Tank Cylinder* class from *Mechanical Components* class) and provided at the designer's toolbox.

Like the custom-made variant, the Visual Studio based tool can be used to model embedded mechatronic system architectures and the medium (e.g. gas) as well as the signal flow between the modelled elements. The full traceability between development artefacts such as *Technical System Requirements* and the *Architectural System Design* can also be

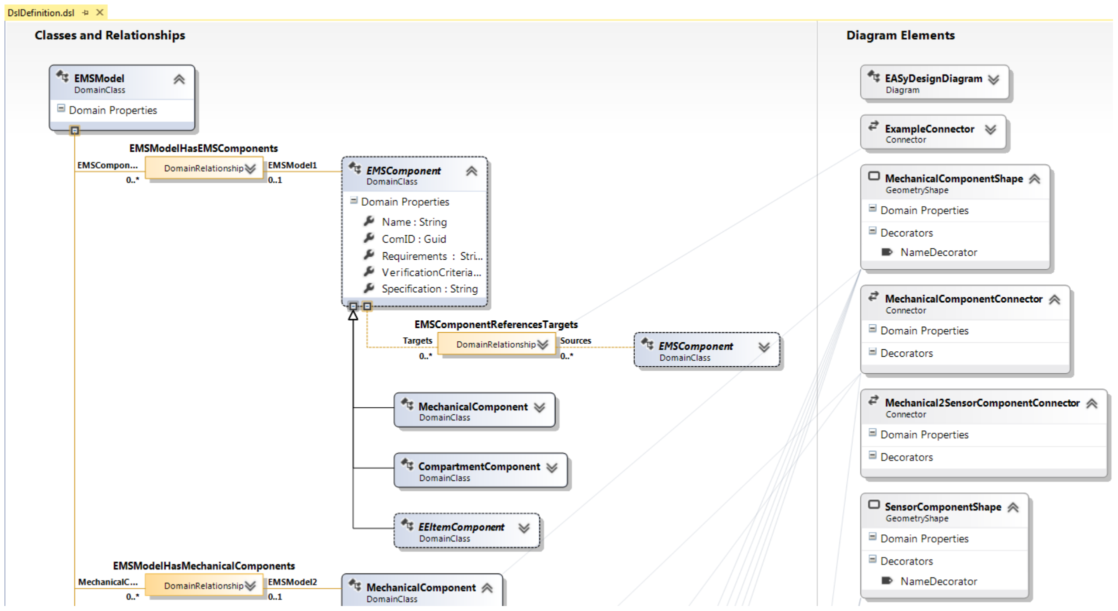


Figure 4: Excerpt of the *EMS-DSM* Definition Transferred to the Visual Studio Environment

established by utilizing the component properties. For an easy distinction between the custom-made tool and the Visual Studio based tool, the latter was named *EASy-Design VS*.

The advantages of implementing the EMS-DSM language definition in the Visual Studio environment are (a) fast implementation aided by a specialized domain-specific language SDK, (b) built-in code generator, which can be simply tailored for customer needs, and (c) instant validation of the DSL definition. The disadvantages are (a) the need to purchase the development software (e.g. Visual Studio), which serves as a runtime environment for each user who is to design the embedded system or parts of it, and (b) limited tailoring of the domain-specific modelling environment (with regard to the *DSL Definition* modelling).

3.4.3 Other Technologies

The tool support possibilities described in the previous subsections are exemplary and not restrictive. The methodology and its C# implementation can also be ported to e.g. Enterprise Architect[3] aided by the provided *Add-in* mechanism. Another alternative is the Eclipse[4] project *Sirius* mentioned in Section 1, which enables the creation of a graphical modelling workbench, by facilitating the Eclipse modelling technologies without writing any line of code. Similar to the Visual Studio-based approach, the domain-specific modelling environment is created by means of graphical modelling. An advantage of Sirius is the free Eclipse IDE, which serves as the domain-specific modelling tool development platform, as well as the runtime environment for the DSM tool itself.

[3]http://www.sparxsystems.com/
[4]http://eclipse.org/

4. APPLICATION

In Section 3, the first three steps towards developing a DSL / DSM, as defined by Hudak, are shown. Below, the last step *Create use cases for the new DSL infrastructure* is described. In the following part of this section, the use case scenario is modelled by means of two different tools. These are the custom-made variant *EASy-Design* and its Microsoft Visual Studio based counterpart EASy-Design VS, both mentioned in Section 3.

4.1 General Use Case Description

For an appropriate scale of the use case, only a small part of the real-world system is utilized. The application is to be recognized as an illustrative material, reduced for internal training purposes for students. Therefore, the disclosed and commercially non-sensitive use case is not intended to be exhaustive or to represent leading-edge technology.

As previously mentioned, both tools - *EASy-Design* and *EASy-Design VS* - have been utilized for an application of the use case. It shall be shown that, apart from the particular pros and cons of each of the approaches, the architectural system design can be created with both. The design of a fuel tank system for compressed natural gas (CNG) was selected as an appropriated automotive use case. In the remaining part of this subsection the tool-independent system properties are described.

The CNG fuel tank system consists of seven mechanical components:

- Tank Cylinder
- Filter
- Mechanical Pressure Regulator

- Gaseous Injector Rail
- 3 x Tubing

At both system designers, the mechanical components are coloured blue. The medium flow between the components, which is CNG in this use case, is displayed using blue lines with an arrow on one end (on the sink side).

Furthermore, five hardware components shall be placed on the *System Design Model* level:

- In-Tank Temperature (Sensor Component)
- CNG High Pressure (Sensor Component)
- On-Tank Valve (Actuator Component)
- Tank ECU (Control Unit Component)
- Engine ECU (External Control Unit Component)

The signal flow between the components is shown using yellow lines, ending with an arrow on the signal sink. A communication bus is inserted between the Control Unit and the External Control Unit component.

In Figure 3, dependencies are defined between *Mechanical Components* and *Sensor Components*, and *Hardware Components*. These relationships enable the direct connection between e.g. the *On-Tank Valve Actuator* (as a hardware component) and the *Tank Cylinder* (as a mechanical component).

4.2 EASy-Design

In Figure 5, the EMS-DSM supporting tool *EASy-Design* is illustrated. The system architectural design of the CNG fuel tank use case with the mechanical and hardware elements listed in Subsection 4.1 is embedded in the *System Design Model* area.

The five hardware components are coloured yellow. A communication bus is inserted between the Control Unit and the External Control Unit component, and is shown by the double compound line type and arrows on both ends.

The *Software Components* cannot be placed on the System Design Level. With a double-click on a Hardware Component, the next modelling level is opened (named *E/E Item Design Level*). Here, the green coloured *Basis Software Components* and *Application Software Components* are put in place.

By double-clicking a connection between any two components, a dialogue is opened and the signal, or signals in the case of a communication bus, can be specified. The properties, such as the name or the link to the corresponding requirements, are easily set by selecting the particular component with a single click and entering the data in the *Element Properties* toolbox.

4.3 EASy-Design VS

In Figure 6, the EMS-DSM tool *EASy-Design VS* is illustrated. The system architectural design of the CNG fuel tank use case with the mechanical and hardware elements listed in Subsection 4.1 is embedded in the *System Design Model* area.

The colour selection for the model components is similar to the one described in the previous subsection. Only the colour gradation of the components of the class *Hardware Component* varies from yellow for the sensor components, to dark yellow for the control unit components, to orange for the actuator components. Another difference is the presentation of the signal bus between the control unit and the external control unit. At *EASy-Design VS* a signal bus is visualised by a dash-dot-dot line instead of the double compound line.

All the components (e.g. *Mech - Tank Cylinder*), their relationship and the respective signal or medium connections (e.g. *Medium Link*), as available at the toolbox window left of the system design area in Figure 6, have been implemented just by modelling the graphically represented *DSL Definition* in Visual Studio. Theoretically, the software components could be modelled on the same diagram, e.g. as nested items within the particular hardware component, but with a growing number of system architectural design elements, the model would quickly get confusing. Hence, the source code of the multi-level handling from the fully custom-made tool *EASy-Design* has been transferred to *EASy-Design VS*. Thus a new diagram level can be opened by double-clicking one of the hardware components at the system architectural model, where items of class *Basis Software* and *Application Software* can be utilized to create a software architectural design.

Similar to the multiple diagram level feature, the C# code for the signal specification procedure has also been transferred to this approach. Thus by double-clicking a connection between two components a dialogue is opened and the signal can be specified.

The properties of each element at the system or software architectural design can be entered by selecting the desired item in the diagram and filling the lines at the property window, shown on the lower left section in Figure 6.

5. CONCLUSIONS

In the previous sections, the current embedded automotive system development challenges have been described. To overcome the problem of often incomplete or incoherent development artefacts, which represents one of the main challenges in this research field, an approach has been shown for a seamless development from engineering and managing the various types of requirements, to creating the system and software architectural design, to generating model-based software frameworks for further implementations.

The key concepts of the presented methodology foster the linking of each development artefact with its predecessor and successor wherever applicable, and provide an architectural design strategy by providing a domain-specific modelling language definition that can be implemented into various development environments. In Sections 3 and 4, two of these possibilities have been shown.

Generally speaking, the overall effort in terms of money and manpower needed to create an appropriate domain-specific

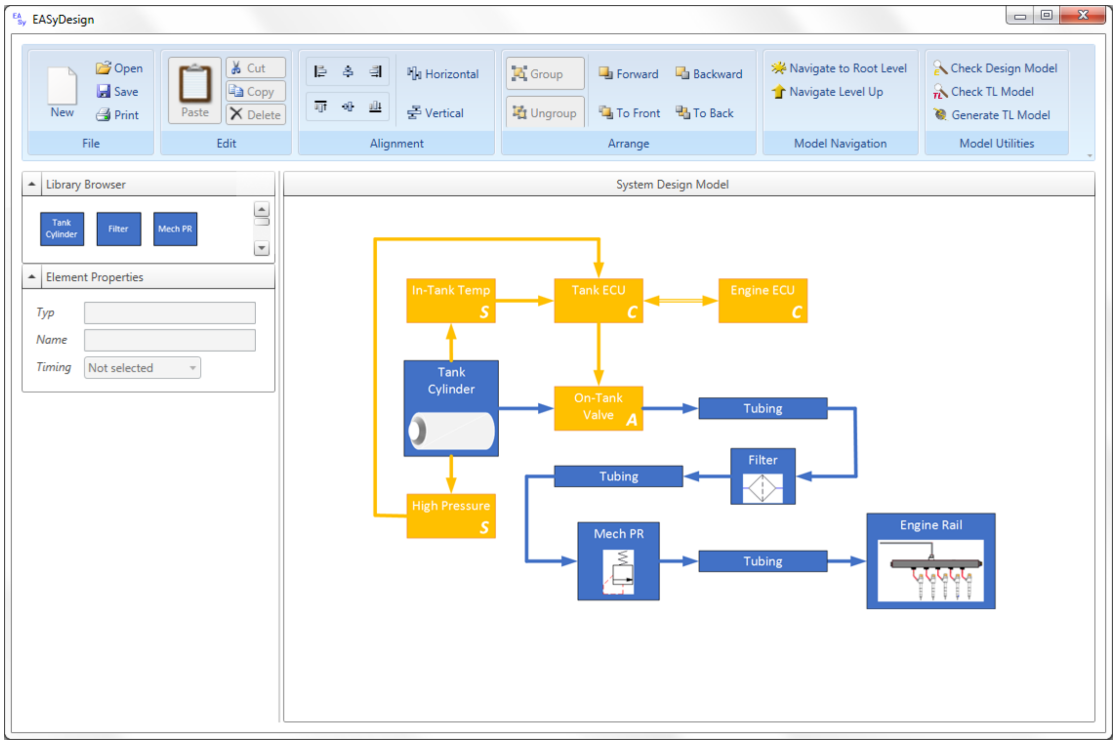
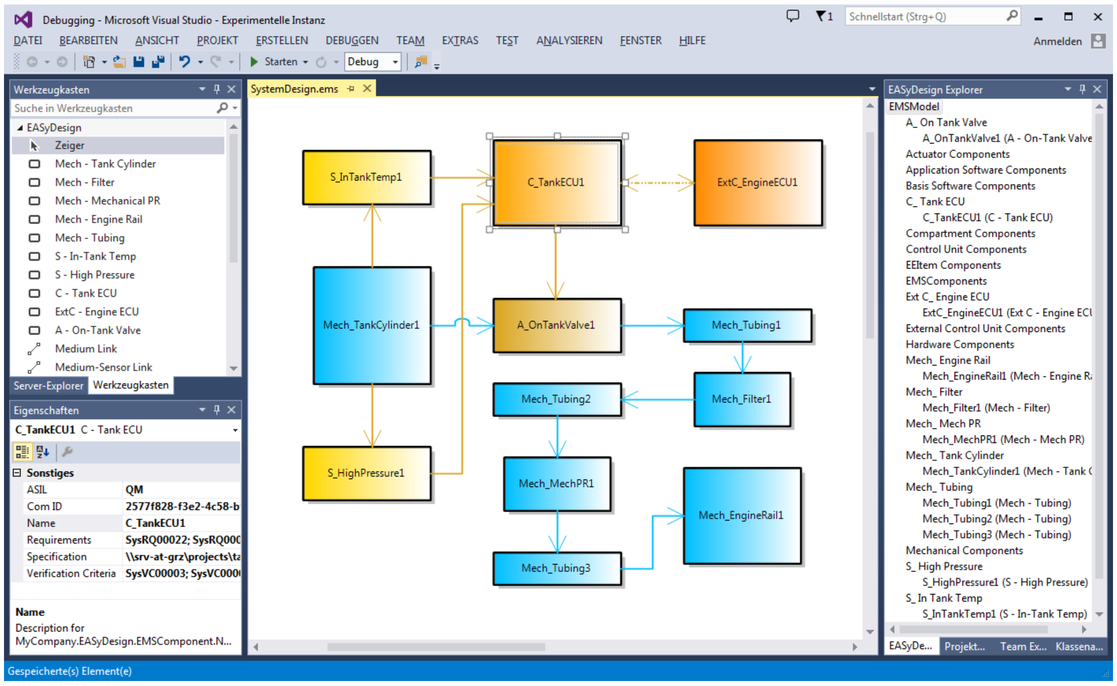


Figure 5: System Model Designed in *EASy-Design*



Eigenschaften

C_TankECU1 C - Tank ECU

Sonstiges

ASIL	QM
Com ID	2577f828-f3e2-4c58-b
Name	C_TankECU1
Requirements	SysRQ00022; SysRQ000
Specification	\\srv-at-grz\projects\ta
Verification Criteria	SysVC00003; SysVC000

Name
Description for
MyCompany.EASyDesign.EMSComponent.N...

Figure 6: System Model Designed in *EASy-Design VS*

development environment for the system architectural design, which also provides full traceability to other development artefacts, is more or less the same for both presented approaches, independent of the decision whether to create a full self-developed tool or to build on a available technology like *Modeling SDK for Visual Studio - Domain-Specific Languages*. None of the approaches based on the considered technologies - this also includes the previously mentioned Eclipse project *Sirius* - work without customization by means of writing code. For instance, the multi-level design feature as it is provided by EASy-Design and described in Subsection 4.2, cannot be established in a feasible way by merely modelling the domain-specific tool. Which approach is best strongly depends on the structure of the E/E System development team or company and cannot be generally stated.

Although the EMS-DSM definition and its implementation *EASy-Design*, is currently in a first trial phase for industrial project applicability, there is already evidence of the benefits of the approach. Moreover, important topics for future work, such as the automatic validation of the system and software architectural designs, and the transformation of a system of systems model in one step, have been identified.

6. REFERENCES

[1] E. Andrianarison and J.-D. Piques. SysML for embedded automotive Systems: a practical approach. In *Conference on Embedded Real Time Software and Systems*. IEEE, 2010.

[2] Automotive SIG. Automotive SPICE®Process Assessment Model. Technical report, The SPICE User Group, May 2010. Version 2.5.

[3] R. Boldt. Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group, July 2009.

[4] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526–545, 2010.

[5] Code Project. WPF Diagram Designer - Part 4. Online Resource. retrieved March 23, 2015, from http://www.codeproject.com/Articles/24681/WPF-Diagram-Designer-Part.

[6] S. Friedenthal, A. Moore, and R. Steiner. OMG Systems Modeling Language (OMG SysML^{TM}) Tutorial. In *INCOSE International Symposium*, 2006.

[7] H. Giese, S. Hildebrandt, and S. Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages 555 –579, 2010.

[8] P. Hudak. Domain-specific languages. *Handbook of Programming Languages*, 3:39–60, 1997.

[9] ISO 26262, Road vehicles - Functional safety. International standard, International Organization for Standardization, Geneva, CH, November 2011.

[10] R. Kawahara, H. Nakamura, D. Dotan, A. Kirshin, T. Sakairi, S. Hirose, K. Ono, and H. Ishikawa. Verification of embedded system's specification using collaborative simulation of SysML and simulink models. In *MBSE'09*, pages 21–28. IEEE, 2009.

[11] G. Macher, E. Armengaud, and C. Kreiner. Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In *7th European Congress Embedded Real Time Software and Systems Proceedings*, pages 256–263, 2014.

[12] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[13] J. Meyer. *Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards*. PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Germany, July 2014.

[14] Microsoft Developer Network. Modeling SDK for Visual Studio - Domain-Specific Languages. Online Resource. retrieved January 19, 2016, from https://msdn.microsoft.com/en-us/library/bb126259.aspx.

[15] C. Preschern, N. Kajtazovic, and C. Kreiner. Efficient development and reuse of domain-specific languages for automation systems. *International Journal of Metadata, Semantics and Ontologies*, 9(3):215–226, 2014.

[16] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[17] H. Sporer. A Lean Automotive E/E-System Design Approach with Open Toolbox Access. In *Systems, Software and Services Process Improvement - 22nd European Conference, EuroSPI 2015*, EuroAsiaSPI '15, Ankara, Turkey, pages 41–50, 2015.

[18] H. Sporer, G. Macher, A. Höller, and C. Kreiner. Bidirectional Crosslinking of System and Software Modeling in the Automotive Domain. In *7th International Workshop on Software Engineering for Resilient Systems*, SERENE '15, Paris, France, 2015.

[19] H. Sporer, G. Macher, C. Kreiner, and E. Brenner. A Lean Automotive E/E-System Design Approach with Integrated Requirements Management Capability. In *9th European Conference on Software Architecture (ECSA 2015)*, ECSA '15, Dubrovnik/Cavtat, Croatia, 2015.

[20] H. Sporer, G. Macher, C. Kreiner, and E. Brenner. A Model-to-Model Transformation Approach at Mechatronics-Based E/E-System Design. In *41st EUROMICRO Conference on Software Engineering and Advanced Applications, Session on "Work in Progress"*, SEAA '15, Funchal, Madeira, Portugal. EUROMICRO, 2015.

[21] V. Vujović, M. Maksimović, and B. Perišić. Sirius: A rapid development of DSM graphical editor. In *18th International Conference on Intelligent Engineering Systems (INES)*, pages 233–238. IEEE, 2014.

A Lean Automotive E/E-System Design Approach with Open Toolbox Access

Harald Sporer( )

Institute of Technical Informatics,
Graz University of Technology, Inffeldgasse 16/1, 8010 Graz, Austria
sporer@tugraz.at
http://www.iti.tugraz.at/

Abstract. Replacing former pure mechanical functionalities by mechatronics-based solutions, introducing new propulsion technologies, and connecting cars to their environment are only a few reasons for the still growing electrical and electronic systems (E/E-Systems) complexity at modern passenger cars. Smart methodologies and processes are necessary during the development life cycle to master the related challenges successfully. One of the key issues is to have an adequate environment for creating architectural system designs, and linking them to other development artifacts. In this paper, a novel model-based domain-specific language for embedded mechatronics-based systems, with focus on the support of different automotive sub-domains, is presented. With the described methodology, the domain-specific modeling (DSM) approach can be adapted to the needs of the respective company or project easily. Though, the model-based language definition can be implemented using various platforms (e.g. Eclipse Modeling Framework), also a custom-made open source editor supporting the DSM technique, is presented.

Keywords: System architectural design · Domain-specific modeling · Automotive embedded systems · E/E-Systems

1 Introduction

The electrical and/or electronic systems (E/E-Systems) in the automotive domain have been getting more and more complex over the past decades. New functionality, mainly realized through embedded E/E-Systems, as well as the growing connectivity (*Car2X-Communication*), will keep this trend alive in the upcoming years. Well-defined development processes are crucial to manage this complexity and to achieve high quality products. Wide-spread standards and regulations, like *Automotive SPICE®* and *ISO 26262*, give a guidance through the development life cycle.

Best practice for the E/E-System development process is still to refer to some kind of the V-Model. Starting with an initialization and analyzing phase, via the subdivided system elements, down to the implementation, and back up by integration and test phases towards the completed system, a multitude of

R.V. O'Connor et al. (Eds.): EuroSPI 2015, CCIS 543, pp. 41–50, 2015.
DOI: 10.1007/978-3-319-24647-5_4

work products arises and have to be managed properly. Trying to keep all of the artifacts consistent manually, is an error-prone and tedious task. Therefore, a lot of effort has been made through the last years to increase the quality by an adequate and highly automated tool support.

To create the system design, most of the existing approaches utilize some kind of UML profile (e.g. SysML[1]). Though these techniques have a lot of advantages, in some scenarios they are not best choice. On the one hand, the possibility to include mechanical parts or the flow of fluids and forces is missing, and on the other hand, a possible lack of UML skills, especially in small project teams, which wants to carry out a lean development, makes the UML-based design to an awkward task.

The main goal of this work is to contribute to the improvement of the existing system architectural design methods. The herein presented approach has been created for the development of embedded mechatronics-based E/E-Systems in the automotive field mainly. However, the techniques are also suitable for other domains. The mentioned improvement is accomplished by extending the wide-spread and common UML-based methods by domain-specific modeling (DSM) techniques. It's crucial to state that the existing design techniques shall not be replaced by the presented work.

Similar to the previous mentioned de facto standard *Automotive SPICE*, full traceability and consistency between the development artifacts are also one of the main objectives of this work. Various types of requirements are linked to the system architectural design elements, and in the case of requirement changes the affected system parts can be determined easily. Moreover, supported by the DSM definition, a software architectural design can be either created within the same environment as the system design, or a established seamless tool chain can be facilitated after a domain-specific model to UML-based model transformation. In both cases, a Simulink®[2] software framework model can be generated from the software architectural design.

In this contribution the highlighted aspect of the novel DSM approach is the methodology of creating new modeling toolboxes for a particular project or company. The definition of the domain-specific language in combination with the support from the newly developed designer tool, allows a straight forward and intuitive procedure for customizing the DSM to the specific needs. A big advantage of this solution is that the customization can be conducted by the user easily and without coding.

In the course of this document, Section 2 presents an overview of the related approaches, as well as of domain-specific modeling. In Section 3, a detailed description of the proposed modeling approach with a focus on the tailoring for specific (sub-)domains is provided. An application of the described methodology is presented in Section 4. Finally, this paper is concluded with an overview of the presented work in Section 5.

[1] http://www.omgsysml.org/
[2] http://www.mathworks.com/products/simulink/

2 Related Work

In recent years, a lot of effort has been made to improve the model-based automotive E/E-System design methods and techniques. Nowadays, the advantages of a model-based approach are clear and without controversy. Meseguer [12] grants much more reliability, reusability, automatisation, and cost effectiveness to software that is developed with modeling languages. However, model transformation within or also across different languages is crucial to achieve all these benefits.

Traceability, as well as consistency, between the development artifacts has always been an important topic. However, due to the increasing number of electronic- and electric-based functionality, these properties have become vital. If it comes to safety-critical functionalities, according to the 2011 released international standard ISO 26262 [8], traceability between the relevant artifacts is mandatory. A description of the common deliverables along an automotive E/E-System development, and a corresponding process reference model is presented by the de facto standard Automotive SPICE [2]. Neither the functional safety standard nor the process reference model enforces a specific methodology, how the development artifacts have to be created or linked to each other. However, connecting the various work products manually is a tedious and error-prone task.

One of the early work products along the engineering process, is the architectural system design. In the field of automotive E/E-System development, a wide-spread and common approach is to utilize a UML-based technique for this design, like the UML2 profile SysML. Andrianarison and Piques [1], Boldt [3], and many other publications (e.g. [6], [9], [13]) present their SysML methodologies for the system design.

To agree with Broy et al. [4], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities of how to customize the UML diagrams, to get a language for embedded system design, drive these drawbacks. On the one hand, the meta model can be extended, and on the other hand, a profile can be defined [13]. Even if there is a agreement to utilize a common UML profile like SysML, a plenty of design artifact variations are feasible. This scenario doesn't provide an optimal base for the engineer who has to design the embedded automotive system from a mechatronics point of view. Ideally, the tool should be intuitive and easily operated also without specific UML knowledge. These findings led the author to the idea to create a more tailored model-based language for the stated domain. The definition and other details of this language can be found at [16].

Mernik et al. [11] describe a domain-specific language as a language that is tailored to the specific application domain. Enhanced by this tailoring, substantial gains in expressiveness and ease of use, compared to general-purpose languages, should be given. Even if a gain regarding the expressiveness is achieved by the utilization of SysML-based modeling techniques, the ease of use regarding an embedded automotive mechatronics system design is out of sight.

Preschern et al. [14] claim that DSLs help to decrease system development costs by providing developers with an effective way to construct systems for a specific domain. The benefit in terms of a more effective development has to be higher than the investment for creating or establishing a DSL at a company or department. Supplementary, the authors argue that in the next years the mentioned DSL development cost will decrease significantly, due to new tools supporting the language creation like the Eclipse-based *Sirius*[3].

Vujović et al. [17] present a model-driven engineering approach to create a domain-specific modeling (DSM). Sirius is the framework for developing a new DSM, respectively the DSM graphical modeling workbench. The big advantage of this tool is that the workbench for the DSM is developed graphically. Therefore, knowledge about software development with Java, the graphical editor framework (GEF) or the graphical modeling framework (GMF) is not needed.

According to Hudak [7], programs written in a DSL are more concise, can be written more quickly, are easier to maintain and reason about. In the authors opinion, this list of advantages is also valid for domain-specific modeling. Furthermore, Hudak determines the basic steps for developing a own domain-specific language as

- Definition of the domain
- Design of the DSL capturing the domain semantics
- Provide support through software tools
- Create use-cases for the new DSL infrastructure

The approach described in this paper is presented according to theses steps in Section 3 and 4.

3 Approach

In this section, the domain specific modeling methodology for automotive mechatronics-based system development, with a focus on the open toolbox strategy, is presented. As mentioned in Section 2, details on the definition of the domain specific modeling can be found in [16]. Therefore, just a brief description is given in the following subsection.

3.1 Domain-Specific Modeling Language

The established SysML-based design method from [10] is extended by the newly developed *Embedded Mechatronics System Domain-Specific Modeling (EMS-DSM)* for the automotive embedded system design. The main goal of this methodology is to provide a lean approach for engineers to facilitate an embedded automotive mechatronics system modeling on a high abstraction level. The focus of the approach is on the model-based structural description of the E/E-System under development. Additionally, the signals and interfaces are an essential part of the modeling.

[3] https://eclipse.org/sirius/

The definition of the newly developed model-based domain specific language is shown in Figure 1. The top node *EMS-DSM Component* is the origin of all other classes at the language definition. Therefore, each of the derived classes inherits the five properties (*ID*, *Name*, *Requirement*, *Verification Criteria*, and *Specification*) from the base class.

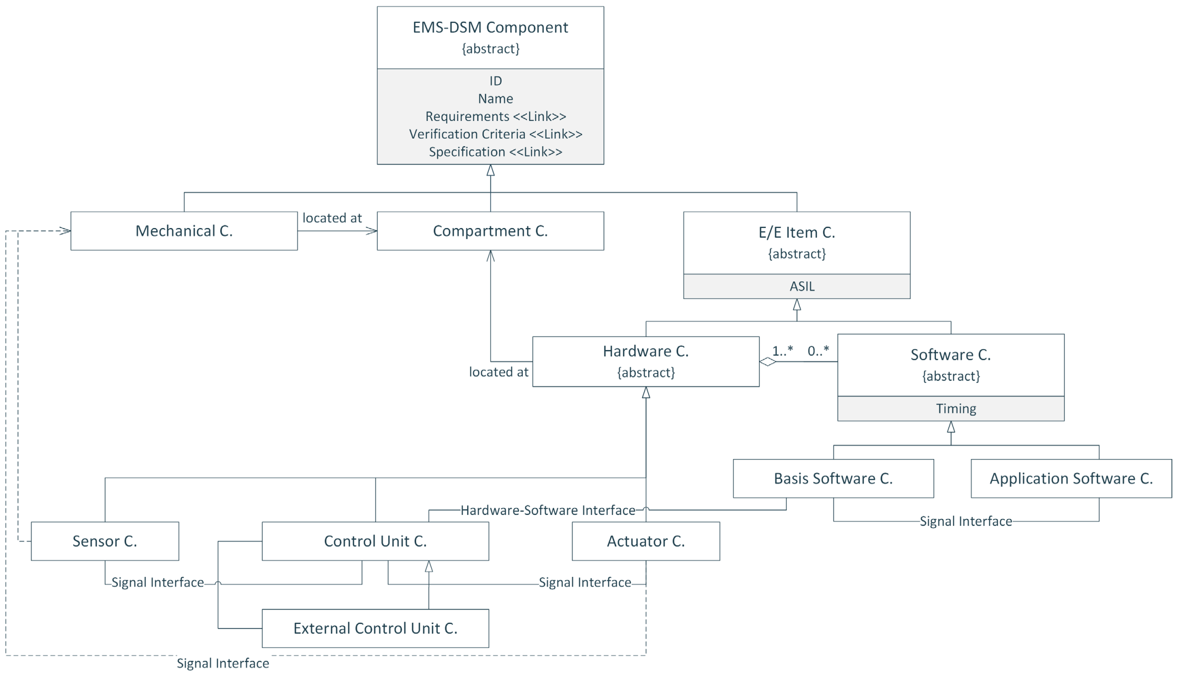


Fig. 1. *EMS-DSM* Definition (UML)

The language definition in Figure 1 represents the meta-domain of the model-based language. Subsequently, the EMS-DSM is tailored to the needs of the domain at the particular project or company. That is, design elements of possible types *Mechnical*, *Compartment*, *Sensor*, *Control Unit*, *Actuator*, *External Control Unit*, *Basis Software*, and *Application Software* are specified for the particular field of application. E.g. the domain of the presented application in Section 4 is *Embedded Mechatronics E/E-System Design for Compressed Natural Gas (CNG) Fuel Tank Systems.*

The EMS-DSM can be supported by a various number of tools, but at the time when the research project was initiated, a highest possible flexibility, as well as full access to the tools source code was desired. To achieve this, an own model editor (***E****mbedded* ***A****utomotive* ***S****ystem Design*) has been developed, based on the open source project *WPF Diagram Designer* [5].

3.2 Traceability Between the Design and Other Artifacts

To achieve a lean development environment for automotive E/E-Systems, the whole engineering life cycle has to be supported. Therefore, not only the system architectural design, but also other artifacts, like requirements and test case specification, are in the scope of this work. For topics like project management and requirements management, the web-based open source application *Redmine*[4] is used in this project. The de facto standard Automotive SPICE [2] defines three different types of requirements at the engineering process group: *Customer Requirements*, *System Requirements*, and *Software Requirements*. Out of the embedded E/E-System view, at least the hardware focus is missing. Additionally, requirements and design items regarding the mechanical components, have been introduced for the design of an embedded mechatronics-based E/E-System. Similar to the Automotive SPICE methodology on system and software level, engineering processes has been defined for these missing artifacts.

Section 3.1 contains the description of how the different types of designs (system level, software level, etc.) are created corresponding to the novel domain specific modeling. To achieve full traceability, these designs, respectively the various components at the designs, have to be linked to the corresponding requirements. This is accomplished by the *Requirements Linker* at EASy Design, which establishes a connection to the MySQL database, and therefore has full access to the requirements data at Redmine. More details about the requirements management capability of the presented project can be found at [15].

3.3 Open Toolbox Approach at the DSM

The main objective of the open toolbox strategy is to provide a possibility for the user to tailor the modeling item set to their particular needs. Every non-abstract EMS-DSM class from Figure 1 can be instantiated and utilized as type for a new toolbox item. By selecting one of the provided types, the behaviour of the new toolbox item is defined. E.g. if a new *Application Software Component* is created at the toolbox, the aggregation "*1..* 0..**" between *Hardware* and *Software Components* guarantees that the item can be used at the model within a *Hardware Component* only. These constraints contributes to the easy and intuitive handling of the modeling language. As mentioned previously, the defined modeling language and all presented features can be implemented on various modeling framework platforms, but at this project a self-made C# implementation has been preferred to achieve a highest possible grade of independence from third-party platforms.

To support the open toolbox methodology, additional functionality has been added to the custom-made software tool *EASy Design*. By selecting the command *Open Library Editor* at the menu bar, a new window (see Figure 2 in Section 4) is opened, offering toolbox modification options. At the window area *Create New Toolbox Item* the properties of the new toolbox item can be set. The drop down menu *Type* provides all non-abstract classes from the language definition. *Name* is a freely selectable identifier for the item, and *Mask* prompts the user to enter the path of a Portable Network Graphic (PNG), which determines the graphical representation of the toolbox item and its later appearance at the model. At the

[4] http://www.redmine.org/

window area *Delete Existing Toolbox Item* the no longer required item can be removed by choosing the respective name.

All library items are stored in an Extensible Markup Language (XML) file, corresponding to the following structure:

```
<EASyDesignLib>
  <LibItem>
    <Type></Type>
    <Name></Name>
    <Mask></Mask>
  </LibItem>
</EASyDesignLib>
```

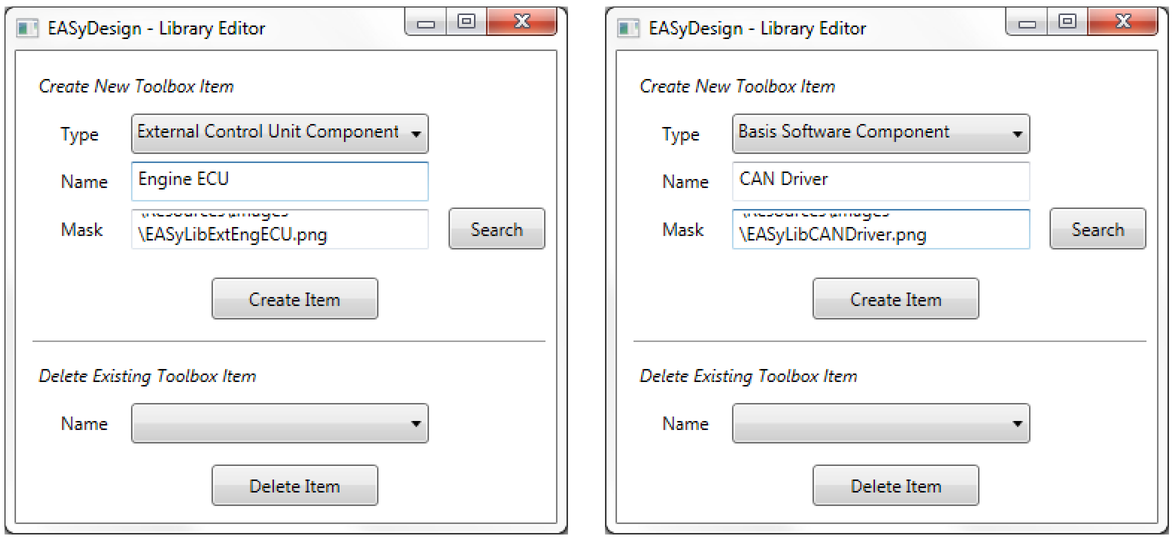
4 Application

In this section the EMS-DSM approach with an open toolbox strategy is applied to the development of an automotive fuel tank system for compressed natural gas (CNG). For an appropriate scale of the use-case, only a small part of the real-world system is utilized. The application should be recognized as an illustrative material, reduced for internal training purpose for students. Therefore, the disclosed and commercially non-sensitivity use-case is not intended to be exhaustive or representing leading-edge technology.

To model the CNG fuel tank system, several mechanical, hardware, and software components are needed. As the main mechanical components, the following items being assumed to exist in the EMS-DSM library: *Tank Cylinder*, *Mechanical Pressure Regulator*, *Filter*, *Engine Rail*, and some *Tubing*. Moreover, four hardware components have already been added to the library: *In-Tank Temperature Sensor*, *CNG High Pressure Sensor*, *On-Tank Valve* (Actuator), and *Tank ECU* (Control Unit). So far, there are no software components at the library.

For a first draft of the system architectural design, an external control unit component *Engine ECU*, and a basis software component *CAN Driver* is needed. Therefore, the steps described in Subsection 3.3 are carried out for these new library items. The corresponding *Library Editor* windows are shown in Figure 2. The new library items are added at the EASy Design *Library Browser*, and the library file *EMS-DSM-Lib.xml* is extended by the following entries:

```
<LibItem>
  <Type>External Control Unit Component</Type>
  <Name>Engine ECU</Name>
  <Mask>"..\images\EASyLibExtEngECU.png"</Mask>
</LibItem> <LibItem>
  <Type>Basis Software Component</Type>
  <Name>CAN Driver</Name>
  <Mask>"..\images\EASyLibCANDriver.png"</Mask>
</LibItem>
```


EASyDesign - Library Editor

Create New Toolbox Item

Type: External Control Unit Component

Name: Engine ECU

Mask: \Resources\Images\EASyLibExtEngECU.png

Search

Create Item

Delete Existing Toolbox Item

Name:

Delete Item

EASyDesign - Library Editor

Create New Toolbox Item

Type: Basis Software Component

Name: CAN Driver

Mask: \Resources\Images\EASyLibCANDriver.png

Search

Create Item

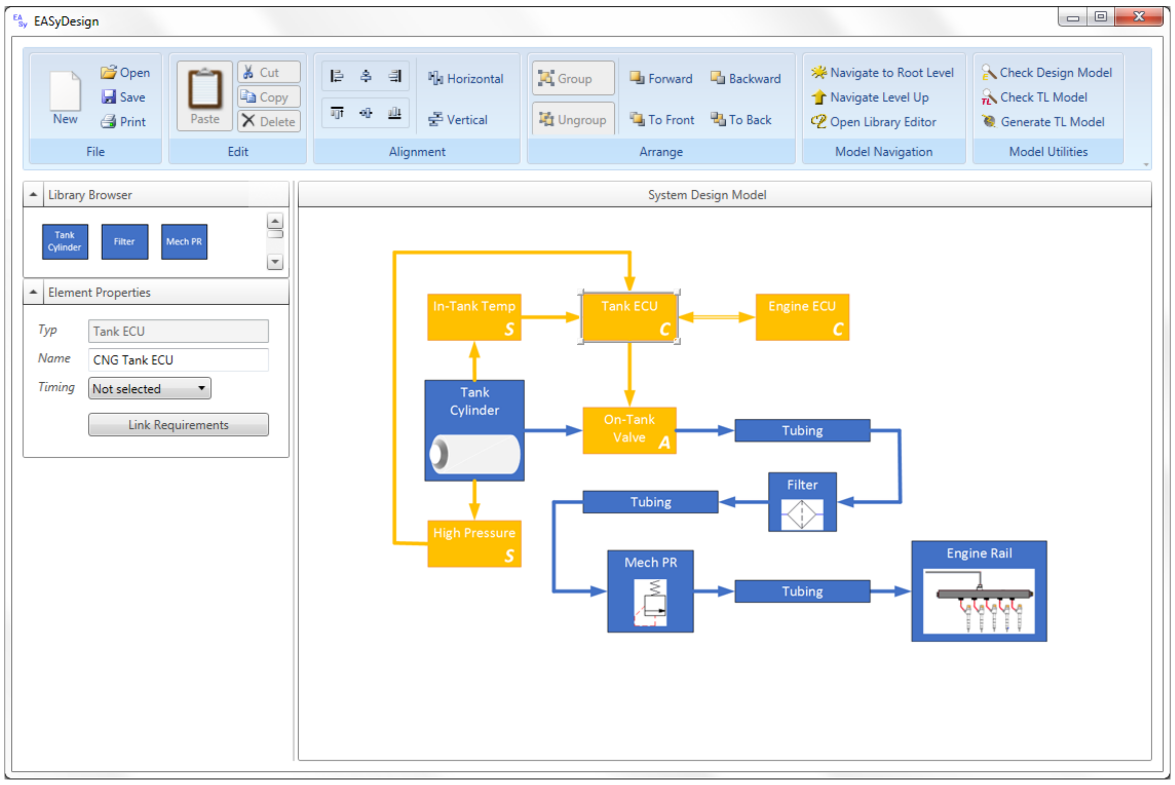
Delete Existing Toolbox Item

Name:

Delete Item

Fig. 2. Library Editor Windows for New Modeling Items

By adding these model items to the library, the system architectural design of the presented use-case can be created as shown in Figure 3. The CNG fuel tank system consists of seven mechanical components, which are blue coloured (Tank Cylinder, Filter, etc.) The medium flow between mechanical components, which is CNG in this use case, is displayed by blue lines with an arrow at the end. Furthermore, five hardware components are placed at the *System Architectural Design Model* level, which are yellow coloured (In-Tank Temperature Sensor,



Typ: Tank ECU

Name: CNG Tank ECU

Timing: Not selected

Fig. 3. CNG Tank System Architectural Design

Tank ECU, etc.) The signal flow between the components is displayed by yellow lines, ending with an arrow. Between the Control Unit and the External Control Unit component, a communication bus is inserted, characterized by the double compound line type and arrows on both ends.

As previously mentioned, the EMS-DSM definition requires at least one hardware component at the model to implement a software component. In this use-case the created basis software component *CAN Driver* shall be integrated at the *CNG Tank ECU*. With a double-click on the hardware component, the next modeling level is opened (named *E/E Item Design Level*), and the *CAN Driver* can be put in place.

5 Conclusions

In the previous sections, a lean method for the design of embedded automotive mechatronics-based E/E-Systems, with a focus on the open toolbox strategy, was presented. This approach has the potential to bring together the different engineering disciplines along the E/E-System development. Many artifacts like requirements, verification criteria, and various specifications can be linked to the models, created with the novel domain-specific modeling language. Supported by the linking of the artifacts, the vital traceability can be established. Depending on the respective tool chain and the organizations process landscape, the EMS-DSM models can also facilitate a single point of truth strategy.

By the model-to-model transformation mentioned in Section 2, a decision between the established SysML design techniques and the presented approach is not necessary. Instead, the EMS-DSM methodology can be utilized as an extension for mechatronics-based system designs to the existing tool chain. However, the possibility of modeling not only the system level, but also the software architectural level enables the presented work to be a standalone solution as well.

First use case implementations show promising results. However, there are several features on the open issue list, which have to be implemented in a next step. On the one hand, the options for describing the systems behavior, like e.g. some kind of task scheduling definition, shall be introduced. On the other hand, an advanced methodology for managing, as well as importing/exporting the signal interfaces has to be developed.

References

1. Andrianarison, E., Piques, J.-D.: SysML for embedded automotive Systems: a practical approach. In: Conference on Embedded Real Time Software and Systems. IEEE (2010)
2. Automotive SIG. Automotive SPICEProcess Assessment Model. Technical report, The SPICE User Group, Version 2.5 (May 2010)
3. Boldt, R.: Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group (July 2009)

4. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. Proceedings of the IEEE **98**(4), 526–545 (2010)
5. Code Project. WPF Diagram Designer - Part 4. Online Resource (March 2008). http://www.codeproject.com/Articles/24681/WPF-Diagram-Designer-Part (accessed March 2015)
6. Giese, H., Hildebrandt, S., Neumann, S.: Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Graph Transformations and Model-Driven Engineering. LNCS, vol. 5765, pp. 555–579. Springer, Heidelberg (2010)
7. Hudak, P.: Domain-specific languages. Handbook of Programming Languages **3**, 39–60 (1997)
8. ISO 26262, Road vehicles - Functional safety. International standard, International Organization for Standardization, Geneva, CH (November 2011)
9. Kawahara, R., Nakamura, H., Dotan, D., Kirshin, A., Sakairi, T., Hirose, S., Ono, K., Ishikawa, H.: Verification of embedded system's specification using collaborative simulation of SysML and simulink models. In International Conference on Model Based Systems Engineering (MBSE 2009), pp. 21–28. IEEE (2009)
10. Macher, G., Armengaud, E., Kreiner, C.: Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In: 7th European Congress Embedded Real Time Software and Systems Proceedings, pp. 256–263 (2014)
11. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys (CSUR) **37**(4), 316–344 (2005)
12. Meseguer, J.: Why Formal Modeling Language Semantics Matters. In: Dingel, J., Schulte, W., Ramos, I., Abrahao, S., Insfran, E. (eds.) International Conference on Model-Driven Engineering Languages and Systems, MODELS 2014, Valencia, Spain. LNCS. Springer International Publishing Switzerland (2014)
13. Meyer, J.: Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards. PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Germany (July 2014)
14. Preschern, C., Kajtazovic, N., Kreiner, C.: Efficient development and reuse of domain-specific languages for automation systems. International Journal of Metadata, Semantics and Ontologies **9**(3), 215–226 (2014)
15. Sporer, H., Macher, G., Kreiner, C., Brenner, E.: A Lean Automotive E/E-System Design Approach with Integrated Requirements Management Capability. In: 9th European Conference on Software Architecture (ECSA 2015), Dubrovnik/Cavtat, Croatia (in press, 2015)
16. Sporer, H., Macher, G., Kreiner, C., Brenner, E.: A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design. In: International Conference on Research in Adaptive and Convergent Systems, RACS 2015, Prague, Czech Republic (2015) (under review)
17. Vujović, V., Maksimović, M., Perišić, B.: Sirius: A rapid development of DSM graphical editor. In: 18th International Conference on Intelligent Engineering Systems (INES), pp. 233–238. IEEE (2014)

RESILIENT INTERFACE DESIGN FOR SAFETY-CRITICAL EMBEDDED AUTOMOTIVE SOFTWARE

Harald Sporer, Georg Macher, Christian Kreiner and Eugen Brenner

Institute of Technical Informatics, Graz University of Technology, Graz, Austria
{sporer,georg.macher,christian.kreiner,brenner}@tugraz.at
http://www.iti.tugraz.at/

ABSTRACT

The replacement of the former, purely mechanical, functionality with mechatronics-based solutions, the introduction of new propulsion technologies, and the connection of cars to their environment are just a few reasons for the continuously increasing electrical and/or electronic system (E/E system) complexity in modern passenger cars. Smart methodologies and techniques have been introduced in system development to cope with these new challenges. A topic that is often neglected is the definition of the interface between the hardware and software subsystems. However, during the development of safety-critical E/E systems, according to the automotive functional safety standard ISO 26262, an unambiguous definition of the hardware-software interface (HSI) has become vital. This paper presents a domain-specific modelling approach for mechatronic systems with an integrated hardware-software interface definition feature. The newly developed model-based domain-specific language is tailored to the needs of mechatronic system engineers and supports the system's architectural design including the interface definition, with a special focus on safety-criticality.

KEYWORDS

Embedded Automotive Systems, Hardware-Software Interface, Model-Based Design, Domain-Specific Modelling, Functional Safety

1. INTRODUCTION

Electrical and/or electronic systems (*E/E systems*) in the automotive domain have grown increasingly complex over the past decades. New functionality, mainly realized through embedded E/E systems, as well as the growing connectivity (*Car2X-Communication*), will keep this trend alive in the upcoming years. Well-defined development processes are crucial for managing this complexity and achieving high quality products. Wide-spread standards and regulations, such as *Automotive SPICE®* and *ISO 26262*, provide guidance through the development life cycle. Some of the key aspects of these concepts are full traceability and consistency between the different development artifacts.

In the automotive industry, the E/E system architectural design models are usually created with techniques based on the *Unified Modeling Language (UML)*. Either the meta-model is extended, or a profile is created to make it possible to use the UML-based approach in embedded automotive system design. A wide-spread example of an UML2 profile is the *Systems Modeling Language (SysML)*, which reuses many of the original UML diagram types (*State Machine Diagram*, *Use Case Diagram*, etc.), uses modified diagram types (*Activity Diagram*, *Block Definition Diagram*, etc.), and adds new ones (*Requirement Diagram*, *Parametric Diagram*) [1].

Even if the UML-based methodologies are valuable for projects with an emphasis on software, they are sometimes too powerful for embedded automotive system design, due to the numerous representation options. Particularly for domain experts who have no or limited knowledge of software development, the large number of elements available for modelling, turns system architectural design into an awkward task. However, it is not the intention of this work to decry the SysML approaches created so far. They are a good choice for a multitude of tasks. Instead, this paper showcases an extension of these SysML approaches, which makes the architectural design process easier, placing a special focus on the specification of the hardware-software interface for UML non-natives.

A model-based domain-specific language and domain-specific modelling (DSM) has been developed for the specific needs of embedded automotive mechatronics systems. Additionally, a software tool has been created to support the new modelling techniques. By linking development artifacts such as requirements (e.g. technical system requirements, software requirements, etc.), and verification criteria to the design model, the traceability mentioned earlier is assured.

The main goal of this work is to contribute to the improvement of the existing system architectural design methods by facilitating the specification of the hardware-software interface. The approach presented has mainly been created for the development of embedded mechatronics-based E/E systems in the automotive field. However, the techniques are also suitable for other domains. Improvements have been made by extending the system modelling approach presented in previous publication using HSI specification capabilities.

Section 2 presents an overview of related approaches, domain-specific modelling and integrated tool chains. Section 3 provides a description of the proposed hardware-software interface specification approach for the model-based system engineering. An application of the methodology described is presented in Section 4. Finally, this work is concluded in Section 5, which gives an overview of the presented work.

2. RELATED WORK

In recent years, a lot of effort has been made to improve the model-based automotive E/E system design methods and techniques. Today, the advantages of a model-based approach are clear and without controversy. Meseguer [2] grants much more reliability, reusability, automatisation, and cost effectiveness to software that is developed with modelling languages. However, model transformation within or across different languages is crucial to achieve all these benefits.

Traceability and consistency between the development artifacts have always been important topics. However, these properties have become even more important due to the increasing number of electronic and electric-based functionalities. According to the international standard ISO 26262 [3], released in 2011, traceability between the relevant artifacts is mandatory for safety-critical systems. A description of the common deliverables relevant to automotive E/E system development, and a corresponding process reference model is presented by the de facto standard Automotive SPICE [4]. Neither the functional safety standard nor the process reference model enforces a specific methodology for how the development artifacts have to be created or linked to each other. However, connecting the various work products manually is a tedious and error-prone task.

One of the early work products found in the engineering process is the system architectural design. In the field of automotive E/E system development, a wide-spread and common approach is to utilize a UML-based technique for this design, such as the UML2 profile SysML. Andrianarison and Piques [5], Boldt [6], and many other publications (e.g. [7], [8], [9]) present their SysML methodologies for system design. As stated by Broy et al. [10], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities of how to customize the UML diagrams and how to get a language for embedded system design, are behind these drawbacks. Even if there is an agreement to utilize a common UML profile such as SysML, there are plenty of design artifact variations. This scenario does not provide an optimal base for the engineer who has to design the embedded automotive system from a mechatronics point of view. Ideally, the tool should be intuitive and it should be possible to use it easily without specific knowledge of UML.

Mernik et al. [11] describe a domain-specific language as a language that is tailored to the specific application domain. This tailoring should lead to a substantial increase in expressiveness and ease of use, compared to general-purpose languages. Even if expressiveness is increased by the utilization of SysML-based modelling techniques, the ease of use for embedded automotive mechatronics system design has not been improved.

Preschern et al. [12] claim that DSLs help to decrease system development costs by providing developers with an effective way to construct systems for a specific domain. The benefit in terms of a more effective development has to be greater than the investment needed to create or establish a DSL at a company or in a department. In addition, the authors argue that the mentioned DSL development cost will decrease significantly over the next few years, due to new tools that support language creation such as the Eclipse-based *Sirius*[1].

Vujovic et al. [13] present a model-driven engineering approach to creating domain-specific modelling (DSM). Sirius is the framework used to develop a new DSM and the DSM graphical modelling workbench. The big advantage of this tool is that the workbench for the DSM is developed graphically. Therefore, knowledge about software development with Java, the graphical editor framework (GEF) or the graphical modelling framework (GMF) is not needed.

Although it is obvious that an unambiguous specification of the various signals between the items of an embedded automotive system design is vital, publications on embedded automotive hardware-software interface definition are rare. This contribution aims to extend a model-based development approach for an ISO 26262 aligned hardware-software interface definition presented by the authors of [14]. More background on the origin of HSI characteristics is presented and the model-based support is shifted from a classic SysML-based methodology to a domain-specific modelling methodology for the E/E system architectural design of mechatronics-based systems. The domain-specific modelling (DSM) language definition is presented in [15].

3. Approach

The main goal of this contribution is to convey the importance of the hardware-software interface for today's *Embedded Automotive Systems* and how it is supported by the approach described. Moreover, the key driving factors for establishing a well-defined interface, which is also suitable for safety-critical applications, will be shown within this section. Before describing the HSI specification approach in detail, the utilized domain-specific model-based system architectural design technique shall be introduced. This domain-specific modelling method has been developed to outline mechatronics-based system architectures in the automotive sector and therefore serves as a basis for the specification of the hardware-software interface found in our approach.

3.1. Embedded Mechatronics System Domain-Specific Modelling

[1] https://eclipse.org/sirius/

The key objective of domain-specific modelling is to provide a lean approach for engineers to facilitate embedded automotive mechatronics system modelling on a high abstraction level. The approach described focusses on the model-based structural description of the E/E system under development. Additionally, the signals and interfaces are an essential part of modelling.

The existing SysML-based design method (see also [14]) is extended by the newly developed *Embedded Mechatronics System Domain-Specific Modeling (EMS-DMS)* for automotive embedded system architectural design. It is not intended to replace the SysML-based solution created so far. Instead, the EMS-DSM is integrated into existing methods. Hence, the whole tool-chain, starting from the SysML-based system architectural design tool and finishing at software / hardware architectural design, can be utilized if desired. An overview of the tool integration is shown in Figure 1.

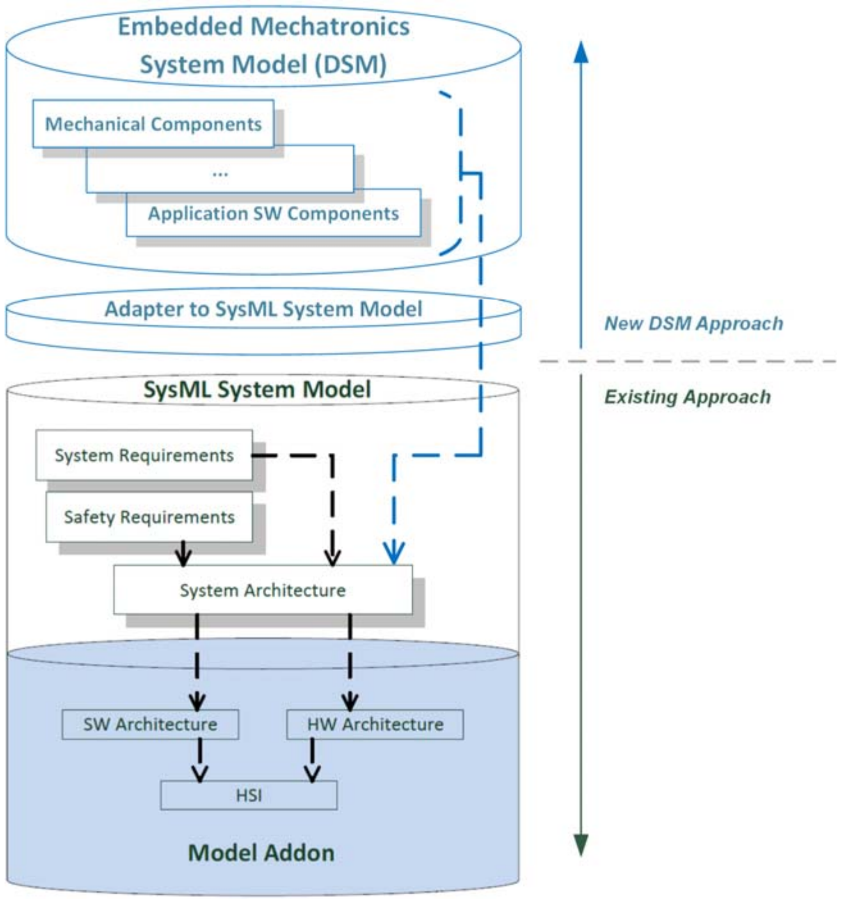


Figure 1. Tool-Chain Integration of DSM and SysML Model Approach (based on [16])

The definition of the newly developed model-based domain-specific language is shown in Figure 2. The *EMS-DSM Component* is the origin of all other classes regarding language definition. The six attributes of this class are

- *ID* - unique identifier of the particular instance in the architectural design model, set automatically.

- *Name* - name or short description of the particular instance, chosen by the design engineer.

- *Mask* - graphical representation of the particular instance, set by the engineer responsible for the design tool.

- *Requirement* - in this approach, a link to the Redmine requirements database is set by the designer.

- *Verification Criteria* - similar to Requirement, a link to the Redmine verification criteria artifact is set by the designer.

- *Specification* - link to further information about the actual component, e.g. a CAD drawing or a data sheet.

The *EMS-DSM Component* serves as the base node of the EMS-DSM definition, and declares the common attributes of the derived classes at the lower levels. Therefore, this component is not instanced for the design process. At the next language definition level, the following component classes are available:

- *Mechanical Components* - used by all mechanical, domain-specific components, e.g. the *Mechanical Pressure Regulator* class in the use-case shown in Section 4.

- *Compartment Components* - gives the opportunity to specify areas or compartments, where mechanical and hardware components are installed.

- *E/E Item Components* - an abstract component class definition, which serves as a basis for the hardware and software components at the lower levels. Additionally, the property *ASIL*, corresponding to the ISO 26262, is stated.

The majority of the non-abstract component classes are derived from the hardware component class:

- *Sensor Component* - used for all domain-specific sensor components.

- *Control Unit Component* - used for all domain-specific control unit components.

- *Actuator Component* - used for all domain-specific actuator components.

- *External Control Unit Component* - special class, to make signals from an external system available in the considered system.

All hardware components and their instances in the system design model, with the exception of the *External Control Unit Component*, are capable of containing a software design model. This means that any kind of software component instance is only allowed to be implemented in a software design model which belongs to an instance of a hardware component. This special language characteristic is defined by the *Aggregation* relationship between hardware and software components, which also implies the hardware-software interface.

The last part of the EMS-DSM definition description is related to the classes (derived from the software component):

- *Basis Software Component* - used for all low-level, hardware-dependent software components.

- *Application Software Component* - used for all functional software components.

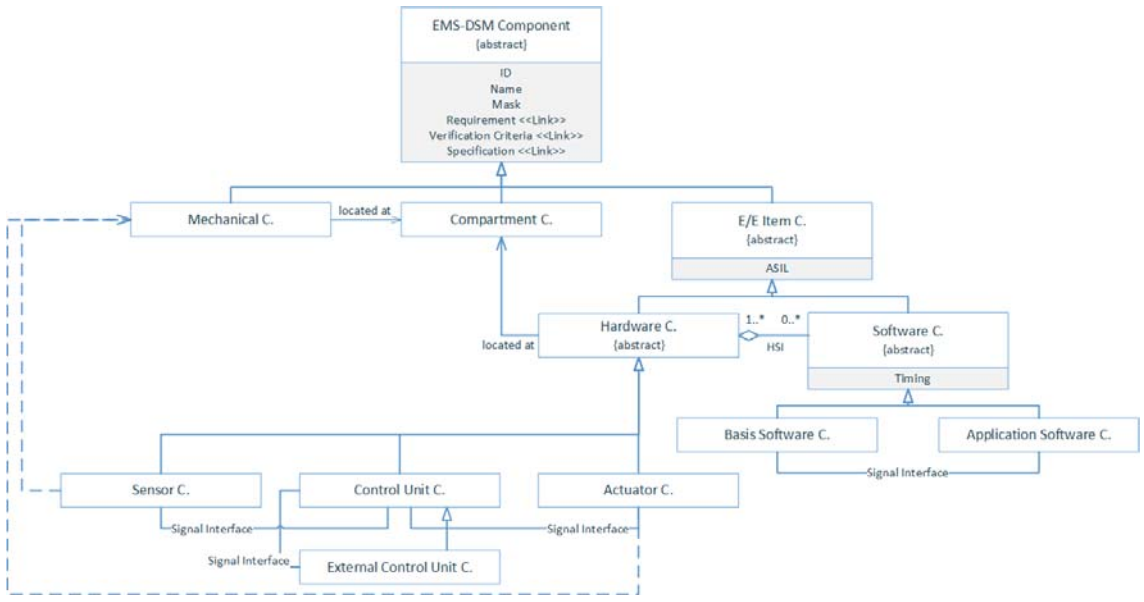


Figure 2. EMS-DSM Language Definition

As mentioned in Section 2, a more detailed description of the domain-specific modelling language can be found in [15].

3.2. Influence of Process Reference Model on HSI Specification

Due to their broad dissemination in the automotive sector, the two most important reference models are Automotive SPICE [4] and CMMI [17]. Both pursue similar targets: they (a) determine the process capability/maturity, and (b) aspire a continuous process improvement in the particular development team and/or company. The reference models do not exist in order to specify how processes have to be implemented. Instead, desired process outcomes (Automotive SPICE) or goals (CMMI) are defined and described in more detail by best practice characterisation (base or generic practices at Automotive SPICE, and specific or generic practices at CMMI). The *Automotive S(oftware) P(rocess) I(mprovement) and C(apability) (D)e(termination)* reference model is based on the international standard ISO 15504 and is primarily used in Europe, as well as in some parts of Eastern Asia. The latest version, which was analysed for this approach, is 3.0 and was released in July 2015. The *C(apability) M(aturity) M(odel) I(ntegration)* reference model has been developed by the Software Engineering Institute (SEI) at Carnegie Mellon University. CMMIs currently exist for *Acquisition*, *Development*, and *Services*. As CMMI is not widespread in the European automotive sector, the remaining part of this section will focus on Automotive SPICE as the relevant process assessment and reference model. The model does not address the demand for a hardware-software interface directly, but some guidance on HSI specification can be extracted from general interface topics.

Table 1 lists the elements of the Automotive SPICE reference model that provide information about interfaces between system components. As expected, interface work products are needed for *Architectural Design* and the *Integration* topics. In addition to the *Process ID* and the *Process Name*, the corresponding *Base Practice IDs* are indicated. These give more detailed information on what the outcome should look like. In SYS.3.BP3, the definition (*identify*, *develop*, and *document*) of system element interfaces is stipulated. This equally applies to the hardware-software interface. In SYS.3.BP4, a description of the dynamic behaviour of and between the system elements is provided. The possible operating modes of the system, which determine the dynamic behaviour, have to be taken into account in the HSI definition. Base

Practice SYS.4.BP3 postulates that the interfaces between system items have to be covered by the system integration test to show consistency between the real interfaces and the architectural design. With regard to the HSI, SWE.2.BP3 and SWE.2.BP4 can be interpreted in a similar way to their system level counterparts (SYS.3.BP3, SYS.3.BP4). SWE.2.BP5 claims the determination and documentation of the resource consumption objectives of all relevant software architectural design elements. To support this using the hardware-software interface definition, information on resource consumption shall be included in the description of the signals, wherever applicable. An interface definition is also demanded at process *SWE.3 - Software Detailed Design and Unit Construction*. However, in this case, the specification belongs to the signals communicated between the components on the lowest (most detailed) software level. Hence, this communication specification does not directly belong to the hardware-software interface, and will not be taken into consideration in this approach. The last process/base practice in Table 1 is SWE.5.BP3. It demands a description of the interaction between relevant software units and their dynamic behaviour. Again, this base practice can be interpreted in a similar way to its system level counterpart (SYS.4.BP3).

Table 1. HSI Accompanying Automotive SPICE Processes.

Process ID	**Process Name**	**Base Practice ID**
SYS.3	System Architectural Design	BP3, BP4
SYS.4	System Integration and Integration Test	BP3
SWE.2	Software Architectural Design	BP3, BP4, BP5
SWE.5	Software Integration and Integration Test	BP3

In the Automotive SPICE reference model, *Output Work Products* are also defined and linked to the base practices previously stated. From this contribution's point of view, the relevant work products are:

- *System Architectural Design* - the main aspects to consider regarding the HSI are *memory/capacity requirements*, *hardware interface requirements, security/data protection characteristics, system parameter settings, system components operation modes*, and the influence of the *system's and system component's dynamic behaviour.*

- *Interface Requirement Specification* - the main aspects to consider regarding the HSI are *definition of critical timing dependencies or sequence ordering* and *physical interface definitions.*

3.3. Influence of Automotive Functional Safety on HSI Specification

The international standard *ISO 26262* for *Functional Safety* in the automotive electrical and/or electronic system domain was released in 2011. Since then, many best practice articles and books have been published on how to develop according to the standard. However, with the exception of the safety-critical view, the hardware-software interface has rarely been highlighted in these publications.

According to ISO 26262, the HSI is to be specified during the phase *Product Development at the System Level* (see Figure 3), which is described in Part 4 of the standard. As a prerequisite for specifying the hardware-software interface, a system design has to be established. While preparing the system architectural design, the technical safety and non-safety requirements are allocated to the hardware and software. Subsequent to this allocation, an initial interface description can be prepared. The HSI shall be continuously refined in the ensuing hardware and software product development phases, which are described in Parts 5 & 6 of the ISO 26262.

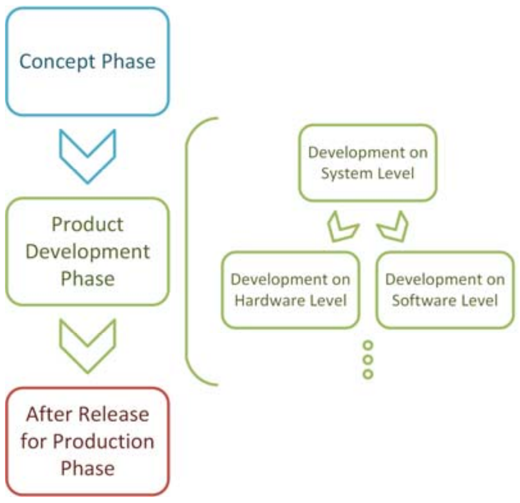

Concept Phase
Product Development Phase
After Release for Production Phase
Development on System Level
Development on Hardware Level
Development on Software Level

Figure 3. Development Phases According to [3]

The majority of information concerning how to specify the interface aligned to functional safety can be found in Clause 7.4.6 of Part 4 of the standard. In our approach, most of the HSI characteristics demanded by this clause, such as *operation modes of the hardware device* and *shared/exclusive use of the hardware resource*, are described in the *Detailed Hardware Specification (DHS)* documents, which are linked to the main HSI document. A detailed description of the various development artifacts and their relationships is presented in Subsection 3.4. Additionally, the informative *Annex B* of Part 4 of ISO 26262 provides information concerning the possible content of the interface definition.

3.4. Incorporated Hardware-Software Interface Specification

Two main objectives have to be achieved when developing a new HSI specification approach:

1. identification, development and documentation of the essential HSI specification attributes & characteristics, and

2. support for the linking of related information to ensure full traceability.

The principle of the hardware-software interface specification approach described here is based on three origins, two of which have been described in the previous subsections:

a. the process reference and assessment model *Automotive SPICE*,

b. the automotive functional safety standard *ISO 26262*, and

c. the industrial experience of authors in past automotive E/E system development projects.

It is important to note that the hardware-software interface specification does not only consist of a single spreadsheet with a description of all signals between hardware and software. Further information belonging to the HSI specification can also be found in various development artifacts. Figure 4 shows the different aspects of our HSI specification approach:

- *Hardware-Software Interface Signal List* - spreadsheet with data of all signals between hardware and software. The attributes describing each signal have been derived from sources (a) - (c), which were mentioned at the beginning of this subsection.

- *Resource Consumption Objectives* - depending on the particular project, the objectives are described in spreadsheet(s) and/or free text document(s). Regardless of the type, the documents are linked to the software components in software architectural design (see attribute *Specification <<Link>> Software Component* class in the EMS-DSM language definition in Figure 2).

- *Detailed Hardware Specification* - depending on the particular project, the objectives are described in spreadsheet(s) and/or free text document(s). Regardless of the type, the documents are linked to the hardware components in system architectural design (see attribute *Specification <<Link>> Hardware Component* class in the EMS-DSM language definition in Figure 2).

- *Model-based Architectural Design* - this item represents the central source of information. The defined domain-specific modelling language facilitates the creation of the system and the software architectural design within the same design environment and allows the linking of all other relevant development artifacts. From a HSI specification perspective, the three previous items in this list are the most important development artifacts to be linked to the architectural design models.

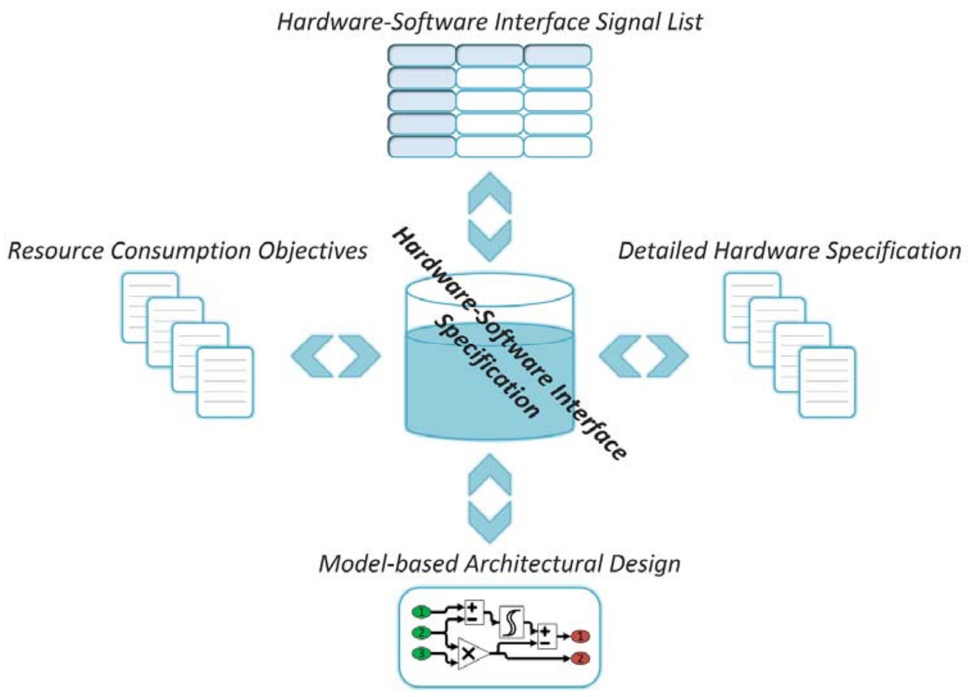

Hardware-Software Interface Signal List
Resource Consumption Objectives
Hardware-Software Interface Specification
Detailed Hardware Specification
Model-based Architectural Design

Figure 4. Distributed Hardware-Software Interface Specification

Establishing full traceability between the *Resource Consumption Objectives*, the *Detailed Hardware Specification*, and the *Model-based Architectural Design* is an easy task, accomplished by linking the related documents in the architectural design.

The integration of the *Hardware-Software Interface Signal List* data into the design model is more technically challenging. In [14] the authors described the functionality of the *HSI Definition Exporter and Importer*, which was developed to achieve a seamless transformation of the HSI representation between the SysML-based architectural design and the spreadsheet tool. The *HSI Definition Exporter* is an extension (*dynamic link library*) for the model-based development (MBD) tool, which is written in C# and allows the modelled HSI to be exported to a spreadsheet document (either in *csv* or *xls* format). The *HSI Definition Importer* is the counterpart of the *HSI Definition Exporter*, which is also implemented as a *dynamic link library* using the spreadsheet tool's API. It allows the import of all HSI information from the spreadsheet document or a selective update of the HSI model artifacts. Using both the export and import functionality leads to a round-trip engineering capability regarding the HSI signal list and the HSI signals modelled in the architectural design. In this approach, the libraries of the exporter and importer extensions are slightly adapted to the needs of the domain-specific modelling language.

To conclude the description of our approach, the HSI signal attributes and their origins are listed in Table 2.

Table 2. HSI Signal List Attributes.

Attribute	**Comments**	**Origin**
Signal Direction	Input or Output, out of the controllers view	Author's Experience
Signal Description	A short signal description or the signals name	ISO 26262-4 (Annex B)
Sensor / Actuator	Type or identifier of signals source/sink	Author's Experience
Supply Voltage	-	Author's Experience
Physical Min Value	-	ASPICE SYS.4.BP3
Physical Max Value	-	ASPICE SYS.4.BP3
Accuracy	In % of range of values	ISO 26262-4 (Annex B)
Physical Unit	E.g. V, A, ...	ISO 26262-4 (Annex B) ASPICE SYS.4.BP3
HW Interface Type	E.g. Digital In, Analog Out, CAN, ...	ISO 26262-4 (Annex B) ASPICE WP 17-08
HW Pin #	Pin number or identifier at e.g. ECU	ISO 26262-4 (Annex B)
Message ID	In case of bus communication	Author's Experience
Start Bit		
Internal Cycle Time	E.g. 10 ms	ISO 26262-4 (Section 7.4.6) ASPICE SYS.4.BP3, SWE.5.BP3, WP 17-08
External Cycle Time	Only applicable for digital signals	Author's Experience
HW Timer / Interrupt / Watchdog	Identifier of triggered e.g. interrupt	ISO 26262-4 (Section 7.4.6)
Operating Modes	Information if signal is needed special operating modes (e.g. start up, calibration, ...)	ISO 26262-4 (Annex B) ASPICE SYS.3.BP4, SWE.2.BP4, WP 04-06
HW Diagnostic Feature	E.g. short circuit detection, ...	ISO 26262-4 (Section 7.4.6)
Memory Type	E.g. RAM, EEPROM, ...	ISO 26262-4 (Annex B)
Security/Data Protection	Information on special security issues	ASPICE WP 04-06
Critical Timing Dependencies or Sequence Ordering	-	ASPICE WP 17-08
Signal Name @ SW	Identifier of signal as used in application software	Author's Experience

Initial Value	-	Author’s Experience
Data Type	E.g. UInt16, Float, ...	ASPICE SYS.4.BP3, SWE.5.BP3
Scaling LSB	Scaling information in case of fixed-point arithmetic	ASPICE SYS.4.BP3, SWE.5.BP3
Scaling Offset		
Min Value @ SW	-	ASPICE SWE.5.BP3
Max Value @ SW	-	ASPICE SWE.5.BP3
Accuracy @ SW	In % of range of values	ISO 26262-4 (Annex B)
Physical Unit @ SW	E.g. km/h, Nm, ...	ASPICE SWE.5.BP3
Default Value @ SW	Default value in case of an invalid input signal	Author’s Experience
Detection Time	Time until a fault is diagnosed	ISO 26262-4 (Section 7.4.6)
Reaction Time	Admissible reaction time after a fault was detected	ISO 26262-4 (Section 7.4.6)
ASIL	Automotive Safety Integrity Level classified A - D, or QM if no safety-relevance is given	ISO 26262-4 (Annex B)
Signal ID	Identifiers required for the support of the domain-specific modelling approach	Author’s Experience
HW Device ID		

4. APPLICATION

In this section, the HSI specification approach is applied to the development of an automotive fuel tank system for compressed natural gas (CNG). For an appropriate scale of the showcase, only a small part of the real-world system is utilized. The application should be seen as illustrative material, reduced for internal training purposes for students. Therefore, the disclosed and commercially non-sensitivity use-case is not intended to be exhaustive or representative of leading-edge technology. Before the showcase is illustrated, tool support regarding both domain-specific modelling and requirements management shall be explained briefly.

4.1. EMS-DSM Language Tool Support

Generally speaking, the EMS-DSM language can be supported by various tools, but at the time when the research project was initiated, the highest possible flexibility was desired, as was full access to the tool’s source code. To avoid developing an application from scratch, the open source project *WPF Diagram Designer* (see [18]) was chosen as a basis for tool development. The corresponding documentation has about 540,000 views and the source code has been downloaded more than 24,000 times. Therefore, the source, which provides standard functionality such as file handling and basic graphical modelling, is well reviewed. The source code is written in C# and provides good expandability. New functionalities have been implemented for the diagram designer, named *EASy-Design* (**E**mbedded **A**utomotive **Sy**stem-**Design**), to facilitate engineering with EMS-DSM models. However, EASy-Design is just one possibility for EMS-DSM tool support. The methodology and its C# implementation can be ported to e.g. Enterprise Architect[2] by the provided *Add-in* mechanism. Another alternative is the Eclipse[3] framework, or rather the Eclipse-based project *Sirius*, which enables the creation of a graphical modelling workbench, by facilitating the Eclipse modelling technologies without writing code.

4.2. Project and Requirements – Management Tool Support

[2] http://www.sparxsystems.com/
[3] http://eclipse.org/

The web-based open source application *Redmine*[4] is used for topics such as project management and requirements management in this approach. Owing to its high flexibility through configuration, new trackers have been added for development according to the de facto standard Automotive SPICE [4]. The process reference model already mentioned in Section 3 defines three different types of requirements of the engineering process group: *Customer Requirements*, *System Requirements*, and *Software Requirements*. The hardware focus is missing from the embedded E/E system view. Additionally, requirements and design items for mechanical components have to be introduced for the design of an embedded mechatronics-based E/E system. Similar to the Automotive SPICE methodology on a system and software level, engineering processes have been defined for these missing artifacts. To sum up, the available requirement and test case types for this approach are: *Customer Req*, *System Req*, *System TC*, *System Integration TC*, *Software Req*, *Software TC*, *Software Integration TC*, *Hardware Req*, *Hardware TC*, *Mechanics Req*, and *Mechanics TC*.

The test case and requirement items are connected to each other by their unique identifier. For a safety-critical development according to ISO 26262, additional issue types such as *Functional Safety Requirements* have been added. By reconfiguring the project management tool Redmine, all requirement types mentioned have been implemented.

4.3. CNG Tank System Showcase

Figure 5 illustrates the EMS-DSM tool *EASy-Design* with the system architectural design model of the simplified showcase. The CNG fuel tank system consists of seven mechanical components, which are blue coloured (Tank Cylinder, Filter, etc.) The medium flow between mechanical components, which is CNG in this case, is displayed by blue lines with an arrow at the end. Furthermore, five hardware components are placed at the system design model level, which are yellow coloured (In-Tank Temperature Sensor, Tank ECU, etc.) The signal flow between the components is displayed using yellow lines ending with an arrow. A communication bus is inserted between the *Control Unit* and the *External Control Unit* component, shown by the double compound line type and arrows at both ends.

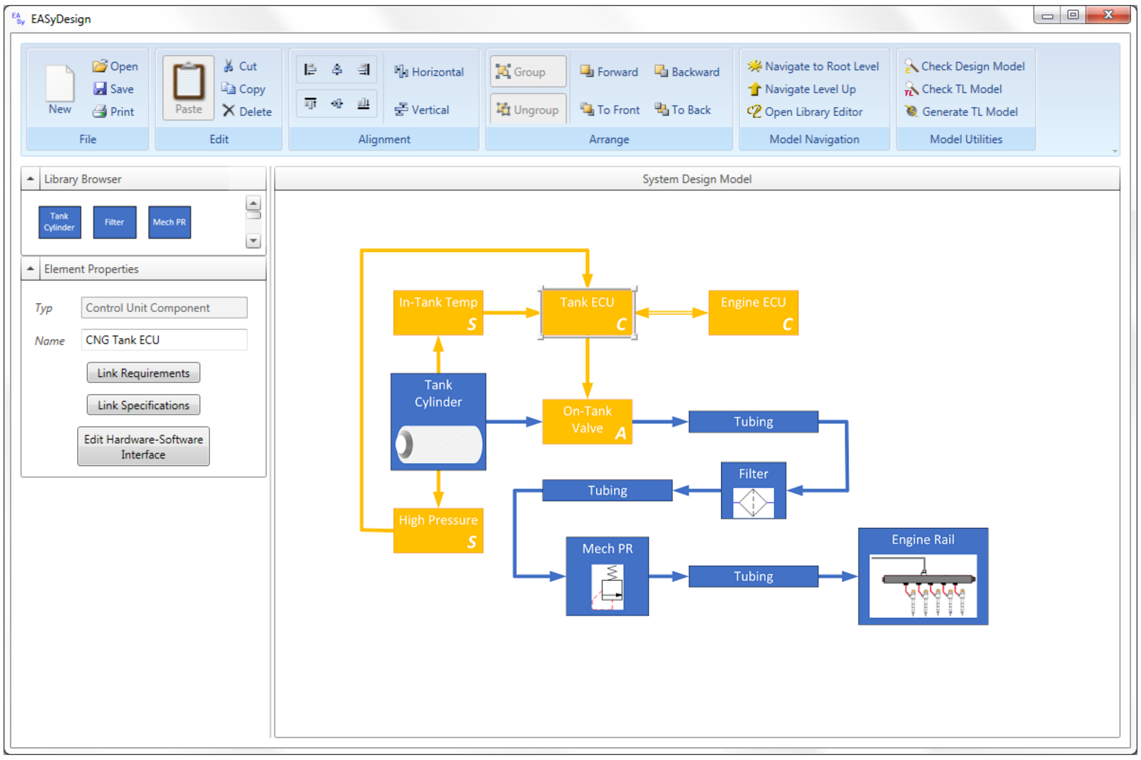
By selecting a model element and clicking the button *Link Requirements*, the elements requirements dialogue is opened and a link between the selected element and an item from the requirements database (e.g. *System Requirement*, see Subsection 4.2) can be established. Already linked requirements from Redmine's MySQL database are listed with their ID, Type, Title, ASIL, and Core functionality attribute. With a click on *Link Specifications*, various documents, such as detailed hardware specifications and datasheets, can be linked to the selected model element.

The *Hardware-Software Interface Specification* emphasis of this contribution is also supported by EASy-Design. Again, a hardware element of the model has to be selected and can be defined with a subsequent click on the button *Edit Hardware-Software Interface* in the *Element Properties* group, the interface of the selected hardware item. In Figure 5, the Tank ECU has been selected and in Figure 6, the newly opened HSI definition dialogue for the Tank ECU is illustrated. Within this dialogue, all operations needed to add, modify or delete signals can be triggered by clicking the relevant button:

- *Add New Signal* - a new dialogue window is opened and a signal can be created by entering the properties described in Table 2 (see Figure 7).
- *Add Connected Signal* - the hardware elements in the architectural system design can be connected by (yellow) lines as described in Subsection 4.1. Every output signal from any connected hardware element can be added as an input signal in the HSI signal definition in the actual hardware element.

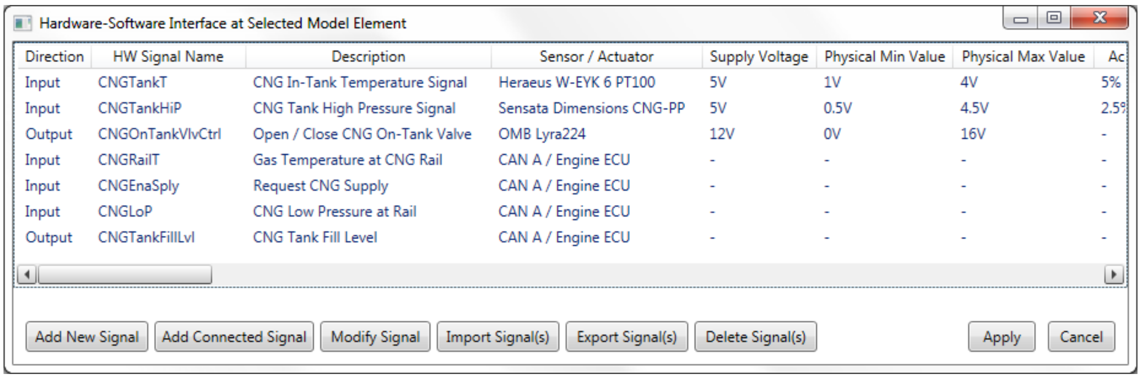
[4] http://www.redmine.org/

- *Modify Signal* - at the HSI signal definition main dialogue (illustrated in Figure 6), a signal has to be selected, for which the modification dialogue will be opened after a click on *Modify Signal*. The signal modification dialogue is similar to the *Add New Signal* dialogue.
- *Import Signal(s)* - the *HSI Definition Importer*, as described in Subsection 3.4, is selected, and signals from a HSI signal definition stored in spreadsheet format can be added to the system architectural design model.
- *Export Signal(s)* - the *HSI Definition Exporter*, as described in Subsection 3.4, is selected and signals from the HSI signal definition in the system architectural design model can be exported to a HSI signal definition in spreadsheet format.
- *Delete Signal(s)* - the signals have to be selected from the main HSI signal definition dialogue and are removed from the interface when the button is clicked.



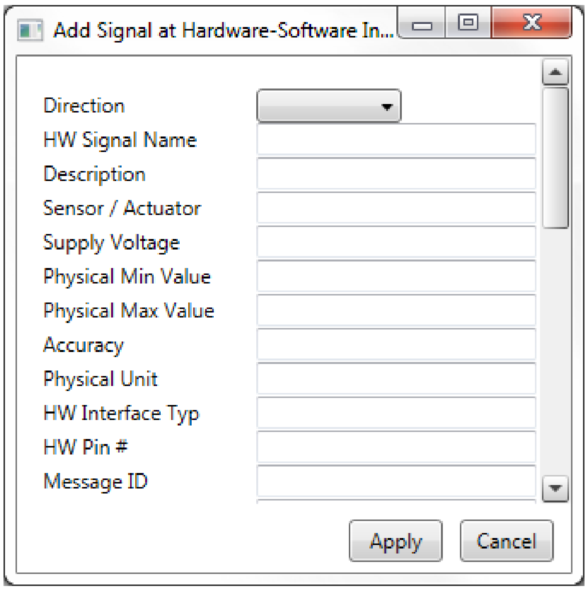
Typ: Control Unit Component
Name: CNG Tank ECU

Figure 5. Self-developed tool *EASy Design* with a Simplified CNG Tank System Architectural Design


Hardware-Software Interface at Selected Model Element

Direction	HW Signal Name	Description	Sensor / Actuator	Supply Voltage	Physical Min Value	Physical Max Value	Ac
Input	CNGTankT	CNG In-Tank Temperature Signal	Heraeus W-EYK 6 PT100	5V	1V	4V	5%
Input	CNGTankHiP	CNG Tank High Pressure Signal	Sensata Dimensions CNG-PP	5V	0.5V	4.5V	2.5
Output	CNGOnTankVlvCtrl	Open / Close CNG On-Tank Valve	OMB Lyra224	12V	0V	16V	-
Input	CNGRailT	Gas Temperature at CNG Rail	CAN A / Engine ECU	-	-	-	-
Input	CNGEnaSply	Request CNG Supply	CAN A / Engine ECU	-	-	-	-
Input	CNGLoP	CNG Low Pressure at Rail	CAN A / Engine ECU	-	-	-	-
Output	CNGTankFillLvl	CNG Tank Fill Level	CAN A / Engine ECU	-	-	-	-

Figure 6. Hardware-Software Interface Dialog at *EASy Design*


Add Signal at Hardware-Software In...

Direction	
HW Signal Name	
Description	
Sensor / Actuator	
Supply Voltage	
Physical Min Value	
Physical Max Value	
Accuracy	
Physical Unit	
HW Interface Typ	
HW Pin #	
Message ID	

Figure 7. Hardware-Software Interface Add New Signal Dialog at *EASy Design*

As can be seen in Figure 5, no *Software Components* are modelled at this level (System Design Model). With a double-click on a *Hardware Component* (e.g. *Tank ECU*), the next modelling level is opened (named *E/E Item Design Level*). The (green coloured) *Basis Software Components* and *Application Software Components* can be placed here. At each basis software component, the input and output signals from the HSI definition in the particular hardware component can be used and therefore connected to the software.

5. Conclusion

Previous sections described the factors influencing the development of our hardware-software interface specification approach as well as the supporting tools. A domain-specific modelling method for the design of embedded automotive mechatronics-based E/E systems formed the basis for this work. This approach has the potential to bring together the different engineering disciplines involved in E/E system development by facilitating the HSI specification process. Additionally, many artifacts such as requirements, verification criteria, and various specifications can be linked to the models, created with the new, domain-specific modelling language. With the help of the linked artifacts, vital traceability can be established. Depending on the respective tool chain and the organisation's process landscape, the EMS-DSM models can also facilitate a single point of truth strategy.

First use case implementations show promising results. However, there are several features that still need to be implemented. Options for describing the system's behaviour, e.g. a kind of task scheduling definition, are to be introduced. Furthermore, the Model-to-Model-Transformer between the domain-specific and traditional SysML system architectural design model has to be extended to achieve an automatic transformation of the HSI signal definition between the different modelling strategies.

REFERENCES

[1] S. Friedenthal, A. Moore, and R. Steiner, "OMG Systems Modeling Language (OMG SysMLTM) Tutorial," in INCOSE International Symposium, 2006.

[2] J. Meseguer, "Why Formal Modeling Language Semantics Matters," in Model-Driven Engineering Languages and Systems, ser. Lecture Notes in Computer Science, J. Dingel, W. Schulte, I. Ramos, S. A. ao, and E. Insfran, Eds., vol. 17th International Conference, MODELS 2014, Valencia, Spain, no. 8767. Springer International Publishing Switzerland, 2014, keynote.

[3] "ISO 26262, Road vehicles - Functional safety," International Organization for Standardization, Geneva, CH, International Standard, November 2011.

[4] VDA QMC Working Group 13 / Automotive SIG, "Automotive SPICE Process Assessment / Reference Model," Tech. Rep. Revision ID: 470, July 2015, version 3.0.

[5] E. Andrianarison and J.-D. Piques, "SysML for embedded automotive Systems: a practical approach," in Conference on Embedded Real Time Software and Systems. IEEE, 2010.

[6] R. Boldt, "Modeling AUTOSAR systems with a UML/SysML profile," IBM Software Group, Tech. Rep., July 2009.

[7] H. Giese, S. Hildebrandt, and S. Neumann, "Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent," LNCS 5765, pp. 555 –579, 2010.

[8] R. Kawahara, H. Nakamura, D. Dotan, A. Kirshin, T. Sakairi, S. Hirose, K. Ono, and H. Ishikawa, "Verification of embedded system's specification using collaborative simulation of SysML and simulink models," in International Conference on Model Based Systems Engineering (MBSE'09). IEEE, 2009, pp. 21–28.

[9] J. Meyer, "Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards," Ph.D. dissertation, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Germany, July 2014.

[10] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments," Proceedings of the IEEE, vol. 98, no. 4, pp. 526–545, 2010.

[11] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," ACM computing surveys (CSUR), vol. 37, no. 4, pp. 316–344, 2005.

[12] C. Preschern, N. Kajtazovic, and C. Kreiner, "Efficient development and reuse of domain-specific languages for automation systems," International Journal of Metadata, Semantics and Ontologies, vol. 9, no. 3, pp. 215–226, 2014.

[13] V. Vujovic, M. Maksimovic, and B. Perisic, "Sirius: A rapid development of DSM graphical editor," in 18th International Conference on Intelligent Engineering Systems (INES). IEEE, 2014, pp. 233–238.

[14] G. Macher, H. Sporer, E. Armengaud, E. Brenner, and C. Kreiner, “Using Model-based Development for ISO26262 aligned HSI Definition,” in Critical Automotive applications: Robustness & Safety, ser. CARS@EDCC2015, Paris, France, 2015.

[15] H. Sporer, “A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design,” in International Conference on Research in Adaptive and Convergent Systems (RACS 2015), ser. RACS ’15, Prague, Czech Republic, 2015.

[16] G. Macher, E. Armengaud, and C. Kreiner, “Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain,” in 7th European Congress Embedded Real Time Software and Systems Proceedings, 2014, pp. 256–263.

[17] Software Engineering Institute, “CMMI for Development, Version 1.3,” SEI, Carnegie Mellon, Tech. Rep. CMU/SEI-2010-TR-033, ESCTR-2010-033, November 2010.

[18] Code Project, “WPF Diagram Designer - Part 4,” Online Resource, March 2008, http://www.codeproject.com/Articles/24681/WPFDiagram-Designer-Part, accessed Mar 2015.

Authors

Harald Sporer received a MSc. degree in Telematics from Graz University of Technology. He worked as software development engineer on Hardware-in-the-Loop (HIL) systems at AVL List GmbH and as functional software developer for embedded automotive systems at Magna Powertrain AG & Co KG. Currently he is working on his PhD at the Institute of Technical Informatics at Graz University of Technology. Parallel to his PhD thesis he is also active in the field of embedded automotive system design, engineering process improvement, and functional safety engineering.

Georg Macher received a MSc. degree in Telematics and worked as software development engineer on prototype vehicles at AVL List GmbH. Currently he joined the R&D department of AVL's powertrain engineering branch and is working on his PhD at Institute for Technical Informatics at Graz University of Technology. Parallel to his PhD thesis is also active in the field of system, software, and functional safety engineering.

Dr. Christian Kreiner graduated and received a PhD degree in Electrical Engineering from Graz University of Technology in 1991 and 1999 respectively. 1999-2007 he served as head of the R&D department at Salomon Automation, Austria, focusing on software architecture, technologies, and processes for logistics software systems. He was in charge to establish a company-wide software product line development process and headed the product development team. During that time, he led and coordinated a long-term research programme together with the Institute for Technical Informatics of Graz University of Technology. There, he currently leads the Industrial Informatics and Model-based Architectures group. His research interests include systems and software engineering, software technology, and process improvement.

Prof. Dr. Eugen Brenner is Associate Professor at the Institute for Technical Informatics of the Graz University of Technology. He completed his master in Electrical Engineering 1983 in Graz. His PhD in Control Theory was finished 1987 also in Graz, dealing with optimal control in systems with limited actuating variables. He joined the institute in 1987, being the first scientific staff member. His post-doctoral lecture qualification in Process Automation was achieved in 1996.He has been member of the senate, of the curricula commission for Bachelor and Master-Programs, and Dean of Studies for Telematics. He currently is head of the Study Commission and Vice-Dean of Studies for Telematics. Eugen Brenner's primary research interests developed from FPGA-based hardware extension to parallel systems, real-time systems and process control systems. The most recent focus targeting embedded systems is on modelling, software-development, systems engineering and systems security, including agile programming methods and smart service engineering.

A Model-to-Model Transformation Approach at Mechatronics-Based E/E-System Design

Harald Sporer, Georg Macher, Christian Kreiner, and Eugen Brenner
Institute of Technical Informatics
Graz University of Technology
Inffeldgasse 16/1, 8010 Graz, Austria
Email: {sporer, georg.macher, christian.kreiner, brenner}@tugraz.at

1. Introduction

High quality standards along the whole E/E-System development are crucial to cope with the growing complexity at modern passenger cars in the upcoming years.To achieve this, methods and techniques from concepts like Automotive SPICE [1] are strongly recommended. Some of the key aspects of these concepts are bilateral traceability, as well as consistency between the different development artefacts, like the designs at the various abstraction levels.

In the automotive industry, the E/E-System design models are usually created with techniques based on the *Unified Modeling Language (UML)*. To enable this de facto standard for the embedded automotive system design, either the meta-model is extended or a profile is created. A wide-spread example of an UML2 profile is the *Systems Modeling Language (SysML)*.

As a finding of recent E/E-System development projects in the automotive field, the authors recognized that in some scenarios a SysML approach is not the best suitable one, for creating a system design. This may be the case, if a system design with focus on mechanical components should be created, or if an automotive domain expert with limited skills in SysML modeling is supposed to prepare a rough architectural design. To improve the development process in such scenarios, a domain specific modeling (DSM) language for embedded mechatronics-based automotive systems was defined.

In this paper a methodology is presented that incorporates the DSM and the SysML system design environment. By the implementation of the model-to-model transformation adapter, the advantages of both techniques can be combined.

2. Related Work

In recent years, a lot of effort has been made to improve the automotive model-based E/E-System design methods and techniques. Traceability, as well as consistency, between the development artefacts has always been an important topic. If it comes to safety-critical functionalities, traceability between the relevant artefacts is mandatory, according to the 2011 released international standard *ISO 26262* [2]. Moreover, here the benefits of a model-based development compared to a code-based development are described.

The common deliverables along an automotive E/E-System development, and a corresponding process reference model are presented by the de facto standard *Automotive SPICE* [1]. Neither the functional safety standard nor the process reference model enforces a specific methodology, how the development artefacts have to be linked to each other. However, connecting the various work products manually is a tedious and error-prone task, and has to be done in an automatic manner. Therefore an adequate seamless tool support, like presented by the authors in [3], [4], and [5], is needed. As in other contributions in this field (e.g. [6]), SysML is utilized for the system architectural design.

To agree with Broy et al. [7], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities of how to customize the UML diagrams, to get a language for embedded system design, drive these drawbacks.

According to Mernik et al. [8], a domain specific language (DSL) gains the expressiveness and ease of use, compared to general-purpose languages. To achieve this enhancements in the field of embedded automotive mechatronics-based system designs, the authors defined a domain specific modeling language, which is presented in [9].

3. Approach

The goal of this work is to transfer the relevant information from the embedded mechatronics system model (DSM representation) to the SysML model in a fully automatic manner, as outlined in Figure 1. The whole DSM model data is stored in an extensible markup language (XML) file. Therefore, not only the various components like *Control Unit* and *Application Software Unit* are retained, also the interface signals between the model elements are available by parsing the particular file. For creating the SysML system design model, the wide-spread software tool *Enterprise Architect*[1] *(EA)* is used for this approach. The model data are stored in a proprietary repository object and can be accessed through the Enterprise Architects Automation Interface.

The adapter's functionality is integrated into the DSM development environment *EASy Design*. The routines are implemented in C# and can be ported to other software tools easily. By utilizing the EA interfaces dynamic library, the class *Repository* is available and provides the necessary

1. http://sparxsystems.com.au/

methods and properties in order to gain full access to the new SysML model.

Due to the definition of the domain specific language, an arbitrary number of control units can be implemented in the system model. Therefore, the DSM model can also depict system of systems (SoS) instead of a single system. The approach at the adapters first version is to transform only one (sub)system per generation cycle. By selecting the particular control unit and executing the command *Generate SysML Model*, the transformation process is enabled for the respective system.

Once the transformation process is triggered, the DSM model at *EASy Design* is analysed and the relevant elements are created in the new SysML model:

- Usually, the SysML model consists of different abstraction levels like *Vehicle*, *System*, etc. Therefore, the engineer executing the transformation is prompted for selecting the proper category for the new model.
- For each signal at the selected control units interface, an object of the class *ConnectorPin* is created and added to the *HSI Structure* view at the SysML model. If this hardware signal is connected to a *Basis Software Component* at the DSM model, an object of type *DataSignal* is added at the *HSI Structure* view and linked to the corresponding *ConnectorPin* instance.
- All software items from the DSM model (*Basis Software Components* & *Application Software Components*), which belong to the selected control unit component, are created at the SysML *SW Components* view as objects of type *AUTOSARComponent*.
- According to the *Timing* property of the DSM model software component, an *AUTOSARComponent* with trigger characteristic is connected to the software module at the SysML model.
- All signals between the software components at the DSM model, are created as objects of type *AUTOSARPort* at the particular module and linked by *AUTOSARConnector* items.

With the steps described above, the DSM model is transformed into the SysML representation. Of course, numerous properties, like *Name*, *ASIL*, and *Data Type*, are set automatically while creating the SysML objects. However, to describe the transformation of each property would go beyond the scope of this paper.

4. Conclusions

The described model-to-model transformation approach combines the advantages of the lean domain specific modeling technique, which is best suitable for a rough system design by automotive domain experts with limited SysML skills, and the established SysML system modeling methodology. Although the adapter is currently in a first trial phase for industrial project applicability, evidences of the approaches benefits are already present. Moreover, important topics for future work, like the transformation of the SysML

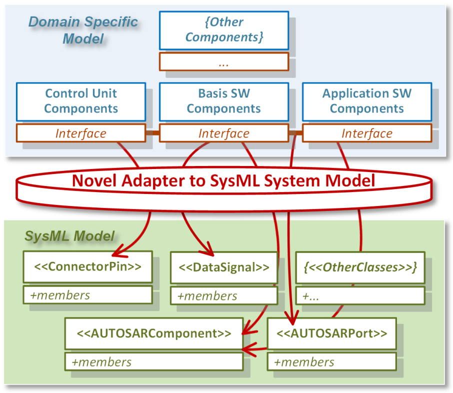


Figure 1. Incorporation of DSM & SysML design

model into the DSM representation (also for consistency check features), and the transformation of a system of systems model in one step, have been identified.

References

[1] Automotive SIG, “Automotive SPICE®Process Assessment Model,” The SPICE User Group, Tech. Rep., May 2010, version 2.5.

[2] “ISO 26262, Road vehicles - Functional safety,” International Organization for Standardization, Geneva, CH, International Standard, November 2011.

[3] G. Macher, E. Armengaud, and C. Kreiner, “Automated Generation of AUTOSAR Description File for Safety-Critical Software Architectures,” in *12. Workshop Automotive Software Engineering (ASE)*, ser. Lecture Notes in Informatics, 2014, pp. 2145–2156.

[4] G. Macher, H. Sporer, E. Armengaud, and C. Kreiner, “A Versatile Approach for ISO26262 compliant Hardware-Software Interface Definition with Model-based Development,” in *SAE Technical Paper*. SAE International, 2015.

[5] G. Macher, M. Atas, E. Armengaud, and C. Kreiner, “Automotive Real-time Operating Systems: A Model-Based Configuration Approach,” in *ACM SIGBED Review Special Interest Group on Embedded Systems*, ser. CEUR Workshop Proceedings, vol. 1291, 2014.

[6] R. Boldt, “Modeling AUTOSAR systems with a UML/SysML profile,” IBM Software Group, Tech. Rep., July 2009.

[7] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, “Seamless model-based development: From isolated tools to integrated model engineering environments,” *Proceedings of the IEEE*, vol. 98, no. 4, pp. 526–545, 2010.

[8] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.

[9] H. Sporer, G. Macher, C. Kreiner, and E. Brenner, “A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design,” in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, under review.

2015 41st Euromicro Conference on Software Engineering and Advanced Applications

Incorporation of Model-based System and Software Development Environments

Harald Sporer*, Georg Macher*†, Eric Armengaud† and Christian Kreiner*

*Institute of Technical Informatics, Graz University of Technology, AUSTRIA
Email: {sporer, georg.macher, christian.kreiner}@tugraz.at

†AVL List GmbH, Graz, AUSTRIA
Email: {georg.macher, eric.armengaud}@avl.com

***Abstract*—Development of dependable embedded automotive systems faces many challenges arising from increasing complexity, criticality, and demand of certifiability. Efficient and consistent development models along the entire development life cycle needs to be ensured. So far, existing solutions are still frequently insufficient when transforming system models with higher level of abstraction to more concrete engineering models (such as software engineering models). De facto industry standards aims to standardize frameworks and to facilitate the exchange of information. However, refinement of system designs into hardware and software implementations is still a tedious task. The aim of this work is to enhance an automotive model-driven system engineering framework with software architecture design capabilities and a model transformation framework to enable a seamless description of safety-critical systems, from requirements at the system level down to software component implementation in a bidirectional way.**

***Keywords*-ISO 26262, automotive, Model-based development, real-time systems, reuse, traceability, model-based software engineering.**

I. Introduction

The trend of replacing traditional mechanical systems with modern embedded systems enables the deployment of more advanced control strategies, which provide additional benefits for the customer and environment, but at the same time the higher degree of integration and criticality of the control application raise new challenges. To handle upcoming issues with modern real-time systems, also in relation to ISO 26262, model-based development is utilized to support the description of the system under development in a more structured way. Model-based development approaches enable different views for different stakeholders, different levels of abstraction, and central storage of information. This improves the consistency, correctness, and completeness of the system specification and thus supports the demands of time-to-market. Nevertheless, such seamless integrations of model-based development are rather exception than the rule and often fall short due to the lack of integration of conceptual and tooling levels [2].

The aim of this paper is to bridge the existing gap between model-driven system engineering tools and software engineering tools. More specifically, the approach is based on the enhancement of an model-driven system-engineering framework with software-architecture design capabilities. Furthermore, a model transformation framework enables a seamless description of safety-critical software, from requirements at the system level down to software component implementation, in a bidirectional way. The model transformation framework automatically generates software architectures in Matlab/Simulink®, described via high level control system models in SysML format. The main goals of this contribution are: (a) to support a consistent and traceable refinement from the early concept phase to software implementation, and (b) to establish the bidirectional update function of the transformation framework, gaining mutual benefits for basic software and application software development from the coexistence of both information within the central database.

II. Related Works

Broy et al. [2] mention concepts and theories for model-based development of embedded software systems. The authors also claim model-based development the best approach to manage the large amount of information and complexity of modern embedded systems with safety constraints. The paper illustrates why seamless solutions have not been achieved so far, they mention commonly used solutions, and arising problems by using an inadequate tool chain (e.g. redundancy, inconsistency and lack of automation).

Holtmann et al. [5] claim the lack of automation for those linking tasks and missing guidance, which model should to be used at which specific development stage, as crucial drawback of model-driven development (MDD). The very specific and non-interacting tools requiring manual synchronization, are often inconsistent or rely on redundant information.

An issue is also addressed by Giese et al. [4]. System design models have to be correctly transferred to the software engineering model, and later changes must be kept consistent. The authors propose a model synchronization approach consisting of tool adapters between SysML models and software engineering models in AUTOSAR representation. A drawback of this approach, each transformation step implies potential sources for ambiguous mapping and model mismatching.

Kawahara et al. [7] propose an extension of SysML which enables description of continuous time behaviour. Their tool integration is based on Eclipse and couples SysML and Matlab/Simulink via API.

An automotive tool-chain for AUTOSAR is also presented by Voget [12]. The work focuses on ARTOP, a common platform for innovations which provides common base functionality for development of AUTOSAR compliant tools. Unfortunately the Eclipse based ARTOP platform serving only as a common base for AUTOSAR tool development, is not a tool solution, and also requires time-consuming initial training to even get started to develop a desired tool.

The approach of bridging the gap between model-based system engineering and software engineering models based on EAST-ADL2 architecture description language and a complementary AUTOSAR representation is also very common in the automotive software development domain [3], [9], [11]. EAST-ADL represents an architecture description language using AUTOSAR elements to represent the software implementation layer of embedded systems [1]. More recently the MAENAD Project[1] is also focusing on this approach.

Tool support for automotive engineering development is still organized as a patchwork of heterogeneous tools and formalisms [1]. On the one hand, general-purpose modeling languages (such as UML or SysML) provide modeling power suitable to capture system wide constraints and behaviour, but lack in synthesizability. On the other hand, special-purpose modeling environments (such as Matlab/Simulink and ASCET) are optimized for fine granular design and being less efficient in high-level design.

The approach presented in this work and the work of Mader et al. [9] base on similar concepts, but in contrast to their work, our contribution support automatic generation of whole software architectures, interface definition, timing setting, and auto-routing of signals in addition to their automatic generation of a software model. For a more detailed overview of the whole tool chain orchestration see [8].

III. Model-Transformation Approach Bridge

The basic concept behind this work is to have a consistent information repository as central source of information, to store all information of all involved engineering disciplines of embedded automotive system development in a structured way. The concept focus on allowing different engineers to do their job in their specific manner, but providing traces and dependency analysis of features concerning the overall system, e.g. safety, security, or dependability. Furthermore, the proposed approach is intended as an affordable, versatile, and tool independent method. This makes the method especially attractive for small and micro-sized companies or small projects with limited resources.

The proposed contribution is part of the framework presented in [8], towards software development in the automotive context. More specifically, our contribution consists of the following parts:

- *UML software modeling framework*: Enhancement of an UML profile for the definition of software development artefacts, more precisely, for the definition of the components interfaces and SW architecture composition. Required for consistent SW system description, see Figure 1 – model addon.
- *SW architecture exporter*: Exporter to generate the designed SW architecture in the third party tool Matlab/Simulink for further detailed development, see Figure 1 – tool bridge.
- *SW architecture importer*: Importer to integrate refined SW architecture and interfaces from the software development tool (e.g. as a result of round-trip engineering), see Figure 1 – tool bridge.

This approach closes the gap, also mentioned by Giese et al. [4], Holtmann et al. [5], and Sandmann and Seibt [10], between system-level development at abstract UML-like representations and software-level development modeling tools (e.g. Matlab/Simulink). The bridging supports consistency of information transfer between system engineering tools and software engineering tools and minimizes redundant manual information exchange between these tools. This contributes to simplify seamless safety argumentation according to ISO 26262 [6] for the developed system. Benefits of this development approach are highly noticeable in terms of re-engineering cycles, tool changes, and reworking of development artefacts with alternating dependencies. As can be seen in Figure 1, the lack of supporting tools for information transfer between system development tools and software development tools can be dispelled by our approach. The implementation of the bridge, based on versatile C# class libraries (dll) and Matlab COM Automation Server, ensures tool independence of the general-purpose UML modeling tool (such as Enterprise Architect or Artisan Studio) and version independence of Matlab/Simulink through API command implementation.

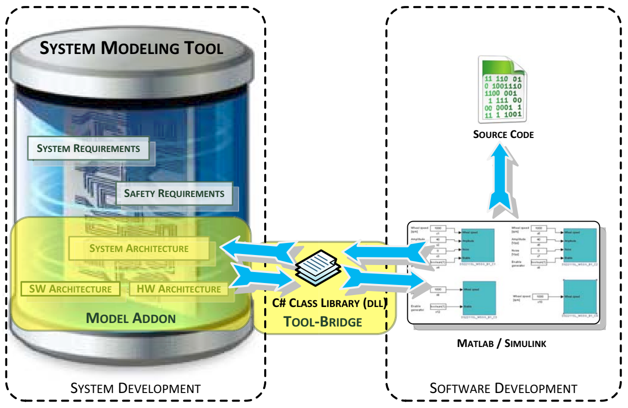


Figure 1. Portrayal of the Bridging Approach Transferring System Development Artefacts to SW Development Phase

[1]http://maenad.eu/

The first part of the approach is the development of a specific **UML modeling framework** within a state-of-the-art system development tool (in this case Enterprise Architect). This EA profile makes the UML representation more manageable for the needs of the design of an automotive software architecture by taking advantage of an AUTOSAR aligned abstraction layer. Furthermore, the profile enables an explicit definition of components, component interfaces, and connections between interfaces. This provides the possibility to define software architecture and ensures proper definition of the communication between the architecture artefacts, including interface specifications. Hence, the SW architecture representation within EA can be linked to system development artefacts, and traces to requirements can be easily established. This further benefits in terms of constraints checking, traceability of development decisions (e.g. for safety case generation), and reuse.

The second part of the approach is an **software architecture exporter**, which is able to export the SW architectural design, component containers, and their interconnections specified in SysML, to the software development tool Matlab/Simulink. The implementation of the exporter is based on Matlab COM Automation Server and generates models through API command implementation. Per user input, the software architecture representation to be transferred is selected and a background task generates a corresponding Matlab/Simulink model. Each model artefact, parameter, and connection is transferred to Matlab/Simulink, blocks are arranged and sized in correct manner, and also unique links to the EA representation and assigned safety-criticality of the artefact are established.

The last part of the approach is the **software architecture import** functionality add-on for the system development tool. This functionality, in combination with the export function, enables bidirectional update of software architecture representation in the system development tool and the software modules in Matlab/Simulink. The importer identifies the unique links to the EA representation and thereby differentiate new and modified model artefacts.

Triggered via user input, an user interface within the system development tool depicts modifications between the two representations (classified as *added*, *deleted*, or *updated*), and enables selective update of the UML based software architecture representation.

On the one hand, this ensures consistency between system development artefacts and changes done in the software development tool. On the other hand, the import functionality enables reuse of available software modules, guarantees consistency of information across tool boundaries, and shares information more precisely and less ambiguously.

IV. Application of the Proposed Approach

This section illustrates the utilization of the framework by applying the methodology to the 3-layer monitoring concept [13]. This use-case is an illustrative material, reduced for internal training purpose of both, students and engineers.

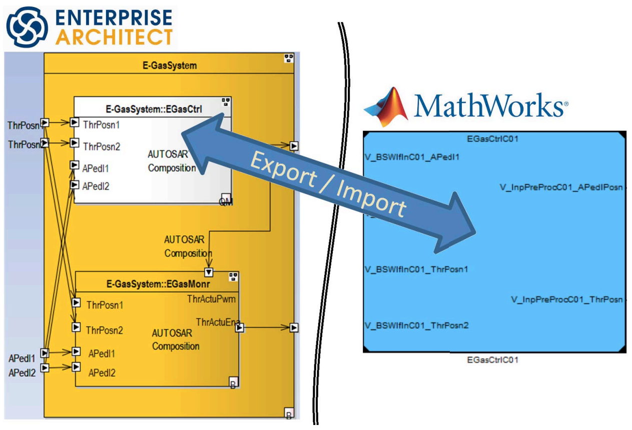


Figure 2. System and Software Model Use-Case Excerpt

Therefore, the disclosed and commercially non-sensitivity use-case is not intended to be exhaustive or representing leading-edge technology.

The definition of the software architecture is usually done by a software architect within the software development tool Matlab/Simulink. With our approach this work package is transferred to the system development tool. This does not hamper the work of the software architect, but enables constraint checking features and helps to improve system maturity in terms of consistency, completeness, and correctness of the development artefacts. Beside this, the change offers a significant benefit for development of safety-critical software in terms of traceability, replicability of design decisions, visualizes dependencies unambiguously, and puts visual emphasis on view-dependent constraints (such as graphical safety-criticality highlighting of SW modules in Figure 2).

The software architectural design of the use-case consists of 41 model artefacts with 361 configuration parameters, and 30 relations between the elements. This small example already indicates that relations between the model elements and number of model elements become confusing. Therefore, manual transformation of the information represented within the models is cumbersome and error-prone and would inherit lots of additional work to ensure consistency of both models in terms of safety-critical software development.

With our approach these information and model artefacts are checked for consistency constraints (such as point-to-point consistency of interface configurations) before automatically transferred via 212 lines of auto-generated Matlab API code. This auto-generation of Matlab API code provides evidence and ensures completeness of the model transformation. Furthermore, the SW import functionality enables round-trip engineering and bi-directional updates, and supports evidence for consistency of both models.

According to the presented bridge approach in Section III, the first step during the transformation is the decomposition of the software architectural design. Each software subsystem (like the Composition *EGasCtrl* in Figure 2) is analysed and the comprised software modules are extracted. An essential information at the software module architecture is the trigger definition, representing the later task timing property of the module at the integrated system.

With the gathered architectural, timing, and interface information the Simulink model is generated by the previously described utilization of the Matlab COM Automation Server. During this process, a new Simulink root model is created and for each trigger type, which appears at the architectural design, an own subsystem is placed at the models top level. Afterwards, each software subsystem is transferred to the appropriate timing subsystem or even split up into multiple tasks if necessary. E.g. the subsystem *E-GasSystem::EGasCtrl* contains software modules with different timing attributes. Therefore, the *EGasCtrl*-Subsystem is available at multiple timing subsystems at the Simulink model. To facilitate a multi developer scenario, a separate model file is created for each software subsystem and linked as a reference model at the root model. To complete the Simulink model subsystem generation, all software modules are transferred into their appropriated software subsystem.

The transformation process described so far, generates a software framework out of the *Composition* and *Application* blocks at the design. To provide a complete model framework, which serves as a basis for the subsequent software unit development, the interfaces as well as their connections are transferred to Simulink in the next step. For an efficient and dependable handling of the signals, a *Data Dictionary* and the related tool *Data Dictionary Manager* is used. Every signal from the architectural design is stored at the data dictionary, including all available attributes like value limits, scaling, etc. Moreover, the exporter algorithm simply link this entry wherever the signal occurs in the Simulink model.

If the Simulink model already exists when the *Software Architecture Exporter* is triggered, the *Software Architecture Importer* is started automatically in the background to check the consistency between the architectural design and the software model. If all artefacts at the model are available at the design, and new items exist at the design, the software model is updated by the exporter analogue to the procedure described above for completely new Simulink models. If there are new or deleted artefacts at the software model, a notification is displayed and the user is prompted to determine the further procedure.

V. Conclusion

This paper presented an approach to avoid the risk of introducing errors while developing the software according to the architectural design, by creating the software model framework fully automated. Moreover, the concept facilitates bidirectional traceability, as well as consistency. These properties are elemental key factors for a high quality development and postulated by the widespread quasi standard *Automotive SPICE*. Additionally, the shown techniques facilitates round-trip engineering by the presented import/export functionality regarding the models on different development levels and tools.

The application of the presented approach has been demonstrated utilizing a simplified version of the well-known E-Gas concept, which is intended to be used for training purpose of students and engineers and not representing an exhaustive or commercial sensitive project.

References

[1] H. Blom, H. Lönn, F. Hagl, Y. Papadopoulos, M. O. Reiser, C. J. Sjöstedt, J. Chen, and R. T. Kolagari. EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems. White paper, 2013.

[2] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526–545, 2010.

[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 *SAFECOMP 2008*, pages 72–85, 2008.

[4] H. Giese, S. Hildebrandt, and S. Neumann. Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent. *LNCS 5765*, pages 555 –579, 2010.

[5] J. Holtmann, J. Meyer, and M. Meyer. A Seamless Model-Based Development Process for Automotive Systems. In *Software Engineering (Workshops)*, pages 79–88, 2011.

[6] ISO 26262, Road vehicles - Functional safety. International standard, International Organization for Standardization, Geneva, CH, November 2011.

[7] R. Kawahara, H. Nakamura, D. Dotan, A. Kirshin, T. Sakairi, S. Hirose, K. Ono, and H. Ishikawa. Verification of embedded system's specification using collaborative simulation of SysML and simulink models. In *MBSE'09*, pages 21–28, 2009.

[8] G. Macher, E. Armengaud, and C. Kreiner. Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain. In *ERTS'14*, pages 256–263, 2014.

[9] R. Mader, G. Griessnig, E. Armengaud, A. Leitner, C. Kreiner, Q. Bourrouilh, C. Steger, and R. Weiss. A Bridge from System to Software Development for Safety-Critical Automotive Embedded Systems. *SEAA'12*, pages 75–79, 2012.

[10] G. Sandmann and M. Seibt. AUTOSAR-Compliant Development Workflows: From Architecture to Implementation-Tool Interoperability for Round-Trip Engineering and Verification and Validation. Technical report, SAE, 2012.

[11] C.-J. Sjöstedt, J. Shi, M. Törngren, D. Servat, D. Chen, V. Ahlsten, and H. Lönn. Mapping Simulink to UML in the Design of Embedded Systems: Investigating Scenarios and Structural and Behavioral Mapping. In *OMER 4 Post Workshop Proceedings*, April 2008.

[12] S. Voget. SAFE RTP: An open source reference tool platform for the safety modeling and analysis. In *ERTS'14*, 2014.

[13] T. Zurawka and J. Schaeuffele. Method for checking the safety and reliability of a software-based electronic system, January 2007.

Bidirectional Crosslinking of System and Software Modeling in the Automotive Domain

Harald Sporer( ), Georg Macher, Andrea Höller, and Christian Kreiner

Institute of Technical Informatics, Graz University of Technology,
Inffeldgasse 16/1, 8010 Graz, Austria
{sporer,georg.macher,andrea.hoeller,christian.kreiner}@tugraz.at
http://www.iti.tugraz.at/

Abstract. Replacing former pure mechanical functionalities by mechatronics-based solutions, introducing new propulsion technologies, and connecting cars to their environment are only a few reasons for the still growing E/E-System complexity at modern passenger cars. Hence, for an engineering company in the automotive embedded system domain it is vital to establish mature development processes, including a smart tool chain orchestration. Starting from the customer requirements until the final release of the product, traceability and consistency between all development artifacts shall be given. However, achieving this by linking the development items manually is a tedious and error-prone task. The aim of this work is to enhance the development process by introducing a fully automatic transformation of a system design model into a software framework model and vice versa. With this novel approach, the full traceability, between the system and software architectural levels, is guaranteed.

Keywords: Automotive · Model-based development · Embedded systems · Traceability · Model-based software engineering

1 Introduction

Embedded systems are already integrated into our everyday life and play a central role in all domains including automotive, aerospace, healthcare, industry, energy, or consumer electronics. In 2010, the embedded systems market accounted for almost 852 billion dollar, and is expected to reach 1.5 trillion by 2015 (assuming an annual growth rate of 12 %) [17]. Current premium cars implement more than 90 electronic control units (ECU) with close to 1 Gigabyte software code [6], are responsible for 25 % of vehicle costs and an added value between 40 % to 75 % [22].

The trend of replacing traditional mechanical systems with modern embedded systems enables the deployment of more advanced control strategies providing additional benefits for the customer and environment, but at the same

A. Fantechi and P. Patrizio (Eds.): SERENE 2015, LNCS 9274, pp. 99–113, 2015.
DOI: 10.1007/978-3-319-23129-7_8

time, the higher degree of integration and criticality of the control application raise new challenges. To cope with this situation, smart methods and techniques have to be applied starting from the very beginning of a systems development. Some kind of guidelines, like the functional safety standard for E/E-Systems at modern passenger cars ISO 26262, has been introduced in recent years.

To handle upcoming issues with modern real-time systems, also in relation to ISO 26262, model-based development supports the description of the system under development in a more structured way. Model-based development approaches enable different views for different stakeholders, different levels of abstraction, and central storage of information. This improves the consistency, correctness, and completeness of the system specification and thus supports the demands of time-to-market (first time right). Nevertheless, such seamless integration of model-based development are rather exception than the rule and often fall short due to the lack of integration of conceptual levels and tooling levels [3].

The aim of this paper is to bridge the existing gap between model-driven system engineering tools and software engineering tools. More specifically, the approach is based on the enhancement of a model-driven system-engineering framework with software-architecture design capabilities. Furthermore, a model-transformation framework enables a seamless description of safety-critical software, from requirements at the system level down to software component implementation in a bidirectional way. The model-transformation framework automatically generates software architectures in Matlab/Simulink described via high level control system models in SysML format. The goal is, on one hand, to support a consistent and traceable refinement from the early concept phase to software implementation. On the other hand, the bidirectional update function of the transformation framework enables facilitation of gaining mutual benefits for basic software and application software development from the coexistence of both information within the central database.

The document is organized as follows: In the course of this paper, Sect. 2 presents an overview of related approaches as well as model-based development and integrated tool chains. In Sect. 3 a description of the proposed bridging approach for the refinement of the model-based system engineering model to software development is provided. An application and evaluation of the approach is presented in Sect. 4. Finally, this work is concluded in Sect. 5 with an overview of the presented approach.

2 Related Works

Model-based systems and software development, as well as tool integration are engineering domains and research topics aimed at moving the development steps closer together and thus improving the consistency of the system over the expertise and domain boundaries. In Pretschner's roadmap [18], the authors highlight the benefits of a seamless model-based development tool-chain for automotive software engineering.

Broy et al. [3] mention concepts and theories for model-based development of embedded software systems. The authors claim model-based development the best approach to manage the large amount of information and complexity of modern embedded systems with safety constraints. The paper illustrates why seamless solutions have not been achieved so far, they mention commonly used solutions, and arising problems by using an inadequate tool-chain (e.g. redundancy, inconsistency and lack of automation).

Nevertheless, the challenge of enabling a seamless integration of models into model-chains is still an open issue [19,20,24]. Often, different specialized models for specific aspects are used at different development stages with varying abstraction levels. Traceability between these different models is commonly established via manual linking due to process and tooling gaps. Holtmann et al. [8] claim this lack of automation for those linking tasks and missing guidance which model should to be used at which specific development stage as crucial drawback of model-driven development (MDD). The very specific and non-interacting tools requiring manual synchronization, are often inconsistent or rely on redundant information.

An issue is also addressed by Giese et al. [7]. System design models have to be correctly transferred to the software engineering model, and later changes must be kept consistent. The authors propose a model synchronization approach consisting of tool adapters between SysML models and software engineering models in AUTOSAR representation. A drawback of this approach, each transformation step implies potential sources for ambiguous mapping and model mismatching.

An important topic to deal with is the gap between system architecture and software architecture - especially while considering component-based approaches such as UML and SysML for system architecture description and AUTOSAR for SW architecture description. Using SysML [2,7,10,12,15] or X-MAN [11] for architectural description and AUTOSAR for software system description are two common variants in the automotive domain. Buchmann et al. [4] present an approach of another domain, based on bi-directional and incremental transformation between XML class diagrams and Java source code. The authors also highlight the crucial importance of powerful tool support for model-driven software engineering. Boldt [2] proposed the use of a tailored Unified Modeling Language (UML) or System Modeling Language (SysML) profile as the most powerful and extensible way to integrate an AUTOSAR method in company process flows.

An automotive tool-chain for AUTOSAR is also presented by Voget [25]. The work focuses on ARTOP, a common platform for innovations which provides common base functionality for development of AUTOSAR compliant tools. Unfortunately, the Eclipse-based ARTOP platform serves only as a common base for AUTOSAR tool development and is not a ready-to-use tool-solution. Moreover, ARTOP also requires time-consuming initial training before a tool can be developed.

The approach of bridging the gap between model-based system engineering and software engineering models based on EAST-ADL2 architecture description language and a complementary AUTOSAR representation is also very common in the automotive software development domain [5,14,23]. EAST-ADL represents an architecture description language using AUTOSAR elements to represent the software implementation layer of embedded systems [1]. More recently the MAENAD Project[1] is also focusing on this approach.

Pagel et al. [16] mention the benefit of generating XML schema files directly from a platform-independent model (PIM) for data exchange via different tools. Performing extra transformation steps would only add potential sources for error and ambiguous mappings could result in unwanted side-effects.

Kawahara et al. [10] propose an extension of SysML which enables description of continuous time behavior. Their tool integration is based on Eclipse and couples SysML and Matlab/Simulink® via API .

Tool support for automotive engineering development is still organized as a patchwork of heterogeneous tools and formalisms [1]. On the one hand, general-purpose modeling languages (such as UML or SysML) provide modeling power suitable to capture system wide constraints and behavior, but lack in synthesizability. On the other hand, special-purpose modeling languages (such as C, Assembler, Matlab, Simulink, ASCET) are optimized for fine granular design and being less efficient in high-level design.

2.1 The Underlying Framework of the Proposed Approach

This section gives a brief overview of the underlying framework and related preliminary work which supports the proposed approach. The basic concept behind this framework is to have a consistent information repository as a central source of information, to store all information of all involved engineering disciplines of embedded automotive system development in a structured way. The concept focuses on allowing different engineers to do their job in their specific manner, but providing traces and dependency analysis of features concerning the overall system, e.g. safety, security, or dependability. Furthermore, the proposed approach is intended as an affordable, versatile, and tool independent method. This makes the method especially attractive for limited resources of small and micro-sized companies or small projects. Especially such projects or start-up companies often struggle with setting up their development processes or achieving adequate quality with limited resources (such as time or manpower). Therefore this approach stir out of common AUTOSAR based approaches and force a direct model transformation from SysML representation to Matlab/Simulink. The reason to make the decision of not fostering an AUTOSAR approach is based on one hand on focusing not only on AUTOSAR but rather generally on Matlab/Simulink based automotive software development. On the other hand, experiences we made with our previous approach [12] confirm the problem mentioned by Rodriguez et al. [20]. Not all tools fully support the whole AUTOSAR standard, because of its complexity, which leads to several mutual incompatibilities and interoperability problems.

The approach presented in this work and the work of Mader et al. [14] are based on similar concepts, but in contrast to their work, our technique supports automatic generation of whole software architectures, interface definition, timing

[1] http://maenad.eu/.

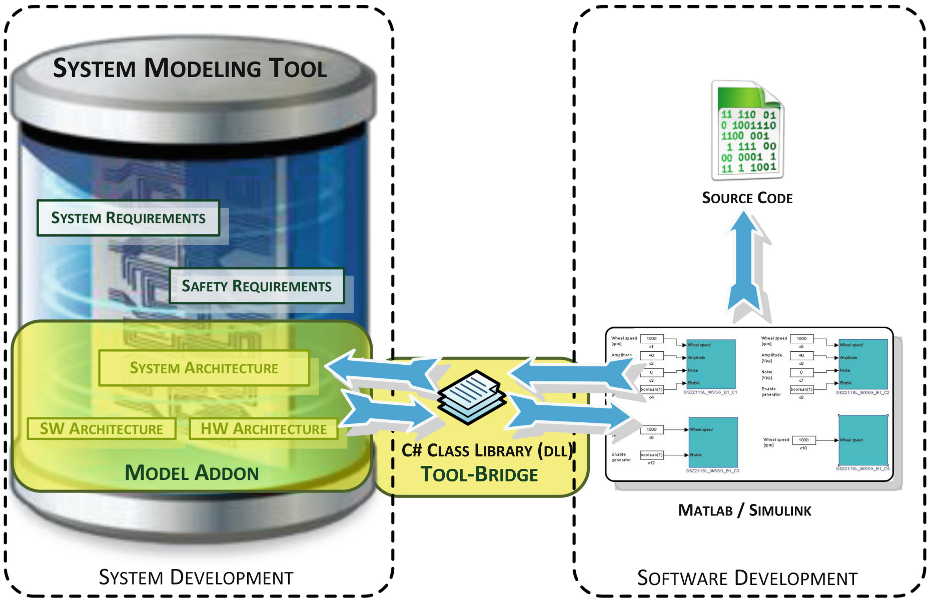


Fig. 1. Portrayal of the bridging approach transferring system development artefacts to SW development phase

setting, and auto-routing of signals in addition to their automatic generation of a software model. Figure 1 shows an overview of this approach and the embedded bridging of abstract system development and concrete software development models. For a more detailed overview of orchestration of the whole tool-chain see [13].

3 Model-Transformation Bridge

As mentioned in the previous section, the fundamental concept behind this framework is to have a consistent information repository as central source of information, to store all information of all involved engineering disciplines of embedded automotive system development in a structured way. This affordable, versatile, and tool independent approach forces a direct model transformation from SysML representation to Matlab/Simulink and is especially attractive for software development without full orchestration of AUTOSAR toolchain or non-AUTOSAR based development.

The contribution proposed in this work is part of the framework presented in [13] towards software development in the automotive context. More specifically, our contribution consists of the following parts:

– *UML Software Modeling Framework*: Enhancement of an UML profile for the definition of software development artifacts, more precisely, for the definition of the components interfaces and SW architecture composition. Required for consistent SW system description, see Fig. 1 – model addon.
– *SW Architecture Exporter*: Exporter to generate the designed SW architecture in the third party tool Matlab/Simulink for further detailed development, see Fig. 1 – tool bridge.

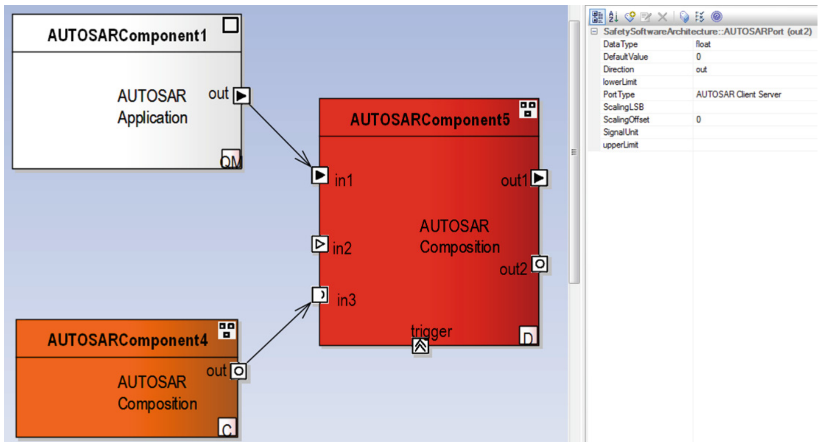
– *SW Architecture Importer*: Importer to integrate refined SW architecture and interfaces from the software development tool (e.g., as a result of round-trip engineering), see Fig. 1 – tool bridge.

This proposed approach closes the gap, also mentioned by Giese et al. [7], Holtmann et al. [8], and Sandmann and Seibt [21], between system-level development at abstract UML-like representations and software-level development modeling tools (e.g. Matlab/Simulink or Targetlink®). The bridging supports consistency of information transfer between system engineering tools and software engineering tools and minimizes redundant manual information exchange between these tools. This contributes to simplify seamless safety argumentation according to ISO 26262 [9] for the developed system. Benefits of this development approach are highly noticeable in terms of more efficient re-engineering cycles, and easy reuse of development artifacts with changing dependencies. As can be seen in Fig. 1, the lack of supporting tools for information transfer between system development tools and software development tools can be dispelled by our approach. The implementation of the bridge, based on versatile C# class libraries (dll) and Matlab COM Automation Server, ensures tool independence of the general-purpose UML modeling tool (such as Enterprise Architect or Artisan Studio) and version independence of Matlab/Simulink through API command implementation. This makes the method especially attractive for projects and companies with limited resources (such as manpower or finances). Especially small projects or start-up companies often struggle with setting up their development processes to achieve adequate quality.

3.1 UML Software Modeling Framework

The first part of the approach is the development of a specific UML modeling framework enabling software architecture design in AUTOSAR like representation within a state-of-the-art system development tool (in this case Enterprise Architect). This EA profile makes the UML representation more manageable for the needs of the design of an automotive software architecture by taking advantage of an AUTOSAR aligned abstraction layer. Furthermore, the profile enables an explicit definition of components, component interfaces, and connections between interfaces. This provides the possibility to define software architecture and ensures proper definition of the communication between the architecture artifacts, including interface specifications (e.g. upper limits, initial values, formulas). In addition this profile ensures the versatilely to also enable AUTOSAR aligned development as proposed in [12].

Hence, the SW architecture representation within EA can be linked to system development artifacts and traces to requirements can be easily established. This further benefits in terms of constraints checking, traceability of development decisions (e.g. for safety case generation), and reuse. Figure 2 shows an example of software architecture artifacts and interface information represented in Enterprise Architect.



SafetySoftwareArchitecture::AUTOSARPort (out2)

DataType	float
DefaultValue	0
Direction	out
lowerLimit	
PortType	AUTOSAR Client Server
ScalingLSB	
ScalingOffset	0
SignalUnit	
upperLimit	

Fig. 2. Snapshot of the SW architecture representation within the system development tool and representation of the interface information

3.2 SW Architecture Exporter

The second part of the approach is an exporter which is able to export the software architecture, component containers, and their interconnections designed in SysML to the software development tool Matlab/Simulink. The implementation of the exporter is based on Matlab COM Automation Server and generates models through API command implementation. This ensures tool version-independence of the presented approach. Per user input the software architecture representation to be transferred is selected and a background task generates a corresponding Matlab/Simulink model. Listing 1.1 shows some excerpts of the automatically generated Matlab API commands. As can be seen in this listing, each model artifact, parameter, and connection is transferred to Matlab/Simulink, blocks are arranged and sized in correct manner and also unique links to the EA representation and assigned safety-criticality of the artifact (Listing 1.1 line 3 and 8) are established.

Listing 1.1. Excerpts of Matlab API commands

```
addpath(genpath('C:\EGasSystem'))
add_block('Simulink/Ports & Subsystems/Model','EGasSystem/EGasCtrl')
set_param('EGasSystem/EGasCtrl','ModelNameDialog','EGasCtrl',\
    ...'Description','EA_ObjectID@1969;ASIL@QM')
set_param('EGasSystem/EGasCtrl','Position',[250 50 550 250])
⋮
add_block('Simulink/Ports & Subsystems/In1','EGasSystem/APedl2')
set_param('EGasSystem/APedl2','Position',[50 200 80 215])
set_param('EGasSystem/APedl2','Outmin','0','Outmax','5',\
    ...'OutDataTypeStr','single','Description','EA_ObjectID@1966;\
    ...ASIL@B');
⋮
add_line('EGasSystem','APedl1/1','EGasMonr/1','AUTOROUTING','ON')
⋮
save_system('EGasSystem')
```

```
close_system('EGasSystem')
cd ..
cd C:\EGasSystem
```

If dSpace[2] tools are used for the subsequent C code generation instead of the Simulink Coder^{TM}, the software architecture can be exported into a TargetLink model optionally. In this case, the Matlab API command generator simply uses the *TargetLink common blockset* for creating the software framework model.

Furthermore, the signals from the software architecture design are analysed and transferred to the Simulink/TargetLink data dictionary by the exporter. This guarantees a consistent handling of the defined component interfaces as well as the connections between the interfaces throughout the development.

3.3 SW Architecture Importer

The last part of the approach is the import functionality add-on for the system development tool. This functionality, in combination with the export function, enables bidirectional update of software architecture representation in the system development tool and the software modules in Matlab/Simulink. The importer identifies the unique links to the EA representation (shown in Listing 1.1 line 3 and 8) and thereby differentiates new and modified model artifacts.

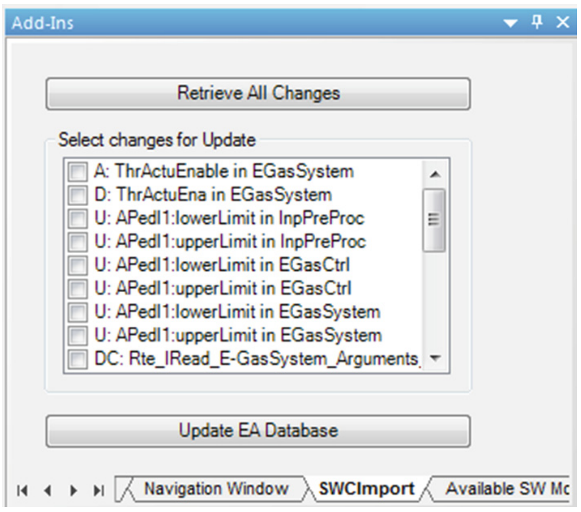
On the one hand, this ensures consistency between system development artifacts and changes done in the software development tool. On the other hand, the import functionality enables reuse of available software modules, guarantees consistency of information across tool boundaries, and shares information more precisely and less ambiguously.

Triggered via user input, a user interface within the system development tool (shown in Fig. 3) depicts modifications between the two representations and enables selective update of the UML based SW representation. As can be seen in Fig. 3, also a highlighting of the type of change (see Table 1) is provided.

Table 1. SW architecture importer indicators of type of change

Indicator	Type of change
A	Model artifact added
AC	Interface connection added
D	Model artifact deleted
DC	Interface connection deleted
U	Model artifact updated
UC	Interface connection updated

[2] http://www.dspace.com/.



A: ThrActuEnable in EGasSystem
D: ThrActuEna in EGasSystem
U: APedl1:lowerLimit in InpPreProc
U: APedl1:upperLimit in InpPreProc
U: APedl1:lowerLimit in EGasCtrl
U: APedl1:upperLimit in EGasCtrl
U: APedl1:lowerLimit in EGasSystem
U: APedl1:upperLimit in EGasSystem
DC: Rte_IRead_E-GasSystem_Arguments

Fig. 3. SW architecture importer user interface

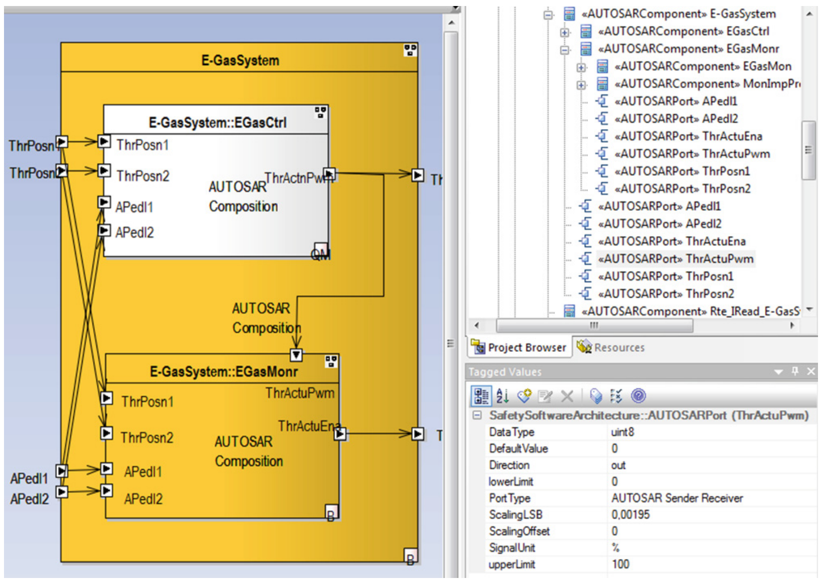
4 Application of the Proposed Approach

This section demonstrates the benefits of the introduced approach for the development of automotive embedded systems. To provide a comparison of the improvements of our approach, we use the 3 layer monitoring concept [26] as evaluation use-case. This elementary use-case is well-known in the automotive domain and because of this reason representative. This use-case is an illustrative material, reduced for internal training purpose of both, students and engineers. Therefore, the disclosed and commercially non-sensitive use-case is not intended to be exhaustive or representing leading-edge technology. An overview of the use-case is given in Table 2.

Table 2. Overview of the evaluation use-case SW architecture

Object type	Element-count	Configurable attributes per element
SW modules	7	3
SW interfaces	34	10
Connections	30	0

The definition of the software architecture is usually done by a software architect within the software development tool (Matlab/Simulink). With our approach, this work package is included in the system development tool (depicted in Fig. 4). This does not hamper the work of the software system architect but enables constraint checking features and helps to improve system maturity in terms of consistency, completeness, and correctness of the development artifacts. Beside this, the change offers a significant benefit for development of safety-critical software in terms of traceability, replicability of design decisions, visualizes dependencies unambiguously, and puts visual emphasis on view-dependent



```
«AUTOSARComponent» E-GasSystem
  «AUTOSARComponent» EGasCtrl
  «AUTOSARComponent» EGasMonr
    «AUTOSARComponent» EGasMon
    «AUTOSARComponent» MonImpPr
    «AUTOSARPort» APedl1
    «AUTOSARPort» APedl2
    «AUTOSARPort» ThrActuEna
    «AUTOSARPort» ThrActuPwm
    «AUTOSARPort» ThrPosn1
    «AUTOSARPort» ThrPosn2
  «AUTOSARPort» APedl1
  «AUTOSARPort» APedl2
  «AUTOSARPort» ThrActuEna
  «AUTOSARPort» ThrActuPwm
  «AUTOSARPort» ThrPosn1
  «AUTOSARPort» ThrPosn2
«AUTOSARComponent» Rte_IRead_E-GasS
```

SafetySoftwareArchitecture::AUTOSARPort (ThrActuPwm)	
DataType	uint8
DefaultValue	0
Direction	out
lowerLimit	0
PortType	AUTOSAR Sender Receiver
ScalingLSB	0,00195
ScalingOffset	0
SignalUnit	%
upperLimit	100

Fig. 4. Top-level representation of demonstration use-case in enterprise architect

constraints (such as graphical safety-criticality highlighting of SW modules in Fig. 4).

The presented use-case amounts to a total count of 41 model artifacts with 361 configuration parameters and 30 relations between the elements. This small example already indicates that relations between the model elements and number of model elements become confusing. Therefore, manual transformation of the information represented within the models is cumbersome and error-prone and would inherit lots of additional work to ensure consistency of both models in terms of safety-critical software development.

With our approach these information and model artifacts are checked for consistency constraints (such as point-to-point consistency of interface configurations) before automatically transferred via 212 lines of auto-generated Matlab API code. This auto-generation of Matlab API code provides evidence and ensures completeness of the model transformation. Furthermore, the SW import functionality enables round-trip engineering and bi-directional updates of both models and therefore supports evidence for consistency of both models.

According to the presented bridge approach in Sect. 3, the first step during the transformation is the decomposition of the software architectural design. Each software subsystem (like the AUTOSAR Composition *EGasCtrl* in Fig. 4) is analysed and the comprised software modules (e.g. *EGasCtrl::InpPreProc in Fig.* 5) are extracted. An essential information at the software module architecture is the trigger definition, representing the later task timing property of the module at the integrated system.

With the gathered architectural, timing, and interface information the Simulink/TargetLink model is generated by the previously described utilization of the Matlab COM Automation Server. During this process, a new Simulink root model is created and for each trigger type, which appears at the architectural design,

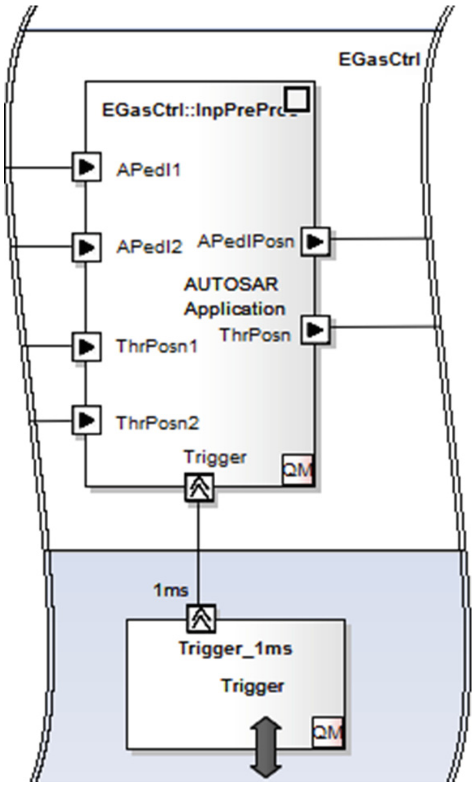
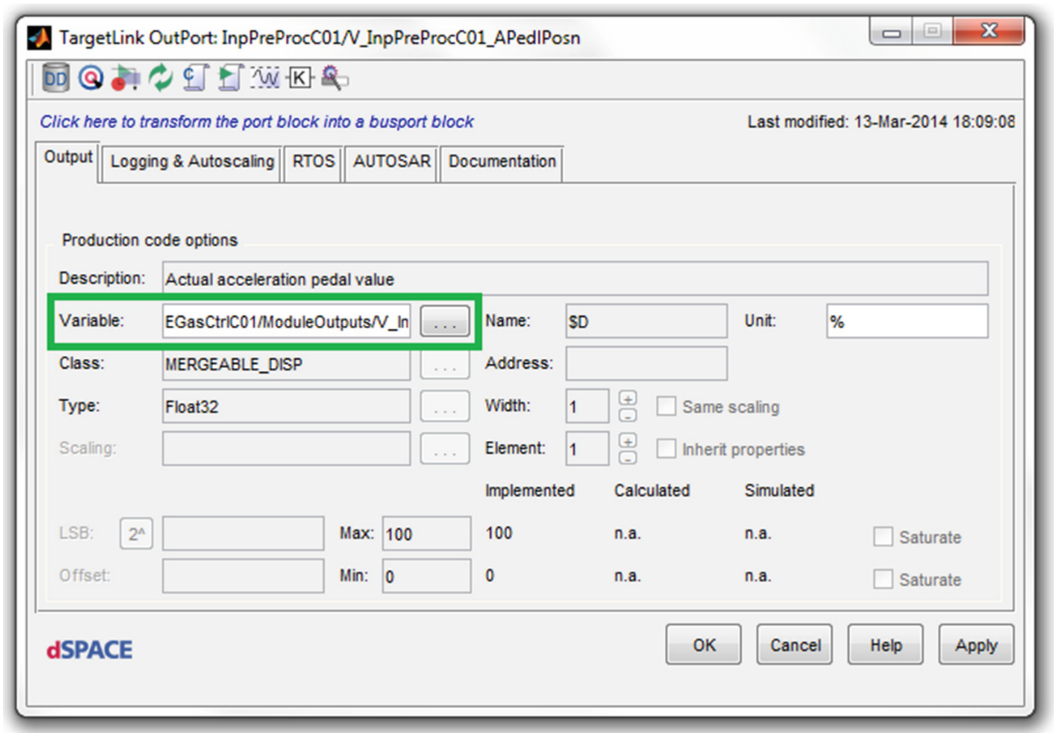


Fig. 5. Software module representation at the demonstrated use-case

an own subsystem is placed at the models top level. Afterwards, each software subsystem is transferred to the appropriate timing subsystem or even split up into multiple tasks if necessary. E.g. the subsystem *E-GasSystem::EGasCtrl* contains software modules with different timing attributes. Therefore, the *EGasCtrl*-Subsystem is available at multiple timing subsystems at the Simulink/TargetLink model. To facilitate a multi developer scenario, a separate model file is created for each software subsystem and linked as a reference model at the root model. To complete the Simulink/TargetLink model subsystem generation, all software modules are transferred into their appropriated software subsystem.

The transformation process described so far, generates a software framework out of the *Autosar Composition* and *Autosar Application* blocks at the design. To provide a complete model framework, which serves as a basis for the subsequent software unit development, the interfaces as well as their connections are transferred to Simulink/TargetLink in the next step. For an efficient and dependable handling of the signals, a *Data Dictionary* and the related tool *Data Dictionary Manager* is used. Again, to facilitate a multi developer scenario, a data dictionary file is created for each software subsystem and included at a data dictionary root file. Every signal from the architectural design is stored at the appropriate data dictionary, including all available attributes like value limits, scaling, etc. Furthermore, the exporter algorithm simply link this entry wherever the signal occurs in the Simulink/TargetLink model (see Fig. 6).



TargetLink OutPort: InpPreProcC01/V_InpPreProcC01_APedlPosn

Click here to transform the port block into a busport block

Last modified: 13-Mar-2014 18:09:08

Output | Logging & Autoscaling | RTOS | AUTOSAR | Documentation

Production code options

Description:	Actual acceleration pedal value				
Variable:	EGasCtrlC01/ModuleOutputs/V_In	Name:	$D	Unit:	%
Class:	MERGEABLE_DISP	Address:			
Type:	Float32	Width:	1	☐ Same scaling	
Scaling:		Element:	1	☐ Inherit properties	

			Implemented	Calculated	Simulated	
LSB: 2^		Max: 100	100	n.a.	n.a.	☐ Saturate
Offset:		Min: 0	0	n.a.	n.a.	☐ Saturate

Fig. 6. Link to the signal at the data dictionary, set by the exporter

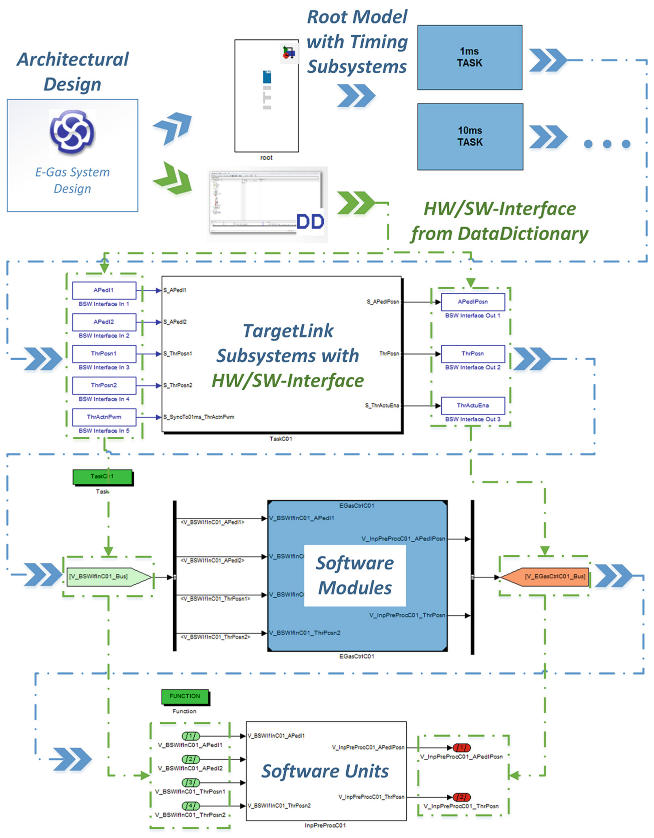


Fig. 7. Architectural design to Simulink/TargetLink model transformation workflow

The process workflow *Architectural Design to Simulink/TargetLink Model Transformation* is shown in Fig. 7, with the blue-coloured subsystem creation path, and the green-coloured signal creation path. Although, the chosen use-case example is not too complex, presenting all generated artifacts would go beyond the scope of this contribution. Therefore, the illustration showcases the creation of the Simulink/TargetLink *root model*, the software subsystem *EGasCtrl* including the signals from and to the basis software, and the software module *InpPreProc*, which was already presented in Fig. 5.

If the Simulink/TargetLink model already exists when the *Software Architecture Exporter* is triggered, the *Software Architecture Importer* is started automatically in the background to check the consistency between the architectural design and the software model. If all artifacts at the model are available at the design, and new items exist at the design, the software model is updated by the exporter analogue to the procedure described above for completely new Simulink/TargetLink models. If there are new or deleted artifacts at the software model, a notification is displayed and the user is prompted to determine the further procedure, like shown in Fig. 3, Sect. 3.

5 Conclusion

Dependable system development is an emerging trend in automotive industry, aiming to provide a convincing argumentation that the system under development has achieved a certain level of maturity. Without an adequate tool chain, which enables a smooth transition between the different levels along the system development, it is hard to obtain this demanded maturity.

Especially creating a software model from the architectural design manually is exhausting and error-prone. The risk to e.g. connect signals incorrect, set wrong attributes or simply overlook a changed parameter is very high.

This paper presented an efficient approach to avoid the risk of introducing errors while developing the software according to the architectural design, by creating the software model framework fully automated. Furthermore, the concept facilitates bidirectional traceability as well as consistency. These properties are elemental key factors for a high quality development and postulated by the widespread quasi-standard *Automotive SPICE*. Additionally, the shown techniques facilitate round-trip engineering by the presented import/export functionality regarding the models on different development levels and tools.

In terms of safety-critical development and reuse the presented approach features are crucial to transfer information between separated tools and link supporting safety-relevant information. Moreover, the approach eliminates the need of manual information rework without adequate tool support, ensuring reproducibility, and traceability argumentation.

The application of the presented approach has been demonstrated utilizing a simplified version of the well-known E-Gas concept, which is intended to be used for training purpose of students and engineers and not for representing an exhaustive or commercial sensitive project.

References

1. Blom, H., Loenn, H., Hagl, F., Papadopoulos, Y., Reiser, M.-O., Sjoestedt, C.-J., Chen, D., Kolagari., R.: EAST-ADL - an architecture description language for automotive software-intensive systems. White Paper 2.1.12 (2013)
2. Boldt, R.: Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group (2009)
3. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: from isolated tool to integrated model engineering environments, IEEE Magazin (2008)
4. Buchmann, T., Westfechtel, B.: Towards incremental round-trip engineering using model transformations. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 130–133, Sept 2013
5. Chen, D.J., Johansson, R., Lönn, H., Papadopoulos, Y., Sandberg, A., Törner, F., Törngren, M.: Modelling support for design of safety-critical automotive embedded systems. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 72–85. Springer, Heidelberg (2008)
6. Ebert, C., Jones, C.: Embedded software: facts, figures, and future. IEEE Comput. Soc. **0018–9162**(09), 42–52 (2009)
7. Giese, H., Hildebrandt, S., Neumann, S.: Model synchronization at work: keeping SysML and AUTOSAR models consistent. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Nagl Festschrift. LNCS, vol. 5765, pp. 555–579. Springer, Heidelberg (2010)
8. Holtmann, J., Meyer, J., Meyer, M.: A seamless model-based development process for automotive systems (2011)
9. ISO - International Organization for Standardization. ISO 26262 Road vehicles functional safety, Part 1–10 (2011)
10. Kawahara, R., Dotan, D., Sakairi, T., Ono, K., Kirshin, A., Nakamura, H., Hirose, S., Ishikawa, H.: Verification of embedded system's specification using collaborative simulation of SysML and Simulink models. In: Proceedings of Second International Conference on Model Based Systems Engineering, pp. 21–28, March 2009
11. Lau, K.-K., Tepan, P., Tran, C., Saudrais, S., Tchakaloff, B.: A holistic (Component-based) approach to AUTOSAR designs. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 203–207, Sept 2013
12. Macher, G., Armengaud, E., Kreiner, C.: Automated generation of AUTOSAR description file for safety-critical software architectures. In: 12. Workshop Automotive Software Engineering (ASE), Lecture Notes in Informatics (2014)
13. Macher, G., Armengaud, E., Kreiner, C.: Bridging automotive systems, safety and software engineering by a seamless tool chain. In: 7th European Congress Embedded Real Time Software and Systems Proceedings, pp. 256–263 (2014)
14. Mader, R., Griessnig, G., Eric, A., Andrea, L., Christian, K., Bourrouilh, Q., Steger, C., Weiss, R.: A bridge from system to software development for safety-critical automotive embedded systems. In: 38th Euromicro Conference on Software Engineering and Advanced Applications, pp. 75–79 (2012)
15. Meyer, J.: Eine durchgaengige modellbasierte Entwicklungsmethodik fuer die automobile Steuergeraeteentwicklung unter Einbeziehung des AUTOSAR Standards. Ph.D thesis, Universitaet Paderborn, Fakultaet fuer Elektrotechnik, Informatik und Mathematik, July 2014

16. Pagel, M., Brörkens, M.: Definition and generation of data exchange formats in AUTOSAR, process independent model. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 52–65. Springer, Heidelberg (2006)
17. Petrissans, A., Krawczyk, S., Veronesi, L., Cattaneo, G., Feeney, N., Meunier, C.: Design of future embedded systems toward system of systems - trends and challenges. European Commission, May 2012
18. Pretschner, A., Broy, M., Kruger, I.H., Stauner, T.: Software engineering for automotive systems: a roadmap. In: 2007 Future of Software Engineering, FOSE 2007, Washington, DC, USA, pp. 55–71, IEEE Computer Society (2007)
19. Quadri, I.R., Sadovykh, A.: MADES: a SysML/MARTE high level methodology for real-time and embedded systems (2011)
20. Rodriguez-Priego, E., Garcia-Izquierdo, F., Rubio, A.: Modeling issues: a survival guide for a non-expert modeler. Models **2010**(2), 361–375 (2010)
21. Sandmann, G., Seibt., M.: AUTOSAR-compliant development workflows: from architecture to implementation - tool interoperability for round-trip engineering and verification and validation. In: SAE World Congress and Exhibition 2012, (SAE 2012–01-0962) (2012)
22. Scuro, G.: Automotive industry: innovation driven by electronics (2012). http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/
23. Sjoestedt, C.-J., Shi, J., Toerngren, M., Servat, D., Chen, D., Ahlsten, V., Loenn, H.: Mapping simulink to UML in the design of embedded systems: investigating scenarios and structural and behavioral mapping. In: OMER 4 Post Workshop Proceedings, April 2008
24. Thyssen, J., Ratiu, D., Schwitzer, W., Harhurin, E., Feilkas, M., München, T.U., Thaden, E.: A system for seamless abstraction layers for model-based development of embedded software. In: Software Engineering Workshops, pp. 137–148 (2010)
25. Voget, S.: SAFE RTP: an open source reference tool platform for the safety modeling and analysis. In: Embedded Real Time Software and Systems Conference Proceedings (2014)
26. Zurawka, T., Schaeuffele, J.: Method for checking the safety and reliability of a software-based electronic system, January 2007

Bibliography

[1] ISO/IEC 15504, Information technology - Process assessment, 2004.

[2] IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, April 2010.

[3] *10 Years AUTOSAR: The Worldwide Automotive Standard for E/E Systems.* ATZ extra. Springer Vieweg, October 2013.

[4] ISO/IEC 33001:2015 - Information technology - Process assessment - Concepts and terminology, 2015.

[5] A. Alderson. Meta-CASE Technology. In *Software Development Environments and CASE Technology*, volume 509 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1991.

[6] L. Apvrille and A. Becoulet. Prototyping an Embedded Automotive System from its UML/SysML Models. In *Proceedings of Embedded Real Time Systems and Software (ERTSS 2012)*, pages 87–96, 2012.

[7] Automotive SIG. Automotive SPICE®Process Assessment Model. Technical report, The SPICE User Group, May 2010. Version 2.5.

[8] AUTOSAR. Current Partners. Online Resource. retrieved February 09, 2016, from http://www.autosar.org/partners/current-partners/.

[9] N. Baetens. Comparing graphical DSL editors: AToM3, GMF, MetaEdit+. *University of Antwerp*, 2011.

[10] B. Biafore. *Visio 2007 Bible.* John Wiley & Sons, 2007.

[11] M. Broy, S. Kirstan, H. Krcmar, B. Schätz, and J. Zimmermann. What is the benefit of a model-based design of embedded software systems in the car industry? *Software Design and Development: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*, page 310, 2013.

[12] R. N. Charette. This car runs on code. Online, February 2009. retrieved March 14, 2016, from http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.

[13] M. Chemuturi. *Requirements Engineering and Management for Software Development Projects.* Springer New York, 2013.

[14] Code Project. WPF Diagram Designer - Part 4. Online Resource. retrieved March 23, 2015, from http://www.codeproject.com/Articles/24681/WPF-Diagram-Designer-Part.

[15] S. Cook, G. Jones, S. Kent, and A. C. Wills. *Domain-Specific Development with Visual Studio DSL Tools.* Microsoft .NET Development Series. Addison-Wesley Longman, Amsterdam, May 2007.

[16] P. B. Crosby. *Quality Is Free: The Art of Making Quality Certain.* Penguin Putnam mass, 1980.

[17] J. Davis. GME: The Generic Modeling Environment. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOPSLA '03*, pages 82–83. ACM, 2003.

[18] T. DeMarco and T. Lister. *Wien wartet auf Dich!* The original English edition: Peopleware - Productive Projects and Teams. Carl Hanser Verlag München Wien, second edition, 1999.

[19] H. Dubois, M.-A. Peraldi-Frati, and F. Lakhal. A Model for Requirements Traceability in a Heterogeneous Model-Based Design Process: Application to Automotive Embedded Systems. In *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 233–242. IEEE, 2010.

[20] European Union. *User guide to the SME definition.* Publications Office of the European Union, Luxembourg, 2015. ISBN 978-92-79-45322-9, Ref. Ares(2015)1914862 - 06/05/2015.

[21] Herstellerinitiative Software . HIS - Working Group Assessment. *HIS Arbeitskreis "Process Assessment"*, (V.31), June 2008.

[22] M. Hillenbrand. *Funktionale Sicherheit nach ISO26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen.* PhD thesis, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, November 2011.

[23] C. R. Hopkins. Creating custom validation rules in visio 2013. Online Resource. retrieved February 23, 2016, from http://blogs.msdn.com/b/chhopkin/archive/2013/01/03/creating-custom-validation-rules-in-visio-2013.aspx.

[24] P. Hudak. Domain-specific languages. *Handbook of Programming Languages*, 3:39–60, 1997.

[25] ISO 26262-2, Road vehicles - Functional safety - Part 2: Management of Functional safety. International standard, International Organization for Standardization, Geneva, CH, November 2011.

[26] ISO 26262, Road vehicles - Functional safety. International standard, International Organization for Standardization, Geneva, CH, November 2011.

[27] E. Juliot and J. Benois. Viewpoints creation using Obeo Designer or how to build Eclipse DSM without being an expert developer? *Obeo Designer Whitepaper*, 2010.

[28] S. Kelly, K. Lyytinen, and M. Rossi. MetaEdit+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Advanced Information Systems Engineering*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin Heidelberg, 1996.

[29] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press. John Wiley & Sons, March 2008.

[30] H. Kern, A. Hummel, and S. Kühne. Towards a Comparative Analysis of Meta-Metamodels. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11 & VMIL'11*, pages 7–12. ACM, 2011.

[31] H. Kern and S. Kühne. Model Interchange between ARIS and Eclipse EMF. In *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA '07*, 2007.

[32] A. E. Kouhen, C. Dumoulin, S. Gerard, and P. Boulet. Evaluation of Modeling Tools Adaptation. *HAL archives-ouvertes.fr*, (hal-00706701), June 2012.

[33] C. Kreiner, R. Messnarz, A. Riel, D. Ekert, M. Langgner, D. Theisens, and M. Reiner. Automotive Knowledge Alliance AQUA - Integrating Automotive SPICE, Six Sigma, and Functional Safety. In F. McCaffery, R. V. O'Connor, and

R. Messnarz, editors, *Systems, Software and Services Process Improvement*, volume 364 of *Communications in Computer and Information Science*, pages 333–344. Springer Berlin Heidelberg, 2013.

[34] G. Macher. *Framework for the Integrated Model-Based Development of Dependable Automotive Systems and Software.* PhD thesis, Graz University of Technology, November 2015.

[35] G. Macher, H. Sporer, E. Armengaud, E. Brenner, and C. Kreiner. A Seamless Model-Transformation between System and Software Development Tools. In *8th European Congress Embedded Real Time Software and Systems*, ERTS '16, Toulouse (France), January 27 - 29, 2016. in press.

[36] S. Martínez-Fernández, C. P. Ayala, X. Franch, and E. Y. Nakagawa. A Survey on the Benefits and Drawbacks of AUTOSAR. In *Proceedings of the First International Workshop on Automotive Software Architecture*, pages 19–26. ACM, 2015.

[37] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[38] R. Messnarz, C. Kreiner, O. Bachmann, A. Riel, K. Dussa-Zieger, R. Nevalainen, and S. Tichkiewitch. Implementing Functional Safety Standards - Experiences from the Trials about Required Knowledge and Competencies (SafEUr). In F. McCaffery, R. V. O'Connor, and R. Messnarz, editors, *Systems, Software and Services Process Improvement*, volume 364 of *Communications in Computer and Information Science*, pages 323–332. Springer Berlin Heidelberg, 2013.

[39] MetaCase. EAST-ADL for automotive embedded architectures. Online Resource. retrieved February 22, 2016, from http://www.metacase.com/solution/east-adl.html.

[40] J. Meyer. *Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards.* PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Germany, July 2014.

[41] Microsoft Developer Network. Modeling SDK for Visual Studio - Domain-Specific Languages. Online Resource. retrieved January 19, 2016, from https://msdn.microsoft.com/en-us/library/bb126259.aspx.

[42] P. Mohagheghi and Ø. Haugen. Evaluating Domain-Specific Modelling Solutions. In *Advances in Conceptual Modeling - Applications and Challenges*, volume 6413 of *Lecture Notes in Computer Science*, pages 212–221. Springer Berlin Heidelberg, 2010.

[43] Object Management Group. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. Online Resource. retrieved February 23, 2016, from http://www.omgmarte.org/.

[44] Object Management Group. Business Process Model and Notation (BPMN). BPMN Specification, Normative Documents formal/2011-01-03, January 2011.

[45] Object Management Group. Model Driven Architecture (MDA) - MDA Guide rev. 2.0. Technical Report ormsc/14-06-01, Object Management Group, Inc., June 2014.

[46] J.-D. Piques and E. Andrianarison. SysML for embedded automotive Systems: lessons learned. *Interfaces*, 3:3b, 2011.

[47] C. Preschern, N. Kajtazovic, and C. Kreiner. Efficient development and reuse of domain-specific languages for automation systems. *International Journal of Metadata, Semantics and Ontologies*, 9(3):215–226, 2014.

[48] I. R. Quadri, A. Sadovykh, and L. S. Indrusiak. MADES: A SysML/MARTE high level methodology for real-time and embedded systems. In *Proceedings of the 10th Embedded Realtime Software and Systems Conference*, 2012.

[49] M. Reke. *Modellbasierte Entwicklung automobiler Steuerungssysteme in kleinen und mittelständischen Unternehmen.* PhD thesis, Fakultät für Mathematik, Informatik und Naturwissenschaften, RWTH Aachen University, 2012.

[50] A.-W. Scheer. *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen.* Springer-Verlag Berlin Heidelberg, third edition, 2013.

[51] A.-W. Scheer and M. Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In W. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 376–389. Springer Berlin Heidelberg, 2000.

[52] B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software, IEEE Computer Society*, 20(5):19, 2003.

[53] Software Engineering Institute. CMMI® for Development, Version 1.3. Technical Report CMU/SEI-2010-TR-033, ESC-TR-2010-033, SEI, Carnegie Mellon, November 2010.

[54] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. the eclipse series. Addison-Wesley, second edition, 2008.

[55] P. Swithinbank, M. Chessell, T. Gardner, C. Griffin, J. Man, H. Wylie, and L. Yusuf. *Patterns: Model-Driven Development Using IBM Rational Software Architect*. Number SG24-7105-00 in IBM Redbooks. International Business Machines Corporation, first edition, December 2005.

[56] The Eclipse Foundation. Graphical Modeling Project (GMP). Online Resource. retrieved February 23, 2016, from http://www.eclipse.org/modeling/gmp/.

[57] The European Parliament and the Council of the European Union. Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on Machinery, and Amending Directive 95/16/EC (Recast). European Union Law L 157/24, Official Journal of the European Union, 2006.

[58] VDA QMC Working Group 13 / Automotive SIG. Automotive SPICE Process Assessment / Reference Model. Technical Report Revision ID: 470, July 2015. Version 3.0.

[59] V. Vujović, M. Maksimović, and B. Perišić. Sirius: A rapid development of DSM graphical editor. In *18th International Conference on Intelligent Engineering Systems (INES)*, pages 233–238. IEEE, 2014.

[60] A. Watson. UML® vs. DSLs: A false dichotomy. *Butler Group Review*, (omg/08-09-03), April 2007.

[61] W. Wolf. *Computers as Components - Principles of Embedded Computing System Design*. Morgan Kaufmann, Waltham, Mass., USA, third edition, 2012.