

Masterarbeit

Connecting Model-Based System Engineering and AVLab SW Test Environment

Müslüm Atas

Institut für Technische Informatik
Technische Universität Graz
Vorstand: Univ.-Prof.Dipl.-Inform. Dr. sc.ETH Kay Uwe Römer



Begutachter: Dipl.-Ing.Dr. techn. Kreiner, Christian Josef
Betreuer: Dipl.-Ing. BSc Macher, Georg Franz Heinrich

Graz, im Oktober 2015

Kurzfassung

Das Hauptziel des Projekts ist, die fehlenden, automatisierten Datenaustauschprozesse zwischen den System- und Software Engineering-Tools in eingebetteten Automotive-Systeme zu vernetzen. Solch eingebettete, vernetzte Systeme in der Automobilindustrie sind auf Grund ihrer Komplexität sehr schwer handzuhaben. Die Steuerung, Entwicklung und ihre komplizierte technische Kommunikation, dieser miteinander vernetzten „Computer“ erfordert erhöhten Arbeitsaufwand.

Die Luxusautos heutzutage arbeiten mit hunderten elektronischen Steuereinheiten (ECUs) und mehreren Millionen Code-Zeilen. Diese Steuereinheiten und Code-Zeilen werden oftmals in verschiedenen großen Institutionen und Teams entwickelt und getestet. Parallel zu diesen Herausforderungen müssen noch zahlreiche funktionelle Sicherheitsstandards (sowie ISO26262[HHA+10]) und -techniken bei allen Entwicklungsprozessen in der Automobilindustrie eingehalten werden. Aus diesen Gründen ist es nahezu unmöglich, dass diese anspruchsvollen Entwicklungsprozesse von Menschen manuell überwacht werden. Um jene Herausforderung zu meistern, werden in der Regel modellbasierende Entwicklungs- und automatisierte Datenaustausch-Methoden angewendet.

Ein auf Modellen basierender Entwicklungsansatz verbessert die Konsistenz, Korrektheit und Vollständigkeit der Entwicklung und ermöglicht verschiedene Sichtweisen für eine bessere Produktentwicklung. Ein weiterer Vorteil ist, dass Markteinführung und Entwicklungsprozesse deutlich effizienter erfolgen können, wie im Laufe dieser Arbeit gezeigt wird. Der Benutzer dieses Tools muss auch nicht über besondere Kenntnisse verfügen und kann daher die Daten zwischen den verschiedenen Domänen mit sehr geringem Aufwand austauschen. Zusätzlich besteht die Möglichkeit die Daten in eine andere Domäne zu transferieren, falls diese für den User nicht verständlich war.

Das Resultat dieses Projektes ist ein Tool, das einen automatischen Datenaustausch zwischen den verschiedenen Abstraktionsebenen gewährleistet und die auf Model basierenden Entwicklungsansätze in eingebetteten Automobile-Systeme unterstützt. Dieser Datenaustausch erfolgt bidirektional über eine Programmierschnittstelle zwischen den folgenden Tools: Enterprise Architect, AVLab, ADD und Microsoft Office Excel.

Schlüsselwörter: *Model-basierende Entwicklung, System Engineering Tool, Software Engineering Tool, ISO 26262, Rückverfolgbarkeit, Datenaustausch über Programmierschnittstelle (API), AVLab, ADD, Enterprise Architect*

Abstract

The main goal of this project was to bridge the gap between the system- and software engineering tools by an automated exchange of model data. Automotive systems have become more and more challenging due to their evolution to on-road computers. These networked on-road computers and their highly complex technical communication concepts take a huge amount of effort to manage and develop.

Nowadays luxury vehicles contain several hundred ECUs and millions of lines of code, which have been developed and tested by different large institutions and teams. In parallel to these challenges, all the development processes have to be handled in compliance to automotive safety standards (such as ISO26262[HHA⁺10]) and techniques. Due to these facts, such complex systems are almost impossible to manage manually. To solve or facilitate such challenges, the approaches of model-based development and automated data transformation should be applied in the automotive domain.

A model-based-development approach improves consistency, correctness and completeness and enables different views regarding a product-development among different work products. This automated information exchange between different tools accelerates the time-to-market and development processes. Due to this fact, the transformation of information between different tools can be ensured without needing to understand the whole of the system. Additionally, failure detection and traceability of information between several tools is realizable and easily manageable.

The contribution is to provide an automated data exchange between different abstraction levels via different SW tools and to support the model-based development approach in embedded automotive systems. The developed tool of this project manages the whole data transfer and its processing automatically and bidirectionally via their „Application Programming Interface“ between the following tools: Enterprise Architect, AVLab, ADD and Microsoft Office Excel.

Keywords: *Model-based development, system engineering tool, software engineering tool, Model driven API tool, ISO 26262, traceability, data exchange via API, AVLab, ADD, Enterprise Architect*

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am

.....
(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

.....
date

.....
(signature)

Danksagung

Das Projekt wurde in Zusammenarbeit mit ITI - Institute für Technische Informatik Graz und der AVL List GmbH Graz durchgeführt. Mein besonderer Dank gilt Dr. Christian Kreiner, Dr. Eric Armengaud und Dipl. Ing. Georg Macher, die mich beraten haben und mir bei der Entwicklung des Projekts zur Seite gestanden sind. Des Weiteren möchte ich mich bei Alexander Cucek, Alvarez De Eulate Maider, Manuela Richter, Christine Zilker, Juergen Konnerth, Dipl. Ing. Dirk Denger und Glashuettner Jakob bedanken, die mich unterstützt haben.

Darüber hinaus möchte ich mich ganz herzlich bei meinen Kollegen Rene Obendrauf, Seidl Matthias und bei meinen Eltern und meiner Freundin, für die hervorragende Ermutigung, bedanken.

Graz, im Oktober 2015

Müslüm Atas

Contents

1	Introduction	9
1.1	Motivation	13
2	Related Works	15
3	Approach	28
3.1	Data Transfer between different tools	30
3.1.1	Export of EA SW model representations	30
3.1.2	Import into EA model	34
3.1.3	Write into Log File	35
3.2	EA SW Modeling Elements	37
3.2.1	AUTOSAR Component	37
3.2.2	AUTOSAR Port	39
3.2.3	AUTOSAR Connector	40
3.3	AVLab/ADD and its Components	41
3.3.1	AVLab	41
3.3.2	ADD	43
4	Implementation	46
4.1	Export EA Model	48
4.1.1	Export To ADD	48
4.1.2	Export To Excel	49
4.2	Import to EA Model	50
4.2.1	Import From ADD	50
4.2.2	Import From Excel	51
4.3	Available Application scenarios	52
5	Application	54
5.1	Application	54
5.2	Case Study	55
6	Conclusion	58
7	Future Works	59
A	Tool Programmer-Guideline	60

A Abbreviation	70
A.1 Definitions	71
References	72
Literaturverzeichnis	72

List of Figures

1.1	V-Model and its phases	10
1.2	Work-flow overview between different tools	13
2.1	System Engineering views [Cla09]	15
2.2	System or SoS V-Model [Cla09]	16
2.3	Model Driven EAI Architecture [15706]	17
2.4	AMD and ASCG global view [KLPK13]	18
2.5	TPT test process [BK08]	22
2.6	process of automated development steps [FNH09]	24
2.7	Model-based systems engineering quality [DF07]	25
2.8	For a special case study modiflicated V-Model [MLD ⁺ 12]	26
3.1	Export EA Model Tab	31
3.2	level1 worksheet with some hidden columns	32
3.3	All Excel columns for <i>SW Level Components</i>	33
3.4	All Excel columns for <i>SW Ports</i>	33
3.5	Import from ADD Tab with all ADD components	35
3.6	Import from Excel Tab with all Excel components	36
3.7	AUTOSAR Component with its properties	38
3.8	AUTOSAR Ports with its properties	39
3.9	Whole EA Model	40
3.10	AVLab as Mindmap with links to related materials	42
3.11	ADD user interface	44
3.12	ADD Components	45
4.1	Program execution path	47
5.1	Time consumption in seconds as a chart	55

List of Tables

3.1	Available ASILs	37
3.2	Available characteristic properties	38
3.3	Association rules between EA elements	41
3.4	Meaning if the association multiplicities	41
5.1	Consumed time comparison	56

Chapter 1

Introduction

The development of embedded automotive systems has become more and more challenging due to the transformation from mechanical systems to complex mechatronic systems in recent years. The embedded automotive systems have nowadays been evolved into on-road computer systems which are considerable complex and somewhat confusing. And all these systems include several electronic control units (ECU) with a large volume of software code that has been developed, analyzed and tested by different large institutions and teams. The process of integration between different parts will raise new challenges and will be nearly impossible for humans to manage in near future. This serious challenge has to be handled in compliance with standards and techniques. One of the main standards for road vehicles is the functional safety standard ISO 26262 [HHA⁺10].

The validity of the different parallel running applications in time- and value domain has to be ensured according to the safety standard ISO 26262 [HHA⁺10]. Safety standards, such as ISO 26262 [HHA⁺10] for road vehicles, help with the identification and mitigation of risks and provide guidance for the development of safety critical systems throughout the whole product life cycle. During this process the relevant information passes through different tools in order to analyze, test and develop a system. Some critical information might get lost while data is passed through several tools. Due to this incomplete information collection, it is very difficult to keep those standards. Another cause of insufficient data transformation can be a lack of knowledge and experience.

To avoid the human errors, all the process steps should be automated and strictly examined. Thus the error factor is reduced and the whole development process is accelerated. According to this approach modification and improvement can be easily accomplished and an error can be quick detected in several different processes. Thus, a speedup of the development process can be achieved.

Process models such as the V-Model [Cla09] and the waterfall model [Ben83] are widely applied for different systems engineering processes. By means of these models, the various product processes are facilitated by developing, testing and verifying a system. The V-Model is one of the most widely used system engineering process model in the automotive domain and depicted in Figure 1.1.

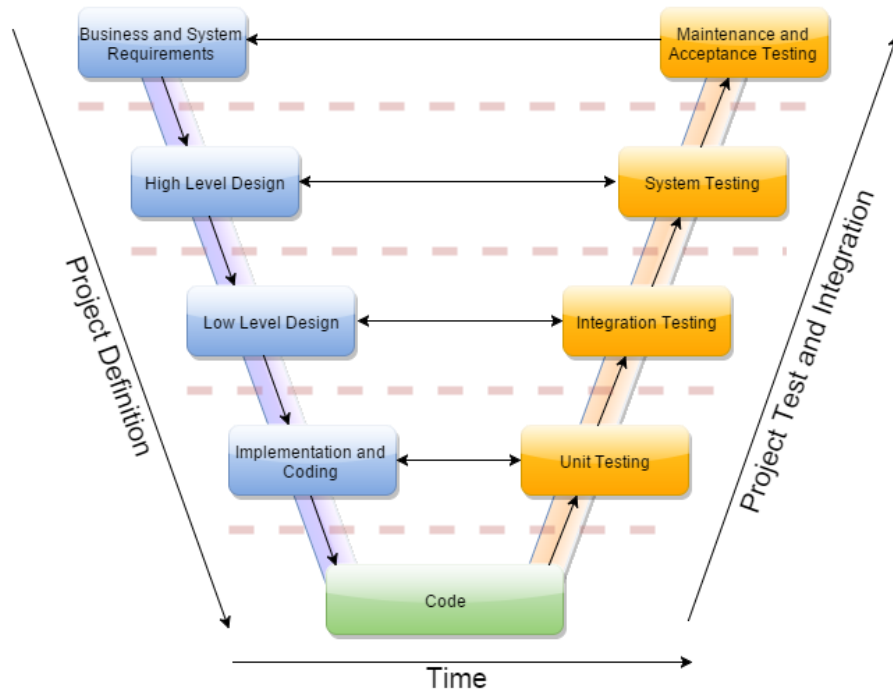


Figure 1.1: V-Model and its phases

The name of the model comes from its V-Shaped figure and provides verification and validation processes for a system. The model accompanies a product from its creation up to time-to-market phase. Whereby each phase of this model has to be completed before the next phase begins. In parallel to each developing process a test phase is executed, which validates the correctness and completeness of the product phases.

The developing process in V-Model begins top left and is divided in following four main parts:

- **Product Concept and Requirements:** All the product specific requirements for a successful product development will be defined in this part and their reliability, availability and serviceability will be checked in parallel. Then all the possible solution paths will be listed, which leads to the required goals. If all the possible solutions are defined and analyzed, then the best developing path/solution will be chosen. To find the best solution for a good product quality and efficient development is usually a hard decision. That's why this process is the most critical phase for a product success.

Whereby this approach is the hardest decision, due to achieve the best quality, least cost and most efficient time-to-market features. This process is that's why the most critical phase for a product success.

- **High Level Design:** The second process step defines the system architecture and design of a project. This enables a complete overview of the system and detects

failures and problems with the main idea. Parallel to this phase, an integration test helps to develop a properly designed architecture. A clear and simple architecture increases the implementation and coding processes but shortens the development cycle.

- **Low Level Design:** This process step refines the design for a software or hardware implementation. An initial class diagram with all the software components and software logic has to be defined here. This class diagram describes the relationship between each component with their attributes and operations and the interoperability of those components. Additionally all the components and their intercommunication will be simultaneously tested and verified.
- **Implementation and Coding:** The last process step defines the product at a low level and delivers a software product at the end. In this phase the product will be tested, developed and refined. The challenge here is the developed SW-Tool with all the possible test cases to generate different outputs, which define passed or failed test cases. That means all the input variants must be well-conceived and those possibilities have to be tested carefully to achieve a perfect product. On this occasion, the candidate software must be tested with single- or/and multi-core architectures to verify the tool on different platforms. The mentioned test cases have to be implemented precisely, so that as a result, each individual line of code is verified.

During the above mentioned process phases all the development artifacts will be tested parallel until the last part. The system- and software developing phases of the V-Model with their test cases are the main focus of this project. The tool developed during this thesis works with model-driven system- and software engineering tools.

The contribution of this thesis is to bridge the existing gap between model-driven system engineering tools like 'Enterprise Architect'¹, 'Artisan Studio'² and software engineering tools like 'AVLab'³ for embedded automotive systems (see Figure 1.1). This model-based-development approach improves consistency, correctness and completeness and enables different views regarding a product-development among different work products. Additionally time-to-market and development processes will be accelerated and the whole system will always be ready to make the required modifications. This thesis focuses on an automated data exchange between model-driven system- and software engineering tools via their API interfaces. The idea and detailed development concepts of this thesis will be explained in the following sections.

The document is organized as follows: Section 2 gives an overview of the available related tools and approaches. In Section 3, instructions regarding the application and

¹<http://www.sparxsystems.at/start/startseite/>

²<http://www.atego.com/products/technology-overview/>

³https://desktop.avl.com/projects/12/0061/Data_Exchange/docs/AVLab/index.html

functions of the tools and techniques used is documented. Furthermore, all working steps are explained and discussed in detail, along with their pros and cons. Section 4 explains the implementation techniques and methods used for this tool. In addition, a detailed description of the development steps will be explained for specific application scenarios. Section 5.1 lists the available application scenarios of the tool and compares a specific application scenario with a manual work procedure. In Section 6, this work is concluded with an overview of the presented approach and methodology. Finally, all the planned future works for the tool are described in Section 7.

1.1 Motivation

In the initial stages of this project, there were several tools which were unable to communicate automatically with each other. Further more, all the tools used have to be designed or filled in manually, which increases the error factor due to human faults. To reduce this, several automated programming interfaces between these tools were built, which enable the exchange of data from one tool to another. This automated information exchange between different tools accelerates the time-to-market and development processes. Due to this advantage, the transformation of information between different tools can be ensured without needing to understand the whole of the system. Additionally, failure detection and traceability of information across tool boundaries is easier realizable and manageable.

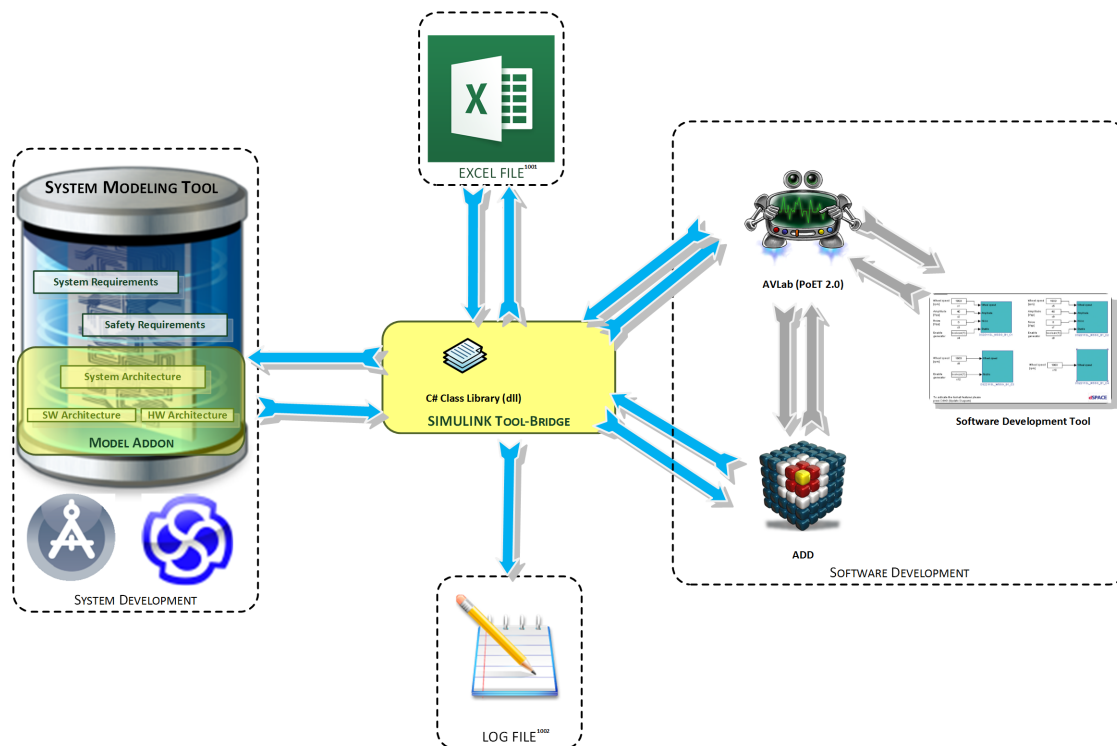


Figure 1.2: Work-flow overview between different tools

This project was developed in programming language C# within a Microsoft Visual Studio environment. The developed tool enables an automated correct, complete and consistent data transfer between the following tools without losing any information:

- Sparx Systems Enterprise Architect (EA)
- Atego Artisan Studio

¹⁰⁰¹<http://goo.gl/JLFFFs>

¹⁰⁰²<http://icons.iconarchive.com/icons/everaldo/crystal-clear/128/App-edit-icon.png>

- Microsoft Office Excel⁴
- AVLab
- AVLab/PoET2⁵-Database (ADD)

An overview of this work-flow between different tools can be seen in Figure 1.2. This figure shows the interaction between the tools in use. As the Figure 1.2 indicates, the tools ADD, Microsoft Excel and EA work bidirectionally over the C#⁶ program. The information will be collected in C# and transferred over the Application programming Interface (API). This principle enables the usage of data read once with several tools, which increases efficiency and productivity.

As can be seen, the data flow to the text file only happens one way. This file contains the log information and some commands about the work procedures and processes of ADD tool. After a data exchange with ADD, the text file will be filled with some information, which will be explained in Section 3.1.3.

⁴<https://products.office.com/en-us/excel>

⁵https://desktop.avl.com/projects/12/0061/Data_Exchange/docs/AVLab/index.html

⁶<https://www.visualstudio.com/de-de/visual-studio-homepage-vs.aspx>

Chapter 2

Related Works

This chapter surveys and analyzes works related to this thesis. All the problems, solutions and distinctions of related works will be analyzed in cooperation to the approach of my project. Thereby the user will see that the related works have in some cases advantages and as well disadvantages versus my approach.

The work of clark [Cla09] describes the perspective of some models, such as V-Model and Dual-V Model for a System of Systems Engineering (SoSE) and Family of Systems Engineering (FoSE). This paper defines the terms: SoS, SoSE and FoSE from a V-Model and Dual-V Model perspective and their integration in a process. The terms V-Model and Dual-V Model are also mentioned and explained.

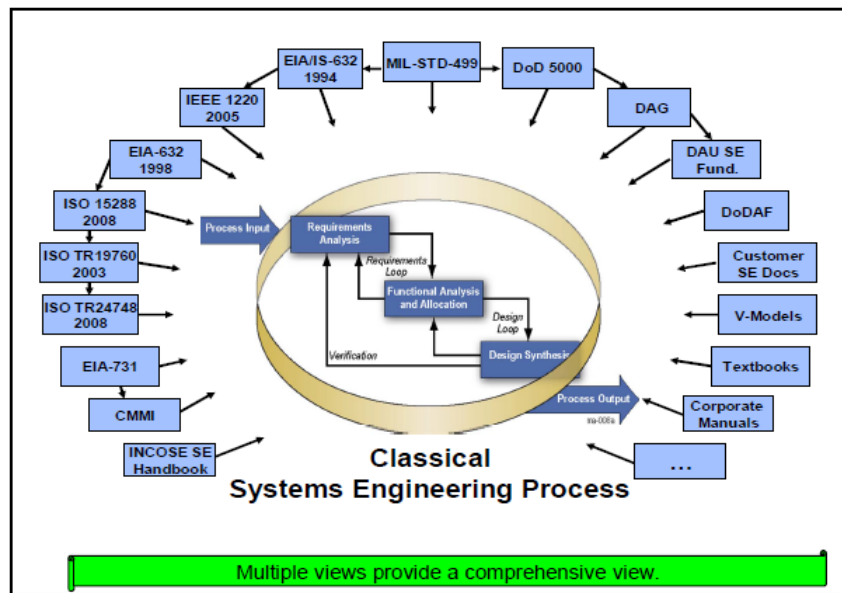


Figure 2.1: System Engineering views [Cla09]

The Figure 2.1 [Cla09] as a classical SE process for integration, reading and testing. The author also explains some other processes of the system. The usual SE views are

somewhat different to the view in Figure 2.1 [Cla09]. The V-Model is interesting section that relates to my thesis. As we know the V-Model is the most used software development process model in the automotive domain. The original V-Model is depicted in Figure 1.1, which begins at the top-left and ends at the top-right position. The integration and usage of the V-Model with a system or SoS is depicted in Figure 2.2 [Cla09].

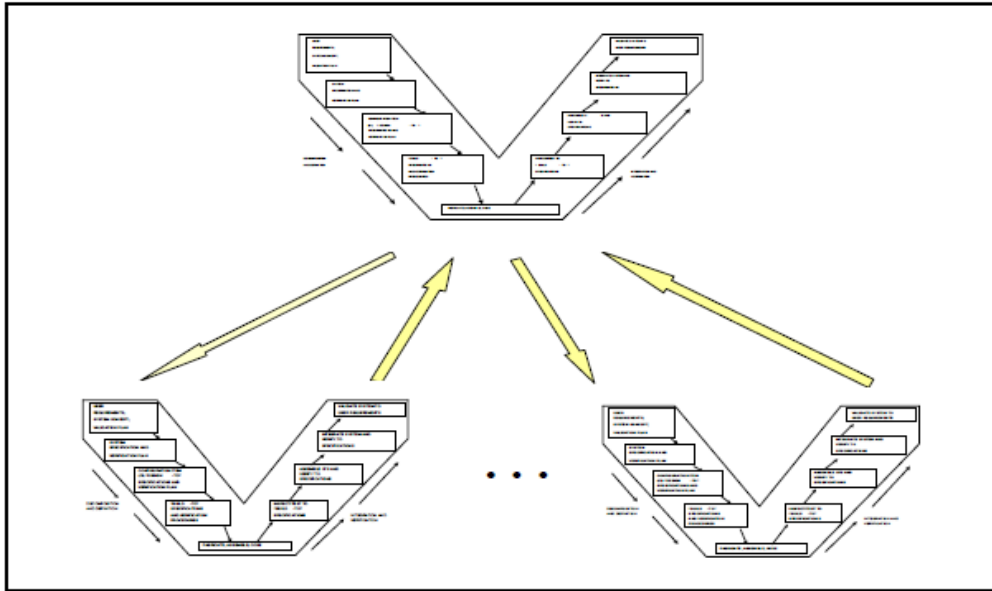


Figure 2.2: System or SoS V-Model [Cla09]

In this case a block of the classical V-Model is represented as another instance of the V-Model. This is done repetitive for each refinement of the system. The author makes process suggestions for a perfect, complete and seamless SoSE procedure. Furthermore he also recommends the application of V-Model, as well Dual-V-Model in SE, so that SoSE procedure is seamlessly applied.

This thesis covers the yellow arrows between the V-Models. This means the automated connection and data-exchange between V-Models is ensured by my tool which will be presented in this thesis. For example we can take the V-Model of EA and V-Model of AVLab and the automated data exchange between those tools has been made with my tool.

The next paper named *A Model Driven Architecture for Enterprise Application Integration (EAI)* [15706] describes several different approaches to solve the integration problem between different systems. This challenge is very complex and difficult to solve without losing any standards and skills. All of the analyzed approaches are compared and categorized according to their weakness and strengths. Additionally a new approach based on the OMG's Model Driven Architecture (MDA) is described. Object Management Group (OMG) standard is an unified object modeling standard, which focuses on visual modeling and model execution and service. This approach is first separated into the following five types of model:

- technology specific model,
- transaction service model,
- generic application service model,
- intra-application model,
- and inter-application model.

These five general types of model are depicted in Figure 2.3 [15706] along with some of their appropriated fields.

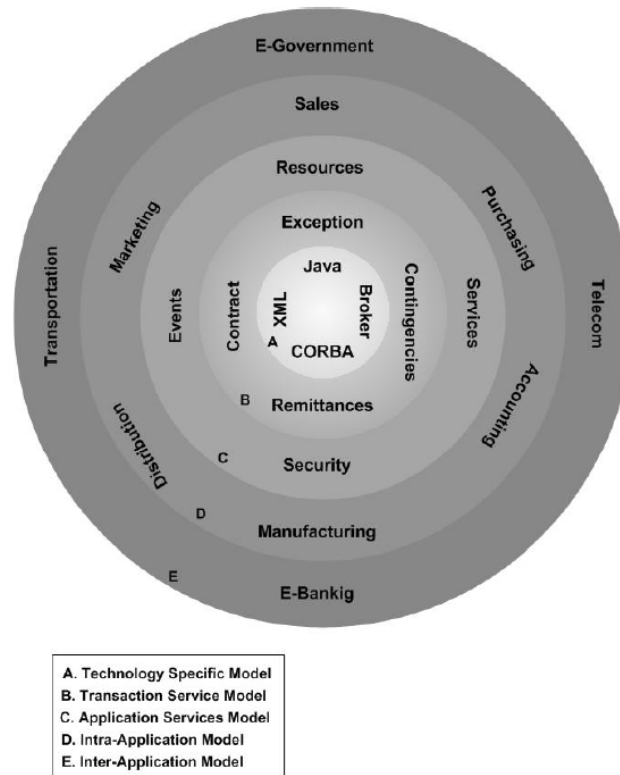


Figure 2.3: Model Driven EAI Architecture [15706]

In this paper [15706] all the available and new (yet to be evolved) infrastructures and mechanisms for different enterprise architect integration approaches will be studied in order to determine the best solution. After detecting, analyzing and discussing all the approaches to EAI, a separation into two main purpose-groups has been made: Category one focuses only on data and their processes and category two focuses on the structures. The same OMG MDA idea will also be used in this project, enable the integration of different models in a project. Not the structure of the data, but rather its correctness, completeness and consistency while exchanging the data is the main focus in this work. Due to this, the whole data transfer happens automatically in my thesis to prevent the human-fault factor.

The publication of Kronic, Letvencuk and Povazan et al. [KLPK13] describes an approach to model driven development and automatic source code generation of GUI controls. Source code generated from a model accelerates time-to-market, reduces development costs and provides a reliable development environment. Differences to our approach are, the CSV file based hardware configuration file and the development in eclipse IDE. This tool saves Special Function Registers(SFR) of a hardware configuration in several CSV files and represents those as GUI controls for the user. Based on this SFR list and GUI controls, some code will be automatically generated. The approaches overview is depicted in Figure 2.4 [KLPK13], which shows the operational sequence of the whole procedure.

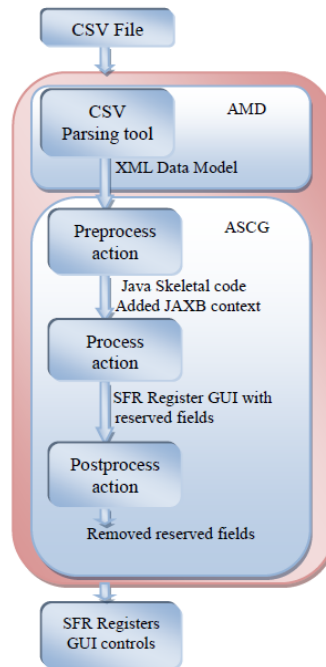


Figure 2.4: AMD and ASCG global view [KLPK13]

Their tool in [KLPK13] first reads a csv file which contains SFRs for a hardware configuration and ultimately represent those finally as GUI controls. In the middle the process is separated in two main parts:

- AMD (Automatic Model Development)
- ASCG (Automatic Source Code Generation)

The aim of this work is to generate source code in a fast, reliable and flexible way. The tool represented in this paper also generates source code from a model, but prepares the source code for another tools, so that these tools can read this generated information. The source code will be read from another tool and therefore ensures an information exchange between different tools.

The next paper [MKR06] describes a model-driven approach to Real-Time Operating System-based (RTOS) embedded software development. Additionally a tool called *TransPI* has been presented, which automatically generates a RTOS-specific C-Code from a model. This approach enables working at a high level without any occupation with hardware or software specification. In this work the Model-Driven architecture (MDA) depends on the following model-types: Platform Specific Model (PSM) and Platform independent Model(PIM). The first type contains the detailed implementation specifications, so that the model can be built again from this code. PIM is a model, which enables modeling with Unified Modeling Language (UML). The software development process happens in this paper automatically over an Application Programming Interface (API). This approach differs in the following points from my project: The tool *TransPI* generates RTOS specific code from a model. This also happens in my paper, however the data was prepared for another tool. Additionally this work only focuses on a model-driven approach to RTOS-based embedded software development and does not transfer SW architectures or application SW modules. But the information exchange or code generation happens in both variants over API and automatically.

Gerard H. Fisher et al. [Fis98] highlights on the importance of model-based systems engineering in automotive industries and describes the design of the automotive personal assistance system (APAS). This related work shows all the model-based processes required from the project creation to the project end. Those processes contain requirement- and behavior analysis, the physical architecture of the system, the verification and validation of the model. The aim is to achieve an approach for automotive system development so the product cost decreases and product quality increases. This idea is particularly applied to the design of the Automotive Personal Assistance System (APAS) to describe the features of the Global Positioning System (GPS). Finally the differences between this approach and the traditional paper-driven approach has been compared. The model-driven approach has the advantage of always being up to date; the traditional documentation can get lost or be

an old version of the document. This thesis has used the same concept because of those advantages. This approach helps the developer to understand the system more easier. With such an up-to-date model, modification and analysis is also very simple and easy to handle.

Monohar Rao et al. [Rao05] explains the importance and necessity of Unified Modeling Language (UML) within Artisan Studio. The author explains in their paper that the UML has fed through the software engineers to become a standardized visual modeling language. The biggest challenge was, as mentioned above, the integration of different systems in Artisan Studio, which makes traceability and usability more difficult. But with the release of UML 2.0 the system and software engineering integration becomes available with artisan studio. The integration of system and software engineering tools is currently not realizable, because of non published libraries by Atego. Research via internet about this topic leads mostly to impasses and gives insufficient results. For a sufficient API access, several libraries from from Atego are necessary. But this project is still not finished and a data exchange via API is planned in the future.

Farkas and Grund et al. [FG07] describe in their paper the rules, which need to be checked with a model-based development of embedded automotive software and safety-critical systems. They mention that a safety- and reliability-check that is performed after the system development is unprofessional and not possible in general. A safety and reliability check of custom rules and industry standards should begin in the early phases of V-Model [Cla09] and must be validated during the whole development process of a system. The paper describes a model based development approach of an automotive vehicle function with a state-flow diagram and MATLAB Simulink tool. It shows how to check the rules and meet the requirement standards during such a development. The technologies Meta Object Facility (MOF) for compiling the meta-models requirements and Object Constraint Language (OCL) are used to describe rules and to transform textual guidelines into formal notations. Finally, the work represents a solution which can check the rules for the formal and tool independent notations, as well as a solution for guideline checking. Tool dependent artifacts were modeled in a MOF meta-model and the transformation of textual guidelines were described in a formal notation with OCL. With such a technique, the prevention and avoidance of modeling errors is also ensured. Regrettably these techniques will not be used in this thesis and a verification and validation must be performed at the end of the process. Such techniques are of course a must for model-based development in automotive industry, but my thesis must also be kept within a limit. Therefore such rule checks would not be made to keep the required and limited time-period.

Trase and Fink et al. [TF14] describe a Model-Driven visualization tool used for Model-Based systems engineering (MBSE) projects. The reason for beginning the project of Trase and Fink is the complex and limited usability, as well as the complicated introduction and design process of available MBSE tools. Therefore the Systems Modeling Language (SysML) Document Traceability Framework (SDTF) for integrating design documentation with a system model has been developed. MBSE enables a clear design, which is easy to understand and additionally provides up-to-date design documentation. The mentioned SDTF helps to integrate existing design documentation within SysML model and another tool “Interactive Visualization Engine for SysML Tools“ called InVEST, generates an automated summary of the model content. Those tools have been principally developed for highly complex systems. Such systems usually have hundreds of design documents and verification reports, which are very sophisticated and extremely difficult to trace. This challenge can be solved by SDTF and InVEST tools. My thesis does not provide for this approach due to the low level of documentation and reporting. The generated documentation and reports for such tools are usually simple and compact.

Bringmann and Kraemer [BK08] mentioned a new model-based development trend in the automotive industry, which models software components in Matlab Simulink, State-mate, MatrixX, LabView and similar tools. Traditional software component development in C or assembler code is easy to verify and validate. With this trend of model-based testing new challenges which are often poorly supported appears. This paper particularly explains the characteristics of model-based development and its testing in automotive systems. Furthermore, the paper presents a test tool “TPT“ which determines the development complexity of model-based testing in the automotive industry. TPT has been developed by Daimler Software Technology Research and focuses on graphical test models. These graphical test models and simulations allow automotive engineers to detect failures and to determine a common functional understanding in early development and design phases. This model-based development approach improves communication within development teams and institutions and ensures that a system is complete and correct. To test such a model is mostly very tedious because of their content and structure. Automotive systems are very complex, so that the system consist of several software, hardware, electrical, mechanical and/or hydraulic parts. Such a system, where different teams and institutions are involved, can only be tested automatically. Manually testing is unfeasible, difficult to replicate and associated with very high costs. For a seamless system development following integration and test methods should be performed:

- **Model-in-the-Loop (MiL):** Is the first integration and test level. These tests evaluate section tests the whole model and its environment. The reason for this test is to detect system failures in early phases. This step also helps to improve the robustness, performance and re-usability of the system in the future.
- **Software-in-the-Loop (SiL):** Tests a software or software group within a simulated environment, but without any hardware components. This helps to detect weak points of an embedded system before its hardware is developed.

- **Processor-in-the-Loop (PiL):** The PiL phase is the most important test phase, because of the highest level of integration, which allows debugging during tests in a favorable way. During this phase the software is tested on a target platform. Faults, which are caused by the target compiler or by the processor architecture can therefore be detected.
- **Hardware-in-the-Loop (HiL):** In this phase, the software is tested on the final ECU. However the testing requires a real-time behavior to ensure that the system runs correctly in a real environment, which will be ensured with the test rig phase.
- **Test rig:** In this case, the software also runs on the final HW and the environment consists of physical components. This means the system is tested with real electrical, mechanical or/and hydraulic components.
- **Car:** The latest integration and testing level is performed in the car. If the testing in previous development phases has performed perfectly, the integration of the system in the car should be flawless.

The work in [BK08] presents a new test approach TPT called “Time Partition Testing“ (TPT), which supports a technique that allows systematic selection of test cases. It provides automated test execution and assessment for real-time environments. Furthermore, it generates a executable language for modeling test cases with a systematic test case selection. This test process of TPT can be seen in Figure 2.5 [BK08].

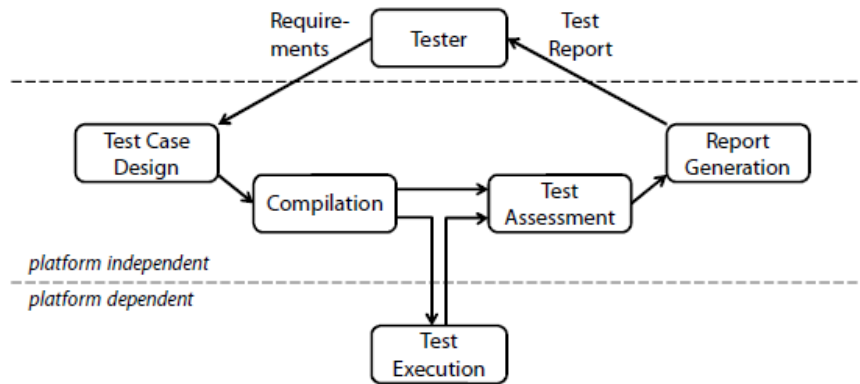


Figure 2.5: TPT test process [BK08]

Figure 2.5 shows, that the TPT test process is initiated by the tester and starts the „Test Case Design“ step. The functional system requirements with their design and test cases is the basis of this process. Next the test step “compilation“ will be presented, which represents test cases by byte code. Those byte codes will be interpreted and executed by a virtual machines (VM). This step ensures that the VM and test cases use as little memory as possible. In the next step “Test execution“, the VM executes the test case. The VM is also available to run other tests in parallel on different platforms. The second to last step assesses whether a test case was successfully or not. The final step is the report generation of test cases.

As mentioned, TPT is a test approach which supports design, execution, assessment and report generation of test cases in the automotive domain. Additionally TPT is reusable, portable and can be used in real-time for model based systems. The usage on test platforms: MiL, SiL or HiL is also supported. TPT additionally owns a precise and clear graphical language for a clean test case operation.

This paper gives the user a lot of information about model based design and testing in automotive industry. Furthermore, the detailed test case generation and execution with real samples has been explained, in a way that the same approach can be used for this project. TPT test process without “Tester“ step in Figure 2.5 is nearly the same as the AVLab-Tool. AVLab also performs processes like design, compilation and report generation. Furthermore, AVLab additionally features ADD, automated code generation, model test, among others. More details to AVLab will be presented in Section 3.3.

Farkas, Neumann and Hinnerichs [FNH09] describe in their paper an integrative approach for Embedded Software Design with UML and Simulink. Their paper first outlines the integration difficulties between C code and model based design. Due to the participation of different large institutions and teams, a seamless migration is nearly impossible. This problem can only be solved with well-defined integration concepts and methods. Such a migration concept for the modeling languages UML and Simulink is presented in this paper. Furthermore, the mentioned concept has been demonstrated in a traditional automotive software engineering process.

As mentioned, the embedded software development process is very complex and usually contains several thousand lines of embedded C code. On the other hand to analyze all the design aspects of an embedded software system within one modeling language is nearly impossible. To cover all the design aspects, more than one modeling language is usually applied. This paper analyzes the cooperation and usage of MATLAB/Simulink & UML for functional specification and code generation in embedded automotive subsystems briefly. Additionally the following points are discussed:

- Pro/Cons of embedded C code development using UML
- Problems of cooperation of MATLAB/Simulink and UML tools
- Consistency improvements of the UML, Simulink control design and CAN simulation environment
- Demonstration and testing of the approach on a BMW car door control device.

The mentioned integration process of a car control device is depicted in Figure 2.6 [FNH09]. As can be seen, the system developer develops a vehicle function based on existing artifacts and models. The integration of these artifacts is handled by an UML model.

Based on the UML model, automated code is generated in a way that the micro-controller in a car control device can work. Finally, the generated micro-controller code is executed on this device to test the code in real-life application.

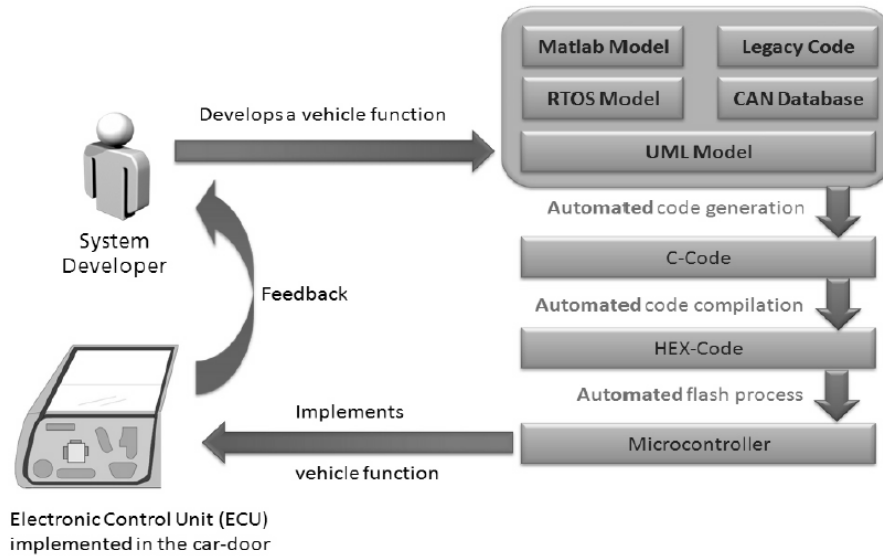


Figure 2.6: process of automated development steps [FNH09]

The paper frequently refers to the importance of automated model based code generation, which is becoming a popular technology. It is also very important to adapt and improve the architecture-oriented and object-based methods and functions in embedded software development. Thus the paper represents a well-defined model-based migration approach for the embedded software development. Furthermore, this development approach using UML and MATLAB/Simulink is explained step by step for code generation, simulation and flashing procedures in a real-life automotive control device.

My thesis also uses an integration approach for embedded automotive software development with UML and MATLAB/Simulink. But this SW integration approach has been made for the tools EA and AVLab.

Davey and Friedman et al. [DF07] describe software system engineering approaches with model-based design. As mentioned before, the model-based design and development becomes popular and complex. Parallel to these difficulties, the reduction of the system complexity and development time is a constant market need. The control and integration process of information from different large institutions and teams will raise new challenges and becomes nearly impossible to be managed by humans. Nowadays some luxury vehicles contain approximately 90 ECUs and consist of over six million lines of source code [DF07]. To solve this problem, model-based approaches will be used. The reason for the model-based development is the hardware-independent production and its favorable development costs. This advantage unfortunately brings some difficulties such as integration, testing and automated code generation of a model. This paper presents software systems engineering approaches to achieve all the desired features based on models. The following three main topics will be discussed in this paper:

- **Risk-based software management strategy**

This is a delivery strategy for large-scale embedded software solutions. The delivery process becomes critical when the developing time exceeds 12 months. The critical key processes in such huge projects are usually the supplier management process, the modeling process and the testing process. For a seamless development of these processes regular analysis and testing is necessary to identify and mitigate failures and incompleteness in early development phases.

- **Supplier software technical design reviews (TDR)**

This key area develops software development processes which are based on an automotive reference model. This reference model lists all the technical standards for an automated development process and will often be called CMMI/SPiCE(ISO15504) or ISO/ICE 15504. The developed and implemented software design processes will be reviewed by these standards. Additionally the following areas will be reviewed: planning, designing, documentation and testing. These constant reviews allow to detection and mitigation of open issues as early as possible.

- **Software systems engineering using risk-driven Model-Based Design for requirements validation**

This key area contains behavioral and implementation models with their complete testing environment. Model-based testing is often more effective than traditional test methods and concepts. It increases time-to-market factor and decreases development costs. The Figure 2.7 [DF07] shows the quality data of Model Based Systems Engineering(MBSE) versus other concepts.



Figure 2.7: Model-based systems engineering quality [DF07]

To sum up, it can be said that this paper supports a risk-based and structured concept for assigning software system engineering tools and resources in refined steps. Furthermore

it shows how the risk assessments arise and how to mitigate them with standardized automotive reference models.

The next paper named “A comprehensive Description of a Model-based, continuous Development Process for AUTOSAR Systems with integrated Quality Assurance“ [MLD⁺12] outlines the complexity of the automotive embedded systems. Keeping an overview of the whole system and properly handling each development step is extremely difficult. In addition to these requirements, planned standards and criteria must also be fulfilled. To tackle all these challenges in automotive industry, a mature technology and concept is required, which is already included in a V-Model.

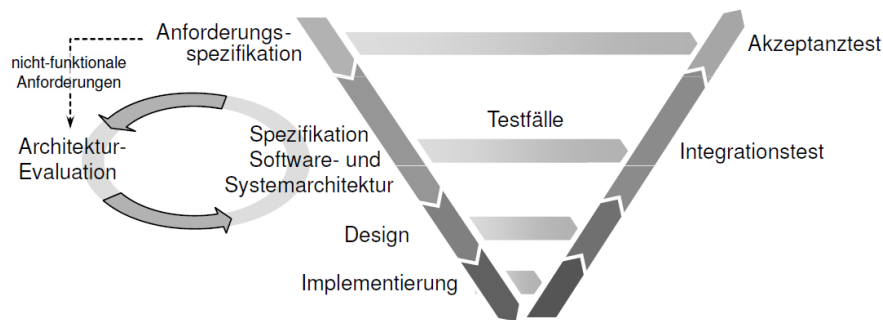


Figure 2.8: For a special case study modified V-Model [MLD⁺12]

Figure 2.8 [MLD⁺12] displays a modified V-Model for a special case study, which simultaneously keeps the AUTOSAR standards. As you can see the V-Model begins at requirement specification and goes down to the implementation of a product. However non-functional requirements must pass an architecture evaluation process. As usual the V-Model begins with analysis and definition of requirements. In parallel to this step a test case will be executed, which checks if the desired requirements are achieved or not. As you can see this V-Model differs from the usual V-Model in the second step, in “Specification SW- and system architecture“. In this case the mentioned SW- and system-architecture has been created and specified with a custom tool named “SystemDesk“ by dSpace. Subsequently this generated architecture has been evaluated and tested. After these steps design and implementation phases with their test cases. This example shows that own functions or concepts can be integrated into V-Model, but it must be kept in mind that those extensions should not violate the interfaces of other V-Model steps. Each used tool in my thesis works with the general known V-Model. That means the validation and testing of each process/product happens in its own environment. E.g. tools like EA, ADD, Atego test and validate their product in their own environment. And the connection between those V-Model/tools happens with my tool.

The next paper named „Automated Transformation of System Models into Ontologies“ [Sei14] describes the transformation of system models into ontologies. There are some technologies used like SysML, XMI and OWL for a standardized and efficient model design. This paper describes how to analyze and to consider a sysML model, so that this can be converted into an ontology. Nevertheless, the reason for including this work to related works is the automated model based development and working procedures within Artisan Studio from Atego. The analysis process of information and transformation into ontologies is not relevant for my thesis, but its automated read and write process with Artisan studio is of high importance.

This thesis [Sei14] takes an Artisan studio model and exports this as a XMI file. However, this file will be analyzed and converted into an ontology. The paper unfortunately uses the standard Atego resources and is not able to modify or create its own Artisan studio elements. Due to this, the developed tool of this work does not make any extensions in Artisan studio. It imports only an XMI model into Artisan studio, which is created by hand or reads a model and saves this as XMI file. As mentioned previously, the challenge in my thesis is to create customized elements and connections also in atego, which is not performed in this paper.

Chapter 3

Approach

This part of the document represents the data transformation between the involved tools and their attributes and properties. Additionally each tool in use will be presented below with a short description.

- **Sparx Systems Enterprise Architect and Atego Artisan Studio:**

Enterprise Architect (EA) is a visual modeling and design software tool based on unified Modeling Language (UML) which was developed by Sparx Systems. It supports the design, construction and modeling in different software systems and processes and allows visual depiction of artifacts, which are easy to understand and handle. Artisan Studio is another visual modeling and design software tool, like EA, based on UML and developed by Atego.

- **Microsoft Excel:**

Microsoft Excel also enables calculation, visualization and macro programming, which makes it well-known and wide-spread in many domains. Excel files consist of several work sheets, where each of them represents a huge table like databases of MBD tools. All the required information can therefore be imported/exported in/from those sheets, which include the same information as a model-driven system- or software engineering tool in another formats.

- **AVLab:**

The basis of AVLab is MATLAB/Simulink tool which was developed by AVL List GmbH company and it is a part of the AVL Powertrain Controls toolbox. It enables design in MATLAB/Simulink from model development to code generation. The reason of the development was the supporting Power Train Engineering (PTE) processes in AVL List GmbH. It also enables testing of models, using MATLAB utilities and communicating over an interface or integration of additional extensions and other features. The MATLAB/Simulink toolbox includes a large number of functions and tools, which are not relevant for this project. That's why the most relevant functions for this thesis will be presented in this paper.

Detailed information about AVLab can be seen in Section 3.3.1.

- **ADD:**

ADD stands for 'Automotive Data Dictionary' and is the interface to AVLab database, which was developed by Visu-IT company in cooperation with AVL List GmbH. From this database all the information will be transferred to other tools. This tool is separated into three software-levels:

- Projects
- Compositions
- Containers

All the SW-Level components are connected unidirectional and unique in the database. That means a project can consist of several compositions and a compositions can consist of several containers. A container is the lowest level in ADD and additionally includes all ports. The projects and compositions use the same ports like containers, however only containers contain their original instances of ports.

For more detailed information about ADD see Section 3.3.2.

All the above mentioned tools will be explained in detail and analyzed in the following sections.

3.1 Data Transfer between different tools

This project is divided into two parts:

- Export of the SW artifacts of the EA model
- Import the SW artifacts into the EA model

The planned import and export implementations with Atego Artisan Studio were not done in this thesis. Due to the limited access, non-existing library and minor references of Artisan Studio, this planned task could not be finished.

Research on the internet did not give sufficient information for an “Application Programming Interface” (API) of this tool. For successful API implementation, several libraries to access the operations, inputs, outputs and their objects would be required.

Artisan Studio unfortunately does not offer such a library, more specifically they does not want to disclose their codes to the programmer. A possible access is to import/export an “Extensible Markup Language” (XML) file in/from Artisan studio. Here the model will be interpreted as a xml file, which contains all the model components with their connections to each other. The problem here is, that the software modeling artifacts are customized profiles, which should work with customized components and connections. There are several project and company specified components with user-defined attributes and values (see Section 3.2 and Section 3.3). This means, that such a solution is also not suitable for our project aim and has been excluded.

3.1.1 Export of EA SW model representations

This section describes how to export the SW artifacts of the EA model to different tools, so that after a reimporting process, the same diagram can be reconstructed.

Two export methods are implemented:

- Exporting to the AVLab/ADD tool
- Or to a Excel file.

The principally work procedure of those mentioned functions will be explained in the following paragraph. By starting the C# project, the visual modeling and design software tool EA starts subsequently. Thereupon the user of this tool has to open an existing EA model or create one in EA. If an existing model is used, the export process can be started immediately. To export EA model to AVLab/ADD, the menu *Extensions/AVLab Extension/Export EA Model* must be selected in EA. In the next step, a new EA Tab/Window with the name *Export EA Model Tab* will be opened, which displays all the *AUTOSAR Components* contained. The EA Tab created is shown in Figure 3.1. In EA, there are several custom components defined for this thesis. All those components along with their

properties will be explained in Section 3.2. One of these components is the “AUTOSAR Component“. For exporting data, an “AUTOSAR Component“ must be selected by the tool-user. After selecting a component, this element and its their children can be exported to ADD or to an excel file.

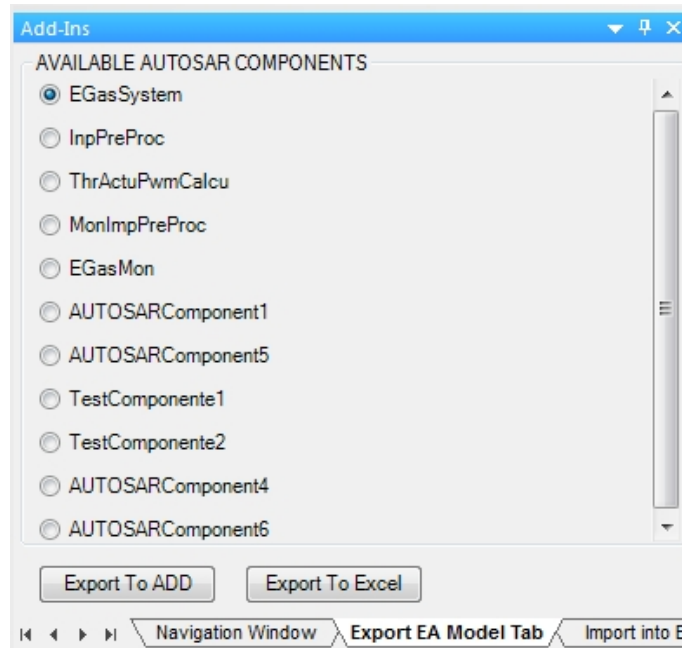


Figure 3.1: Export EA Model Tab

Export SW artifacts of the EA model to AVLab/ADD

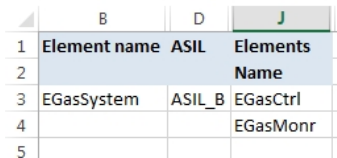
This section describes the export procedure of an EA model to ADD tool. As mentioned the procedure can be started in the EA menu: *Extensions/AVLab Extension/Export EA Model*. This command opens a new EA tab named *Export EA Model Tab*, which lists all the elements used of type “AUTOSAR Component“. By choosing “AUTOSAR Component“, the selected component it self and all its sub components with their properties and attributes will imported into ADD. This procedure is started by clicking on „Export To ADD“ button as you can see in figure 3.1. For more details read the tool user guide(see Appendix).

After successfully exporting an EA model to ADD, those exported elements can be immediately used and modified by AVLab or ADD. As mentioned the ADD is the database of the AVLab tool and means “Automotive Data Dictionary“. ADD works with three SW-Level and contains only unique entries. This means the exported elements must be unique and instanced.

Export EA Model to Excel File

This section describes how to export an EA model to an excel file. The exporting procedure begins as described in Section 3.1.1. The opened EA Tab *Export EA Model Tab* contains all the *AUTOSAR Components*. After selecting an *AUTOSAR Component* and pressing the button *Export to Excel*, the export process begins. The selected component itself and all its sub components with their properties and attributes will be exported to an excel File. Thereby the components will be saved separately in four following excel sheets:

- **Level1:** This excel sheet contains only the selected AUTOSAR component and its properties. At the beginning, some columns are hidden for a simple overview. This hidden structure of the excel sheets contains additionally all the necessary information for an exact regenerating of the EA Model (depicted in Figure 3.2). This level contains only the parent component and is named „projects“ in ADD.



	B	D	J
1	Element name	ASIL	Elements
2			Name
3	EGasSystem	ASIL_B	EGasCtrl
4			EGasMonr
5			

Figure 3.2: level1 worksheet with some hidden columns

- **Level2:** Level2 contains all the children of selected AUTOSAR component and their properties. The excel structure of the first three levels are the same, but contain different information. This SW level is the middle level and also contains hidden columns for an easy overview. This level is referred as „composition level“ in ADD.
- **Level3:** This Level is the last SW level and this worksheet contains the atomic components, beside ports, of the project. The components have no children and the excel worksheet structure is the same as in level1 and level2 worksheet. This worksheet component is equal to a „container“ in ADD.
- **Ports:** This worksheet contains all the ports and their properties, which were used for SW level1-3 components. The structure and contained data is depicted in Figure 3.4. Each entry of this sheet is the same as a „data objects“ in ADD. All ports are unique and only referenced if used by another component.

The excel approach has been used for more versatility of the approach and can also be used for reporting and documenting. In the following paragraph, each data structure used in excel and their attributes will be presented.

Figure 3.2 shows the initial state of the automatically filled excel file. This figure shows only the name, ASIL of the element and their children. However the Figure 3.3 gives distinctly more information than the Figure 3.2. With all this information a redesign of the EA model is possible. The columns like ASIL, characteristic, stereotyp.. etc. will be explained in Section 3.2. The first columns show the element name and id, which will be used in EA. The next column shows the type of the element. The columns “Package ID“, “Package Name“ and “Model Name“ are self explanatory and not necessary to explain. As

you can see all the excel worksheets information is very easy to understand and also easily traceable.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Element ID	Element name	Stereotype	ASIL	Characteristic	Package ID	Package Name	Model name	Elements		Ports	
2									ID	Name	Name	Direction
3	1962	EGasSystem	AUTOSARComponent	ASIL_B	AUTOSAR Composition	64	L4 - Structure	L4 - EGasSystem	1969	EGasCtrl	APedl1	in
4									1970	EGasMonr	APedl2	in
5											ThrActuEr	out
6											ThrActuPv	out
7											ThrPosn1	in
8											ThrPosn2	in

Figure 3.3: All Excel columns for *SW Level Components*

The next figure shows the worksheet at the ports, which also contains all the necessary properties of the ports. All the SW level worksheets additionally contain the two following columns: “Port Name“ and “Port Direction“. The reason to have these entries in the SW-Level worksheet is, that each port of the SW level components can have different directions. The rest information about the ports is always the same. Because of this, the SW level worksheets only contain both of those columns and the rest of the information about the ports is contained in worksheet “Ports“. Please take a look at Section 3.2 for more details about the column names.

	A	B	C	D	E	F	G	H	I	J	K
1	Port ID	Port name	Stereotype	DataType	DefaultValue	lowerLimit	PortType	ScalingLSB	ScalingOffset	SignalUnit	upperLimit
2	1965	APedl1	AUTOSARPort	float	0	0,5	AUTOSAR Sender Receiver	0,00195	0	V	4,5
3	1966	APedl2	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver	0,00195	0	V	5
4	1968	ThrActuEna	AUTOSARPort	bit	0	0	AUTOSAR Sender Receiver	0,00195	0		1
5	1967	ThrActuPwm	AUTOSARPort	uint8	0	0	AUTOSAR Sender Receiver	0,00195	0	%	100
6	1963	ThrPosn1	AUTOSARPort	float	0,5	0	AUTOSAR Sender Receiver	0,00195	0	V	4,5
7	1964	ThrPosn2	AUTOSARPort	float	0	1	AUTOSAR Sender Receiver	0,00195	0	V	4
8	1975	ThrActnPwm	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver	0.0019	0	%	100
9	1988	APedlPosn	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver		0	%	100
10	1989	ThrPosn	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver		0	%	100
11	1994	Trigger	AUTOSARPort	float	0		AUTOSAR Trigger		0		
12	2009	AccPosn	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver		0	%	100
13	2010	TPSPosn	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver		0	%	100
14	2011	PWMOutput	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver		0	%	100
15	2013	TPSEnable	AUTOSARPort	float	0	0	AUTOSAR Sender Receiver		0		1

Figure 3.4: All Excel columns for *SW Ports*

As seen, the information in worksheets are easy modifiable, adaptable and traceable. Additionally the user-friendly presentation of information allows a good overview of the whole system.

3.1.2 Import into EA model

This section describes how to import of model artifacts from different tools to EA can be done. The import process is divided into two methods:

- Import from AVLab/ADD into EA Model
- Import from Excel File into EA Model

Before importing data from a source tool to the target model, some preparations have to be made. The target model must be created or an available model has to be opened in EA. The user has to consider following rules, if an available model is to be opened. If EA model package contains the same elements like imported elements, these elements will be updated after a successful import procedure. If an update of those elements are not desired, then a new model has to be created in a new EA package.

After those necessary steps, the import procedure can begin by clicking on the EA menu *Extensions/AVLab Extension/Import into EA Model*. This command opens an EA Tab named “Import into EA Model Tab”, which contains all the elements of the source tool. After selecting an element, not only the project itself, but also their children will be imported.

Import from AVLab/ADD into EA Model

An AVLab project can also be imported in an EA model. By importing a project not only the project itself, but also its sub-components with their references and properties will be imported. The communication over the programming interface can be ensured with an AVLab library. The communication over the API reads all the elements of Automotive Data Dictionary (ADD) and lists this in an EA tab/window. As mentioned, by starting the C# application, also the tool EA starts automatically. The user then has to open or create a new EA model. All the ADD components can just be imported in this model. The menu *Extensions/AVLab Extension/Import into EA Model* is necessary. This step fills the new opened EA tab with ADD elements and represents each element type by a radio button. Next, the button “Import From ADD” must be pressed, a component type and a component must be selected. There are basically three components levels/types, which are available in ADD:

- Project (SW Level 0)
- Composition(SW Level 1)
- Container(SW Level 2)

More details about those levels can be read in Section 3.3.2. Finally, after importing a click on the button “Draw Model” draws the whole model in EA. A representation depicting this can be seen in Figure 3.5.

If the whole process is finished, a message box will appear which gives information about the finalization of the process. The ADD elements will just be drawn in EA with their connections and properties and is ready to be used or modified.

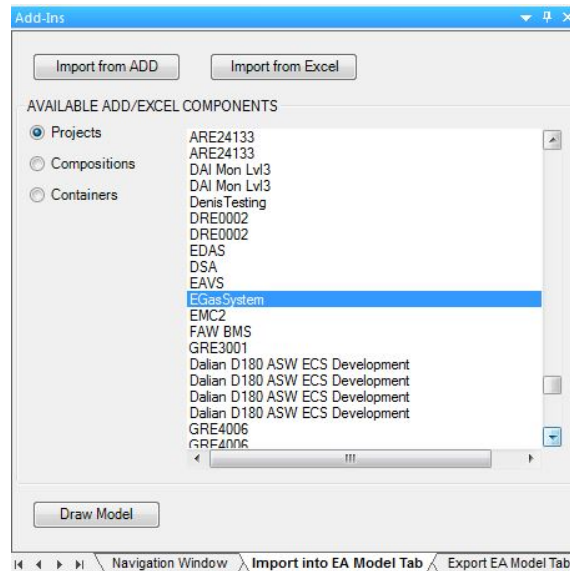


Figure 3.5: Import from ADD Tab with all ADD components

Import from Excel File into EA Model

This process imports an excel file, which contains a model description and structure into an EA model with all their connections and components. The data exchange between EA and Excel file happens again over the Application Programming Interface (API). This reads all the elements of the excel file and lists this in an EA Tab/Window. After the EA is automatically started and a model is chosen, the import process can begin. The menu *Extensions/AVLab Extension/Import into EA Model* opens a new tab in EA which is depicted in figure 3.6.

The user has to click on button “Import From Excel“. This step displays all the elements of the excel file in a list-box and all the element types represented by radio buttons. After choosing a SW level component and pressing the button “Draw Model“, the selected element and the sub-components of this component will be imported into the model with their references and properties. At the end, the model will be drawn in EA and a message box will appear, which gives information about the finalization of the process.

3.1.3 Write into Log File

The Log file is a text file, which logs all the data exchange processes. This file informs the user about the important process steps, gives a summary and some metrics about the imported/exported elements. Additionally warnings and errors will be listed below for clarification for the user. If an error occurs, the user can trace where and why the tool caught an error.

This log file contains at the beginning a minimal process description, like *Export EA Model to Excel File*. Next the selected component and project location will be logged:
Selected reference element: EGasSystem

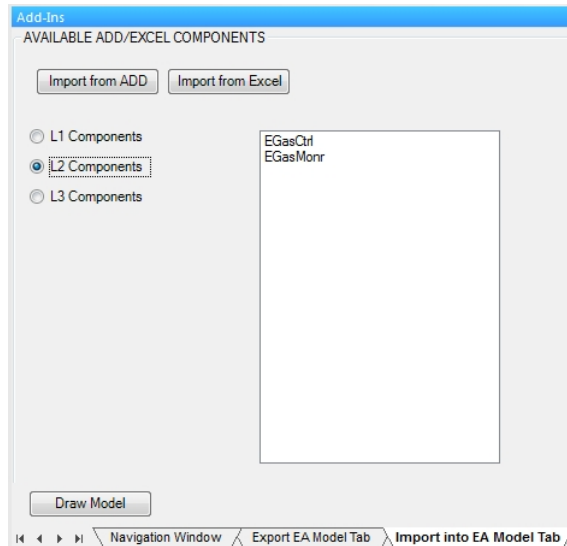


Figure 3.6: Import from Excel Tab with all Excel components

Project saved in folder: MasterThesis\bin\Testfolder

At the end of this file, some metrics and a summary will be logged. The metrics part contains, which component class has been exported or imported. An entry looks like this:

L2 Elements:

EGasCtrl

EGasMonr

The summary part mentions if the required process was successful or not. If the process was successfully, then a list of each imported or exported components will be presented. Additionally the number of components in use will be listed.

If the data exchange needs an ADD Tool, the log file contains also all the used ADD commands. Those lines first contain the definition of the ADD command beginning with two slash symbols. In the next paragraph, the executed command will be presented. A small section from this part looks like this:

```
[CREATE ADD PROJECT]
```

```
Set <IADDChangeRequest> changeRequests = ADDAdapterFactory.GetAdapterInstance().GetAllChangeRequests();
```

```
//public static IADDProjectID GetProjectIDInstance(string name,int variant,int release,int revision)
```

```
IADDProjectID projectID = ADDAdapterFactory.GetProjectIDInstance(EGasSystem, 1, 1, 0);
```

As mentioned, this file clarifies for the user detailed processes and gives metrics for the elements used.

Value	Description
QM	No special safety requirement
ASIL_A	Hazardous requirement with few harm
ASIL_B	More Hazardous then ASIL_A
ASIL_C	More Hazardous then ASIL_B
ASIL_D	Extreme hazardous requirement

Table 3.1: Available ASILs

3.2 EA SW Modeling Elements

This section lists all the used EA components with their relationships and properties. There are the main EA components:

- AUTOSAR Component
- AUTOSAR Port
- AUTOSAR Connector

These are the only relevant element for this project. Other components can also be integrated to this project, which helps to use this tool for other areas and purposes. The AUTOSAR components in this section describes the EA Elements of stereo-type *AUTOSARComponent*.

3.2.1 AUTOSAR Component

This self defined EA component has the following properties:

- ASIL
- Characteristic

For an automotive safety system the safety engineer has to detect and identify potential hazards of the system. This is also called hazard analysis and risk assessment. After identifying potential hazards, all of them have to be classified with a safety requirement level. This level is defined by ISO standard ISO26262 for road vehicles from A to D and means Automotive Safety Integrity Level (ASIL). All the non-safety relevant requirements will be classified as QM, which means *quality management*. The rest of the classification of requirements will be calculated by the severity (S), exposure(E) and control-ability(C) of the requirement. ASIL D means typically, that the requirement is very effective, hard to expose and hard to control. Requirements of this level will often be marked in red, which signalsizes a potential harmful hazard. The mentioned ASIL possibilities are listed in the Table 3.1.

Value	Description
AUTOSAR Composition	Is software hierarchy component and determines the SW architecture structure of the used components
AUTOSAR Application	Represents a single SW Module
AUTOSAR Sensor Actuator	Describes SW modules related to sensors and actuators, which are available at the ECU
AUTOSAR Parameter	Defines AUTOSAR parameter properties
AUTOSAR Complex Driver	Represents a complex device driver which has direct access to peripherals for a better performance
AUTOSAR Service	This manages the standard services like memory and flash management
AUTOSAR ECU Abstraction	Represents the abstraction type of the microcontroller abstraction layer
Trigger	Determines the trigger behavior of the ECU
BSW Interface	Determines Basic software interface properties

Table 3.2: Available characteristic properties

The first AUTOSAR component has already been handled. Next, the AUTOSAR Characteristic will be discussed. It defines behavior and properties of the used component. These can e.g. be trigger-, performance- and interface-behavior of the used ECU. Table 3.2 shows the possible entries for an AUTOSAR characteristic.

The below mentioned properties are selectable with a drop-down list, so that the user cannot give an incorrect value. An overview of these components can be seen in Figure 3.7. The background-color of this component is dependent from its ASIL value. It begins with white for the value QM and ends with red for the value ASIL_D, which helps to focus or visualize the most important properties for the user. Additionally an arrow in both directions signalsizes an interface for the property Characteristic (in that case BSW Interface).

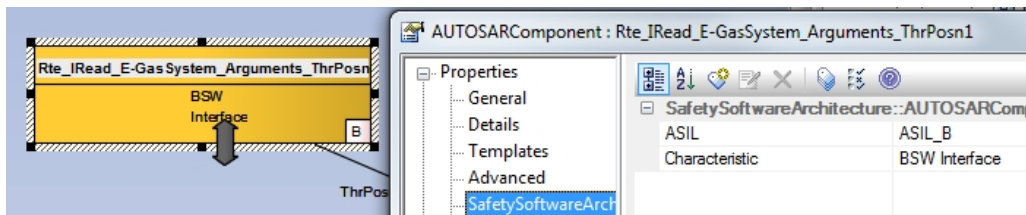


Figure 3.7: AUTOSAR Component with its properties

3.2.2 AUTOSAR Port

This EA component is a Port, which is an interface to outside or/and inside with the following properties:

- **Data Type:** Signal data type (int, float, char)
- **Default Value:** Initial or replacing value in case of signal loss
- **Direction:** Port direction (IN, OUT, INOUT)
- **Lower Limit:** Lowest allowed signal limit
- **Port Type:** Interface type of port
- **Scaling LSB:** Least Significant Bit of signal
- **Scaling Offset:** Offset value of signal
- **Signal Unit:** Unit of port signal (V, A)
- **Upper Limit:** Highest allowed signal limit

All of the mentioned items are saved in *Properties/SafetySoftwareArchitecture*. Some of the above mentioned properties are selectable with a drop-down list and some of them free selectable. All of these elements have a great number of properties, but only necessary properties will be explained in order to not distract from the main subject. An overview of AUTOSAR Port component with their properties can be seen in Figure 3.8.

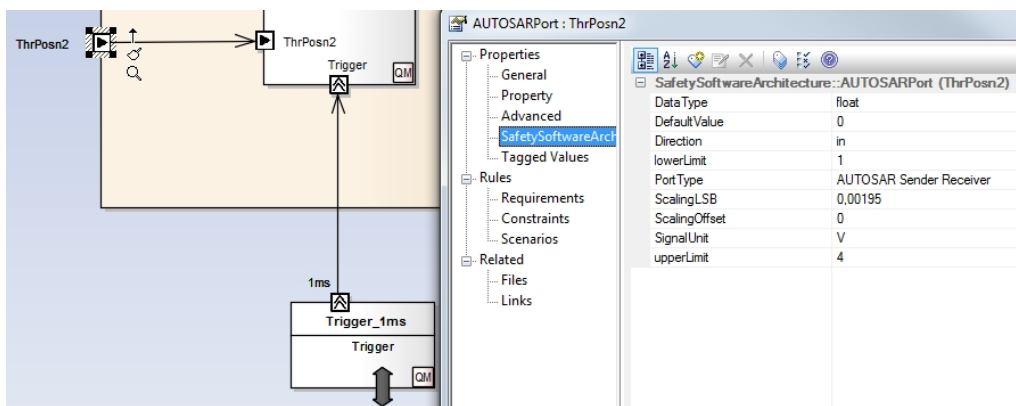


Figure 3.8: AUTOSAR Ports with its properties

3.2.3 AUTOSAR Connector

AUTOSAR connector is an EA element which connects AUTOSAR Port elements with each other. The properties of this element must not be filled by hand. The necessary properties are only source and target properties. Those will be filled up automatically, if connector from a source port to a target port is established.

An overview of a whole SW architecture can be seen in Figure 3.9 [EGa].

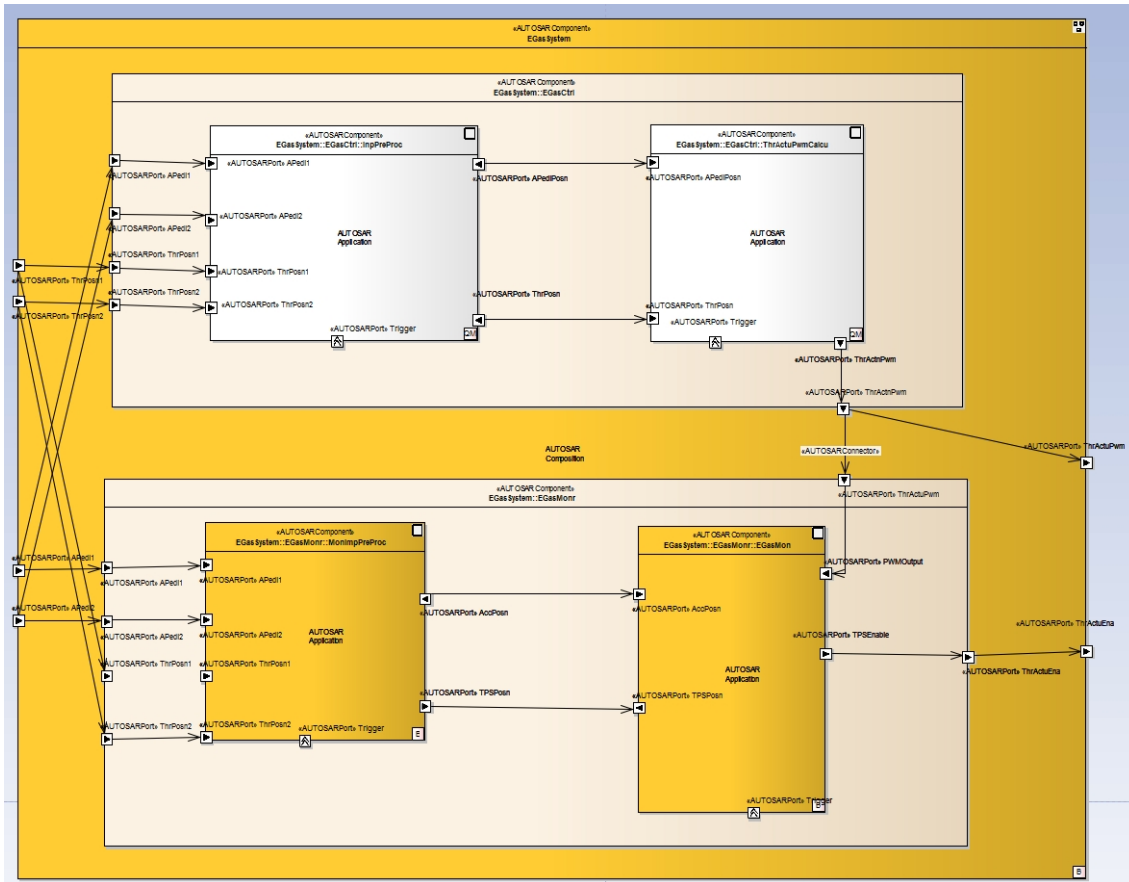


Figure 3.9: Whole EA Model

The whole Model is separated into three levels as you can see in Figure 3.9. Those levels will be called in EA, SW Level1 to SW Level3 and called in ADD Projects, Composition and Ports. The ports are saved in ADD as data objects and can be used in all levels. However in EA all the components can directly use/display required ports. Next the relationships between all EA components will be shown in Table 3.3.

Source Element	Destination Element	Multiplicity	Description
«AUTOSAR Component»	«AUTOSAR Component»	1 - *	Association between SW-L1, -L2 and -L3 Elements
	«AUTOSAR Port»	1 - *	Association between SW-L1, -L2, -L3 Elements and their ports

Table 3.3: Association rules between EA elements

*	Zero or more instances
1 - *	One or more instances

Table 3.4: Meaning if the association multiplicities

3.3 AVLab/ADD and its Components

AVLab is a customized tool, which supporting Model-based design in MATLAB/Simulink¹ in an AVL process aligned way. The main goal of this tool is to support different PTE development processes for increasing productivity and efficiency. Section 3.3.1 gives more detailed information about the AVLab tool.

The ADD tool serves as the database of AVLab, which contains all the used SW components of AVLab. For more details please read Section 3.3.2.

3.3.1 AVLab

AVLab is the AVL Powertrain Controls toolbox in MATLAB/Simulink. It provides model-based design and development in PTE processes and enables to design in MATLAB-Simulink from model development to MATLAB code generation. It also enables to test a model, to use MATLAB utilities and communication over an interface or integration of additionally extensions.

As mentioned AVLab is a function development and test environment tool in MATLAB/Simulink and includes following main features:

Model Development: The model development features covers following four parts: Model Template, synchronize tool, Simulink Toolbar and MiL test. The first part “Model Template“ takes AVL customized components and creates predetermined models with these components. Additionally a simulation of these models can be realized in MATLAB/Simulink simulation tool, which comes very close to the real behavior of the corresponding model. The second part “synchronize tool“ ensures the synchronization between AVLab/ADD elements with Matlab simulink model. This tool helps to be up-to-date in both development environments. The tool “Simulink Toolbar“ enables to work with MATLAB/Simulink tool. This enables the user an easy management and a clearly overview of

¹<http://de.mathworks.com/products/matlab/>

the whole system. Processes such as component linking, UI modification or redesigning are technically feasible. The last sub-tool “MiL Test“ is a MATLAB/Simulink test tool. This helps to test the model in the early development phases and so detect failures before they become a major problem. This step is key for a robust, quick and mature product.

An overview of AVLab with its links is depicted in Figure 3.10 on form of a Mindmap:

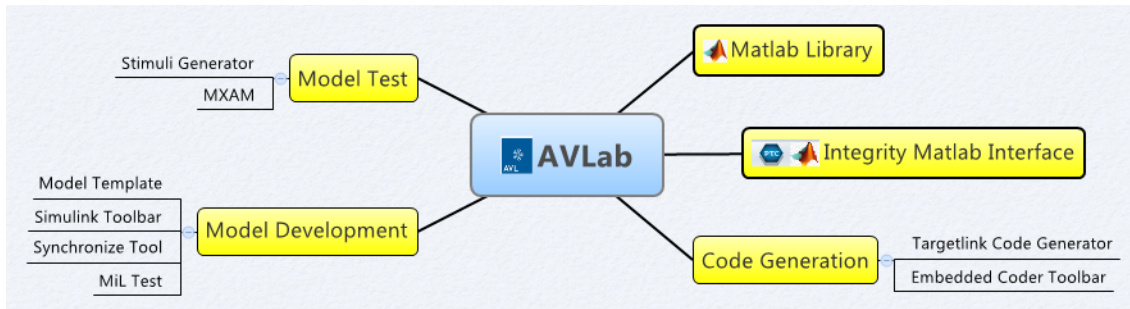


Figure 3.10: AVLab as Mindmap with links to related materials

Integrity Matlab Interface: This feature enables a data exchange between MATLAB and Integrity tool. At start of each project a requirement analysis and definition must be made. This complex process will usually be made with help of a requirement management tool. One of these tools is the “Integrity“, which enables the management the system and software requirements of a project. This requirement tool supports the entire development process of the product. Therefore, data exchange and synchronization is even more important.

Matlab Library: This feature takes advantage by the usage of MATLAB utilities and libraries. Some of those advantages are function plotting and visualization, interface to different tools, Simulink utilities, simple usage of vectors, functions, structures and matrices.

Code Generation: Is a software from the dSPACE² company named „Targetlink“, which is able to generate automated code based on a MATLAB/Simulink models. This feature helps to design and test a model in a simple way. After generating a code, the source code can be simulated and tested with the following concepts: **MiL** (Model in the Loop), **SiL** (Software in the Loop) and **PiL** (Processor in the Loop).

Model Test: The last main feature is the „Model Testing“ within AVLab. The first tool named „Stimuli Generator“ supports open, load or modify procedures for a simulink model behavior impulses. This helps to detect the reason a failure occurred in simulation or can be used for a detailed review of the model. Additionally such a stimuli can be saved for comparison with other measurements. The tool „MXAM“, which also known as „MES MXAM integration tool“ enables the MATLAB/Simulink model checking against standards and constraints.

The depicted mindmap contains an AVLab session object as focus/main object. AVLab

²<https://www.dspace.com/de/gmb/home.cfm>

session includes all the required data for AVLab and PoET². PoET² is the basis of AVLab. In the following the necessary parts and functions of ADD will be briefly described here, otherwise the information about AVLab will be confusing and distract the reader from the main subject of this project.

Main Functions of ADD:

- Visu-IT³ ADD \iff Simulink
Data synchronization between Matlab Simulink and ADD database.
- PTC Integrity⁴ \iff MATLAB
Integrates PTC Integrity in MATLAB and MATLAB/Simulink. PTC Integrity is a tool, which manages system and software requirements. Validation and verification of those requirements are also possible.
- Matlab/Simulink \iff AVL Concerto⁵
Helps to visualize AVL Concerto plots in MATLAB/Simulink. AVL Concerto is an AVL tool, which evaluates and graphically visualizes data graphically.
- MIL/SIL/Back2Back
Supports testing with Model In the Loop (MIL), Software In the Loop (SIL) and Back2Back methods. Back2Back test method compares different model versions and their results, which are typically separated in three different types: Float vs Fixed, Float vs SiL and Fixed vs SiL.
- Code Generation
The main code generation tools are TargetLink and EmbeddedCoder. These sub tools of AVLab helps to generate C-Code from a single or several MATLAB/Simulink models. Additionally an automatically predefined string replacement of generated code is possible.

3.3.2 ADD

ADD stands for Automotive Data Dictionary, is the interface to the AVLab database(oracle) and manages all model and code variables, which were used during the whole development process. This enables a clear separation between variables and implementation and ensures the use of unique variables. This independent storage accelerates the development process of a system due to parallel working on the implementation and declaration part. The ADD data will be used for documentation, simulation, calibration and SW development processes, which are managed by AVLab.

ADD is structured in three SW-Levels and which will be listed below.

¹⁰⁰³https://desktop.avl.com/projects/12/0061/Data_Exchange/docs/AVLab/_downloads/AVLab_Platform.xmind

³http://www.visu-it.com/index_de.php

⁴<http://de.ptc.com/product/integrity>

⁵<https://www.avl.com/-/avl-concerto-data-post-processing>

ADD Components

Figure 3.11 shows the Graphical User Interface (GUI) of ADD tool. As mentioned SW-Levels in ADD are separated in three layers, however the smallest SW module additionally includes all the used ports.

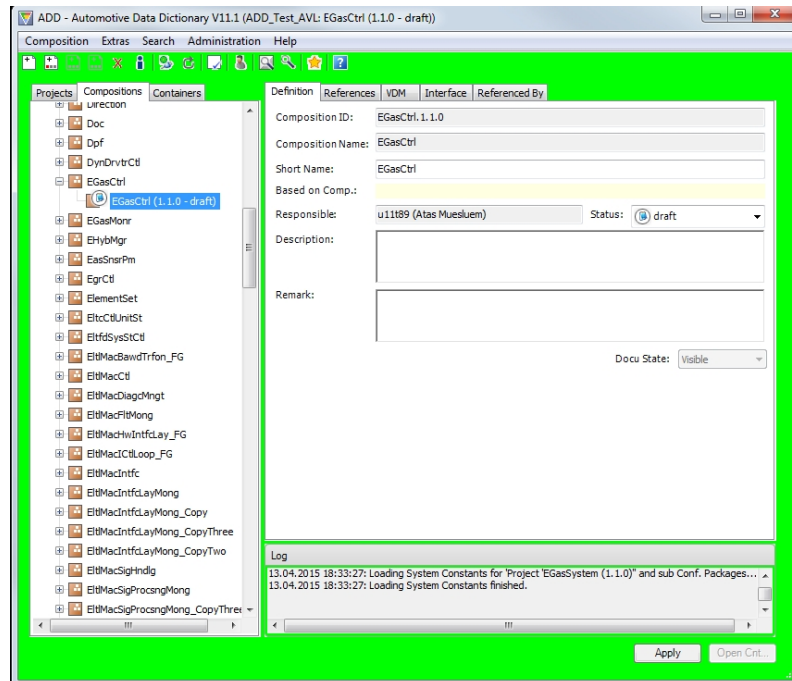


Figure 3.11: ADD user interface

The hierarchy structure of ADD can be seen in Figure 3.12. The SW-Levels represent projects, compositions and containers and SW ports, the data objects in ADD. Only one project with their sub components are represented here. As depicted, a component can contain different sub-components, whereby different sub-components can also belong to several parent components. Each entity in ADD is unique, but can be used several times in different models. This means only instances of each component will be used in a model. A modification of a component itself, leads to a modification of all the used instances. This modification risk can lead to an unintended component and model modification in undesired models. For this reason, modifications of components that are used multiple times should be prevented. Instead, the user should create a new component, if changes are required.

- **Project**

This project representation is the highest level of software split, which is also called SW Level1 and is a collection of several compositions, whereby a composition is also a collection of containers.

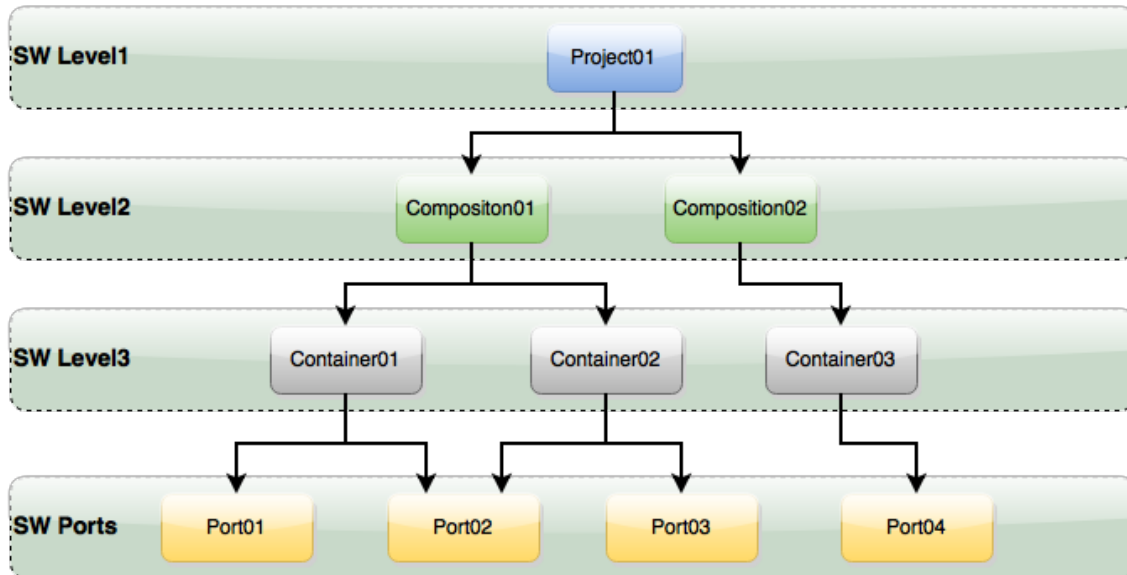


Figure 3.12: ADD Components

- **Composition**

Composition is the middle SW Level of ADD elements and can contain several containers and is also called SW Level2. This abstraction level is a collection of containers, which are again a collection of data objects.

- **Container**

This level will known as SW Level3 and is the lowest level of abstraction in ADD. It consists of available ports (data objects) and is the smallest powertrain software architecture entity. The ports of all three levels will only be declared here.

- **Data Object**

A data object declares a variable with all the necessary attributes. It consists of variable attributes and properties like:

- Type: Axis, cDefine, Online, Map, parameter
- Status: Draft, fixed, undefined,..
- Name
- Description
- Classification: Input, output, Local
- Base type: float, uint, boolean,...
- Conversion: Conversion of variables like string,

Chapter 4

Implementation

This chapter describes the implementation part of the thesis. The tool is implemented in object-oriented programming language C# with an integrated development environment (*Microsoft Visual Studio* from Microsoft). For versatility regarding computing platforms, such as windows API, Windows Store and Windows Forms have been used. Additionally the development tool „Microsoft Visual Studio“ contains an integrated source level and a machine level debugger for a simple implementation and testing.

The tool has been developed using a Graphical User Interface(GUI) application. This Windows Forms designer includes GUI controls, like buttons, list-boxes, text-boxes...etc. Thus the approach represents commands, programs and some other options as visual elements, so that programmers can easily develop new programs or tools. This design makes the tool more attractive, user-friendly and helps to make a professional impression.

The flow chart 4.1 shows the execution sequence of the implemented code. After starting the tool, the program Enterprise Architect(EA) from Sparx Systems starts automatically. For the control and management of EA, an EA library is needed in Microsoft Visual Studio project. The Library „MDS.EnterpriseArchitect.Library“ should be imported and used in the C# project. This library enables the creation of new windows, tabs, menus and some other extensions in EA. If the EA starts correctly, the whole control of the system will be kept by EA tool. All the export and import procedures are available through the customized menus of EA. After selecting one of these menus, the whole data exchange and control will be done via the Application Programming Interface (API).

If EA is started, an available or new diagram has to be created and opened. After opening a diagram, the customized sub-menu called *AVLab Extension* can be used via the *Extensions*-menu in EA. In addition this, the menu contains two other sub-menus:

- *Export EA Model*
- *Import into EA Model*

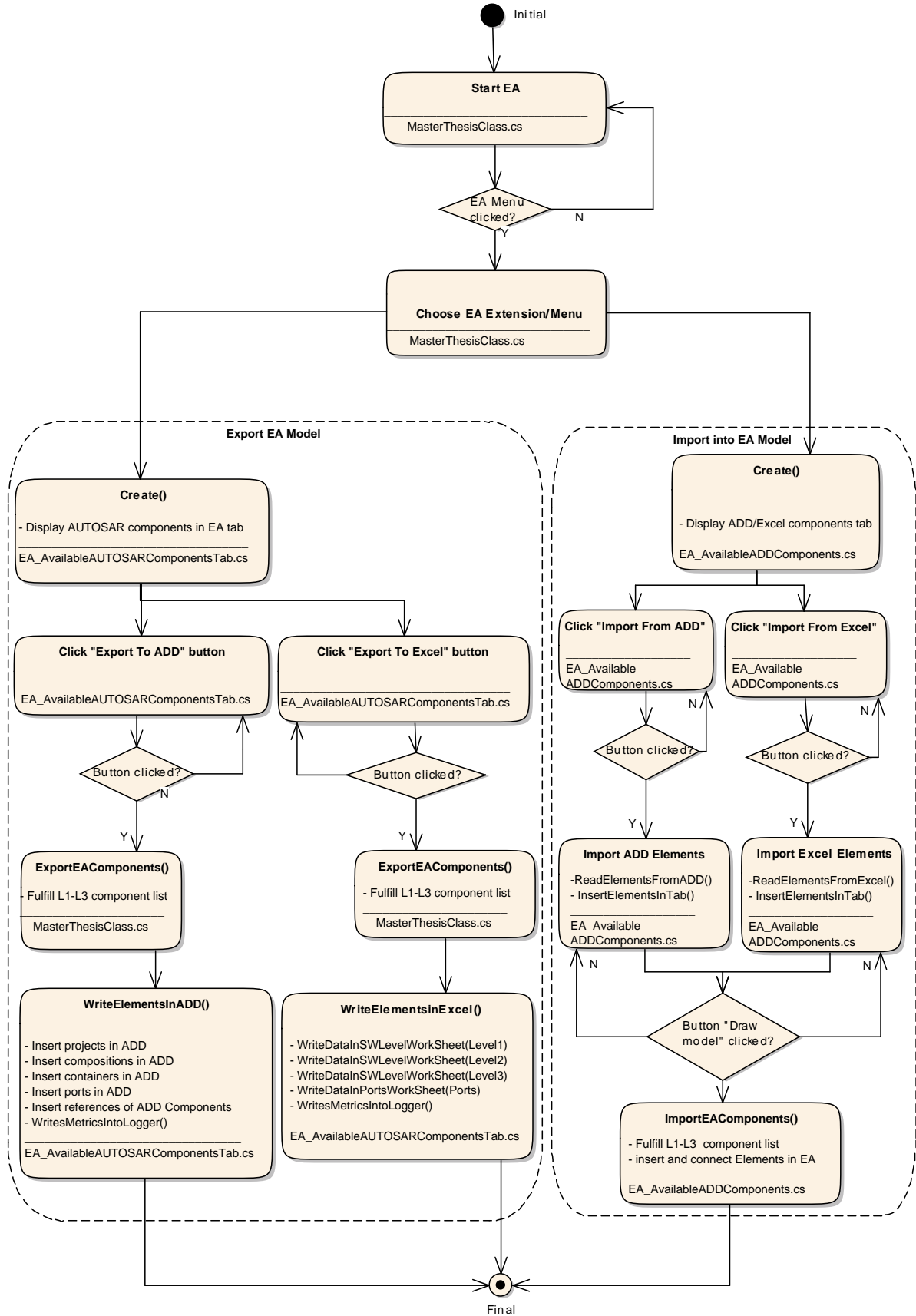


Figure 4.1: Program execution path

By clicking one of these menus, the process will begin and one of the execution path in Figure 4.1 will be executed. A click on the *Extensions/AVLab Extension/Export EA Model*-menu causes an execution of the left path in Figure 4.1. The occurring events and further steps will be explained in Section 4.1.

The other extension *Extensions/AVLab Extension/Import into EA Model* executes the right path of the Figure 4.1. All detailed steps will be explained in Section 4.2.

4.1 Export EA Model

A click on the *Extensions/AVLab Extension/Export EA Model*-menu creates at first an EA Tab named “Export EA Model Tab“ which lists all the EA elements of the type “AUTOSARComponent“. After selecting one of the “AUTOSARComponent“ from the opened model, the target tool for the data transfer can be chosen. There are two buttons displayed in the opened EA-Tab „Export To ADD“ and „Export To Excel“. Additionally each displayed “AUTOSARComponent“ will be represented by a radio button. After choosing an “AUTOSARComponent“ which is represented by a radio-button and clicking one of the buttons, the whole data transfer begins.

4.1.1 Export To ADD

By pressing the button “Export To ADD“, the selected element, its sub components and ports will be transferred to ADD. This procedure works in two main parts. First all elements will be saved in temporary lists. These contain all the SW Level1-3 elements, their ports and their attributes and properties. This step is implemented in *ExportEAComponents()* function, which is located in *MasterthesisClass.cs* file. The reason for the temporary lists is the prevention of permanent read processes of the EA elements. This has the advantage that the read procedure happens once and the write procedure can be repeated several times. This further means the tool can write the non-recurring reading data in different tools. This accelerates the execution time of the tool and prevents repetitive cycles. For a successful connection to ADD, some tools and properties are pre-requisites. One of those is the establishment of a connection to the Automotive Data Dictionary (ADD) tool. The preparation for a successfully ADD connection can be done by the company Visu-IT and AVL, which are the developer of this product. For the establishment of ADD, the program *oracle client* must be available and installed on the computer. In addition some admin rights changes are necessary, to prepare the environment for the ADD tool. After a successful preparation and establishment of ADD, the first connection to ADD elements via API can be established. As mentioned the whole data exchange base on the application programming interface (API) happens automatically.

In the next phase all the elements will be imported into the ADD tool. Parallel to this step, a logger file is written with the single instructions performed. The second phase is

implemented in *WriteElementsInADD()* function, which is located in *EA_AvailableAUTOSARComponentsTab.cs* file. At first all the SW-Level1 elements will be generated as ADD Projects in ADD tool. If the element is already available, only a data update is necessary, otherwise a new project will be created. The same rule shall apply for compositions, containers and ports. The SW Level2 elements will be generated in the same way as compositions in ADD. After inserting SW Level3 elements as containers, the ports can be included into ADD. Ports of SW Level1-3 are inserted at once as *Data Objects* in this database. After inserting all the projects, compositions and containers, the connection between these element must be established. This step connects all the ADD elements with each other, so that the links between the elements are not lost and a reconstruction is possible.

In parallel to these steps, a log file is generated. This file contains all the steps, as well as process warnings and errors, so that the user can track the steps and find solution if a problem occurs. Detailed information about this file can be read in Section 3.1.3. Finally the connection to the ADD tool will be closed and data in ADD is ready to be synchronized with AVLab. After finishing the whole process, a summary and some metrics of the exported elements are listed in the log file.

4.1.2 Export To Excel

By pressing the button “Export To Excel“, the selected element, its sub components and ports will be exported into an excel file. First a file dialog window is opened, where the user has to select a Microsoft Office Excel file as destination. Afterwards all the information will be transferred to excel file, as in Section 4.1.1. This working method enables us an efficient and accelerated operation of a system.

If the selected excel file is new, the file will be filled with new data, else all the data in the excel file will be overwritten with the new information.

After this process, the new excel worksheets will be prepared and filled with new data. Each SW Level is represented as a separated excel worksheet. Moreover the used ports are listed in their own excel worksheet. There are also four worksheets: three for SW levels and one for the ports. Please read Section 3.1.1 for more detailed information about the excel file data and its structure. After finalizing this step, a message box will appear, which signals that the whole process is finished. In addition, a summary of metrics, warnings and errors that occurred will be listed in a log file.

4.2 Import to EA Model

The working method of „Export EA Model“ and its implementation has already been explained in Section 4.1. Next the reverse method “Import elements to EA Model“ will be explained. This process starts with a click on *Extensions/ AVLab Extension/Import into EA Model*-menu in EA. It causes an execution of the right path of the Figure 4.1. It first creates a new EA Tab named “Import into EA Model Tab“, which can be seen in Figure 3.6. This tab contains two buttons in the header, an empty list box at the initial state in the middle and a single button in the footer. The buttons in the header determine the source tool, also where the source data comes from. The first button called “Import from ADD“ imports ADD elements and prints all the element types represented by radio buttons in the left-middle position of the tab. More details and its detailed work procedure will be explained in Section 4.2.1.

A click on another button („Import From Excel“) imports the data from an Excel file, which will be explained in Section 4.2.2.

4.2.1 Import From ADD

The „Import from ADD“ button imports all the ADD elements and lists these on the opened EA-Tab. These elements will be differentiated by their types. There are following element types available in ADD:

- Project
- Composition
- Container
- Data Object

These types have been already explained in section 3.3.2. „Data Objects“ in ADD represents EA ports and will be used by SW Level components. This means selecting a port and drawing this in an EA Model makes no sense without the SW Level elements. For this reason, only the first three types are interesting for us. Those three types will be displayed on EA-Tab and are usually represented by radio-buttons. A click on a radio-button (element types) lists all the ADD components of this type. Furthermore, a click on *Project* radio-button lists all the available projects of ADD. This list is located in the right-middle position of the opened tab. After selecting an element type and an element, the import procedure can begin. The last button in the footer named “Draw Model“ inserts all the elements in the opened model and displays them, so that all elements are completely connected to each other. Only the selected element, its sub components and ports will be imported and drawn in the opened EA model. The rest of the data will no longer be used and is uninteresting for the user. As usual, the availability of the imported elements will be again checked. Available elements will be updated and not available elements will be newly created in the EA model. This means if an EA element is used several times, an update of this element causes a modification all of the referenced elements. Therefore attention

should be paid. After graphically representing the model, a message box will appeared, which signals the finishing of the whole process. Please read Section 3.2 for more detailed information about the EA model and its components.

As mentioned in previous sections, a log file will be updated with useful information. This contains a summary, some metrics, warnings and errors that occurred during the process.

4.2.2 Import From Excel

A click on the „Import from ADD“ button imports all the Excel elements and lists these elements in the opened EA-Tab. At first, the data from all four excel worksheets will be saved in temporary lists to prevent a permanent access to the file. A permanent file access would make the imports process slow, because of the continuous file open and close operations. Therefore each worksheet will opened and read once at the beginning. These elements will be differentiated again by the following types:

- Project
- Composition
- Container
- Data Object

As in the previously discussed sub-section, only the first three types will be represented by radio-buttons and a click on such a radio-button lists all the elements of this type. To import and draw elements in a model, the following steps are necessary: selection of an element type, selection of an element and pressing the button in the footer. If the user clicks one of the element types which are represented as radio buttons, the left-middle positioning list box will be filled with the corresponding elements. After a click on the “Draw Model“ button, the whole importing process will be started. The selected element, their sub components, ports and connections will be imported in the opened EA model. As mentioned, the updating process of the available elements must be made with attention. Otherwise an update of multi-used element causes a modification of all the referenced elements.

After a successful process, a message box appears, which signals that the whole process is finished. Please read Section 3.1.1 for more detailed information about the Excel file and its worksheets.

Again a log file will be constantly updated with useful information containing a summary, some metrics, warnings and errors during the whole process.

4.3 Available Application scenarios

This section presents all the available application scenarios for the developed tool. There are principally three tools used for data management/exchange: Excel, EA and AVLab/ADD. The log file will not be mentioned here as it only logs the steps that are performed and it is not used for data-exchange or -management. The main goal and reason for starting this project was to ensure the automated data exchange between EA and ADD. Therefore these application scenarios will be handled first.

- **Data transfer from EA to ADD:**

This application scenario takes an EA model with its elements and imports them into the database of AVLab (ADD). The automated concept in this case is complex and consumes a lot of time. The higher the number of the transferred EA model, the better the automated approach is compared to the normal version. Therefore, it only makes sense, if the same operation is repeated several times.

After importing of all the elements with their properties and attributes in ADD, they can be used in AVLab after a database synchronization. The application scenario automatically creates the low level design in MATLAB/Simulink from a high level model with less expense or effort. The imported elements in ADD are also already connected with each other. This means, after a database synchronization in AVLab, the low level model can be drawn with a single click.

- **Data transfer from EA to Excel:**

The elements are in this case imported into a Microsoft Office Excel file. The elements in the Excel file will not be used by another tool, but they will be used for clarification regarding the imported elements and their properties. The Excel file will also be used like a report file in this case.

- **Data transfer from ADD to EA:**

This application scenario takes an ADD component with its connections and children and transfers it into an EA model. First, the desired EA elements with their properties will be inserted into the EA package. Then, the elements will be connected to each other. This means the child elements will be inserted into their parent element and their ports will be connected to each other. After this step, the model can be used or modified by another engineers/domains/tools.

- **Data transfer from ADD to Excel:**

All required ADD elements will be transferred to an Excel file. Depending on their software levels the elements will be saved in different Excel worksheets to ensure a clear separation. Additionally all available properties and connection information for these elements will be saved in the target file. As mentioned above, the Excel file will be used as a report file to provide clarification regarding the imported elements and their properties. In other words, this file is a hard copy of the imported ADD elements in another file format.

- **Data transfer from Excel to EA:**

This application scenario takes all the available Excel elements and imports them into an EA Model. The user can add, remove or modify all the excel file information,

but must ensure consistent naming of worksheets and worksheet structing. Furthermore, element properties such as *Element ID* must not be modified or deleted. The entered *element ID* should be available in EA, otherwise an error will occur and the information cannot be imported. After importing of information to EA, the model can again be used immediately or modified for another purpose.

- **Data transfer from Excel to ADD:**

This step is not directly possible. The only possibility is to import the Excel data to an EA model and then transfer it to ADD. The data transformation from Excel to an EA model and from an EA model to ADD has already been explained in previous sections.

One of these application scenarios will be studied in detail and explained in Section 5.2. Additionally a direct comparison between manual and automated development will be discussed based on the consumed energy, time and effort.

Chapter 5

Application

5.1 Application

This section of the thesis describes the intent and purpose of the developed tool and additionally lists its application areas. The tool has been developed for the companies AVL List GmbH and the Technical University of Graz. It will be used in automotive industries to accelerate the work processes and shorten development time. It enables engineers to view different descriptions and perspectives of a process, which leads to a substantial improvement in the development process. Thus enabling the detection and avoidance of basic errors in early development steps.

The implementation concept and skills used have already been discussed in Section 4. All the achieved goals and limitations will be discussed in the following sections. In addition the usefulness and reliability of the extensions and features will be discussed and analyzed in the last sections.

5.2 Case Study

This section focuses on one of the six available application scenarios of this tool. The chosen scenario is the „Data transfer from an EA model into ADD“. This application scenario exports an SW Architecture from EA to the database of AVLab (ADD). There is one SW-Level1, two SW-Level2, four SW-Level3 elements and fourteen ports to export. This scenario automatically creates the low level design in MATLAB/Simulink from a high level model with less expense or effort. Furthermore, the imported elements in ADD are already connected with each other, so that a database synchronization with AVLab leads to a complete low level model.

The comparison between my approach and a manual approach will be analyzed in detail and discussed based on the following areas:

- **Consumed time:**

This part of the document shows the time consumed to import an EA model into ADD procedure. It should be kept in mind, that the manual working steps are of course user-dependent and can vary. Due to this fact variations of the measured times can occur.

There is one SW-Level1, two SW-Level2, four SW-Level3 elements and fourteen ports to import. All of those have several and different properties (see 3.2). As you can see in Table 5.1, there is a huge difference in the time consumption between the automated and manual working steps.

The time consumption can also be seen in Figure 5.1 as a chart, which illustrates the time differences more representative than the table.

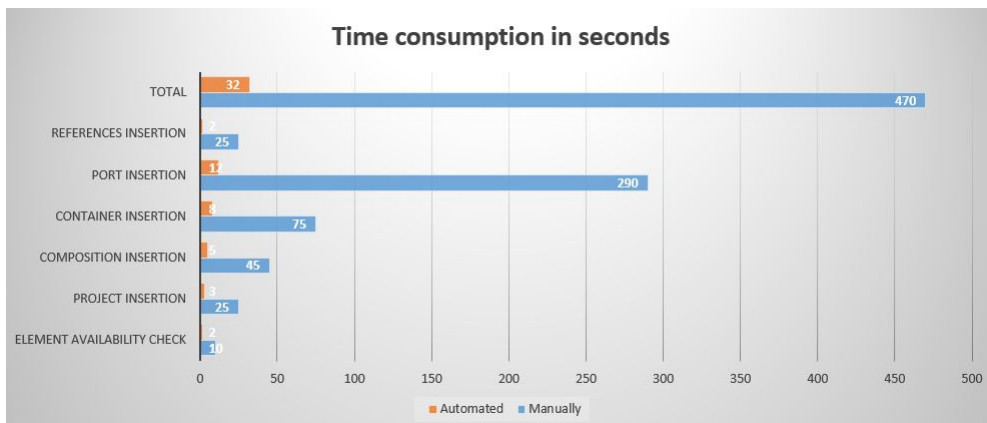


Figure 5.1: Time consumption in seconds as a chart

	Manually	Automated
<i>Availability of the elements checking</i>	~10 seconds	~2 seconds
<i>SW Level1 elements(Projects) insertion</i>	~25 seconds	~3 seconds
<i>SW Level2 elements(Compositions) insertion</i>	~45 seconds	~5 seconds
<i>SW Level3 elements(Containers) insertion</i>	~75 seconds	~8 seconds
<i>SW Ports elements(Data Objects) insertion</i>	~290 seconds	~12 seconds
<i>References insertion</i>	~25 seconds	~2 seconds
<i>Total consumed time</i>	~7.8 min.	~0.5 min.

Table 5.1: Consumed time comparison

- **Total Cost:**

This part focuses on the cost factor of both the approaches. The automated case needs additional computer energy, which will be consumed from the developed tool. However the consumed time of the automated approach is extremely low in comparison with manual operating.

I assume that the hourly work rate for an engineer is about 20 Euro. I assume again, that my tool needs about 200W per hour and that 1 kWh energy costs about 20 cent. That means my tool costs about $0.2\text{kWh} \cdot 0.2\text{Euro} = 4$ Euro cent per hour. The total cost of each case is as follow:

Manual: 7.8 min = 0.13h, $0.13\text{h} \cdot 20 \text{ Euro/h} = 2.6$ Euro

Automated: 0.5 min = 0.0083h, $0.0083\text{h} \cdot 20\text{Euro/h} = 0.16$ Euro.

The manual approach costs ~17 times more than the automated one, which is a huge difference. Whereby the tool has been developed in four months, which makes about: 4 months = ~100 days of work = 800 hours of work

$800 \cdot 20\text{Euro/h} = 16000$ Euro.

As mentioned previously, it only makes sense to develop a tool, if the same operation will be used or/and repeated several times.

- **Testability:**

The testability of an automated tool is in any case better than the manual solution. We know that a computer is always deterministic and the same input always delivers the same output by the same algorithm. Whereas a human will usual obtain a different output using a complex algorithm. That means, the computer does not have a human-fault factor and thus leads to a secure test environment. In addition the automated approach is effective, efficient and the test cases will be executed faster than the traditional method. We can conclude, that the automated process is definitely better than the manual one.

- **Expandability and Flexibility:**

An extension in the automated field is a little easier. If you extend the tool once, this will be saved forever. Whereby, the manual one needs this extension at each execution time. That means, the user does the same extension/work at each work procedure. The flexibility characteristic is also better in the automated version. The automated methods are usually easy to convert and to modify.

- **Reliability:**

The reliability characteristic is better in the automated version. It is mostly the same and doing the same procedures. However a manual operating procedure can be unintentionally manipulated by a human, which can be called the „human-fault factor“. The automated version is more reliable because of the same work procedure.

Chapter 6

Conclusion

The aim of this paper is to bridge the existing gap between model-driven system engineering tools and software engineering tools for embedded automotive systems. The following tools are used to achieve the main goals of this paper: Enterprise Architect, AVLab/ADD and Microsoft Office Excel. The model based approach has become popular and complex. This is due to embedded automotive systems evolving to in-car computers. These complex systems are in most cases developed, analyzed and tested by different large institutions and teams. Parallel to these difficulties, the system complexity and development time must be reduced and safety standards such as ISO 26262 [HHA⁺10] must be met. This challenge cannot be handled manually by humans. Therefore an automated, tested, verified and standardized application is essentially important. The tool in this paper has been developed to close the gap between model-driven system engineering and SW engineering tools. It manages the whole data transfer automatically via the tools „Application Programming Interface“.

The tool ensures a bidirectional data transfer between ADD, EA and Excel via providing data conversion between those model based systems and software engineering tools. This automated approach usually has positive and negative aspects. The comparison between automated- and manual development over the entire product life-cycle will be discussed in the Section 5.2.

The automated data transfer will be controlled in a tool which has been developed in object-oriented programming language C#. The tool of this work has been developed using a Graphical User Interface(GUI) application, which makes it user-friendly and more attractive. As mentioned, the data will be transferred automatically and enables different views and improves consistency, correctness and completeness of a product. Additionally this approach accelerates time-to-market and reduces the development costs.

After each transfer procedure, a log file will be generated. This file contains all the steps, as well as process warnings and errors, so that the user can track the steps and find solution if a problem occurs.

Chapter 7

Future Works

This section of this paper describes the planned features and extensions for the project. One of the main future works is the extension of the project, in a way that the tool supports the engineers in different work steps. Also usage in different companies and institutions is conceivable.

The functional safety standards for road vehicles, ISO26262 supports the automobile industry through different work processes, which will be increasingly refined and simultaneously automated. Since the tool keeps up with functional safety standards, it is conceivable that this project can be applied to other automobile industry fields. As mentioned at the beginning, the tool uses and is able to create customized components. This means that the tool can be adapted for the needs and processes of other companies.

The next proposal is to exchange information with with other SysML authoring tools, which is not implemented in the current version.

Another future work is the automated generation of the test cases and reports. This will help the user to design, implement, test and validate a system with a single click.

Appendix A

Tool Programmer-Guideline



Connecting Model-Based System Engineering and AVLab SW Test Environment

Programmer Guideline

Document Version: 1.0
Status: draft

Revision History

Rev. Index	Date	Author	Status	Changes
0.1	12/04/2015	Muesluem Atas, AVL	draft	Initial state described
1.0	06/10/2015	Muesluem Atas, AVL	draft	Document updated for a new release

Copyright © 2015 AVL LIST GMBH, all rights reserved.

internal

Table of Contents

1. Introduction	3
2. Prerequisites	3
3. Step-by-Step instructions of Microsoft Visual Studio	3
3.1 First Steps	3
3.1.1 Manage Start programs of Microsoft Visual Studio	3
3.1.2 Create the add-In library	4
3.1.3 Add the registry key	7
3.1.4 Arm Automotive Data Dictionary (ADD)	8

1. Introduction

This document is a Programmer-Guideline for the “Connecting Model-Based System Engineering and AVLab Environment“ tool. Use at your own risk.

The developed tool bridges the existing data exchange gap between model-driven system- and software engineering tools for embedded automotive systems. The information will be automatically exchanged between the following tools via an application programming interface (API): Enterprise Architect, ADD and Microsoft Office Excel. This approach enables model-based design and development, which is becoming increasingly popular in embedded automotive systems.

2. Prerequisites

The following software candidates have to be installed on your computer to enable successful use of this tool:

- Enterprise Architect
- Microsoft Visual Studio
- Microsoft Office Excel
- Local version of AVLab Database interface (ADD) + ADD access rights

3. Step-by-Step instructions

3.1 First Steps

3.1.1 Manage Start programs of Microsoft Visual Studio

To automatically start “Enterprise Architect (EA)” after running a “Microsoft Visual Studio” project, perform following points:

1. Open the C# project file (.csproj) as text file.
2. Add the following lines into the opened file.

```
<PropertyGroup Condition=" '$(Configuration)' == 'Release' ">
  <StartAction>Program</StartAction>
  <StartProgram>C:\Program Files (x86)\Sparx Systems\EA\EA.exe</StartProgram>
</PropertyGroup>
```

```
<PropertyGroup Condition=" '$(Configuration)' == 'Debug' ">
  <StartAction>Program</StartAction>
  <StartProgram>C:\Program Files (x86)\Sparx Systems\EA\EA.exe</StartProgram>
</PropertyGroup>
```

3. Save and close the file.

3.1.2 Create the add-In library

- To create such a project in Visual Studio, an add-In library is required. First start Microsoft visual Studio and choose “class Library” for the project type as in Figure 1.

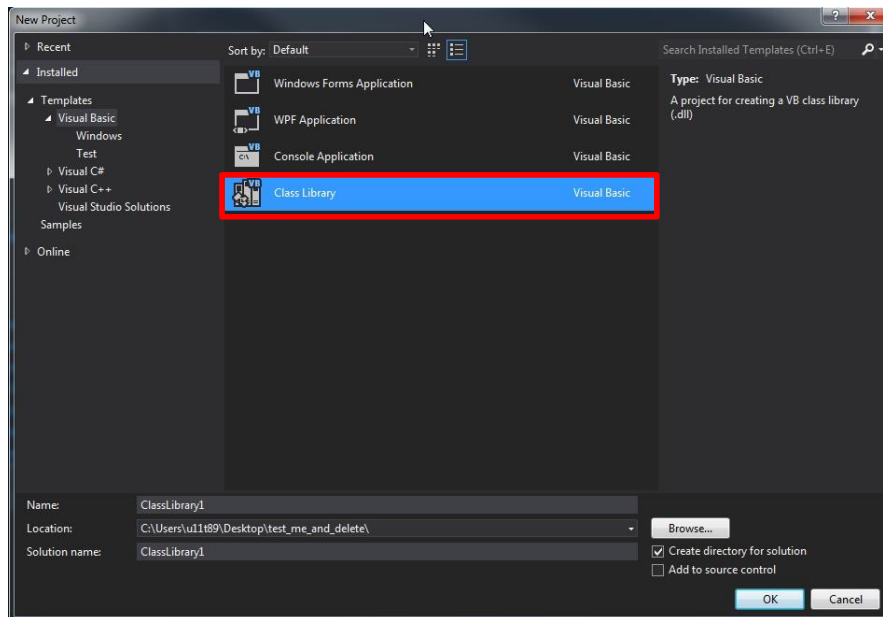


Figure 1: Select type of the new Visual Studio project

- For successful access to EA, add EA library into the visual studio project references. This reference allows an access to the EA classes/interfaces and its properties via API. This can be done under **right mouse click + Add Reference...** menu on the solution explorer of the C# project (see Figure 2).

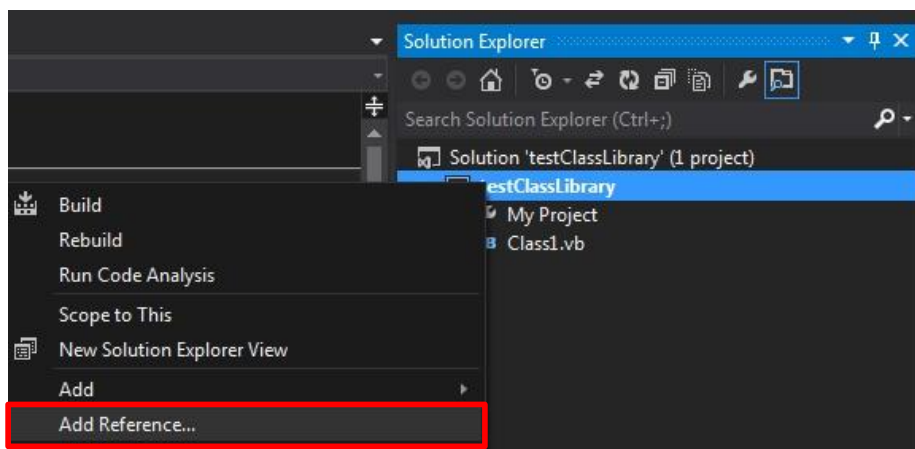


Figure 2: Add references to a project

- After this step, Figure 3 will appear which shows all the available references.

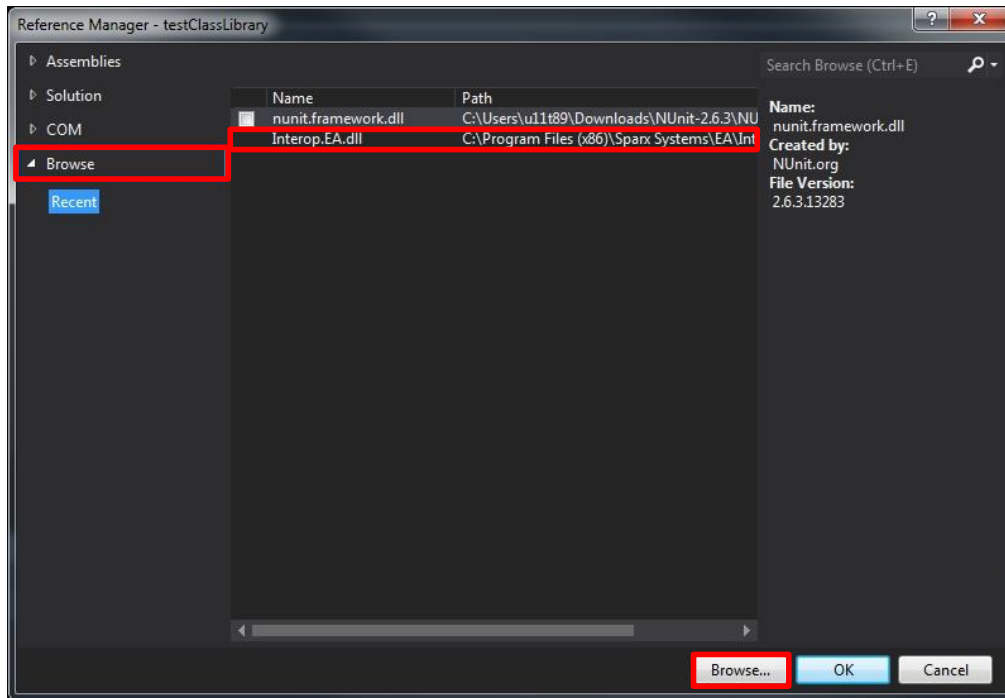


Figure 3: Reference manager window

- If EA.dll has never been used before, import the Interop.EA.dll into the references list. This step can be made as follow:
 - Click **Browse** tab on the *Reference Manager* window (see Figure 3).
 - Click **Browse...** button in Figure 3 and select the EA installation folder (*C:\Program Files\Sparx Systems\EA*).
 - Choose the file **Interop.EA.dll** and click **OK** button.

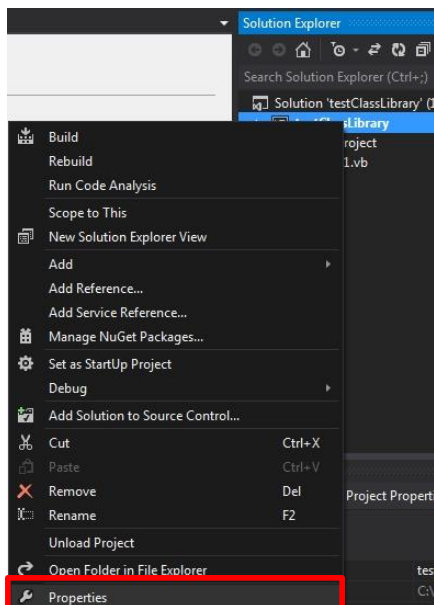


Figure 4: Open project properties

- The imported library has to be defined as COM object in Microsoft Visual studio. To perform this, apply the following settings:
 - Click **right mouse** on the project in the solution explorer + click **properties** menu (see Figure 4).
 - Open **Application** tab in the project properties window
 - Click **Assembly Information...** button (see Figure 5).
 - Set **Make assembly COM-Visible** checkbox in Assembly Information window (see Figure 6).

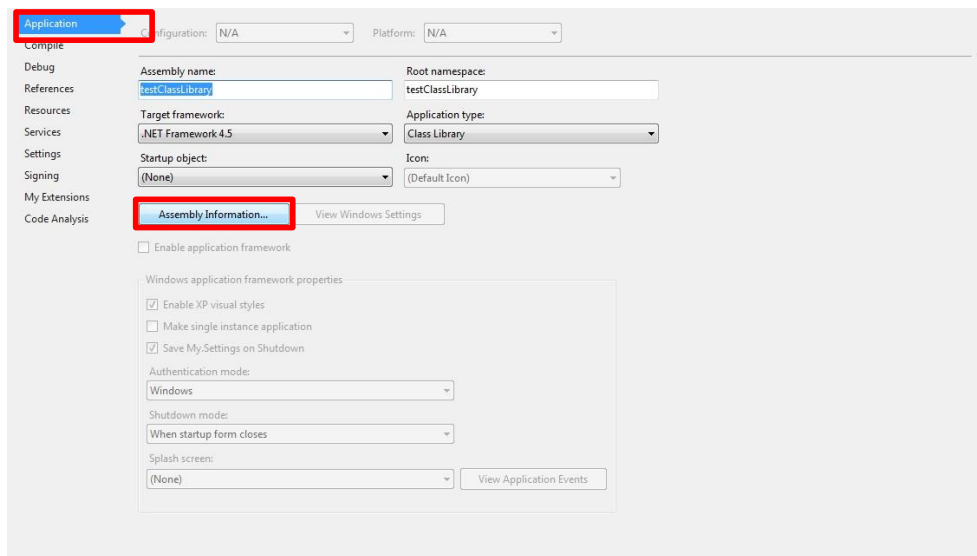


Figure 5: Application tab of the project properties window

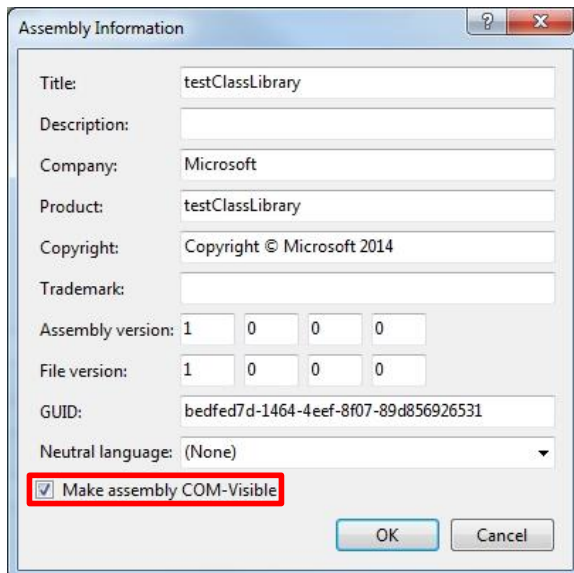


Figure 6: Assembly Information window

- Microsoft Visual Studio should register the imported library in the COM codebase. Otherwise the performed setting must be made at each execution/compile procedure. To prevent this, apply the following settings:
 - Change from Application- to **Compile** tab in project properties window (see Figure 7).
 - Set **Register for COM interop** checkbox (see Figure 7).

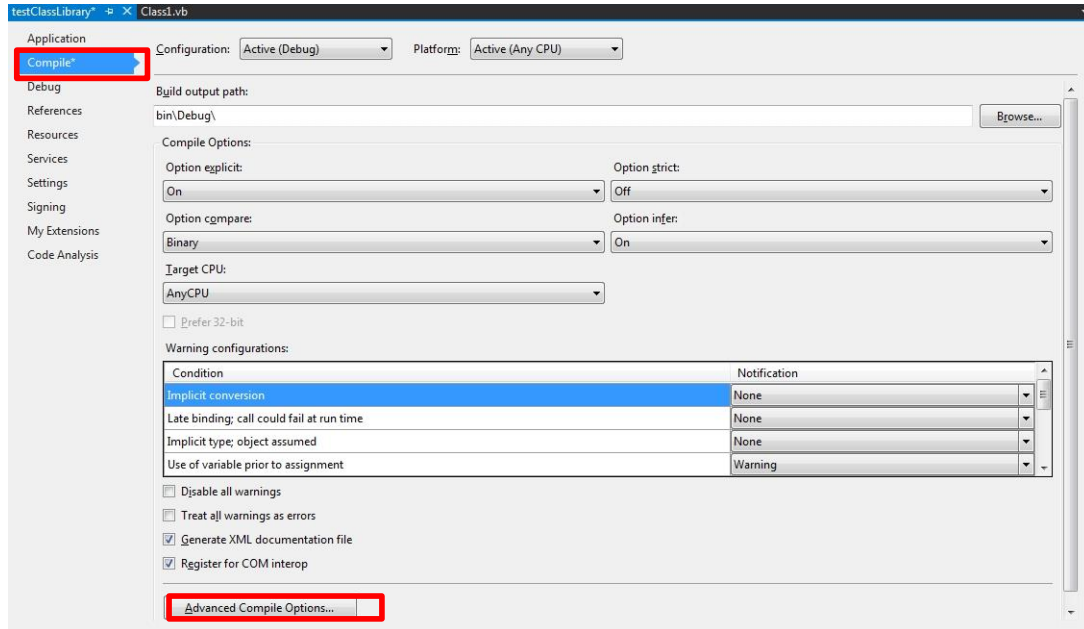


Figure 7: Compile tab in project properties window

3.1.3 Add the registry key

Another required setting in order to start the EA is the registration of the add-Ins in the registry editor. To do this: open the registry editor with a click on **Start - Run** menu and type **regedit** (see Figure 8).

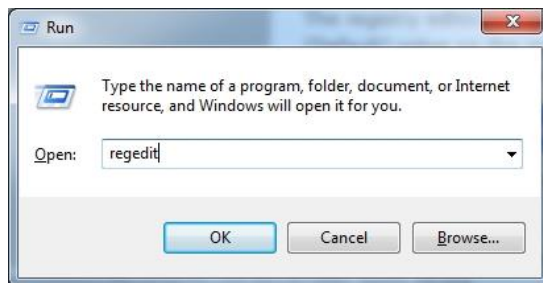


Figure 8: Open the registry key

Next, click the **OK** button, to open the required window. After opening this window, browse to the key **HKEY_CURRENT_USER\Software\Sparx Systems\EAAddins** in registry key and add a new key via a right click (see Figure 9).

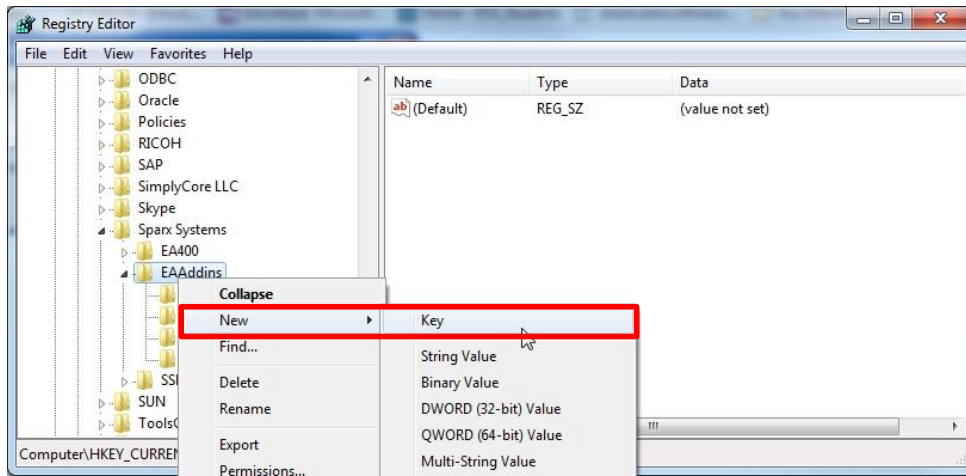


Figure 9: Add a new key in the registry editor

The registry editor will automatically create a default value for the new key. The data of the created key must be changed via a double click in the following form: **[ProjectName].[ClassName]**, whereby the **projectName** is your project name and **ClassName** is the visual studio class name, which will be used in your project. Figure 10 shows such an example.

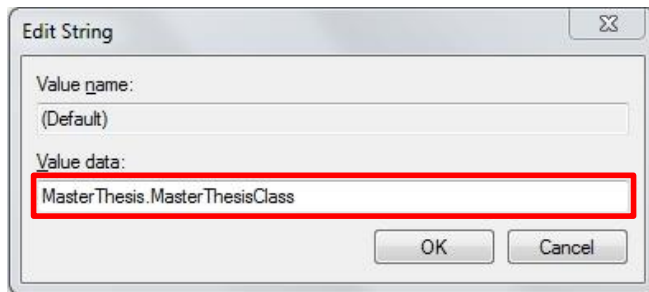


Figure 10: Edit the data of the new value

3.1.4 Arm Automotive Data Dictionary (ADD)

ADD is the interface to the AVLab database and manages all the elements of AVLab that are used. This tool supports different powertrain control processes of the automotive industry to provide an easy and efficient working method.

To arm the Automotive Data dictionary of the AVLab, the following steps are necessary:

- Provide access to the ADD Database.
 - The database behind this tool will be provided by oracle. To have access to this, the user must be registered in this database. This can only be made through the developer company (Visu-IT GmbH)
- Install the Oracle Client 11.2 software on your personal computer.
- For complete access to ADD, the user requires admin rights on the installed Oracle client software. This can be performed with the following points:

- Open the file “tnsnames.ora” in folder “C:\Oracle\ora11g\client\network\admin” with a text editor.
- Add the following text into the file “tnsnames.ora” (see Figure 11).

```

1 PDM955.WORLD =
2 (DESCRIPTION =
3 (ADDRESS_LIST =
4 (ADDRESS = (PROTOCOL = TCP) (HOST = atgrzs11121.avl01.avlcorp.lan) (PORT = 1521))
5 (ADDRESS = (PROTOCOL = TCP) (HOST = atgrzs11123.avl01.avlcorp.lan) (PORT = 1521))
6 )
7 (CONNECT_DATA =
8 (SERVICE_NAME = PDM955)
9 )
10 )
11

```

Figure 11: A part of the tnsnames.ora file

- Provide an access to the newest ADD offline version.
 - Get the newest offline version of the ADD from the developer company and copy it on to your personal computer.
- Open this ADD folder and double click on the file “Register.cmd” to prepare the ADD for the first usage.
- After these steps, the ADD tool can be opened with a double click on the file “ADD.exe” (see Figure 12).

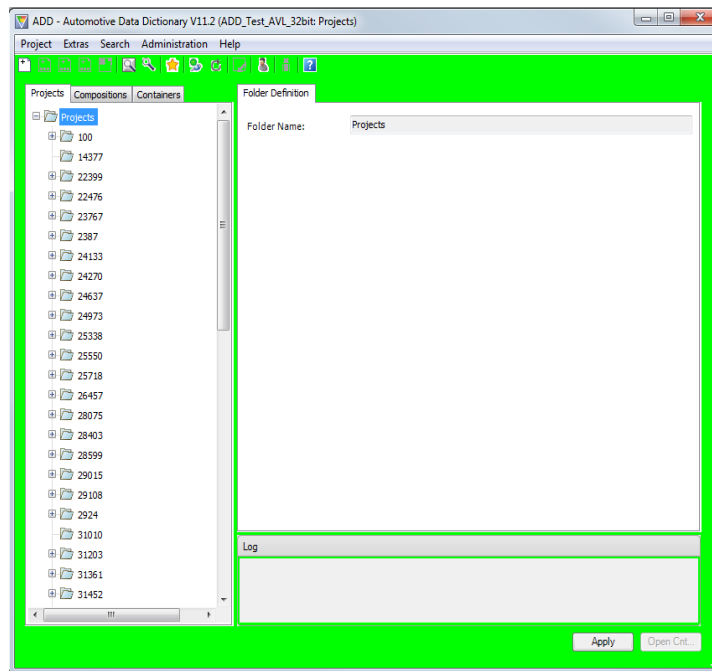


Figure 12: ADD User Interface

Appendix A

Abbreviation

A.1 Definitions

EA	Enterprise Architect
EAI	Enterprise Application Integration
ADD	Automotive Data Dictionary
API	Application Programming Interface
OMG	Object Management Group
UML	Unified Modeling Language
SysML	Systems Modeling Language
XML	Extensible Markup Language
XMI	XML Metadata Interchange
OWL	Web Ontology Language
OCL	Object Constraint Language
MDA	Model Driven Architecture
MBSE	Model-Based Systems Engineering
AMD	Automatic Model Development
ASCG	Automatic Source Code Generation
ECU	Electronic Control Unit
GUI	Graphical User Interface
ASIL	Automotive Safety Integrity Level
SDTF	SysML Document Traceability Framework
InVEST	Interactive Visualization Engine for SysML Tools
TPT	Time Partition Testing
SFR	Special Function Register
RTOS	Real-Time Operating System
PSM	Platform Specific Model
PIM	Platform independent Model
APAS	Automotive Personal Assistance System
GPS	Global Positioning System
MOF	Meta Object Facility
MiL	Model-in-the-Loop
SiL	Software-in-the-Loop
PiL	Processor-in-the-Loop
HiL	Hardware-in-the-Loop

Bibliography

- [15706] A model driven architecture for enterprise application integration, Jan 2006.
- [Ben83] Herbert D. Benington. Production of large computer programs. *Annals of the History of Computing*, 5(4):350–361, Oct 1983.
- [BK08] E. Bringmann and A. Kramer. Model-Based Testing of Automotive Systems. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 485–493, April 2008.
- [Cla09] J.O. Clark. System of systems engineering and family of systems engineering from a standards, v-model, and dual-v model perspective, March 2009.
- [DF07] C. Davey and J. Friedman. Software systems engineering with model-based design. In *Software Engineering for Automotive Systems, 2007. ICSE Workshops SEAS '07. Fourth International Workshop on*, pages 7–7, May 2007.
- [EGa] *Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units*. 5.5. Audi and BMW and Daimler and Porsche and VW.
- [FG07] T. Farkas and D. Grund. Rule checking within the model-based development of safety-critical systems and embedded automotive software. In *Autonomous Decentralized Systems, 2007. ISADS '07. Eighth International Symposium on*, pages 287–294, March 2007.
- [Fis98] G.H. Fisher. Model-based systems engineering of automotive systems. In *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE*, volume 1, pages B15/1–B15/7 vol.1, Oct 1998.
- [FNH09] T. Farkas, C. Neumann, and A. Hinnerichs. An integrative approach for embedded software design with uml and simulink. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 516–521, July 2009.
- [HHA⁺10] M. Hillenbrand, M. Heinz, N. Adler, J. Matheis, and K.D. Muller-Glaser. Failure mode and effect analysis based on electric and electronic architectures of vehicles to support the safety lifecycle iso/dis 26262. In *Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on*, pages 1–7, June 2010.

- [KLPK13] M. Krunic, I. Letvencuk, I. Povazan, and V. Krunic. An approach to model driven development and automatic source code generation of GUI controls. In *Intelligent Systems and Informatics (SISY), 2013 IEEE 11th International Symposium on*, pages 63–68, Sept 2013.
- [MKR06] Ji Chan Maeng, Jong-Hyuk Kim, and Minsoo Ryu. An rtos api translator for model-driven embedded software development. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 363–367, 2006.
- [MLD⁺12] Tobias Carsten Müller, Malte Lochau, Stefan Detering, Falko Saust, Henning Garbers, Lukas Märtin, Thomas Form, and Ursula Goltz. A comprehensive Description of a Model-based, continuous Development Process for AUTOSAR Systems with integrated Quality Assurance. In *A comprehensive Description of a Model-based, continuous Development Process for AUTOSAR Systems with integrated Quality Assurance*. Institut für Regelungstechnik, TU Braunschweig, Institut für Programmierung und Reaktive Systeme, TU Braunschweig, Institut für Verkehrssicherheit und Automatisierungstechnik, TU Braunschweig, December 2009-12.
- [Rao05] M. Rao. Sysml with artisan studio. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 15–, Sept 2005.
- [Sei14] Matthias Seidl. Automated Transformation of System Models into Ontologies. In *Automated Transformation of System Models into Ontologies*. Know Center - Graz University of Technology, September 2014.
- [TF14] K. Trase and E. Fink. A model-driven visualization tool for use with model-based systems engineering projects. In *Aerospace Conference, 2014 IEEE*, pages 1–10, March 2014.