UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Liem Radita Tapaning Hesti

Model Driven Development and Analysis for Embedded Automotive Software

Master's Thesis (30 ECTS)

Supervisor: Dr. Christian Saad
Supervisor: Dr. Kalmer Apinis

Tartu 2019

Model Driven Development and Analysis for Embedded Automotive Software

Abstract:

Model-driven development and analysis is the state of the art method in the automotive industry. One of the reasons for its heavy utilization is coming from the black box nature of the components developed by the automotive vehicle manufacturers. The other reasons are coming from the pressure to produce quality software that complies with all regulatory standards but can fit the pricing model of automotive vehicle manufacturers.

Validity and standard compliance of the components can be verified using models before the actual piece of software is deployed into an automotive vehicle. The utilization of the model also creates challenges: how to produce final software that precisely reflects how the model works. An automatically generated software from a model is deemed as an answer since it is coming from the already verified model and also will inherently retain consistency with the model. As software gets more and more critical inside an automotive vehicle, a model to create the software is getting more and more complicated and along with the automated software generation process.

This thesis examines the model-driven development and analysis process for automotive software by conducting model conversion from MATLAB/Simulink model into AUTOSAR. The application developed for this thesis provides analysis and insights for every step of the conversion process. From the insights gathered along the process, it shows that the different model and transformation method creates a different model representation that affects the final structure of the AUTOSAR result. In the end, there are several possible alternatives on the way a model can be seen and transformed into an AUTOSAR file. It is also concluded that the iterative process in this project is not final and can be further improved.

Keywords:

Model Driven Development, Model Driven Analysis, Embedded System, AUTOSAR, Automotive Software

CERCS: P170 Computer science, numerical analysis, systems, control

Autotööstuse tarkvara mudelipõhine arendamine ja analüüs

Lühikokkuvõte:

Mudelipõhine arendamine ja analüüs on autotööstuses kasutatav uus meetod. Seda rakendatakse mootorsõidukite tootjate poolt, kuna hajusale komponentide arendusele sobib olemuslikult spetsifitseerimine musta-kasti printsiibil. Muud põhjused tulenevad survest toota kvaliteetset tarkvara, mis vastab kõigile regulatiivsetele standarditele, kuid mis sobib autotööstuse tootjate hinnamudeliga. Mudeli kasutamisel saab komponentide kehtivuse ja standardse vastavuse kontrollida enne, kui tegelik tarkvara on autosse paigaldatud.

Mudeli kasutamine tekitab ka väljakutseid, et toota lõpuks tarkvara, mis kajastab täpselt mudeli toimimist. Mudelist automaatselt genereeritud tarkvara loetakse vastuseks, kuna see on stabiilne ja pärit juba kontrollitud mudelist. Kuna tarkvara muutub autotööstuses üha olulisemaks, muutuvad tarkvara loomise mudel ja genereerimise protsess üha keerulisemaks.

Käesolev töö uurib mudelipõhist autotööstuse tarkvara arendamise ja analüüsimise protsessi — teisendades MATLAB/Simulink mudel AUTOSAR mudeliks. Lõputöö raames loodud programmid teostavad analüüsi erinevate teisendussammude tarbeks. Protsessi analüüsides selgus, et teisenduse meetoodika mõjutab oluliselt mudeli esitust ning ka lõpptulemuseks saadud AUTOSAR mudeli struktuuri. Näeme erinevaid võimalikke alternatiive sellele, kuidas mudelit saab vaadata ja muuta AUTOSAR-failiks. Selles lõputöös vaadeldud iteratiivne prosess pole lõplik ja seda saab veel täiustada.

Võtmesõnad:

mudelipõhine arendamine, mudelipõhine analüüs, manussüsteem, AUTOSAR, autotööstuse tarkvara

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction **5**
- 1.1 Automotive Software Engineering 5
- 1.2 Model Driven Development 6
- 1.3 Motivation & Thesis Goals 6

2 Model Mapping **8**
- 2.1 Automotive Software Models 8
- 2.2 Control Theory Model 10
- 2.3 Information Processing Model 11
- 2.4 AUTOSAR 12

3 Mapping Methodology **18**
- 3.1 Simulink to AUTOSAR Mapping 18
- 3.2 Component Visualization 20
- 3.3 Graph Coloring 21

4 Sanity Testing & Data Dependency Analysis **23**
- 4.1 Sanity Testing 23
- 4.2 Data Dependency Analysis 23

5 Software Development **25**
- 5.1 Model Assessment Application 25
- 5.2 AUTOSAR Generator 30

6 Result & Validation **32**

7 Conclusions & Future Improvements **35**
- 7.1 Conclusions 35
- 7.2 Future Improvements 36

References **40**

Appendix **41**
- I. Model Converter Application Walkthrough 41
- II. List of Reference Tables 50
- III. Licence 62

1 Introduction

1.1 Automotive Software Engineering

Automotive Software Engineering is one of the niche fields in software engineering, and it has increasing importance within automotive vehicle development. Nowadays, automotive vehicles manufacturers are using software to give competitive advantage and differentiation on their products. Around 80% of the value creation came from the computer system [LH02]. The incorporation of software into an automotive vehicle itself was quite a new thing. It was started from the usage of an Electronic Control Unit (from now on will be addressed as ECU), which is an embedded electronic device with sensor and actuator. At the beginning of automotive software integration inside a vehicle, an ECU runs an isolated and dedicated task/program such as braking, steering, and controlling the engine. ECU itself uses a highly optimized machine code that focuses on consuming resource as small as possible because the automotive industry uses price unit model where each component must fit the pricing constraint of the vehicle manufacturing process. As the automotive industry grows, more and more ECUs integrated into an automotive vehicle to do various functionalities. There're also needs of data sharing between ECUs that creates the bus system to enable effective communication between ECUs.

The modern car nowadays can have more than 100 ECUs [EJ09] with more functions require ECUs to communicate with each other. One of the examples of the feature where it integrates several independent functions is the Central Locking System. A Central Locking System integrates the locking and unlocking car doors functionality with comfort functions (such as adjusting seats), with safety (such as locking the car beyond a minimum speed), and with human-machine-interface functions (such as signaling the locking and unlocking using the car's interior and exterior lighting system) [Bro03]. Since the ECU uses a highly optimized code to do a specific task, integrating a new feature that needs to have subsystems previously never interact with each other to work together is not a trivial task to do. On top of there's no straightforward way to integrate the system, the car integrated system also faces problems that commonly can be found in the distributed system [FSN+03b, FSN+03a].

An automotive software engineering has its quirks. Most of the time the exact way software works inside the vehicle parts are unknown because the components can come from different manufacturers and suppliers. Manufacturers and suppliers are usually not sharing the code inside the components since it is part of their trade secret. Each car manufacturer (referred to as "OEMs") have to make their tests against this black box system to figure out the full functionalities and limitations of the components. OEMs also need to adjust their software to accommodate coordination with these vast component differences. With this software engineering environment, solving the distributed system problems in an automotive vehicle becomes even more challenging. The recent partner-

ship project between various OEMs and suppliers named AUTOSAR (AUTomotive Open System ARchitecture) tried to provide better standardization by creating a model-based middleware layer. AUTOSAR is a worldwide partnership that was started by the automotive industry leaders such as BMW, Bosch, Continental, DaimlerChrysler, Siemens, and Volkswagen to create an open standard for electronic/engineering architecture for automotive vehicles [AUTb]. This open standard allows automotive software engineers to work together to create better architecture that can solve the multiplex communication problem.

1.2 Model Driven Development

Since the automotive software engineering process is strongly affected by the black box nature of the components inside an automotive vehicle, model driven development becomes the most popular method in automotive software development [BBR+05, TKWE03]. The components' capabilities and limitations are mapped into models that can represent how the component works and interact with the other component. In order to create a model that can represent the real thing, car assemblers have to conduct tests to find out of the functionalities and the limitations of the components and how it interact with the other components in static or dynamic way. A detailed report on how each component behaves is critical to ensure the final product's successful integration.

There are several common modelling approaches in the automotive software engineering. The most common procedure is using models of control theory created with tools like MATLAB/Simulink [WM95]. Another commonly used model is information processing model that is represented using UML/State Machine [RBvdBS02]. Although information processing model is much more common in the business IT field, it starts to gain popularity in the automotive software engineering due to the growing complexity of the data exchange inside automotive vehicle's components. Both approaches are used together in automotive software engineering to create a comprehensive view of how the system works since only using one model will never be enough to cover the entire aspects of the integrated system inside an automotive vehicle [SS04].

1.3 Motivation & Thesis Goals

This thesis is trying to explore two most commonly used model in the automotive software engineering: control theory model and information processing model and to do auto-generation from existing control theory and information processing model into AUTOSAR compliant file. Creating an auto-generation software itself is an endless pursuit in automotive software engineering with several companies and research groups have dedicated their time and effort to create the most reliable auto-generation software. In the ideal world, good auto-generation software can expedite the automotive software

development process, reducing time and cost needed to create the model, and also enhancing safety by reducing errors from manual creation process.

The auto-generated AUTOSAR compliant file underwent a sanity testing and data dependency analysis to test its validity. Sanity check provides a broad overview of the system and checks the potential static error as the result of model conversion and data dependency analysis is needed to asses the complex communication problem of the distributed system. During model conversion, there is a lot of simplification and design decision on the way to convert the model. The decision relies heavily on data visualization to show components dependencies. Graph coloring technique is also employed in the visualization to understand better the relationship between components and checking data dependencies. This technique is useful to solve distributed system problem such as resource management and scheduling to create an assessment to convert the model. This thesis provides the thought process from the initial MATLAB/Simulink model assessment, AUTOSAR model transformation, and analysis of the auto-generated AUTOSAR file. This thesis also provides suggestions for future improvement for this whole process.

This thesis is organized into seven chapters. The first chapter contains an introduction to the automotive software engineering world. It provides context on why model-driven development is the most common practice on the automotive software industry and became the main idea of this thesis. The second chapter explores the different types of models commonly used in automotive software engineering with the emphasize on the MATLAB/Simulink control theory model, UML based information processing model, and AUTOSAR. The third chapter describes the mapping strategies to transform the MATLAB/Simulink model to AUTOSAR along with visualization combined with graph coloring technique to identify complex interconnected components. Chapter four lists methods to check result validity by conducting a sanity testing and data dependency analysis and chapter five details the software developed to analyze the models and converting MATLAB/Simulink model into AUTOSAR. The result of the analysis and software developed is described in chapter six. Chapter seven contains the conclusion from the whole model conversion process and future improvements. There are also appendixes that give the walkthrough details of the application developed and lists of the components inside the model used for this thesis

2 Model Mapping

This section gives an overview of automotive software models and provides some idea model creation process to represent an automotive system. Software components of an automotive vehicle mapped into a model with different angles of abstraction and different point of views. The two most popular approaches in mechanical engineering / electrical engineering and software engineering: control theory model and information processing model are also explored in this section since it is the main idea of creating auto-generation software from MATLAB/Simulink file into AUTOSAR compliant file.

2.1 Automotive Software Models

In automotive software engineering, one single model won't be able to represent the whole view of the integrated system inside an automotive vehicle. Automotive software modeling process starts from the user level (the highest level of the abstraction) until the software can be deployed to the hardware and work together as an integrated system inside an automotive vehicle. The level breakdown of an automotive software engineering architecture can be seen in figure one with an explanation as follow [Bro03]:

1. **Functionality Level - Users View**

   This functionality level-user view aims to capture all the software-based functionality of an automotive vehicle to the users. Users in this context are not only limited to drivers and passengers but also garage and maintenance staff, people involved in car manufacturing and so on. This level provides the understanding of how the software services are offered, its dependencies, and on how it interacts with other services. Function hierarchy is usually utilized to model this level. Techniques such as Message Sequence Charts can be used to capture services and their interactions [RFH+05, IT99, AHKPM05].

2. **Design Level - Logical Architecture**

   The design level captures the logical component architecture. The functional hierarchy from the user level is decomposed into a distributed system of interacting components. Whether the components are implemented by hardware or software and the number of different components performs the function are not described in this design level. By reducing the system into an interaction between services, common services needed across vehicle infrastructures can be identified. It also promotes conceptual reuse for services that differ only in the way they are deployed from one vehicle line to another [NP04].

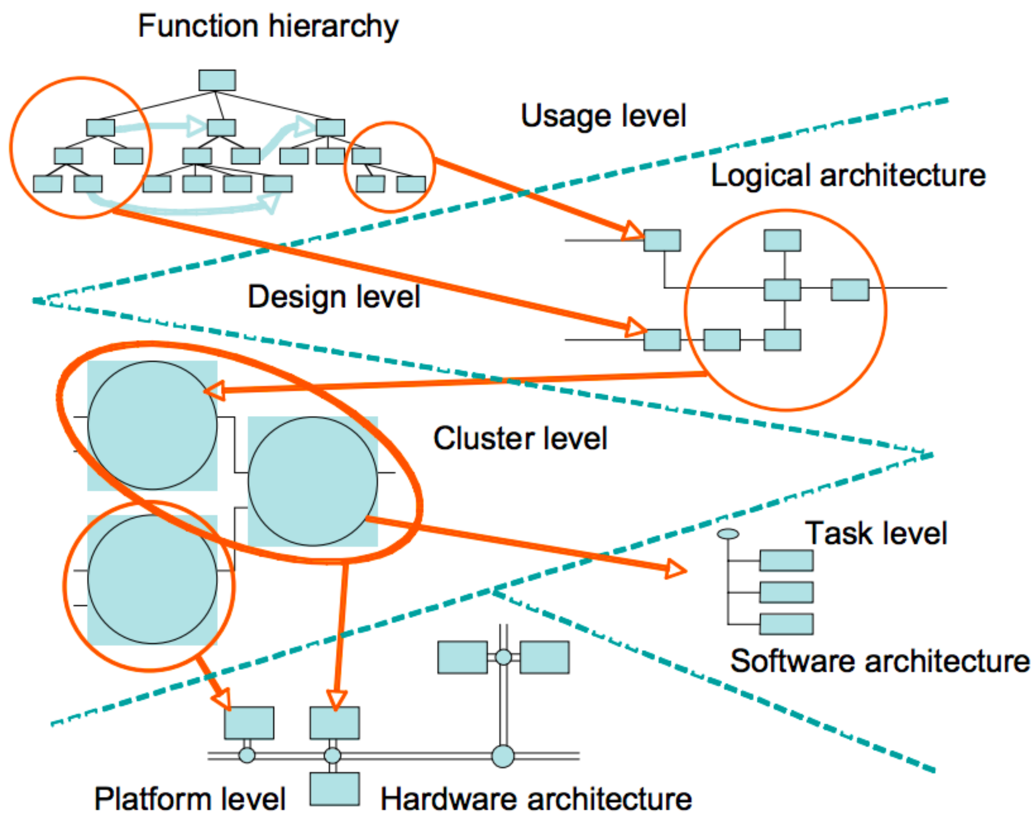


Figure 1. Comprehensive automotive software engineering architecture [Bro06].

3. **Cluster Level**

   This level is one level before software architecture. The logical architecture is rearranged in a way so it can be ready for the deployment. Software components are decomposed further until a sufficiently fine-grained granularity is reached and then reorganized into clusters. The clusters itself form the units of deployment.

4. **Task Level - Software Architecture**

   The software architecture consists of the classical division of software in platforms like operating systems and bus drivers on one side and the application software represented by tasks, which are scheduled by the operating system, on the other side. This software has to be deployed onto the hardware. The high-level software architecture is derived quite straightforwardly from the logical architecture. It is a representation of the logical architecture by programs.

5. **Platform Level - Hardware Architecture**

   The hardware architecture consists of all the devices including sensors, actuators, bus systems, communication lines, ECUs, MMI, and many more. The macroscopic view of the hardware architecture consists of the network of buses, the ECUs, the gateways that route and adapt the signals between several domains and busses, and the sensors and actors that are connected within the network. The tiny part is the hardware architecture inside an ECU (for example processor, memory, I/O).

When the software is finally deployed to the actual hardware, hardware/software components interaction should be the implementation of the logical architecture. In the modern automotive vehicle, the design of this hardware/software interaction grows more and more similar to the chip design process due to the distributed nature of computing in a car [GFL+02]. Currently, there are two popular models used to represent this distributed nature: control theory model and information processing model.

2.2 Control Theory Model

Control theory is a discipline used in mathematics and engineering field. This field deals with the dynamical response of a system to input. The goal is to create a control model for controlling system that is optimum without delay or overshoot and ensuring safety by managing the system response to inputs and disturbance. Control theory explores problems such as stability (the way system response on inputs and disturbance), robustness (tolerance in the variation of the parameters), tracking (small error for specific input) [MT]. In the automotive vehicle, systems are connected to sensors and actuators where the software/hardware have to deal with real-time input and output of physical data. In this sense, the classical control theory model is suitable to represent how the system works.

Control theory is usually visualized using a block diagram. One of the tools that commonly utilized to create this block diagram is MATLAB/Simulink [Thec]. MATLAB/Simulink is a block diagram environment for multi-domain simulation and model-based design. In automotive component design, MATLAB/Simulink simulates the complete control system, including the control algorithm in addition to the physical plant. MATLAB/Simulink is especially useful for generating the approximate solutions of mathematical models that might be difficult to solve manually. Figure two shows MATLAB/Simulink's user interface describing vehicle's (a train locomotive with one additional car) movement and the braking system where the sequence of controls are described using blocks to represent the physical forces that control the train's movement with the locomotive provides signals to control of the whole system.

There have been numerous efforts in building frameworks that can link control theory model with implementation languages for both hardware and software [FGI+05, Thea].

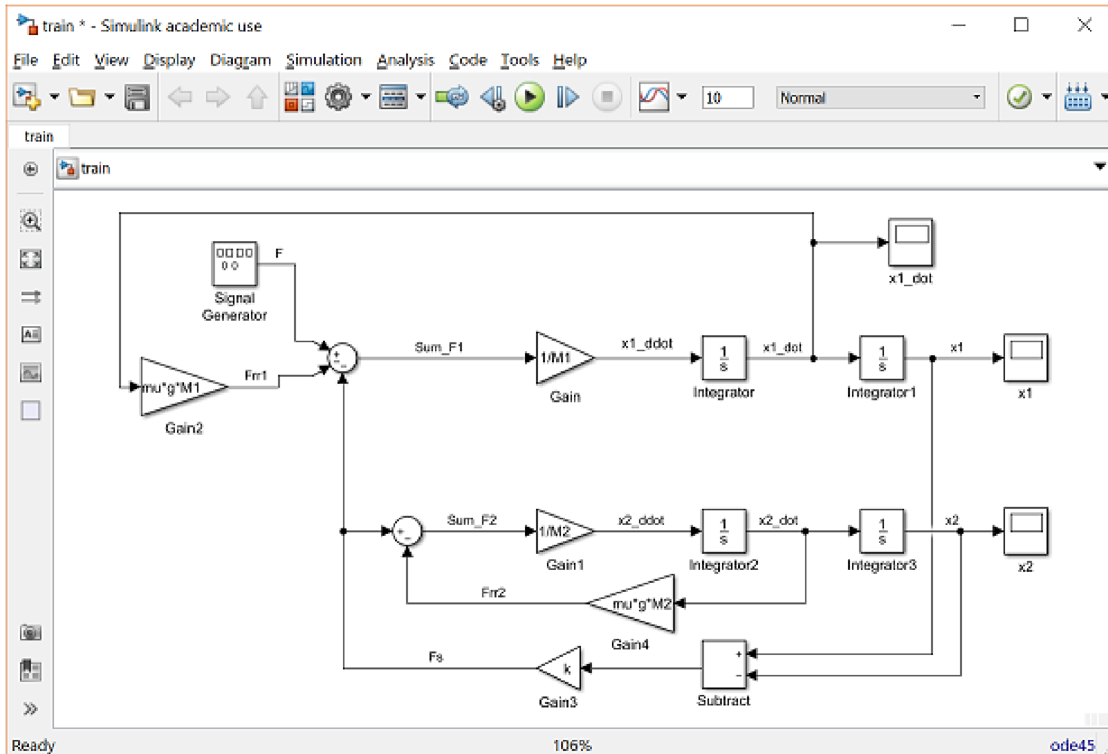


Figure 2. Simulink control model user interface example. Vehicle's movement and braking system [MT].

This thesis also starts from the Simulink model as its base to generate the AUTOSAR model that's ready for the actual deployment of an automotive vehicle.

2.3 Information Processing Model

Another common approach in modeling software engineering is to model services interaction in the form of data flow commonly found in information processing discipline. This approach is getting more popular due to the data transfer process is getting more complex inside the modern-day automotive vehicle. In designing the model using information processing model, there are differences between the practice in business IT and automotive software engineering. In automotive software engineering, the goal is not to change the system or component structure at runtime, but to have more flexibility for the distribution of software components over different ECUs, while designing the board net (the set of implemented functions and their mapping to hardware, including communication dependencies) of a car [Bro06].

There is already an existing effort to convert MATLAB/Simulink model directly to

UML-like model, called Massif [Via]. Massif is an Eclipse plugin created by Viatra that can convert the MATLAB/Simulink model to an XML file for Eclipse Modeling Framework (EMF). EMF itself is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model. All information for each MATLAB/Simulink block's type is stored in the generated meta-model that makes it possible to convert the model back to the Simulink file.

In this thesis, Massif is utilized to convert MATLAB/Simulink model into Eclipse-EMF model before converting it into an AUTOSAR model. The reasoning behind this process is to make the development of AUTOSAR model converter application easier since the model runs in Java-Eclipse environment where the other AUTOSAR helper tools are already available there. Converting the MATLAB/Simulink model into a UML-like diagram of the Eclipse-EMF model also provide another view about how the system works that can help the mapping process from a Simulink model into an AUTOSAR model.

2.4 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture that is jointly developed by OEMs, suppliers and tool developers who are the industry leaders in the automotive industry. In the automotive software modeling view, AUTOSAR positioned itself as a middleware layer where it provides an abstraction between hardware and custom software for automotive vehicle system with technical goals of modularity, scalability, transferability, and reusability.

Modularity enables tailoring software elements to the individual requirements of electronic control units and their tasks. AUTOSAR model should be scalable so that common software modules can be easily adapted to different vehicle platforms to avoid software redundancy with similar functionalities. Transferability ensures that the optimization of the use of resources available throughout a vehicle's electronic architecture. In reusability, the product quality and reliability can be easily reinforced across product lines [Sch05].

To achieve these goals, AUTOSAR provides a common software infrastructure for automotive systems of all vehicle domains based on standardized interfaces for the different layers. The architecture of this AUTOSAR infrastructure is designed in layers to enable separation of concerns that is useful to maintain and debug the software. Three main layers in AUTOSAR architecture are the Application layer, Basic Software layer, and AUTOSAR Runtime Environment (RTE). AUTOSAR itself is an operating system that is running inside the microcontrollers that power the automotive vehicle system. This layered architecture can be seen in figure 3.

This thesis model conversion takes place in the application layer of AUTOSAR that consists of software components, the smallest part of a software application that has specific functionalities. Application functionality resides in this application layer, and it's

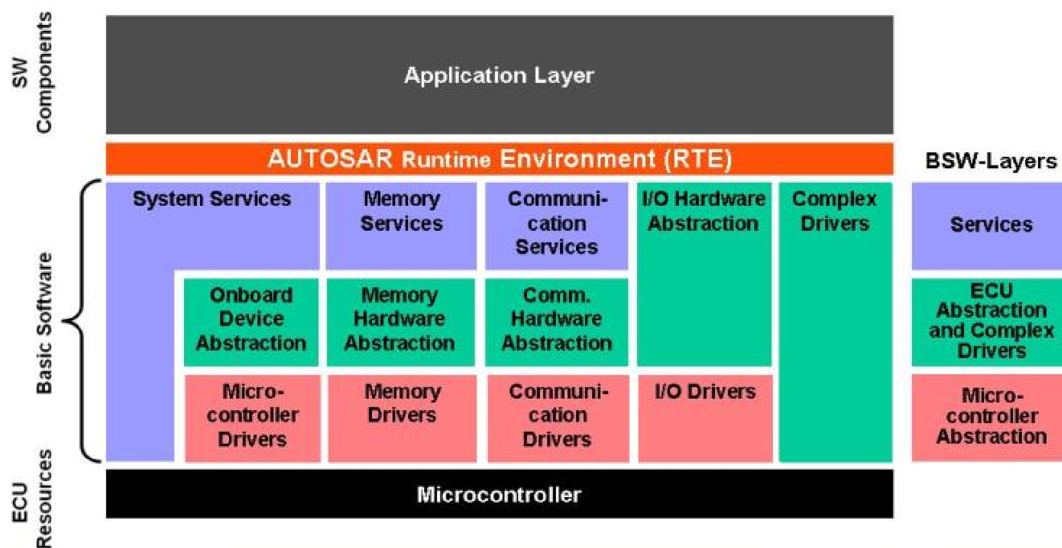


Figure 3. AUTOSAR main layers: Application layer, Basic Software layer, and AUTOSAR Runtime Environment (RTE) [Ges11].

a custom application that can utilize more low-level functionality in the Basic Software (BSW) layer using APIs provided in AUTOSAR Runtime Environment (RTE). One of the examples of how AUTOSAR works can be found in figure 4. This figure describes a wiper washer system of an automotive vehicle using AUTOSAR model. Each washer in the front and the back of the car is controlled by a software component (*Washer* atomic software component). *WiperWasherMgr* atomic software component does the triggers to activate the front and rear washer where it gives the order to activate components in *Washer* such as rear washer for example.

Below is the list of commonly found AUTOSAR components in automotive vehicle modeling:

- **AUTOSAR Package (ARPackage)**

  An AUTOSAR Package (ARPackage) is a bundle to group Software Components, data types, and other elements of AUTOSAR. It creates a namespace where it within one system with unique naming. with standardized naming convention [AUTc]. Its sub-packages have categories such as standard, blueprint, and sample. Wrapping components into package ensure separation of concern that is important for code reusability and modularity. The standardized package structure can be seen in figure 5.

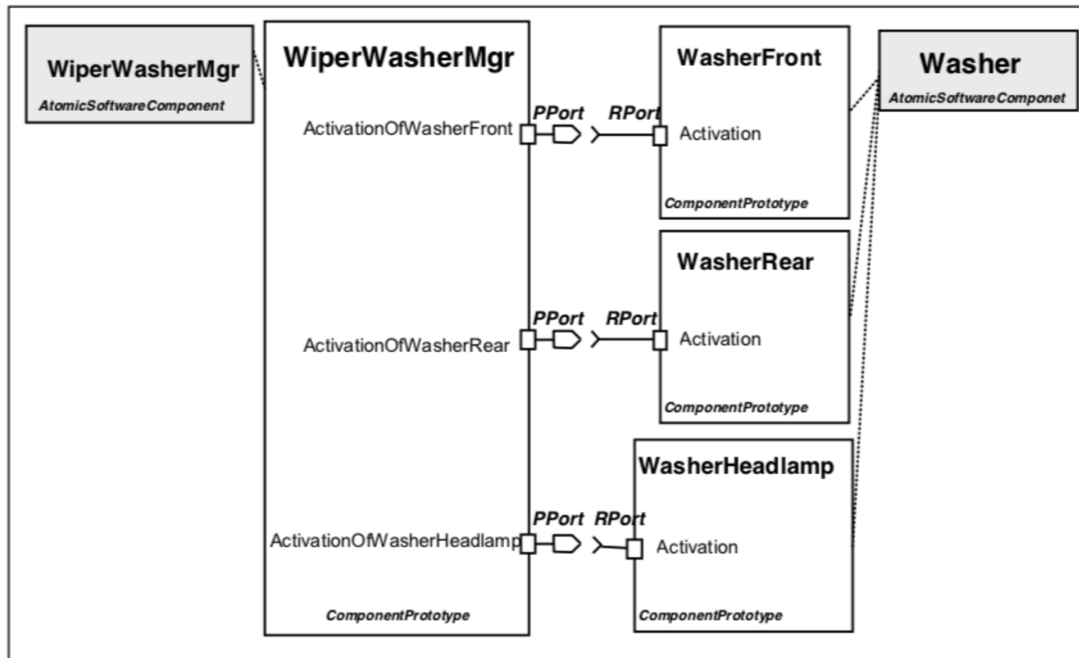


Figure 4. Multiple component prototype interaction between two software components in the wiper washer system [AUT18].

- **AUTOSAR Software Component (SWC)**

  An AUTOSAR Software component (formally *SwComponentTypes*) is a self-contained unit where application software within AUTOSAR are organized. It is an encapsulation of the implementation of software functionality and behavior also providing well-defined connection points using ports (formally *PortPrototypes*) for external access. The software component can also be found in RTE layer and BSW layer.

  An AUTOSAR software component (referred to as AUTOSAR SWC) exists in several flavors. The most prominent of these flavors are atomic software component (formally *AtomicSwComponentType*) and composition software component (formally *CompositionSwComponentType*). Composition software component has the purpose to aggregate *SwComponentTypes* to create units of higher complexity allowing system and subsystem abstraction. It groups existing software-components and hides the complexity when viewing or designing logical software architecture without any possible functionality.

  An atomic software component, on the other hand, is the smallest possible granularity of software component (atomic, as the name already indicates). It contains

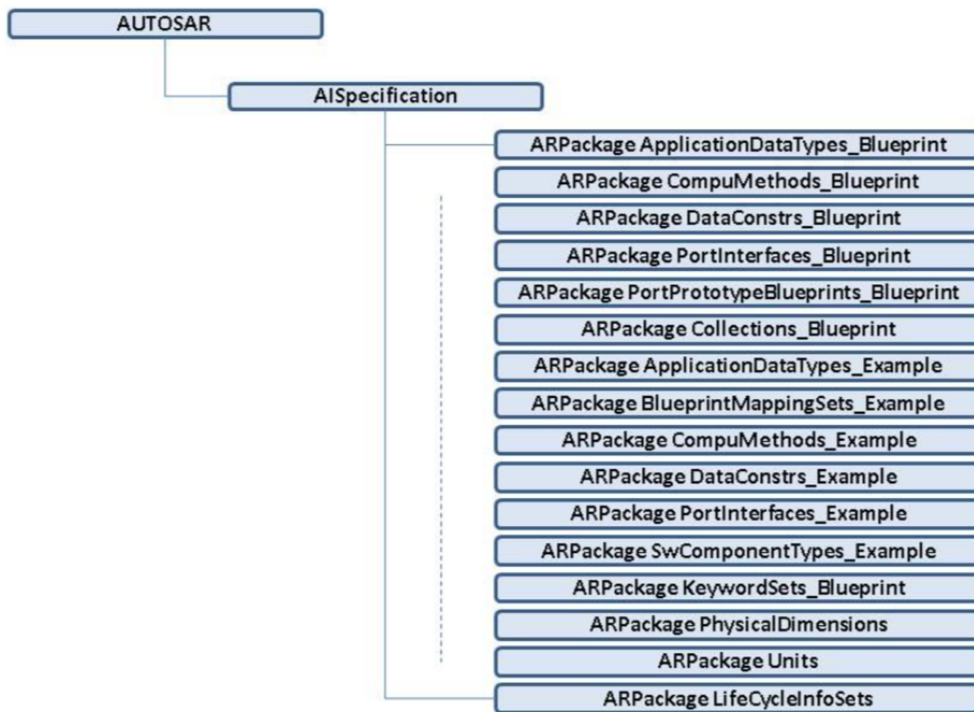


Figure 5. ARPackage structure [AUTc]

Internal Behavior (formally *InternalBehavior*) that carries a detailed description of the inner structure of an atomic software component in terms of internal data and execution units (formally called *RunnableEntities*) as well as the links between the internal structure to the ports on the surface of the atomic software component. The actual implementation of automotive application software happens within the atomic software components. Figure 6 illustrates the software component's and its Runnables content.

- **Ports**

  Ports (formally *PortPrototypes*) are parts of an AUTOSAR SWC to interact with other AUTOSAR SWCs, and it's where the data flows. A port belongs to exactly one AUTOSAR SWC and characterized by port interfaces (formally *PortInterfaces*). Port interfaces describe the communication paradigm of the ports. Several communication paradigms that are supported in AUTOSAR:

  – services or data are required (this specialization utilizes *RPortPrototype*)
  – services or data are provided (this specialization utilizes *PPortPrototype*)

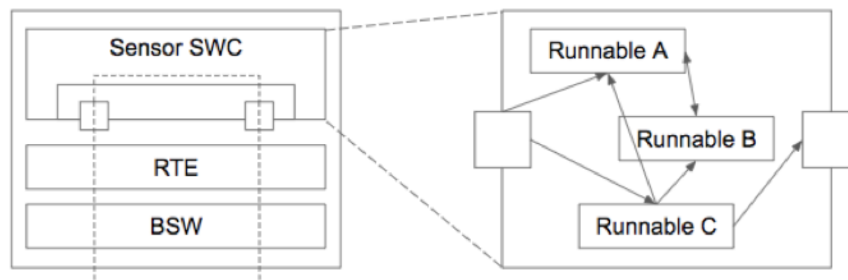


Figure 6. AUTOSAR Software component structure and its runnables

  – services or data are provided and requested at the same time (this specialization utilizes *PRPortPrototype*)

- **Internal Behavior**

  The internal structure of an atomic software component is described by Internal Behavior (formally *InternalBehavior*). It is characterized mainly by runnable entities, RTE events. A runnable entity is a sequence of instructions that can be started by the run time environment in the context of a task. It also can be executed concurrently by mapping them into different tasks.

  An RTE event can be described as all possible situations that can trigger the execution of a runnable entity by the run time environment. RTE events can address timing, data sending and receiving, invoking operations, call server returning, mode switching, or external events. It also can either activate a runnable entity or wakeup a runnable entity at its wait points.

- **Virtual Function Bus**

  Virtual function bus is the abstraction of the AUTOSAR SWCs interconnections of the entire vehicle, and Virtual Function Bus defines the communication pattern inside AUTOSAR. The central structural element in AUTOSAR is the component where it has a well-defined port for interaction between components. Virtual function bus assembles and integrates software components to a virtual AUTOSAR system to verify the consistency of the communication relationship between software components. This approach allows car manufacturers to break down the complexity of their systems in a very early design phase of the product development cycle.

  AUTOSAR interface defines the services or data that are provided on or required by a port of a component. The most commonly used AUTOSAR Interfaces are

Client-Server Interfaces and Sender-Receiver Interfaces, which allows the usage of data-oriented communication mechanisms over the Virtual function bus. Other kinds of interfaces are allowing the communication of modes, non-volatile or fixed data, and the triggering of processes.

3 Mapping Methodology

This thesis uses a single Simulink file that describes the control flow of a braking system inside an automotive vehicle provided by Augsburg University named RISA. This MATLAB/Simulink file is a complex model where it consists of in total 1820 components excluding port connections and also 85 subsystems and sub-subsystems. There are several steps involved during the process of converting this MATLAB/Simulink model into an AUTOSAR file. The conversion from MATLAB/Simulink file itself utilizes an open source existing tools called Massif to convert MATLAB/Simulink model into much more readable EMF (Eclipse Modelling Framework) XML file before the model is converted into AUTOSAR model. Figure 7 describes the file conversion flow

There was a realization during the conversion process and conversion experiments that each component in this Simulink model can't be translated one by one and have to be simplified and analyzed. The converted model is validated over and over using the standard sanity testing, and data dependency analysis is conducted to examine component interaction within the model. This thesis employs different visualization in the EMF model to get a comprehensive view of the component interactions.

3.1 Simulink to AUTOSAR Mapping

There are several challenges during the model conversion process from MATLAB/Simulink file into AUTOSAR compliant file. One of them because MATLAB/Simulink is proprietary software. A proprietary software behaves like a black box that sometimes its behavior is hard to predict, and the users have no other way to tweak or figure out the solution by themselves. Reading the MATLAB/Simulink file directly from its source turns out to be a complicated ordeal. Several methods have been used to read and convert the

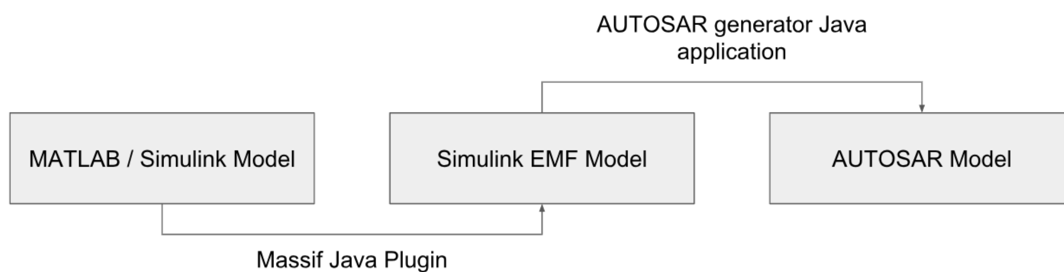


Figure 7. Model conversion flow in this thesis

MATLAB/Simulink with one of the solutions is reading creating an application to read the file without MATLAB/Simulink application to understand the structure and transform it as needed without the limitation from the original MATLAB/Simulink application.

This thesis utilizes an already existing tool from Viatra named Massif to simplify the reading process of the MATLAB/Simulink file. Massif converts MATLAB/Simulink control theory model into information processing model. This step creates a much easier to read file since the model is converted into EMF (Eclipse Modelling Framework) XML file that is an open source product that can be easily tweaked or adjusted. The final XML file is also more familiar file to read with many library options to read the file. The conversion from Simulink to EMF also provides a broader understanding of how the system works in the information processing model perspective.

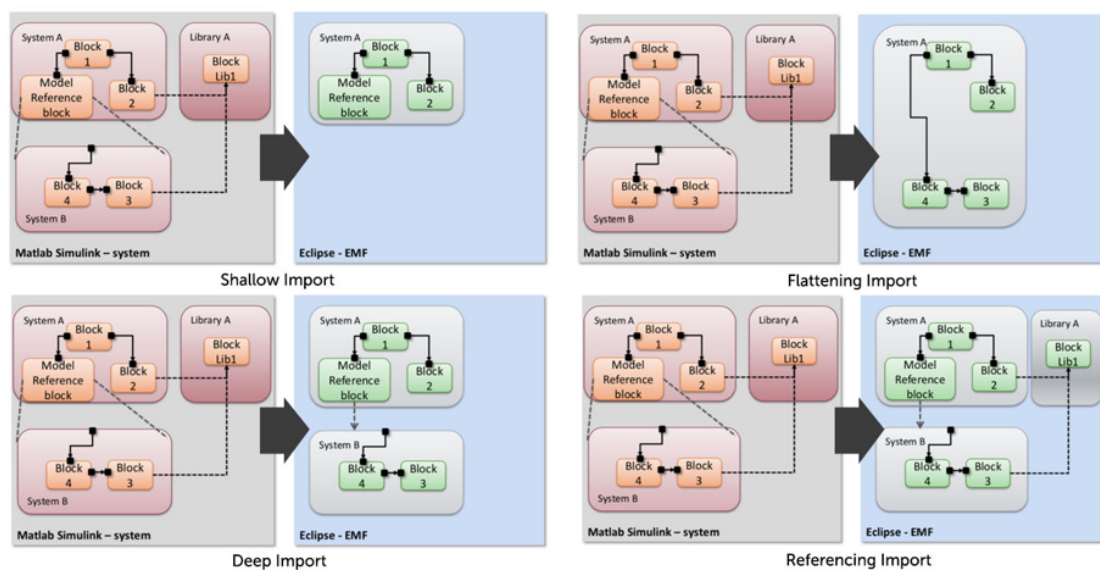


Figure 8. Different import modes in Massif plugin [HRS]

Massif reorganizes each Simulink components into an XML block under tag contains. Information about value and variables of a Simulink block is stored in property under the properties tag. The *SimulinkRef* tag is used to store information about the Simulink block name and the *sourceBlockRef* tag is used to store information about the block type and category. Ports and connections define data flow structure between blocks. Massif also provides various conversion options. It has several modes that can be used to convert (called import in the application) the model. The options are are: Shallow import (only blocks within non-referred systems are imported hierarchically), Deep import (each block inside each subsystem is imported. Each referenced model is imported as an individual model with direct model referenced in the parent model), Flattening import (each model reference block is imported as though it was a subsystem), Referencing

import (For blocks with active library links, each source library is imported once as an individual model but may be referenced multiple times). This thesis also explores the use of different import methods to find the suitable mode for model conversion. The illustration of the different import modes can be seen in figure 8

Another hard challenge is coming from the AUTOSAR documentation itself. AUTOSAR version 3.x has mapping guidelines to map Simulink components into AUTOSAR Components [AUTa]. However, since AUTOSAR version 3.x is no longer utilized and this thesis will convert the model into AUTOSAR version 4.x these mapping rules need to be updated and improved based on AUTOSAR version 3.x mapping rules.

3.2 Component Visualization

Component visualization plays a vital role in the various aspect of the component such as ensuring converted model validity, provides a broad overview of the unknown system, debugging process and giving insights to convert the model better. There are two applications utilized to do visualization, model assessment application to assess the MATLAB/Simulink model, EMF model, and AUTOSAR model and **AutoAnalyze** [KSS$^+$17] application that is used only for AUTOSAR.

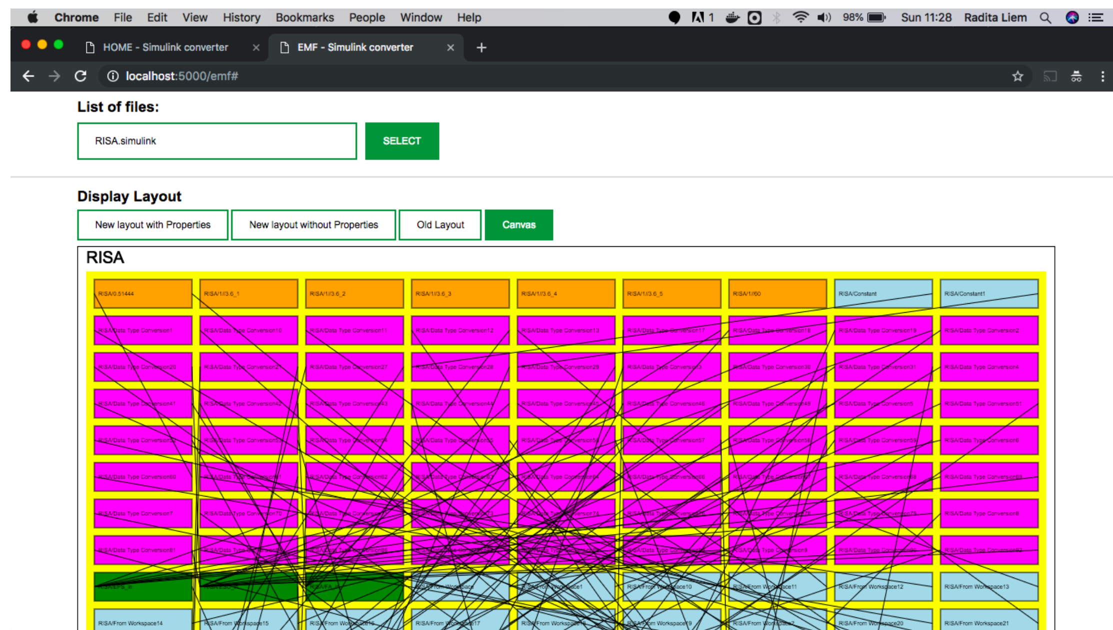


Figure 9. Blocks visualization of the generated EMF model

The model assessment application has undergone multiple iterations to provide the most concise view possible of the system. At first, the MATLAB/Simulink model is

displayed in a block model. The visualization result is hard to read and can't provide much information more than some rough perception of the system's complexity as can be seen in figure 9. After several iterations to improve the readability of the model, the EMF model transformation into a tree force diagram creates a better visualization of the system (figure 10).

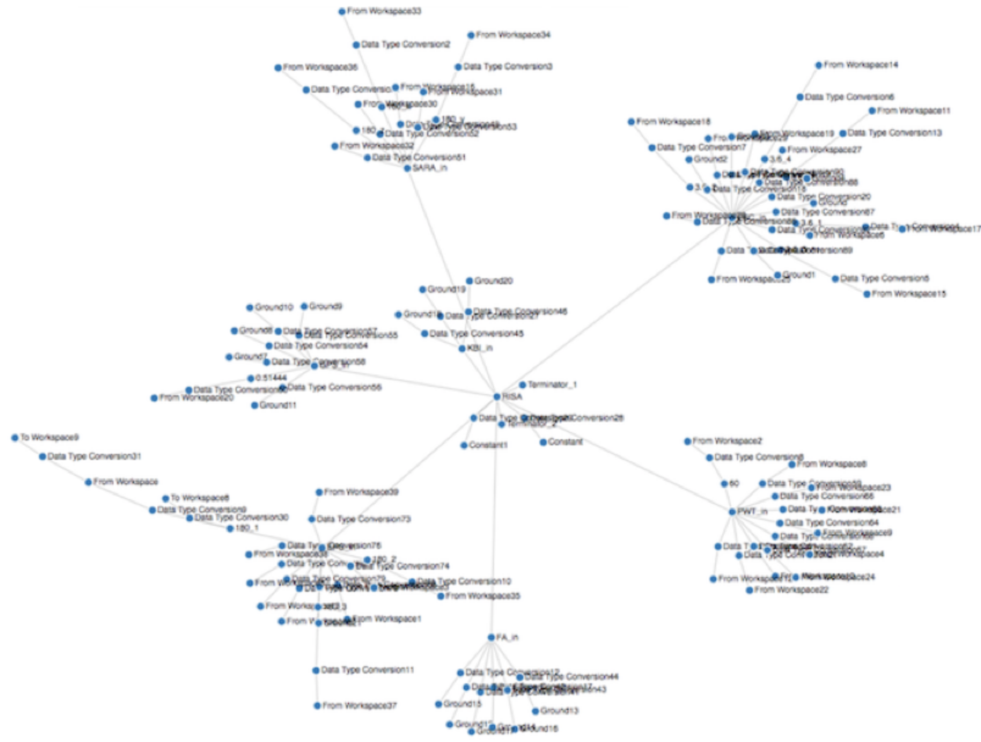


Figure 10. Force diagram visualization of the EMF model

3.3 Graph Coloring

Graph Coloring is a way to label vertices of a graph using colors such that no two adjacent vertices share a color. Graph coloring has many applications including task scheduling, parallel computation, network design, and many more. In the component visualization graph coloring to quantify the complexity of interaction between components in the EMF file. Graph coloring is integrated inside component visualization and implemented quite later in model analysis application after finding complex subsystems that are difficult to translate.

Backtracking algorithm is selected to color the nodes since this algorithm ensures that all components are colored correctly and there's no urgency to perform a fast algorithm.

Each subsystem in the model is a small graph consist of less than 100 components (vertices that need to be colored). This algorithm works by assigning colors one by one to different vertices, starting from the vertex 0. Before assigning a color, the algorithm checks the already assigned colors to the adjacent vertices and iterates recursively within the graph until all adjacent nodes don't have the same color [Lew15].

4 Sanity Testing & Data Dependency Analysis

Sanity testing and data dependency analysis are used in this project due to the unknown final result. Sanity testing process is chosen as it is simple and can provide an overview of the system and dependency analysis is selected because the model contains complex interconnected components that can result in data race condition. The analysis is done iteratively and continuously in every step of the model conversion process.

4.1 Sanity Testing

The purpose of a sanity testing is to ensure that the system does not fail and the returned results can pass the most straightforward correction test. Sanity testing runs on the premise that if the system can't pass the simple test, it will not be able to withstand more complicated test [ZKP+08]. Sanity testing saves time and effort before creating a more sophisticated test. In this thesis, the sanity testing can also give a broad overview of the system when checking the information discrepancy during the model conversion process.

Sanity testing process for this thesis is written in the form of ad-hoc scripts. One of it is a Python script to check whether there are missing MATLAB/Simulink component names on the generated EMF file. During the EMF model checking process, the tree graph also highlights possible defect components in the form of parts that have no relation with other components. If it occurs that some components have no connection with the other components, the original model gets double checked to ensure its validity.

The verification of the final generated AUTOSAR file relies heavily on the **AutoAnalyze** application developed by the University of Augsburg. AUTOSAR file with obvious component defects can't be to run on **AutoAnalyze** application or won't be able to produce a component graph.

4.2 Data Dependency Analysis

The modern automotive vehicle software has complex bus level communication architecture that creates a high possibility of the data race condition between its component. The data race condition is a situation that is commonly found in the distributed system and the compiler theory. It can be generally described as the condition where two programs (in the compiler theory, it is threads) are accessing a single data source with at least one of the two accesses is a write [HJM04]. The problem occurs when at least one of the program's action is dependent on the accessed variable value to make a decision. This condition can be catastrophic if it happens in the safety function of an automotive vehicle.

There are several approaches to detect data race possibility with the majority of the approaches are based on the *lockset* principle and or *happens-before* principle [Bec]. The *lockset* principle lies on the assumption that race condition occurs because of shared

variables are not appropriately connected by an appropriate lock. By using this principle, the interconnected component is flagged as having the possibility to create race detection, and this approach might not be suitable for a big and complex system with numerous interlocking parts since it creates a combinatorial number of possibilities where the data race might happen.

Happens-before principle can be commonly found in the data flow or state flow simulation. It detects data races between current access and maintained previous accesses by comparing their happens-before relation using logical time stamp like vector clocks [Ha13]. It might not check every possibility where the data race can occur so the analysis can be conducted within a reasonable amount of time. The majority of tools to monitor data race condition combines these two principles. This thesis tried to combine these two principles where the produced graph is trying to flag all interlocking components that have a high level of data dependencies using graph coloring visualization. In checking the validity of the generated AUTOSAR compliant file, this project uses AutoAnalyze application which provides a comprehensive analysis to describe data dependencies between components inside the AUTOSAR file.

5 Software Development

There are two applications created for this thesis: **Model assessment application** and **AUTOSAR file generator application**. Both applications work together with **AutoAnalyze** application to improve the auto-generated AUTOSAR file as in figure 11 below. The model assessment application works as the initial step for the research where all materials to generate an AUTOSAR file is read, summarized, and transformed into various graphs. The goal of this application is to provide insights and guidance for translating the MATLAB/Simulink model into an AUTOSAR model.

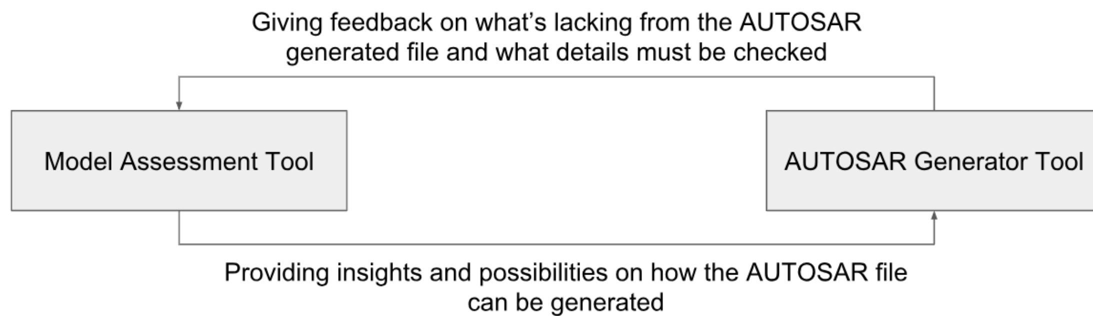


Figure 11. Interaction between softwares developed for this thesis

Since the AUTOSAR documentation only consists of the standards for the components' structure and there are no strict rules for mapping MATLAB/Simulink components into AUTOSAR components, the conversion procedure is usually tuned to the automotive vehicle manufacturers needs and this thesis is exploring several possibilities of the model conversion rules and methods. This thesis is doing continuous iteration between AUTOSAR generator tool and model assessment tool is needed to improve and optimize the generated AUTOSAR file. The features in the model assessment application are added gradually to provide a better granularity of the system. The different degree of dimensions and granularity is important when it comes to translating MATLAB/Simulink component into its AUTOSAR counterpart.

5.1 Model Assessment Application

The model assessment application is a web-based application to open and assess different models explored in this thesis: MATLAB/Simulink model, Massif generated EMF model and AUTOSAR model. The purpose of this application is to give insights for generating AUTOSAR model from the target MATLAB/Simulink model. This application provides

various model visualization that can give an overview of the model from a different angle or perspective. The application has three main features:

- **MATLAB/Simulink Model Assessment**

  This feature opens the MATLAB/Simulink model directly without using its native application. It gives general file information about the number of components inside the model, name, and type of the components, and the structure of the elements in block representation. This feature also displays the sanity test result from checking the component name difference between Massif generated EMF file and the actual MATLAB/Simulink file. The user interface of this feature can be seen in figure 12.

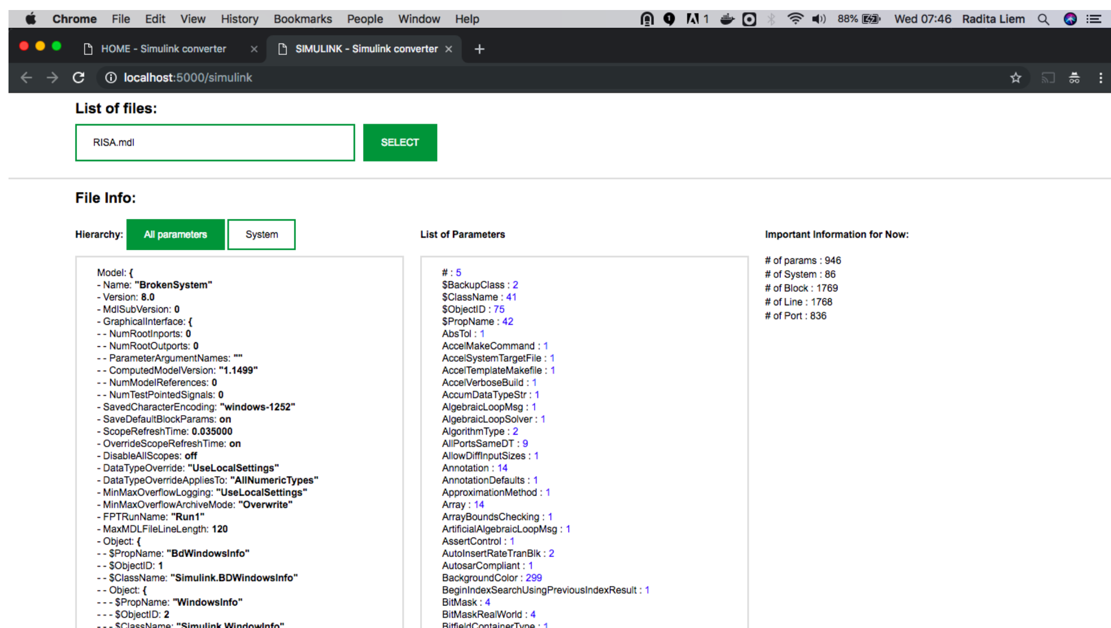


Figure 12. User interface for MATLAB/Simulink component assessment. The left side shows the file structure, the middle part shows all unique components inside the model, and in the right side is the ad-hoc script result to display important information

  An important insight that came out from this feature is that Massif application discards other components outside basic Simulink block diagrams. Stateflow components [Thed], HDL coder components [Theb], and MATLAB formulas are missing in the generated EMF file. This feature tried to reconstruct Simulink blocks components, but it has inferior quality compared to the visualization from the EMF Model Assessment feature. The assessment shows that the generated EMF file retains all needed information to create a complete graph of the system.

Because of that, the development to improve block component visualization from MATLAB/Simulink file is not pursued further.

- **Simulink EMF Model Assessment**

  This feature is assessing the model by creating a visualization from the generated EMF model. It shows various kind of visualizations that is the result of the iterative process of EMF model assessment. The visualization from the first iteration tried to recreate block diagrams similar to MATLAB/Simulink model and managed to provide a rough overview of the system complexity where it has multi-level subsystems and a high degree of interconnected components. Based on this initial visualization, more improvement is created to assess this system complexity. The depth of the system can be seen in figure 13.

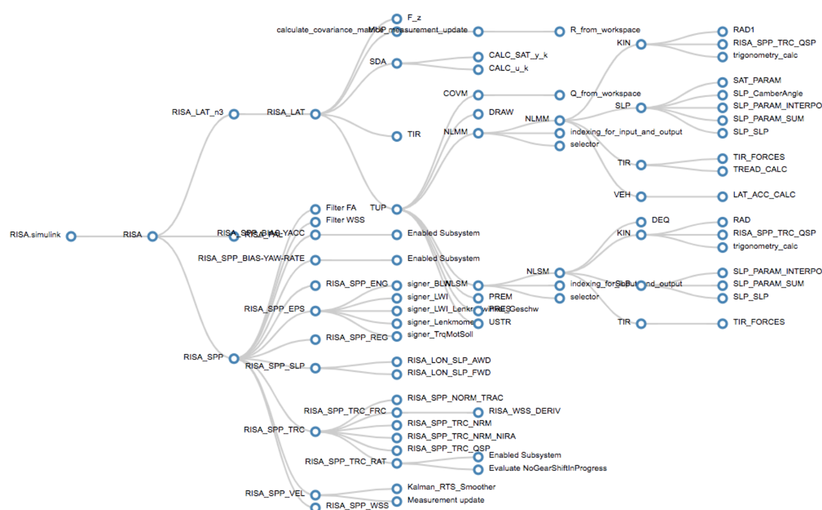


Figure 13. The hierarchy structure of the target RISA MATLAB/Simulink file shows the depth of the system

In the later iteration, the difficulty in comprehending the structure of the components inside the system and subsystems was tackled by changing block visualization method into interconnected node graph created with the D3 visualization library. The system depth is shown using a hierarchy diagram and component interactions in each subsystem is represented using a force diagram. This visualization shows a fresh perspective on the system's structure and gives some ideas on how the MATLAB/Simulink model can be structured into AUTOSAR software components.

In the component interaction force diagram (figure 14), each node has a different color code based on the MATLAB/Simulink block reference type and category.

This coloring is given to make it easy to find out which components can be merged or discarded. Discarding component is necessary since not all Simulink components must or can be translated into AUTOSAR components [AUTa] and adding color to components can help the process of discarding unused components. Components such as the arithmetic function, logical function, and data conversion function are merged into a software component or a runnable entity.

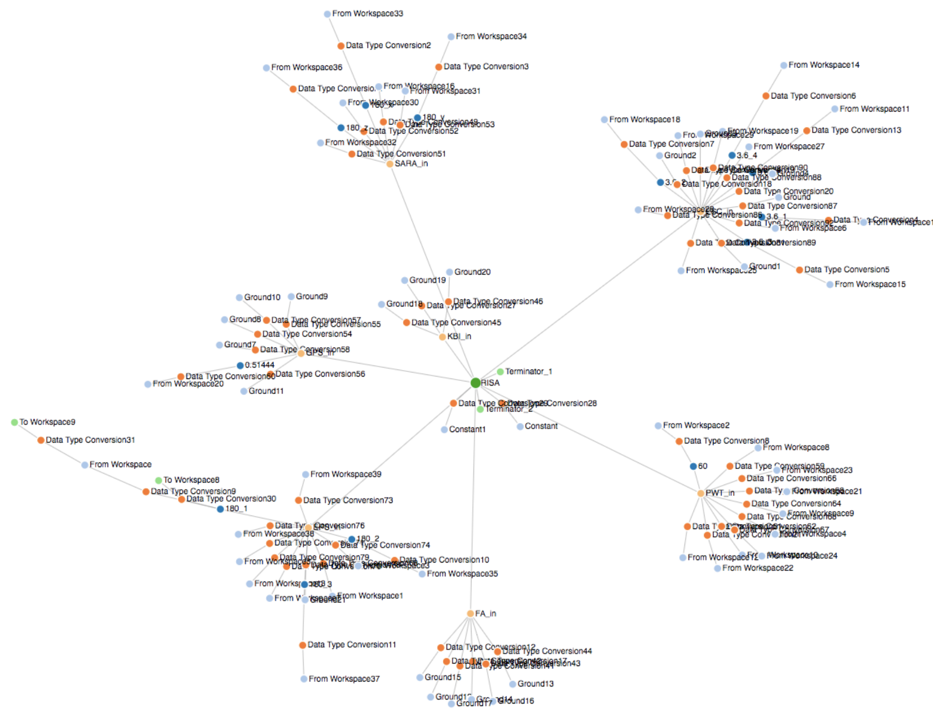


Figure 14. The interconnected components force diagram visualization on the first level of the target RISA MATLAB/Simulink file. Node is colored based on block category

Besides coloring based on MATLAB/Simulink block reference type and category, graph coloring technique is also implemented in this feature were the connected nodes always have different colors. It addresses the cyclic data dependency pattern that might cause data race and highlight parts that more attention when transforming the components into AUTOSAR components.

- **AUTOSAR Model Assessment**

  This feature provides general information on the building block of an AUTOSAR model as can be seen in figure 15. This tool created to get familiar with AUTOSAR file structure by showing several samples of the valid AUTOSAR files and analyze its structure. Along with AUTOSAR modeling guideline [AUTd], this feature

provides some ideas on the strategy that can be employed to structure the EMF model into a valid AUTOSAR file.

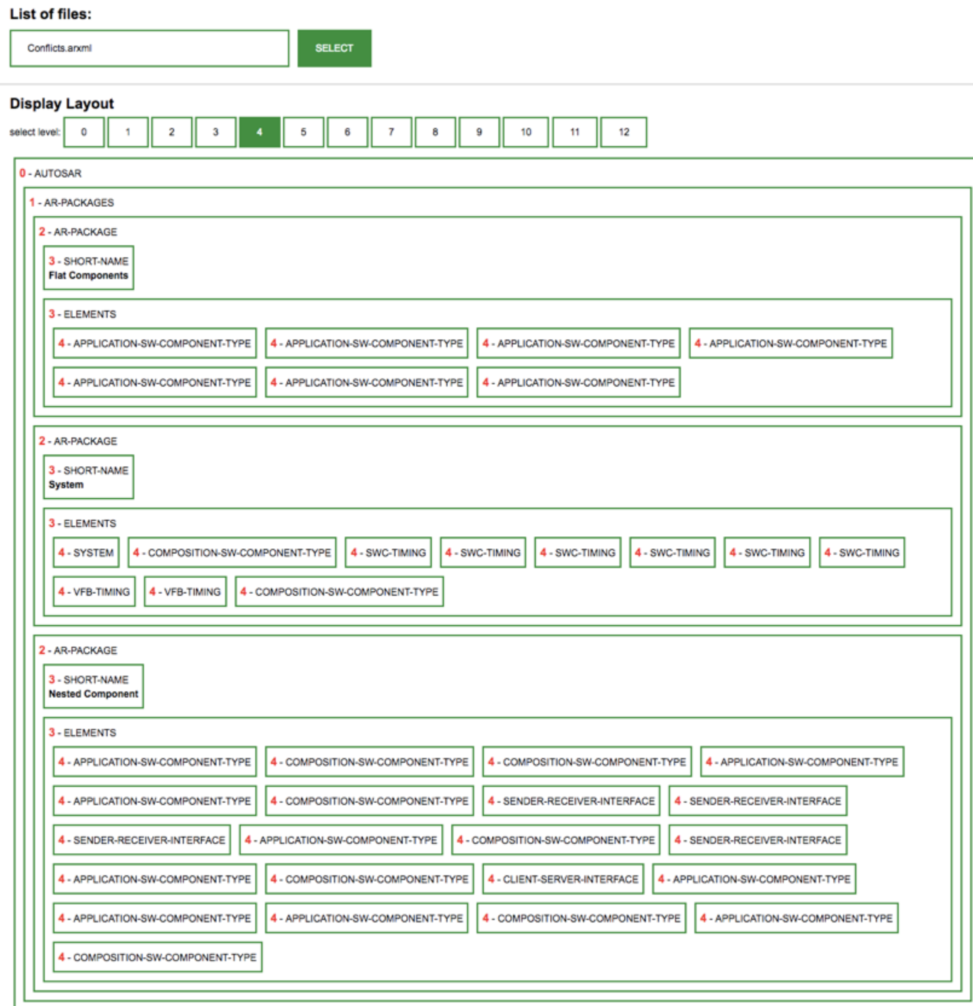


Figure 15. AUTOSAR Model Assessment feature shows AUTOSAR components in block structure and its depth level

There's no further improvement for this feature because **AutoAnalyze** software can provide better AUTOSAR file validation and comprehensive visualization. The feature is kept for debugging purpose of retracing the AUTOSAR XML components validity when the **AutoAnalyze** application can't read the generated AUTOSAR file.

5.2 AUTOSAR Generator

AUTOSAR Generator is a straightforward Java program without any interactive graphical user interface to convert the EMF model generated from the MATLAB/Simulink file. This application uses Java programming language since the AUTOSAR library provided by AUTOSAR Consortium is already in Java, so it is more convenient for accessing valid AUTOSAR version 4.x components.

Since the official AUTOSAR documentation shows there's no strict instruction on how each MATLAB/Simulink components can be mapped into an AUTOSAR and the document only gives MATLAB/Simulink to AUTOSAR model conversion for version 3.1 [AUTa], there are a lot of researches, improvisations, and iterations to fill the gap. MATLAB/Simulink creator already has a product named Embedded Coder [Thea] that can transform MATLAB/Simulink file into an AUTOSAR file, but this product is only available for their industry or academic clients focused on automotive software engineering. Based on the Embedded Coder documentation, users need to decide which Simulink component transformed into what AUTOSAR component. It shows that there are still many rooms for improvement for AUTOSAR model transformation especially in the open source area.

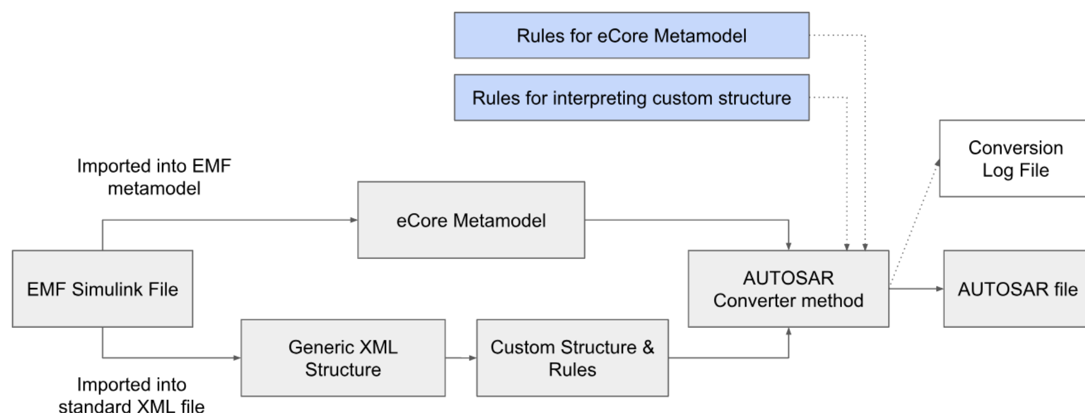


Figure 16. AUTOSAR generator program stucture

The benefit of transforming the Simulink file to EMF model is that it can be transformed into EMF eCore metamodel. By using metamodel, accessing eSimulink system and subsystem classes for a direct component to component transformation can be much easier to do. It is a straightforward conversion action if the counterpart AUTOSAR components are already defined. However, this method is unsuitable if there are a lot of experimentations on assembling the AUTOSAR structure. The second process inside this program was reading the Simulink EMF file into generic XML and create a custom

structure as it can be seen in figure 16. This process gives more freedom to create experiments on the AUTOSAR file result.

The custom structure requires back and forth validation with the graph model in the EMF assessment application. EMF model visualization was improved several times to give a better understanding of the component interactions. The custom structure is necessary to discard or merged components so it won't be translated into AUTOSAR component or transformed into an AUTOSAR component if it matches the rule given. Without the custom structure, the transformation result was a valid but nonsensical AUTOSAR file since it doesn't represent how the system works. Modifying custom structure also provides more options to do the conversion. There are a lot of things that can be done using custom structure and the current state of the software is less than ideal where the structure is hardcoded and tuned into Simulink RISA structure.

The first appendix of this thesis covers the detailed walkthrough of the model assessment application and AUTOSAR generator application. Each feature inside model assessment application and the model transformation result is explained in this part along with the relevant visualization results. In the AUTOSAR generator, since it doesn't have a user interface, figure 29 describes the start difference between not using custom structure and using custom structure.

6 Result & Validation

The result of this project comprises of the step by step journey for MATLAB/Simulink to AUTOSAR model transformation along with the analysis involved, supporting applications, and the AUTOSAR compliant file result. The model conversion processes described in this project be repeated iteratively to create more improvements in the future.

- **Model Assessment Result**

  The web application software developed for model assessment is heavily focused on the EMF model assessment because the generated EMF file and **AutoAnalyze** application already cover everything to assess the AUTOSAR file. The main challenge is to understand the system better and create a reasonable conversion rule based on it. Because of it, the application has undergone multiple iterations to improve components visibility and highlight the interconnected components. The application shows that the target RISA MATLAB/Simulink file is quite complex in terms of depth and also components interaction.

  Model assessment application maps complex component interaction inside every subsystem into a color-coded force diagram. Several color code rules such as coloring based on its Simulink block category and graph coloring technique are utilized to give better information on the structure of the system. Different color coding method serves a different purpose on the component transformation decision with all methods complement each other to fill in the gap where the other methods are lacking.

  Sanity testing is used continuously in building model assessment application to find missing components or structure during the conversion process. Massif discarded components that are not part of basic MATLAB/Simulink component such as Stateflow and HDL Coder and this may or may not important for the AUTOSAR result file. Data dependency analysis can be seen in the graph coloring assessment where subsystem with complex data flow can be seen at a glance. The future use of this technique is to regulate the scheduling analysis that is not implemented in this thesis. Figures below show one of the complex subsystems structure.

- **AUTOSAR file generator result**

  The goal of this thesis is to be able to produce an AUTOSAR compliant file from the MATLAB/Simulink file. Since the transformation from MATLAB/Simulink component into AUTOSAR component is not defined strictly by the AUTOSAR Consortium, this thesis focus on iteration and experimentation on how to produce a valid AUTOSAR file that can represent the system. This thesis relies on **AutoAnalyze** software developed by Augsburg University to verify the final AUTOSAR

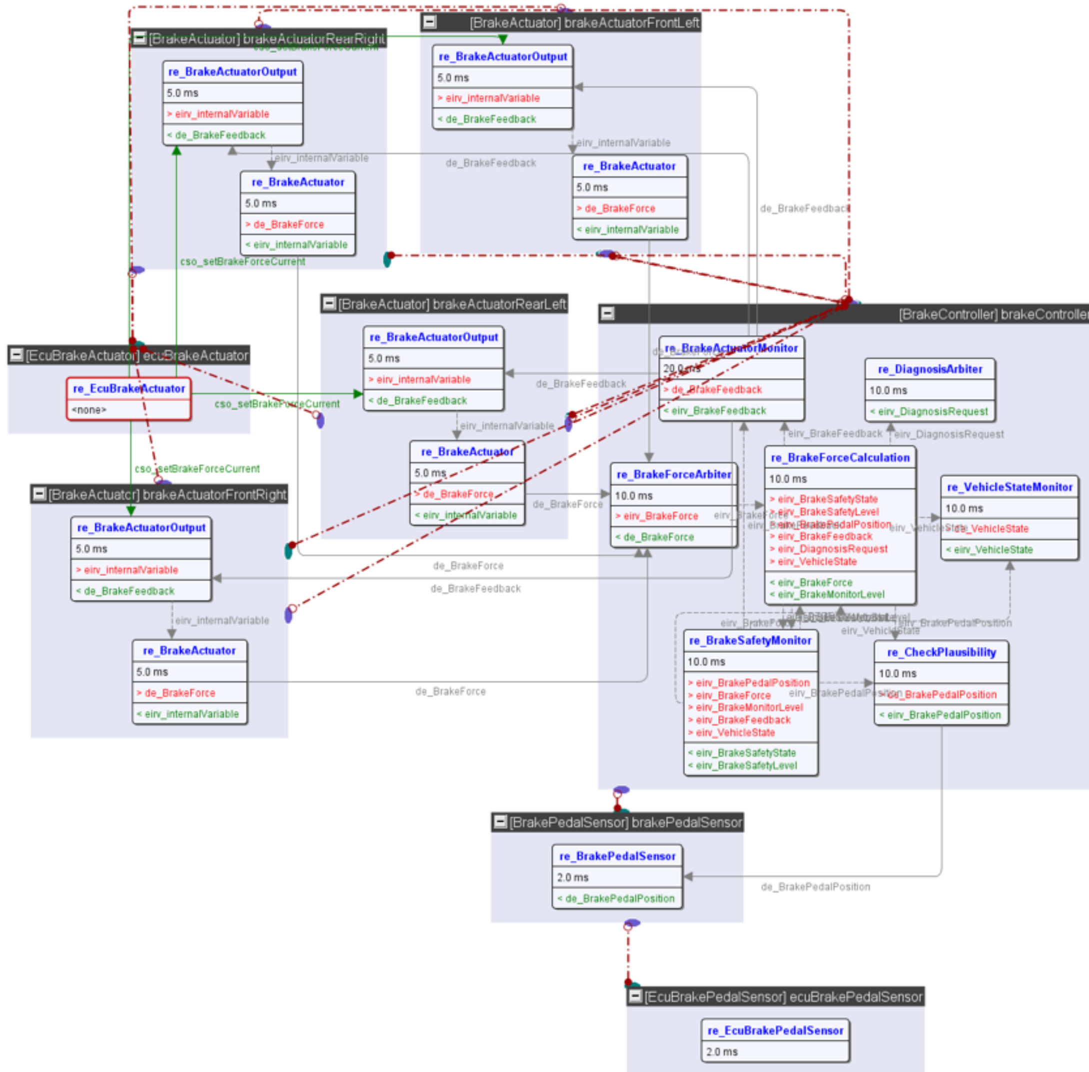


Figure 17. AUTOSAR model visualization in AutoAnalyze

file generated by the application. Visualization and verification of **AutoAnalyze** software are in figure 17.

During AUTOSAR file generation, a valid AUTOSAR file can look nonsensical and don't represent the system correctly. In order to shape the AUTOSAR end result to look more reasonable, the AUTOSAR generator application include custom structure from imported generic XML structure of the Simulink EMF model. This generic XML structure is rearranged to be able to detect sequential Simulink blocks that consist of mathematical or logical functions data block. These

components are then assembled into a runnable entity. Figure 18 shows the idea of how transformation for sequential blocks is performed.

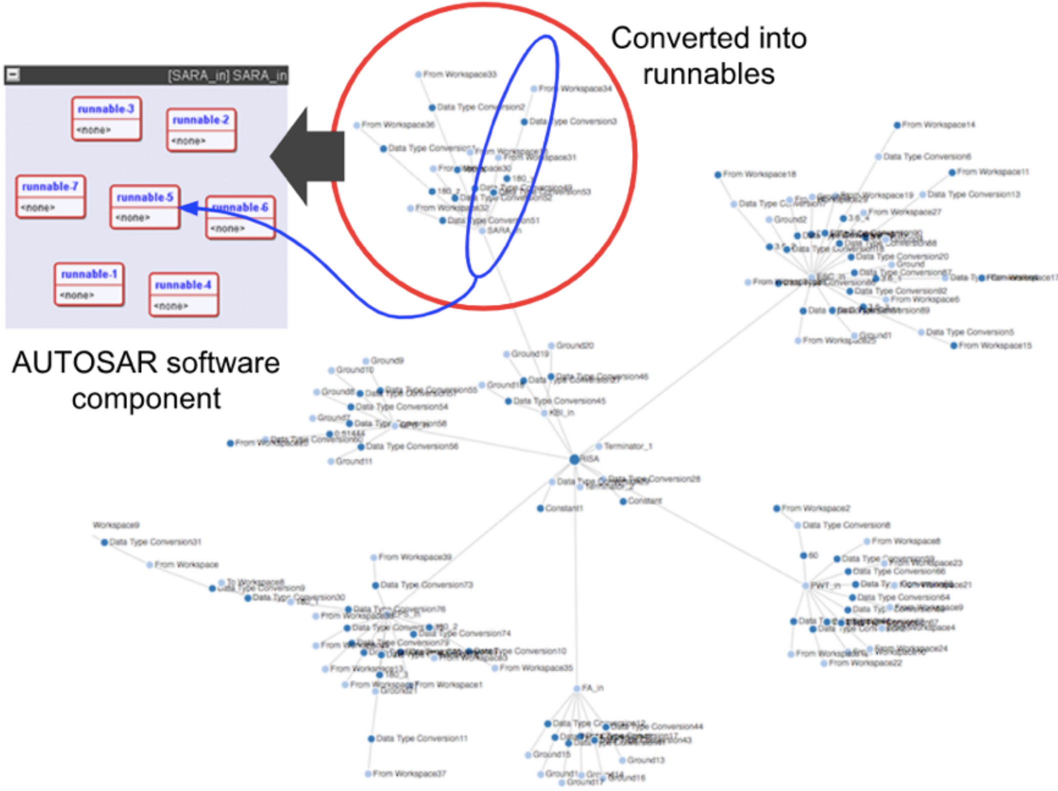


Figure 18. Custom structure that converts sequence of controls into a runnable entity

One of the custom structure runs recursively through the system and subsystems then generates separate AUTOSAR file based on the subsystem. This custom structure works on a complex subsystem but doesn't make sense for a simple subsystem that only has one or two blocks. It also not suitable for components with cyclic connections between variables. Another custom structure experimentation creates a nested component from the deepest level of the MATLAB/Simulink model until the top level of the component that produces a single AUTOSAR file. These custom structure experimentations give different insights on the way components can be transformed into an AUTOSAR model. The actual deployment to the automotive vehicle depends on the specification of the car manufacturers and it's is outside the scope of this thesis project

7 Conclusions & Future Improvements

7.1 Conclusions

Model-driven development and analysis have a vital role in automotive software engineering due to the black-box nature of the components integrated inside a vehicle. The automotive software engineering adopts various modeling concepts such as control theory model from mechanical engineering discipline and information processing model from business IT. Both models are complementing each other to provide a better way to understand a complex system inside an automotive vehicle

This thesis can meet the initial goal to convert MATLAB/Simulink model into an AUTOSAR compliant file. The experimentation to create the conversion processes is largely done on a MATLAB/Simulink file named RISA that is provided by Augsburg University. The first step was creating a web application to extract MATLAB/Simulink file content and structure. The web application is also able to check missing components as the result of the conversion to the EMF model. Although several parts are missing during conversion into the EMF model, it's not vital, and the conversion result can still be used to convert the model into the AUTOSAR model. The EMF model still retains the overall hierarchical structure of the essential MATLAB/Simulink component interactions.

The EMF model was visualized in various manners, and the visualization features have undergone several iterations. This continuous improvement is necessary to provide as many as possible useful insights that can be utilized for converting the model into the final product, AUTOSAR compliant file. The insights from the EMF model visualizations are integrated into custom structure code in the AUTOSAR file generator application. The AUTOSAR generator application is a straightforward Java application without a user interface that converts the EMF model into an AUTOSAR compliant file. The result is several valid AUTOSAR files from different custom structures that can be validated using **AutoAnalyze** application. Both model visualization and AUTOSAR generator are described in detail inside application walkthrough appendix.

Within the development process, the transformation and visualization of the Simulink data into EMF model managed to provide various insights on the way a model can be explored. Instead of sticking into a block diagram structure style in Simulink, transforming the information processing model into another form like force diagram gave a fresh point of view to analyze and transform the component into the AUTOSAR model. Exploration with the color code and graph coloring technique provide early information on data dependency and data race pitfalls even before the code is generated into the AUTOSAR model. The fact that there's no strict guide to convert MATLAB/Simulink model into AUTOSAR model provides an opportunity for further exploration, but it is also problematic since the correct solution entirely depends on the automotive vehicle manufacturers to deploy the final result into the actual machine. As a result, the current

result of the AUTOSAR generated file is too broad and has many assumptions.

7.2 Future Improvements

There are a lot of spaces for future improvements for this project. In the technical side, creating a graphical user interface for the AUTOSAR generator software to create custom structure rules and insert model dynamically can be a priority. Creating a tool chaining from model assessment application into AUTOSAR file generator application also can be an interesting future project to make model transformation smoother. Testing with more MATLAB/Simulink files can help to find the current modeling process shortcomings along with implementing more sophisticated testing method than sanity testing and *happens-before* data dependency analysis.

References

[AHKPM05] Jaswinder Ahluwalia, Ingolf H. Krüger, Walter Phillips, and Michael Meisinger. Model-based run-time monitoring of end-to-end deadlines. pages 100–109, 01 2005.

[AUTa] AUTOSAR Consortium. Applying Simulink to AUTOSAR. https://www.autosar.org/fileadmin/user_upload/standards/classic/3-1/AUTOSAR_SimulinkStyleguide.pdf.

[AUTb] AUTOSAR Consortium. AUTOSAR. https://www.autosar.org/.

[AUTc] AUTOSAR Consortium. Modeling Guidelines of Basic Software EA UML Model. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-2/AUTOSAR_TR_BSWUMLModelModelingGuide.pdf.

[AUTd] AUTOSAR Consortium. Software Component Template. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_SoftwareComponentTemplate.pdf.

[AUT18] AUTOSAR Consortium. SW-C and System Modeling Guide. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_SWCModelingGuide.pdf, 2018.

[BBR+05] Andreas Bauer, Manfred Broy, Jan Romberg, Bernhard Schätz, Peter Braun, Ulrich Freund, Nuria Mata, Robert Sandner, and Dirk Ziegenbein. AutoMoDe - Notations, Methods, and Tools for Model-Based Development of Automotive Software. In *SAE Technical Paper*. SAE International, 04 2005.

[Bec] Nels E. Beckman. A Survey of Methods for Preventing Race Conditions. http://www.cs.cmu.edu/~nbeckman/papers/race_detection_survey.pdf.

[Bro03] Manfred Broy. Automotive Software Engineering. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 719–720, Washington, DC, USA, 2003. IEEE Computer Society.

[Bro06] Manfred Broy. Challenges in Automotive Software Engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 33–42, New York, NY, USA, 2006. ACM.

[EJ09] Christof Ebert and Capers Jones. Embedded Software: Facts, Figures, and Future. *Computer*, 42(4):42–52, April 2009.

[FGI+05] L. Fanucci, A. Giambastiani, F. Iozzi, C. Marino, and A. Rocchi. Platform Based Design for Automotive Sensor Conditioning. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 3*, DATE '05, pages 186–191, Washington, DC, USA, 2005. IEEE Computer Society.

[FSN+03a] Joakim Fröberg, Kristian Sandström, Christer Norström, Hans Hansson, Jakob Axelsson, and Björn Villing. Correlating Business Needs and Network Architectures in Automotive Applications - a Comparative Case Study. In *Proceedings of the 5th IFAC International Conference on Fieldbus Systems and their Applications (FET)*, pages 219–228. IFAC, July 2003.

[FSN+03b] Joakim Fröberg, Kristian Sandström, Christer Norström, Björn Villing, and Jakob Axelsson. A Comparative Case Study of Distributed Network Architectures for Different Automotive Applications. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-69/2003-1-SE, Mälardalens högskola, February 2003.

[Ges11] Gesellschaft für Informatik. AUTOSAR – The Standardized Software Architecture. `https://gi.de/informatiklexikon/autosar-the-standardized-software-architecture/`, 2011.

[GFL+02] P. Giusto, A. Ferrari, L. Lavagno, J. . Brunel, E. Fourgeau, and A. Sangiovanni-Vincentelli. Automotive virtual integration platforms: why's, what's, and how's. In *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 370–378, Sep. 2002.

[Ha13] Ok-Kyoon Ha. Case Study of Dynamic Detectors for Data Races. *IERI Procedia*, 4:174 – 180, 2013. 2013 International Conference on Electronic Engineering and Computer Science (EECS 2013).

[HJM04] Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Race Checking by Context Inference. *SIGPLAN Not.*, 39(6):1–13, June 2004.

[HRS] A. Horvath, I. Rath, and R Starr. Massif - the love child of Matlab Simulink and Eclipse. `https://www.eclipsecon.org/na2015/sites/default/files/slides/massif_eclipsecon_15_ha.pdf`.

[IT99] ITU-TS. Z.120: Message Sequence Chart (MSC). `https://www.itu.int/rec/T-REC-Z.120`, 1999.

[KSS+17] Julian Kienberger, Stefan Schmidhuber, Christian Saad, Stefan Kuntz, and Bernhard Bauer. Parallelizing highly complex engine management systems. *Concurrency and Computation: Practice and Experience*, 29(15), 2017.

[Lew15] R.M. R. Lewis. *A Guide to Graph Colouring: Algorithms and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[LH02] Gabriel Leen and Donal Heffernan. Expanding Automotive Electronic Systems. *Computer*, 35(1):88–93, January 2002.

[MT] B. Messner and D. Tilbury. Control Tutorials for MATLAB and Simulink (CTMS). `http://ctms.engin.umich.edu/`.

[NP04] Edward C. Nelson and K. Venkatesh Prasad. Service-Based Software Development for Automotive Applications. 2004.

[RBvdBS02] Martin Rappl, Peter Braun, Michael von der Beeck, and Christian Schröder. Automotive Software Development: A Model Based Approach. In *SAE Technical Paper*. SAE International, 03 2002.

[RFH+05] Sabine Rittmann, Andreas Fleischmann, Judith Hartmann, Christian Pfaller, Martin Rappl, and Doris Wild. Integrating Service Specifications at Different Levels of Abstraction. In *Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, 2005.

[Sch05] Thomas Scharnhost. AUTomotive Open System ARchitecture (AUTOSAR): An Industry-wide Initiative to Manage the Complexity of Emerging E/E Architectures. `https://www.itu.int/ITU-T/worksem/ict-auto/200503/presentations/s6-scharnhost.pdf`, 2005.

[SS04] Christian Salzmann and Thomas Stauner. *Automotive Software Engineering*, pages 333–347. Springer US, Boston, MA, 2004.

[Thea] The MathWorks, Inc. Embedded Coder - MATLAB & Simulink. `https://www.mathworks.com/products/embedded-coder.html`.

[Theb] The MathWorks, Inc. HDL Coder - MATLAB & Simulink. `https://www.mathworks.com/products/hdl-coder.html`.

[Thec] The MathWorks, Inc. Simulink - Simulation and Model-Based Design - MATLAB & Simulink. `https://www.mathworks.com/products/simulink.html`.

[Thed] The MathWorks, Inc. Stateflow - MATLAB & Simulink. `https://www.mathworks.com/products/stateflow.html`.

[TKWE03] Ken Tindell, Hermann Kopetz, Fabian Wolf, and Rolf Ernst. Safe Automotive Software Development. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10616–, Washington, DC, USA, 2003. IEEE Computer Society.

[Via] Viatra. GitHub - viatra/massif: Massif is a Matlab Simulink Integration Framework for Eclipse. `https://github.com/viatra/massif`.

[WM95] Robert W. Weeks and John J. Moskwa. Automotive Engine Modeling for Real-Time Control Using MATLAB/SIMULINK. In *SAE Technical Paper*. SAE International, 02 1995.

[ZKP+08] R. S. Zybin, V. V. Kuliamin, A. V. Ponomarenko, V. V. Rubanov, and E. S. Chernov. Automation of broad sanity test generation. *Programming and Computer Software*, 34(6):351–363, Nov 2008.

Appendix

I. Application Walkthrough

The target Simulink model in this project is a specific Simulink file named RISA. This Simulink file describes the control flow of a braking system inside an automotive vehicle provided by Augsburg University named RISA. RISA MATLAB/Simulink file has 1820 components excluding port connections and also 85 subsystems and sub-subsystems. The first level of the system opened using its native application can be seen from figure 19 below.

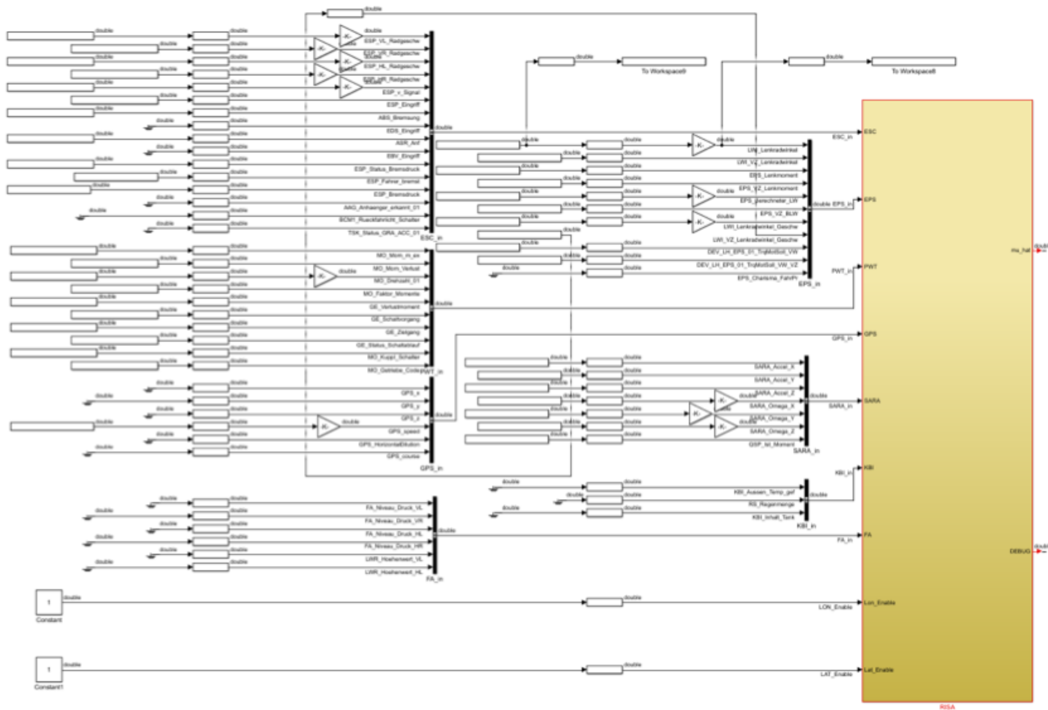


Figure 19. First level visualization of RISA MATLAB/Simulink model in MATLAB application

MATLAB/Simulink application doesn't have much flexibility to skim the whole components at a glance which makes it difficult to start the project. A separate web application is created to overcome this problem. This application can do a thorough assessment of the file such as providing the model's hierarchical structure, the list of the components, and other custom views to understand the model better. The user interface of the application can be seen in figure 20 where the left side shows the file structure, the

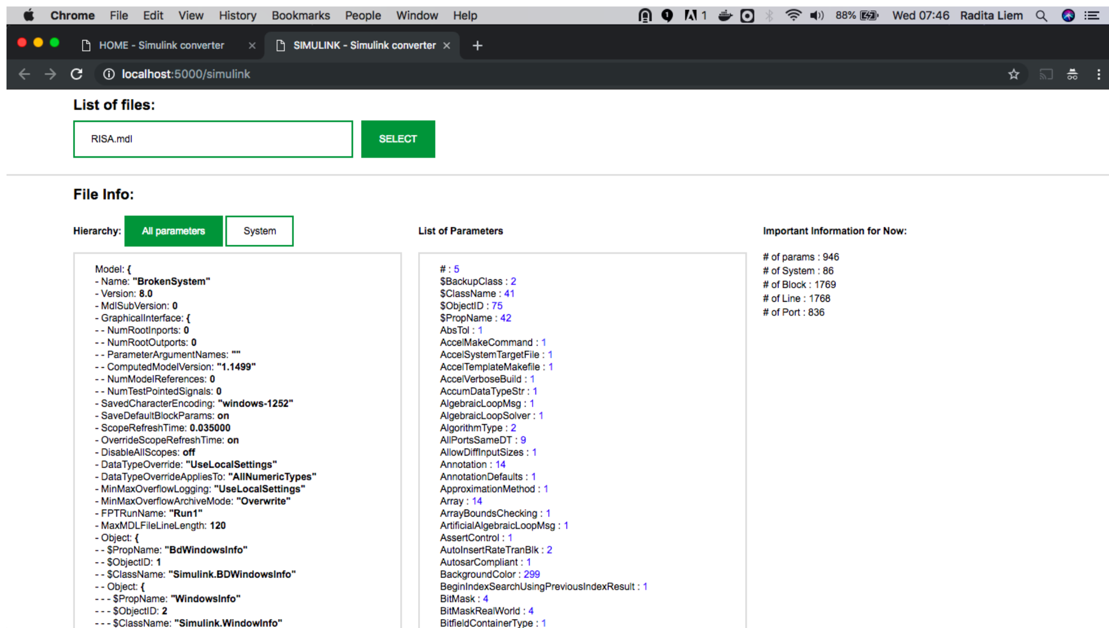


Figure 20. Custom reader application for MATLAB/Simulink shows different assessment on the components inside RISA file

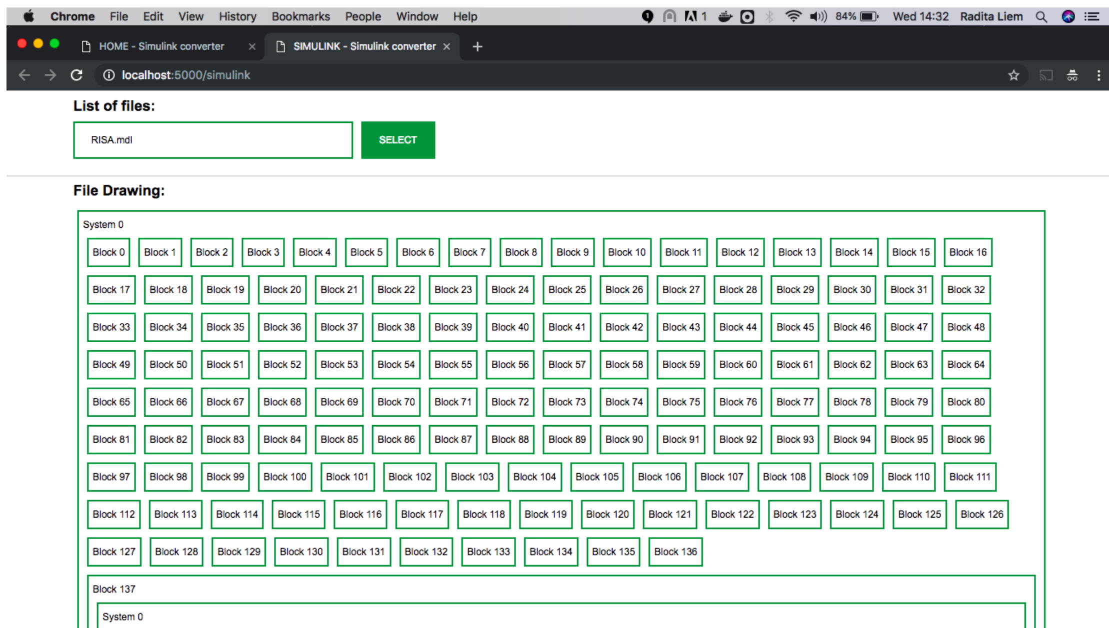


Figure 21. Block diagram created by custom reader application for MATLAB/Simulink

middle part shows all unique components inside the model, and in the right side is the ad-hoc script result to display relevant information.

This application also tries to recreate block component visualization directly from MATLAB/Simulink file, but the result is far inferior compared to the visualization using EMF model generated by Massif as it can be seen in figure 24. The EMF model produces the same structure as the structure built using custom MATLAB/Simulink reader application that can be seen in figure 24. Since EMF model is an XML file that is much easier to read and manipulate, the further improvement for custom MATLAB/Simulink reader block visualization is stopped, and the development is focused more on the EMF model visualization.

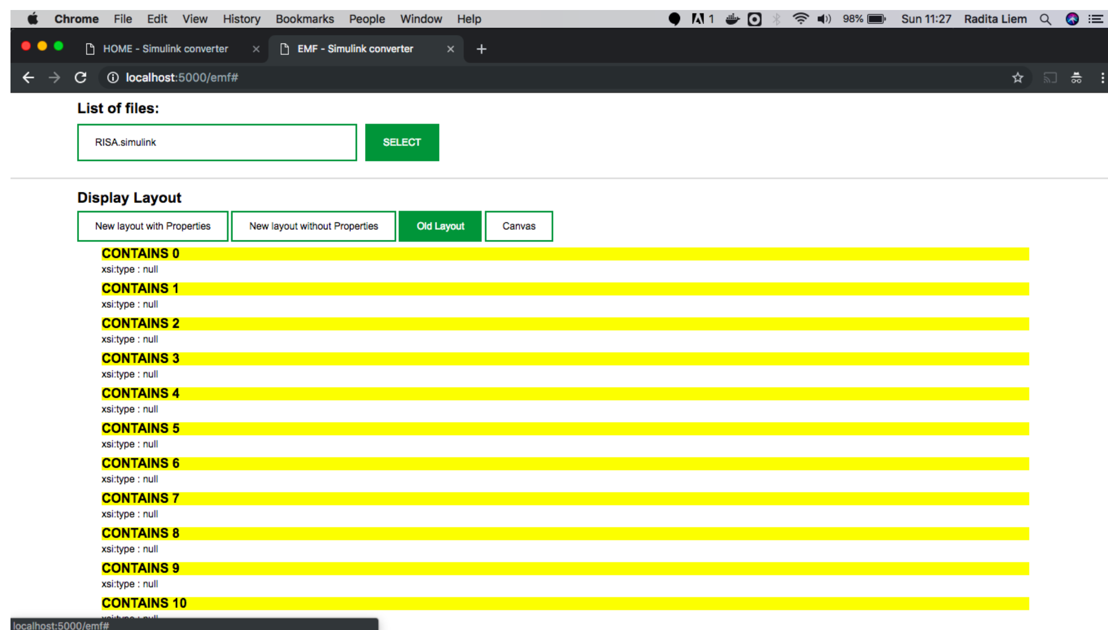


Figure 22. The first version of the EMF model visualization where all information of the block component is stored inside yellow dropdown box

Custom MATLAB/Simulink application also runs a comparison between converted EMF model and the original MATLAB/Simulink model. The result shows that the generated EMF model retains the structure to produce block diagrams but missing additional components that's not part of the basic MATLAB/Simulink application such as Stateflow and HDL Coder items. Since these components are not that crucial for the conversion, the EMF model is still used in the conversion to AUTOSAR file. Ideally, Massif can be improved to retain these components since it's an open source project.

EMF model visualization has undergone multiple iterations that can be seen in figure 22 - 24. Figure 22 is the initial version of the visualization where subsystems are stored

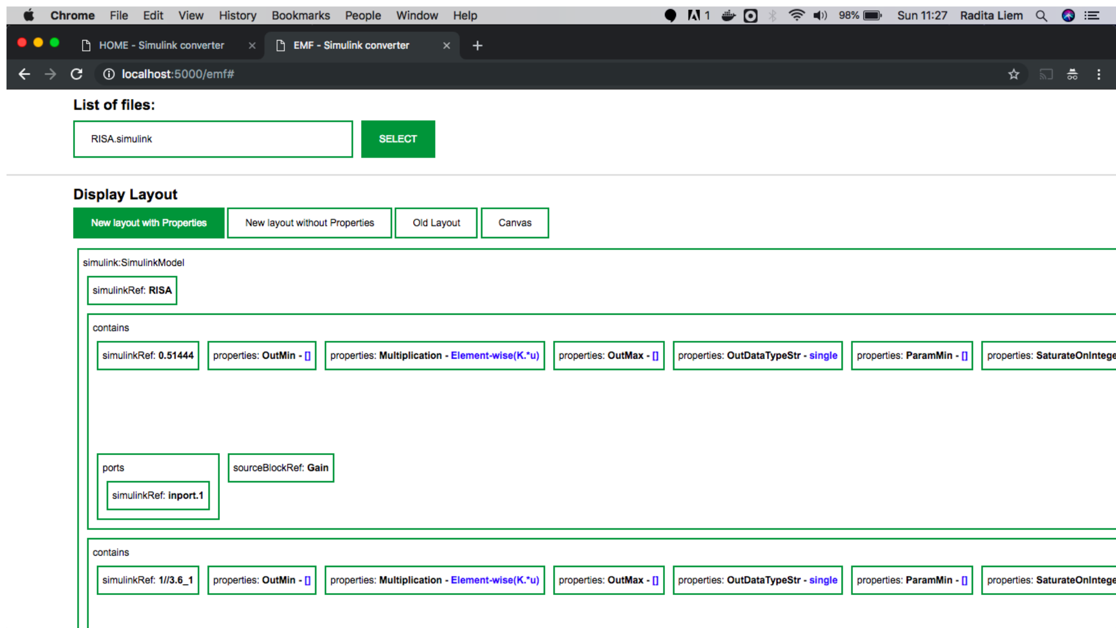


Figure 23. The second iteration where all components are shown inside nested blocks

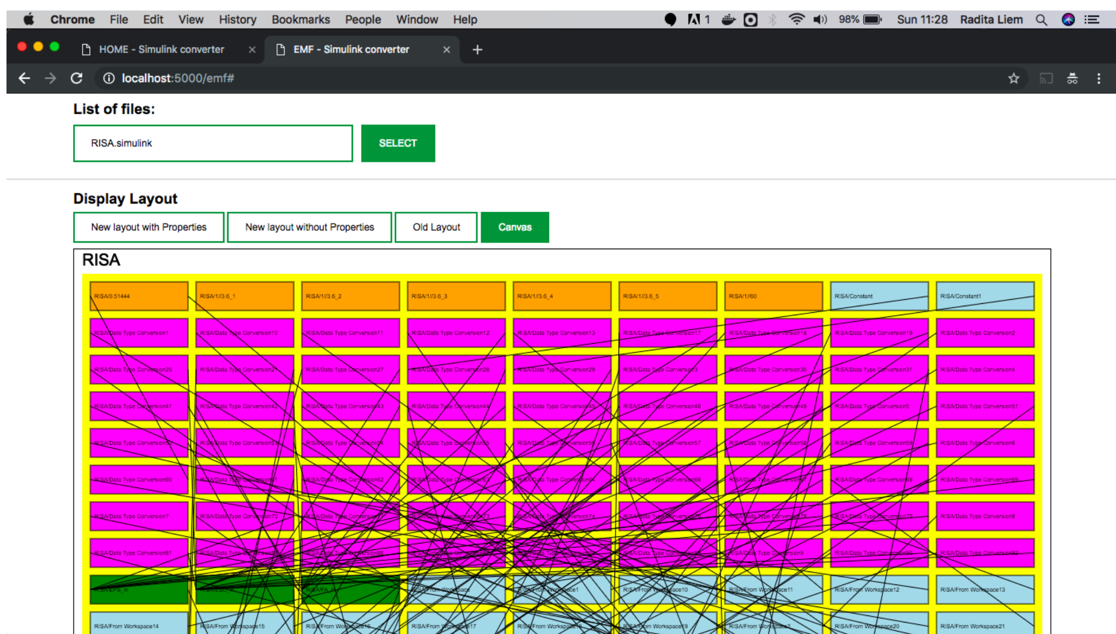


Figure 24. Third iteration removes non-essential information and starts to connect components. The block color is based on its MATLAB/Simulink category

in dropdown boxes. It was difficult to comprehend the system's depth using this method and figure 23 is trying to answer this shortcoming by displaying everything without dropdown. This version also not very good since there's too much information and also tricky to skim the system since there's a long scrolling to get to the end of the block structure. In figure 24, the block diagram is much more readable compared to the previous iteration after discarding much of the non-essential information. However, this visualization is also not yet satisfying since it is still difficult to see component interactions. Coloring components based on its MATLAB/Simulink category helps a little bit to see what's happening within the system.

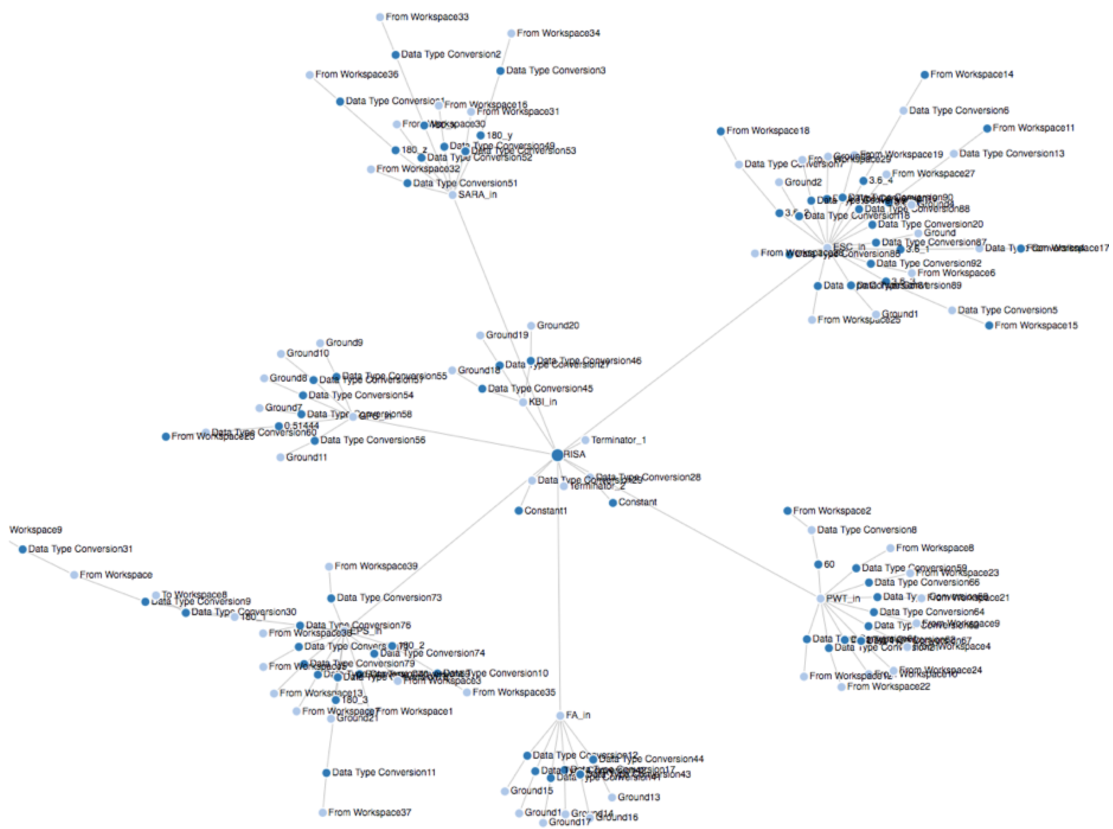


Figure 25. Graph coloring technique doesn't show an interesting view in the first level of the system

The final version can be seen in figure 27 where it's decided to used the D3 visualization library. The depth of the system is represented using a hierarchy diagram and the connection between components is described using a force diagram. Users also can check the structure of each subsystem. This visualization is much easier to understand

with each node is colored based on certain rules such as based on its MATLAB/Simulink category or based on graph coloring technique.

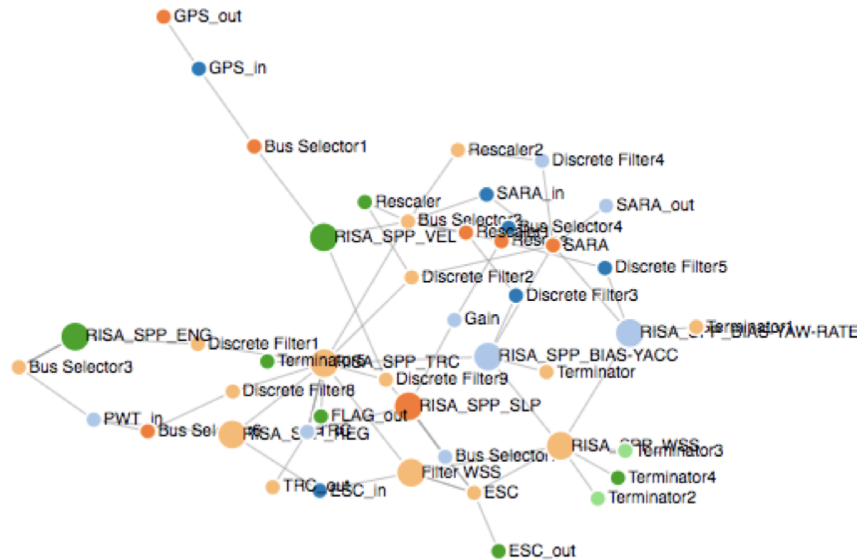


Figure 26. Components colored with graph coloring technique only works for complex interconnected component

Different coloring method is useful for different cases. In figure 25, the first level RISA system components are colored using a graph coloring technique, and it's not showing anything interesting. Coloring with graph coloring technique is much more reasonable for complex interconnected components such as RISA_SPP subsystem that can be seen in figure 26. Because of this reason, several coloring modes are kept inside the application to provide understanding depending on the case.

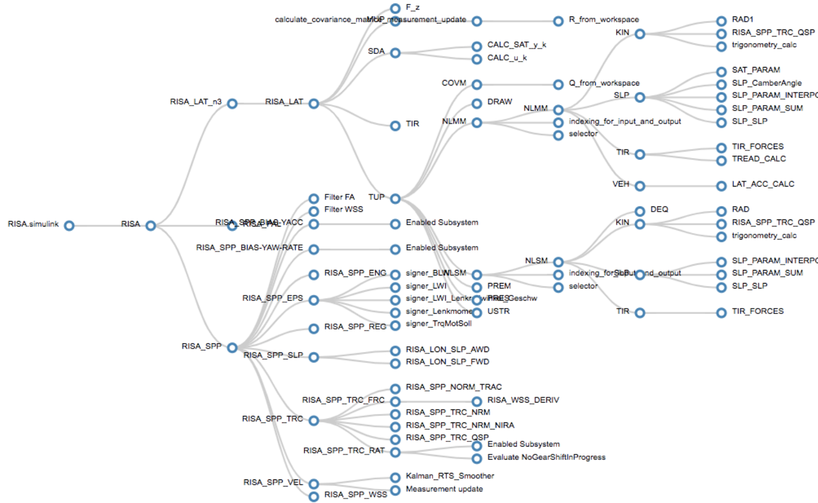
The last feature in the assessment application is AUTOSAR reader, this feature is initially created to get familiar with the AUTOSAR environment. Later this feature is used to do AUTOSAR file debugging when the **AutoAnalyze** application can't open the AUTOSAR file to check components' structure validity. The user interface can be seen in figure 28 where it shows AUTOSAR component blocks in different granularity level.

Since there's no user interface for AUTOSAR generator, the general concept of the application has been explained in figure 16 in chapter 5.2. The initial version of the application is a valid AUTOSAR file that can be read by **AutoAnalyze** software but nonsensical as it can be seen in the left-hand side of figure 29. The implementation of custom structure creates a much more reasonable AUTOSAR file that can be seen in the right-hand side of figure 29. Figure 30 shows the user interface of the **AutoAnalyze** software

List of files:

RISA.simulink SELECT

Hierarchy Structure



Levels: RISA SELECT

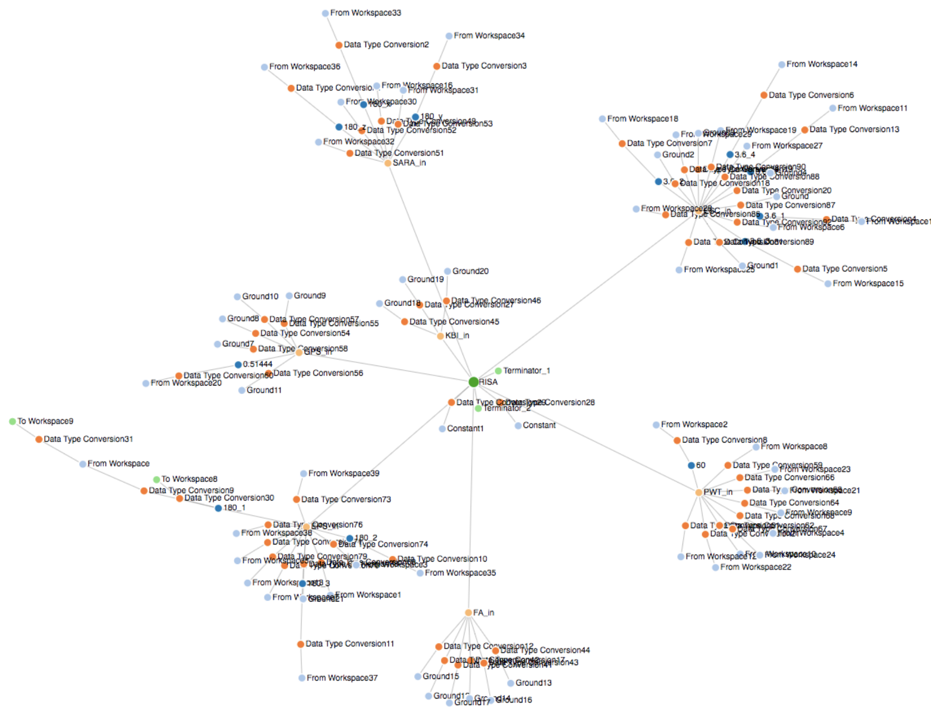


Figure 27. Final EMF visualization using D3 Javascript library. System depth visualized using hierarchy diagram and component interaction with force diagram

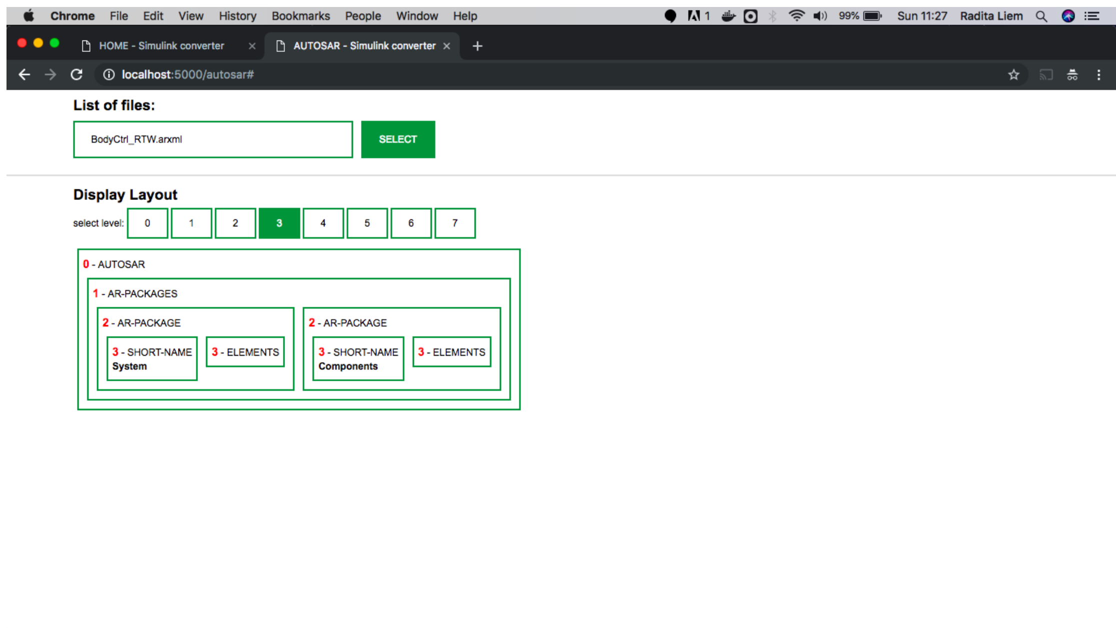


Figure 28. Custom AUTOSAR components visualization

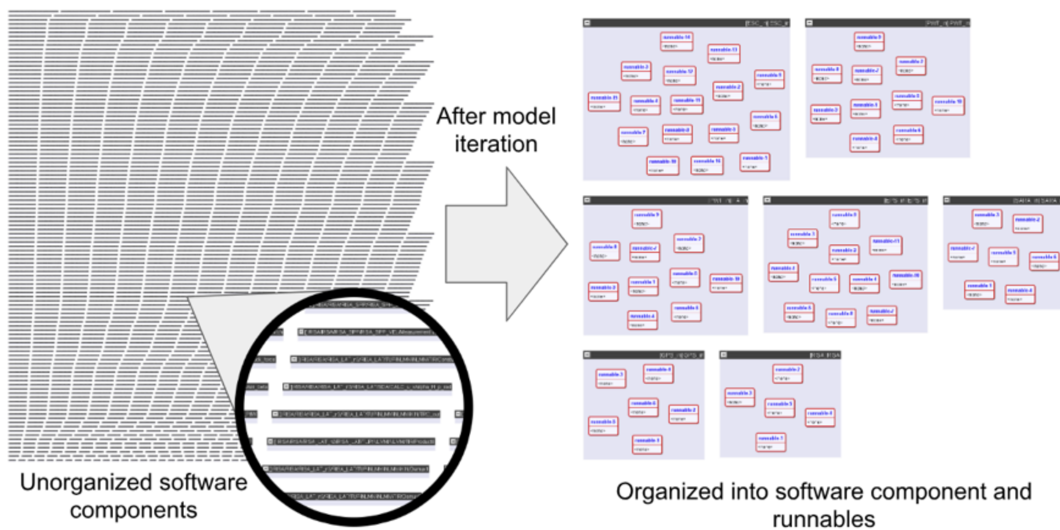


Figure 29. Comparison between the initial direct component conversion in AUTOSAR and the component conversion using custom structure

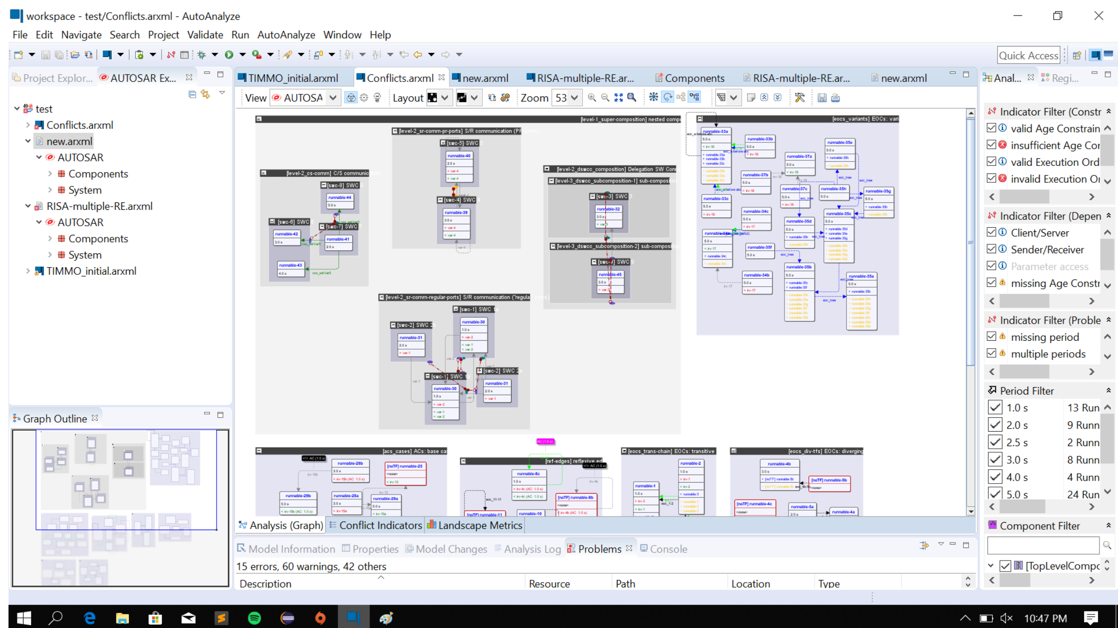


Figure 30. User interface of the AutoAnalyze software

II. List of Reference Tables

Below's table is the Simulink blocks inside RISA Simulink file. It provides information on the block functionality for consideration when transforming the MATLAB/Simulink model into AUTOSAR model.

Table 1. Extracted Simulink Block Libraries

Name	Description
Continuous	Continuous function blocks such as Derivative and Integrator
Dashboard	The Dashboard Scope block shows connected signals during simulation on a scope display.
Discontinuities	Discontinuous function blocks such as Saturation
Discrete	Blocks in the Discrete library that are optimized for HDL Code generation
Logic and Bit Operations	Logic or bit operation blocks such as Logical Operator and Relational Operator
Lookup Tables	Lookup table blocks such as Cosine and Sine
Math Operations	Mathematical function blocks such as Gain, Product, and Sum
Model Verification	Blocks for self-verifying models, such as Check Input Resolution
Model-Wide Utilities	Model-wide operation blocks such as Model Info and Block Support Table
Ports and Subsystems	Blocks related to subsystems, such as Inport, Outport, Subsystem, and Model
Signal Attributes	Modify signal attribute blocks such as Data Type Conversion
Signal Routing	Route signal blocks such as Bus Creator and Switch
Sinks	Display or export signal data blocks such as Scope and To Workspace
Sources	Generate or import signal data blocks such Sine Wave and From Workspace
String	String manipulation blocks
User-Defined Functions	Custom function blocks such as MATLAB Function, MATLAB System, Simulink Function, and Initialize Function
Additional Math and Discrete	Mathematical and discrete function blocks such as Decrement Stored Integer
HDL Coder	HDL-optimized blocks

Below are all block details inside the RISA MATLAB/Simulink file of the Simulink block libraries listed in table one.

Table 2. Extracted Blocks and Its Description

Blocks	**Description**	**Block Libraries**
1-D Lookup Table	Approximate one dimensional function	Lookup Tables
Abs	The Abs block outputs the absolute value of the input.	Math Operations
Assignment	The Assignment block assigns values to specified elements of the signal.	Math Operations
Bitwise Operator	The Bitwise Operator block performs the bitwise operation specified on one or more operands. Unlike logic operations of the logical operator block, bitwise operations treat the operands as a vector of bits rather than a single value.	Logic and Bit Operations
Bus Creator	The Bus Creator block combines a set of signals into a bus.	Composite Signals
Bus Selector	The Bus Selector block outputs a specified subset of the elements of the bus at its input. The block can output the specified elements as separate signals or as a new bus.	Composite Signals
Constant	The Constant block generates a real or complex constant value. The block generates scalar, vector, or matrix output, depending on the dimensionality of the constant value parameter and the setting of the interpret vector parameters as one dimension parameter	Sources
Data Store Memory	The Data Store Memory block defines and initializes a named shared data store, which is a memory region usable by Data Store Read and Data Store Write blocks that specify the same data store name.	Data Stores
Data Type Conversion	The Data Type Conversion block converts an input signal of any Simulink data type to another specified data type.	HDL Floating Point Operations

Continued on next page

Table 2 – *Continued from previous page*

Blocks	**Description**	**Block Libraries**
Demux	The Demux block extracts the components of an input vector signal and outputs separate signals. The output signal ports are ordered from top to bottom.	Signal Routing
Discrete Filter	The Discrete Filter block independently filters each channel of the input signal with the specified digital IIR filter	Discrete
Divide	The Divide block' outputs the result of dividing its first input by its second. The inputs can be scalars, a scalar and a nonscalar, or two nonscalars that have the same dimensions.	Math Operations
Enable	The Enable' block allows an external signal to control execution of a subsystem or a model.	Ports and Subsystems
Fcn	The Fcn' block applies the specified mathematical expression to its input.	User-Defined Functions
For Iterator	The For Iterator' block, when placed in a Subsystem block, repeats the execution of a subsystem during the current time step until an iteration variable exceeds the specified iteration limit.	Ports and Subsystems
From Workspace	The From Workspace' block reads signal data from a workspace and outputs the data as a signal.	Sources
Gain	The Gain block multiplies the input by a constant value (gain). The input and the gain can each be a scalar, vector, or matrix.	Math Operations
Ground	The Ground block connects to blocks whose input ports do not connect to other blocks. If you run a simulation with blocks that have unconnected input ports, Simulink issues warnings. Using a Ground block to ground those unconnected blocks can prevent these warnings.	Sources
Inport	Inport blocks are the links from outside a system into the system.	Sources

Continued on next page

Table 2 – *Continued from previous page*

Blocks	**Description**	**Block Libraries**
Logical Operator	Logical Operator block performs the specified logical operation on its inputs. An input value is TRUE (1) if it is nonzero and FALSE (0) if it is zero.	Logic and Bit Operations
Math Function	The Math Function block performs numerous common mathematical functions.	Math Operations
MinMax	The MinMax block outputs either the minimum or the maximum element or elements of the inputs.	Math Operations
Mux	The Mux block combines its inputs into a single vector output. An input can be a scalar or vector signal. All inputs must be of the same data type and numeric type.	Signal Routing
Outport	Outport blocks are the links from a system to a destination outside the system.	Sinks
Reciprocal Sqrt	Calculate square root, signed square root, or reciprocal of square root (HDL Coder)	Math Operations
Relational Operator	Perform specified relational operation on inputs	Logic and Bit Operations
Saturation	The Saturation block produces an output signal that is the value of the input signal bounded to the upper and lower saturation values.	Discontinuities
Selector	The Selector block generates as output selected or reordered elements of an input vector, matrix, or multidimensional signal.	Signal Routing
Subsystem	A Subsystem block contains a subset of blocks within a model or system. The Subsystem block can represent a virtual subsystem or a nonvirtual subsystem.	Ports and Subsystems
Switch	The Switch block passes through the first input or the third input signal based on the value of the second input. The first and third inputs are data input. The second input is a control input.	Signal Routing

Continued on next page

Table 2 – *Continued from previous page*

Blocks	**Description**	**Block Libraries**
Tapped Delay	Tapped Delay block delays an input by the specified number of sample periods and outputs all the delayed versions. This block is used to discretize a signal in time or resample a signal at a different rate.	Discrete
Terminator	Use the Terminator block to cap blocks whose output ports do not connect to other blocks.	Sinks
To Workspace	The To Workspace block inputs a signal and writes the signal data to a workspace. During the simulation, the block writes data to an internal buffer. When the simulation is completed or paused, that data is written to the workspace. Data is not available until the simulation is stopped or paused.	Sinks
Trigonometric Function	The Trigonometric Function block performs common trigonometric functions and outputs the result in rad.	Math Operations
Unit Delay	The Unit Delay block holds and delays its input by the sample period you specify. When placed in an iterator subsystem, it holds and delays its input by one iteration.	Discrete

Below is the list of the tags inside Massif generated EMF file. It comprises all information that matches with the elements inside MATLAB/Simulink file

Table 3. Massif Class name and its explanation

Name	Description
Block	This EClass represents the basic building block of Simulink systems. Each block has properties, ports and can refer to a source block that was used as the template from a library to create the element. The properties are not a map, so the block may have multiple properties with the same name, or even same name-value pair. Attribute name value is computed from the name feature of the SimulinkReference stored in simulinkRef. The different type of ports are accessible through computed filtered lists.
Block (subBlock)	The value of the reference is computed by finding the element with the same name and qualifier as stored in the sourceBlockRef.
BusCreator	This EClass represents a bus creator block that bundles the signals on its inports into a bus on its outport. This reference points to the creator of the incoming bus signal. Either a bus creator or a bus selector with outputAsBus = true. Always determined as backward navigation on signals is deterministic.
BusSelector	This EClass represents a bus selector block that separates the signals from the bus received on its inport into its outports. Since it is possible to select only some of the signals and even embedded signals from a bus inside the bus, mapping entries (BusSignalMapping) are used to indicate which outport (mappingTo) selects which signals originating from a given outport (mappingFrom). The outputAsBus attribute is true if the selected signals are bundled into a bus and placed on a single outport.
BusSignalMapping	This EClass represents a signal mapping entry in the BusSelector to define which signals are selected from a bus.
BusSpecification	This EClass is an abstract supertype for blocks that handle bus signals. A bus signal is used for bundling a set of signals into one signal to reduce the number of ports and connections required in the model.
Connection	This EClass represents the connection between Block elements in order to trans- fer data from an outport to one or more inports.

Continued on next page

Table 3 – *Continued from previous page*

Name	**Description**
Enable	This EClass represents an enable port of a Block. The values are computed by fltering Enable ports from the values of the ports feature.
EnableBlock	This EClass represents a port block of a Enable port with an outport that can be used by blocks inside the subsystem.
EnableStates	This EEnum represents the possible settings of a Enable port for specifying what happens to the states of blocks in the enabled system upon disabling.
From	The From block accepts a signal from a corresponding Goto block, then passes it as output. The data type of the output is the same as that of the input from the Goto block. From and Goto blocks allow you to pass a signal from one block to another without actually connecting them.
Goto	The Goto block passes its input to its corresponding From blocks. From and Goto blocks allow you to pass a signal from one block to another without actually connecting them.
GotoTagVisibility	The Goto Tag Visibility block defines the accessibility of Goto block tags that have scoped visibility. The tag specified as the Goto tag parameter is accessible by From blocks in the same subsystem that contains the Goto Tag Visibility block and in subsystems below it in the model hierarchy.
IdenfitierReference	This is a specific class used as a unique identifier for Simulink elements.
InPort	This EClass represents an inport of a Block. The values are computed by filtering Inports from the values of the ports feature.
InPortBlock	This EClass represents a port block of a InPort with an outport that can be used by blocks inside the subsystem.
LibraryLinkReference	This is a specific class used for representing links to Simulink elements. Disabled links mean that the block was originally copied from a library but it was modified later.
ModelReference	This EClass represents a SimulinkModel included as a block in this model.
MultiConnection	This EClass represents a connection between a single Out-Port and multiple InPort. Each inport is connected by a SingleConnection contained by this con- nection.

Continued on next page

Table 3 – *Continued from previous page*

Name	Description
OutPort	This EClass represents an outport of a Block. The values are computed by filtering Outports from the values of the ports feature.
OutPortBlock	This EClass represents a port block of a OutPort with an inport that can be used by blocks inside the subsystem.
Port	This EClass represents the abstract supertype of block ports that are used for allowing data communication and signaling between blocks.
PortBlock	This EClass represents the abstract supertype of blocks that represent ports of a subsystem.
Property	This EClass represents properties of Block elements. Each property has a name, a type and a value. The value is stored as a character string but is validated based on the type.
SimulinkElement	This EClass represents the abstract supertype of elements in Simulink systems that can be identified uniquely with a fully qualified name consisting of a name and a qualifier.
SimulinkModel	Elements on a Simulink model that can be identified and named are subtypes of SimulinkElement, which stores the identifier as a SimulinkReference element. The root element, SimulinkModel, stores the file path and version for the original Simulink system it represents to help in handling changes in the represented system. The Simulink model contains a hierarchy of Block elements that may have properties and specify a source block from a Simulink library. The source block is set if the internal structure and behavior of the block is defined by a library block. The communication between blocks is done through Port elements, that can be either output or input. Each port is represented by a PortBlock inside the block. The output ports are connected to input ports using Connection elements that can be one-to-one single connections or one-to-many multiconnections.

Continued on next page

Table 3 – *Continued from previous page*

Name	Description
SimulinkReference	This abstract EClass represents a reference for a Simulink element. The identifier is a fully qualified name constructed from a qualifier (the fully qualified name of the parent of the element) and a name. Since a SimulinkElement can be identified based on its fully qualified name, it is possible to reference an element by cloning and storing the reference instead of a direct link to the element itself.
SingleConnection	This represents port connection
SourceBlockRef	This represents original location and block name inside Simulink file
SubSystem	This abstract EClass represents another complex system inside an already existing system
TagVisibility	The Tag Visibility is a parameter of Goto blocks to determine the location of From blocks that access the signal.
Trigger	This EClass represents a trigger port of a Block. The values are computed by filtering Trigger ports from the values of the ports feature.
TriggerBlock	This EClass represents a port block of a Trigger port with an outport that can be used by blocks inside the subsystem.
TriggerType	This EEnum represents the possible events that can trigger the execution of a subsystem with a Trigger port.
VirtualBlock	This EClass represents the abstract supertype of blocks that do not explicitly affect the simulation of the Simulink system. These blocks are called virtual and are added as syntactic sugar, for example Goto and From can be used instead of a direct Connection to connect blocks.

AUTOSAR has provided guidelines to do the transformation from Simulink components to AUTOSAR version 3.x [AUTa]. The complete list of components' equivalent can be found on the table below:

Table 4. Extracted Blocks and Its Description

Blocks	**Description**	**Block Libraries**
Atomic Software Component	Smallest non-dividable software entity, connected to the AUTO- SAR Virtual Functional Bus, relocatable.	Can be represented as any type of subsystem (virtual & non-virtual) also by a model. AUTOSAR's notion of atomic should not to be confused with a Simulink atomic subsystem.
P-Port Provide-Port	Specific Port providing data or providing a service of a server.	Outport for sender/receiver communication
R-Port Require-Port	Specific Port requiring data or requiring a service of a server.	Inport for sender/receiver communication
PortInterface	A PortInterface characterizes the information provided or required by a port. Can be either sender/receiver interface or client/server interface.	Abstract class without realization in Simulink.
ComSpec	ComSpec defines specific communication attributes	No Simulink representation
Sender/Receiver Interface	A sender-receiver interface is a special kind of port-interface used for the case of sender- receiver communication. The sender-receiver interface defines the data-elements which are sent by a sending component (which has a p-port providing the sender-receiver interface) or received by a receiving component (which has an r-port requiring the sender-receiver interface).	BusObject and bus selector/creators
Client/Server Interface	The client-server interface is a special kind of port-interface used for the case of client-server communication. The client-server interface defines the operations that are provided by the server and that can be used by the client.	Specific blocks, realizing RTE-API

Continued on next page

Table 4 – *Continued from previous page*

Blocks	**Description**	**Block Libraries**
Sender Receiver Annotation	Annotation of the data elements in a port that realizes a sender/receiver interface.	Description field assigned to a specific DataElementPrototype inside the Runnable subsystem
Sensor Actuator Software Component	AUTOSAR SWC dedicated to the control of a sensor or actuator.	Virtual subsystem
Services	An AUTOSAR Service is a logical entity of the basic software offering general functionality to be used by various AUTOSAR software components.	Virtual subsystem
Runnable	A Runnable Entity is a part of an Atomic Software-Component which can be executed and scheduled independently from the other Runnable Entities.	Function call subsystem
RTEEvents	An RTEEvent encompasses all possible situations that can trigger execution of a runnable entity by the RTE.	Function calls
Exclusive Areas	Exclusive Areas prevent runnables from being preempted by other runnables.	Atomic subsystem marked as ExclusiveArea.
Composition	Composition encapsulates a collaboration of Components thereby hiding detail and allowing the creation of higher abstraction levels.	Virtual subsystems
Datatypes	AUTOSAR datatypes are either primitive or complex they are used to type data-elements, arguments of the operations in a client-server interface and constants.	Simulink built-in types
Primitive Datatype	All primitive datatypes allow an efficient mapping to programming languages like C	Simulink built-in types

Continued on next page

Table 4 – *Continued from previous page*

Blocks	**Description**	**Block Libraries**
Complex data types	Composite or complex datatypes are either arrays or records. An array consists of numberOfElements elements that each have the same type, arrays have zero based indexing. A record describes a non- empty set of objects, each of which has a unique identifier.	Simulink wide signal for arrays. Simulink bus signal for records.
Characteristics	Values of characteristics can be changed on an ECU via calibration data management tool or an offline calibration tool	Overloaded Simulink.Parameter class suitable for online calibration.

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Liem Radita Tapaning Hesti**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Model Driven Development and Analysis for Embedded Automotive Software

   supervised by **Dr. Christian Saad and Dr. Kalmer Apinis**

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Tartu, 22.02.2019