SUPPORTING MULTIDISCIPLINARY ANALYSIS USING SYSTEM ARCHITECURES IN SYSML

A Thesis
Presented to
The Academic Faculty

By

Jaclyn Marie Branscomb

In Partial Fulfillment
of the Requirements for the Degree of
Master of Science in Mechanical Engineering

Georgia Institute of Technology

August 2012

SUPPORTING MULTIDISCIPLINARY ANALYSIS USING SYSTEM ARCHITECURES IN SYSML

Approved by:
Dr. Chris Paredis, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Bert Bras
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Judy Che
Electrification Research & Advanced Engineering
Ford Motor Company

Date Approved:
May 2, 2012

ACKNOWLEDGEMENTS

After having spent 5 years at Georgia Tech completing my undergraduate degree and my master's degree, my time at Tech is finally coming to a close. I have grown tremendously as an individual and an intellectual, and I couldn't have done so without the invaluable help and support of many different people.

First and foremost, I would like to thank my parents, Chuck and Tia, for their continued unconditional love and support. They have always been the first people that I turn to vent my frustrations when I have problems, and there have been many. They tell me to look at the big picture, take it one step at a time, and remember that I can accomplish anything to which I set my mind. I wholeheartedly thank them; I am not sure I would have survived Tech without them. I would also like to thank my loving fiancé, Ash, for all of his help and encouragement. Over the last 5 years, he has been a constant source of inspiration, and it means so much to me how much he believes in me. Finally, I must thank my grandfather, Charles, for his assistance in funding my studies at Georgia Tech. As a third generation master's degree in mechanical engineering, I know I have made him proud.

I have had many inspirational teachers over the years, but one who stands out most in my mind is Dr. Chris Paredis. He has helped me gain the confidence I need to be successful in the professional engineering world. I came into his research group knowing practically nothing about model-based systems engineering, and I feel that I have learned much from him in a very short time. He has such a deep knowledge in a vast number of subjects and that continuously impresses me.

I must also thank my lab mates of MARC 264, Alek Kerzhner, Ben Lee, Roxanne Moore, and Sebastian Herzig. Especially I must thank Alek Kerzhner for all of his help in teaching me Java and helping fix the bugs in my Java code. I learned a tremendous amount from Alek about how to program and debug, and I am very grateful. Due thanks also goes to Axel Reichwein, who was my mentor in the early stages of my research. I definitely look up to each of my lab mates for their perseverance through the Georgia Tech doctoral program.

I would also like to thank Mark Jennings and Judy Che from Ford Motor Company for their support of this research project. It truly has been a pleasure working with both of them, and I am very grateful for their patience. I hope I have been able to contribute to their overall systems engineering goals at Ford.

Finally, I acknowledge the support of the George W. Woodruff School of Mechanical Engineering and Ford Motor Company. Also, I appreciate the academic software licenses provided by No Magic Inc.

TABLE OF CONTENTS

LIST OF FIGURES

Appendix A

Appendix B

Appendix C

SUMMARY

To develop competitive vehicles with ever increasing complexity, automotive designers need to improve their ability to explore a broad range of system architectures efficiently and effectively. Whereas traditional vehicle systems are based upon internal combustion (IC) engines, today's environmentally conscious vehicle manufacturers must consider alternatives to the IC engine-only systems such as hybrid or electric systems. To help the engineers to model these multiple alternatives, it would be ideal to start from a base vehicle architecture. To design a good vehicle, it is necessary for each of these system architectures to be analyzed from a variety of attributes including performance, fuel economy, or even thermal behavior. Creating the necessary analysis models for each system architecture would be time-consuming, expensive, and could be error prone. To aid in overcoming such challenges, we have developed an approach for supporting the generation of subsystem model templates to support the integration of analysis models. The approach is based on formally modeling the system architecture in the Systems Modeling Language (OMG SysML^{TM}) and then using model transformations to generate stubs for corresponding analysis model templates in Modelica and Simulink. In this manner, we assist designers in managing large systems with multiple analyses, ensuring that the systems remain consistent, and enabling the reuse of generic architectures through specialization and redefinition. The starting point is a reference architecture, called the Vehicle Model Architecture or VMA, in which all the key subsystems and interactions between subsystems are formally modeled. In addition, we have created a generic template that is a specialized version of the VMA. This specialized template can then be adapted by the systems engineer to represent a specific vehicle program. In

addition, pre-defined, generic analysis templates can be redefined for the specific vehicle program under analysis. The SysML VMA system model is transformed through two model transformations, one that translates the physical portion of the system to Modelica, and one that transforms the logical controls portion of the system to Simulink. By automating these transformations and reusing a set of fixed templates for further specialized architectures, this approach helps to manage the complexity, and reduces the modeling time by enabling system model reuse. The entire approach taken in this thesis has been named the Vehicle Architecture Modeling Framework, VAMF, which includes the SysML VMA, the corresponding analysis templates, and the tools developed to support this approach. Throughout this thesis, the specific (fictitious) vehicle program "C100" and a 0-to-100 kph performance analysis test are used as examples for demonstration.

NOMENCLATURE

- **VAMF:** The Vehicle Architecture Modeling Framework is the name for the entire approach to vehicle modeling in this thesis. The VAMF includes a reusable reference architecture, templates for common vehicle analyses, tools to support the generation of analysis model templates, and a process flow for vehicle modeling.
- **VMA:** The Vehicle Model Architecture is the reference architecture that is modeled in SysML. The VMA models the vehicle's key subsystems and the interaction of energy and signal flows between subsystems.
- **MBSE:** Model-Based Systems Engineering is the approach to modeling complex systems through the use of models as opposed to traditional document-based approaches.
- **SysML:** The Systems Modeling Language is used in this thesis to formally model the VMA, specialized vehicles, and analysis architectures.
- **Modelica:** Modelica is a text-based, object oriented modeling language used to model the physical aspects of the VAMF
- **Dymola:** Dynamic Modeling Language, Dymola, is a tool that is based off of the analysis language Modelica
- **Simulink:** Simulink is the analysis language and tool used to model the controls aspect of the VAMF

CHAPTER 1:

INTRODUCTION

There are numerous challenges when designing and modeling complex vehicle systems. Some of these challenges include making many design decisions, designing with knowledge from different disciplines, and modeling the system architectures in a concise and easy to understand manner while representing the true complexities of the system [24]. Also, because alternatives to the conventional IC engine are being explored, it is desirable to develop a method to formally capture these alternatives in a reusable manner. Traditionally, a document based approach to systems engineering has been employed, which is inefficient, not reusable, and it creates a large potential for human modeling error [9]. This thesis presents the Vehicle Architecture Modeling Framework that encompasses the entire approach to support complex vehicle modeling. Included in the VAMF is a Vehicle Model Architecture in SysML to formally model the energy and signals flows and interactions between subsystems. Also included in the VAMF are analysis architecture templates for common analyses such as a 0 to 100 kph performance analysis, and tools to help system engineers generate the specialized analyses in order to validate different vehicle alternatives. Finally, a process flow for the VAMF has also been developed in order to formally guide engineers in the definition, analysis, and validation of vehicle models. Before getting into the details of the VAMF, this chapter provides justification for choosing SysML to develop the vehicle architecture, the shortcomings of SysML, how the VAMF supports Model-Based Systems Engineering, and proposes motivating questions to be answered in this thesis.

1.1 Using SysML in Support of MBSE

As a result of the issues above, model-based techniques are becoming more widely employed by designers. Model-Based Systems Engineering (MBSE) is one of these methods, and it allows engineers to encapsulate their knowledge of these complex systems in a set of models [6, 10]. This model-based approach differs from conventional engineering design approaches in that the models are continuously growing and changing whereas the traditional document-based approach tends to adhere to a "waterfall" design model beginning with system definition and ending with design qualification [9]. MBSE methods encompass a wide variety of ways to represent systems including representing the definition of the systems, the system requirements, and analyzing the different design solutions in order to validate the requirements. The Object Management Group (OMG) has created the System Modeling Language (SysML™) to support MBSE. SysML is a language based on the Unified Modeling Language (UML™), but with an emphasis on systems engineering applications. SysML is a general-purpose modeling tool that uses diagrams and clear, well-defined modeling constructs to characterize engineered systems [2]. Designers can implement SysML to represent and document complex systems in a manner that is understandable, and these systems can be revised and reused with relative ease.

1.2 Shortcomings of SysML

It is increasingly advantageous to have a method that formally captures the entirety of the engineering design problem using system architectures, and SysML makes

this possible. However, in addition to a purely static system architecture model, it is also necessary to have analysis models to evaluate and compare design alternatives.

There is one main weakness in using SysML as the modeling language for MBSE. MBSE centers on the design and specification of engineering systems and how they should be communicated to stakeholders. To actually analyze systems modeled in SysML, models must be manually translated into additional tools by the modeler. This process can be time consuming and costly, and the analysis models would need to be repeated for every slight change to the original system. Therefore, it is desirable to assist engineers in this transformation process. It is necessary to be able to compare, through analysis, a variety of system architectures, and completing this task manually would consume a lot of valuable modeling time and be error prone. However, there is a plug-in for MagicDraw called Paramagic that can analyze algebraic equations, but it is not suited for this work because it cannot handle differential algebraic equations [23]. Paramagic is discussed further in the Related Work of Chapter 2.

1.3 Modeling Vehicle Systems using MBSE

Currently, many vehicle manufacturers are not implementing MBSE practices in their vehicle models. It is common practice for different users to create portions of the vehicle system, and there may be no overall repository where these models can be stored. Each different modeling group, such as the Powerplant team or the Transmission team, must ensure that their models will interface with the rest of the vehicle system model and that they are using the most up to date models from each team. The problem of the vehicle modeling becomes less of a focus on the accuracy and results of the model and

more on checking whether the all of the models and data used in an analysis are consistent and current. When a new vehicle model is tested, the analyzers must first check to ensure the correct models were used from every subsystem before deciding if an anomaly is a true output of the new vehicle. Managing the consistency is paramount to the success of vehicle modeling to get accurate results. Also, if it is desired for a model to be translated into another modeling language, it must be done manually because there is little integration between modeling languages. All of these processes described above leave a potential for human modeling error, another problem with the current vehicle modeling practices.

1.4 Motivating Questions

As modern vehicles are increasing in complexity, it is necessary to find a method to model the entire vehicle system and perform analyses on that system to trade-off on attributes such as fuel economy or performance. The goal of this thesis is to create an overall specific vehicle template and support the generation of analysis model templates that can then be fully defined with appropriate domain models and analyze the specific vehicles.. This would save automotive companies a tremendous amount of cost and time to production, and by automatically supporting the generation of analysis model templates, this approach would also aid in reducing the potential for human error.

SysML has been chosen as the language to represent the Vehicle Model Architecture for its ability to capture complex systems. In order to advance SysML's capability to support MBSE design process, the following question should be answered.

Motivating Question:

How can engineers use SysML to model complex vehicle system architectures and the corresponding multidisciplinary analyses effectively, consistently, and accurately?

As stated above, the solutions of this question should support the MBSE design process in the following ways: modeling systems with multidisciplinary domains accurately, improving model consistency, and supporting the generation of analyses templates for model consistency verification. By incorporating the knowledge and information from vehicle systems engineering problems into SysML, engineers can ensure that the structure of various models remains consistent throughout the vehicle system using dependencies and associations. Through this traceability, engineers can create analysis templates that remain consistent with the structural vehicle model. This traceability also allows for the generation of analysis model templates to be automated, which saves modeling time and cost and reduces the potential for user error.

Because the motivating question is broad, it has been narrowed into three sub-questions which are given below. This thesis focuses on defining the Vehicle Model Architecture in SysML, which answers Question 1, creating a framework of analysis architectures, which answers Question 2, and generating analysis templates in Simulink and Modelica, which answers Question 3.

Question 1:

How can engineers effectively model vehicle architectures in SysML representing the energy and signal flows between all aspects of the vehicle?

The answer to this question is the groundwork for the Vehicle Model Architecture. If the entire vehicle system can be accurately represented by the current

SysML modeling constructs, then the model traceability and consistency can be authenticated.

However, purely representing the VMA in SysML is not the desired outcome. It is necessary for each vehicle to undergo certain analyses in the design process. These analyses must also be maintained and be compatible with the VMA structure. This results the formation of Question 2.

Question 2:

How can engineers create analysis architectures that can be reused for different vehicles while remaining consistent with the base vehicle architecture?

It would be ideal if these analysis architectures could be contained within SysML in order to preserve the model traceability and remain consistent with the rest of the VMA. The approach to solve this question is to create a framework and base set of generic, reusable analyses templates in SysML and associate them with the generic vehicle model architecture. Then, all of the necessary modeling structure and interference information is contained within the analysis architectures, and they can be translated to analysis languages, which leads to Question 3.

Question 3:

How can Vehicle Model Architecture be used to generate analysis templates in Simulink and Modelica?

Because SysML does not have the internal capability to simulate various analysis tests, the virtual vehicle models must somehow be validated. Creating analysis models manually in both Simulink and Modelica without the use of generated templates would be time consuming and error prone because there is no traceability between the

architecture and analysis models. Modelers would have to constantly ensure that model interfaces in each language remain consistent. This thesis explores automatic generation of analysis model templates based off of the Vehicle Model Architecture to support the integration between SysML, Simulink, and Modelica. Previous approaches to systems architecture modeling have used either Modelica or Simulink as analysis languages, which are discussed in more detail in Chapter 2. This approach however, suggests that a combination of languages is used in order to capture the best aspects of each language. Modelica is chosen for its abilities in modeling physical energy flows, and Simulink is picked for its prowess with logical control signals. This thesis presents the integration of the three languages, SysML, Modelica, and Simulink, in creating a logical Vehicle Model Architecture, specializing it for a specific vehicle, and then generating corresponding analyses to evaluate the specific vehicle.

1.5 Thesis Organization

The remaining chapters of this thesis are organized as follows: Chapter 2 gives a brief review of the relevant SysML modeling constructs and reviews of the analysis languages, Modelica and Simulink. Chapter 2 also includes a review of the recent research on the subject of multidimensional system architecture modeling in SysML and supplementing these architectures with analyses. Before delving into the Vehicle Architecture Modeling Framework, Chapter 3 gives some background as to why the approach in this thesis is an improvement from current vehicle modeling practices. The overall process flow of creating an entirely new vehicle project is also described in this chapter. Chapter 4 first defines the naming convention followed in this approach, and

then illustrates the approach of the Vehicle Architecture Modeling Framework by describing how the base architecture is structured along with the structure for the vehicle specialization template. The analysis architecture framework is also described in this chapter. To demonstrate the capability of the VAMF in validating a particular vehicle, Chapter 5 goes into detail how an engineer would create a specialized vehicle and run analyses on the vehicle. Finally, Chapter 6 gives the contributions of this work, along with a few limitations, and the desired future work.

CHAPTER 2:

RELATED WORK AND LANGUAGE REVIEW

2.1 An Introduction to SysML, Modelica and Simulink

Prior to discussing the details of using SysML to define the Vehicle Architecture Model and translating the SysML model to Modelica and Simulink templates to support building models for analyses, first a review of the relevant SysML constructs that will be used in the Vehicle Model Architecture is provided in this section. Also the modeling constructs for Modelica and Simulink will be briefly described in this section.

2.1.1 Relevant SysML Constructs

In SysML the basic unit of modeling is the *block*, and it can be employed to describe a type of a system, components, entities that flow through a system etc. as described in Chapter 7 of the SysML specification [11]. In the example problem used throughout this thesis, blocks represent anything from a vehicle's subsystem, such as the Powerplant, to its physical and logical parts, the Powerplant Plant and Powerplant Controller, respectively, to the type of energy that flows between two subsystems, such as MechRotatE (mechanical rotational energy). In this thesis, the block represents the basic building unit for the characterization and breakdown of a system into its logical and physical components.

A Block Definition Diagram (BDD) is used to declare blocks, their characteristics, and the structural connection with other blocks. Several BDD's can be used to define one system, and it is not required to represent all of a system's

relationships in a single diagram. For each application, it is acceptable to only give a certain level of detail needed for that specific application. For example, Figure 1 depicts a BDD for the Vehicle Domain. Within this BDD, the Vehicle Domain is portrayed as the top level of abstraction. Contained as a component within the Vehicle Domain is the Environment, Driver, and the Vehicle itself, and the contained within the Vehicle block are all of the Vehicle's subsystems. Figure 1 only shows down to a certain level of abstraction; each subsystem also contains ports and parts, but these are represented on separate diagrams.

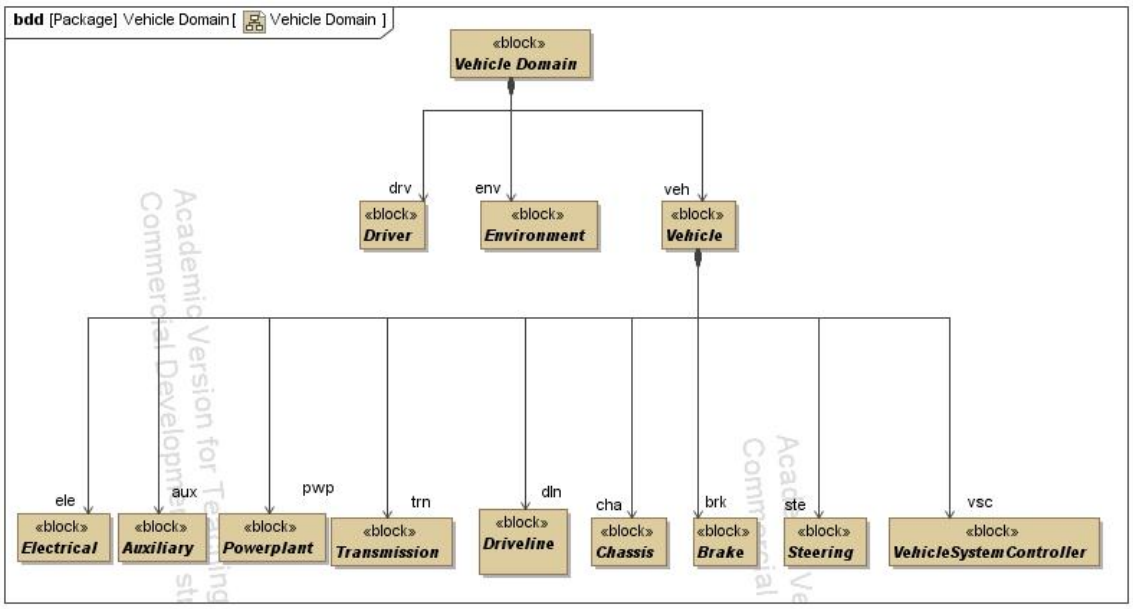


Figure 1. BDD of the Vehicle Domain

An Internal Block Diagram (IBD) is used to represent the connections between parts of a block. The frame of the diagram represents the block in which contains the content represented in the IBD. Ports on a block can be connected to ports on the frame of the diagram or to other blocks within the IBD using connectors. In this example, the ports are typed by either energy flows or control signals. For example, Figure 2 shows an IBD of the Powerplant subsystem, where the Powerplant block is the frame of the

diagram. The Powerplant's two parts, the Powerplant Plant and Powerplant Controller are connected with two ports, a sensor signal and an actuator signal. The Powerplant Plant is connected to the vehicle's other subsystem Plant components for the Driver, the Auxiliary, Transmission, and Electrical Subsystems. The Powerplant Controller receives signals from the Global Bus through the "ctrlGloblBusMs" port, and sends signals to the Global Bus through the "pwpGloblBusMs" port.

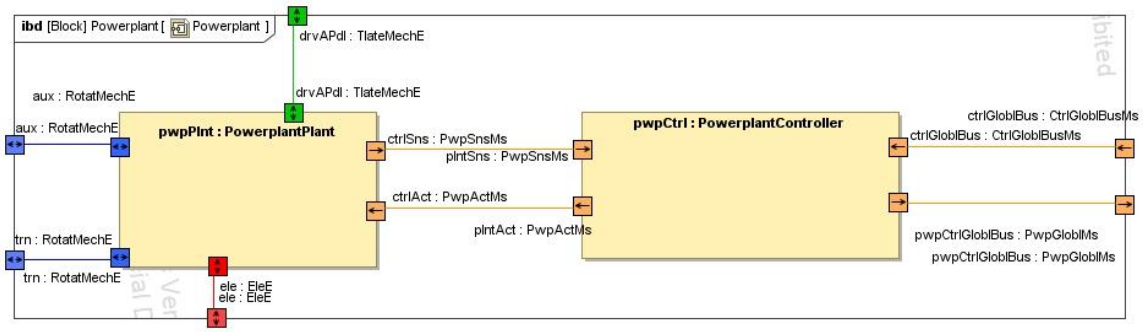


Figure 2. IBD representing the internals of the Powerplant block

2.1.2 Modelica/Dymola Review

In this thesis, the Modelica language is implemented as the analysis language for the physical aspects of the system. We use the Dynamic Modeling Language, Dymola, as the modeling environment for Modelica. The structure of Dymola is similar to SysML in that they both are object-oriented, using predefined models in diagrams, except that Modelica is text-based, based upon the Modelica language. Modelers can either begin with one of the predefined objects and add more functionality to it by modifying the Modelica code, or they can start with a blank model and create their own object.

Dymola has several multidisciplinary pre-made block libraries that modelers can use or modify. For example, Dymola has a standard library, which contains common logical,

continuous, discrete, and interfaces blocks. Dymola also has other libraries for electrical, mechanical, translational, fluid, and thermal systems. Figure 3 shows how the standard libraries appear in Dymola.

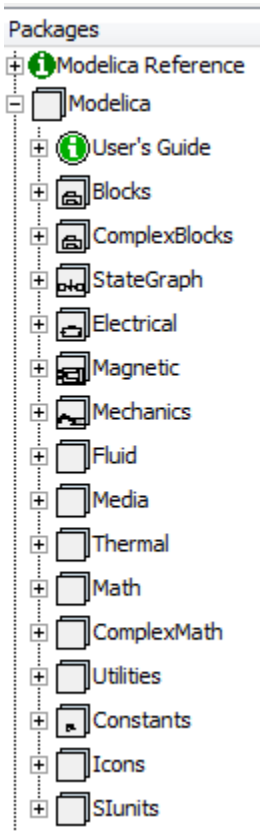


Figure 3. Dymola Block Libraries

In order to interface SysML and Dymola, a SysML profile is defined that stereotypes SysML constructs to Modelica constructs. This profile is the *SysML4Modelica* profile [19], and it supports the generation of Modelica models from SysML. In this thesis, the physical portion of the SysML model that defines the energy flows between subsystems is translated to a Modelica template with input and output interfaces defined. This template is given to the domain model developer to populate with appropriate model content with desired physics. Then, the Modelica model is imported into the Simulink environment as an S-Function for analysis.

2.1.3 Simulink Review

Simulink, produced by MathWorks, is a multidisciplinary simulation environment that also supports Model-Based Systems Engineering [14]. Simulink is widely used in many different industries for its ability to handle complex control systems and time-varying signal processing. An example of Simulink's capabilities can be seen in the example of a combat aircraft flight control system in Figure 4 [3]. Simulink is also currently a staple of the automotive industry because it can be used to map, test, and design the control systems that can be directly translated to actual vehicle controllers.

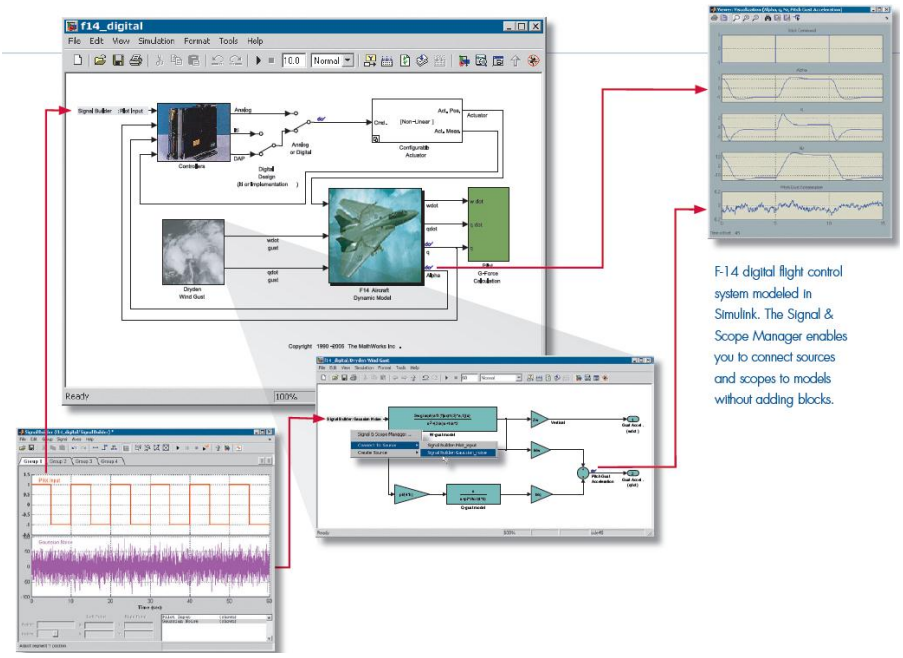


Figure 4. Example of Simulink's modeling capabilities [3]

Simulink also has a hierarchical structure similar to SysML and Dymola. Modelers in Simulink can use predefined blocks from the Simulink library, shown in Figure 5 to create models for analysis.

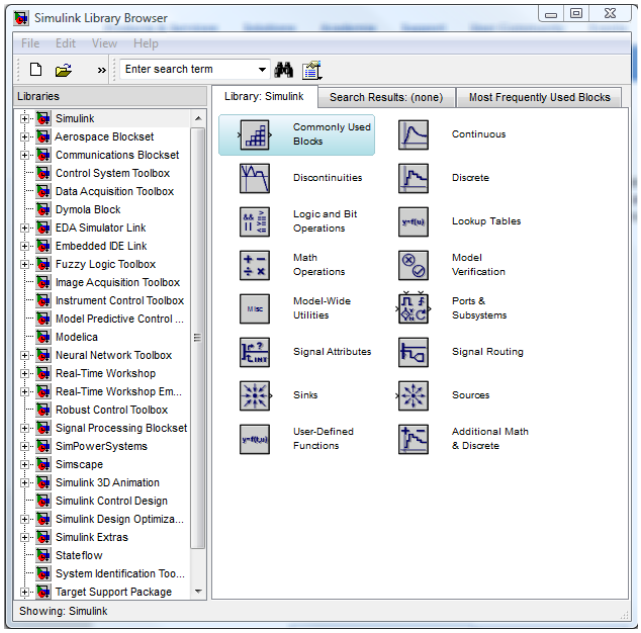


Figure 5. Simulink predefined block library

In this work, Simulink is the program that is the base simulation environment for the entire system. The control portion of the overall system is imported into Simulink, and then the physical portion of the system can been imported from Dymola into Simulink. The physical and control halves of the vehicle system are connected then connected in Simulink by connecting the corresponding input and output ports. Finally, the analysis model is complete and the system is ready to be tested and analyzed.

2.1.4 Justifications for Selecting these Languages and Tools

SysML, Modelica, and Simulink are all languages commonly used in systems engineering, but they are not currently being integrated to combine the best aspects from each language. No Magic's MagicDraw with the SysML plug-in is selected to model the overall vehicle system problem for several reasons. MagicDraw is currently the tool that best supports SysML through its complete integration with the SysML plug-in. Also, No

Magic publishes their application programming interface (API), which gives the ability to increase the functionality of MagicDraw by creating additional plug-ins to customize the language.

The Modelica language has the capability to model complex systems over a variety of environments from mechanical, to electrical, thermal, and hydraulic. Modelica implements acausal modeling, which allows models to be evaluated simultaneously at each time step [18]. Relations in the models act as constraints between variables and the models interact through these constraints. Through these acausal models, symbolic manipulation can be used to achieve very efficient simulations. The prowess of Modelica is in modeling physical interactions between components. Dymola is chosen as the tool to model and simulate the Modelica models generated from SysML because of its predefined libraries and ability to interface with Simulink.

Finally, Simulink is used to model the controls interactions within the Vehicle Architecture Modeling Framework. Currently, Simulink is a standard for modeling controllers in the automotive industry. It has very well supported code-generation capabilities so that production code can be generated from control-block-flow diagrams, eliminating the opportunity for software implementation errors, and reducing the time needed for software implementation tremendously. Simulink excels at modeling logical flows between the controllers of the vehicle system, and it also has the capability to import a Dymola model into Simulink to integrate these two programs. By using SysML for the overall system architecture model, Modelica to model the physical energy flows, and Simulink to model the control and signal flows, this approach implements the best

aspects of each language and integrates them to support efficiently generating templates used for creating analysis models.

Through the integration of these three languages, a base generic architecture, and analysis templates, the VAMF allows the use and reuse of the vehicle architecture models to support quantitative analysis and design. By incorporating Simulink and Modelica with SysML, different analysis tests can be defined for any system. Both Simulink and Modelica can be used separately from SysML to model systems in their own environment, but these analytical system models are not linked back to an overall system architecture in SysML. However with the VAMF, the three system models that are created, the overarching system architecture model in SysML, the structures of the physical portion of the system in Modelica, and the control side of the system in Simulink can be traceable and consistency can be maintained. Without this approach, all three models would have to be continuously manually updated every time a change is made to one of the models. Other than the tedious nature of this task, it also introduces the possibility for human modeling error. The motivation of this research is to be able to formally model multiple system architectures, support creation of models for multiple analyses on the system, and automatically cascade changes to the architecture models to keep the analysis model templates consistent. This thesis describes how to formally model complex system architectures and how the generation of analysis model templates is supported.

2.2 Related Work

As system engineering problems continue to grow in complexity, more researchers are recognizing the benefits of Model-Based Systems Engineering and integrating system architectures in SysML with analysis programs. Managing complex system architectures in a formal way is necessary, but very challenging, as discussed in [16], which described the model management effort for combat submarines. [16] discusses the difficulty in managing several product configurations simultaneously, which is also true for the automotive industry. In this section, a synopsis of several approaches to combine the formal model architectures created in SysML with analysis languages is presented. Also, additional background in the area of MBSE is discussed.

First, there is a plug-in for MagicDraw called Paramagic that can solve equations within SysML that are linked to a SysML model [20]. Parametric modeling in SysML implements constructs that are not defined in UML such as constraint blocks, parameters, and value properties. This plug-in gives the ability for a SysML model to be analyzed within SysML, however it is constrained to analyzing only algebraic equations. Therefore Paramagic is not suitable for this research because it cannot analyze differential algebraic equations.

In [15], McGinnis and Ustun integrate SysML with the simulation language Arena [4], which is a discrete event simulation software by Rockwell Automation. They have created a translator from SysML to Arena by using MS Access, which can easily interface with Arena. In order to accomplish the translation, McGinnis and Ustun developed an Access domain specific language (DSL) in SysML because both SysML and MS Access can import and export XMI files. However, this translation process was

only completed on a limited example, so there is uncertainty whether this approach can scale up to larger more complex systems.

One such approach for implementing the Modelica analysis language with SysML formal model architectures has been discussed by Johnson et al. [12]. A major shortcoming of SysML is its inability to explicitly analyze the complex system models created with the SysML language. This approach recognizes the value in creating a formal architecture in SysML and exporting the model to Modelica for analysis. To integrate these two languages, triple graph grammar (TGG) transformations are used. TGG's are used to map SysML constructs to their corresponding Modelica constructs through the use of a correspondence relationship. Through these relationships, tools can be used to automatically generate code that executes the language transformation. Figure 6 shows a high-level overview of how the models are transformed from SysML to Modelica. While this approach gives analysis capability to complex system architectures modeled in SysML, it does not include the ability to use a combination of Simulink and Modelica in order to better model the logical control signals and the physical plant signals.

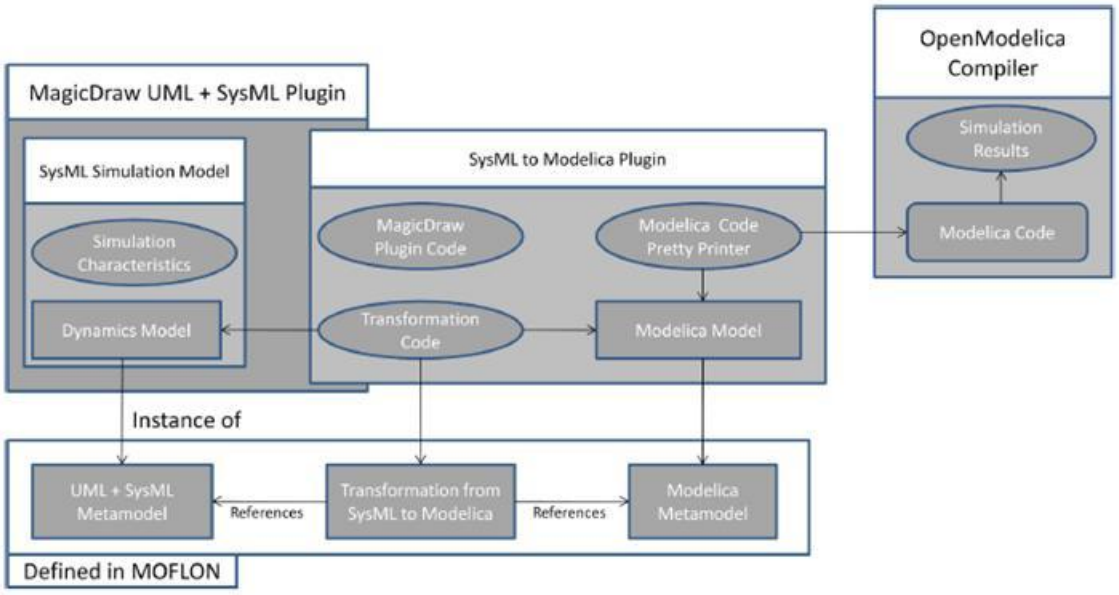


Figure 6. The SysML to Modelica transformation framework [12]

Another approach was completed by Fritzson and Pop [21], who created a ModelicaML profile for UML/SysML in order to aid in modeling and simulation of continuous dynamic models. The ModelicaML profile is intended to allow users to represent a Modelica simulation model along with the UML/SysML informational models. This profile reuses some UML and SysML constructs, but it also establishes an entire new set of language constructs. Examples of these new constructs are the Modelica class diagram, the simulation diagram, and the equation diagram. The weakness with this approach is that instead of modifying the existing UML/SysML modeling constructs, many new potentially unnecessary constructs are added.

Similar to the previous approach, Nytsch-Geusen [17] also created an additional profile named UML^H, which is a specialized version of UML. In this work, it is used to develop model-based hybrid systems in Modelica. The Modelica models utilized in this approach are based on differential algebraic equations (DAEs) and the Modelica state chart extension. Then a UML^H editor and a Modelica tool that assists in code generation

can be implemented to automatically create stubs generated from the UML^{H} diagrams. The user then only needs to insert the equation-based behavior of the system into the Modelica model stubs. This approach also implements TGG style model transformations in order to translate between SysML and Modelica.

Where the last three approaches focused on the SysML to Modelica transformation, there has also been research on integrating SysML with Simulink. Qamar, During, and Wikander have implemented a translation from Simulink to SysML through the use of a plug-in that parses an *mdl* file containing the Simulink model and transforms it to an *Eclipse* UML2 framework according to a mapping design [22]. The model constructs from the Simulink model are stored for the transformation to the UML2 framework, which is recognized by SysML. In this manner, Simulink models can be generated in SysML. This approach allows for a better connection between a functional system and a logical architecture, but it begins with a functional Simulink model and translates that to a SysML architecture. In this thesis, the opposite approach is used. We suggest that the system architecture framework should be created first, which would allow engineers to completely define the systems engineering problem before creating analysis models.

The approach taken in this thesis for the translation from SysML to Modelica centers on the *SysML4Modelica* profile, which is one step in defining the bidirectional SysML to Modelica translation given in [19]. The goal of [19] is to better integrate Modelica and SysML through the *SysML4Modelica* profile, which is an addition to SysML that represents the most common Modelica language constructs. To implement this profile, modelers first create a SysML model, and then decide which elements need

to be analyzed in Modelica, and stereotype the SysML model with «Modelica*Contructs*» to be translated to Modelica for analysis.

CHAPTER 3:

VEHICLE MODELING PROCESS

3.1 Overview of Vehicle Modeling with MBSE

As vehicle systems are continuously increasing in complexity, it is essential that there is a process flow to handle the modeling of the vehicle system from beginning to end. Ford Motor Company has created a process flow of how they envision virtually creating this vehicle system, given in Figure 7 [7].

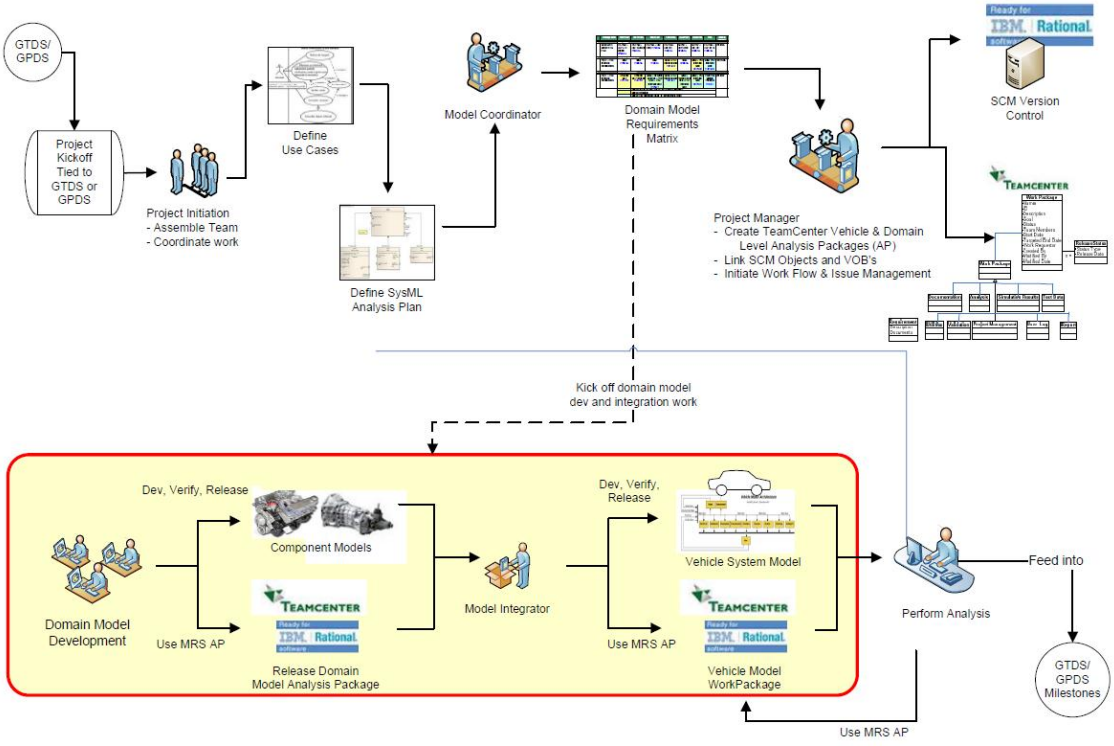


Figure 7. Overall vehicle modeling process flow

Figure 7 shows how a project would begin by the Global Technical Development System (GTDS) or the Global Program Development System (GPDS) deciding a new vehicle system modeling work stream should be kicked-off. Then a team would be

assembled, the use cases of the new vehicle models would be defined, and the SysML analysis plan for the new vehicle would be created. This step leads to creating Model Coordination Matrix (MCM), which is a table that shows the desired level of fidelity required for each subsystem and which models need to be coordinated. From the MCM, the process flows into the red box, which defines the scope of the VAMF. Inside this box is where the specific SysML vehicle model templates are created and the corresponding analyses are generated and validated. The main work in this thesis supports the generation of the elements contained within the red box. Figure 8 more clearly illustrates the details contained within the red box of Figure 7 [8].

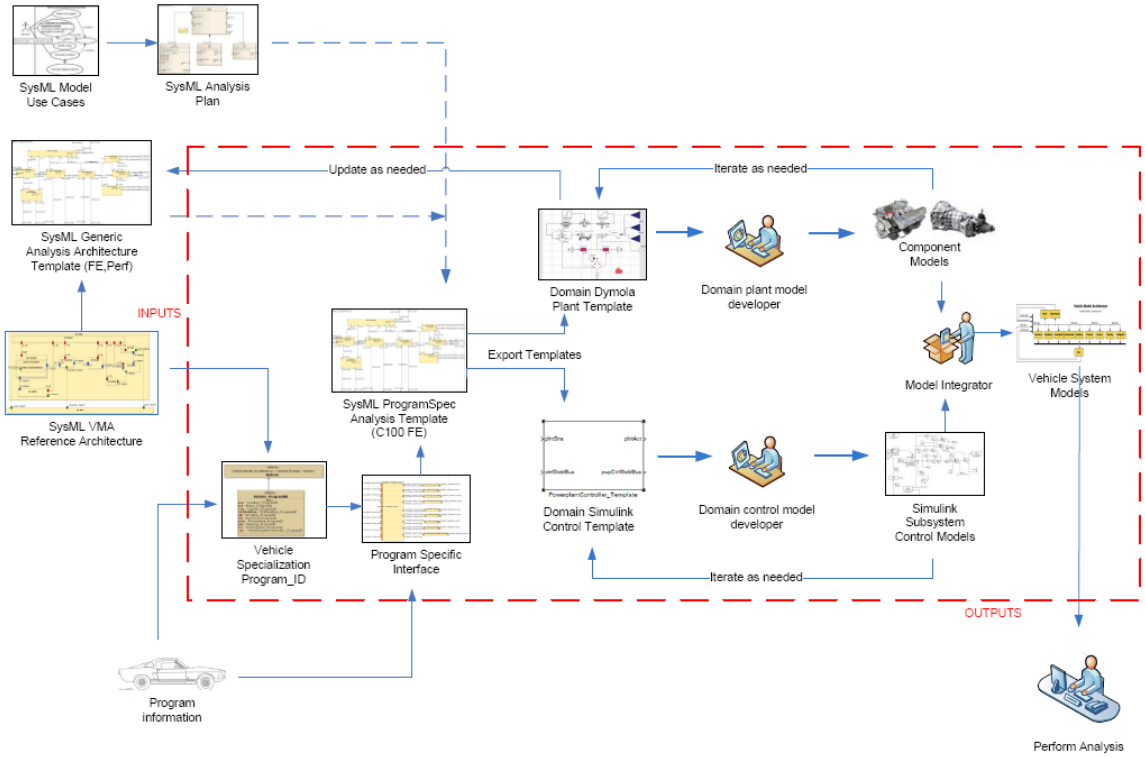


Figure 8. Domain and vehicle model development

In Figure 8, a simplified work flow that modelers would follow to create a specialized vehicle system model for a given scenario is illustrated. The goal of the

SysML model is to create a template for a specific vehicle for a specialized analysis. To achieve this model, several inputs are needed. The first input is the SysML analysis plan, which is fed by the SysML use cases. Another input is the generic VMA reference architecture template and the SysML generic analysis template, described in Chapter 4. The final input is the vehicle program specific information for a given vehicle. All of these inputs are required to create a formal specialized vehicle model and specialized analysis templates. From the program specific, specialized SysML vehicle model, transformations can be performed to translate the specialized vehicle model architecture to the templates in the analysis languages Modelica and Simulink. Once these templates have been translated to Modelica and Simulink, they are handed off to domain experts. To fully define the analysis models, the domain experts need to supplement the analysis templates with appropriate Modelica plant and Simulink controller models. The Modelica model is then integrated with Simulink, and the entire vehicle system is analyzed.

An even more detailed view of the process flow is given in Figure 9. This figure shows a SysML Activity diagram with different swim lanes that define which actions should be performed by which people. The actions in the first swim lane are performed by the Program Systems Analysis Coordinator, and this person is responsible for creating the SysML use cases and analysis plans. Then he should also "check out" the generic analysis architecture and determine whether a change is required. If so, then he should initiate the change process on the generic architecture, and if not, then he should hand off the analysis plan and architecture to the SysML Program Architect. This person should take the specialized vehicle, analysis plan, analysis architecture, and the program specific interface as inputs to the program specific analysis template. From the specific analysis

template, the specific analysis and architecture are inputs into the exportation of the specific Modelica and Simulink template models. The SysML Program Architect sends these outputs to the Domain Model Developer. This person then takes the model templates and completes them with appropriate Modelica and Simulink models. He also calibrates and validates these models. If changes are necessary, he determines whether the change is needed at the domain level or the generic architecture level. Finally, he outputs the validated relevant subsystem analysis models to the Model Integrator. This person integrates the Modelica and Simulink models together in Simulink, calibrates and validates the entire vehicle model. He also decides whether changes are needed, and if so, at which level, domain or generic architecture. Finally, he outputs the complete validated vehicle analysis to the Simulation Analyst, who performs the final vehicle analysis. The entire work flow process described above guides system engineers in implementing the VAMF.

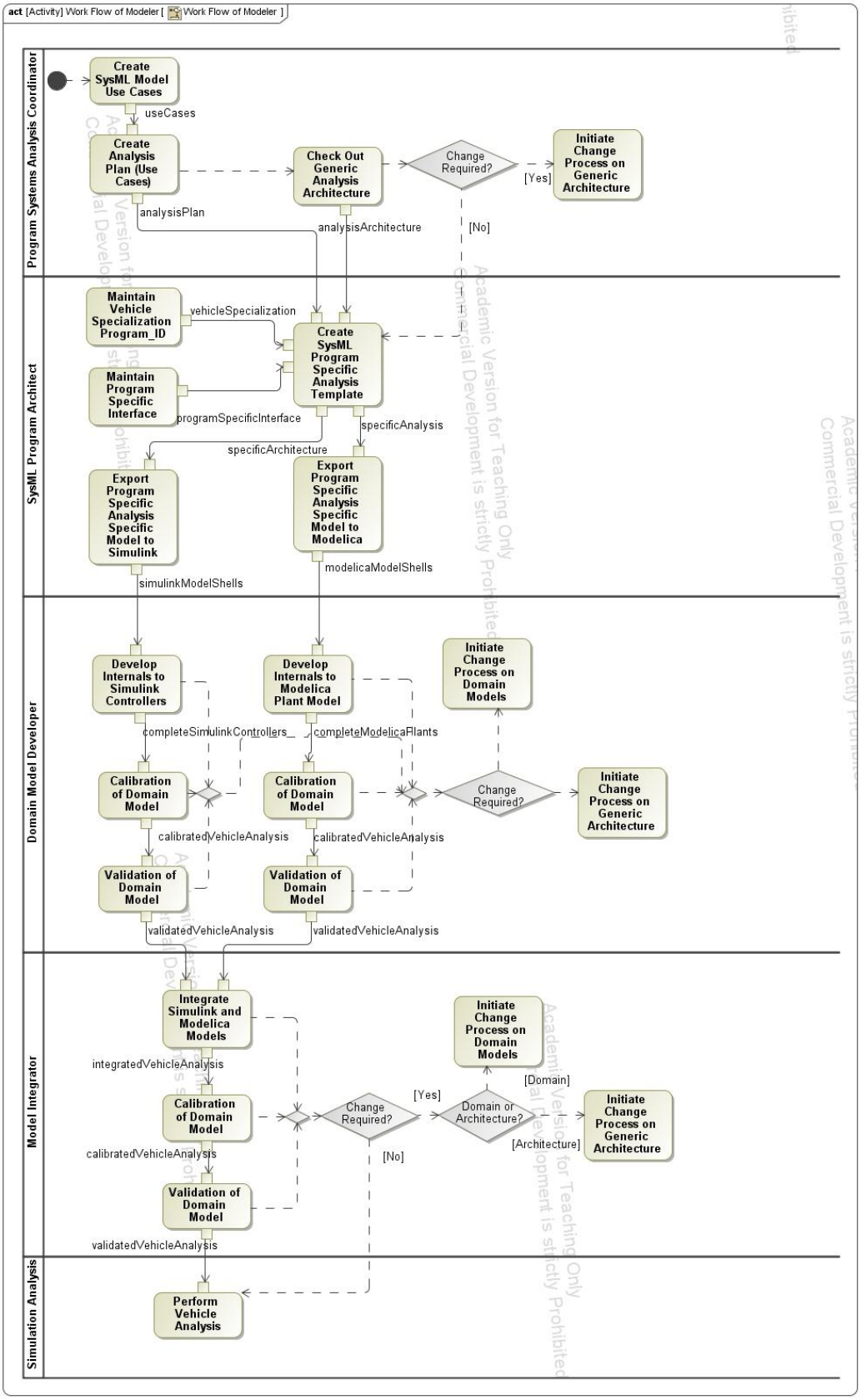


Figure 9. Activity diagram showing the process flow of modelers

As described in Chapter 1, the many current practices of vehicle modeling are archaic. There may be no central repository or central hard drive for storing models from different engineers and no unifying overall vehicle systems model. Therefore, individual subsystem groups, such as the Powerplant team or the Transmission team, must ensure that their models remain consistent in terms of the correct interfaces, version, and compatibility. Every time a change is made to one model, it needs to be manually updated to every necessary model. The problem becomes more centered on consistency rather than accuracy of the vehicle analysis outputs. If there is an anomaly in the output, the analyzers much first check to ensure that the correct versions of each subsystem are being used before considering other possibilities. Therefore, it is the goal of this thesis to unify that process, ensure model consistency, and cascade changes to the models automatically.

3.2 Approach to Achieving a Complete Vehicle Model with Analysis

Figure 10 takes the overview of the process described above and depicts the model perspective of the framework of how the vehicle system architecture and analysis templates are actually modeled in SysML. The top level of Figure 10 shows the generic aspects of the VAMF, and the bottom level depicts how the generic models are specialized for a specific vehicle. As can be seen on the left of Figure 10, the generic vehicle architecture is specialized for a specific vehicle, and then global and local vehicle signals are imported into the specialized model. Then, through an association relationship, the generic vehicle architecture corresponds to the analysis template of the total vehicle. Multiple analysis templates are created that depend upon the necessary

subsystems and ports of the total vehicle analysis template. From the analysis templates, a specialized subset of analyses can be generated corresponding to the desired analyses for a specific vehicle. Finally on the right of Figure 10, the tool specific models show how generic analysis models can be expressed mathematically, not in any specific tool or language. However, for these mathematical models to be solvable, they need to be implemented in specific tools and languages, as designated by the Modelica and Simulink model blocks. Then the specialized vehicle model, along with the specialized analyses can be automatically translated into Modelica and Simulink model templates

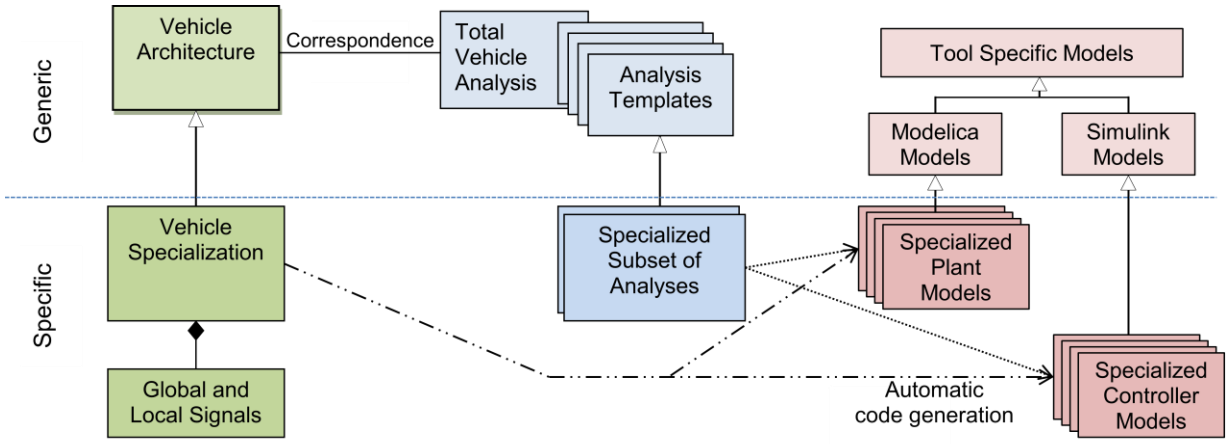


Figure 10. Framework of vehicle modeling approach

CHAPTER 4:

THE VEHICLE ARCHITECTURE MODELING FRAMEWORK

4.1 Approach for the VAMF

In this chapter, an approach is illustrated for representing the Vehicle Architecture Modeling Framework using SysML modeling constructs and generating corresponding analysis model templates. More specifically, the approach describes creating a generic structure that can be reused to create specific architectures, which then can be the basis for creating component models that can be analyzed for specific analysis tests. First, the naming conventions used in the VAMF are described, then the generic architecture and how it can be specialized for a specific vehicle is outlined, and finally the analysis architectures are defined.

4.1.1 Naming Conventions used throughout the VAMF

When creating the Vehicle Architecture Modeling Framework, strict naming conventions and abbreviations are followed. As with standard modeling convention defined in [11], block names and types are always capitalized. Ports, parts, and anything contained in a block are in camel case starting with a lowercased first letter, for example, the "Vehicle" block is capitalized and it has a part "pwp" with the type of "Powerplant".

For choosing the names of the subsystems, their abbreviations, and other system details, the VMA [5], and the Ford New Product Architecture Abbreviations list are employed [1]. The Vehicle Model Architecture has nine subsystems: Electrical, Auxiliary, Powerplant, Transmission, Driveline, Chassis, Brake, Steering, and the

Vehicle System Controller. Each of these subsystems has a three character abbreviation: ele, aux, pwp, trn, dln, cha, brk, ste, vsc, respectively. These abbreviations are used in a variety of places, such as in front of a signal corresponding to a given subsystem, e.g. "pwpCtrlGloblBus" is the Control Global Bus signal coming from the Powerplant subsystem. Also, these abbreviations are used for port names. For example, there is a rotational mechanical energy port on the Powerplant subsystem, which is named "trn" signifying that it connects to the Transmission subsystem. Similarly, there is an analogous port on the Transmission subsystem that is named "pwp." It is necessary to adhere to a strict naming convention in order to maintain the model consistency throughout the VAMF

4.1.2 Generic Vehicle Model Architecture

This section describes how the generic VMA has been created and structured in SysML. The clearest way to illustrate the contents of the VMA is to describe the contents of the generic VMA package, as seen in Figure 11. The top level package is called the Vehicle Model Architecture. This model contains three sub-packages for the Vehicle Domain, Units and Flows, and Documentation. The Vehicle Domain contains the bulk of the Vehicle Model Architecture. The Vehicle Domain is a generic representation of all of the subsystems and parts of the Vehicle Model Architecture [5]. To show that this model is generic and needs to be specialized before further definition, all of the blocks inside the Vehicle Domain are abstract, represented by the block names being italicized. The Vehicle Domain BDD shows an overview of the components of the Vehicle Domain, as shown by Figure 12.

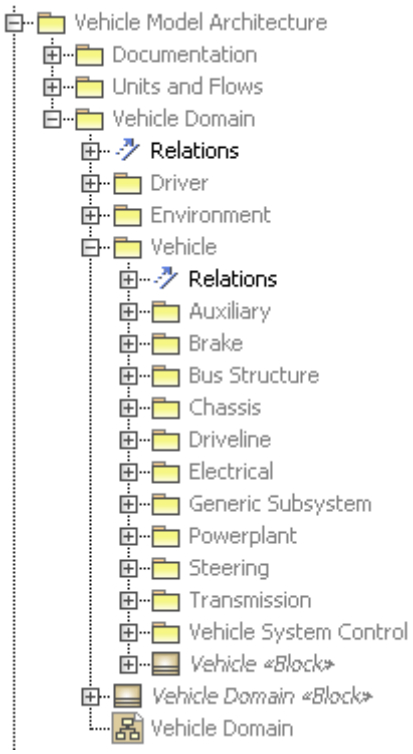


Figure 11. Package structure for the VMA

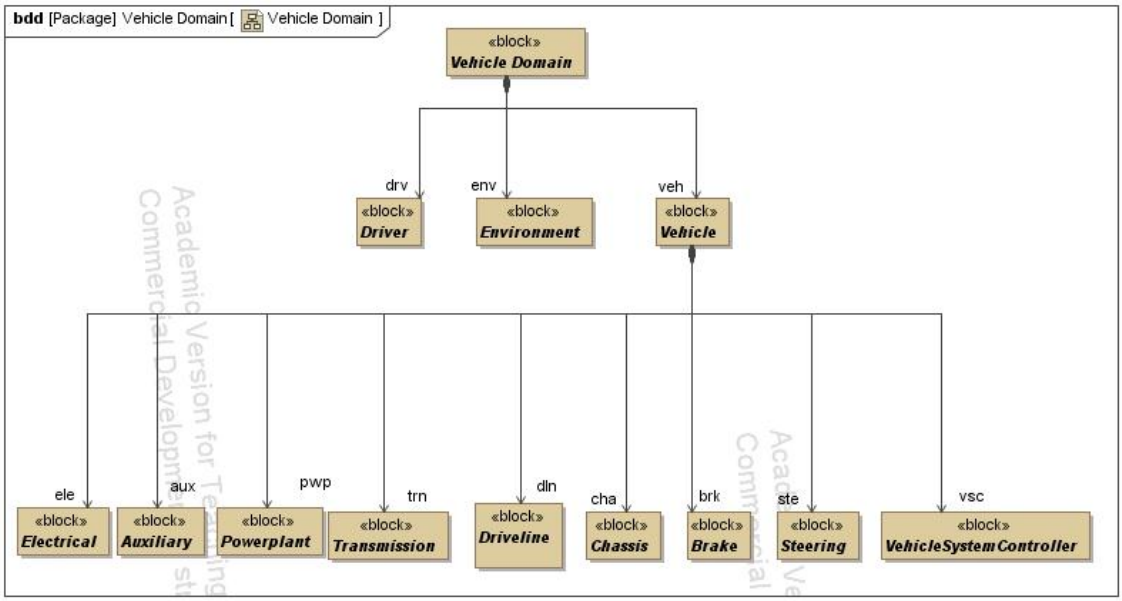


Figure 12. Vehicle Domain BDD showing components of the Vehicle Domain

Inside the Vehicle Domain package, the Vehicle Domain is broken up into the Driver, Environment, and Vehicle sub-packages. The Driver package contains the Driver block, and this block contains the flow ports for all of the driver's interactions with the vehicle. The Environment package contains the Environment block, but for simplicity, the environment is not analyzed in this work.

Because all of the Vehicle's subsystems have the same structure, a Generic Subsystem package, contained in the Vehicle package, defines the basic structure for a subsystem. A subsystem is divided into a logical controller, which represents all of the non-physical components of a subsystem, and a physical plant, which represents the physical aspects of a subsystem. To define these relationships, an abstract Subsystem is created that only contains two parts, "plnt" and "ctrl," which are of type Plant and Controller and each have a multiplicity of 1. Then, for each subsystem of the vehicle, e.g. *Electrical* or *Powerplant*, generalization relationships are made from the generic *Subsystem* block. On each specialized subsystem block, two parts are added, which are each subsystem's specific controller and plant components. For example, the generic *Powerplant* block, specialized from the generic *Subsystem* block, contains a *PowerplantController* and a *PowerplantPlant*. The usages of the each subsystem's plant and controller pieces are the three letter abbreviation for the subsystem, followed by either "Plnt" for plant or "Ctrl" for controller. For example, the "pwpCtrl" and "pwpPlnt," are the usages of the *PowerplantPlant* and *PowerplantController* as parts of the Powerplant subsystem. The usages are redefined and specialized from the generic *Plant* and *Controller* blocks of the generic *Subsystem*. The Figure 13 and Figure 14 show

this specialization and redefinition relationships for each of the subsystems' plant and controller pieces.

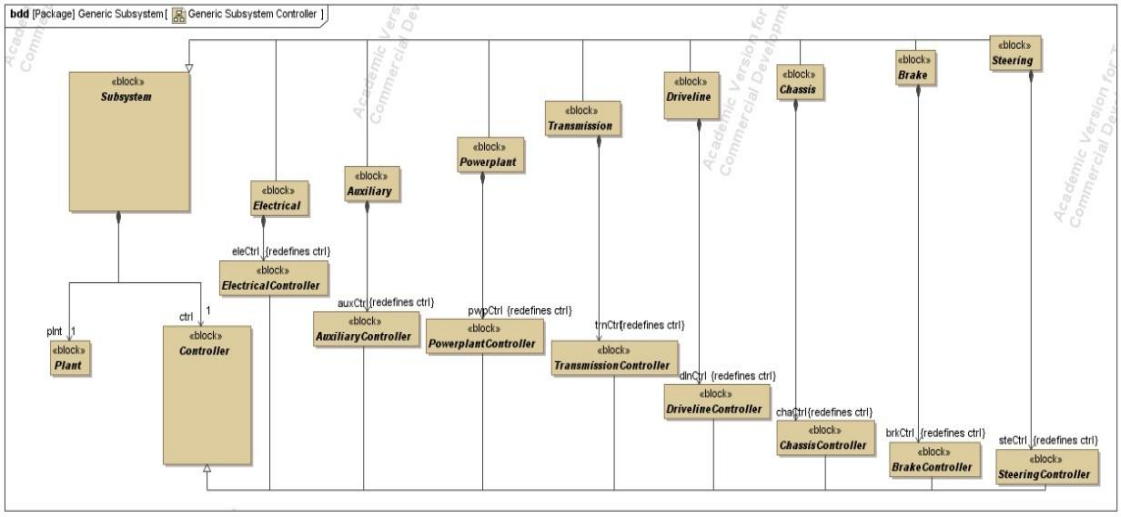


Figure 13. Generic Subsystem definition: Controller view

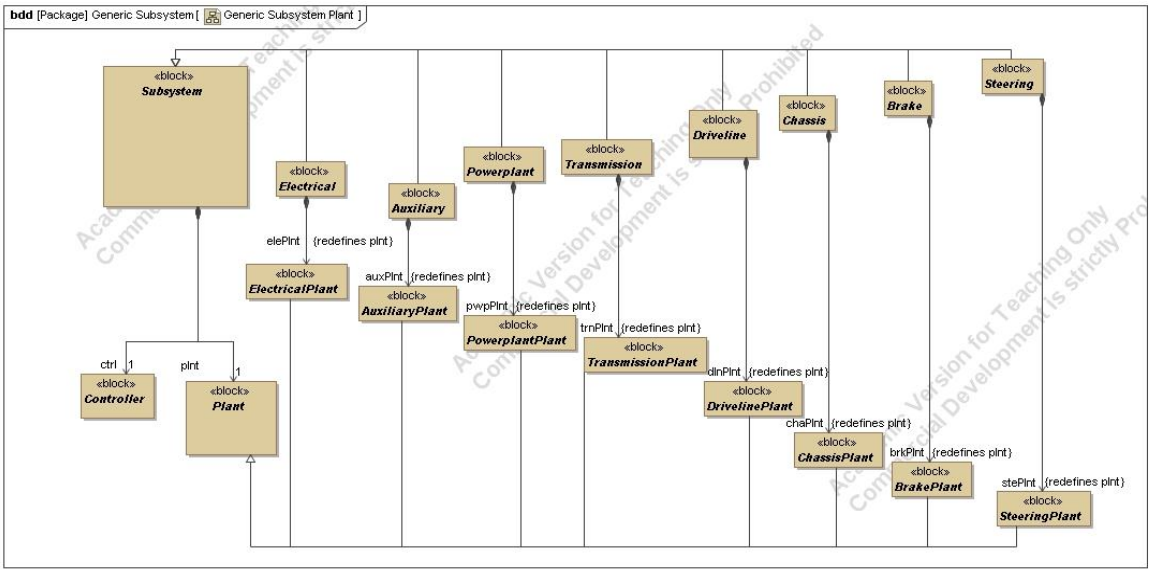


Figure 14. Generic Subsystem definition: Plant View

The Vehicle package contains all of the subsystems of the Vehicle in their respective packages. The nine subsystem packages are Auxiliary, Electrical, Powerplant,

Transmission, Driveline, Chassis, Brake, Steering, and the Vehicle System Controller. The generic subsystem block contains two parts, its respective plant and controller, as shown in Figure 14 above. Each of these subsystem packages contains an abstract subsystem block, e.g. *Powerplant*, a plant block, and a controller block. The generic subsystem block also contains flow ports, which connect the subsystem to the rest of the generic vehicle. An example of a port on the Powerplant subsystem is the "trn: RotatMechE" port, which is the Powerplant's hardware connection with the transmission. The RotatMechE is the type of the port, which in this case it is rotational mechanical energy. There is also a corresponding port on the generic Transmission block called "pwp: RotatMechE."

While the plant portion of the subsystem contains all of the ports that represent energy flows or hardware connections of the vehicle, the controller portion maps the local signals between subsystems and the global signals between each subsystem and the Control Global Bus. There is one global input signal to a subsystem's controller block, which represents all of the signals sent from the Control Global Bus. The usage of the signal is named "ctrlGloblBus" with its type named "CtrlGloblBusMs," where Ms stands for Mixed Signal. There is also one global output signal that represents all of the signals sent from a given subsystem to the Control Global Bus. This signal is named "pwpCtrlGloblBus" with its type named "PwpGloblMs," for example.

In addition to the global signals described above, there are also local signal ports between each plant and controller part of a subsystem, which are logical mixed signal types. The signal port coming from the plant part going to the controller part represents a sensor signal. The signal port coming from the controller part going to the plant part

represents the actuator signal. Examples of these two signal port types are *PwpSnsMs* and *PwpActMs* for the Powerplant sensor and actuator signals, respectively.

Each subsystem block also has an internal block diagram (IBD) associated with it. The IBD shows the subsystem's two parts: the plant and controller. All of the physical ports are shown on the border of the plant part, along with the two ports representing the connection to the subsystem's controller part. The logical ports are shown on the controller part, along with the corresponding ports to connect the plant and controller parts. The ports that appear on the border of the diagram are ports on the subsystem itself, which is represented by the boundary of the diagram. An example of a subsystem IBD is given in Figure 15.

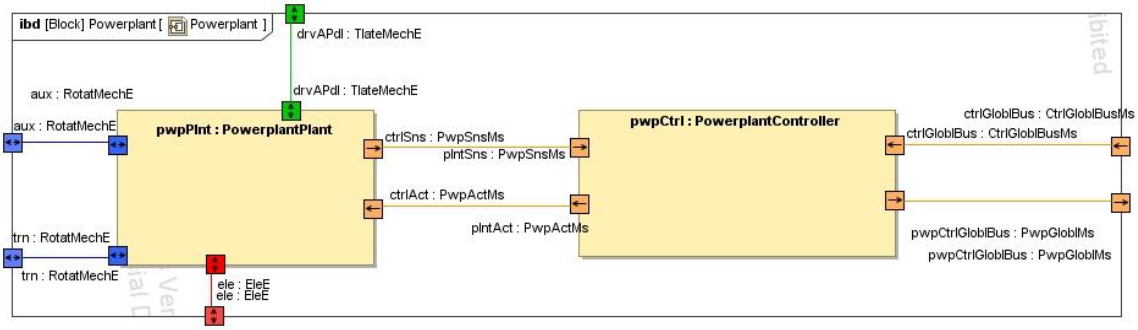


Figure 15. Example of a generic subsystem IBD

In addition to the Vehicle's subsystem packages, there is a package that contains the elements relating to the Global Bus Structure. Inside this package, there is a block named "CtrlGloblBus", or the Control Global Bus, which is a generic bus block that receives each of the subsystem mixed signals and concatenates these signals to flow through one output port, named "CtrlGloblBusMs." The "CtrlGloblBusMs" port has sub-ports, which connect to each of the subsystems. For example, the "pwpCtrlGloblBus," which is of type "PwpGloblMs," is the sub-port on the "CtrlGloblBusMs" that

encompasses all of the global signal ports relevant to the Powerplant subsystem. Figure 16 shows the CtrlGloblBus with its ports.

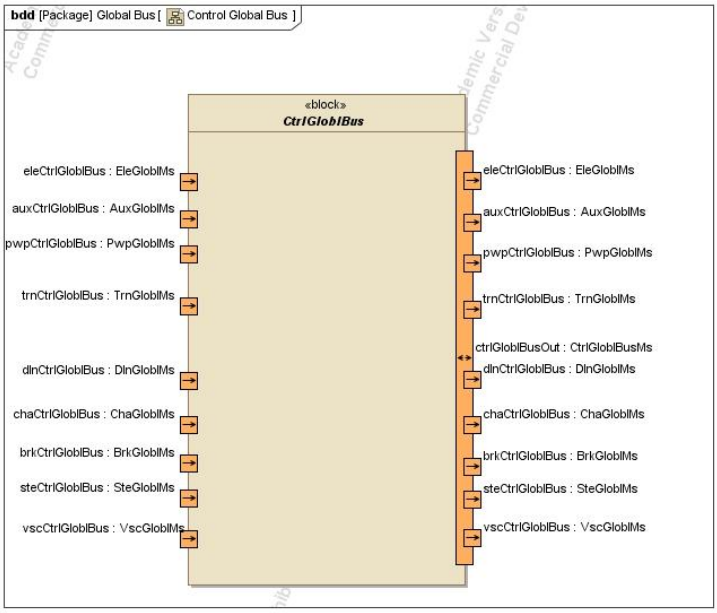


Figure 16. Generic CtrlGloblBus block with the global vehicle signal ports

Also contained within the Vehicle package is the Vehicle block itself. This block contains all of the subsystems as parts. Each usage of a subsystem part is named with its corresponding three letter abbreviation [1]. For example, the Powerplant part name is "pwp." There is also an IBD within the vehicle block that shows an overview of the global signals of the vehicle, Figure 17. This figure shows the CtrlGloblBus, the Control Global Bus, with all of its ports and sub-ports, and then all of the subsystem parts with their respective input and output mixed signals. For example, the "pwp" part has an output signal "pwpCtrlGloblbus" of type "PwpGloblMs" that connects to the "pwpCtrlGloblBus" port on the CtrlGloblBus, and this port passes through the Control Global Bus to the "PwpCtrlGloblBus" port on the output "CtrlGloblBusMs," which then connects to the Powerplant's input signal from the Control Global Bus, the "ctrlGloblBus" of type "ctrlGloblBusMs."

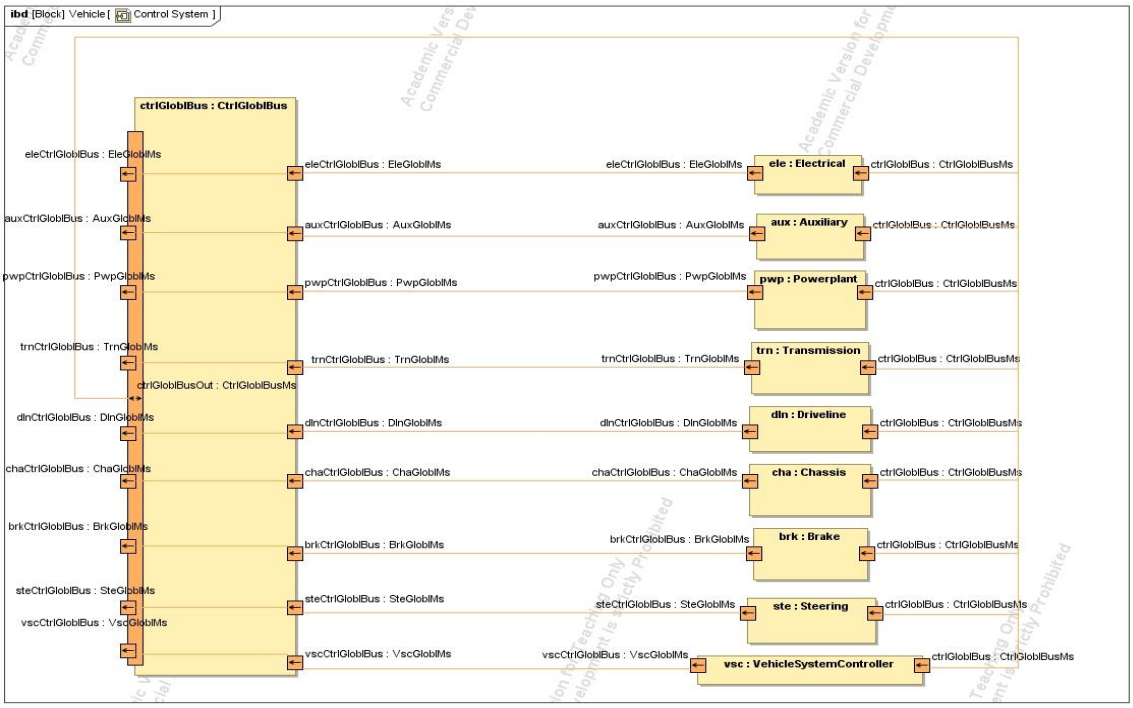


Figure 17. Vehicle's Global Signals Control System

The Vehicle Domain package also contains an abstract *Vehicle Domain* block, which has three parts, the Environment (env), the Vehicle (veh), and the Driver (drv). There are two IBDs that describe the internal interactions within the vehicle domain. First, the Driver Subsystem IBD shows the driver's interactions with the vehicle, as seen in Figure 18. The driver has translational mechanical energy connections with the Powerplant, Transmission, and Brake subsystems, rotational mechanical energy connections with the Steering subsystem, and mixed signal connections with the Electrical and Chassis subsystems.

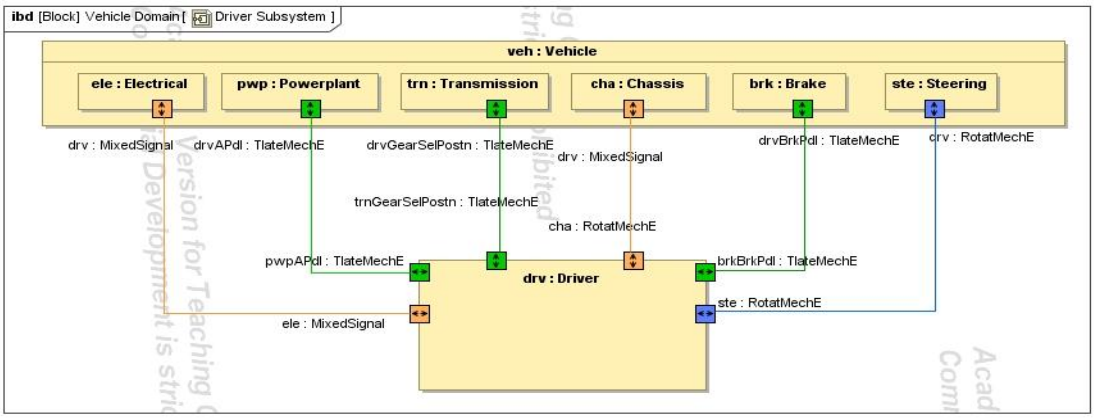


Figure 18. Vehicle Domain IBD describing the Driver's interactions with the Vehicle

The second IBD, labeled the Vehicle Domain, describes all of the interactions between the parts of the vehicle along with the connections to the driver, as shown in Figure 19.

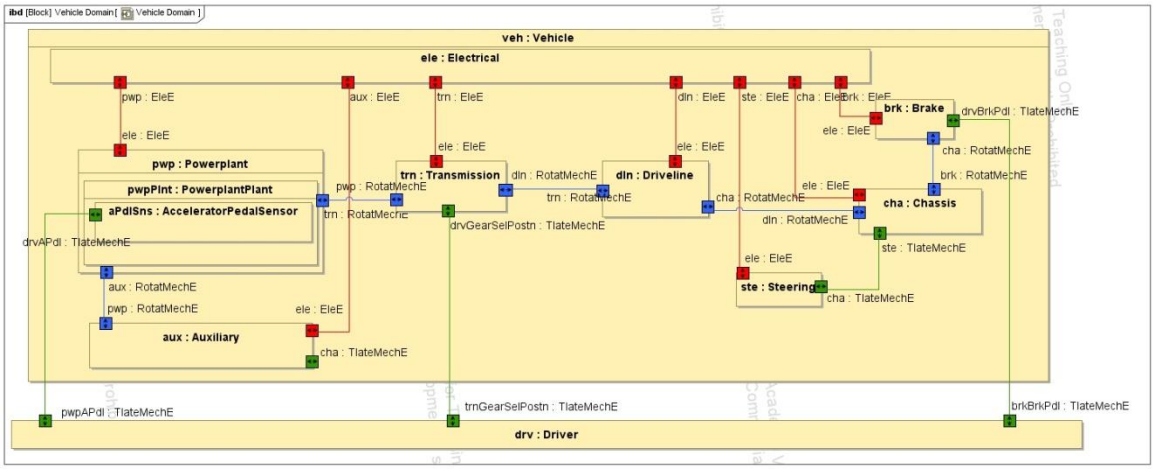


Figure 19. Vehicle Domain IBD describing all the interactions between the Vehicle's subsystems and the Driver

The Vehicle Model Architecture package also contains a package named Units and Flows. Inside the Units package are all of the units created that are not predefined in the standard SysML unit library and might be useful in the definition of the VAMF. Each new unit has three parts. For example, "newtonMeter" is of type DerivedUnit and stereotyped by «Unit». Then it has a «QuantityKind» stereotype named "torqueQK," and

finally, there is a «ValueType» stereotype named "Nm." Each of the new units is created in this fashion. An example of some of the defined units can be seen in Figure 20.

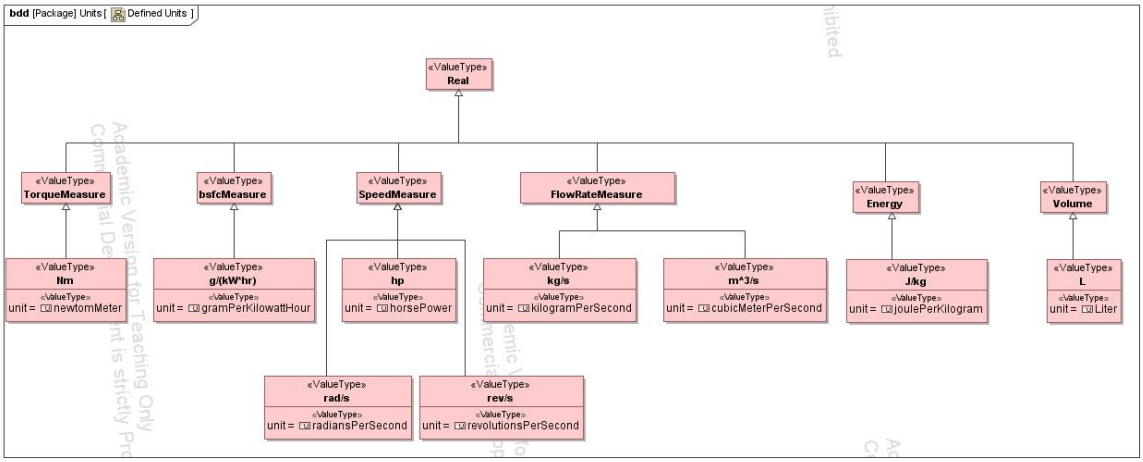


Figure 20. Definition of new units

The Flows package includes Energy, Fluid, and Mixed Signal packages. There is a Flow BDD that defines the hierarchy of energy flow for the VMA system, and it is shown in Figure 21. Three blocks are within the Energy package, which are EleE, for electrical energy, RotatMechE, for rotational mechanical energy, and TlateMechE, for translational mechanical energy. Each energy type has two value properties, one across variable and one through variable. For example, the RotatMechE block has an across variable angle, measured in radians, and a through variable torque, measured in Newton meters. The Fluid package contains blocks for the different types of fluids that could possibly flow through the vehicle system. Finally, the Mixed Signal package includes all of the global and local system mixed signals (e.g. PwpGloblMs, PwpSnsMs, and Pwp ActMs).

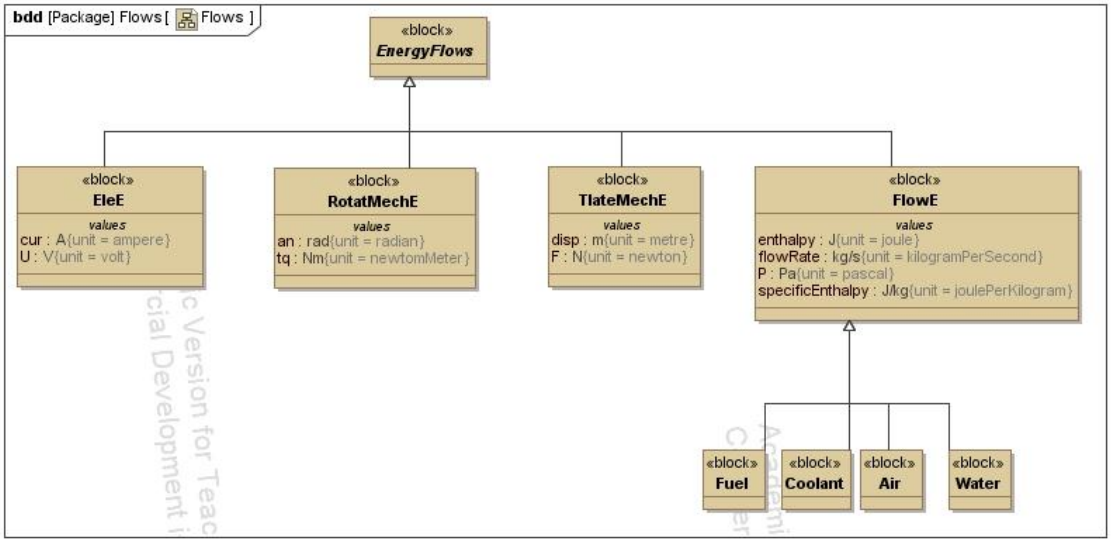


Figure 21. Energy flows hierarchy for the VMA

The last package of the Vehicle Model Architecture package is the Documentation package, where the VMA project can be documented using the *DocBook* profile [13]. DocBook is a plug-in for MagicDraw that aids modelers in documenting his work in SysML and generates a PDF that can be re-generated if there are updates in the model or model's documentation. Using the plug-in, modelers can create a "Book" that contains "Chapters," "Sections," "Paragraphs," "Figure Diagrams," "Queries," and many other elements, as shown in Figure 22. DocBook's plug-in assists the modeler assigning the stereotypes listed in the previous sentence to SysML constructs. For example, the modeler can right click on a package and "Create Book" and a new SysML package is created with the additional stereotype of «book». The plug-in will also ask the user to choose a title for the book, issue, volume, and other descriptors to help define the book. From the newly created «book», modelers can use the Edit Panel to add chapters, sections, paragraphs, etc. to their book, more information as to how to use the DocBook plug-in is given in Appendix C.

DocBook will support the generation and regeneration of documentation for the VAMF. Because the DocBook elements such as diagrams and blocks reside in the VMA model, any change made automatically updates the DocBook elements. Therefore, to update the documentation, all the modeler needs to do is regenerate the DocBook «book».

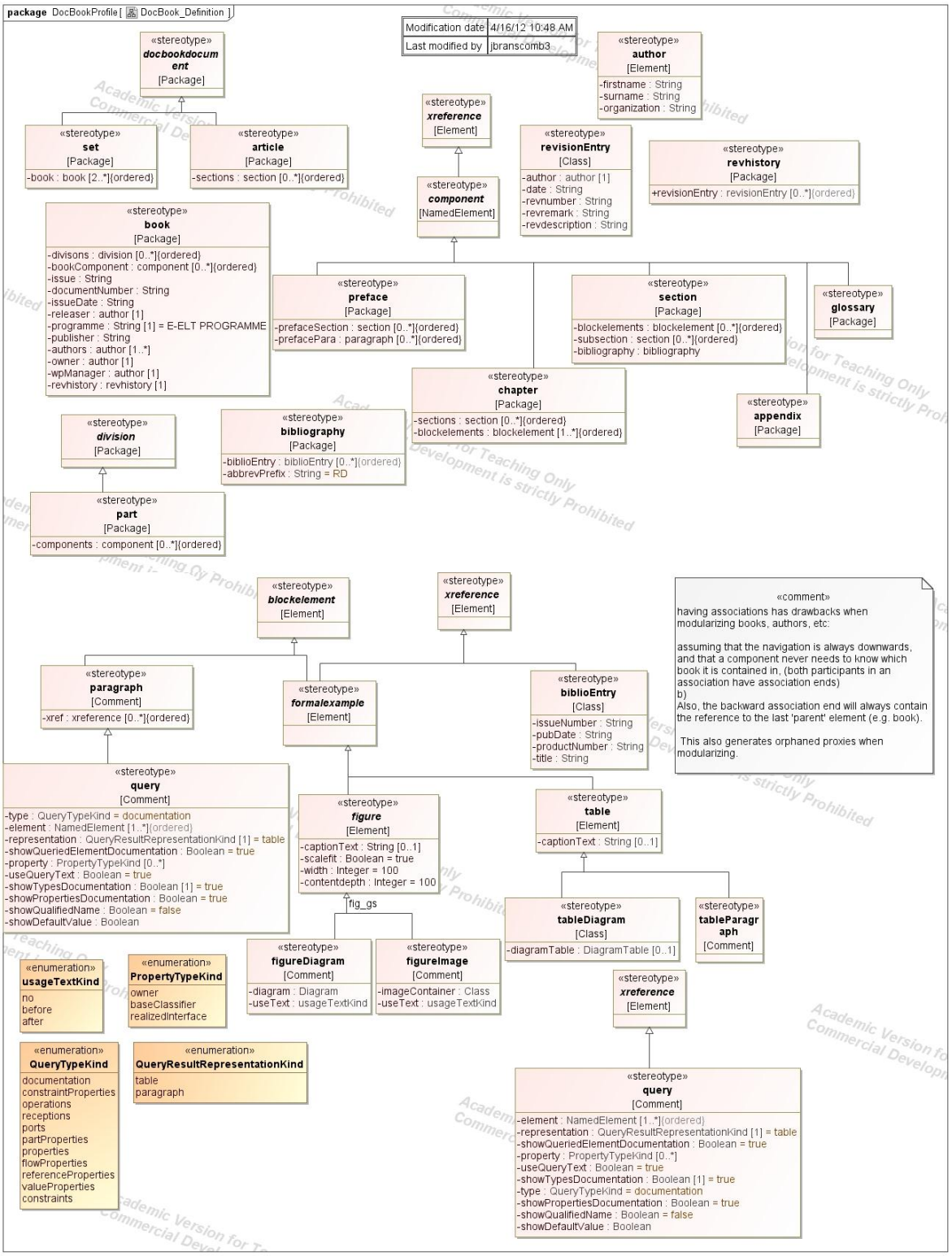


Figure 22. Definition of the DocBook Profile

4.1.3 Specialization of the generic Vehicle Model Architecture

The Vehicle Model Architecture package gives a framework for a generic model of a vehicle. However, in order to actually implement this framework, a Vehicle Specialization package is created for a specific ProgramID, which is the specific name given to a specific vehicle. The structure of the Vehicle Specialization package is very similar to the VMA's Vehicle package in that there are sub-packages for each of the subsystems. The counterpart to the generic *Vehicle* block from the VMA package is the corresponding specialized Vehicle_ProgramID block. The Vehicle_ProgramID's BDD describes how the Vehicle_ProgramID block is a specialized and redefined version of the abstract *Vehicle* block, as shown in Figure 23.

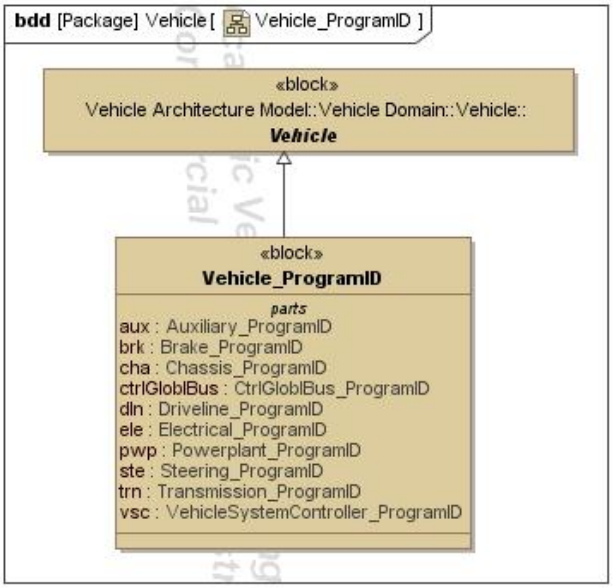


Figure 23. Vehicle_ProgramID block specialized from generic *Vehicle* block

Inside each subsystem package, there is a specific subsystem block, e.g. Powerplant_ProgramID. This block specializes and redefines the generic *Powerplant* block. The parts of the block are also specialized and redefined versions of the

subsystem's *Plant* and *Controller* blocks. Figure 24 shows a BDD of this redefinition mapping.

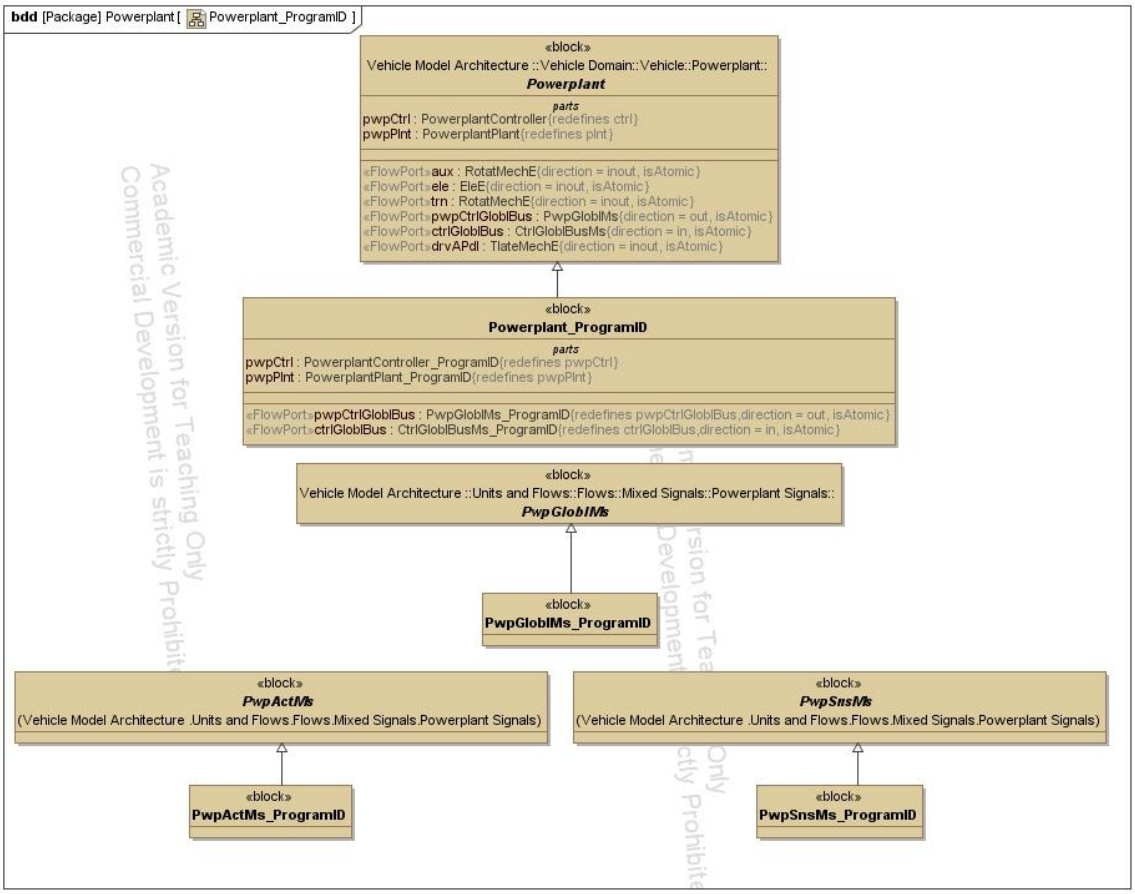


Figure 24. Specific Powerplant BDD showing the specialization from the generic Powerplant

The Bus Structure package shows how the CtrlGloblBus_ProgramID and CtrlGloblBusMs_ProgramID redefine and specialize their generic counterparts from the VMA, as shown in Figure 25. Similar to Figure 17, Figure 26 shows the specialized Control System IBD for the specialized vehicle.

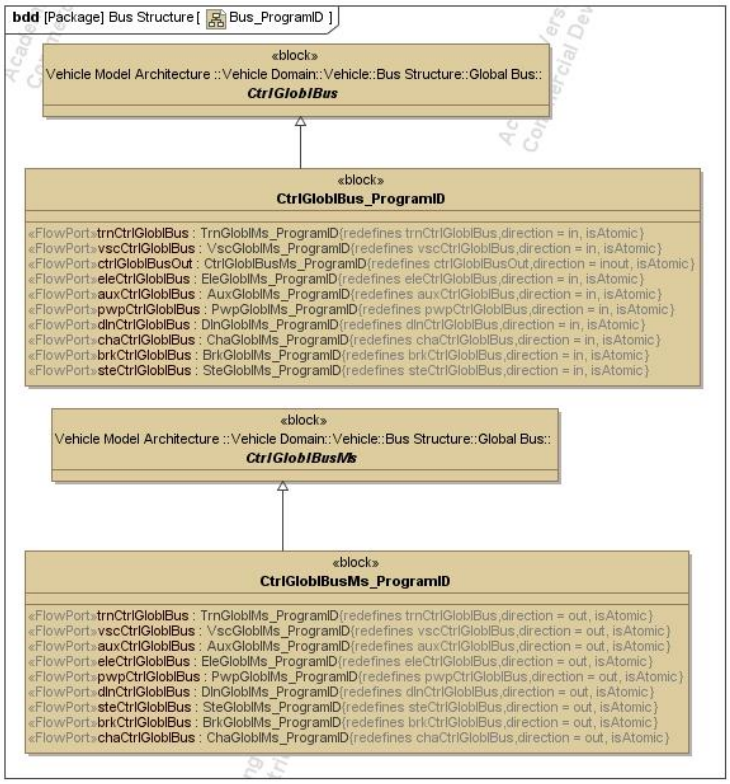


Figure 25. Control Global Bus signal redefinition

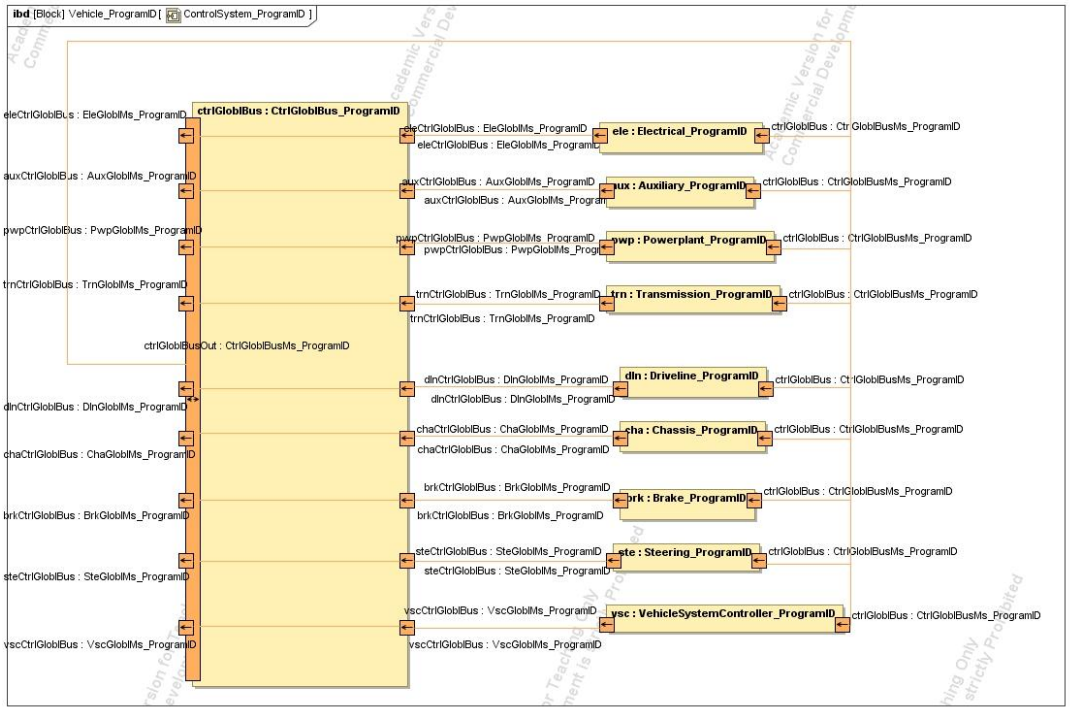


Figure 26. Specialized Control System IBD for the specialized vehicle

One addition to the Vehicle package within the Vehicle Specialization is the Signals package. This package contains sub-packages for each subsystem, and within each subsystem signal package, the corresponding global and local signals are located. For example, the global signal, PwpGloblMs_ProgramID, and the local signals, PwpSnsMs_ProgramID and PwpActMs_ProgramID, are located within the Powerplant Signals package within the Signals package.

4.2 Generation of Analyses

4.2.1 Creating Analysis Architecture for Translation to Modelica

In order to evaluate specific vehicles, these vehicles must pass certain analysis tests set by the company or other government regulations. It would be ideal if a framework for the analyses could be represented in SysML, so that the modeler would know which analyses need to be run, and it would greatly reduce modeling time if modelers could easily generate the analysis architecture for a specific vehicle. This section describes how the analysis templates for the translation to Modelica are defined. The templates are defined in SysML with «Modelica» stereotypes added to the SysML blocks, parts, ports, connections, etc. using the *SysML4Modelica* profile [19]. The *SysML4Modelica* profile allows modelers to assign the «Modelica» stereotypes, and each stereotype additionally has tags for Modelica constructs. For example, a block stereotyped by «ModelicaModel» can have Boolean tags defining the model as being partial or replaceable. An overall generic analysis template for the physical portion of the total vehicle is defined, as shown in Figure 27. In order to ensure the consistency

between the analysis template and the base VMA, there is an association relationship between the structure SysML Vehicle block and the analysis Vehicle Modelica model, as depicted in Figure 28.

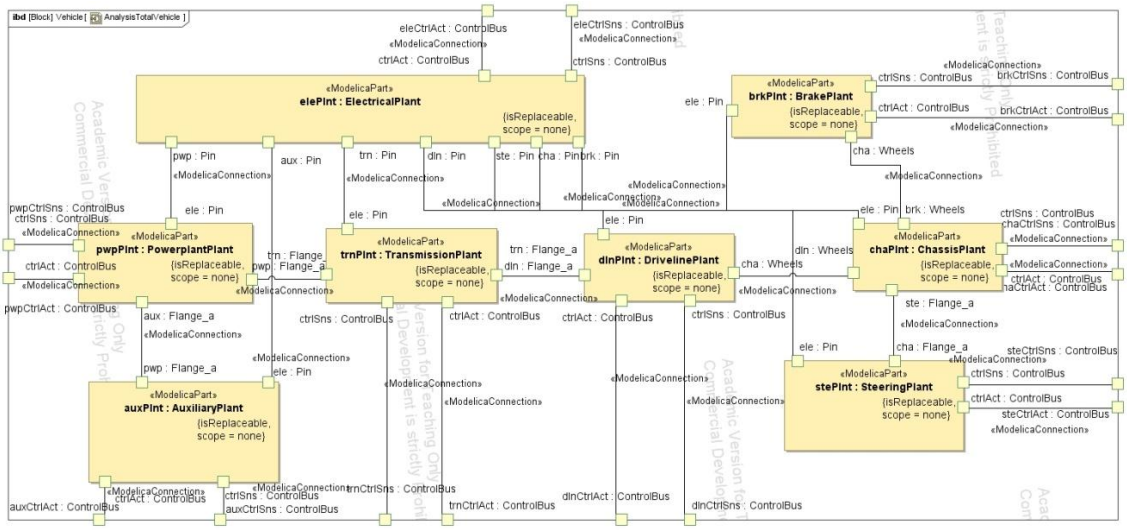


Figure 27. The analysis template for the total vehicle as defined in the *SysML4Modelica* profile

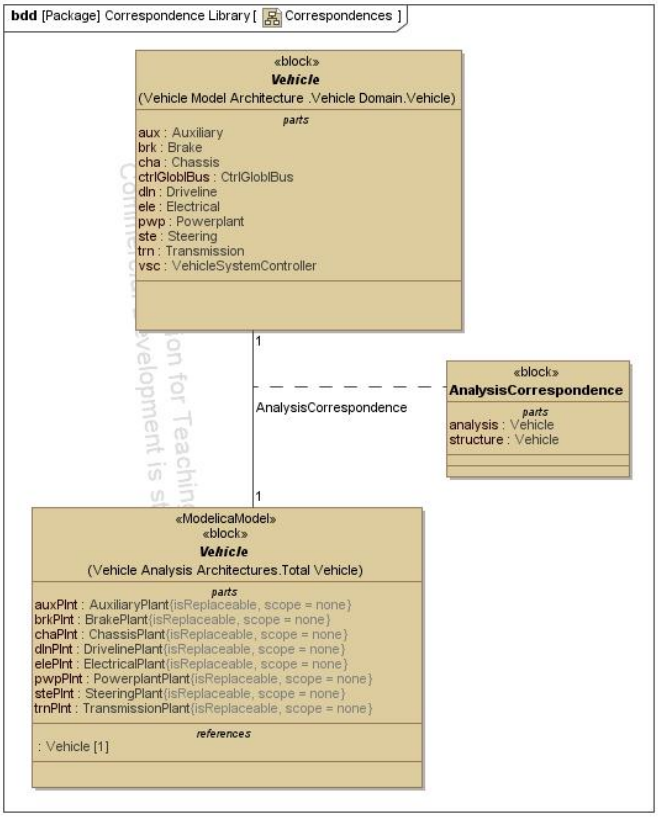


Figure 28. The Correspondence between the generic Modelica Vehicle model and the generic SysML Vehicle block

Figure 29 depicts an IBD that is associated with the "Analysis Correspondence" block, which is shown in Figure 28. The left side of Figure 29 illustrates the generic structure of the SysML Vehicle, and the right side of the figure illustrates the generic analysis Modelica stereotyped Vehicle. This IBD shows how the Modelica analysis template corresponds to the SysML structure template.

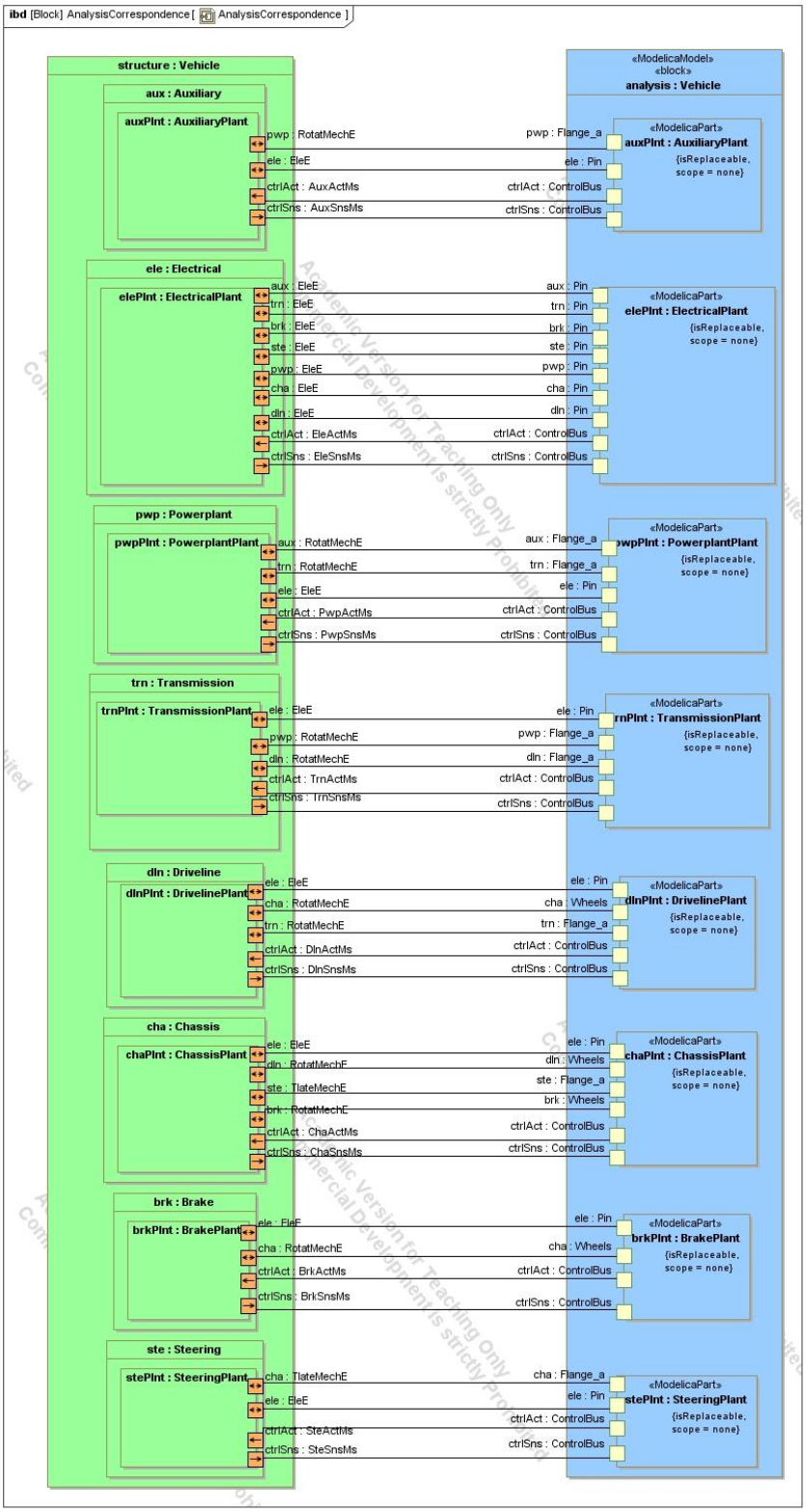


Figure 29. IBD of the "Analysis Correspondence" association block showing the relations between the SysML Vehicle model and the Modelica Vehicle model.

Because not all of the subsystems, ports, and parts are needed in each analysis, originally separate templates were created for each analysis, for example fuel economy, performance, etc. However, some of the analyses templates are identical, except they analyze different aspects of the vehicle. This issue may have led to inconsistencies in the model because these identical templates would not be correlated with each other. To solve this problem, one generic template of the entire vehicle is created with all of the subsystems, ports, and parts that are stereotyped by «Modelica» stereotypes defined by the *SysML4Modelica* profile. A new SysML block stereotyped by «ModelicaModel» is created for each analysis, and this new analysis relates to the entire vehicle model through dependencies. Each new analysis is automatically updated to the analysis configuration matrix, as shown in Figure 30.

Figure 30. Analysis Configuration Matrix for three example analyses

To aid the modeler in creating a specific analysis for a specific vehicle, a plug-in for MagicDraw has been developed that asks the modeler which of the pre-defined analyses he wishes to implement for a specific vehicle program. The plug-in then generates a specialized analysis for the given vehicle program by copying the template and renaming the blocks and parts to represent the specific program. Once the chosen analysis has been specialized for a specific program, it can then be translated into Modelica through another developed MagicDraw plug-in. The plug-in takes the analysis, which is stereotyped by «Modelica» stereotypes and generates a ".mo" file that can be

opened in a Modelica based program, such as Dymola. The file only generates the framework template for each of the subsystems. The internals of each subsystem must be populated with the appropriate Modelica model content. This entire translation process from SysML to Modelica is further discussed in the example in Chapter 5.

4.2.2 Defining the Translation to Simulink

In addition to generating analyses for the physical portions of the VMA that will be translated into Modelica, the controls logic portion of the vehicle system architecture will be translated to Simulink. From the vehicle program specific package in SysML, the modeler can run another developed MagicDraw plug-in that will transform the program specific SysML model into an ".m" file, which can be opened in the Matlab editor and run to generate corresponding Simulink models. A separate Simulink model is created for each subsystem depicting the subsystem's inputs and outputs, and within the subsystem there are the local and global program specific signals connected by buses, as seen in Figure 31 and Figure 32. The subsystem templates can be used by the modeler to fill in the appropriate control models. This process of translation the SysML model to Simulink is further described in Chapter 5.

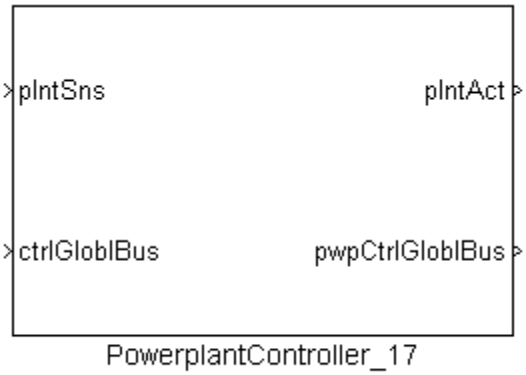


Figure 31. Example of a Simulink subsystem model

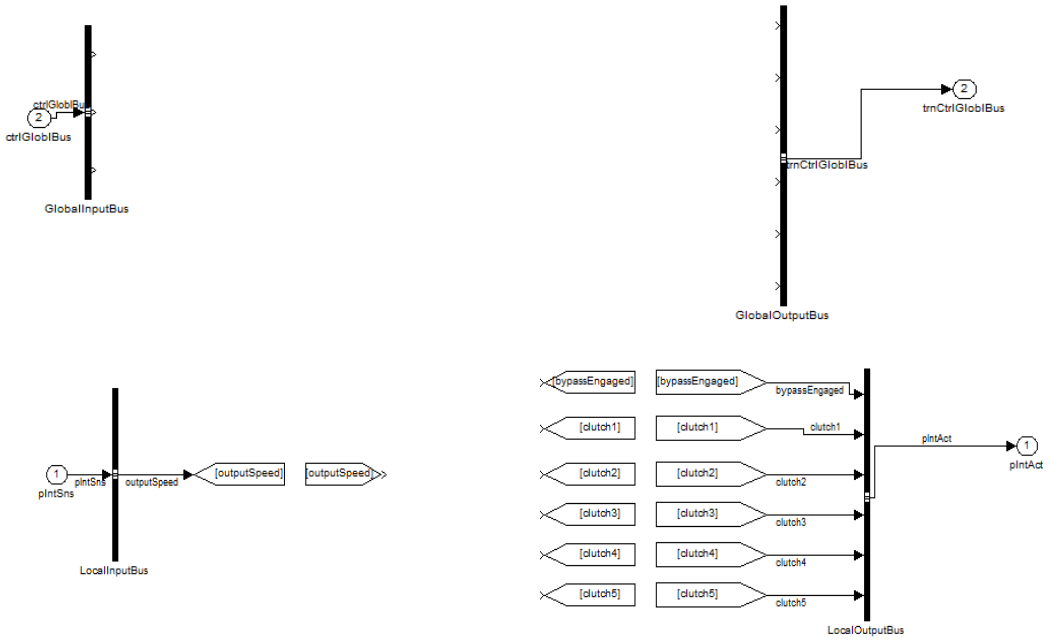


Figure 32. Example of a generated subsystem template that needs to be further defined

4.3 Summary

In summary, the Vehicle Architecture Modeling Framework is an approach to formally support the vehicle system model development process for complex modern vehicles. It gives engineers the capability to adapt, adjust, and reuse the base architecture while ensuring consistency throughout the model. First a generic vehicle architecture is created, and from this architecture, a template for a specialized and redefined vehicle is defined. From the specialized vehicle, engineers can create an architecture model for a vehicle with a specific Program ID, for example, "C100". Because SysML does not have the ability to analyze and validate alternatives, the specific vehicle is then translated into Modelica and Simulink component model templates through the use of several plug-ins. The model templates for the physical side of the specific vehicle containing all of the hardware connections and energy flows are transformed to Modelica. Then the subsystem templates are handed off to the domain experts to be augmented with appropriate Modelica subsystem models. Similarly, the framework for the controls side of the specific vehicle containing the logical signals flows is transformed to a Simulink template. Again the subsystem templates are given to the domain experts to be supplemented with existing Simulink controllers to complete the model. Once these processes have been completed, the Modelica model is imported into Simulink and the two models are connected using Simulink constructs. Then the entire vehicle model can undergo simulation and validation. An example of this whole approach is given in Chapter 5 for the C100 specific vehicle and a 0 to 100 kph performance analysis.

CHAPTER 5:

EXAMPLE: ACCELERATION ANALYSIS FOR THE C100 VEHICLE MODEL

5.1 Introduction to the C100 Specialized Vehicle Example

The example model analyzed in this chapter is intended to demonstrate the capabilities of using the VAMF approach given in Chapter 4 as a basis for creating specialized vehicles and validating these specialized vehicles through multiple analyses. If the example of the specific performance analysis test of "0 to 100 kph time" is able to be verified for the specialized vehicle, named "C100", then the VAMF approach is proved successful and would greatly support engineers creating complex vehicle system models.

5.2 Aptness of the C100 and the 0 to 100 kph Performance test

The "C100" Program ID was chosen as the specific vehicle to be used in this example; however, this process can be completed for any vehicle Program ID. The 0 to 100 kph Performance test is a simple example that is used to demonstrate the capability of the entire VAMF approach. Only a subset of the vehicle's nine subsystems is necessary, and the number of local and global signals is minimized for simplicity.

5.3 Guide to Creating Specific Vehicle C100 Example

The purpose for this work is to use model-based systems engineering to structure a vehicle architecture model that includes an overall logical framework, which can be specialized for a specific vehicle, and then perform different scenario analysis testing incorporating both the logical and physical portions of the vehicle. The process flow developed for this thesis can be seen in Figure 33. This chapter will highlight portions of the process flow and describe the steps necessary in creating and analyzing a new vehicle.

Before the process flow can begin, an overall generic system model in SysML is created, which represents the vehicle's subsystems, logical signals, physical energy flows, and connections between the logical and physical portions of the vehicle. Once the generic model is created, then it can be specialized within SysML for a specific vehicle, which corresponds to 5 and 6 in Figure 33. Several analysis tests must be performed on each specific vehicle. To complete these analyses, the analysis architecture template must be checked out, as in 3 of Figure 33, and then the specific vehicle and program information can be combined with the analysis template to create a specific analysis for a specific vehicle, as in 7 of Figure 33.

Then the structure and interfaces of the model are translated into the Modelica language in order to run analyses on the plant side of the vehicle, as in 9 of Figure 33. The translation process includes adding «Modelica» stereotypes to the specific physical portions of the specialized vehicle model and converting the SysML constructs into Modelica constructs. In parallel with this progress, the structure and interfaces of the controls logical side of the vehicle model is translated into Simulink, as in 10 of Figure

33. Then the internals of the Modelica plant and Simulink controls models are developed and populated. This process is done by several Domain Model Developers in both Modelica and Simulink. The Modelica Domain Model Developers receive the plant model templates and the Simulink Domain Model Developers receive the control model templates from the SysML Program Analyst and each adds in appropriate domain models in their environments. In order to link the logical and physical portions of the vehicle together, shown by 18 of Figure 33, we implement the Dymola block, which interfaces between Simulink and Modelica in order to simultaneously solve Modelica models during Simulink simulations. Through the Dymola block's parameter window, the complete Modelica vehicle model containing all of the physical portions of the model can be imported into Simulink. Then buses were used to link the signals between the local plant and controllers. Once this process is complete, the analyses can be run in Simulink and the results examined, as in 23 of Figure 33. A general guide to completing these tasks for the C100 vehicle and the 0 to 100 kph performance analysis is given in this section, and a step by step guide is given in Appendix A.

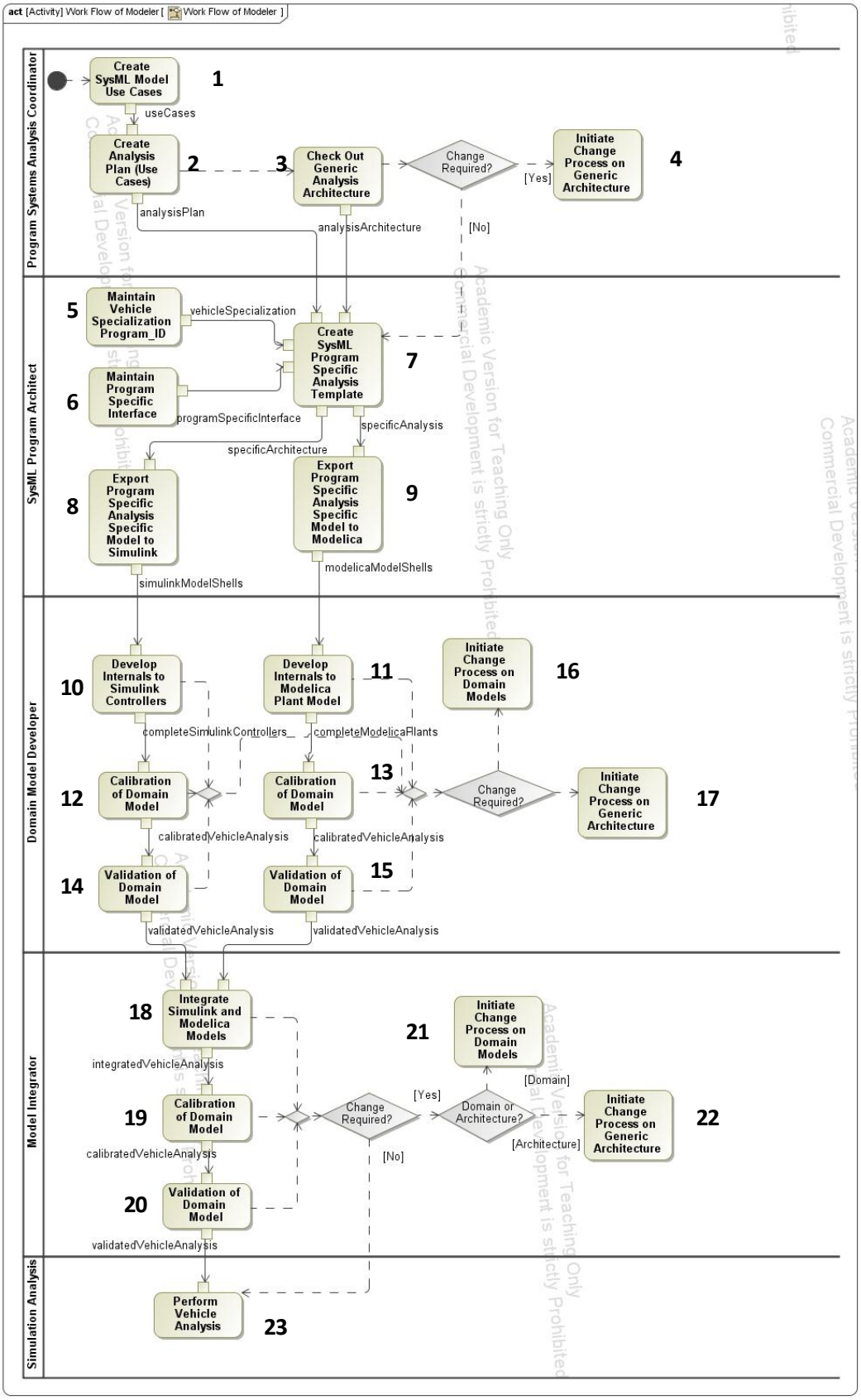


Figure 33. Process flow modelers of the VAMF

5.3.1 Defining the SysML Model of the C100 Vehicle

Chapter 4 describes how a generic specialized vehicle is created through redefinition and specialization. This specialized vehicle is for a generic "ProgramID" template that can be used to create a specific vehicle. To accomplish this task, the modeler must make a copy of the Vehicle Specialization package and rename all of the blocks by removing the "ProgramID" generic name and replacing it with "C100," which is completed in 5 of Figure 33. For example, the "Powerplant_ProgramID" block would be renamed "Powerplant_C100". Once every block has been renamed, the new specialized vehicle package would look like Figure 34 (Right).

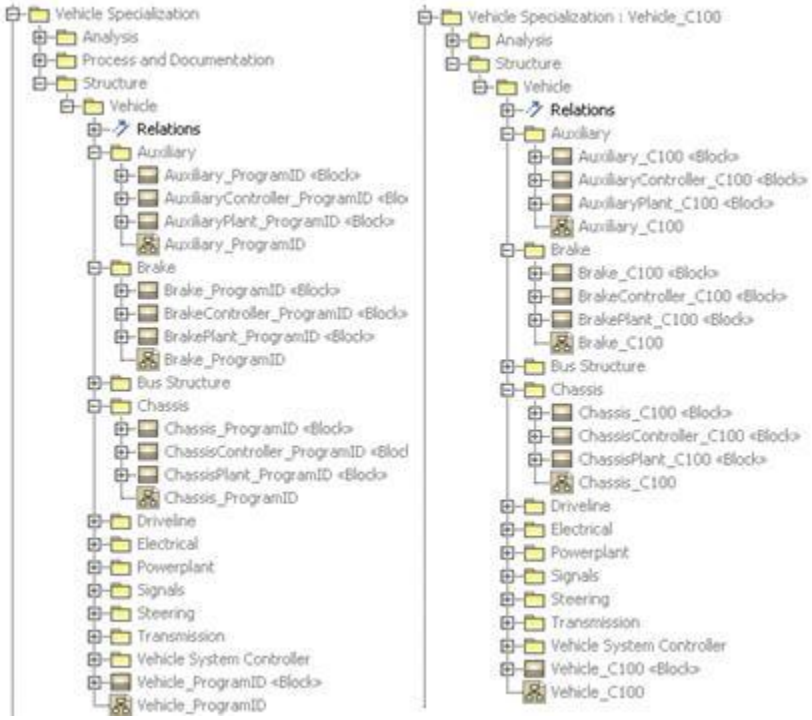


Figure 34. (Left) Generic specialized vehicle package, (Right) C100 Vehicle package

Each of the blocks in this package still are specialized and redefined versions of their generic, abstract VMA counterparts. In this manner, the new specific vehicle can remain consistent with the base vehicle template.

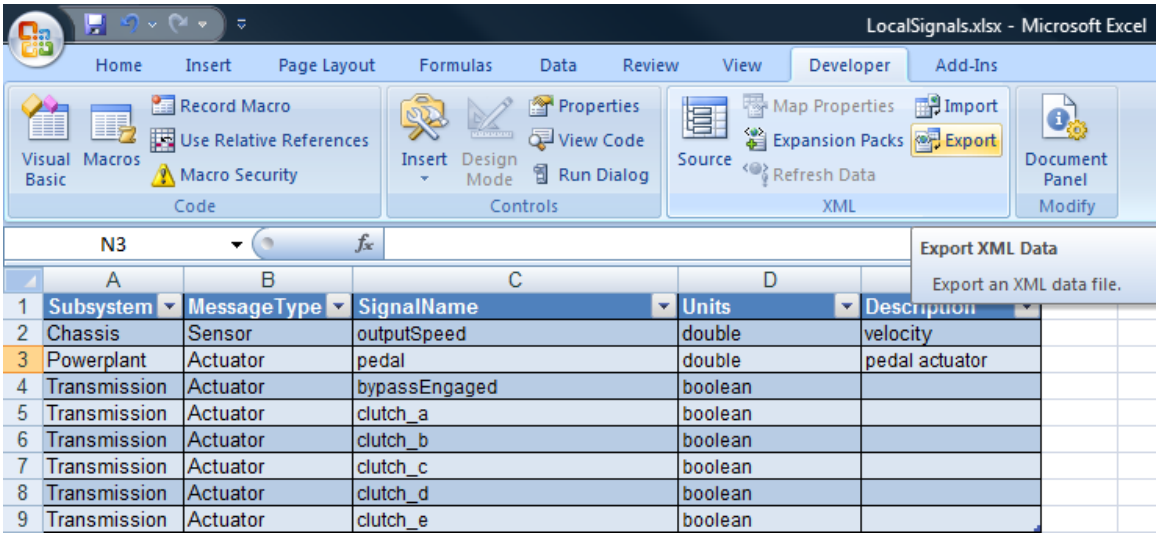
5.3.2 Importing the Global and Local Signals

Before the specific vehicle component templates can be exported to Modelica and Simulink, the modeler must first setup and then import the specific vehicle's global and local signals, which is accomplished during 6 of Figure 33. The global signals are the signals that pass between each subsystem on the Control Global Bus, and the local signals are signals that pass between each subsystem's local plant and controller pieces.

The local signals are setup in an Excel file whose cells have been designated by an Extensible Markup Language Schema Definition (XSD) to map to the correct nodes when generating an Extensible Markup Language (XML) file. The modeler can modify the Excel template to add the necessary local signals, as shown in Figure 35, and then export the file to XML using the export function of the Developer tab, as depicted in Figure 36.

Subsystem	MessageType	SignalName	Units	Description
Chassis	Sensor	outputSpeed	double	velocity
Powerplant	Actuator	pedal	double	pedal actuator
Transmission	Actuator	bypassEngaged	boolean	
Transmission	Actuator	clutch_a	boolean	
Transmission	Actuator	clutch_b	boolean	
Transmission	Actuator	clutch_c	boolean	
Transmission	Actuator	clutch_d	boolean	
Transmission	Actuator	clutch_e	boolean	

Figure 35. Local Signals for the C100 Excel file with XML mapping



	A	B	C	D	E
1	Subsystem	MessageType	SignalName	Units	Description
2	Chassis	Sensor	outputSpeed	double	velocity
3	Powerplant	Actuator	pedal	double	pedal actuator
4	Transmission	Actuator	bypassEngaged	boolean	
5	Transmission	Actuator	clutch_a	boolean	
6	Transmission	Actuator	clutch_b	boolean	
7	Transmission	Actuator	clutch_c	boolean	
8	Transmission	Actuator	clutch_d	boolean	
9	Transmission	Actuator	clutch_e	boolean	

Figure 36. Export File to XML using the Export Function of the Developer Tab

To support the modeler in adding these signals, a plug-in has been developed. By right-clicking on the specialized vehicle block, in this case “Vehicle_C100”, the modeler can click on the plug-in named “Import Signals” and then either select “Import Global Signals” or select “Import Local Signals” as depicted by Figure 37.

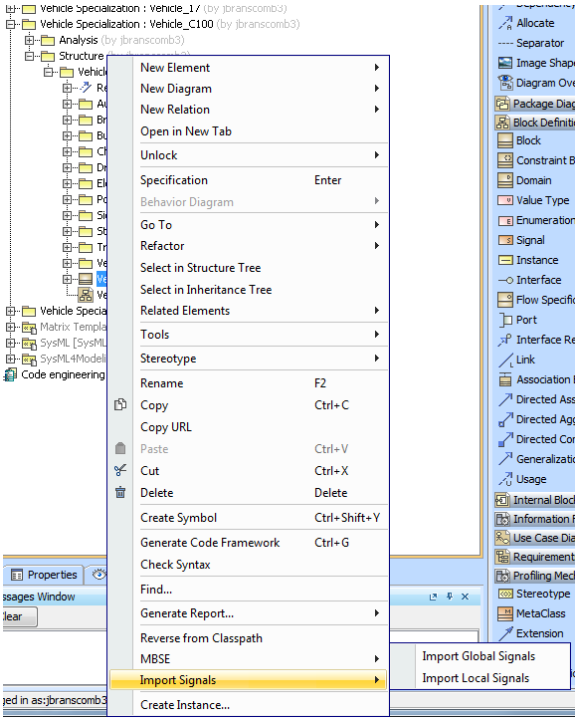


Figure 37. Import Signals Plug-in

Then a graphical user interface (GUI) window appears with the file browser asking the modeler to select the XML file that contains either the global or local signals, depending on which option the modeler chooses, as seen in Figure 38.

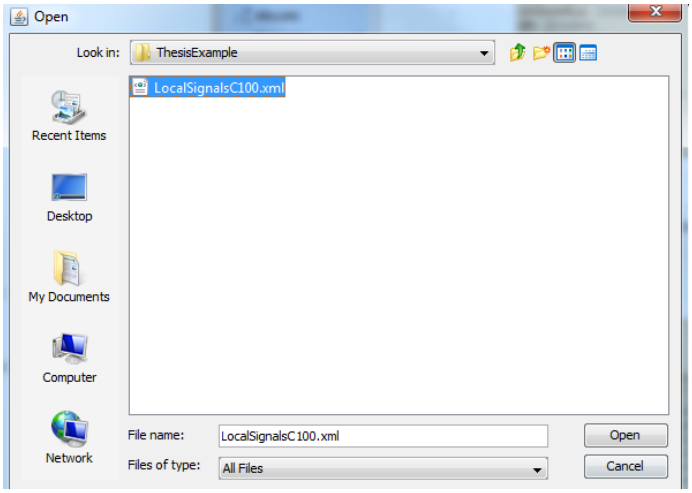


Figure 38. File Browser with the Local Signals for the C100 XML File

Once the XML file is opened, the plug-in then creates the signals as flow properties of the corresponding subsystem's signal blocks. In this example, the C100 has a local Powerplant actuator signal named "pedal" which is added as a property of the PwpActMs_C100 signal block. The added local signals for the C100 example can be seen in Figure 39.

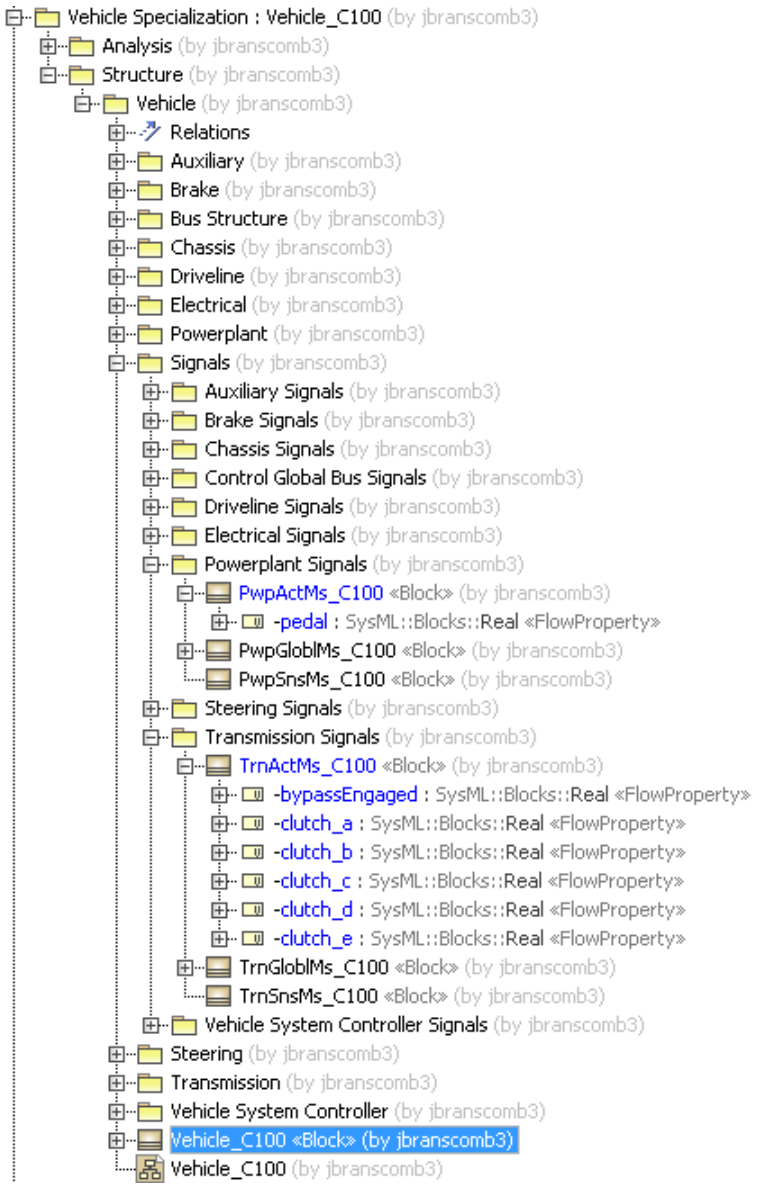


Figure 39. The Imported Local Signals augment the C100 Vehicle Model

A similar process is followed for adding the global signals to the specific vehicle model. The global signals list can be exported to XML using Vector CAN tool features. However instead of adding properties to the Sensor and Actuator signals of the subsystems, the global signals for the Powerplant are added to the "PwpGloblMs_C100" block, for example. Once the signals have been imported to the specialized vehicle model, the specific analysis can be added for the specialized vehicle, as in 7 of Figure 33.

5.3.3 Translating the SysML C100 Model to Modelica and Simulink

After the local and global signals have been imported into the specialized vehicle model, the model is ready to be translated into Modelica and Simulink for analysis, which occurs in 8 and 9 of Figure 33. These transformations are a multi-step process, which is also supported by plug-ins.

5.3.3.1 SysML to Modelica Translation

Translating the C100 SysML model to Modelica is a two-step process. First the modeler must run the plug-in that creates *SysML4Modelica* models corresponding to the necessary subsystems for the desired analysis. As described in Chapter 4 there are pre-defined, program generic analysis templates. These generic analysis templates must be made specific to a specific vehicle program before translated to Modelica. Therefore, the modeler again right-clicks on the "Vehicle_C100" block and runs the "SysML Transformations" plug-in choosing the "Generate Modelica Analysis" option, as shown in Figure 40. Then a GUI window appears asking the modeler to choose the desired analysis, as seen in Figure 41. Once the analysis is chosen, another GUI asks the user to input the desired ProgramID, "C100" in this example, which is depicted in Figure 42.

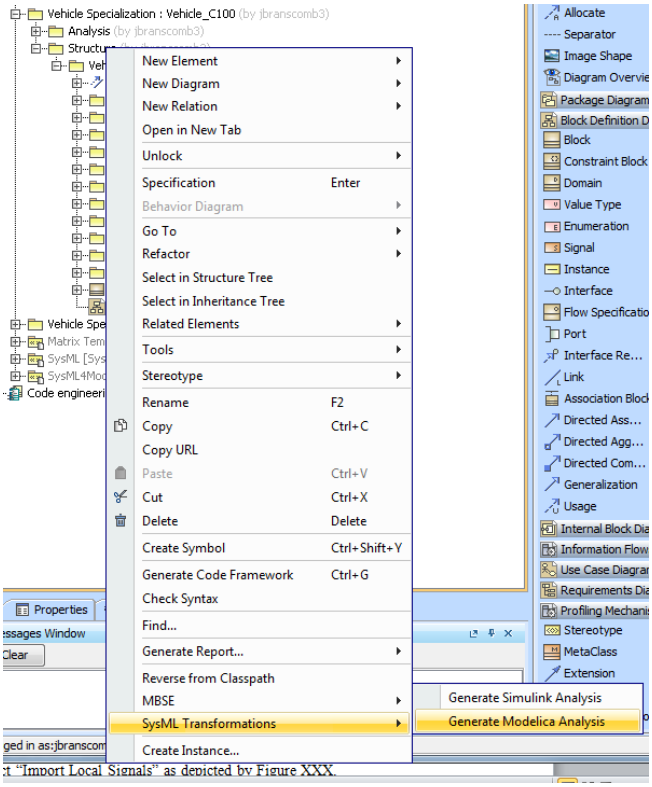


Figure 40. Running the Modelica Analysis Plug-in

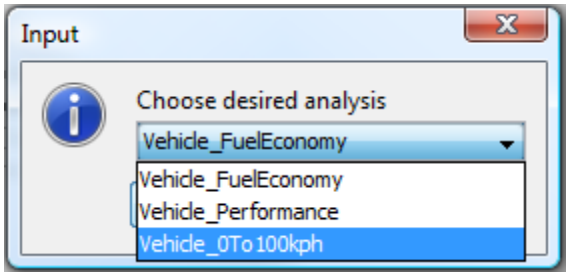


Figure 41. Choosing the desired analysis

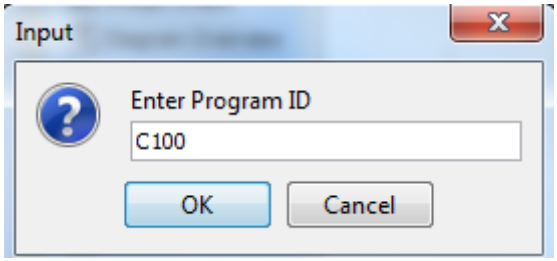


Figure 42. Choosing the desired ProgramID

After the specific ProgramID has been entered, a new package for the specific analysis of the specialized vehicle appears in the Analysis package. Because the 0 to 100 kph performance test does not require all of the subsystems for analysis, only a subset of the vehicle's subsystems appear, as seen in Figure 43. The Powerplant subsystem and the Vehicle are expanded for demonstration. These newly created subsystems are Modelica models, as can be seen by their «ModelicaModel» stereotype. They also only have the necessary ports to capture the interactions between the necessary subsystems. The analysis model is now ready to be translated into Modelica as in 9 of Figure 33.

Figure 43. Performance Analysis for the C100 vehicle

5.3.3.2 Supplementing Transformed Modelica Model with Preexisting Models

The analysis *SysML4Modelica* architecture model is now able to be translated to Modelica, through the use of another plug-in. The modeler right-clicks on the Vehicle_C100 «ModelicaModel», which is under the newly created Vehicle_0To100kph package within the Analysis package of Vehicle Specialization: Vehicle_C100, as seen in Figure 44. The modeler runs the plug-in named "SysML to Modelica" and chooses the "Generate Modelica" option. The output of this file is a ".mo" file which can be opened in Dymola. The newly created Modelica file is a template Modelica model, only containing templates of the subsystems with input and output ports. The subsystems must be augmented with fully defined subsystem models. An example of the C100 Transmission subsystem template is given in Figure 45.

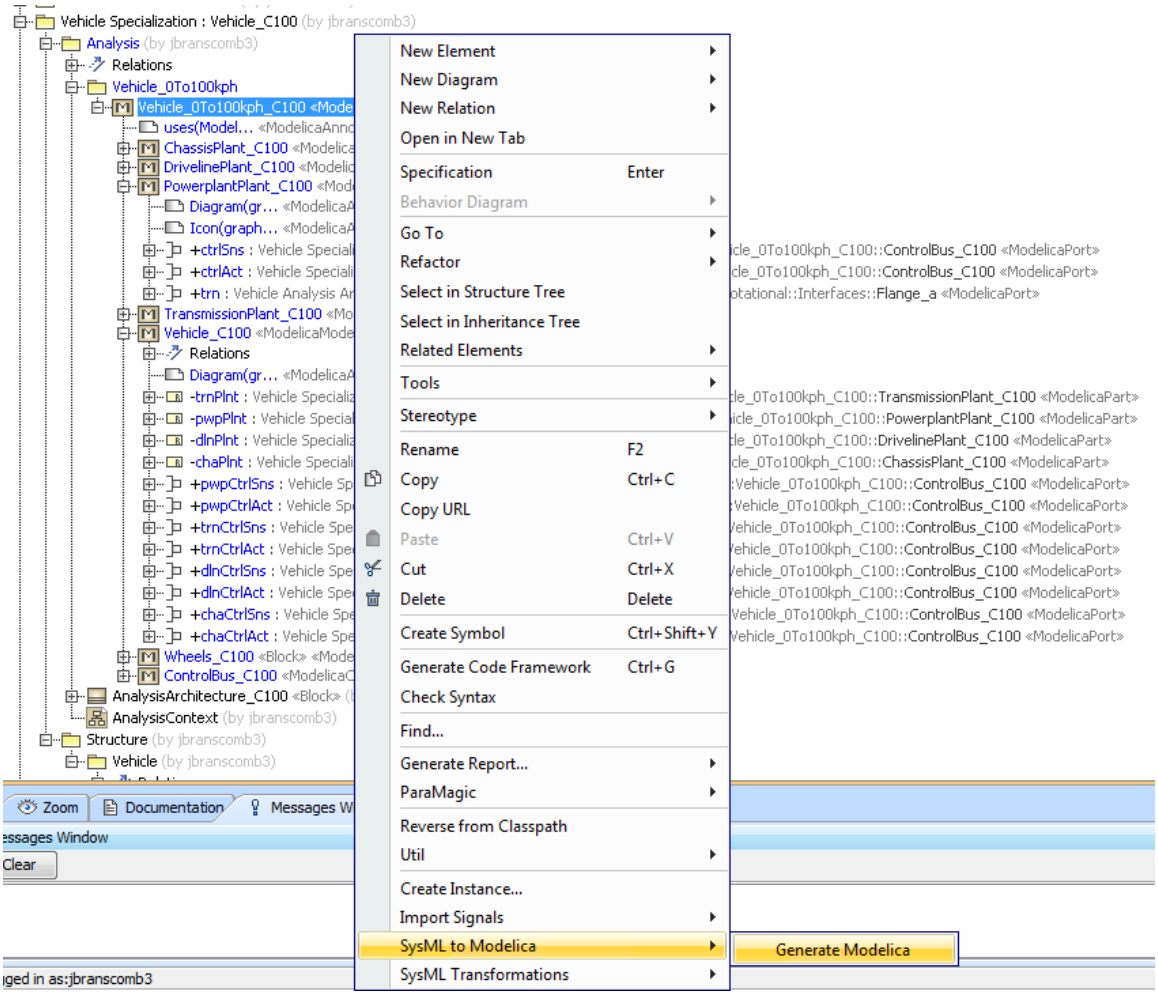


Figure 44. Generating the file containing the SysML to Modelica translation

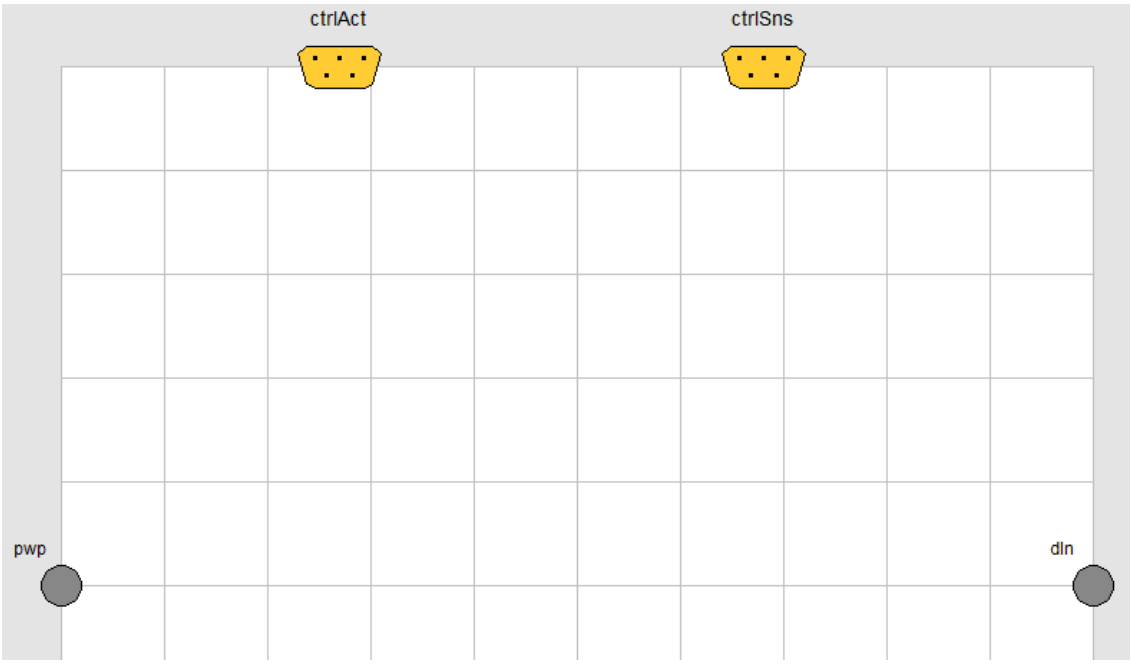


Figure 45. TransmissionPlant_C100 subsystem showing its local Control Buses and Rotational Connections to the Powerplant and Driveline

In order to complete the models, the modeler must fill in the details of each subsystem and the vehicle with pre-existing Modelica models, as in 11 of Figure 33. If there is a preexisting subsystem model, the domain expert needs to open the file containing the existing model in the same Dymola window as the template. Then in the overall Vehicle_C100 system view, domain expert can right-click on the plant subsystems and choose "Change class." Then they can browse through the existing Modelica models and select the corresponding model. If a subsystem model does not already exist, then the domain expert can develop the physical model inside the template generated from SysML, seen in Figure 45, using Modelica blocks and modifying the Modelica text to represent the desired fully defined subsystem model. The fully defined Vehicle_C100 system would then appear as in Figure 46.

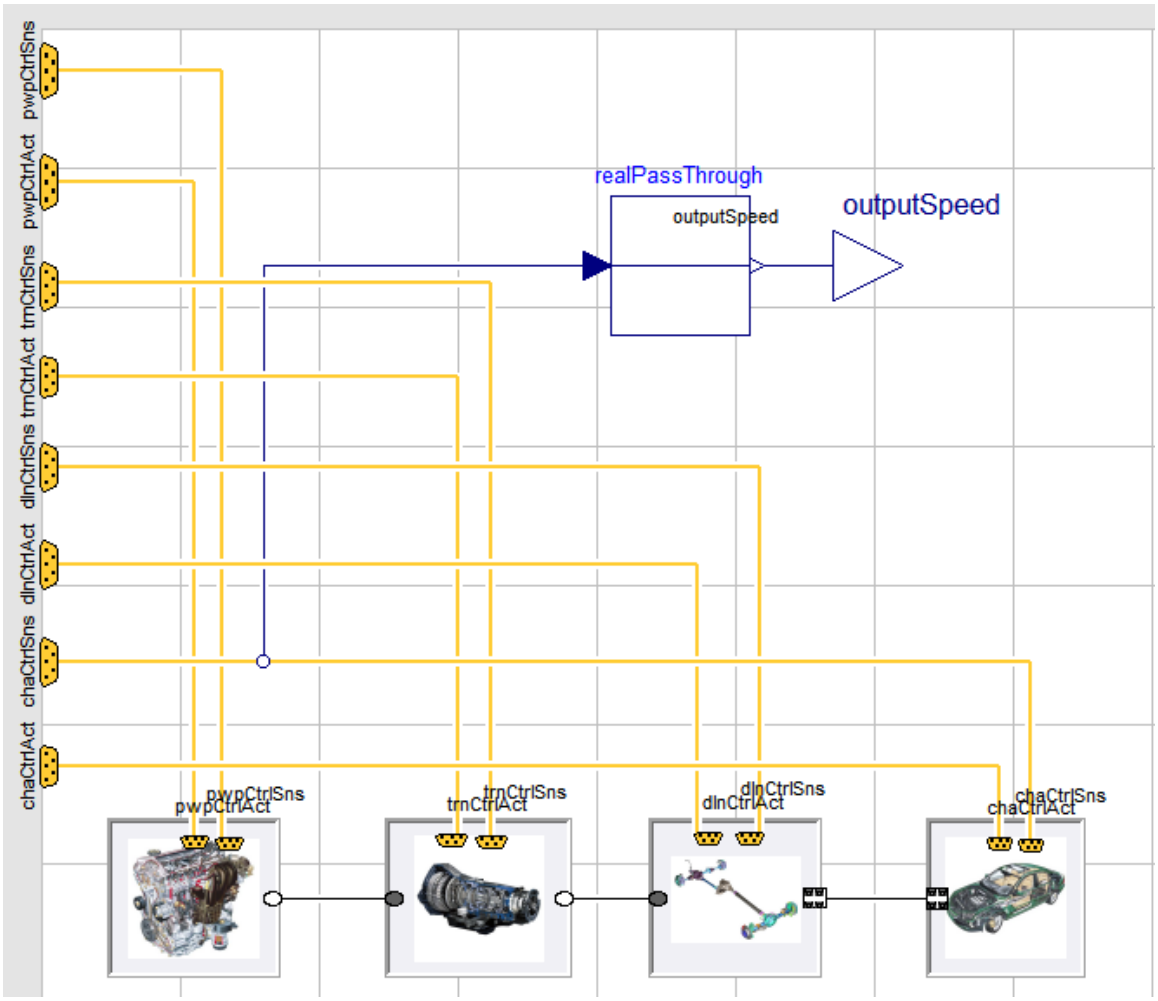


Figure 46. Vehicle_C100 system fully-defined

5.3.3.3 SysML to Simulink Translation

In conjunction with the translation of the physical plant analysis architecture model to Modelica, the logical controls architecture model is exported to Simulink through the use of another plug-in, which is completed as part of 8 in Figure 33. To begin the plug-in, the modeler would again right-click on the "Vehicle_C100" block within the "Structure" package of the C100 specialization package. The modeler again chooses the "SysML Transformations" plug-in, but this time picks the "Generate Simulink Analysis" option, as seen in Figure 47.

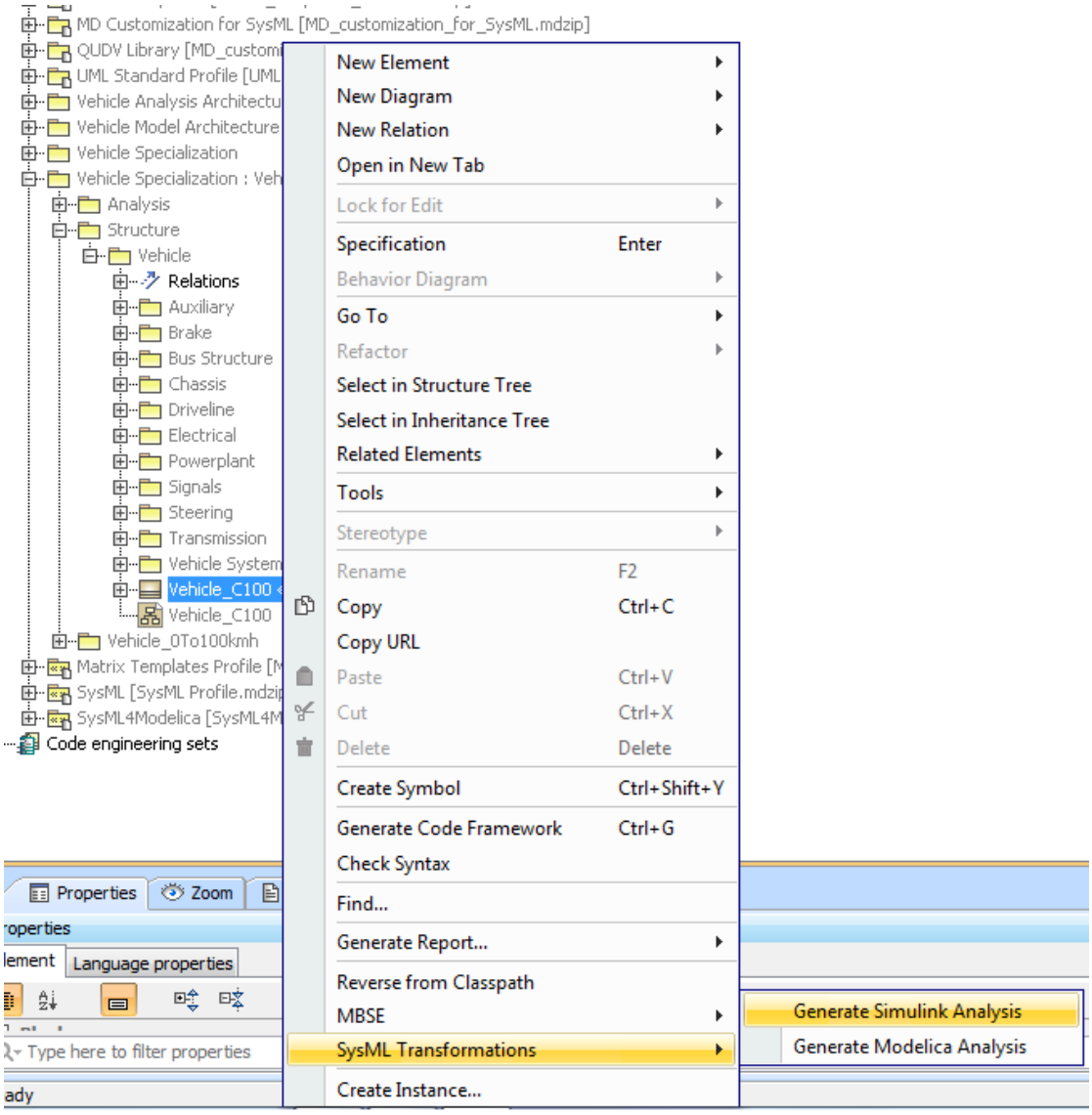


Figure 47. Initiating SysML to Simulink Transformation

The plug-in generates an ".m" file script, which can be opened in a Matlab editor and run to create the necessary Simulink subsystem controller models. The script contains all of the local and global signal information to populate the 9 subsystem controller models. As can be seen in Figure 48 each subsystem controller has two input and output pairs, one for the local signals and one for the global signals.

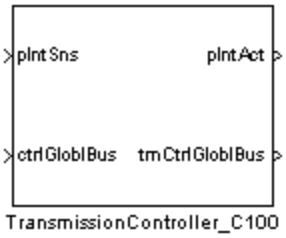


Figure 48. Example of the TransmissionController_C100 Subsystem

Inside of each subsystem, there are 4 buses, which are connected to each of the inputs or outputs. Connected to the buses are signal pairs, in the form of Simulink's "Goto" and "From" tags. Each signal has a pair of these tags that link the signals to the bus and to the internals of the subsystem, as shown by Figure 49. The next step in this process is for the domain expert to supplement the template models of each subsystem with appropriate Simulink controller models.

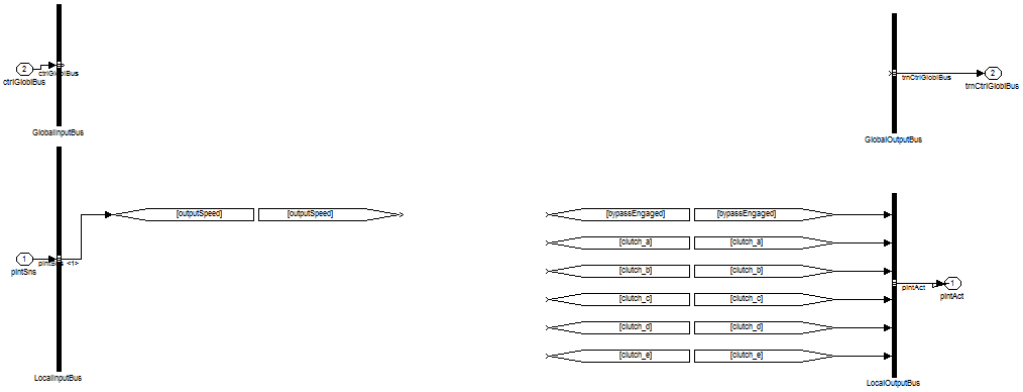


Figure 49. Internals of the C_100 Transmission Controller

5.3.3.4 Supplementing Transformed Simulink Models with Appropriate Models

In order to complete each of the logical controllers, the domain expert must populate each subsystem with the desired controls algorithms for the specialized vehicle, as in 10 of Figure 33. An example of a completed subsystem can be seen in Figure 50.

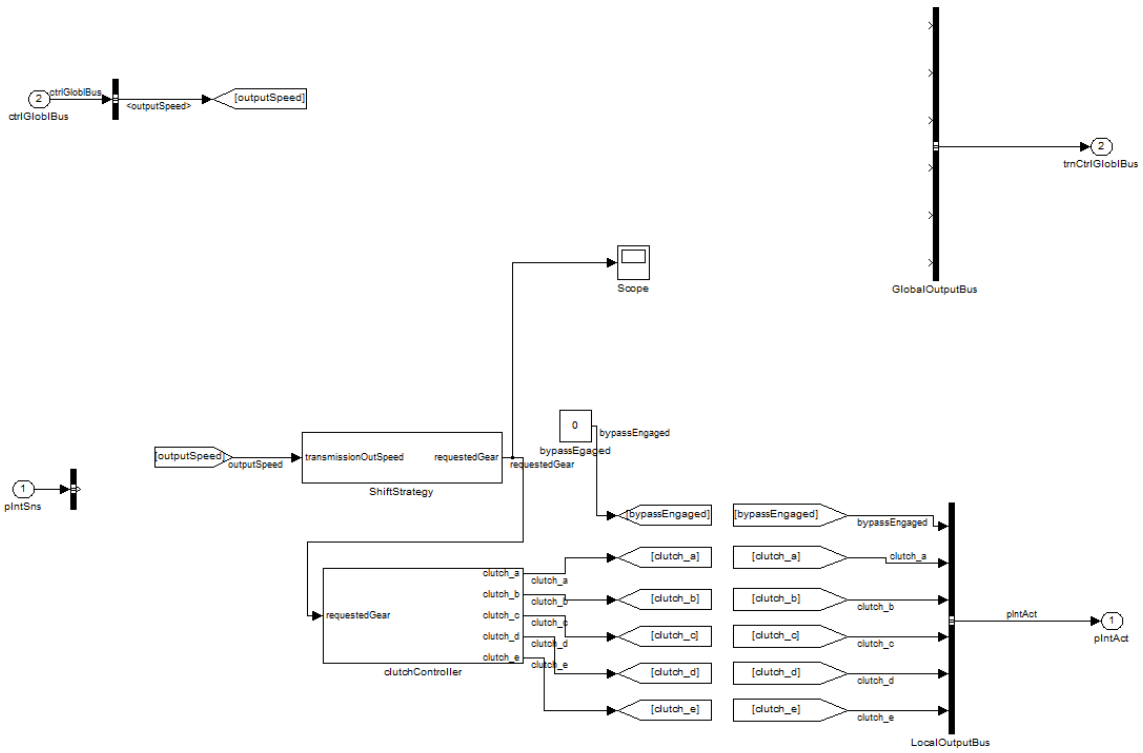


Figure 50. Fully-defined internals of the TransmissionController_C100 subsystem

5.3.3.5 Replacing Simulink Controllers

Once the controllers have been fully defined, they need to be inserted into the overall Simulink VMA model. This top level model was created by hand because the organizational framework of the Simulink VMA remains the same, and it only needs to be updated with the latest controllers and Modelica model to be complete and validated. In the VMA_Simulink_C100.mdl, the modeler needs to navigate to the Vehicle Subsystem Controller view and replace each subsystem with the updated version, Figure 51.

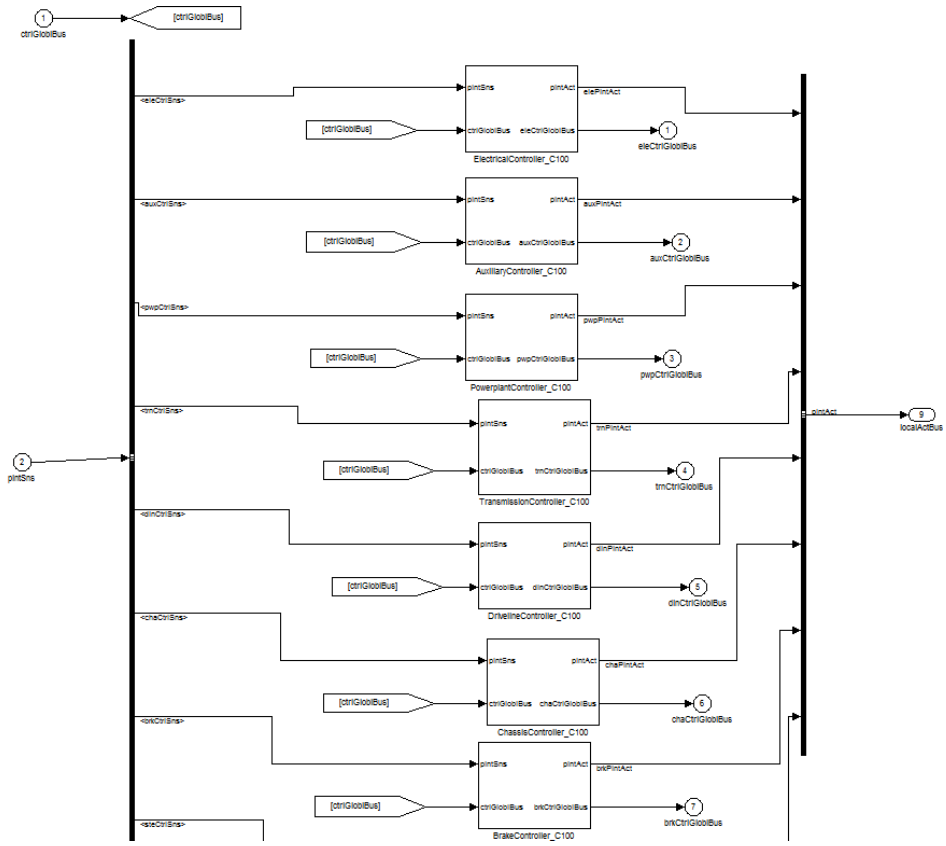


Figure 51. Replacing controller subsystems with updated controllers

5.3.4 Connecting the Plant and Controller Models in Simulink

Finally, the last step in this process is to import the fully-defined Modelica model into Simulink as an S-Function and connect the physical plant models to their logical controller counterparts, using a harness, shown in Figure 52, which occurs during 18 of Figure 33. This integration of models is accomplished in Simulink through the use of a "Dymola Block." By double clicking on the Dymola block, the modeler can specify the Dymola model he wishes to analyze, which in this case is the overall vehicle model in Dymola, as seen in Figure 46. Once the Modelica model has been imported, its scalar signals are grouped into appropriate buses. One shortcoming of this approach is that the

Control Bus blocks in Modelica are not fully supported by Dymola in the translation to Simulink. In Modelica, these buses contain multiple signals, and it would be ideal if these buses were mapped to Simulink as buses with multiple signals. However, the signals map from Modelica to Simulink as scalars, and every scalar signal can be seen on the Dymola Block, as can be seen in Figure 52. To solve this problem, we have created a wrapper around the Dymola Block, which again groups the scalar signals into buses. The hope is that in the near future the importation of Control Buses from Modelica into Simulink is supported. In Figure 52, only the necessary ports are connected to the Dymola block.

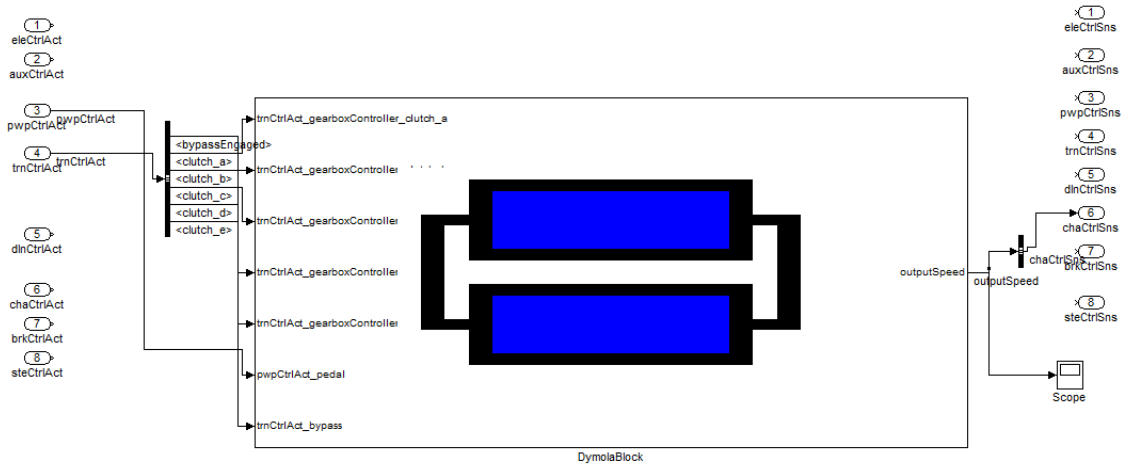


Figure 52. Connecting the physical Modelica model to the logical Simulink model

After all of the signals have been connected, the entire vehicle model can be run in Simulink. The results of the analysis test from 0 to 100 kph for the Vehicle_C100 model are given in Figure 53. In the process flow of Figure 33, the Simulation Analyst would take Figure 53 and determine whether the output is desirable. If it is, then the process flow continues to 23 of Figure 33, and if not, then the iteration process continues.

It is not the goal of this research to generate Figure 53, but to support the entire vehicle modeling process to be able to generate Figure 53.

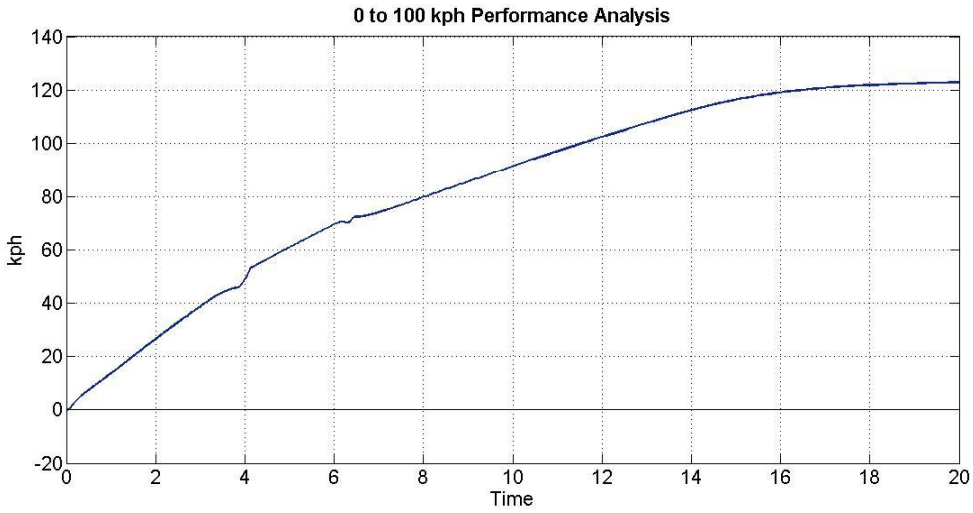


Figure 53. Output speed of vehicle in the 0 to 100 kph test

5.4 Summary

This chapter describes how an engineer would approach implementing the VMAF by reusing the generic VMA to specialize it for a specific vehicle and then analyzing the results from the vehicle models for a variety of different analysis tests. The example used throughout this chapter is the C100 vehicle and the example analysis is the 0 to 100 kph performance test. The process flow in Figure 33 is followed in creating and analyzing a new vehicle. This procedure in creating the C100 vehicle, generating, and simulating the analysis is aided by several plug-ins, which have been developed to reduce the modeling time and the potential for user error. If changes are made to the C100 model in SysML, the analysis architectures can be easily regenerated and updated.

CHAPTER 6:

CONCLUSIONS AND FUTURE WORK

In this thesis, an approach for supporting Model-Based Systems Engineering of a multidisciplinary vehicle system is introduced. The approach starts from a formal SysML-based specification of a generic vehicle architecture that can be specialized for specific vehicle programs. From the SysML model, the systems engineer can also generate corresponding vehicle analysis model templates by automatically transforming part of the SysML model into integrated analysis models in Modelica and Simulink. This last chapter reviews the motivating questions from Chapter 1 and discusses the contributions of this approach to the MBSE field, the limitations of this approach, and the future work that would be desirable to enhance the VAMF. Finally, the thesis is brought to a close with some concluding comments.

6.1 Reviewing the Motivating Questions

In Chapter 1, there is a discussion of the problem at hand: vehicle systems are growing in complexity, which in turn gives rise to a need to formally model these complex systems accurately and consistently. There are many aspects that must go into representing modern vehicle systems due to their span over multidisciplinary domains. The approach taken in this research is to create a vehicle system architecture that encompasses the entire system including structure and analysis. The overall motivating question for this thesis is repeated below.

Motivating Question:

How can engineers use SysML to model complex vehicle system architectures and the corresponding multidisciplinary analyses effectively, consistently, and accurately?

In response to the first part of this question, the Vehicle Architecture Modeling Framework was devised in order to formally represent the architecture of modern, complex vehicle systems. Because this question contains multiple important parts, it was divided into three sub-questions, as given below.

Question 1:

How can engineers effectively model vehicle architectures in SysML representing the energy and signal flows between all aspects of the vehicle?

As part of the VAMF, the Vehicle Model Architecture is defined in SysML using the necessary modeling constructs, as described in Chapter 4. The VMA definition contains the vehicle's nine essential subsystems, and diagrams depict how the subsystems interact through physical energy flow ports and logical signal flow ports. By formally modeling the VMA in SysML, the consistency is maintained and changes to the structure are cascaded throughout the model.

As to the second part of the motivating question, Question 2 is restated below.

Question 2:

How can engineers create analysis architectures that can be reused for different vehicles while remaining consistent with the base vehicle architecture?

It would be ideal to have all of the different analyses that are necessary to test vehicle system alternatives and can be reused for different vehicle system alternatives within the VMA SysML model. The solution to Question 2, as presented in this thesis, is to employ the *SysML4Modelica* profile and create a vehicle analysis template for the entire vehicle. Then, each new analysis can depend upon the necessary parts of the entire vehicle template. Dependency matrices are employed to aid in the new analysis definition.

In order to analyze the VAMF alternatives, it is necessary to integrate analysis languages with SysML because SysML does not inherently contain the ability to internally analyze systems. This shortcoming leads to the rise of Question 3.

Question 3:

How can Vehicle Model Architectures be used to generate analysis templates in Simulink and Modelica?

Previous works have explored exporting SysML models to an analysis and simulation language in order to analyze multiple system alternatives. This thesis proposes integrating both Modelica and Simulink with SysML to more accurately analyze each alternative. Modelica is implemented for its abilities in analyzing physical energy flows, and Simulink, on the other hand, is employed for its prowess with logical control flows. Combined, the two languages provide a more accurate representation of the analyses for each alternative. The physical plant side of the Vehicle Model Architecture can be translated to Modelica to model the physical energy flows between the subsystems. Then the logical control side of the VMA can be translated to Simulink. Through the use of plug-ins, this translation process from SysML is supported for the

automatic generation of Modelica and Simulink templates. In this manner, changes in the SysML model can be easily cascaded through to the analysis languages. Also, by reusing the analysis templates, multiple vehicle architectures can be created based off of the VMA to analyze different vehicle alternatives.

6.2 Contributions

The research described in this thesis has produced several contributions to complex system modeling and the modeling process itself. First, the research contributions consist of creating the entire Vehicle Architecture Modeling Framework that formally models all of the essential aspects of modern vehicles. The generic Vehicle Model Architecture, which models all of the vehicle's essential subsystems, physical and control interfaces, and the interaction between subsystems in a standardized fashion, is defined in SysML. This architecture can be redefined and specialized for a specific vehicle including adding specific messages and signals to the specialized vehicle for defining the global signals and local sensors and actuators. In order to analyze the specialized vehicle, common vehicle simulation analysis templates have been created, and through developed tools, these templates can be also specialized for a specific vehicle.

Another academic contribution is integrating the specialized vehicle and analysis architecture models in SysML with analysis simulation models in the languages Modelica and Simulink through the support of developed tools. In this manner, the generated interface model templates can easily be filled by domain experts for each of the subsystems, and such that they can subsequently be seamlessly combined into a complete

vehicle-level simulation model. The whole process of creating the VAMF and validating different vehicle alternatives enhances the current Model-Based Systems Engineering practices by enabling more efficient consistency management, reducing modeling time, and decreasing the potential for human modeling error.

This research has also contributed to the MBSE practices by developing a process flow for system engineers, depicted in Figure 33, which shows how the VAMF can be used to create and analyze a new vehicle alternative. Figure 7 and Figure 8 show how the entire vehicle modeling process flows from the initial decision to create a new vehicle to the final validation of the model. The VAMF fits into the red box depicted in Figure 8, which shows how the architecture models are defined, specialized, and translated to analysis models to be validated.

6.3 Limitations and Future Work

Even though the VAMF approach can greatly reduce the necessary modeling time, there still are some limitations of this approach. The limitations are given below, and for each limitation, suggestions for future work are also provided.

- **Program specific architecture:** In the VAMF approach, a generic vehicle architecture is defined. From the generic structure, specific vehicles can be created through redefining and specializing the base VMA. This modeling mechanism is currently quite cumbersome. It is suggested that a different modeling approach should potentially be adopted, such as including the variant modeling constructs being defined in the upcoming SysML 1.4 release, or developing better tool support to make the specialization and redefinition process easier.

- **Generating signal stubs in Modelica:** The local plant signals are not currently supported by the SysML to Modelica conversion. Stubs for these signals should be added automatically as "RealInputs" or "RealOutputs," but currently these signals are manually added to the Modelica model. Also, it would be ideal if the SysML to Modelica transformation took into account the difference between signals that are sensors and actuators versus signals that are only monitored for analysis. Additionally, it would be desirable if Modelica "Sensors" could be added automatically to the Modelica models, such as adding a torque sensor which connects to the "RealOutput" for a torque signal stub.
- **Active consistency checking:** The VAMF approach does greatly increase the ability to maintain the consistency in complex models through the developed plug-ins and formal modeling. However, if the modeler realizes that a local signal has been left out of the translated Simulink model, the modeler must go back to SysML and add the signal to its proper block and regenerate the Simulink model in order to remain consistent. It would be convenient if all three languages, SysML, Modelica, and Simulink could actively interact and changes in one program would be updated by the other two, but this is a lofty goal.
- **Better support for Modelica-Simulink interface:** The Control Bus blocks in Modelica are not fully supported in the Dymola Block importer in Simulink. It would be ideal if the local signals could be put onto these Control Buses in Modelica, and then these Control Buses would appear on the border of the Dymola Block in Simulink as buses. However, at this point each signal on the bus appears as a scalar

signal. To adapt to this in the short term, Simulink Bus Selector and Bus Creator blocks are used to rewrap the local Modelica signals into buses.

- **Analysis Configuration matrix support:** The SysML Analysis Architectures model would also be easier to use if functionality was added to MagicDraw's dependency matrix, as depicted in Figure 30. Currently, the dependencies must be added manually to the matrix by the modeler. The modeler must either create a new BDD and add the new analysis and connect dependencies to the specific parts of the generic analysis by hand, or add the dependencies through the specification window. Either way, this task is cumbersome. It would be ideal if the modeler could just create a new analysis and then double-click on the specific subsystems, ports, and parts in the analysis matrix that correspond to the new analysis to turn on or off the dependencies.
- **Additional diagrams in the VAMF:** Finally, the VAMF would be improved if the future work includes adding representations of the VAMF Use Cases to SysML through Use Case diagrams. These diagrams would contain the different vehicle operating scenarios and help to better define the entire vehicle system. Also, a Vehicle Program Analysis Plan should be created, which includes the domain model requirements and the list of desired analyses that need to be completed by analysts. Adding these additional items to the current Vehicle Architecture Modeling Framework would greatly benefit the approach and better define the vehicle system.

6.4 Summary

As engineering systems are becoming more complex, engineers must be able to effectively manage a considerable amount of design information and knowledge.

Previously, this information and knowledge was captured in a document-based fashion [9], but continuously updating these documents leads to human error, and this process is becoming antiquated with the advent of Model-Based Systems Engineering. This thesis presents an alternative method to document-based engineering, namely using MBSE, which gives designers the capability to actively update models and propagate these changes throughout system models.

To advance the MBSE process, this thesis presents a method to create and analyze vehicle system architectures that encompass multidisciplinary domains. SysML is used to formally model the vehicle system architecture through the approach of the Vehicle Architecture Modeling Framework. Because SysML does not inherently contain the ability to analyze various alternatives, the SysML models are translated into Modelica and Simulink to be analyzed. This process is aided by the use of several developed plug-ins for MagicDraw that aid modelers in creating a specific vehicle and translating the analysis architectures to Modelica and Simulink. These plug-ins reduce the modeling time and the potential for human translation error. They also give the ability to automatically cascade changes made in the SysML model to the Modelica and Simulink analysis models.

Hopefully, the work in this thesis not only gives engineers the ability to model and analyze vehicle architectures, but also gives direction to future researchers who wish to further multiple model integration and the overall MBSE process. MBSE in general is a relatively new concept and it is gaining momentum in the industrial world as companies understand the benefits MBSE provides over current modeling practices.

APPENDIX A:

STEP BY STEP GUIDE TO CREATING SPECIALIZED VEHICLE

1. Copy-paste "Vehicle Specialization" package at the top level of the SysML project and rename it to specify the desired Program ID (ex: Vehicle Specialization: Vehicle_C100).

2. Open each sub-package of the newly created specialization package and rename all blocks from "_ProgramID", Figure A1(a), to the desired ProgramID (ex: "_C100"), Figure A1(b).

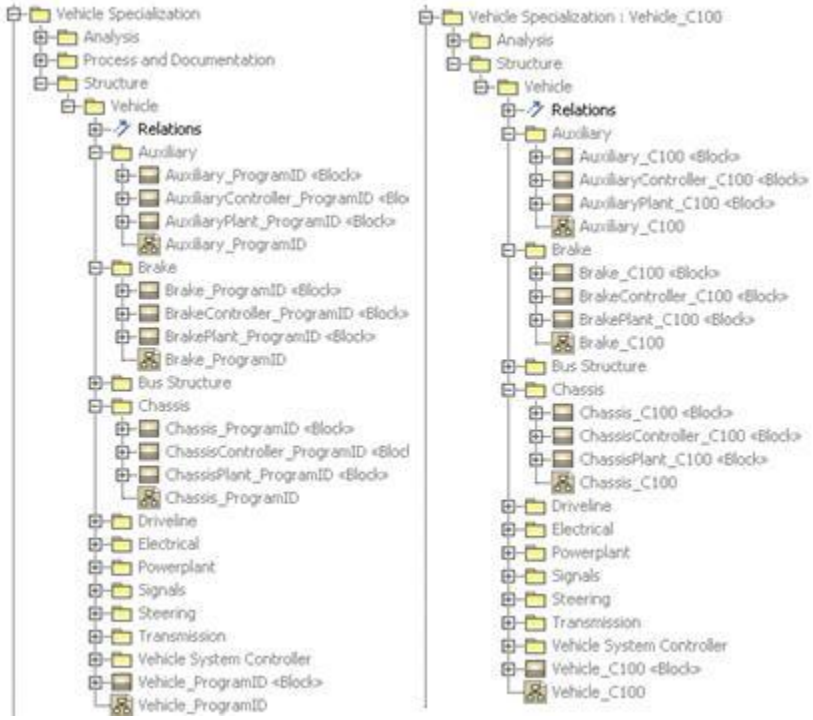


Figure A 1. (a) Vehicle Specialization package with generic "_ProgramID" block names. (b) C100 Specific Vehicle Package with specific "_C100" block names.

3. Local Signals Excel Setup

a. Open the Local Signals Excel file

b. Make desired changes to the Local Signals file

c. For Excel 2003, on the Data menu, click on XML and then Export to export the Local Signals as an XML file

4. Import Global Signals

a. Right click on the newly created Vehicle_C100 (or whichever ProgramID is specified).

b. Run the plug-in named “Import Signals” and choose the “Import Global Signals” option, Figure A 2.

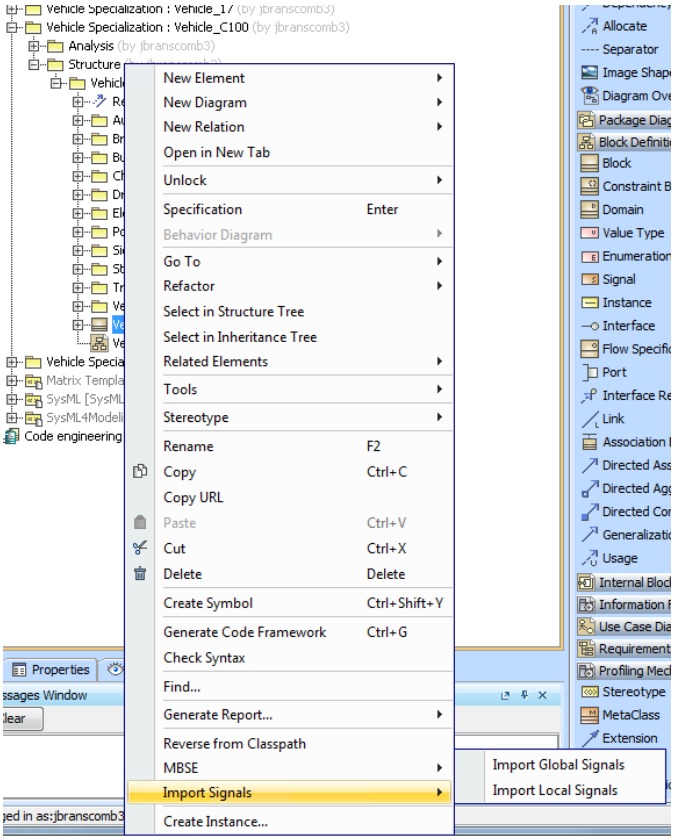


Figure A 2. Import Signals Plug-in

c. Choose the XML file for the C100 global signals from the file browser. The global signals should appear in the Vehicle Specialization: Vehicle_C100::Structure::Vehicle::Signals package as flow properties of

each of the subsystems global signals blocks (ex. PwpGloblMs_C100 should have flow properties for the Powerplant global signals)[1].

5. Import Local Signals

a. Right click on the newly created Vehicle_C100 (or whichever ProgramID is specified).

b. Run the plug-in named “Import Signals” and choose the “Import Local Signals” option.

c. Choose the XML file for the C100 global signals from the file browser. The local signals should appear in the Vehicle Specialization: Vehicle_C100::Structure::Vehicle::Signals package as flow properties of each of the subsystems local signals blocks (ex. TrnActMs_C100 should have flow properties for the Transmission local actuator signals, Figure A 4).

[1] The Global Signals XML file came directly from the CAN DBC export

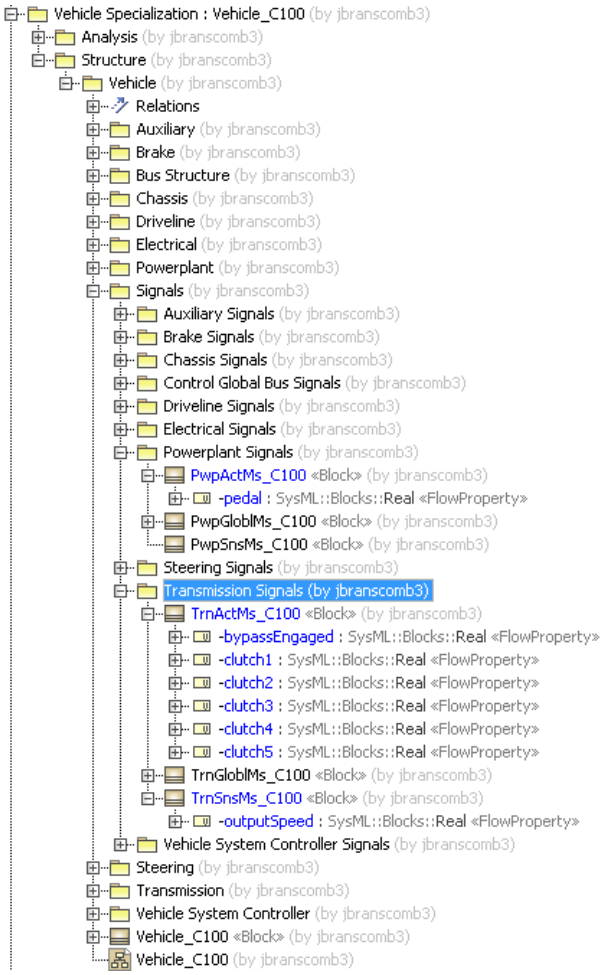


Figure A 3. The Imported Local Signals augment the C100 Vehicle Model

6. Run the SysML Transformations plug-in to setup Modelica transformation.

   a. Right click on the same Vehicle_C100 block as in Step 3. Run the plug-in named "SysML Transformations" and choose the "Generate Modelica Analysis", Figure A 4.

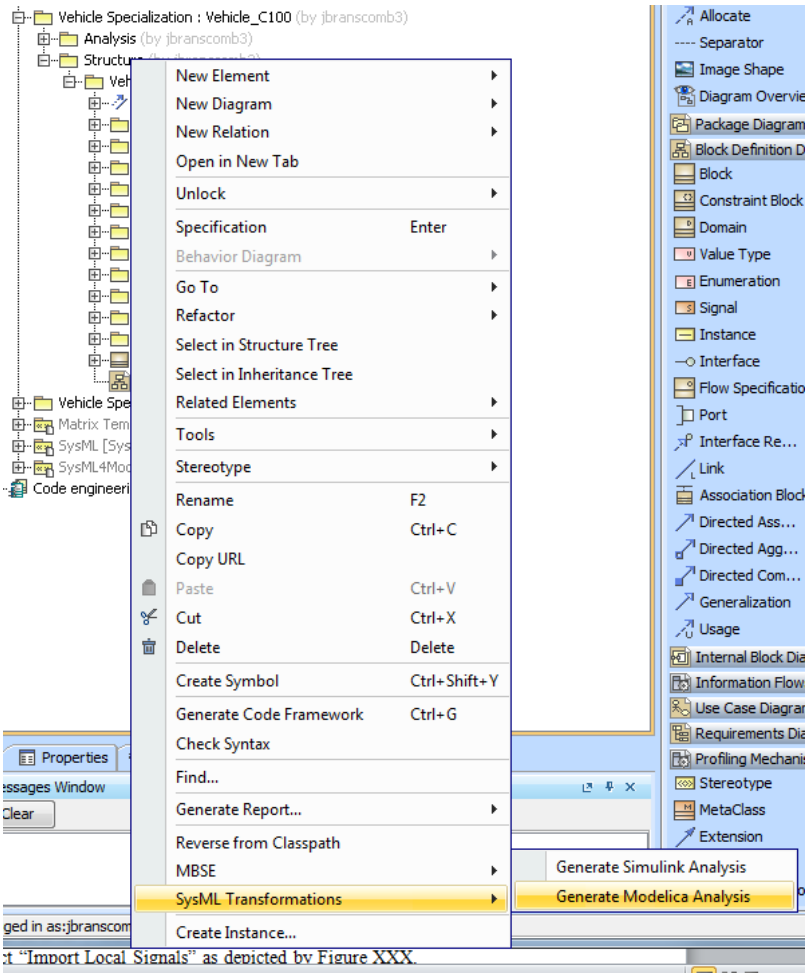


Figure A 4. Running the Modelica Analysis Plug-in

b. Specify the desired analysis from the drop down menu, Figure A 5.

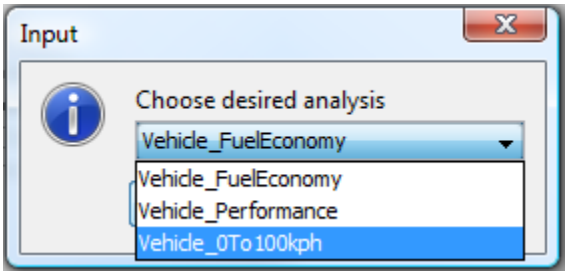


Figure A 5. Choosing the desired analysis

c. Specify the specific ProgramID (ex: C100), Figure A 6.

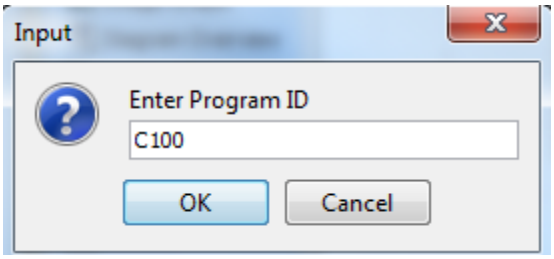


Figure A 6. Choosing the desired ProgramID

d. The analysis should appear in the Analysis package under the Vehicle Specialization: Vehicle_C100 package, Figure A 7.

Figure A 7. Performance Analysis for the C100 vehicle

7. Run the plug-in to generate the Modelica code.

a. Open the Vehicle Specialization: Vehicle_C100::Analysis package and expand the analysis package created in Step 4.

b. Right click on the Vehicle_0To100kph_C100 «ModelicaPackage» stereotyped block, and run the plug-in "SysML to Modelica" and choose the option "Generate Modelica," Figure A 8.

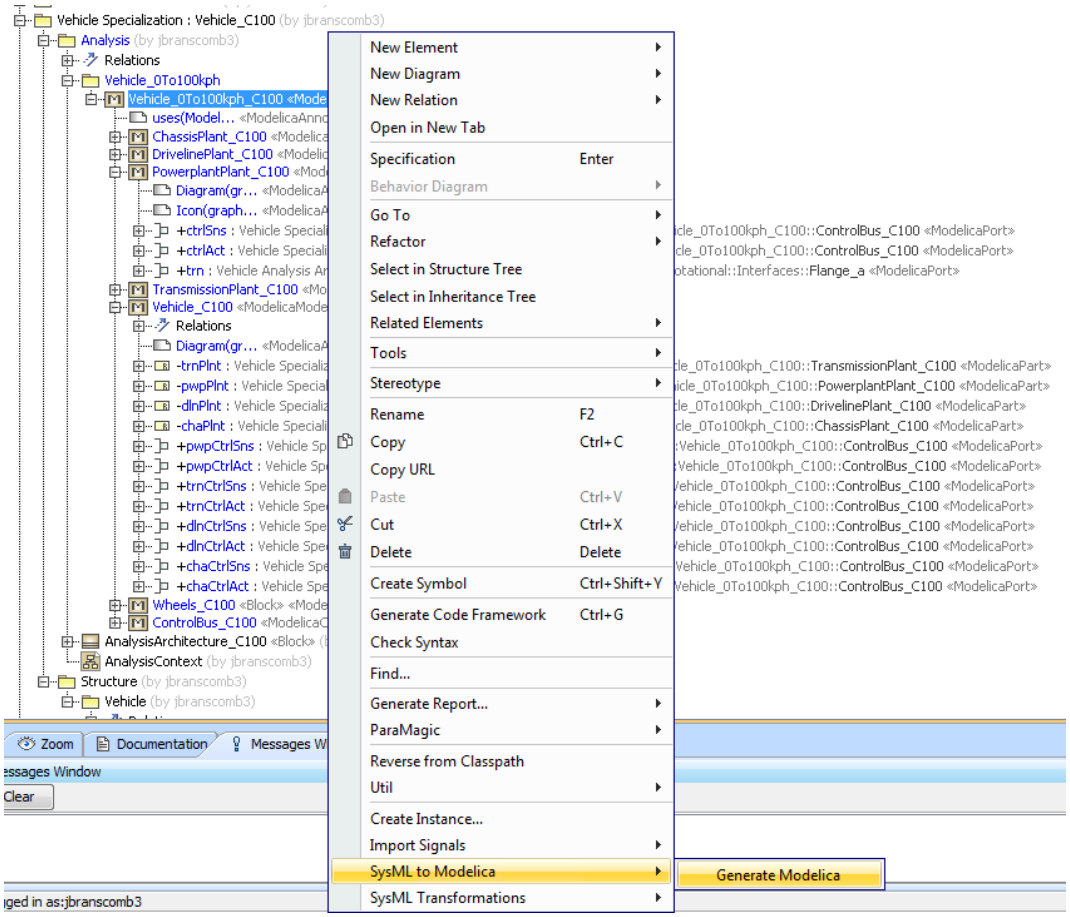


Figure A 8. Generating the file containing the SysML to Modelica translation

c. The plug-in creates a ".mo" file which can be opened in Dymola.

8. Open the newly created ".mo" file in Dymola and fill in the subsystems with Modelica models. There are 2 ways to fully define the vehicle plant templates. The first method is summarized in a-f, and the second method is summarized in g.

a. Figure A 9 shows an example of the template of a subsystem generated from the SysML model[2].

b. Navigate to the Vehicle_C100 system diagram, Figure A 10. Ensure that the desired existing Modelica models are loaded in the Dymola window

[2] Eventually need to add scalar signal inputs in the translation from SysML to Modelica

c. Open up libraries of Modelica models with existing Plant content in Dymola

d. Right click on one of the subsystem block, e.g. Powerplant_C100 and select “Change class.”

e. Choose “Browse” and find the corresponding existing Powerplant model

f. After replacing all of the subsystems, model should look like Figure A 11[3].

g. Alternatively, if there are no pre-existing plant models, the domain expert must create the models using the Modelica model library or manually using the Modelica language.

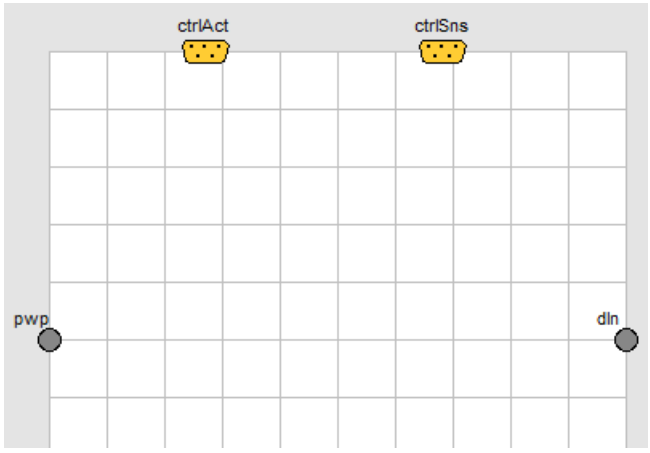


Figure A 9. TransmissionPlant_C100 subsystem showing its local Control

[3] It may be necessary to add some Modelica blocks to model the outputs because the buses aren’t yet fully supported in the Modelica to Simulink conversion. In this example, the realPassThrough block with the outputSpeed Real Output were added

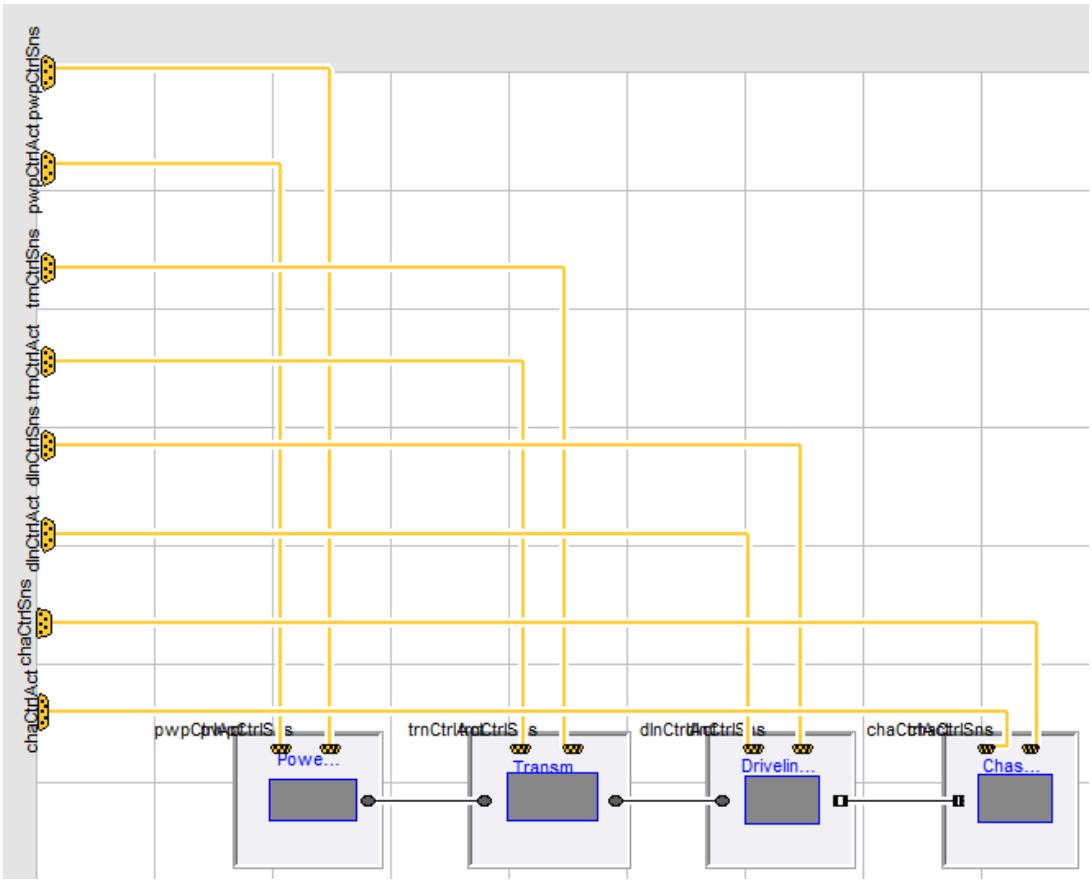


Figure A 10. Vehicle_C100 system

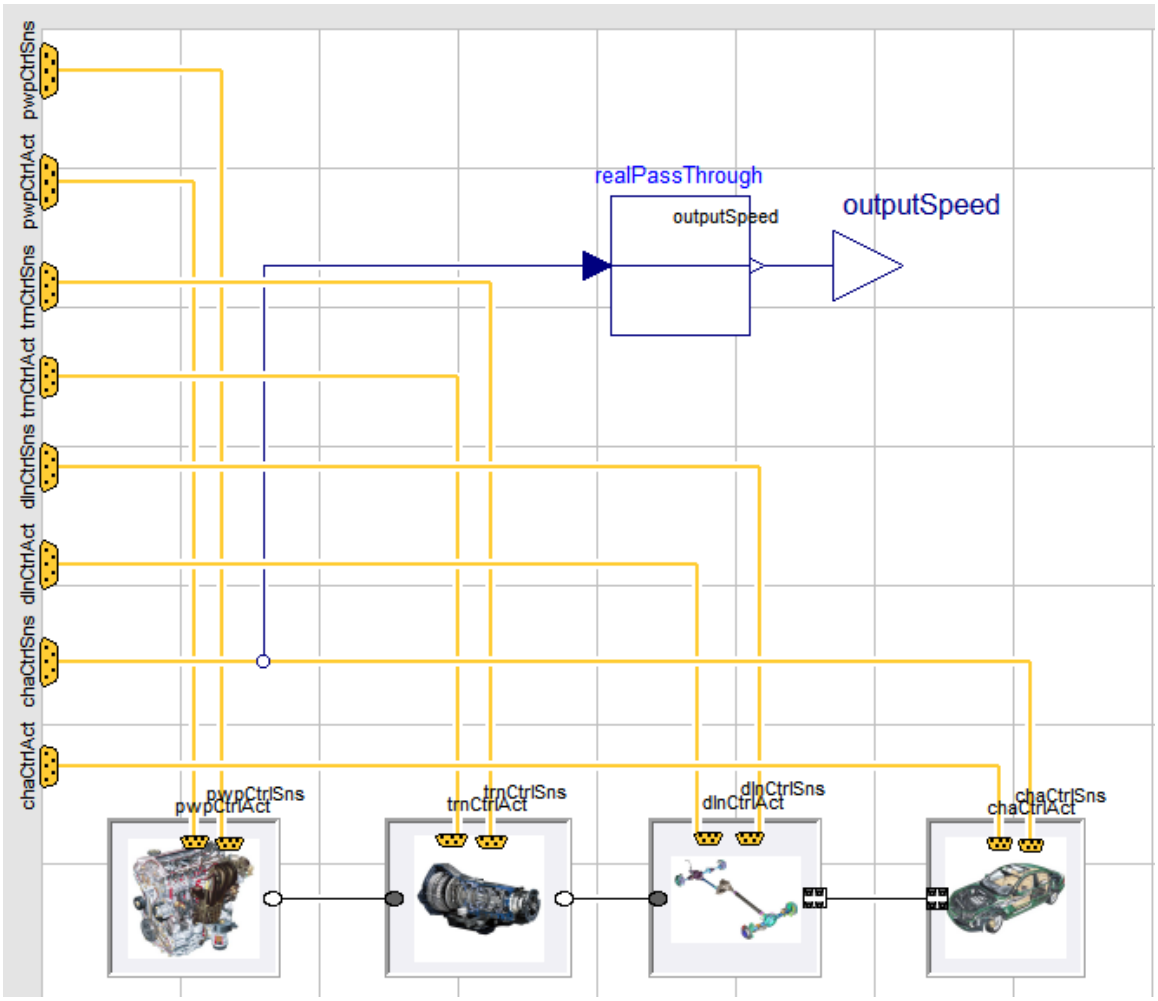


Figure A 11. Vehicle_C100 after replacing subsystems with existing models

9. Return to SysML, run SysML to Simulink plug-in.

    a. Right click on the Vehicle_C100 block within the Vehicle Specialization: Vehicle_C100::Structure::Vehicle package.

    b. Run the "SysML Transformations" plug-in again, but choose the "Generate Simulink Analysis" option, Figure A 12.

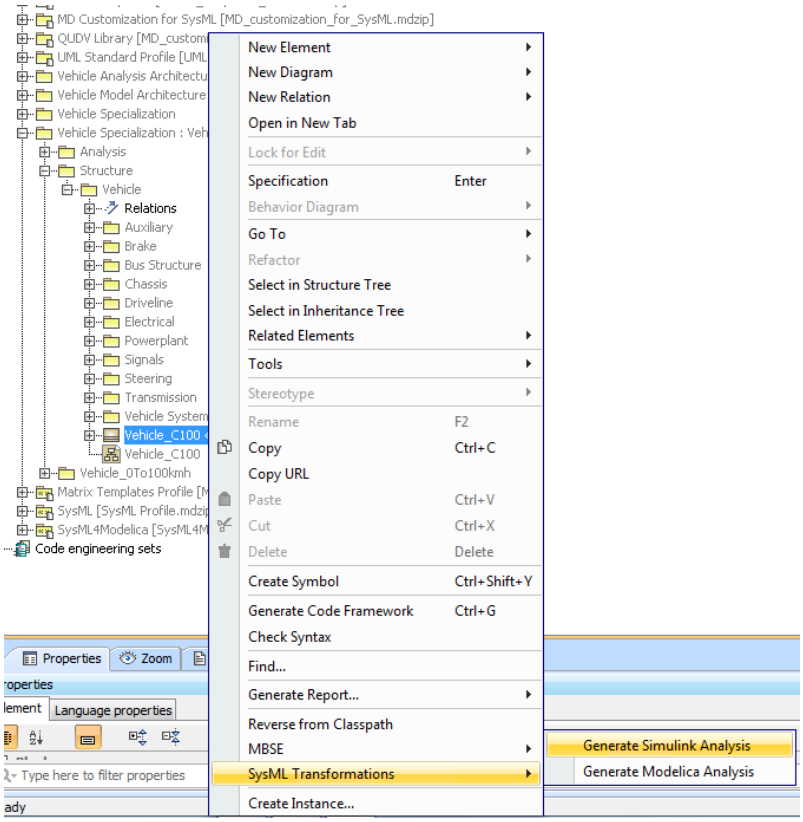


Figure A 12. Initiating SysML to Simulink Transformation

c. An ".m" file is created, which can be opened in a Matlab editor.

10. Open the newly created ".m" file in a Matlab editor and run the script.

a. Simulink models for each subsystem are created in the present directory, Figure A 13.

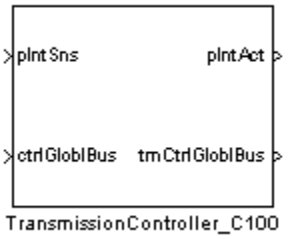


Figure A 13. Example of the TransmissionController_C100 Subsystem

b. The models are just the framework for the controller, so the model must be supplemented with fully-defined controller models, Figure A 14.

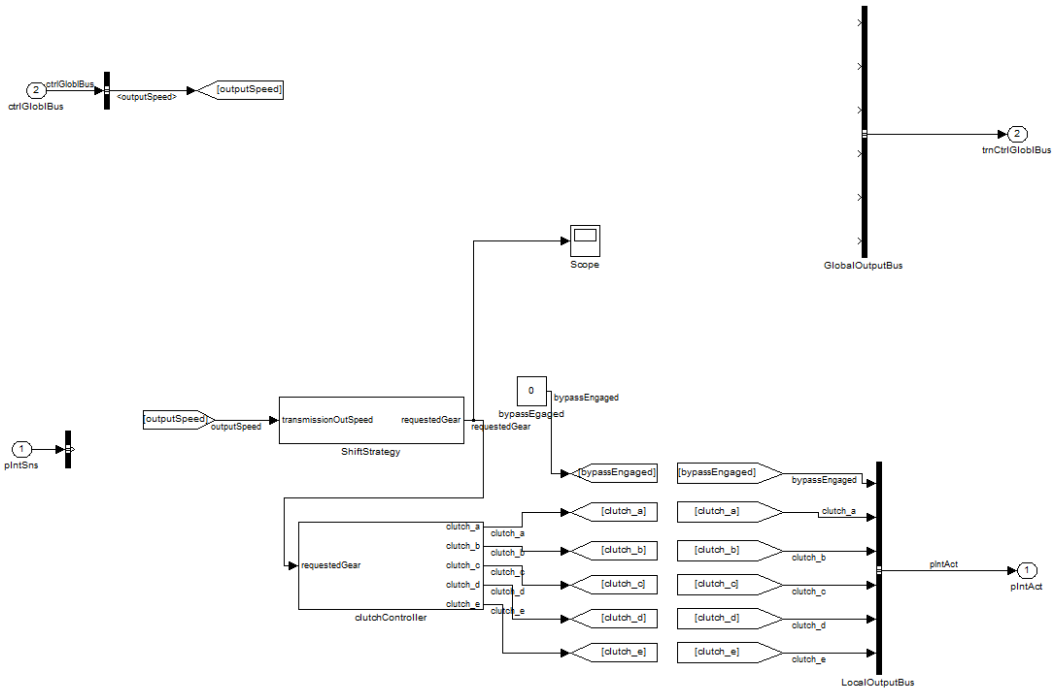


Figure A 14. Fully-defined internals of the TransmissionController_C100 subsystem

11. Replace controllers in the VMA_Simulink model

    a. Navigate to the VMA_Simulink::VehSys::VehSubSysCtrl window and replace the necessary controller subsystems, Figure A 15.

    b. Re-connect any connections that get deleted in this process

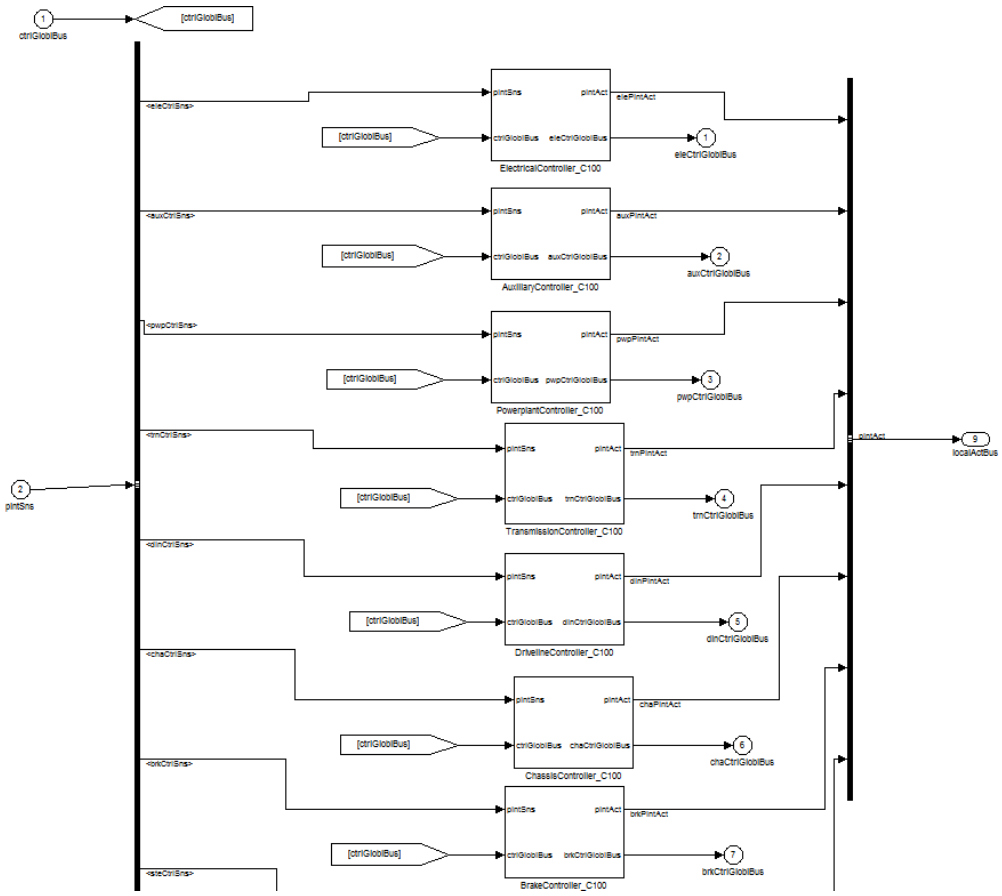


Figure A 15. Controller view of Simulink model

12. Replace Plant model in Simulink

a. Navigate to the VMA_Simulink_C100::VehSys:: VehSubSysPlnt::VehiclePlant_C100.

b. Double click on the Dymola Block and choose the overall vehicle model in Dymola (Figure A 11) and compile the model.

c. Ensure that the scalar signals on the boarder of the Dymola Block to group the signals to Simulink buses are connected, Figure A 16.

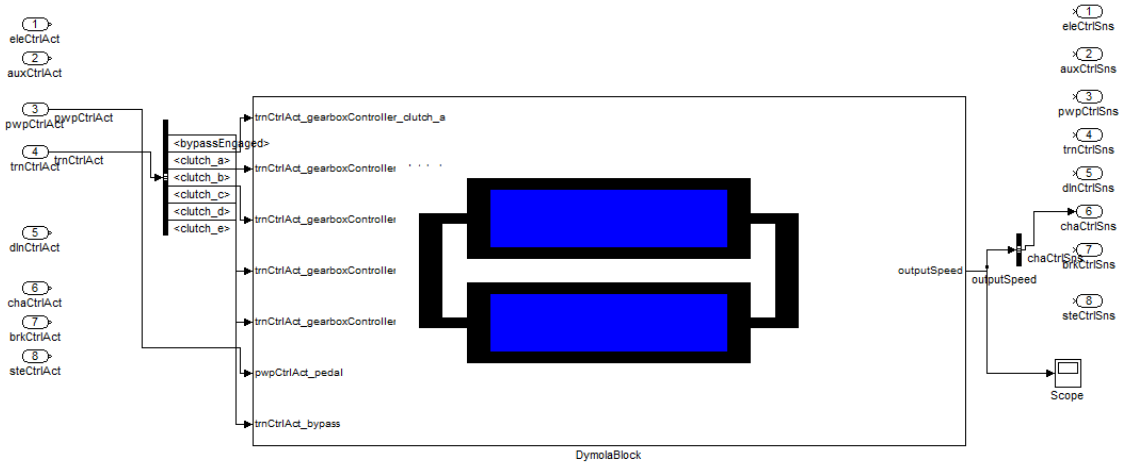


Figure A 16. Connecting the physical Modelica model to the logical Simulink model

13. Run the Simulation in Simulink and analyze results.

    a. Navigate back to the overall VMA_Simulink_C100 view and open the scope by double clicking on it.

    b. Press play and analyze results, Figure A 17.

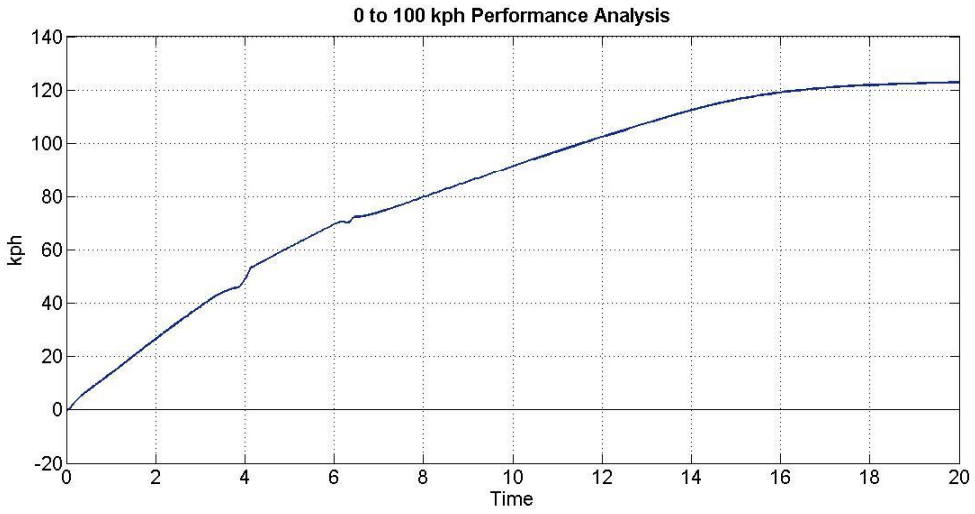


Figure A 17. Output Speed of Vehicle_C100

APPENDIX B:

MATLAB AND DYMOLA: SET UP TO RUN

Matlab Setup

1. Install Matlab 2010a Win32 version (Matlab 2007b has formatting issues, but other Matlab versions might work too)
    a. 64-bit Matlab is not compatible with Dymola 7.4
2. Run "mex –setup" in the Matlab command prompt. Select same compiler as Dymola.
3. Add Dymola Folders to Matlab path
    a. C:\Program Files (x86)\Dymola 7.4\Mfiles
    b. C:\Program Files (x86)\Dymola 7.4\Mfiles\dymtools
    c. C:\Program Files (x86)\Dymola 7.4\Mfiles\tmf
    d. C:\Program Files (x86)\Dymola 7.4\Mfiles\traj

Dymola Setup

1. This research uses Dymola 7.4 with the Modelica 3.2 library.

To Simulink/Dymola Run Model

1. Open the Dymola model containing the specialized vehicle analysis model (C100 0 to 100 kph performance analysis in this case)

2. Navigate to Vehicle_C100 model, Figure B 1

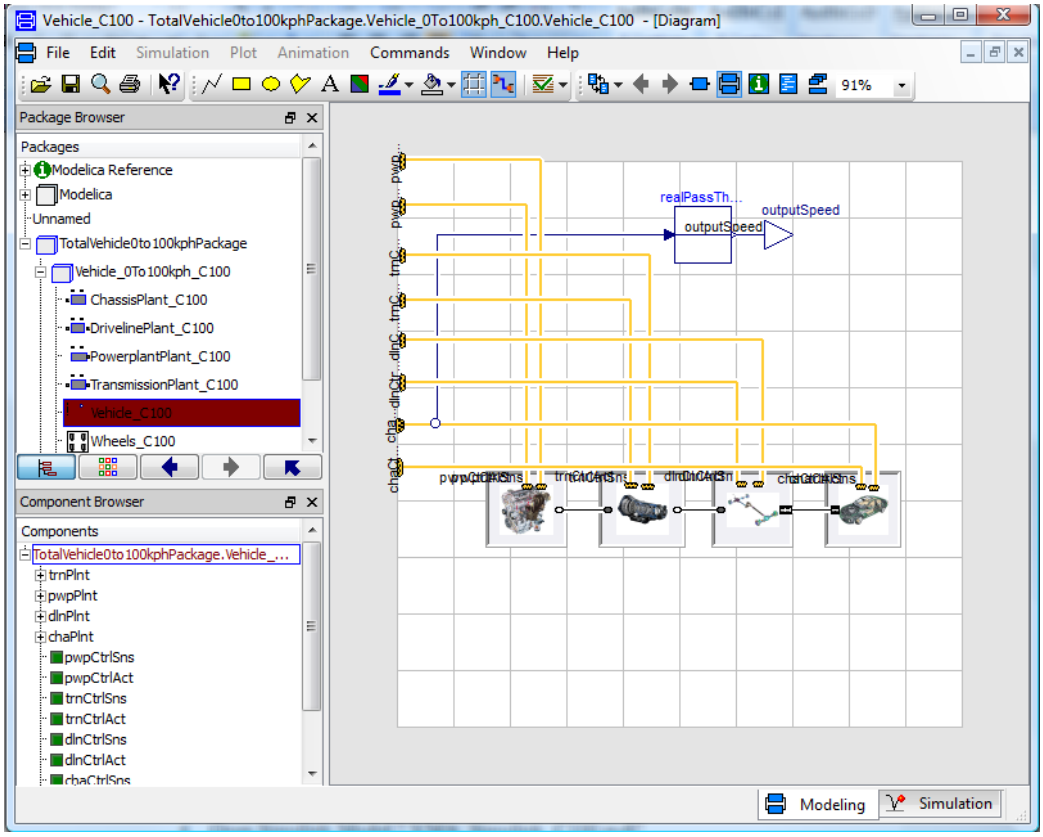


Figure B 1. Dymola model under analysis

3. Open Simulink model containing the VAMF, “VMA_Simulink_C100.mdl" for this example
4. Go to Dymola Block and double click on it. The Dymola parameter window opens, Figure B 2
5. Click on "Select from Dymola" and "Compile model" buttons

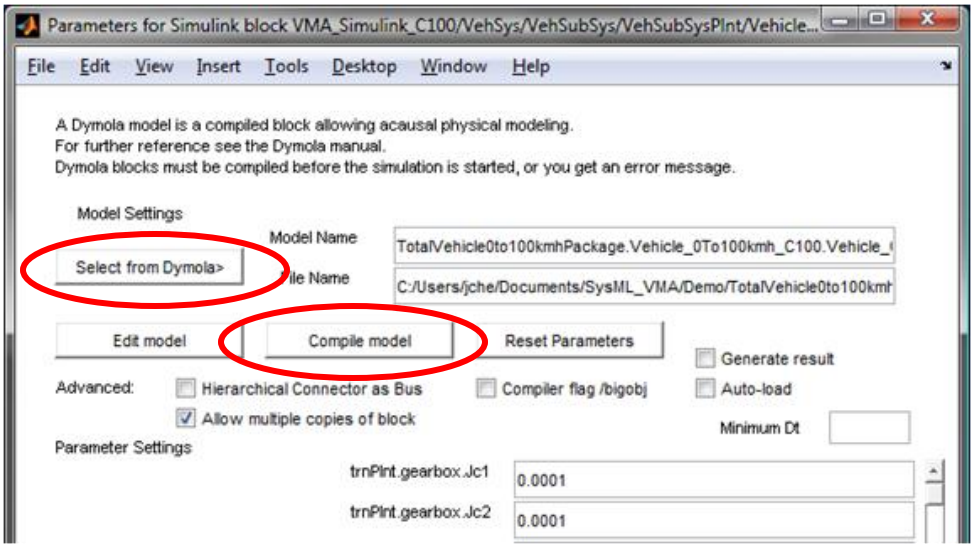


Figure B 2. Dymola parameter window in Simulink

6. Run the Simulink Model

APPENDIX C:

DOCBOOK USER'S GUIDE

1. Decide where the documentation package should reside in the SysML model
    a. Create a SysML package called "Documentation" in that location
    b. Right-click on the Documentation package and select the plug-in "MBSE" and choose the option "SE2: create Book", Figure C 1

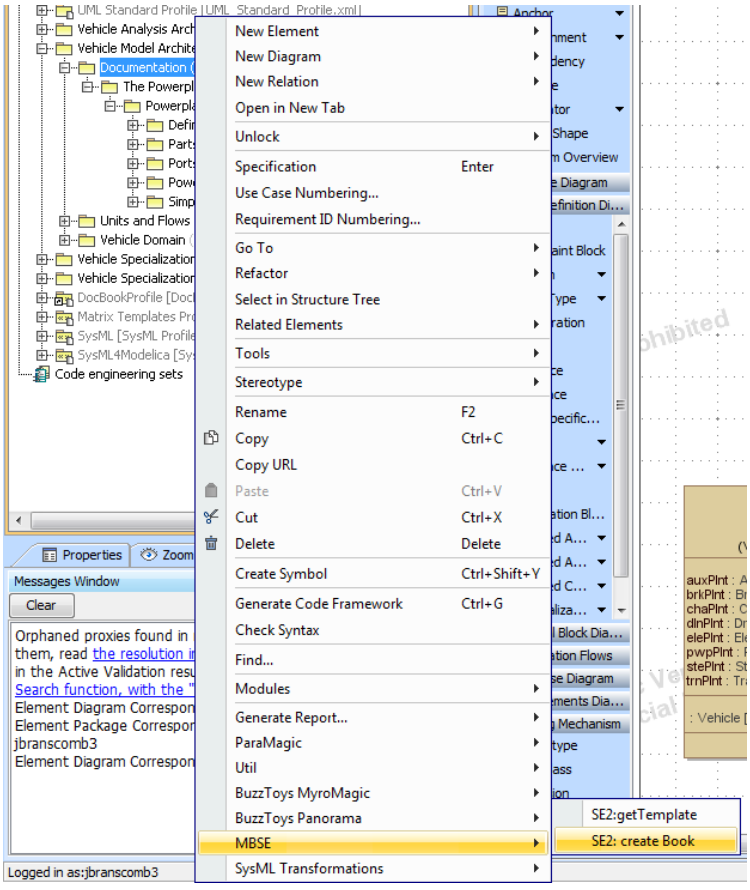


Figure C 1. Creating a DocBook "Book"

    c. Name the book, and enter desired information in the GUI, Figure C2.

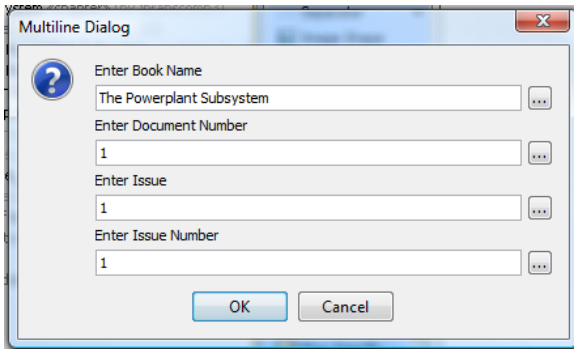


Figure C2. Entering Book information

2. Right-click on the newly created package stereotyped by «book»

   a. Again navigate to the "MBSE" plug-in and choose one of the options to create the beginning of the book: "SE2: create Part", "SE2: create Chapter" etc, Figure C 3.

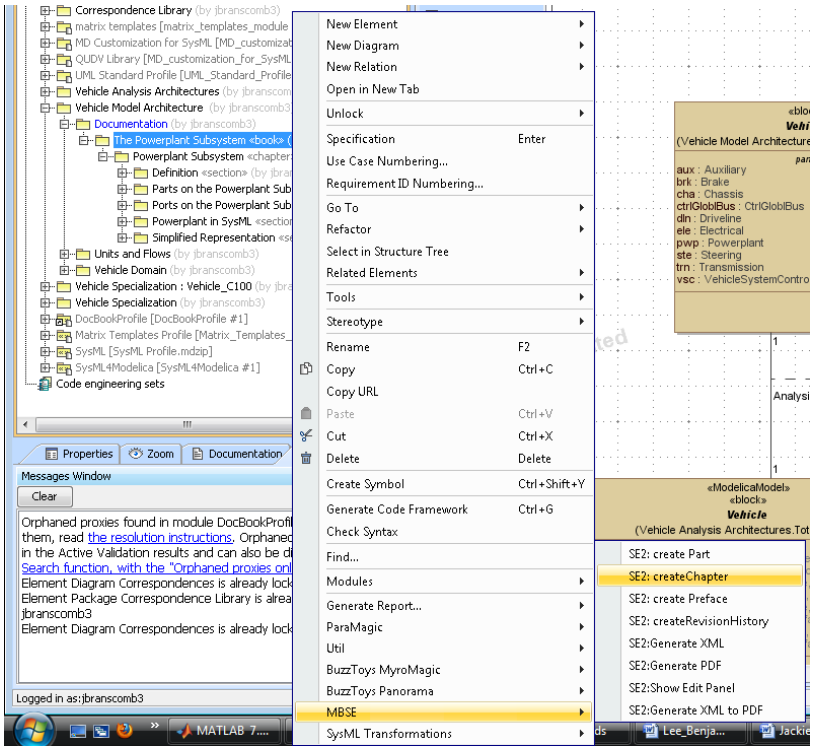


Figure C 3. Creating a Chapter, Part, or Preface

   b. Name the chapter, part, or preface and repeat this process to generate the framework for the book

3. Instead of creating the book through by right-clicking on the stereotyped elements, modelers can also create the book in the Edit Panel.
    a. Right-click on the «book» and select “MBSE” and choose the option “SE2: show Edit Panel”
    b. The Edit Panel appears on the right side of the SysML diagram frame, Figure C 4.

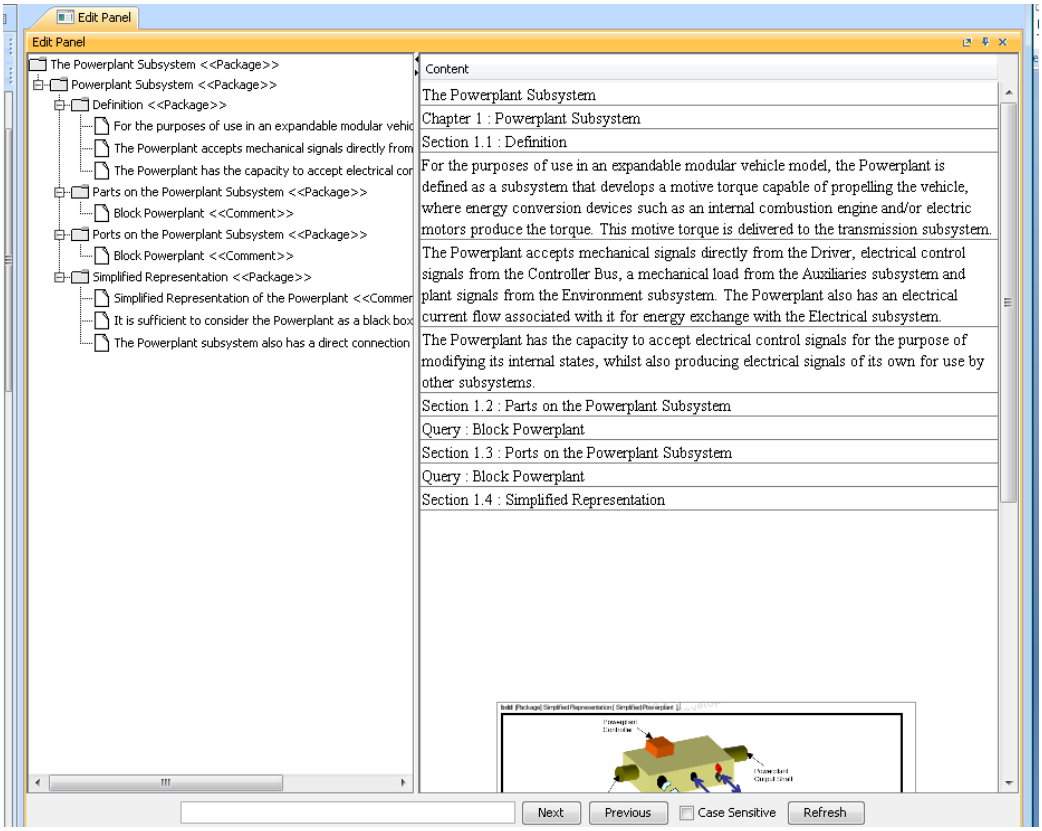


Figure C 4. DocBook Edit Panel

    c. From the Edit Panel, modelers can right-click on any stereotyped element and create portions of the «book», Figure C 5.

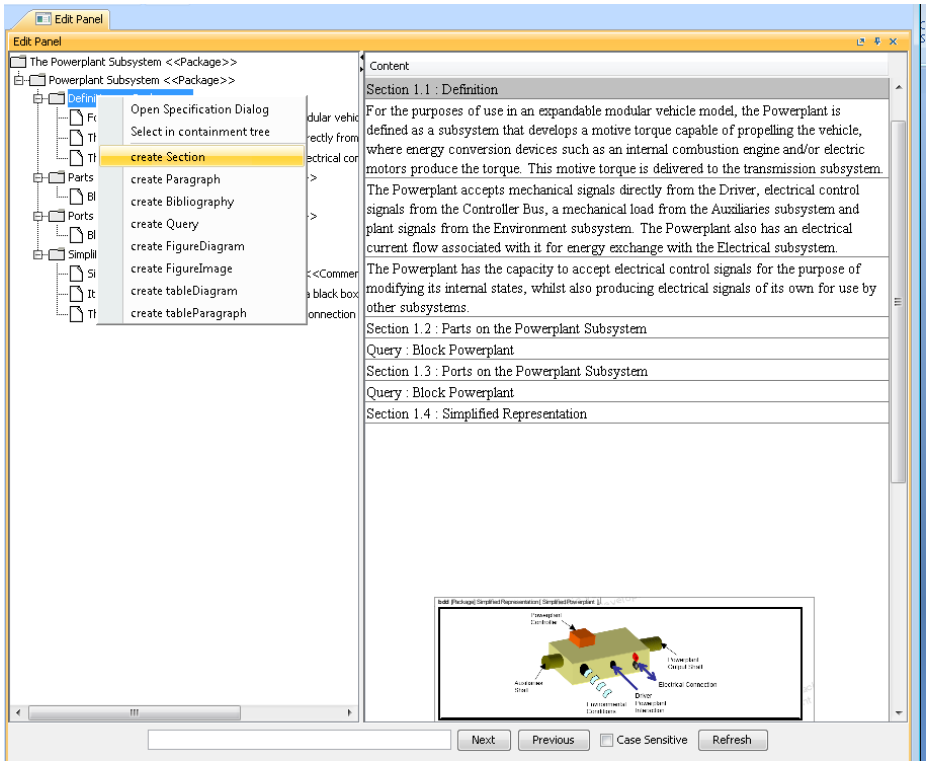


Figure C 5. Creating book elements in the Edit Panel

4. If there is text that is not contained within any block's documentation, then modelers can use «paragraphs» to add any desired text
    a. In the containment tree of the MagicDraw project, either from a «chapter» or «section» within the «book», right-click, navigate to "MBSE" and select "SE2: create Paragraph"
    b. Double-click on the newly created paragraph to open the specification window, and entire the desired text in the "body" field, Figure C 6.

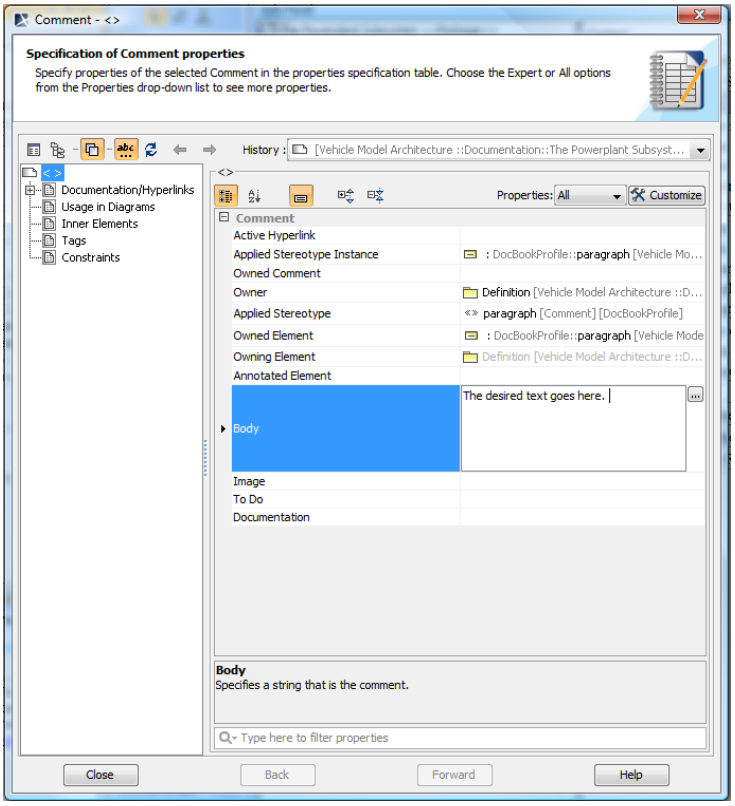


Figure C 6. Specification window for a «paragraph»

c. Alternatively, modelers can right-click on an element in the Edit Panel, choose “create Paragraph”

d. Right-click on the new paragraph and select “Open Specification Dialog” and repeat step 4. b)

5. To add diagrams that are contained within the SysML model, right-click on a «section», under a «chapter» and run the “MBSE” plug-in and choose “SE2: create Figure Diagram”

a. Enter the diagram caption

b. Select the diagram from the SysML project

6. To add tables containing the documentation for parts, ports, reference properties, etc, right-click on a «section» and select “MBSE” and choose “SE2: create Query”

    a. Choose the block that contains the parts, ports, reference properties, etc

    b. Once the query has been created, double click on it to open the specification window

    c. Navigate to the Tags and initiate the desired tags, Figure C 7. The most important tag that must be selected is the “type.” As seen on the right side of Figure C 7, modelers choose whether they wish to represent the part properties, ports, etc.

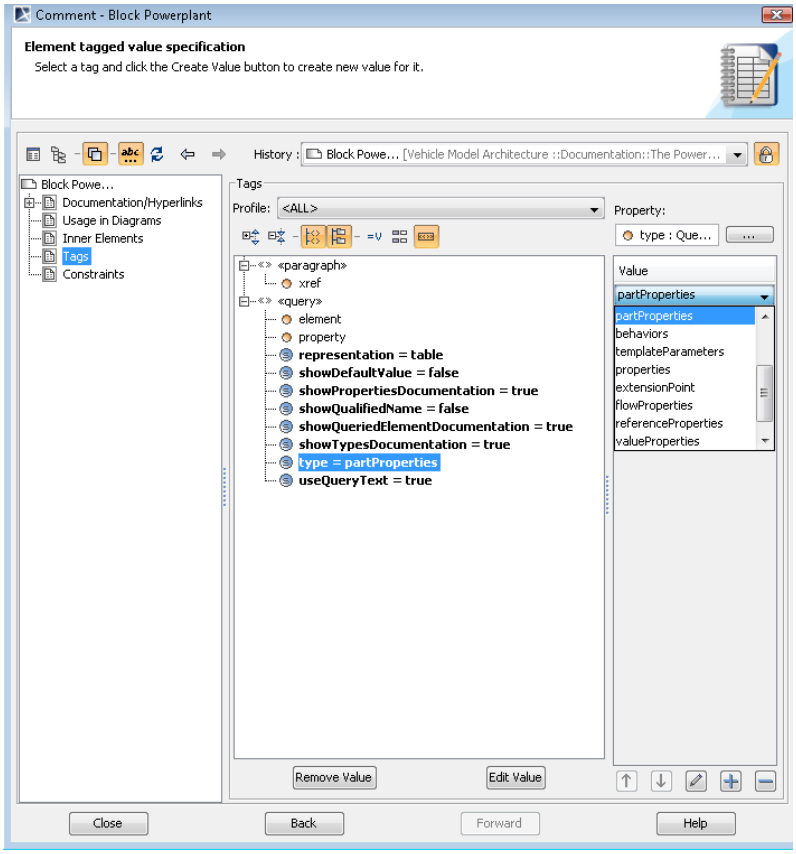


Figure C 7. Specification window for a Query showing the Tags

7. Finally, when the modeler is satisfied with the contents of the «book», they can once again right-click on the «book» and choose “MBSE” and select “SE2: generate PDF.”
    a. The modeler then chooses the name and where to save the document and a PDF is generated.

REFERENCES

[1] "Ford New Product Architecture Abbreviations List," Ford Motor Company.

[2] 2006, *SysML*, OMG Systems Modeling Language (OMG SysML), http://www.omgsysml.org.

[3] 2007, "F-14 Digital Flight Control System Modeled in Simulink," MathWorks.

[4] 2012, Arena Simulation Software, Rockwell Automation, http://www.arenasimulation.com/Arena_Home.aspx.

[5] Belton, C., Bennett, P., Burchill, P., Copp, D. et al., 2003, "A Vehicle Model Architecture for Vehicle System Control Design," *SAE Technical Paper 2003-01-0092*.

[6] Byung I. Min, A. A. K., Christiaan J. J. Paredis 2011, "Process Integration and Design Optimization for Model-Based Systems Engineering with SysML," in *ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Washington, DC, USA, pp. 9.

[7] Che, J., 2012, "Process Flow of Vehicle Modeler," Ford Motor Company.

[8] Che, J., 2012, "Detailed Process Flow of Domain Vehicle Modeler," Ford Motor Company.

[9] Estefan, J., 2007, "Survey of Model-Based Systems Engineering (MBSE) Methodologies," *Incose MBSE Focus Group*, **25**.

[10] Fisher, J., 1998, "Model-Based Systems Engineering: A New Paradigm," *INCOSE Insight*, **1**(3), pp. 3-16.

[11] Friedenthal, S., Moore, A., and Steiner, R., 2008, *A Practical Guide to SysML: The Systems Modeling Language*, Morgan Kaufmann.

[12] Johnson, T., Kerzhner, A., Paredis, C. J. J., and Burkhart, R., 2012, "Integrating Models and Simulations of Continuous Dynamics into SysML," *Journal of Computing and Information Science in Engineering*, **12**(1), pp. 011002.

[13] Karban, R., Zamparelli, M. et. all. , 2011, MBSE for Magicdraw: Plugin for Magicdraw to Support MBSE with SysML http://sourceforge.net/projects/mbse4md/.

[14] Mathworks, S., 2008, "Simulink," vol. 2008.

[15] McGinnis, L., and Ustun, V., 2009, "A Simple Example of SysML-Driven Simulation," *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pp. 1703-1710.

[16] Mitchell, S. W., 2011, "Efficient Management of Configurations in the Model-Based System Development of a Common Submarine Combat System," pp. 8.

[17] Nytsch-Geusen, C., 2007, "The Use of UML within the Modelling Process of Modelica-Models," Linköping University Electronic Press.

[18] Paredis, C. J. J., Diaz-Calderon, A., Sinha, R., and Khosla, P. K., 2001, "Composable Models for Simulation-Based Design," *Engineering with Computers*, **17**(2), pp. 112-128.

[19] Paredis, C. J. J., Bernard, Y., Burkhart, R. M., de Koning, H.P., Friedenthal, S. Fritzson, P., Rouquette, N. F., Schamai, W. , 2010, "An Overview of the SysML-Modelica Transformation Specification," in *International Council on Systems*

Engineering (INCOSE), International Council on Systems Engineering, Chicago, pp. 14.

[20] Peak, R. S., Burkhart, R. M., Friedenthal, S. A., Wilson, M. W., Bajaj, M., and Kim, I., 2007, "Simulation-Based Design Using SysML-Part1: A Parametrics Primer," in *INCOSE Intl. Symposium*, San Diego, CA.

[21] Pop, A., Akhvlediani, D., and Fritzson, P., 2007, "Towards Unified Systems Modeling with the Modelicaml UML Profile," Linköping University Electronic Press.

[22] Qamar, A., During, C., and Wikander, J., 2009, "Designing Mechatronic Systems, a Model-Based Perspective, an Attempt to Achieve SysML-Matlab/Simulink Model Integration," *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pp. 1306-1311.

[23] R. S. Peak, B. R. M. F. S. A. W. M. W. B. M., and Kim, I., 2007, "Simulation-Based Design Using SysML-Part1: A Parametrics Primer."

[24] Sage, A. P., and Armstrong, J. E., Jr., 2000, *Introduction to Systems Engineering*, John Wiley & Sons, New York, NY.