USING MODEL-BASED METHODS TO SUPPORT VEHICLE ANALYSIS PLANNING

A Thesis
Presented to
The Academic Faculty

by

William Cristopher Bailey

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

Georgia Institute of Technology
December 2013

USING MODEL-BASED METHODS TO SUPPORT VEHICLE ANALYSIS PLANNING

ACKNOWLEDGEMENTS

Since I was young, it has always been my dream to go to Georgia Tech. There have been moments throughout my time here that I regretted ever wishing for that, but in the end I can look back and say that every struggle that I had was worth it in the end. After a little more than five years here, it seems that this chapter in my life is coming to a close at last. I've accomplished a lot since I started here, but I never could have done it on my own.

I would first like to thank my parents, Jeff and Tammy, for all of their love and support throughout my life. They were the first to promote my intellectual curiosity, and I have always been motivated by their incredible work ethic. They've always given me the encouragement and advice that I need to overcome any obstacles arise in my life, and I'm unbelievably thankful to always have them to rely on. I must also thank my younger siblings, Michael, Matthew, and Nikki. They've each been a source of inspiration for me in their own ways, and I couldn't be prouder of them.

Second, I would like to thank my advisor, Dr. Chris Paredis, for his continued support and patience with me. I began this project knowing practically nothing about model-based systems engineering, SysML, or programming. It has been an incredible learning experience from start to finish, and I never could have reached this point without his continued guidance and huge breadth of knowledge.

I would also like to thank Mark Jennings and Judy Che from Ford Motor Company. They have been extremely helpful throughout this project and have always been glad to help me whenever I had questions about their work. They also provided me the opportunity to spend the summer working on this project at Ford, which was a fantastic experience for me on its own.

I would also like to thank my third committee member, Dr. Leon McGinnis, for taking the time to read my work and provide his insights into my research.

I must also thank my lab mates: Ben Lee, Sebastian Herzig, Bill Binder, and Douglas Broadwell. They have all supported me throughout this project in some way or another, whether than meant looking through my Java code, discussing SysML modeling, or just making me laugh when I needed it. It’s been a great lab to work in.

Finally, I also acknowledge the support of the George W. Woodruff School of Mechanical Engineering and Ford Motor Company. I would also like to thank No Magic, Inc. for providing the software licenses to perform this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Vehicle system analysis models are becoming crucial to automotive designers wishing to better understand vehicle-level attributes and how they vary under different operating conditions. Such models require substantial planning and collaboration between multidisciplinary engineering teams. To improve the process used to create a vehicle system analysis model, the broader question of how to plan and develop *any* model should be addressed. Model-Based Systems Engineering (MBSE) is one approach that can be used to make such complex engineering tasks more efficient.

MBSE can improve these tasks in several ways. It allows for more formal communication among stakeholders, avoids the ambiguity commonly found in document-based approaches to systems engineering, and allows stakeholders to all contribute to a single, integrated system model. Commonly, the Systems Modeling Language (SysML) is used to integrate existing analysis models with a system-level SysML model. This thesis, on the other hand, focuses on using MBSE to support the planning and development of the analysis models themselves.

This thesis proposes an MBSE approach to improve the development of system models for Integrated Vehicle Analysis (IVA). There are several contributions of this approach. A formal process is proposed that can be used to plan and develop system analysis models. A comprehensive SysML model is used to capture both a descriptive model of a Vehicle Reference Architecture (VRA), as well as the requirements, specifications, and documentation needed to plan and develop vehicle system analysis models. The development of both the process and SysML model was performed alongside Ford engineers to investigate how their current practices can be improved.

For the process and SysML model to be implemented effectively, a set of software tools is used to create a more intuitive user interface for the stakeholders involved. First, functionality is added to views and viewpoints in SysML so that they may be used to formally capture the concerns of different stakeholders as exportable XML files. Using these stakeholder-specific XML files, a custom template engine can be used to generate unique spreadsheets for each stakeholder. In this way, the concerns and responsibilities of each stakeholder can be defined within the context of a formally defined process. The capability of these two tools is illustrated through the use of examples which mimic current practices at Ford and can demonstrate the utility of such an approach.

CHAPTER 1

INTRODUCTION

Analysis models are becoming a critical tool for vehicle development. These models can be used to better understand the impact of different operating conditions on vehicle-level attributes, such as the effect of a cold winter day on a hybrid vehicle's fuel efficiency. To accurately represent a given vehicle, these analysis models are created by integrating different models of the subsystems within a vehicle, such as the engine or transmission. These subsystem models will be referred to as "domain models".

Developing an integratable set of domain models requires that the domain engineers creating them have a deep understanding of both the system-level requirements for the vehicle model, as well as how their models must interface with other domain models. These issues can be partially addressed by performing extensive planning before any models are developed. However, when using a document-based approach to analysis planning, there are still ample opportunities for issues to arise, such as requirements documents becoming obsolete or interfaces between analysis models becoming inconsistent.

This thesis proposes a model-based approach to support analysis planning. This approach is specifically designed to support analysis planning at Ford Motor Company to perform high-level vehicle attribute trade-offs, but the lessons learned can be applied to any analysis modeling process. There are two key components presented in this approach. First, a formal process is defined for system engineers and domain engineers to adhere to when planning and developing analysis models. Second, an infrastructure framework is introduced that can be used within this formal process. The infrastructure framework consists of SysML models and software tools that can be used by both system engineers and domain engineers to plan, develop and verify analysis models. Before introducing this approach, however, the remainder of this chapter discusses the motivation for applying a model-based approach to analysis planning and what characteristics an effective approach should possess.

1.1 Motivation

Pressure to create more fuel-efficient, low-emission vehicles has led to an increased interest in both electrified vehicles and more efficient traditional internal combustion vehicles. To keep pace with the rapid improvements being made to these technologies and maintain competitiveness with other manufacturers, design cycles have become shorter and more efficient. The use of models for Integrated Vehicle Analysis (IVA) has been used to support these shorter design cycles. These models are created by integrating various domain analysis models into a complete vehicle model. The models used for IVA allow different dynamic vehicle-level attributes, such as energy management and performance, to be examined and optimized for various operating scenarios that a vehicle might undergo, such as driving on the highway or towing a boat. These vehicle-level attributes are tightly coupled; investigating the tradeoffs between these attributes is crucial for system design. In addition, these models used for IVA can be extremely useful for evaluating the value of different vehicle architectures, such as weighing the advantages of a plug-in hybrid-electric vehicle (PHEV) versus those of a battery-electric vehicle (BEV).

However, developing these analysis models is an extremely complex task itself. Creating an IVA model is a multidisciplinary task that requires significant input from the many domain experts involved. These teams of domain experts must be able to effectively communicate and coordinate such that when their respective domain models are integrated, the assembled model is an accurate representation of the vehicle being modeled and is appropriate for the intended analysis or cycle. Extensive planning is needed so that the underlying goals and assumptions of these models can be consistent at both the vehicle and domain levels.

Systems engineering practices can be used to develop solutions to complex problems such as these. Systems engineering is a “multidisciplinary approach to develop balanced system solutions in response to diverse stakeholder needs” (Friedenthal, et al., 2012). In this case, the stakeholders concerned are primarily the domain- and system-level engineers. Traditionally, many engineering practices have used document-based approaches for systems engineering, in which documents (such as reports or presentations) are exchanged between teams to communicate information. Such approaches have significant pitfalls. Different disciplines often use different vocabularies, so that semantic differences can cause misunderstandings between teams. Continuously managing and updating a large number of documents can be extremely tedious and error-prone, which can result in significant inconsistencies between the documents. Generating and maintaining these documents must be done manually, which introduces the potential for human error and disregards the potential to use the abundant computing power available to improve this process. An alternative is to take a model-based approach to systems engineering.

1.2 Applying MBSE to Systems Analysis Modeling

Model-based systems engineering (MBSE) makes use of formal modeling methods to support complex engineering tasks in lieu of a document-centric approach. There are no constraints placed on what constitutes a model or on the systems that can be designed using such an approach. A model can potentially take any form and serve any purpose, whether it be a 3D CAD model or a MATLAB script. Often, these models are created to provide different, specialized views of the same system for specific stakeholders.

To organize and capture the relationships between these different views, a system-level model is often used. The Systems Modeling Language, or SysML, is becoming the de facto language for producing such a “descriptive” system model. Significant work has been done towards linking domain-specific analysis models, such as those created in Modelica or Simulink, to a system-level SysML model.

Traditionally, MBSE is used to design a physical system, such as an automobile or airplane. In the approach presented by this thesis, however, the “system” being designed is an analysis model itself. Our goal is to use MBSE and SysML to support the preliminary planning needed to define the set of desired analyses and to create the appropriate set of integrated vehicle analysis models. Developing these models requires efficient communication, collaboration, and understanding among multidisciplinary engineering teams, which MBSE and SysML can improve significantly. Some background and justification for the selection of both MBSE and SysML is presented in Chapter 2.

1.3 Motivating Question

To improve the capabilities of SysML and other modeling tools to support analysis model planning, a broader motivating question must be addressed:

Motivating Question:

How should one plan and guide the development of analysis models?

To support model consistency within an MBSE methodology, extensive work has been performed by others to support the transformation from SysML models to other modeling languages. Integration frameworks using SysML to unify many different existing models in a variety of simulation toolsets have also been researched heavily. However, very little emphasis has been placed on supporting the definition and integration of those models. Developing integrated vehicle analysis models is a complex, multidisciplinary process. To add value to this process, this thesis proposes a SysML-based approach with a complementary Excel user interface to define a set of analyses and plan for development and verification of the analysis models. This approach is intended to support a transition from a document-based workflow to a more model-based one. Because quantifying the value of such an approach can be subjective, it will be judged instead on several different "Desired Characteristics" that an ideal approach would possess.

Desired Characteristic:

Follow a formal, precise process

Oftentimes, engineering tasks are performed in an ad hoc manner and rely on the expertise and experience of the engineers involved to produce results. However, for complex engineering tasks that require coordination among many individuals, this can be a particularly risky approach. Developing an approach that follows a formal, precise process can provide a better understanding of the responsibilities of different engineering teams and areas where issues may arise. A formal, precise process should define not only the different tasks that stakeholders must perform, but also more specifically, which pieces of information are expected to be exchanged between stakeholders and how this should occur.

Desired Characteristic:

Provide an intuitive graphical user interface

To make the analysis model planning and development process more efficient, the graphical user interface (GUI) should be intuitive to the many different stakeholders that will use it. These users may have different levels of expertise and experience with the planning tools being used, so it is important to consider all levels of familiarity when selecting (or creating) a user interface. For our approach, SysML models are used extensively to support the analysis planning process. However, many engineers today are still unfamiliar with SysML. Because of this, the training and licenses needed to put

SysML tools into practice across large engineering teams can be cost-prohibitive. As a result, a more intuitive GUI is considered desirable.

Desired Characteristic:

Minimize the opportunity for inconsistencies

There are many different areas where inconsistencies can arise during the development of analysis models. Misunderstandings about the goals of an analysis model, miscommunication between teams, and even simple human error can all contribute to inconsistent analysis models. These inconsistencies, if not found and addressed, can lead to longer development cycles and even erroneous results from the analysis models. An effective approach should identify and minimize these opportunities for inconsistency wherever possible.

Desired Characteristic:

Support the roles of all stakeholders directly involved

As mentioned, analysis modeling is an inherently multidisciplinary process. There are many different stakeholders with many different unique concerns. As a result, an effective approach to analysis modeling must consider all of these different stakeholders and their responsibilities and skill-sets. More specifically, an effective model-based approach should not only support these many different tasks, but also make them easier to perform. This characteristic is very strongly related to the final characteristic which must be considered.

Desired Characteristic:

Add value to the process

For a model-based approach to analysis model development to be useful, it must be more valuable than the existing (and in this case, document-based) approach. To add value to the process, the benefits of using such an approach must be weighed against the costs of implementing it and training users. Although a quantitative measure of value is not used, this is still an extremely important aspect to consider. Current practices at Ford have been given particular attention to ensure that any proposed approach would improve the existing processes, rather than necessitating entirely new workflows.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents the related work and the context within which our research has been performed. Chapter 3 introduces the formal process outlined for the planning and development of complex analysis models. Chapter 4 presents the reusable set of tools and models that are used to support analysis model planning. In Chapter 5, these tools and models are applied to the process outlined in Chapter 3. Finally, Chapter 6 discusses the conclusions drawn from this research and provides a response to the motivating question posed in this chapter.

CHAPTER 2

RELATED WORK

2.1 Model-Based Systems Engineering

Model-based systems engineering (MBSE) is an alternative to traditional document-based approaches to systems engineering. INCOSE defines MBSE as the “formalized application of modeling to support…activities beginning in the conceptual design phase and continuing through development and later life cycle phases” (2007). The benefit of a model-based approach is dependent on the quality of the descriptive models themselves, which are only valuable if they increase “a decision maker’s ability to design a better system at an acceptable cost” (Keeney, 1994). To support MBSE, these models must have a clearly defined purpose. A particular model may only represent part of a larger system; therefore, it is often advantageous to have a system-level model to understand how these different specialized models are related.

(Estefan, 2007) provides an overview of various MBSE methodologies that have been used across different industries. Some of these methodologies are designed around certain toolsets, such as the IBM Rational Unified Process for Systems Engineering (RUP SE) or the Vitech MBSE Methodology. Others, however, are defined as broader methodologies that can be implemented through whatever tools can best support the process. It is important to note that regardless, there is no all-encompassing methodology that can be applied to any MBSE process. Whichever methodology an organization chooses—or creates—must be modified and adapted to support that organization’s project lifecycle. To do so, one must consider the knowledge, skills, and abilities of all of the stakeholders that will be involved.

There are many different tools that can be used with the MBSE methodologies presented in (Estefan, 2007), but the Systems Modeling Language (SysML) is quickly becoming the de facto standard. SysML was developed by the Object Management Group™ (OMG™) as an extension to the Unified Modeling Language (UML) (Object Modeling Group, 2012). SysML was designed specifically for systems engineering applications and is intended to help unify the many different modeling languages used by systems engineers. It uses a graphical model representation with a formal set of semantics to define different characteristics of a system, including the requirements, structure, behavior, and parametric relationships between system elements. SysML reuses some UML diagrams and introduces two new ones—requirement diagrams and parametric diagrams. SysML, like UML, is also extensible. This allows the possibility for SysML to be customized for specific domains as needed. A much more detailed overview of SysML and its role in MBSE methodologies is presented in (Friedenthal, et al., 2012). It is worth noting that the system models presented in this thesis are created in SysML using No Magic's MagicDraw tool and SysML plugin (No Magic, 2013).

MBSE has been applied across many different industries to support different phases of engineering design. Often, the cost of implementing a complete, model-based workflow can be difficult to justify. To move towards such an approach, some organizations have used small pilot projects as test beds for MBSE methodologies. One such project was started at JPL to investigate and model the behavior of an antenna on an Earth-orbiting satellite (Ingham, et al., 2012). While the pilot project focused on a small test case, the goal was to use it to investigate the broader benefits of integrating MBSE into JPL’s existing systems engineering practices, and how it could be used to develop better products in general. This pilot project provided the opportunity to investigate different modeling patterns and revise them for future projects. Taking small steps in this way can be a useful measure to determine the value of MBSE for a particular organization.

Small pilot projects are not the only test case used to assess the value of MBSE. (Robinson, et al., 2010) introduces a methodology applied to the development and support of a ground-based air defense system. In this work, MBSE was used to improve the preliminary process of outlining and documenting the capabilities desired for the system. The authors found that a model-based approach was completely compatible with their current development processes, and had the added advantages of increased efficiency, standardization, and greater stakeholder understanding of the system. It is worth noting that the MBSE methodology applied in this work was the Vitech MBSE Methodology—one of the methodologies introduced in (Estefan, 2007). This methodology was found to be particularly effective; it was chosen because many of the staff were already familiar with the CORE suite of software tools. Again, considering the expertise and experience that stakeholders already have is very important when selecting an MBSE methodology.

The benefits of MBSE are not limited to any particular industry, either. MBSE has been successfully implemented for other aerospace applications (Graves and Bijan, 2011, Spangelo, et al., 2013), for telescope design (Claver, et al., 2010, Karban, et al., 2009), and even for modeling a disaster management system (Soyler and Sala-Diakanda, 2010). Its benefits are also not limited to organizations with substantial funding—a university satellite team found that using MBSE and SysML to capture the structure, behavior, requirements, and parametrics of their design improved member understanding and the productivity of their design reviews (Nottage and Corns, 2012).

MBSE has also been used across the automotive industry to different extents. In (Piques and Andrianarison, 2012), SysML was used to support the systems engineering activities for developing embedded automotive systems. In (Marco and Vaughan, 2010), high-level system models in SysML were used to support the understanding and development of automotive electronic control systems. (Dumitrescu, et al., 2013) presents an approach which used SysML models to capture variability within a family of parking brakes to better understand how SysML can be used to support mass customization.

In (Branscomb, 2012), a different approach is used to support automotive applications. Integrated vehicle analysis models are already used extensively to support vehicle design. These models can be complex and require collaboration and communication between many different multidisciplinary teams. Branscomb et al. proposed an MBSE approach where the "system" being developed is in fact a model itself (Branscomb, 2012). A SysML model of a vehicle architecture was created and used to auto-generate model templates for Modelica and Simulink. This approach ensured that domain models remained consistent and reusable for future applications. It is important to note that this work is the direct predecessor to the work presented in this thesis; many of the models and methods used in our approach are heavily influenced by the work in (Branscomb, 2012).

2.2 Using SysML with Analysis Models

A descriptive system model can provide an integration framework for relating different, more specialized models (Friedenthal, et al., 2012). There are many different ways to implement such an integration framework. One such example is the Formal United System Engineering Development Language, or FUSED (Boddy, et al., 2011). FUSED is a meta-language specification that is intended to unify nine different languages (with the possibility of adding more later), including Modelica, SysML, and Excel. The goal was to make it easier to integrate MBSE practices into existing tools. While the authors claim that the higher-order grammar used in FUSED is an advantage over UML, it has only been applied to a limited use case.

In (Eisenmann, et al., 2010), an integration framework is presented which was created by the European Space Agency (ESA) for virtual spacecraft design. This framework is built around a reference database, rather than a particular model or modeling language. The database is used to store all of the design data in a central location and manage its integration into other tools. Users can use a series of Design Editor tools built on SysML notation and the Eclipse environment to design and specify a virtual spacecraft model. The database can also be accessed through the Space Systems Visualization Tool, which is capable of producing diagrams, 3D model representations, tables, and charts to help support design reviews and simulations. A prototype of the environment is available for download for free by users living in a member country of the ESA (European Space Agency).

An alternative to FUSED and the ESA's approach is the Systems Lifecycle Management workspace, or SLIM (Bajaj, et al., 2011). Rather than utilizing a new meta-language, SLIM proposes a framework built around SysML to integrate different domain-specific models. Such a framework could be used to integrate any number of domain-specific models, including CAD models, simulations, or requirements, among others. Because SysML is focused on system-level architecting and modeling, it can serve as an acceptable integration framework for many different engineering tasks and tools. The SLIM technology is available from InterCAX at InterCAX.com.

To integrate SysML with analysis models, formal transformations must be used to make the tools interoperable. Such transformations have been developed for many different cases, including SysML to Arena (McGinnis and Ustun, 2009), SysML to Modelica (Johnson, et al., 2012), and SysML to Simulink (Qamar, et al., 2009), among others. (Branscomb, 2012) provides a more thorough overview of these many different model transformations. All of these approaches have one thing in common—they are designed to integrate existing analysis models with SysML, but do not directly support the planning and development of those models. The aim of this thesis is to support this gap in analysis model development. To do so, developing model transformations is only part of the problem. It is also important to formally manage the knowledge generated when planning and developing these models.

2.3 Knowledge Management in MBSE

In a traditional document-based approach, each stakeholder works from their own domain-specific tools and documents that they need to perform their tasks. For a model-based approach, this necessity remains. Ideally, each stakeholder would only use models which have been customized to their "view" of the system, in lieu of any documents.

Practically speaking, however, this is not yet possible. Documents and presentations are still an integral part of the engineering design process. To support this functionality for a SysML-driven MBSE approach, many tools have been developed that allow users to generate and modify documents linked to a SysML model.

The most basic functionality needed is the ability to generate documents from a SysML model. Many tools exist to do so. MagicDraw includes a built-in "Report Wizard" tool that can be used to generate PDF reports from a SysML model (No Magic Inc., 2013). It supports the use of text-based templates to generate these documents and includes a query tool for parsing through different SysML elements and their attributes. However, using the Report Wizard to generate reports requires that users understand the Report Wizard user interface, the Velocity Template Language, and the MagicDraw API. Once the templates are created, they can be reused extensively. However, creating them initially requires an experienced, expert-level user, which can be a hindrance when first transitioning to a model-based approach. In addition, information can only be transferred in one direction. Users can view the reports generated, but any edits to them will not be reflected by the SysML model.

An alternative to MagicDraw's built-in Report Wizard is the DocBook plugin for MagicDraw (INCOSE SE2 Challenge Team for Telescope Modeling). The plugin is built around the DocBook standard—a mature schema used to capture the structured content of a document without making any assumptions about what *kind* of document it is (Walsh, 2011). The plugin for MagicDraw allows users to create a "model" of their document using a specialized SysML profile. Unlike the Report Wizard, the DocBook plugin provides its own basic user interface and simplified model queries that can be used to create a document in SysML and export it to a PDF. However, like the Report Wizard, information can only flow in one direction; users cannot use these documents to update the SysML model.

In (Delp, et al., 2013), a different approach is taken to model-based knowledge management. A much more formal method is used to describe the different views and concerns of stakeholders. SysML includes “View” and “Viewpoint” elements to model the perspective and concerns of different stakeholders, but these elements are not functional. In (Delp, et al., 2013), these elements are redefined and linked to functional Activity diagrams that describe how to construct each stakeholder’s view. These views can take many different forms, including pop-up, editable tables, web interfaces, or exported documents. This work is capable of exporting DocBook XML files, but also adds the ability to package information in a REST interface that allows users to update the model from an HTML view. When a large design team is working from a SysML-based integration framework, the ability to work with user interfaces other than SysML models and exported documents can be extremely valuable.

However, there are still many gaps remaining. Certain SysML elements or properties may need to be modified by users without SysML experience, so a user interface that can support this is desirable. While the environment presented in (Delp, et al., 2013) is extremely powerful, it is the culmination of years of research and funding. To transition from a document-based approach to a model-based one, a simpler approach for supporting different stakeholder views is preferred (at least initially).

2.4 Summary

There are many different methodologies for MBSE and many different possible applications. However, there is no universal MBSE methodology. No set of tools and processes can fully address the needs of every specific engineering task. As a result, either the process, the tools, or both must be customized for each application.

However, the true value of an MBSE approach lies in the quality of the models. Significant work has been done to create integration frameworks which can unify existing models into a single framework, but very little research has been done to support the planning and creation of those models. The remainder of this thesis presents a model-based approach that can be used to plan and develop complex analysis models within a large, multidisciplinary engineering team.

CHAPTER 3

ANALYSIS PLANNING AND RELATED PROCESSES

In this chapter, a formal process for planning, developing, and using analysis models is introduced. This process is modeled in detail in SysML using Activity diagrams to capture the flow of information and the stakeholders involved. Using these diagrams, the overall process can be better understood. In addition, mapping activities to certain stakeholders provides a better understanding of who the ultimate decision-makers are for each step. Because this approach focuses on *planning* the analysis models, special attention is given to that aspect of the process.

3.1 Overview of Integrated Vehicle Analysis Planning and Development

The use of integrated vehicle analysis models is becoming crucial to automotive engineers wishing to better understand how the vehicle system performs under different operating scenarios. These models are particularly useful when performing propulsion system design and optimization. To create these analysis models, it is necessary to first plan and develop detailed domain analysis models according to vehicle-level goals and requirements. Once these domain models have been planned, developed, verified, and validated, they can be integrated together to simulate a complete vehicle. This vehicle model can then be optimized for key system-level attributes, such as fuel efficiency. Conclusions drawn from these simulations can be used to support the development of both physical components and control algorithms for real-world vehicles.

This process of developing detailed analysis models may be performed upstream of the vehicle’s design and may be used to support different milestones in the vehicle’s design. Depending on the milestone being targeted, different sets of data or levels of fidelity may be required to produce an accurate model.

Coordinating all of the different tasks needed to produce an accurate vehicle model requires developing a thorough understanding of the underlying engineering processes. To do this, a set of activity diagrams is used to demonstrate how these vehicle analysis models are developed, from start to finish. Figure 1 shows a high-level overview of this process. Note that there are three distinct activities shown here: *Analysis Planning*, *Model Development*, and *Design Analysis*. The focus of our research is on supporting analysis planning, so that is the focus of this thesis and this chapter, in particular. However, the latter two phases are also discussed in some detail to give the reader a more complete understanding of the entire process.

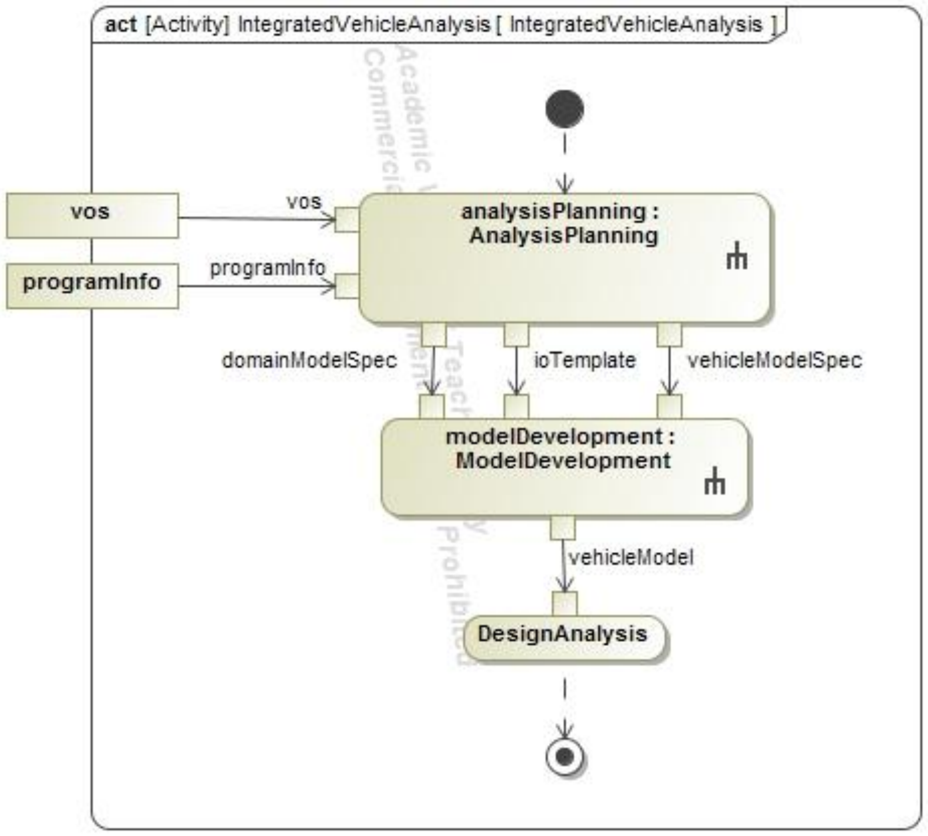

act [Activity] IntegratedVehicleAnalysis [IntegratedVehicleAnalysis]
vos
programInfo
vos
programInfo
analysisPlanning : AnalysisPlanning
domainModelSpec
ioTemplate
vehicleModelSpec
modelDevelopment : ModelDevelopment
vehicleModel
DesignAnalysis

Figure 1. Simulation development process activity diagram

3.2 Analysis Planning Process

The first phase of the process shown in Figure 1 is the “Analysis Planning” phase. In this phase, the different sets of analyses which are needed to support a given vehicle development milestone are identified and defined in some detail. In most cases, this means running a variety of vehicle analyses in order to reach some conclusion. This set of analyses will hereafter be referred to as a “System Analysis Application Plan”. The majority of this research is focused around supporting this analysis planning phase, as many inconsistencies can arise out of the planning process. It is important that the different teams of domain subject matter experts are all working under the same set of assumptions and understand what the overall objective of a particular analysis actually is. To identify these sources of inconsistency and to develop tools and methods to reduce them, the analysis planning phase is modeled in a more detailed SysML activity diagram. This diagram is shown in Figure 2. Each activity presented in Figure 2 also includes a review and approval process. To make the diagrams more legible, the series of reviews is shown separately in Figure 3.

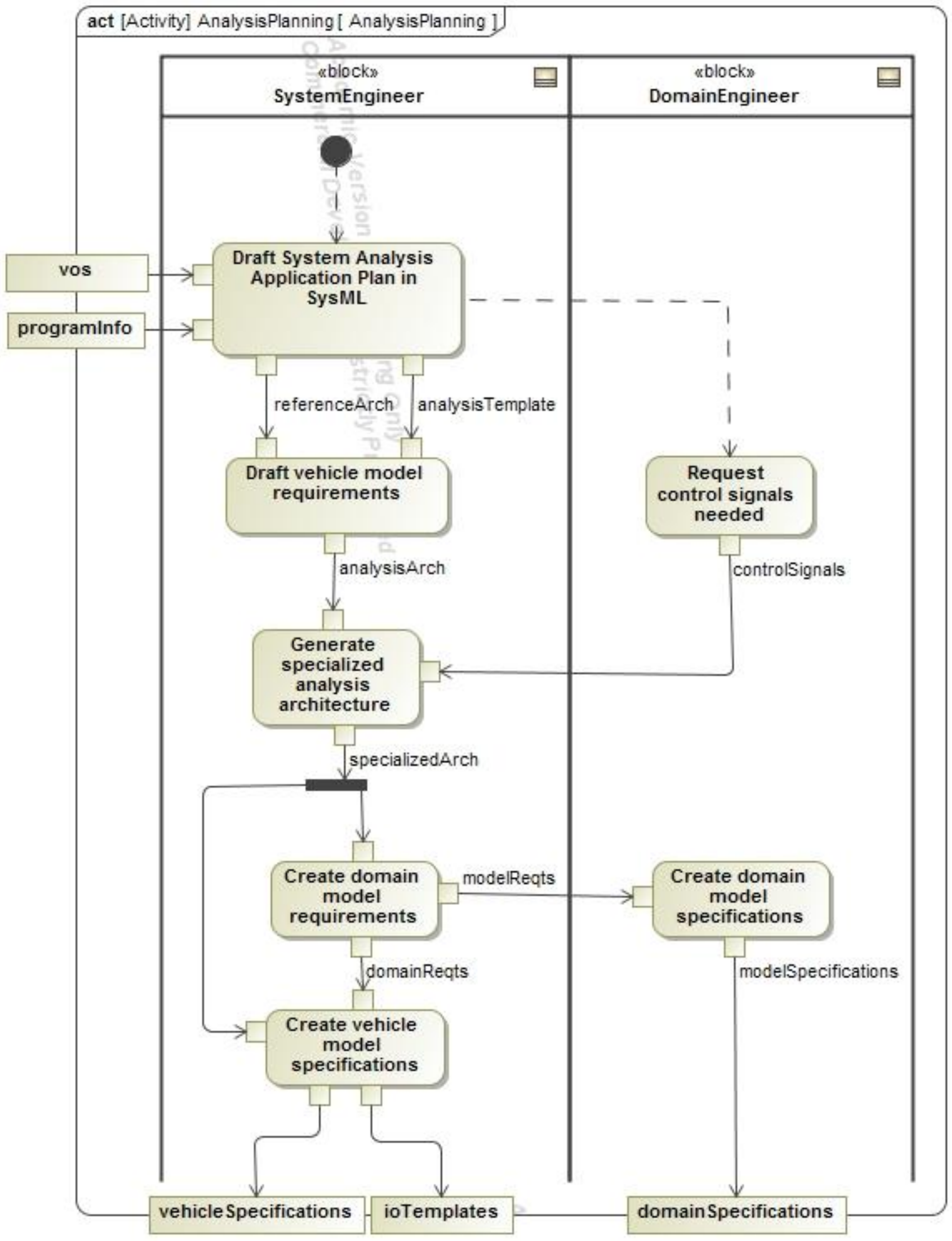

act [Activity] AnalysisPlanning [AnalysisPlanning]
«block»
SystemEngineer
«block»
DomainEngineer
vos
programInfo
Draft System Analysis Application Plan in SysML
referenceArch
analysisTemplate
Draft vehicle model requirements
Request control signals needed
analysisArch
controlSignals
Generate specialized analysis architecture
specializedArch
Create domain model requirements
modelReqts
Create domain model specifications
domainReqts
modelSpecifications
Create vehicle model specifications
vehicleSpecifications
ioTemplates
domainSpecifications

Figure 2. Analysis planning activity diagram

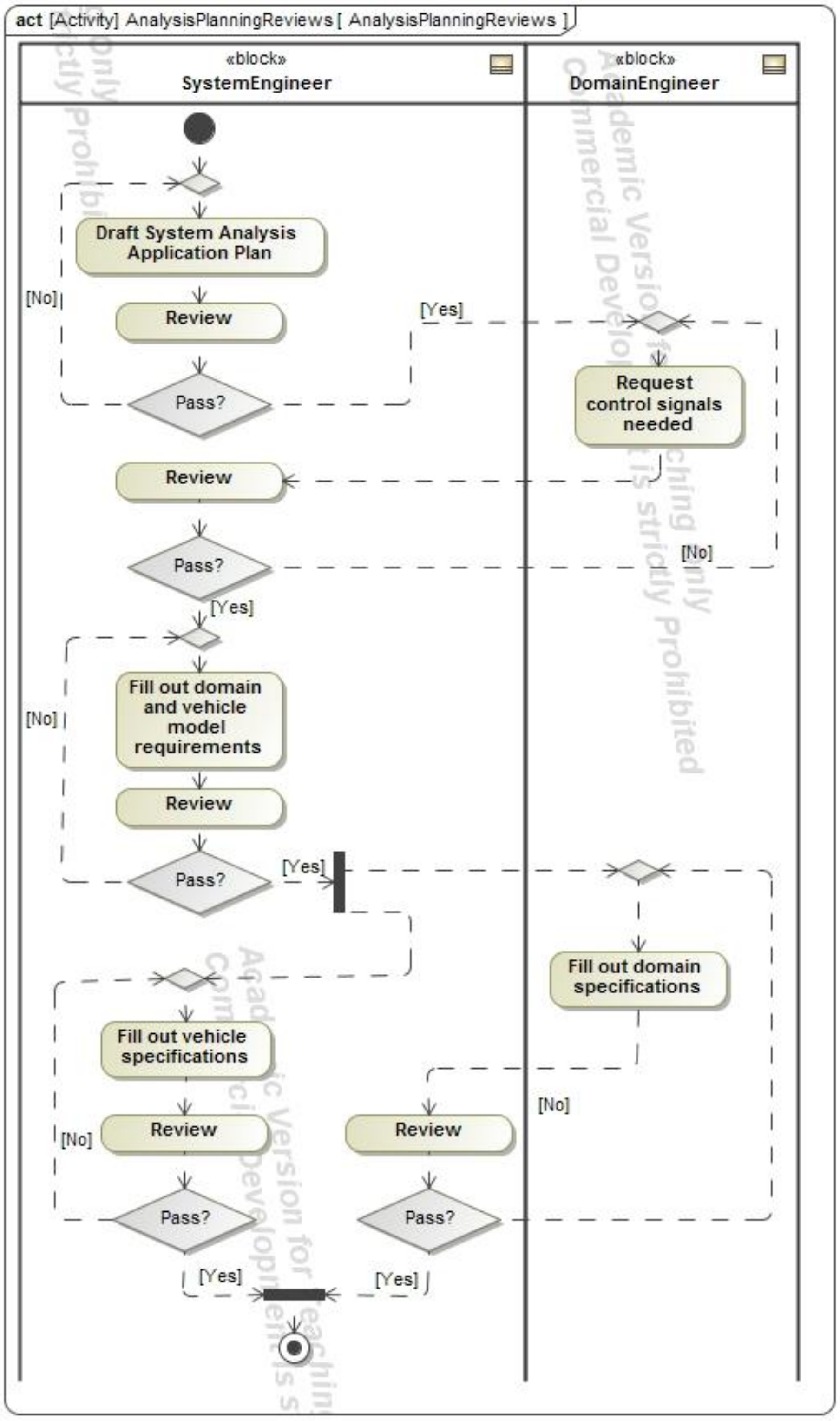

act [Activity] AnalysisPlanningReviews [AnalysisPlanningReviews]
«block»
SystemEngineer
«block»
DomainEngineer
Draft System Analysis Application Plan
Review
Pass?
[No]
[Yes]
Request control signals needed
Review
Pass?
[No]
[Yes]
Fill out domain and vehicle model requirements
Review
Pass?
[No]
[Yes]
Fill out domain specifications
Fill out vehicle specifications
Review
Pass?
[No]
Review
Pass?
[No]
[Yes]
[Yes]

Figure 3. Activity diagram for analysis planning reviews

3.2.1 Inputs

Figure 2 shows that there are two inputs to the analysis planning process. The first is the set of potential Vehicle Operating Scenarios (*VOS*). A VOS is a specific test event that will be used to execute a vehicle simulation, such as a drive cycle or vehicle launch scenario. These VOSs include some qualitative details about what the test event is meant to describe, as well as more quantitative information such as operating ranges and conditions to which the vehicle may be exposed. Depending on the goal of the overall system design study, the set of VOSs selected may vary. For instance, if the goal of the study is only to evaluate fuel efficiency attributes, the set of VOSs may only include selected regulatory drive cycles.

The second input to the analysis planning process is the program information for the vehicle being modeled. The program information contains several different details about the vehicle model to run. First, it specifies which vehicle architecture should be modeled, such as a traditional internal combustion engine or power-split hybrid architecture. Second, it describes the control signals available to use.

Note that this program information may also contain unique information about the types of vehicles to be modeled. For instance, if the analyses are being used to evaluate a drivetrain that will be used across multiple vehicles, the program information might include details about different vehicle top hats (body styles) that will need to be investigated. In addition, many analyses are used to simulate unique subsystem technologies that have not been used previously or are not well-understood. In this case, it is important to emphasize those technologies up front and pinpoint changes that may need to be made to the standard analysis modeling procedures to account for them.

These two inputs—the set of Vehicle Operating Scenarios to consider and program-specific information about the vehicles to model—can be used to infer many details about the types of analysis models that need to be run. Using these inputs, a more detailed plan for the analyses that need to be run can be drafted. This is known as the *System Analysis Application Plan*.

3.2.2 Drafting the System Analysis Application Plan

Once the inputs to the analysis planning phase have been evaluated, the next step is to draft the System Analysis Application Plan. This step is performed by a team of system engineers, as it requires taking a holistic view of the system design and the related vehicle attribute tradeoffs that need to be assessed. These system engineers typically represent a range of different disciplines, such as transmission hardware or engine controller modeling. In this way, the needs of all of the different domain engineers—who create the models—may be taken into consideration when the system analysis application plan is created.

The primary purpose of the System Analysis Application Plan is to identify all of the analyses that need to be performed to support the design and system trade-offs for a vehicle or vehicle subsystem. These different analyses are referred to as the "analysis applications". Each analysis application is a detailed description of a specific analysis that needs to be run. This description includes information about the purpose of the analysis, what vehicle models will be simulated, and which attributes will be looked at. The full set of analysis applications can be used to investigate the tradeoffs between vehicle system attributes. There are several different components of an analysis application that must be defined, and it is extremely important to understand the relationships between them. Without this understanding, it is possible that the analysis application specified by the system engineers may be contradictory or confusing.

These relationships are illustrated by the influence diagram shown in Figure 4. This influence diagram graphically depicts how the two inputs to the analysis planning phase—the program information and set of vehicle operating scenarios—can be used to systematically define an analysis application and how the information contained by this analysis application influences outside elements as well.

In many cases, experienced system engineers should understand these influences implicitly. However, semantic misunderstandings and simple human error can still lead to inconsistencies when defining an analysis application. Formally mapping these influences can be used to better understand how and where such inconsistencies arise. This formal map can also be used to build a system of checks-and-balances into the user interface where analysis applications are defined.

Several pieces of information specified for an analysis application do not directly influence anything else. These elements are shown at the top-left of Figure 4. First, the analysis application is given a name and short description so that it is clear to domain and system engineers what each model is intended to represent and why it is needed to support a particular system study.

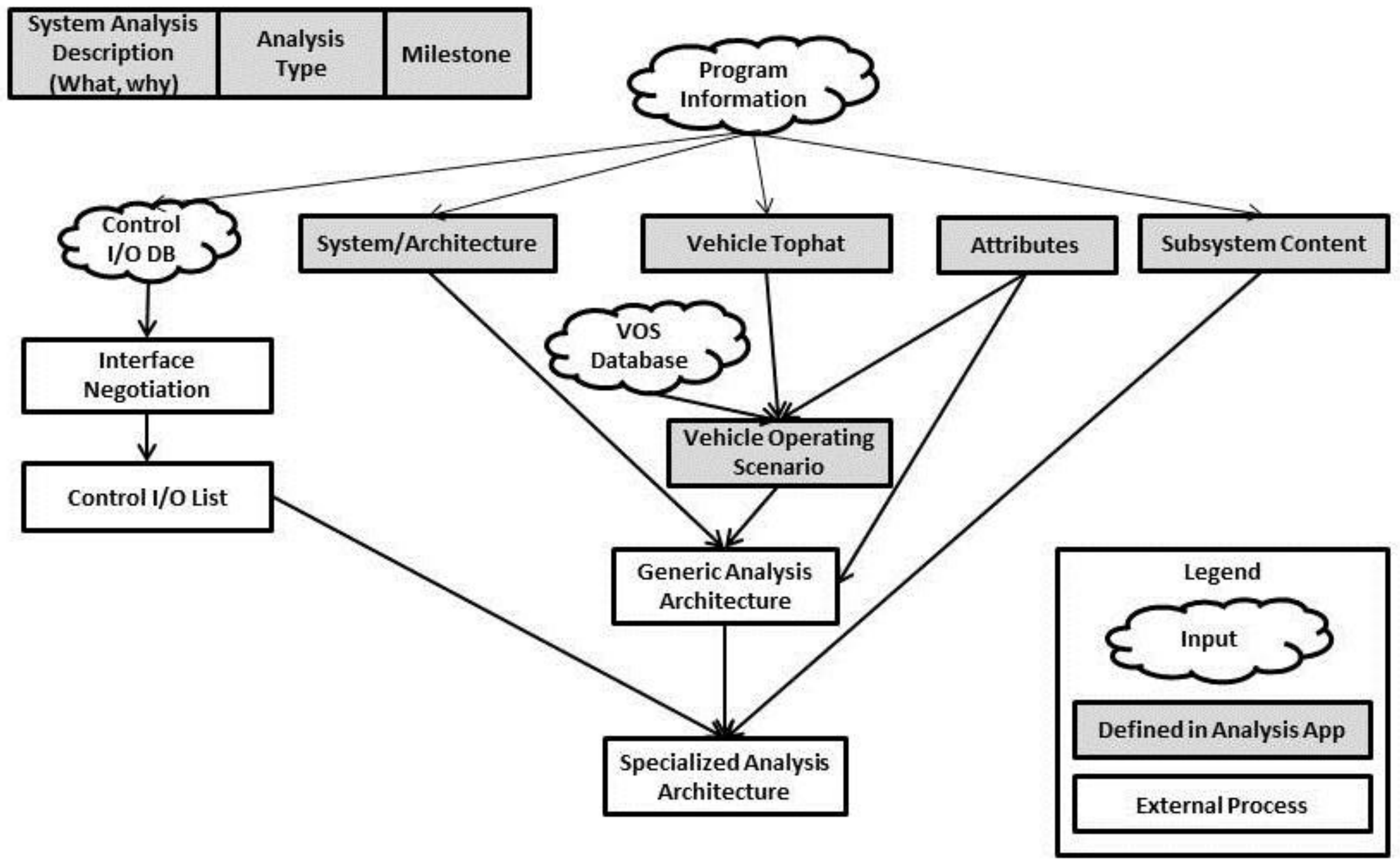


Figure 4. Analysis application influence diagram

Second, system engineers define the “Analysis Type” for each analysis application. The analysis type describes the purpose of the results from each analysis. For instance, the analysis may be used to check a correlation between simulation models and physical tests, or to model potential technologies that do not have extensive real-world test results to assess their utility. While this information is important for the *perspective* that domain engineers take when creating analysis models, it does not directly affect any of the other pieces of information.

The final piece of non-influencing information that must be defined is the “Milestone” targeted by the analysis. The milestone indicates the stage in the vehicle development process at which these results should be delivered by. It serves not only as a deadline for the analysis model’s development, but also implies what level of model fidelity is required for the analysis. In addition, because each system analysis model must be verified and validated against different vehicle datasets, the milestone can also be used to indicate which datasets will be available to use at that time.

Once these three elements have been defined, system engineers can use the program information to define an analysis application, as shown in Figure 4. The program information contains details about the complete set of control signals available on the real-world vehicle at that point in its development. The process of narrowing down this list of signals to only those needed for an analysis model is performed in an iterative process by both system and domain engineers. This process is discussed in Section 3.2.4.

System engineers must also be clear about which vehicle tophat and vehicle architecture to model, as a vehicle program may have many different possible combinations of these two elements. For example, if a system design study is used examine a particular drivetrain rather than a specific vehicle model, multiple vehicle models might need to be constructed to understand all of the different operating scenarios of that drivetrain. Alternatively, if the goal of a system design study is to examine a particular vehicle, that vehicle might have several possible architectures, such as a traditional internal combustion engine version or a full-hybrid version. It is of the utmost importance that this information is presented explicitly; otherwise it could result in major inconsistencies between the models that domain engineers produce.

The analysis application must also identify the vehicle-level attributes to be evaluated. Examples of vehicle-level attributes include fuel efficiency, electric drive range, or emissions, among others. One or several of these attributes may be targeted by a particular analysis application.

The second input to the analysis planning phase is the set of available VOSs to model. By knowing the attributes to examine and the system architecture and vehicle tophats to model, system engineers can narrow down this list of VOSs. Certain VOSs may be used for different vehicle architectures (such as special drive cycles for electric vehicles), different vehicle tophats (such as towing simulations for trucks and SUVs), or different vehicle attributes (such as EPA drive cycles for testing fuel efficiency). It is critical to define these pieces of information up front, so that only valid VOSs can be chosen for analysis work.

Finally, system engineers may also need to identify subsystem content that should be modeled, such as the engine or transmission type. This content is included as part of the program information, but depending on the analysis application, may or may not need to be modeled for a given analysis.

Once system engineers have defined the pieces of information denoted as "Defined in Analysis App." in Figure 4, the analysis application can be used to select a "Reference Architecture" and "Analysis Template". This is shown by the outputs of the "Draft System Analysis Application Plan" activity in Figure 2. Using these two resources, system engineers can draft a set of vehicle model requirements and use them to create a specialized analysis architecture. These two tasks are discussed together in the following section.

3.2.3 Vehicle Model Requirements and Analysis Architecture

Using the analysis application defined in the previous step, system engineers can draft a set of requirements for the vehicle system model. This includes specifying the computer and operating system that the model should be run on, which solvers and file formats to use, what language(s) the model should be written in, etc. These requirements are used to drive the development of the individual domain subsystem models, whose requirements are defined later. Part of this process also involves using a reference architecture and analysis template to define the vehicle model's structure.

A reference architecture is an extremely useful resource for a complex, multidisciplinary process such as vehicle system analysis. Reference architectures provide a holistic view of a system and allow different stakeholders to work together from a common vehicle system definition. Many different reference architectures exist for capturing vehicle architectures. AUTOSAR, for instance, is the product of an industry-wide effort to produce a standardized architecture for controls design and development that can be used among major OEMs and their suppliers (AUTOSAR, 2013).

For our approach, a reference architecture developed internally at Ford Motor Company is used. This reference architecture is known as the Vehicle Model Architecture, or VMA. There were two primary motivations behind the creation of the VMA. First, like AUTOSAR, the VMA allows models to be exchanged between different organizations under a common format. Second, the VMA simplifies the process of modeling different system configurations. The VMA contains a hierarchical breakdown of the different subsystems contained within a vehicle. In addition, the primary physical interfaces between all of these subsystems are specified. Because this common set of domain-to-domain interfaces is used, different subsystem models (such as different engine models) may be more easily interchanged within the same vehicle architecture (Tiller, et al., 2003). A SysML representation of the VMA is presented in Section 4.1.1.

Using information about the analysis application, such as the type of vehicle being modeled and vehicle operating scenarios to use, system engineers can create a specialized version of the reference architecture to build their analyses around. This is known as the "analysis architecture". An analysis architecture captures only the set of interfaces and hierarchy of domain models needed for a particular analysis. Starting from a given reference architecture such as VMA, a finite set of generic analysis architectures can be pre-specified for most common analysis applications. The same analysis architecture can potentially be modified to address several combinations of vehicle operating scenarios, vehicle-level attributes, and types of analyses, so there are significantly fewer analysis architectures than there are combinations of analysis attributes.

Despite the broad range of applications that an analysis architecture can address, it is still necessary to specialize this generic analysis architecture for each vehicle program. Creating a specialized analysis architecture is performed by adding the program-specific set of control signals and making any changes to the analysis architecture that may be needed to capture the unique subsystem content. The process of selecting these signals is discussed in the following section. Once a specialized analysis architecture has been developed, it can be passed to domain SMEs to use as the basis for their analysis models.

3.2.4 Requesting Control Signals

For each analysis application, domain engineers must identify the control signals that they anticipate will be needed to execute their analysis models. The initial program information provided to them includes details about the control signals available at that time. Depending on the phase of vehicle development at which this occurs, domain engineers may have the ability to request new signals that they believe they will need to execute their models. At later phases of vehicle development, however, the signals used in analysis models will be fixed as they begin to represent the signals used in the actual vehicle. In addition, analysis models created to support later development milestones may be used for hardware-in-the-loop (HIL) testing and to support code generation for control units, so correctly modeling the real-world set of control signals becomes crucial.

Defining the list of control signals to use in an analysis model is an iterative process. Domain engineers must specify both the control signals that they need and the control signals that they plan to provide from their models. In addition, they may request new signals that they believe should be available. This complete list of signals is then reviewed by the team of system engineers, who negotiate with each team of domain engineers to develop a consistent set of signals for the entire vehicle. These system engineers must negotiate with both the domain engineers who will provide certain signals and those who will be receiving those signals, so that all of the analysis models can interface correctly. Because this is an iterative process, it may require several rounds of negotiations with all of the different teams of domain engineers before a vehicle-wide set of control signals can be agreed upon by everyone involved. Tools such as signal databases are used to capture this information in a central location. However, because the full list of control signals for a vehicle model can be quite large, it is still possible that inconsistencies can arise if this list is not checked extensively.

3.2.5 Domain Model Requirements

Once an analysis application and set of vehicle requirements have been defined, the team of system engineers develops a set of requirements for each domain model that will be created. These requirements are first created by the system-level team to ensure that any assumptions made at the vehicle level will also be reflected by each domain model. In addition, this holistic view allows the team of system engineers to establish computational restrictions on the analysis models. This may include defining the operating system that models should be compatible with, which simulation environments will be used to run the integrated vehicle model, and any other guidelines that the domain model should comply with.

In addition, system engineers can also add more domain-specific requirements at this stage, such as a particular engine that should be modeled. Once all of the domain requirements have been created, they are then sent to their respective domain engineers for review. After the reviews are complete and all parties are in agreement, the requirements are sent back to the domain engineers, who use them to create a set of domain model specifications.

3.2.6 Domain Model Specifications

Once the domain model requirements have been created by the system engineers, domain engineers are responsible for developing their own set of specifications for the analysis models they plan to create (as shown by Figure 2). These model specifications are intended to capture more detailed information about how they plan to create, verify, and validate their respective domain models. This includes identifying specific data sets that will be necessary, how that data will be processed and used, and the desired fidelity of the model. Detailed plans to verify and validate the models are also needed so that the accuracy of the domain models can be confirmed before they are integrated into a complete vehicle model. Although problems can still arise when integrating these models, preliminary verification and validation at the domain-level can eliminate some of these problems.

To create this set of domain model specifications, domain engineers must work from the information defined by the team of system engineers. This information includes system-wide criteria defined by the analysis application, as well as their domain-specific requirements. While domain engineers can view this information, they do not have the authority to directly modify any of it. This distinction is important and is made at several other steps throughout this process. By identifying who has the ultimate authority to make decisions during different phases of the modeling process, the software tools created to support it can be tailored to different users. Once the domain specifications have been reviewed and approved by the system engineers, the analysis planning phase can be concluded.

3.2.7 Vehicle Model Specifications

Like the domain models, each vehicle model also has its own set of specifications. However, their purpose is somewhat different. While the domain model specifications are intended to specify what kind of model to create, the vehicle model specifications specify *how* all of those domain models will be integrated. This complete process which defines how a vehicle model should be assembled and tested is known as the "integration plan". Its use is discussed in more detail in Section 3.3.2.

3.2.8 Outputs from Analysis Planning

There are three primary outputs from the analysis planning phase. The first is the set of domain model specifications created by the domain engineers. These domain model specifications are essentially the product of all of the work done in the analysis planning phase, and should provide enough background information for the domain engineers to create their analysis models.

The second two outputs are related to the vehicle model itself—the vehicle model specifications and the I/O templates, as shown in Figure 2. These I/O templates are model templates in either Modelica or Simulink (for plant or control models, respectively) that the domain engineers must fill in with their model content. Each template has a predefined set of interfaces so that it may correctly integrate with the other domain models. The set of interfaces modeled is derived directly from the details of the specialized analysis architecture.

One benefit of adhering to the domain/vehicle model specifications and I/O templates is that it allows a repository of analysis models to be built up around a common format. Once this repository has been built up, it becomes possible to more easily search through existing models and their documentation—at both the domain and vehicle level—to find models that can be reused or modified to address new analysis applications. With a large enough repository, this can significantly reduce the time and cost needed to create an integrated vehicle analysis model.

3.3 Model Development Process

Once the specifications for each domain model have been defined and the set of I/O templates has been generated, domain engineers can begin work developing their system analysis models. This process is illustrated by Figure 5. The I/O templates used in this process contain no content other than a pre-defined set of primary interfaces; the content of the models is left to the domain engineers.

Supporting and improving the model development process was the primary focus of previous work by Branscomb et al. (Branscomb, 2012, Branscomb, et al., 2013). The

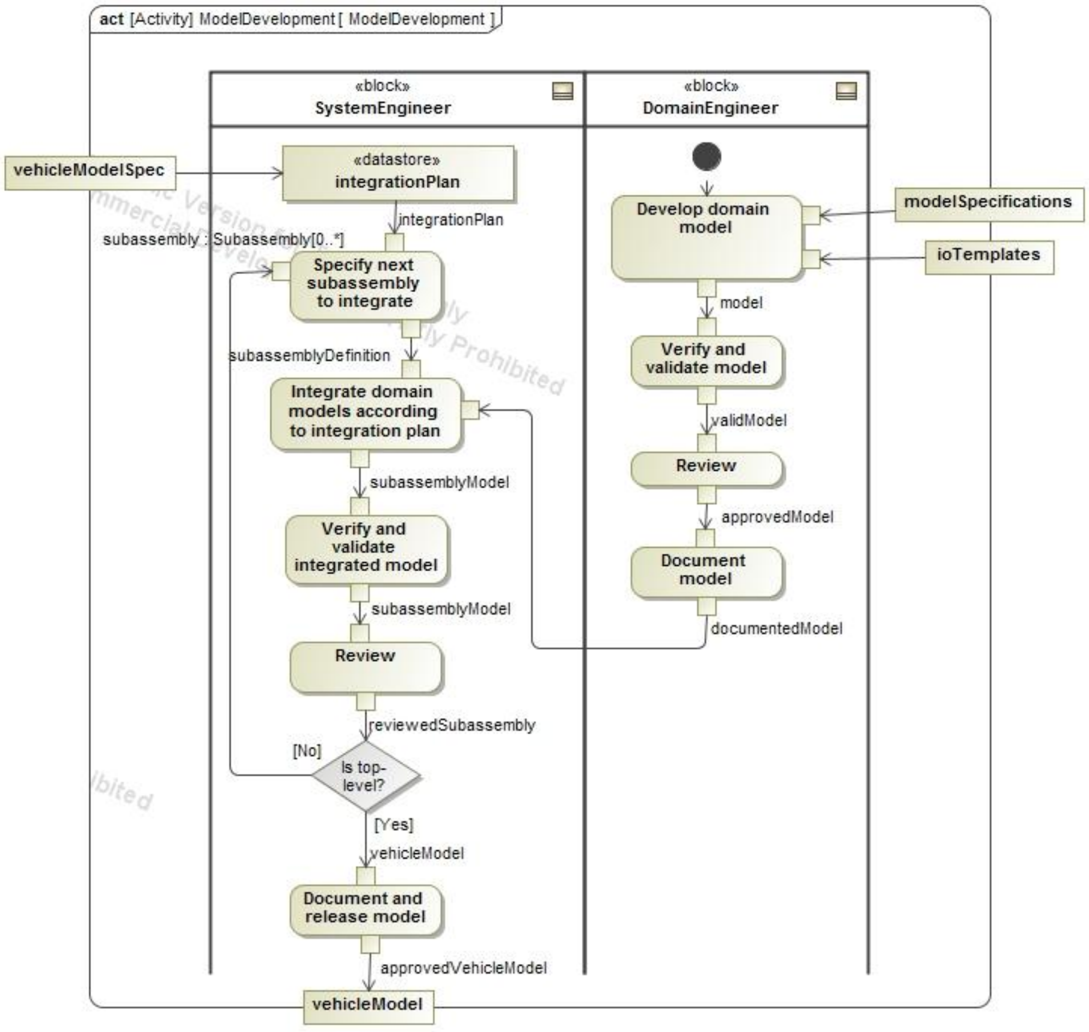


Figure 5. Model development Activity diagram

authors used a slightly simplified model development process, whereby plant and control models were assumed to be created exclusively in Modelica and Simulink, respectively. This assumption is carried throughout our approach as well, as it is representative of the majority of modeling work being considered. Rather than discussing each activity presented in Figure 5, the broader responsibilities of the two actors (shown by the *DomainEngineer* and *SystemEngineer* swimlanes) are discussed in this section.

3.3.1 Domain Engineer Swimlane

The model development process, unlike the model planning process, starts with the domain engineers. Once an analysis model has been created, verified/validated, and reviewed, the model must then be documented. The model created should adhere to the set of specifications defined before its development; if deviations from the specifications are necessary, the specifications must be updated, reviewed by the team of system engineers, and approved, before continuing that model's development. The documentation for each model does not duplicate this information, as doing so could potentially result in inconsistencies between this information and the model specifications.

Instead, it is primarily used to describe the results of the verification and validation efforts. In addition, the documentation for an analysis model contains some administrative information, such as who created the model or who to contact to get a copy. The documentation typically also includes some instructions for how to use the model and for what applications it is intended. In some cases, a single analysis model may be used to meet multiple sets of requirements and specifications. When this occurs, the documentation should also include references to any sets of model requirements and specifications to which it adheres.

3.3.2 System Engineer Swimlane

Once each team of domain engineers has created, verified, and validated their analysis models, these models are handed off to a system engineer who is responsible for integrating the domain models to produce a complete vehicle model. The system engineer also receives the vehicle model's specifications, as shown by the input parameter in Figure 5. The most important component of the specifications is the integration plan for the vehicle model. The integration plan describes the order in which domain models should be integrated and how they should be verified and validated at each step along the way.

The integration plan may call for intermediate subassemblies to be created and tested, rather than immediately creating the complete, integrated vehicle model. By doing so, it is easier to identify problem areas in domain models as these subassemblies are built up. When necessary, domain models may need to be updated and the integration process restarted. Once the completed vehicle model has been created and approved by the team, it is documented and passed on to the next phase of the process, the Design Analysis Phase.

3.4 Design Analysis Process

The third and final phase is the "Design Analysis Phase". This phase is not addressed by this thesis or in previous work by Branscomb et al., but it is worth introducing at a high level. The goal of this phase is to execute one or more analysis applications and use the results to improve the system design of the vehicle. This may involve performing a design of experiments on different vehicle attributes or an analysis of different system alternatives, whether at the vehicle- or subsystem-level. Once this has been completed, the conclusions drawn from these results can be saved, documented, and delivered to other engineers to use in the engineering design process of the real-world vehicle.

3.5 Applying MBSE to Analysis Planning

With the components of the analysis planning phase formally defined, the next step is to identify how MBSE principles can be applied to make the process more efficient. Again, one of the issues with MBSE is that there is potentially a substantial cost associated with it, and the benefits may not be quantifiable or even necessarily obvious. To add value to this process, an approach is needed which would not require significant capital to implement. Previous work by Branscomb et. al made steps towards a model-based approach for the model development phase, but much of the analysis planning phase still required a document-based approach.

To address this, our approach aims to capture the knowledge generated during the analysis planning process using MBSE principles. By doing so, the time required to manually create, update, and communicate this information can be significantly reduced. In addition, by using models to link the analysis planning phase to the model development phase, a comprehensive repository of models and their documentation can be more easily created and maintained. Chapters 4 and 5 present the models and tools developed to support this approach in much greater detail.

CHAPTER 4

INFRASTRUCTURE FRAMEWORK

This chapter proposes a set of models and tools intended to be used to support the analysis planning process introduced in Chapter 3. These models and tools will henceforth be referred to as the "infrastructure framework" for our approach. The infrastructure framework includes a SysML model used to capture both the vehicle model architecture and the analysis artifacts needed to support its development. In addition, a template engine is used in Excel which is capable of importing data about the SysML model in the form of an XML file and producing a unique spreadsheet for engineers to input information. This infrastructure framework is capable of aiding the transition to MBSE practices by presenting a familiar user interface to stakeholders and using formal SysML models to minimize inconsistencies that may arise during the development of a multidisciplinary analysis model.

4.1 SysML Model

This section presents the different SysML models which have been created to support analysis planning and development. These SysML models can be used to understand information about the vehicle itself, such as the relationships between different domains, as well as the different information elements needed to fully plan and describe the models being created. By capturing all of this information using the common set of semantics that SysML provides, the entire analysis modeling process can be more easily understood by all stakeholders involved.

4.1.1 Reference Architecture

A Vehicle Reference Architecture (VRA) is modeled in SysML to capture the set of physical interfaces and logical hierarchy of a vehicle architecture. The goal of the VRA is to create a structured model which can be used to not only present and understand the vehicle architecture, but also more practically, to generate a set of model templates in Modelica and Simulink with a predefined set of interfaces. In this way, the potential for inconsistency when creating a multidisciplinary vehicle model can be minimized in many respects when analysis models from across many domains are integrated.

This reference architecture conforms to a version of the Vehicle Model Architecture Version 3 (VMA v3) specification, which is still under development. Currently, the set of interfaces in VMA v3 is not fully defined, so modeling efforts are focused around capturing the interfaces known for the propulsion system and developing modeling patterns that can be followed once the VMA v3 specification is finalized. Creating this reference architecture is not the primary focus of this work; many of the decisions, details, and challenges of modeling the VRA are discussed by (Branscomb, 2012, Branscomb, et al., 2013). However, the previous iteration of the VRA used both an older VMA specification and SysML specification. As a result, many model elements of the previous VRA are updated and modified. In addition, some previous modeling decisions have been changed, which are also worth noting.

The current iteration of the VRA introduces a new domain element hierarchy, as shown in Figure 6 below. At the highest level of the hierarchy is the *VehicleDomain* block, which encompasses both the vehicle (*ICHEVVeh*) and the environment (*Env)* with which it interacts. The vehicle is broken down into six subsystems: Power (*Pw*), Chassis (*Cha*), Vehicle System Controller (*Vsc*), Cabin (*Cabn*), Air (*Air*), and Climate (*Climt*). Note that at this level of detail, this hierarchy would be consistent across any vehicle architecture (such as either a hybrid-electric or battery-electric vehicle). However, in order to define the interfaces in more detail, each vehicle architecture must currently have its own Vehicle Reference Architecture model in SysML. The current SysML specification does not natively support variants, so an all-encompassing VRA which captures all of these potential vehicle architectures would be both time-consuming to create and unwieldy to use. In addition, because the VMA v3 specification is still in development and the interfaces are not completely defined, such an effort is not yet possible. Instead, a single, semi-complete VRA is used in SysML to model a traditional hybrid vehicle. This VRA model serves as the foundation for the tools and methods developed to improve the analysis planning process.

All of the subsystems and their components in the VRA, with the exception of the Vehicle System Controller, are composed of both a plant element and a control element. Plant and control elements are denoted in the SysML model by appending either “Plnt” or “Ctrl” to the element names, respectively. These plant and control elements are all specializations of the generic “Plant Element” and “Control Element” blocks shown in Figure 7.

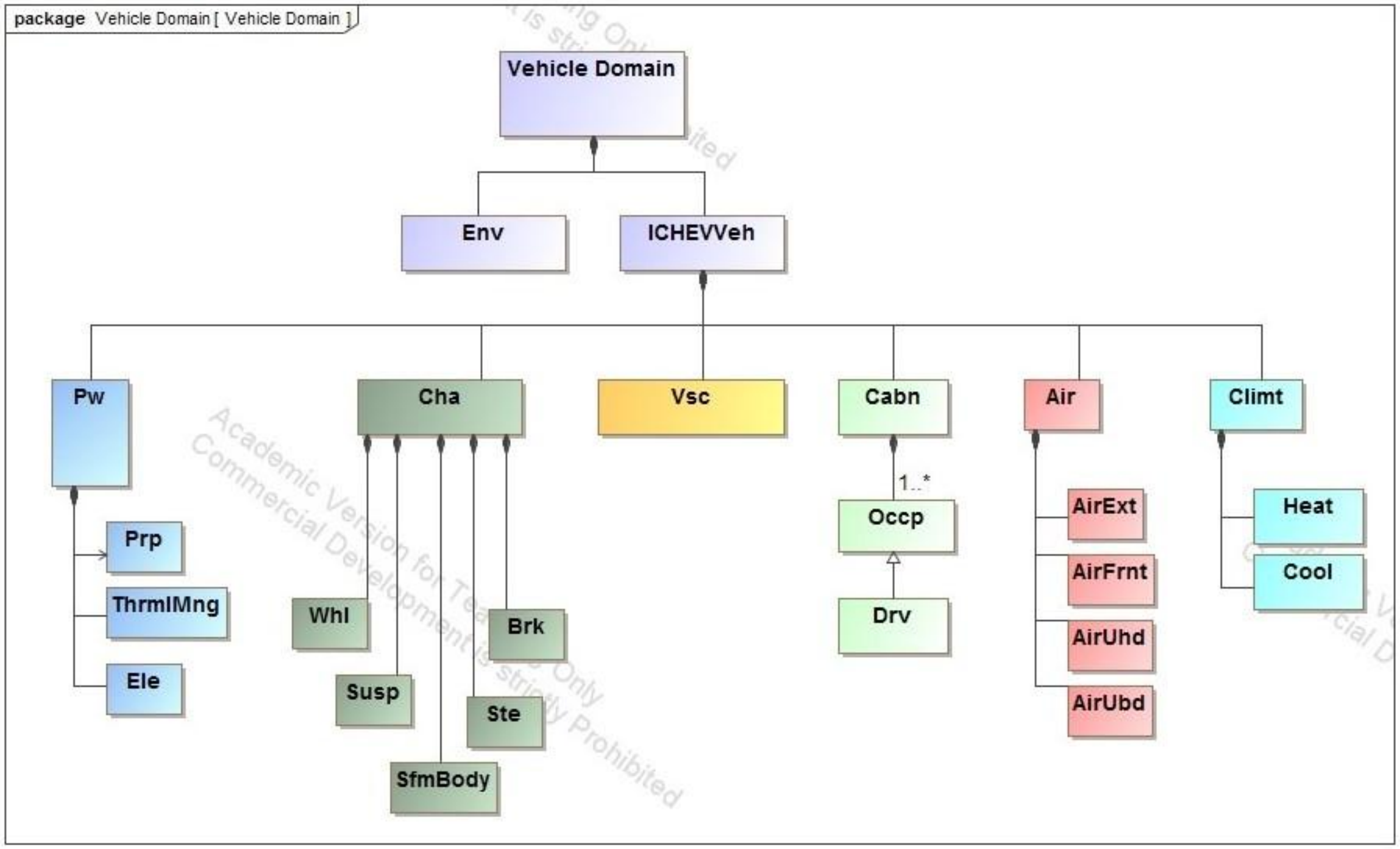


Figure 6. Vehicle Domain Hierarchy

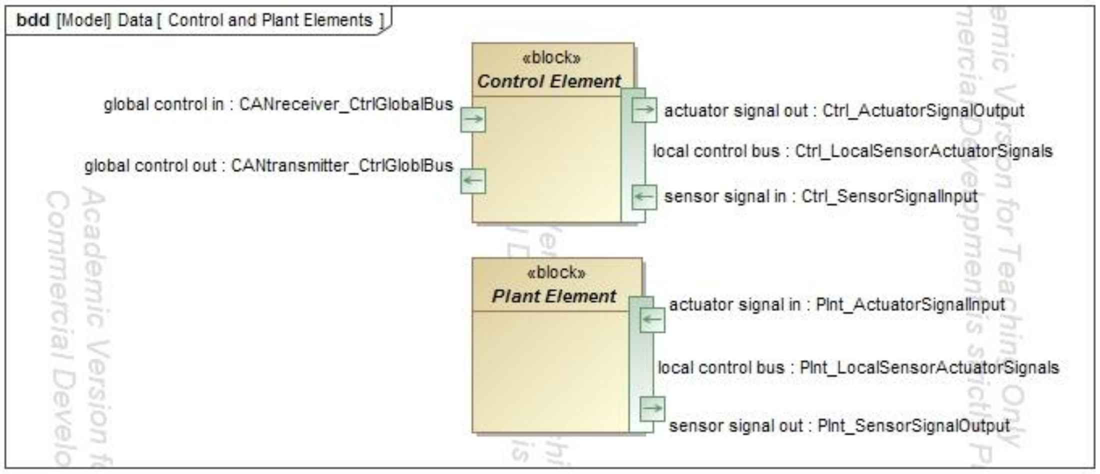

bdd [Model] Data [Control and Plant Elements]
«block»
Control Element
global control in : CANreceiver_CtrlGlobalBus
global control out : CANtransmitter_CtrlGloblBus
actuator signal out : Ctrl_ActuatorSignalOutput
local control bus : Ctrl_LocalSensorActuatorSignals
sensor signal in : Ctrl_SensorSignalInput
«block»
Plant Element
actuator signal in : Plnt_ActuatorSignalInput
local control bus : Plnt_LocalSensorActuatorSignals
sensor signal out : Plnt_SensorSignalOutput

Figure 7. Generic plant and control elements

These elements are used to model the generic set of control signal buses for control and plant elements. For control elements, there are three proxy ports to consider. The first two, *global control in* and *global control out*, are used to model the signals sent and received between the control element and the global CAN bus, respectively. The second proxy port, *local control bus*, represents the signals passed between the plant and control element within a single domain. This port has two nested ports. The first, *actuator signal out*, refers to signals sent to actuators in the complementary plant element. Similarly, *sensor signal in* refers to signals sent to the control element from its related domain element. For plant elements, these local ports are reversed. On the local control bus for plant elements, actuator signals are received and sensor signals are sent. Because knowledge about these signals is unique to each vehicle program and tends to evolve as the vehicle design becomes more detailed, the control signals needed for a particular analysis model cannot be known entirely in advance. As a result, a Vehicle Reference Architecture cannot capture all of the control signals that might be used for an analysis model like it can for physical interfaces. Instead, placeholders are used in the VRA so

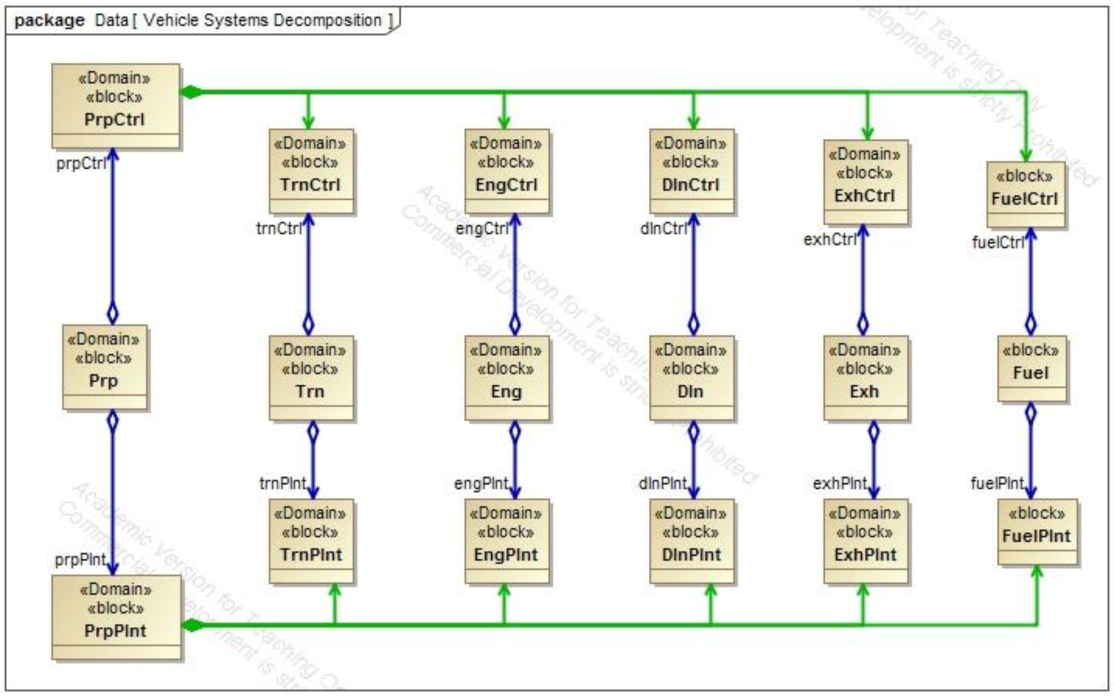


Figure 8. Propulsion system decomposition

that these signals may be added later to the Specialized Analysis Architecture. This is discussed further in Section 4.1.4.

An example of the decomposition of the "Propulsion" domain into its respective sub-domains and their plant and control elements can be seen by Figure 8 below. Note that the *Prp* block also owns *Trn*, *Eng*, *Dln*, *Exh*, and *Fuel*, but those relationships are not shown in this diagram.

In the VMA specification, physical interfaces are fixed and well-understood for the plant elements. However, VMA v3 introduces significantly more physical interfaces to the architecture than in VMA v2. As a result, port nesting is implemented to organize the interfaces in SysML. Because the interfaces defined in the VMA specification refer to energy exchanges rather than physical connections, proxy ports are chosen to represent them in SysML. Proxy ports do not represent physical parts of a system like full ports;

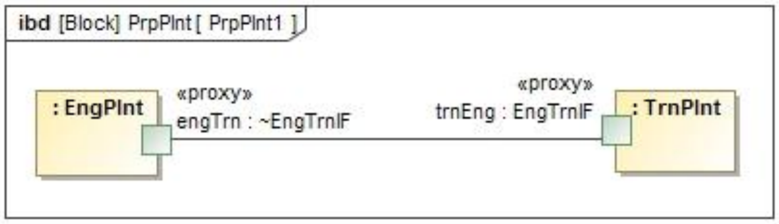


Figure 9. Propulsion plant IBD

instead, a proxy port is used to expose “features of either its owning block or parts of that block” (Friedenthal, et al., 2012). In this case, the “features exposed” refer to the types of energy being exchanged between two domains.

To organize the substantial number of interfaces defined by the VMA v3 specification, ports are grouped according to which two domains are interacting. An example of this is shown in part by Figure 9. In this figure, a single connector is used to define the connection between the *engTrn* and *trnEng* ports on an engine plant and transmission plant, respectively. Both ports are typed by an *EngTrnIF* interface block.

The definition of the *EngTrnIF* interface block is provided by Figure 10, which defines two internal proxy ports, *work* and *thrml*, to capture rotational and thermal energy exchanges, respectively. In addition, the figure also shows an association block, *EngTrnInct*, which is used to define the connection between two “EngTrnIF”-typed ports. Using this association block, an IBD is created to model the connectors between the nested ports, as shown by Figure 11. Creating these three diagrams provides a complete, formal method for modeling a nested set of physical interfaces. Through the use of hyperlinks to navigate between these diagrams in MagicDraw, it also serves as a useful interface for examining and analyzing pieces of the VRA model in increasing levels of detail. This pattern is repeated to capture all of the known interactions between domains.

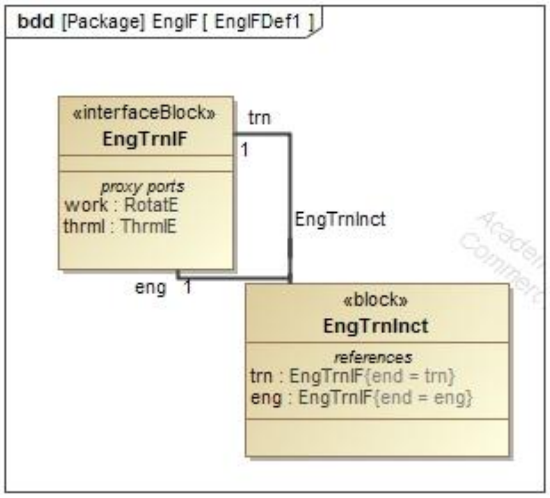


Figure 10. Definition of connection between Engine and Transmission

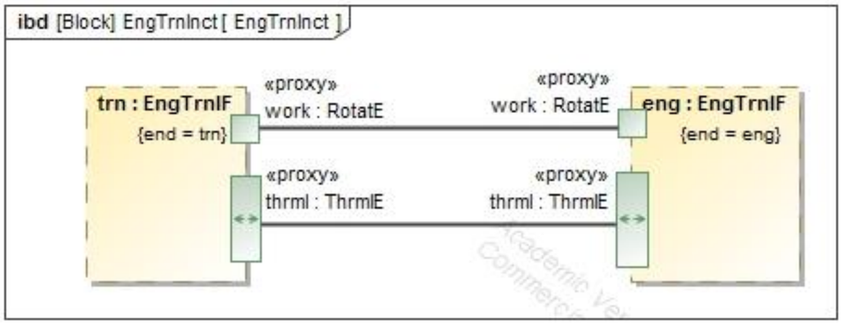


Figure 11. EngTrnInct IBD

4.1.2 Analysis Artifacts

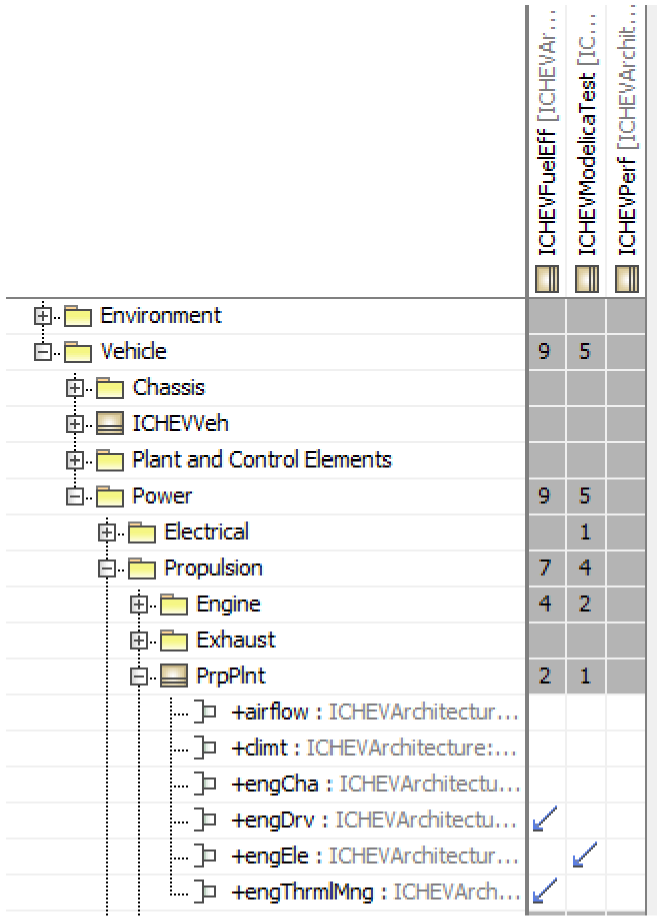
A substantial amount of information must be generated and organized in order for analysis models to be properly planned, developed, and run. This may include details about the kinds of analyses to be run, the requirements and specifications for each domain model or complete vehicle model, or the documented results of the analyses. This broad category of information produced during analysis modeling will be referred to as the “analysis artifacts”. To better organize the analysis artifacts and the information that they contain, they are formally captured in a SysML model and linked to the VRA. In most cases, this information is stored as instance data inside a SysML package created for each analysis.

The analysis artifacts modeled are adapted from existing documents used during the analysis model development process (Jennings 2013). Some elements were removed and other elements added to better support the approach proposed by this thesis. Each information element is classified based on several categories. First, each is categorized based on whether the information should belong to the model requirements, specifications, or documentation. In addition, each is grouped according to whether it pertains to domain models, vehicle models, or both. Finally, where possible, a finite list of options is defined for information elements in order to restrict the possible user inputs and to limit ambiguity. For instance, the VOS used to drive a particular analysis may be selected from the set of operating scenarios imported before defining the system analysis application plan (see Chapter 3). In some cases, these lists of choices may vary from domain to domain, so specialized analysis artifact models are used where needed. It is not the goal of this research to completely define these lists of choices, but rather to provide the framework for more experienced users to do so. These analysis artifact models are presented in more detail in Chapter 5.

4.1.3 Analysis Templates

Analysis models typically will not use the full set of interfaces available from the VRA. To define the interfaces needed for a particular analysis (such as a fuel efficiency study), it is necessary to identify the specific ports which will be used in an analysis model. To do this, SysML “analysis templates” are defined in advance.

An analysis template is modeled as a block in SysML which has dependencies to the ports needed for a particular vehicle analysis. It is possible to create these dependencies manually on a BDD, but the number of ports and their nesting in the VRA makes this process incredibly tedious. Instead, dependency matrices are used to manage the analysis templates and their dependencies. An example of one of these dependency matrices is shown by Figure 12. Each column corresponds to a particular analysis template, while rows are used to show all of the ports modeled in the VRA. The version of MagicDraw used for this thesis (17.0.4) allows these matrices to be set up such that users can easily double-click in a cell to create a dependency between an analysis template and a port. This makes the process of capturing analysis templates in a SysML model much more straightforward than in previous efforts (Branscomb, 2012). A set of analysis templates can pre-defined for each VRA, as the same set of interfaces can be reused across many different analysis applications. Consequently, this also allows the possibility for domain models to be reused across multiple analyses.



	ICHEVFuelEff [ICHEVAr…	ICHEVModelicaTest [IC…	ICHEVPerf [ICHEVArchit…
Environment			
Vehicle	9	5	
Chassis			
ICHEVVeh			
Plant and Control Elements			
Power	9	5	
Electrical		1	
Propulsion	7	4	
Engine	4	2	
Exhaust			
PrpPlnt	2	1	
+airflow : ICHEVArchitectur…			
+climt : ICHEVArchitecture:…			
+engCha : ICHEVArchitectu…			
+engDrv : ICHEVArchitectu…	↙		
+engEle : ICHEVArchitectur…		↙	
+engThrmlMng : ICHEVArch…	↙		

Figure 12. Analysis template dependency matrix

4.1.4 Specialized Analysis Architecture

A specialized version of the Vehicle Reference Architecture is created in SysML for each vehicle analysis that needs to be run. This is referred to as the Specialized Analysis Architecture. The Specialized Analysis Architecture has two components—a specialized copy of the VRA, and the analysis artifacts needed to define it. The specialized copy of the VRA includes only the subset of ports specified by a particular analysis template. Ports that will not be used to run the analysis are excluded so that the Specialized Analysis Architecture more accurately reflects the analysis models to be created. A custom plug-in for MagicDraw is used to generate this Specialized Analysis Architecture automatically by copying the VRA and deleting any ports that do not have dependencies to the analysis template specified. Additionally, any domain elements which have all of their ports and parts deleted are also removed from the analysis architecture, so that only the relevant domains are included. An example of the transformation from the VRA to a Specialized Analysis Architecture can be seen in the transformation from Figure 13 (the vehicle reference architecture) to Figure 14 (the specialized analysis architecture). In this example, only the Engine and Transmission models are assumed to be needed for the analysis. All extraneous ports and unneeded domain models are automatically removed when creating the Specialized Analysis Architecture.

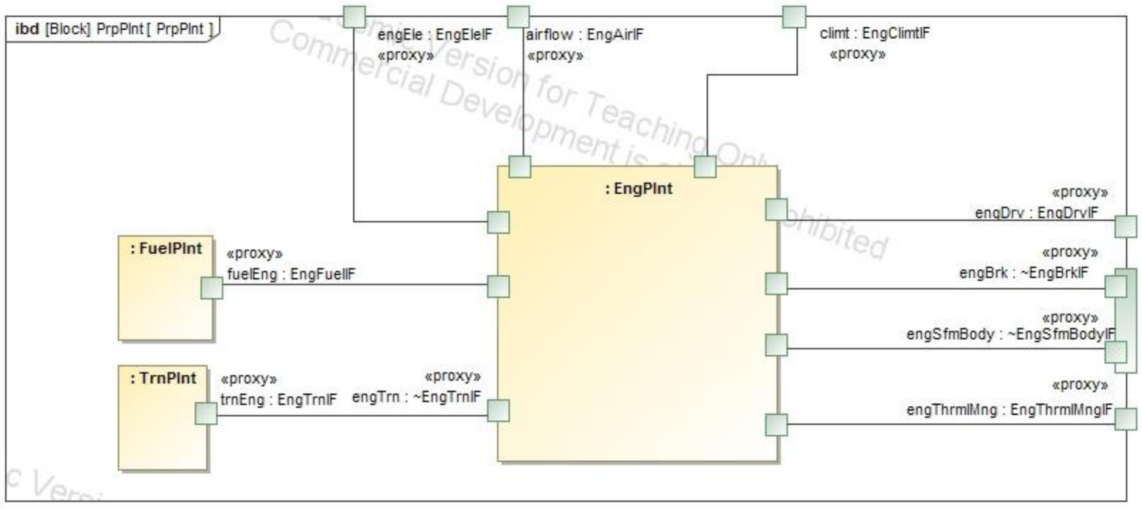


Figure 13. Propulsion plant IBD in VRA

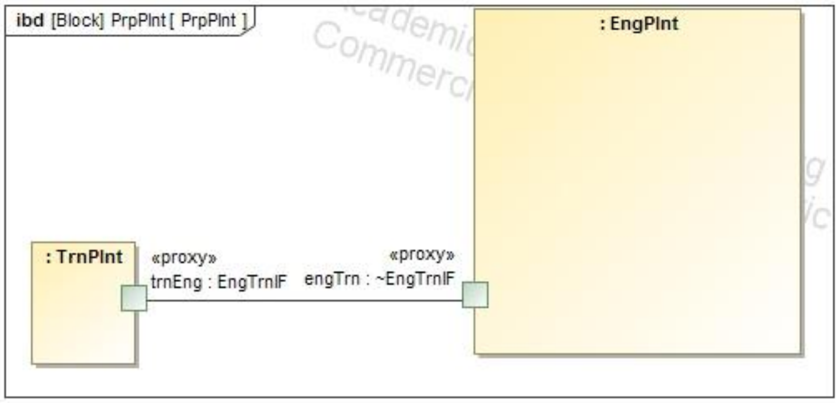


Figure 14. Propulsion plant IBD in Specialized Analysis Architecture

To fully specialize the analysis architecture, it is also necessary to import the unique set of control signals that will be needed to run a particular analysis. Placeholder proxy ports, as shown in Figure 7, are modeled on plant and control elements such that they can be specialized once this set of control signals is known. However, the methods and standards for specializing these ports are not yet finalized, and as a result, are not considered as part of this work.

Once the specialized analysis architecture is created in SysML, it is used to generate a set of blank model templates for Modelica. A representation of the complete Modelica vehicle model is modeled alongside the VRA in SysML using stereotypes from the open source SysML-Modelica profile (Object Modeling Group, 2012). The plugin used by (Branscomb, 2012) to generate Modelica templates from MagicDraw is modified to accommodate the port nesting introduced in Section 4.1.1.

To map the relationships between the VRA and the Modelica elements, association blocks are used to map the correspondence between a SysML element and its Modelica representation (modeled using the SysML-Modelica stereotypes). Note that these relationships are only modeled for plant elements; control elements are intended to

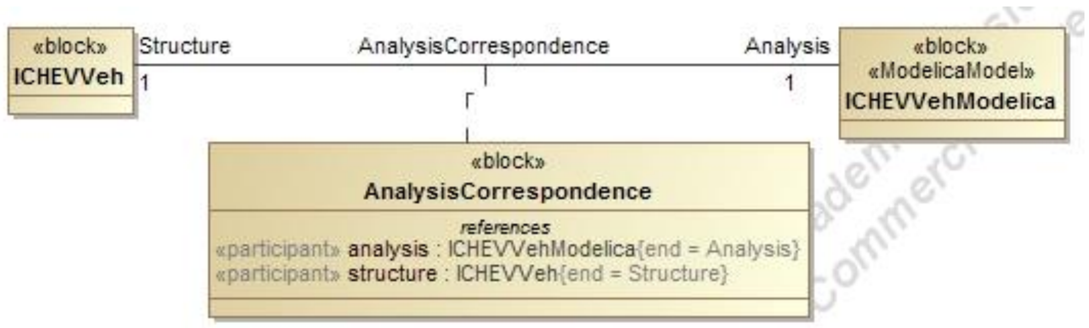


Figure 15. Analysis correspondence between VRA and Modelica model

be modeled in Simulink. An example of one of these correspondences for the complete vehicle model is shown in Figure 15. The VRA is shown by the *ICHEVVeh* block on the left; its Modelica counterpart is shown by the *ICHEVVehModelica* block on the right, which also has a <<*ModelicaModel*>> stereotype. The *AnalysisCorrespondence* association block is used to map the correspondence between the two and their part properties.

These correspondences are modeled at each level of the VRA hierarchy so that the correct part property hierarchy can be traced and maintained. Using IBDs for each association block, the relationships between a system's part properties is modeled using connectors. Figure 16 shows a simplified example of this for the power system owned by the vehicle. The Modelica model is shown by the *analysis:ICHEVVehModelica* element, while the VRA model is shown by the *structure:ICHEVVeh* element.

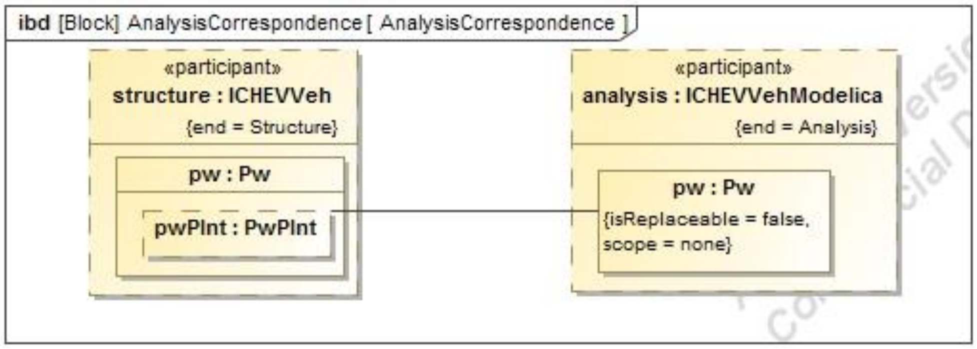


Figure 16. Analysis correspondence IBD

The second component of the specialized analysis architecture is the set of analysis artifacts which describes the analysis that is to be performed. This includes information about what kind of vehicle analysis will be run, the requirements and specifications for the domain-specific and vehicle models, the plans for validating and verifying the analysis results, and documentation of the analysis results themselves. These artifacts are modeled alongside the SysML VRA, such that the relationships and ownership of particular sets of information can be formally captured. This approach allows a repository of analysis architectures and the information about how and why they were created to be stored within a SysML model.

To fully understand these analysis artifacts and the information they contain, it is also necessary to understand how information is built up throughout the model development process, and who has the ultimate authority to *author* pieces of that information. To address this, tools were developed to add functionality to Views and Viewpoints in SysML.

4.1.5 Views and Viewpoints

In SysML, Views and Viewpoints can be used to model the perspectives of different stakeholders and their interests. A viewpoint describes a particular perspective of interest to a set of stakeholders, while a view is a stereotyped package that is said to conform to a particular viewpoint (Friedenthal, et al., 2012). While viewpoints in SysML include properties to identify the stakeholders, their concerns, and even the methods used to establish the view, there is no functionality currently associated with these two SysML elements.

A group at OMG is attempting to establish a standardized approach for generating views from SysML models, but there is no established timeline for its adoption into the SysML specification ("Auto-View Generation Working Group Wiki", 2013). However, by taking advantage of external software tools and SysML's inherent extensibility, customized approaches can be developed to add such functionality. As mentioned in Section 2.3, work has been done by groups at JPL, for instance, to allow for the generation of both web- and document-based tools using Views and Viewpoints in SysML (Delp, et al., 2013).

Our approach can also support the generation of a user interface from a SysML model. In this approach, the capabilities of Views and Viewpoints are extended through Java plugins so that they can be used to export stakeholder-specific information about the SysML model to XML files. Viewpoints are used as filters—they use dependencies to point to specific metaclasses or stereotypes that elements must have in order to be exported to XML. Views, which conform to a particular Viewpoint, are used to specify the packages and specific elements that should be filtered. Once these elements have been filtered, they can be exported to an XML file. This XML file is then used to generate a stakeholder-specific user interface, but does not place any constraints on how that interface must be constructed. An Excel user interface is used, as discussed in Section 4.2. In this way, only the information considered pertinent to that stakeholder must be processed by the user interface; extraneous information is excluded. An example of this use of Views and Viewpoints is shown in Figure 17.

In Figure 17, the *InstanceInfo* viewpoint has dependencies to three metaclasses: *InstanceSpecification, Slot,* and *InstanceValue*. This viewpoint can be used to export any SysML elements that are derived from these metaclasses. The view shown in this figure, *InstanceView*, conforms to this Viewpoint. In addition, it has a dependency, with an <<*import*>> stereotype, to the *InstanceData* package. Using the software plugins developed for our approach, this *InstanceView* can be used to export all of the instance data contained by the *InstanceData* package to an XML file.

The advantage of this approach, which may not be entirely obvious from this trivial example, is that it leverages SysML's visual capabilities to allow users to export specific collections of information from the model. Rather than modifying Java or Python scripts to change how information gets exported, users need only drag a few elements onto a diagram and create dependency relationships between them. Several additional capabilities are added to make this use of views and viewpoints more intuitive to SysML users; these are discussed in more detail in Appendix A.

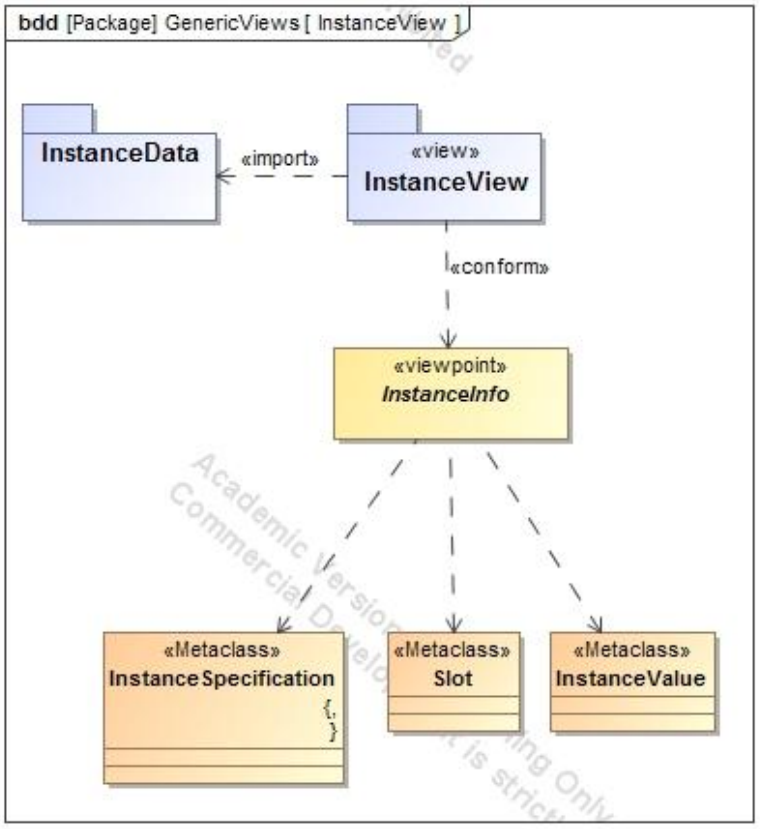


Figure 17. View/Viewpoint definition for exporting instance data

4.2 User Interface

One issue that arises when using a SysML model to handle the many different analysis artifacts is that SysML tools, such as MagicDraw, are not particularly efficient when it comes to entering, modifying, and viewing large sets of information. To develop an effective user interface, it is necessary to first identify the capabilities that are most desirable to end-users unfamiliar with model-based systems engineering and its related tools. From there, existing solutions can then be evaluated and considered, or a custom solution can be created to meet the needs of the various stakeholders being considered.

4.2.1 Desired Capabilities for a User Interface

There are several characteristics of an effective user interface that were identified. Although SysML tools like MagicDraw meet some of these requirements on their own, the costs to train users to use SysML, to modify existing workflows and processes, and to purchase licenses can seem prohibitively high if the benefits of doing so are not immediately obvious. To address these concerns, a set of desired capabilities were outlined that an effective user interface should possess. These capabilities are presented below in order of their relative importance to our approach.

Interface with SysML model

The first capability desired for the user interface is that it be capable of interacting with the SysML model discussed in Section 4.2. This means allowing users to view information about the model, modify it, or add new information. Although SysML tools such as MagicDraw obviously provide the ability to create and modify SysML models, this requires that the users have a sufficient understanding of SysML and its semantics. Otherwise, allowing large numbers of users to interact with the same SysML model directly can introduce as much inconsistency as traditional document-based approaches if users misuse SysML elements or if user permissions are not strictly regulated. MagicDraw also is not particularly efficient when entering large amounts of data. Because analysis artifact data is stored as instances in the VRA SysML model, it can be extremely tedious to view and edit such data without specialized tools.

Familiar to stakeholders

Another capability desired is that the user interface should be familiar to as many stakeholders using it as possible. Training and licenses for SysML tools can be expensive, particularly when needed for many different teams of engineers. By providing a user interface that is already familiar to stakeholders, or at least simple enough to learn without extensive training, many of these costs can be significantly reduced.

Customized for stakeholders

In addition to being familiar to stakeholders, an ideal user interface should also be customizable for particular individuals or groups of stakeholders. These stakeholders may need to view different sets of information to perform their tasks and may need to interact with the SysML model in different ways. It is important that the user interface selected can be customized to these different needs.

Enumerated lists

The ability for users to select from a list of options is a seemingly minor yet immensely important capability. Because creating full-vehicle simulations is a multidisciplinary process, it is important that the same set of semantics be used across different teams of domain experts when possible. By limiting large groups of users to the same vocabulary and set of options where possible, inconsistencies arising out of miscommunication or misinformation can be reduced significantly. This can be accomplished using any number of common graphical user interface elements, such as dropdown menus, radio buttons, or checkboxes.

Consistency checking

It is also important that the user interface have some notion of automated or semi-automated consistency checking in order to make sure that the inputs provided by different stakeholders are consistent. Parsing and comparing long strings of text is obviously beyond the scope of this research, but it would be advantageous to check if different options selected from dropdown menus or checkboxes are consistent with one another. For instance, if a user specifies that an analysis should examine attributes typically associated with a fuel efficiency vehicle model, the user interface would ideally prevent the end-user from selecting unrelated options associated with cabin comfort models.

4.2.2 Excel Template Engine

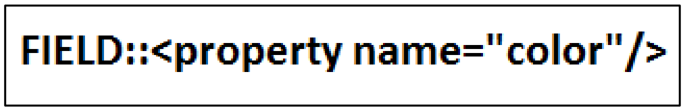
Excel is already used extensively by both domain and system engineers during the analysis planning process. To ensure a minimal impact on current workflows, it was considered to be the most desirable choice for a user interface. However, on its own, it does not have all of the capabilities outlined in Section 4.2.1.

To address some of these concerns, a template engine created for Excel adds the ability for users to auto-generate user-forms based on special templates defined directly in Excel spreadsheets. This template engine was written to be fairly generic, such that if a new kind of user-form is desired or if an existing one needs to be reformatted, it can most likely be done without requiring any VBA or Java code to be updated. Note that only a small number of administrative-level users would be required to create and modify these templates; the majority of the stakeholders introduced in Chapter 3 would only interact with spreadsheets that had been pre-generated for them. In this way, generating these user-forms requires only knowledge of how the template engine and SysML models interact, rather than an extensive programming background and understanding of the underlying code.

The template engine first imports an XML file which is generated by exporting Views from a SysML model, as discussed in Section 4.1.5. This XML file is then mapped to its own spreadsheet in the workbook. On the sheet in which the user-form will be generated, “commands” can be specified in either the first row or first column of the sheet to process XML elements.

These commands can be used to different effects. They can be used to specify which properties from the SysML model should be presented as fields for the user to enter information. Commands can also denote which pieces of information should be presented for users to *view* in a spreadsheet, but not change. In addition, they can be used to specify how these pieces of information should be imported back into the SysML model—such as importing fields as instance data or modifying existing elements. In this way, the notion of *Ownership* versus *Viewership* of information can be built into the user-forms, rather than relying on users to adhere to these rules on their own.

A command has two components. An example of a command is shown by Figure 18. The first component is the *reference*, shown on the right after the two colons. The reference is used to find a particular SysML element by some attribute, whether it be its name, ID, or type (in the case of properties). In the template engine, a reference is formatted as an XML element with an element type and any number of attributes. In the example shown below, the reference points to a “property” whose name is “color”. The second component of a command is the *action*, shown to the left of the colons. The action tells the template engine what to do with the SysML element that it finds. For instance, the action might tell the template engine to create a user field based on a SysML element, to export a row or column as instance data, or to present some information for the user to see while they fill out other fields. In the example presented, the action "FIELD" tells it to create a field using the property pointed to by the reference.



```
FIELD::<property name="color"/>
```

Figure 18. Example of a command in the Excel template engine

In this simple example, the reference points to the *color* property (typed by the *ColorKind* enumeration) defined in Figure 19. Once the command has been processed as a "FIELD", as specified by the action, a dropdown menu will be created in the Excel spreadsheet. The enumeration literals specified in the SysML *ColorKind* enumeration map directly to the choices available in the dropdown menu, as shown in Figure 20.

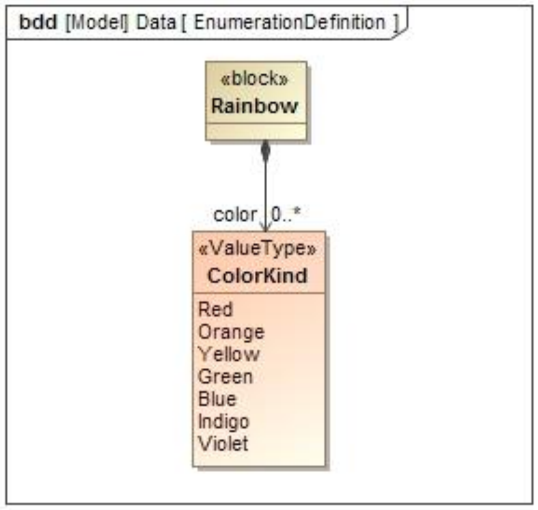


Figure 19. Simple BDD defining a block and enumeration

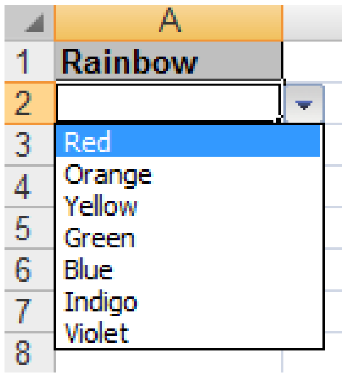


Figure 20. Excel dropdown menu generated by template engine

Although this example is trivial, the template engine can be extremely powerful when used to interact with a complex SysML model. Using these commands in concert with XML files exported from SysML, an administrative-level user can create reusable Excel templates which allow different stakeholders to view and modify a central SysML model. Updating the spreadsheet to reflect changes to the SysML model is as simple as importing the most up-to-date XML file from the model. A more detailed discussion of how this template engine works is presented in Appendix B.

4.2.3 Excel User Interface Additions

The template engine on its own can automatically create the data fields for users to input information, but there are additional capabilities that were added to make the user-interface more accessible and user-friendly. These auxiliary capabilities are presented below, in no particular order.

Multi-selection in dropdown menus

One feature that Excel drop-down lists do not natively support is the ability to select multiple options. This is an extremely important capability to have. Many data

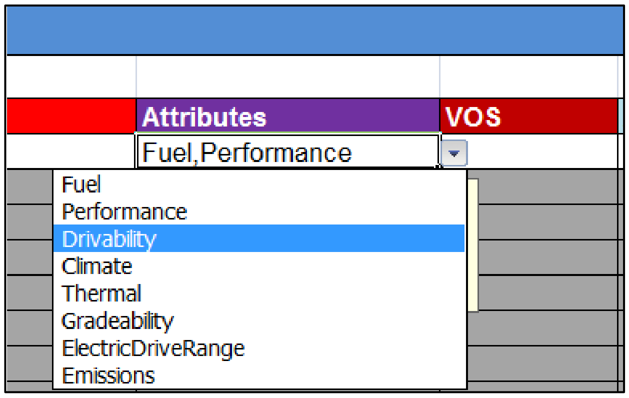


Figure 21. Multi-select capability in Excel

fields in the analysis artifact documents allow, and in some cases require, multiple choices to be selected. When a user selects multiple options, they appear as a comma-separated list in the cell that the user is entering information. This capability is enabled automatically if the dropdown menu created by the template engine is linked to a SysML property with a multiplicity of "*0..**". An example of this is shown by Figure 21, where multiple vehicle attributes have been selected to examine for a particular analysis.

Context-specific dropdown menus

While dropdown menus and enumerated lists can significantly reduce inconsistencies that may arise when planning analyses, it is still possible that certain options selected may contradict one another. To address these concerns, a feature is included in the user interface which can automatically limit the options available to the user. To map how different fields in the user interface relate to one another, dependencies are used in SysML. An example of such a set of dependencies is shown by Figure 22.

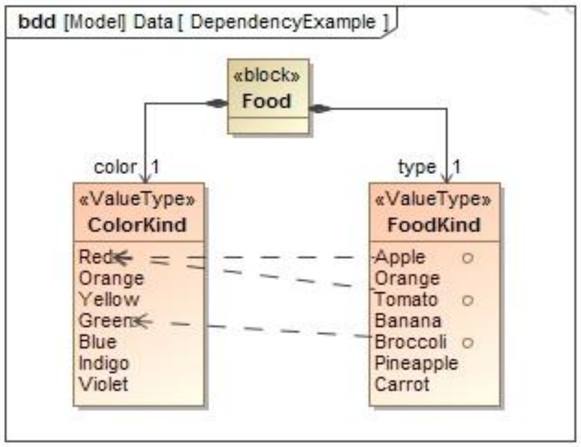


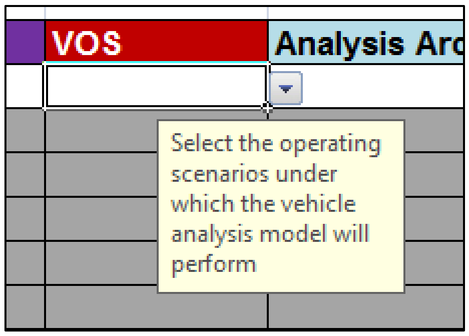
Figure 22. Example of dependencies defined in BDD

In this example, dependency relationships are drawn between enumeration literals—from *Apple* and *Tomato* to *Red*, and from *Broccoli* to *Green*. This implies, for instance, that the ability to select *Apple* is contingent on selecting the *Red* enumeration literal. If another enumeration literal is selected to which does *Apple* not have a dependency, then *Apple* cannot be selected by the user. Note that managing these dependencies in BDDs can quickly become confusing; it is much simpler to add and modify them through dependency matrices in MagicDraw.

Like other elements in the SysML model, these dependencies can be chosen for export to the Excel user interface through the use of Views and Viewpoints. Using this capability, inconsistencies can be prevented by only permitting users to select valid combinations of choices.

Supplemental tooltips

While the added capabilities of dropdown menus can help to bridge the semantic gaps between multidisciplinary teams of domain experts, it is still possible that the Excel user interface may seem ambiguous or confusing to the end-user. Traditionally, sets of



Select the operating scenarios under which the vehicle analysis model will perform

Figure 23. Tooltips in Excel UI

reference documents would be used to define any unclear sets of terms. However, such documents can quickly become obsolete as the user interface changes and evolves over time. Instead, it is more desirable to handle this kind of information using a model-based method.

To do this, this information is maintained in comments in the SysML model. By storing the information in this way, the SysML model can remain the master source of any and all information about the analysis artifact models. These comments are exported to XML from the SysML model with the rest of the information and displayed as tooltips in Excel when the user selects a particular cell. An example of a tooltip that has been generated in this way is shown in Figure 23.

Cell color-coding

The final capability added to the Excel user interface is the ability to visually display which data fields a user can edit, and which are only meant to be viewed. This is a seemingly minor detail, but for users unfamiliar with the template they are using, it is a critical feature. The template engine allows two actions to be specified when creating these templates: *VIEW* and *FIELD*. Data fields which a user can edit are specified by

FIELD and displayed in white, while data fields which are only meant to be viewed are denoted by *VIEW* and are left gray.

CHAPTER 5

APPLYING INFRASTRUCTURE FRAMEWORK TO ANALYSIS PLANNING

This chapter details how the infrastructure framework introduced in Chapter 4 is applied to the analysis planning process presented in Chapter 3. The chapter follows the general progression of the "Analysis Planning" activity diagram shown in Figure 2. In addition, this chapter includes a brief discussion about how the infrastructure framework can be applied to the model development process. Each section discusses the user interface and underlying SysML models which pertain to a particular activity. The focus of this chapter is primarily on analysis planning for *domain* models, as they require the most interdisciplinary coordination. Note that each vehicle system model also has its own set of requirements, specifications, and documentation, but those are not presented here as they are largely the same as their domain model counterparts (with the exception of the integration and verification/validation plans discussed in Sections 3.2.7 and 3.3.2).

There are, however, two components of the infrastructure frameworks which are not discussed explicitly in this chapter. The tools created to use SysML Views/Viewpoints in conjunction with the Excel template engine are used at every step in this process. The combination of these two tools allows users to export View-specific XML files from SysML and use them to produce a user interface in Excel. Their functionality is independent of the activity to which they are applied. As a result, the details of the implementation for SysML Views/Viewpoints and the Excel template engine are left to Appendix A and Appendix B, respectively.

5.1 Creating a Draft of the System Analysis Application Plan

The first step in planning a system analysis application plan is defining the full set of analyses that need to be performed—the set of analysis applications. All of the analysis applications for a given system study must be defined together, so that any similarities between them can be identified. Identifying and understanding these similarities can potentially allow a single vehicle model to be used to address multiple analysis applications.

To apply the infrastructure framework to the process of drafting a system analysis application plan, a SysML model is used to capture the information which must be defined for each analysis application. This SysML model is shown in the BDD in Figure 24. The diagram is used to define several different components of an analysis application which are owned by the *AnalysisApplication* block. Once the SysML model has been defined, it can be used to store analysis application data by creating instances of the *AnalysisApplication* block. There are four aspects of this model worth noting.

First, value properties are used to represent fields in which users can write textual information. In this case, these fields are the name of the analysis application (*name*), a short description of it (*description*), and the name of the engineer who will lead the model development effort (*lead*). Second, a set of enumerations and enumeration literals is used to define the system design study. These enumerations are used to define several different aspects of a given analysis application, including the different vehicle attributes to examine (*AttributeKind),* the type of analysis being considered (*AnalysisKind*), possible vehicle top hats (*VehicleKind*), any subsystems which are unique for the vehicle program (*UniqueSubsystemKind*), the set of operating scenarios that could be modeled

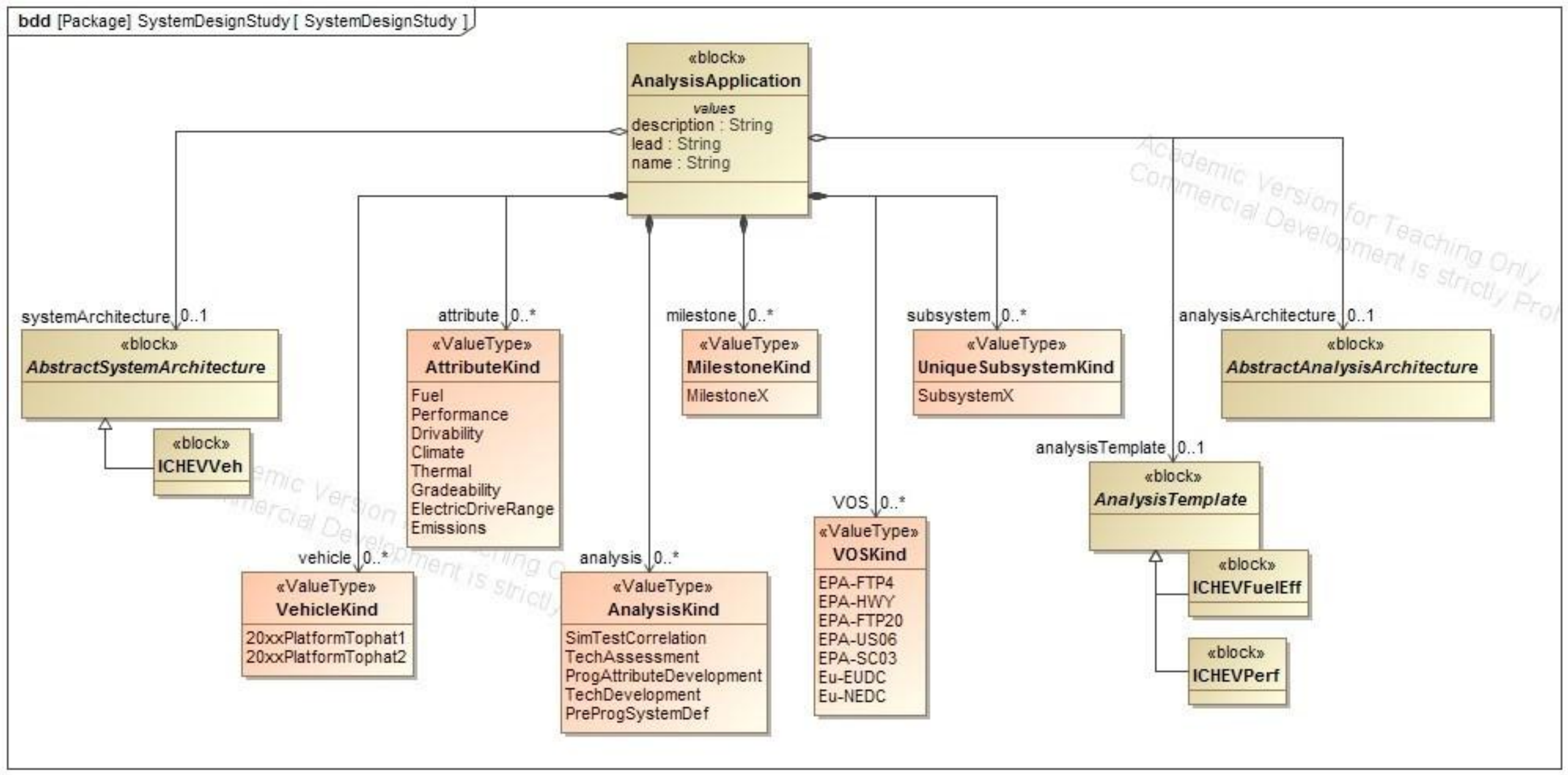
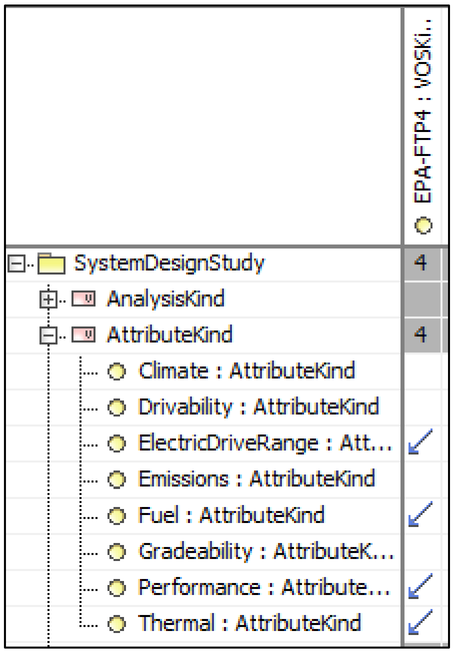


Figure 24. Analysis application SysML model

(*VOSKind*), and the possible development milestones to target (*MilestoneKind*). The enumeration literals owned by each of these elements are used to create dropdown menus in the Excel user interface.

The third component of the analysis application SysML model is the set of dependencies between the elements. These dependencies are used to ensure that the options available for a user to select in the Excel user interface are all consistent with one another, as discussed in Section 4.2.3. The most important selections are the *AbstractSystemArchitecture* and *AnalysisTemplate* chosen. Each analysis template is only valid for a single system architecture, so it is important to ensure that the user selects a valid option. These dependencies are more easily managed through a dependency matrix created for each element, rather than through a BDD like the one shown in Figure 24. A condensed example of a VOS dependency matrix is shown in Figure 25 for the "EPA-FTP4" operating scenario. This example shows that the ability to select that operating scenario will depend on *ElectricDriveRange*, *Fuel*, *Performance*, *Thermal*, or some combination of these attributes being selected. Any other choices will prevent the user from selecting "EPA-FTP4".

The fourth and final aspect of the analysis application model worth noting is the reference properties owned by the *AnalysisApplication* block. These are shown by the white diamond relationships between *AnalysisApplication* and the abstract blocks (which have italicized names) in Figure 24. In this model, there are three such reference properties. First, it references *AbstractSystemArchitecture* block. Any VRA modeled in SysML must be a specialization of this block. Only one VRA has currently been modeled in SysML, which is shown by the *ICHEVVeh* block.



	EPA-FTP4 : VOSKi...
SystemDesignStudy	4
AnalysisKind	
AttributeKind	4
Climate : AttributeKind	
Drivability : AttributeKind	
ElectricDriveRange : Att...	↙
Emissions : AttributeKind	
Fuel : AttributeKind	↙
Gradeability : AttributeK...	
Performance : Attribute...	↙
Thermal : AttributeKind	↙

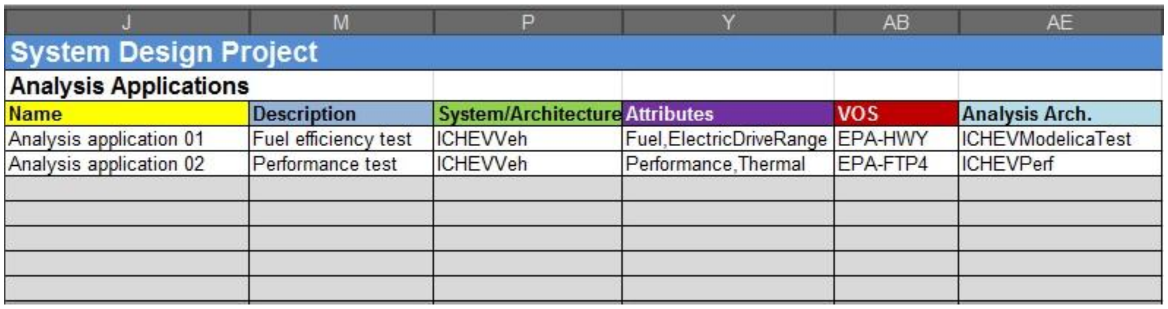
Figure 25. Dependency matrix for a VOS choice

Second, the *AnalysisApplication* block references the set of analysis templates defined in SysML, given by *AnalysisTemplate*. Again, any analysis template defined in SysML must be a specialization of this block in order for it to be referenced by an analysis application. In this case, two sample analysis templates have been created for fuel efficiency and performance simulations—*ICHEVFuelEff* and *ICHEVPerf*, respectively. These analysis templates are not complete; the goal of our approach is to create the tools and approach necessary to define them, rather than to define them ourselves. Note that a given analysis template is only valid for a single VRA. To make sure that the analysis template selected is valid for the system architecture chosen, dependency matrices like the one shown in Figure 25 can be used to map these relations.

Finally, *AnalysisApplication* also has a reference property to *AbstractAnalysisArchitecture*. Using the system architecture and analysis template selected by a user, an analysis architecture can be created as a specialization of the

AbstractAnalysisArchitecture block for each analysis application (as discussed in 4.1.4). Analysis architectures are not generated until the other data fields for an analysis application have been defined, so this property is left as a placeholder in the analysis application SysML model.

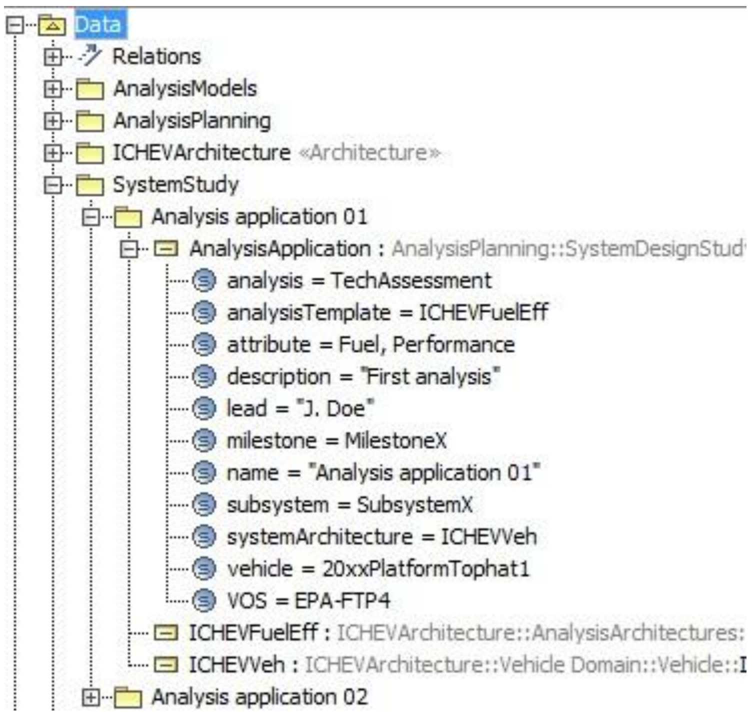
By exporting this SysML model to XML, and loading that XML file through our Excel template engine, a user interface similar to the one shown in Figure 26 can be generated. This user interface has been truncated to show only the most important fields, including the name, system architecture, vehicle attributes, VOS, and analysis architecture for each analysis application. Note that in the user interface shown, each row corresponds to one analysis application. System engineers can define as many analysis applications here as needed. Once the full set of analysis applications has been defined, they can be imported back into SysML.



J	M	P	Y	AB	AE
System Design Project					
Analysis Applications					
Name	Description	System/Architecture	Attributes	VOS	Analysis Arch.
Analysis application 01	Fuel efficiency test	ICHEVVeh	Fuel,ElectricDriveRange	EPA-HWY	ICHEVModelicaTest
Analysis application 02	Performance test	ICHEVVeh	Performance,Thermal	EPA-FTP4	ICHEVPerf

Figure 26. Analysis application spreadsheet

Importing the analysis application data into SysML creates a package and instance of the *AnalysisApplication* block for each row defined in Figure 26. For a given system analysis application plan, all of the individual analysis application packages are stored in one parent package. All of the data and models associated with an individual analysis application are stored within that application’s package. This can be seen by Figure 27 below.



```
Data
  Relations
  AnalysisModels
  AnalysisPlanning
  ICHEVArchitecture «Architecture»
  SystemStudy
    Analysis application 01
      AnalysisApplication : AnalysisPlanning::SystemDesignStud
        analysis = TechAssessment
        analysisTemplate = ICHEVFuelEff
        attribute = Fuel, Performance
        description = "First analysis"
        lead = "J. Doe"
        milestone = MilestoneX
        name = "Analysis application 01"
        subsystem = SubsystemX
        systemArchitecture = ICHEVVeh
        vehicle = 20xxPlatformTophat1
        VOS = EPA-FTP4
      ICHEVFuelEff : ICHEVArchitecture::AnalysisArchitectures:
      ICHEVVeh : ICHEVArchitecture::Vehicle Domain::Vehicle::I
    Analysis application 02
```

Figure 27. System study package structure

Figure 27 shows what the information from Figure 26 looks like when imported back into SysML. A package is created for each row from the Excel spreadsheet, and an instance of the *AnalysisApplication* block from Figure 24 is created to store the information entered by a user. In addition, instances are created for the *ICHEVVeh* and *ICHEVFuelEff* blocks and stored in the slots for the *systemArchitecture* and *analysisTemplate* properties, respectively. In this way, a formal relationship between the analysis application and the VRA is created. By creating this relationship, traceability between an analysis application and the appropriate Vehicle Reference Architecture can be maintained within the SysML model.

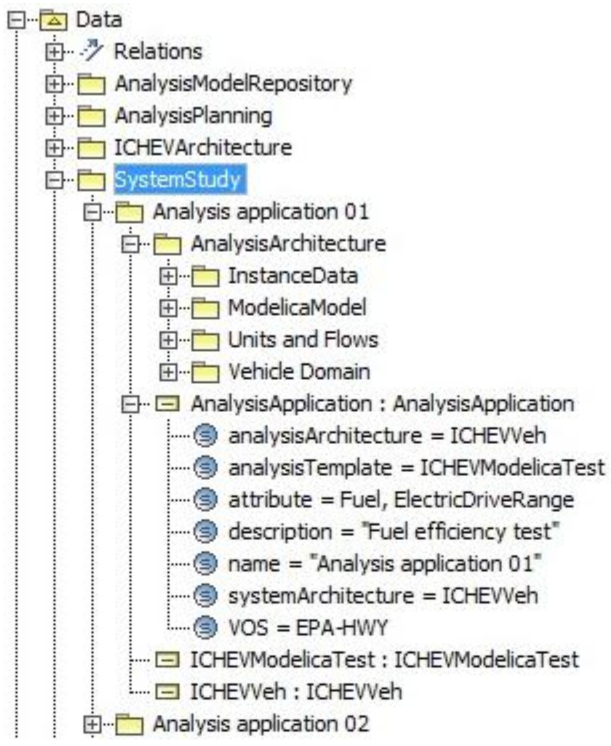
As shown in Figure 2, the set of control signals for each analysis application should be negotiated and defined concurrently with this process. However, the mechanisms and standards for doing so are not finalized, so that step is omitted from this example. Consequently, the next step in the process is to generate a specialized analysis architecture for each analysis application.

5.2 Creating a Specialized Analysis Architecture

Once the system engineers have defined each analysis application, their information can be used in SysML to generate a specialized analysis architecture simply by right-clicking on any analysis application package (such as *Analysis application 01* in Figure 27) and selecting “VMA Analysis Planner > Generate analysis architecture”. This function parses the instance data and confirms that a valid system architecture and analysis template have been selected by the user. If this is true, then the specialized analysis architecture is created. This is an automated, multi-step process, so these steps will be briefly presented in order.

First, a copy of the chosen VRA is made in the analysis application package. The plant and control elements are then checked to see whether they, or any of their owned parts or ports, have dependencies to the analysis template selected. Elements that are not needed by a given analysis template are deleted, so only pertinent elements are left. This also includes deleting the elements with Modelica stereotypes like the *ICHEVVehModelica* element shown in Figure 15, which can be used to create a Modelica model template. By tracing through the correspondences like the one shown in Figure 15, the SysML vehicle model and Modelica vehicle model can be reduced to the same analysis architecture. Note that the copy of the top-level vehicle block (*ICHEVVeh*) is also automatically modeled as a specialization of the *AbstractAnalysisArchitecture* block from Figure 24.

Once this copy has been created and extraneous elements have been removed, an instance of the analysis architecture is created. This serves two purposes. First, the instance created can be stored in the *analysisArchitecture* slot, as shown in Figure 28, because of the specialization relationship given to it in the previous step. This allows the analysis application to have an explicit relation to its analysis architecture. Second, creating an instance of the entire analysis architecture also creates instances of the full set of analysis artifact documents (such as requirements, specifications, and documentation) which are needed to plan and develop the remaining domain elements. System engineers can then export these instances to XML as needed and allow domain engineers to populate them from the Excel user interface.



```
Data
  Relations
  AnalysisModelRepository
  AnalysisPlanning
  ICHEVArchitecture
  SystemStudy
    Analysis application 01
      AnalysisArchitecture
        InstanceData
        ModelicaModel
        Units and Flows
        Vehicle Domain
      AnalysisApplication : AnalysisApplication
        analysisArchitecture = ICHEVVeh
        analysisTemplate = ICHEVModelicaTest
        attribute = Fuel, ElectricDriveRange
        description = "Fuel efficiency test"
        name = "Analysis application 01"
        systemArchitecture = ICHEVVeh
        VOS = EPA-HWY
      ICHEVModelicaTest : ICHEVModelicaTest
      ICHEVVeh : ICHEVVeh
    Analysis application 02
```

Figure 28. Analysis application package with an analysis architecture

The final package structure for the analysis application is shown in Figure 28. Note that the copied VRA is stored in the *Vehicle Domain* and *Units and Flows* packages. The exportable Modelica elements and instance data are stored in the *ModelicaModel* and *InstanceData* packages, respectively. This *ICHEVVeh* instance also contains additional analysis artifacts, such as domain and vehicle model requirements. The following sections detail how these different analysis artifacts are used to support the analysis planning process.

5.3 Domain Model Requirements

Once the specialized analysis architecture has been created, the team of system engineers can then define the requirements for all of the necessary domain models. Many of these requirements should be common across all domains, such as operating systems to use, so a generic domain requirements model serves as the foundation for more specialized models. This model is shown in Figure 29.

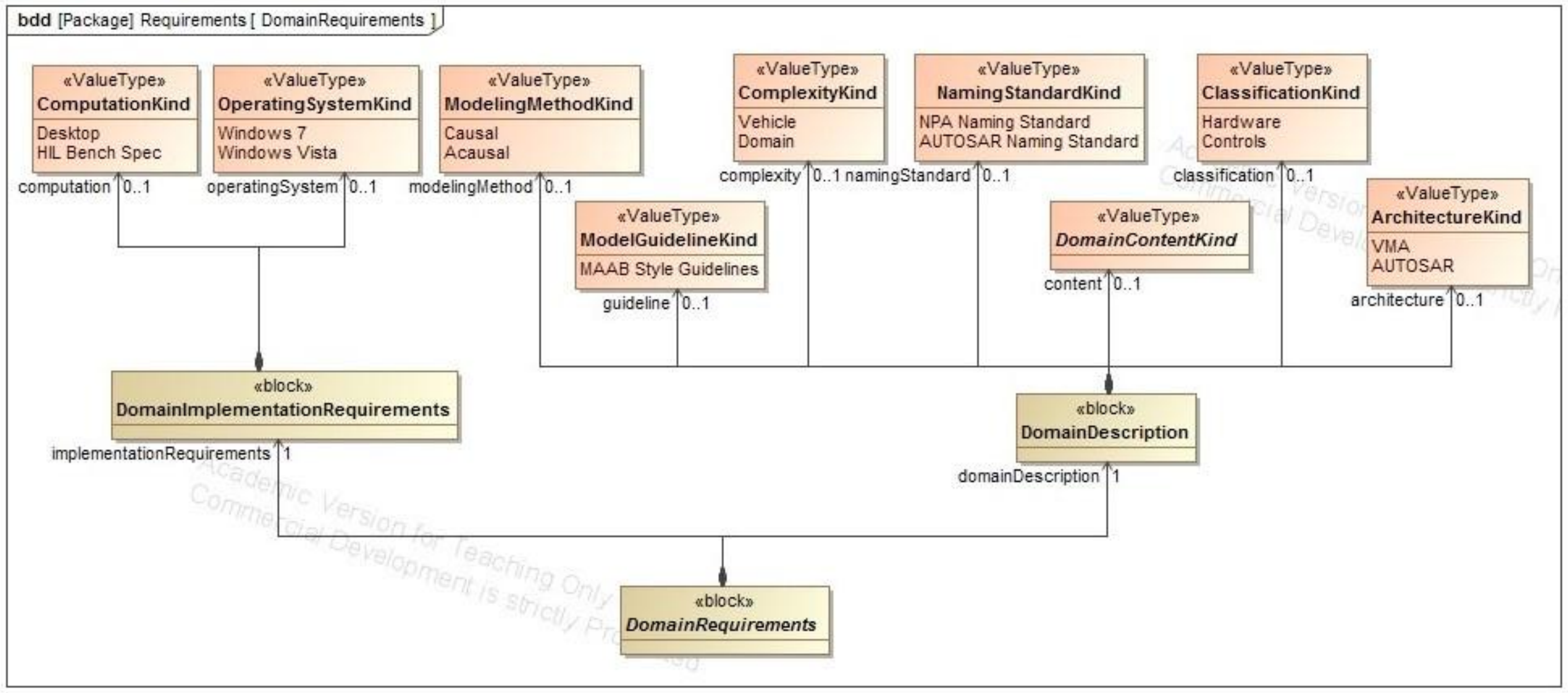
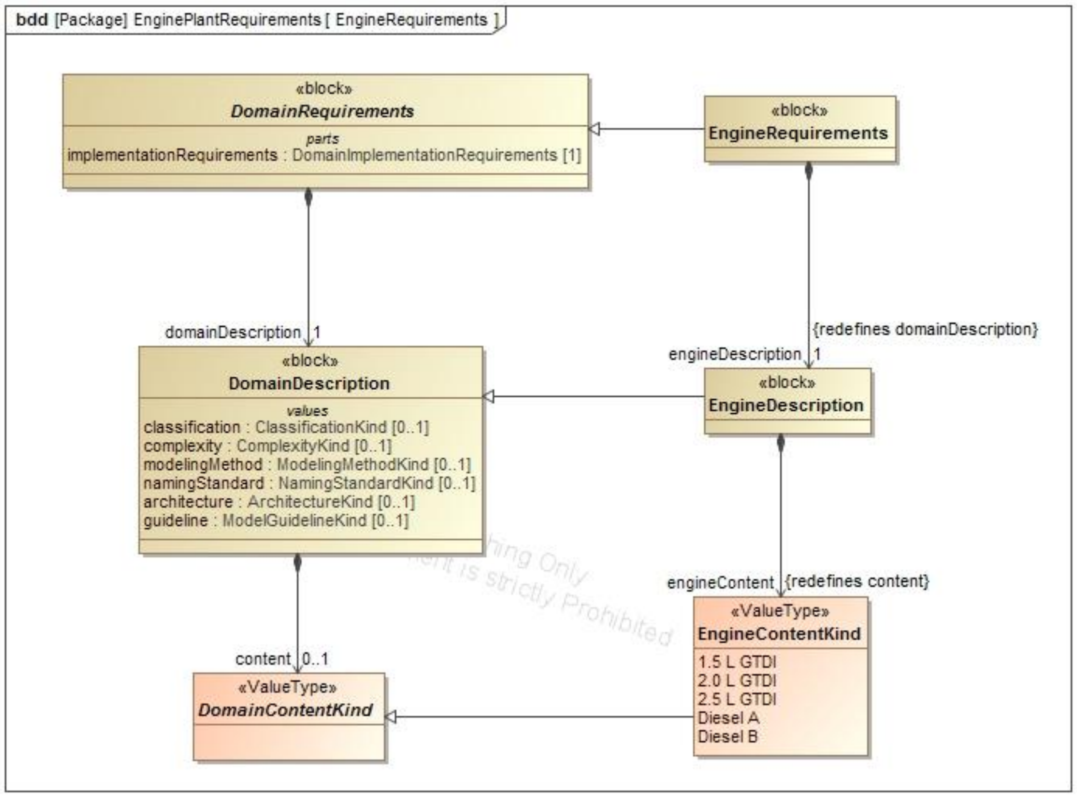

bdd [Package] Requirements [DomainRequirements]
«ValueType» ComputationKind
Desktop
HIL Bench Spec
«ValueType» OperatingSystemKind
Windows 7
Windows Vista
«ValueType» ModelingMethodKind
Causal
Acausal
«ValueType» ModelGuidelineKind
MAAB Style Guidelines
«ValueType» ComplexityKind
Vehicle
Domain
«ValueType» NamingStandardKind
NPA Naming Standard
AUTOSAR Naming Standard
«ValueType» ClassificationKind
Hardware
Controls
«ValueType» DomainContentKind
«ValueType» ArchitectureKind
VMA
AUTOSAR
computation 0..1
operatingSystem 0..1
modelingMethod 0..1
guideline 0..1
complexity 0..1
namingStandard 0..1
classification 0..1
content 0..1
architecture 0..1
«block» DomainImplementationRequirements
«block» DomainDescription
implementationRequirements 1
domainDescription 1
«block» DomainRequirements

Figure 29. Generic requirements model

This model follows essentially the same format as the analysis applications model shown in Figure 24; the enumerations shown are used to create dropdown menus in an Excel spreadsheet. One difference, however, is that the top-level block—*DomainRequirements*—is abstract. For a particular domain to have its own set of requirements, it must specialize the abstract *DomainRequirements* block and redefine any abstract or generic enumerations to their domain-specific counterparts. For instance, the *DomainContentKind* block is intended to define particular domain variants, such as an inline 4-cylinder engine or a 6-speed manual transmission. This block needs to be redefined for each specialized set of domain model requirements, but other blocks in the model can also be redefined as needed. An example of this specialization and redefinition is shown in Figure 30.

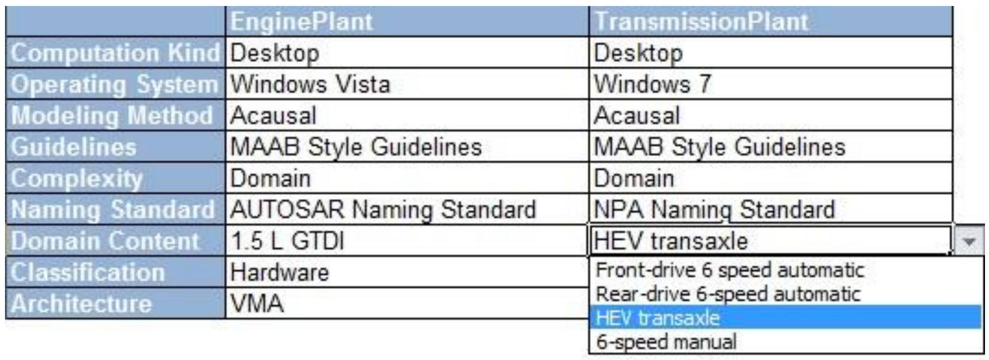
In this example, a specialized set of engine requirements is being modeled. The *EngineRequirements* block is a specialization of *DomainRequirements,* and redefines the *domainDescription* property to own the *EngineDescription* block. This block, also a specialization of *DomainDescription,* then redefines the *DomainContentKind* enumeration in favor of *EngineContentKind*. This enumeration contains a list of different engine variants that could be modeled. Using this pattern, requirements can be specialized for each domain from a common, generic model.



bdd [Package] EnginePlantRequirements [EngineRequirements]
«block» DomainRequirements
parts
implementationRequirements : DomainImplementationRequirements [1]
«block» EngineRequirements
domainDescription 1
{redefines domainDescription}
engineDescription 1
«block» DomainDescription
values
classification : ClassificationKind [0..1]
complexity : ComplexityKind [0..1]
modelingMethod : ModelingMethodKind [0..1]
namingStandard : NamingStandardKind [0..1]
architecture : ArchitectureKind [0..1]
guideline : ModelGuidelineKind [0..1]
«block» EngineDescription
engineContent {redefines content}
«ValueType» EngineContentKind
1.5 L GTDI
2.0 L GTDI
2.5 L GTDI
Diesel A
Diesel B
content 0..1
«ValueType» DomainContentKind

Figure 30. Specialized engine requirements model

Once these requirements models are created, they are used to generate a comprehensive user interface that allows the team of system engineers to view and edit all of them at the same time. An example of this interface with completed user fields is shown in Figure 31. Note that other than the leftmost column, the UI shown was generated automatically using the template engine in Excel. Here, only the requirements for the Engine Plant and Transmission Plant are shown. However, this approach can be used to show the domain requirements models for the entire analysis architecture simultaneously. By presenting the domain requirements in this way, system engineers are provided with an interface that promotes a more holistic view of the vehicle system. Once this information has been filled out by the system engineers, it can be imported back into SysML, which serves as the master source of information for analysis planning.



	EnginePlant	TransmissionPlant
Computation Kind	Desktop	Desktop
Operating System	Windows Vista	Windows 7
Modeling Method	Acausal	Acausal
Guidelines	MAAB Style Guidelines	MAAB Style Guidelines
Complexity	Domain	Domain
Naming Standard	AUTOSAR Naming Standard	NPA Naming Standard
Domain Content	1.5 L GTDI	HEV transaxle
Classification	Hardware	
Architecture	VMA	

Front-drive 6 speed automatic
Rear-drive 6-speed automatic
HEV transaxle
6-speed manual

Figure 31. Domain requirements spreadsheet

5.4 Domain Model Specifications

Once the domain model requirements have been specified at the system level, domain engineers can create specifications for the model that they will create. Like the analysis applications and domain requirements, the domain specifications are modeled in SysML, as shown in Figure 32. Note that unlike the *DomainRequirements* block shown in Figure 29, *DomainSpecification* is not abstract. The domain specification model was left intentionally generic so that domain engineers have some freedom in creating their model specifications. This is needed, in part, because the Excel user interface does not lend itself well to all applications. In many cases, more detailed Word documents or PDF files may be needed to explain the rationale behind certain modeling decisions. To support this, URLs can be used to reference external documents, as shown by the *documentURL* value properties owned by several blocks in Figure 32.

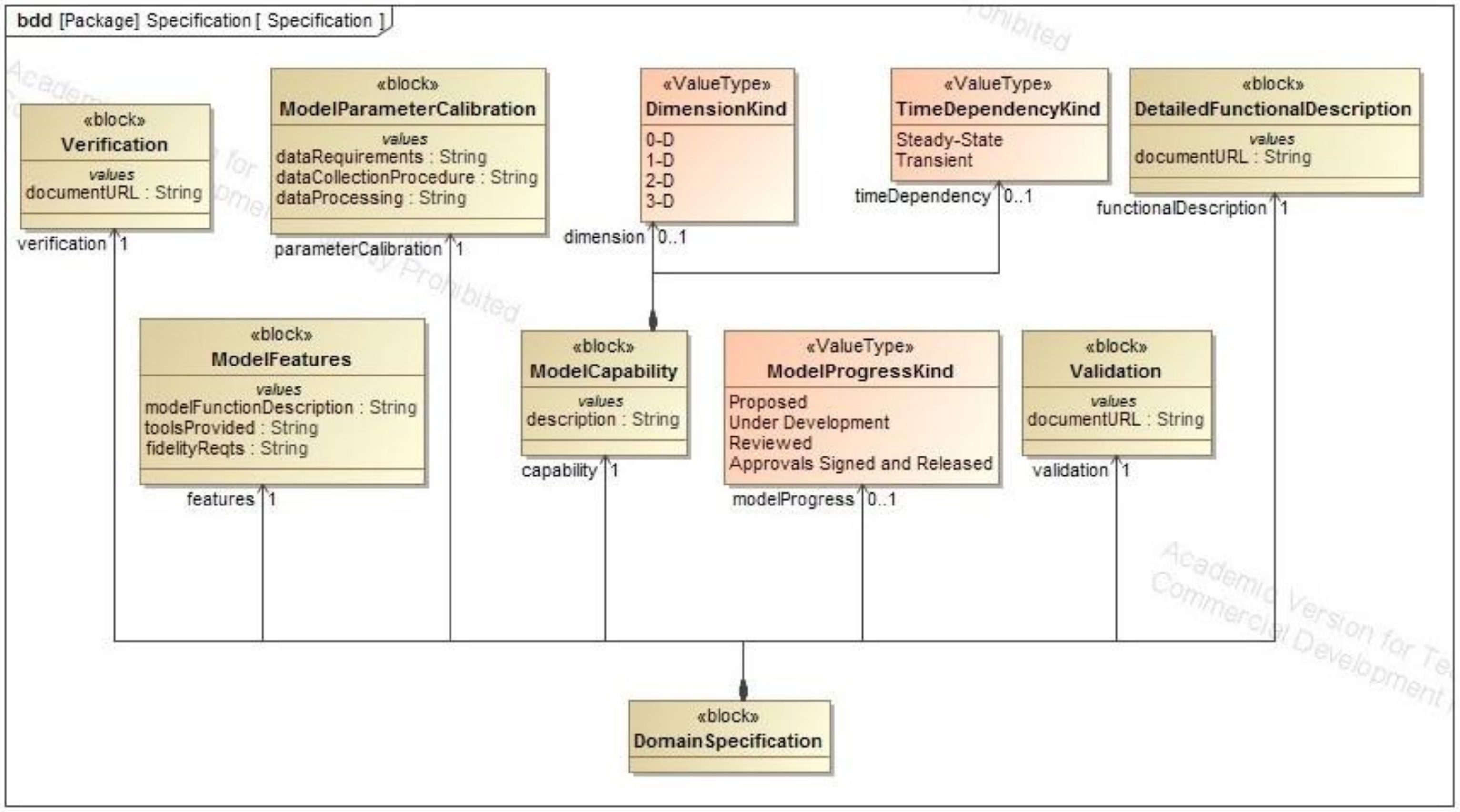
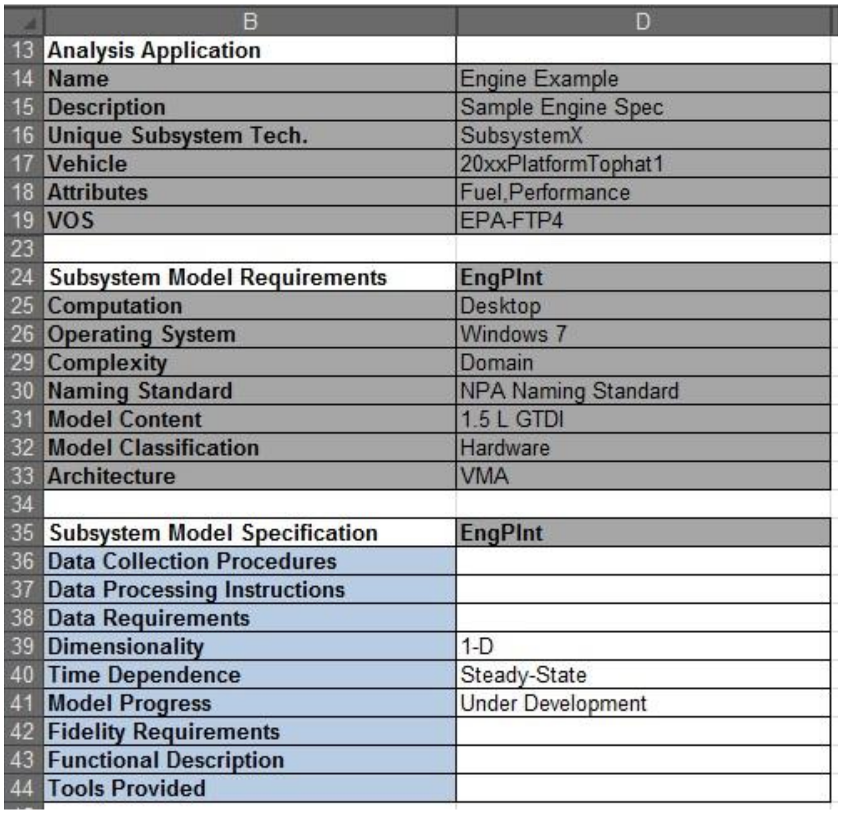


Figure 32. Domain specification BDD

Creating these specifications requires that the domain engineers understand both the analysis application that the model is being developed for and the requirements for their domain model. However, they do not have the authority to directly change any of that information. To address this issue of viewership versus ownership of information, the template engine can visually depict which fields are editable in Excel. This can be seen in Figure 33, which shows the user interface for defining engine plant specifications in Excel. Here, the information defined by the analysis application and domain model requirements is presented for the domain engineer in gray in Column D. The data fields that the user can modify are shown in white. This color-coding can be created automatically using the Excel template engine. The fields shown in blue were created manually to provide users with another visual cue about which fields need to be filled out. Given the information provided, domain engineers can fill out the data fields under the "Subsystem Model Specification" heading and export the data back to SysML.



	B	D
13	Analysis Application	
14	Name	Engine Example
15	Description	Sample Engine Spec
16	Unique Subsystem Tech.	SubsystemX
17	Vehicle	20xxPlatformTophat1
18	Attributes	Fuel,Performance
19	VOS	EPA-FTP4
23		
24	Subsystem Model Requirements	EngPlnt
25	Computation	Desktop
26	Operating System	Windows 7
29	Complexity	Domain
30	Naming Standard	NPA Naming Standard
31	Model Content	1.5 L GTDI
32	Model Classification	Hardware
33	Architecture	VMA
34		
35	Subsystem Model Specification	EngPlnt
36	Data Collection Procedures	
37	Data Processing Instructions	
38	Data Requirements	
39	Dimensionality	1-D
40	Time Dependence	Steady-State
41	Model Progress	Under Development
42	Fidelity Requirements	
43	Functional Description	
44	Tools Provided	

Figure 33. Excel UI for engine model specifications

5.5 Creating Modelica Templates

Once the analysis planning phase is complete and the model specifications have been imported back into SysML, system engineers can use the specialized analysis architecture to generate blank Modelica templates that the domain engineers will use to create their models. This process is as simple as right-clicking on the Modelica model package in the analysis application package and selecting *SysML to Modelica > Generate Modelica.* This process is shown in Figure 34 below. Note that it is currently not possible to generate Simulink templates for control signals in the same way, as they have not been captured in the SysML model yet.

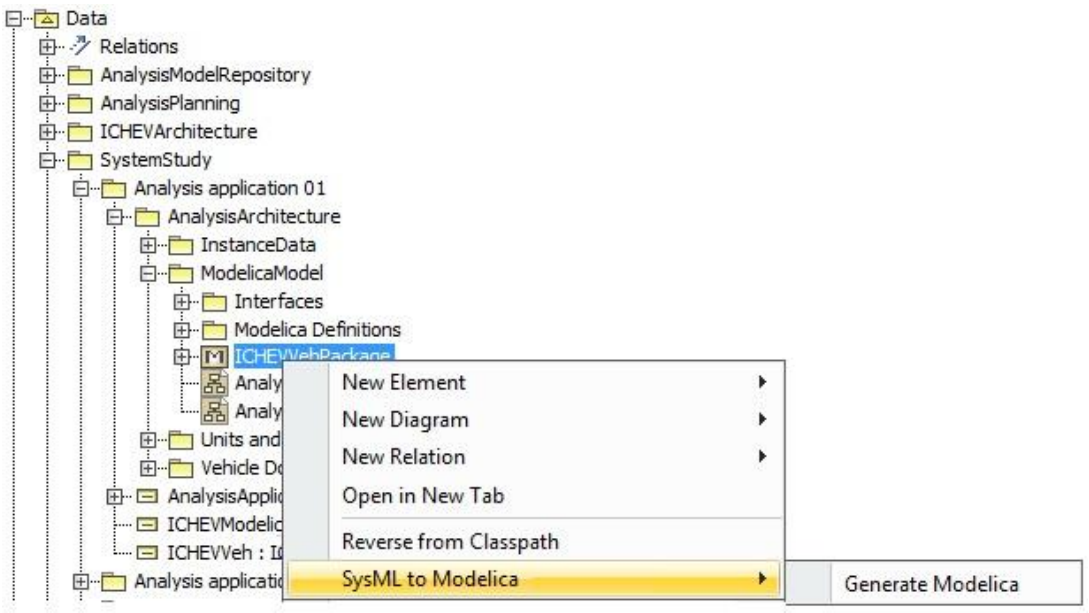
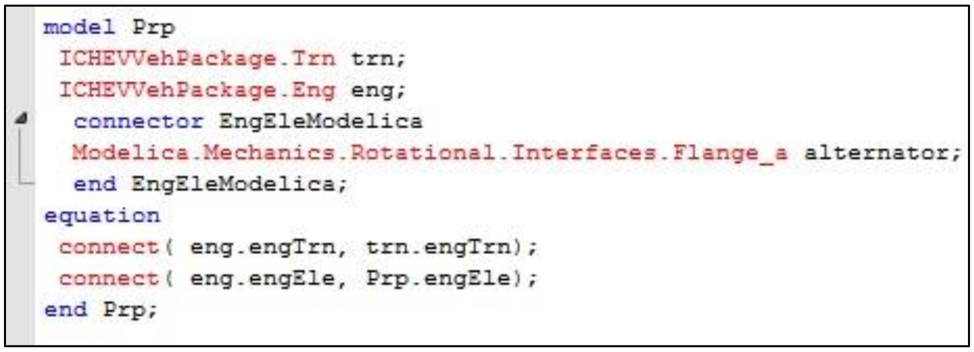


Figure 34. Generating Modelica model from SysML

Once the Modelica templates have been generated from SysML, domain engineers can open them and add the content needed create their analysis models. An example of the code generated for one of these templates is shown in Figure 35.



```
model Prp
 ICHEVVehPackage.Trn trn;
 ICHEVVehPackage.Eng eng;
  connector EngEleModelica
  Modelica.Mechanics.Rotational.Interfaces.Flange_a alternator;
  end EngEleModelica;
equation
 connect( eng.engTrn, trn.engTrn);
 connect( eng.engEle, Prp.engEle);
end Prp;
```

Figure 35. Modelica code generated from SysML model

Note that these templates capture the port nesting modeling pattern described in Section 4.1.1. In Figure 35, the “EngEleModelica” connector captures all of the interfaces between the engine plant and electrical plant in the same way that they are modeled in SysML. In this case, only a single rotational mechanical interface, “alternator”, is included, but any number of interfaces can be nested in this way. This approach provides an organized method for managing a large number of interfaces and because it is backed by a system-level SysML model, it ensures that the interfaces across domain analysis models will be consistent.

5.6 Summary

This chapter outlined how the SysML models and software tools presented in Chapter 4 can be applied to the analysis planning process defined in Chapter 3. Excel is used as a supplementary user interface for domain and system engineers, as it is commonly used in current document-based approaches. A method was described for using Excel to interact with a SysML model by graphically creating “views” for each stakeholder which define the information they need to see, as well as the information they need to author. In addition, while this work is primarily focused on the planning that goes into analysis model development, it was created in a way that allows it to integrate with the work done by Branscomb et al. for generating vehicle analysis model templates from SysML (Branscomb, 2012). While the tool is fairly robust, there are certainly limitations, and the ability to directly support other user interfaces as well would be ideal.

CHAPTER 6

CONCLUSIONS

The value of any model-based systems engineering approach depends on the quality of the models themselves. The previous chapters introduced an approach that can be used to plan the development of not just complex vehicle system models, but also of the corresponding analysis models. This chapter provides a response to the motivating question and desired characteristics posed in Chapter 1. In addition, some insight is provided into the limitations of such an approach. Finally, possible future research areas are identified.

6.1 Response to Motivating Question

The development of vehicle system analysis models has traditionally followed a document-centric approach. To improve this process, though, a more generic question was posed in Chapter 1.

Motivating Question:

How should one plan and guide the development of analysis models?

The goal of this thesis is to improve the existing document-based approach used to create analysis models. This thesis proposes a transitional model-based approach. Because the benefits of MBSE are not easily quantifiable and depend heavily on the nature of the project, taking smaller steps to implement MBSE practices can make the benefits of such an approach more apparent. The approach proposed by this thesis is designed to complement current development practices, rather than replace them entirely. To evaluate whether the approach presented is in fact better than current practices, the desired characteristics of an effective approach must also be examined.

Desired Characteristic:

Follow a formal, precise process

The first characteristic desired for an effective approach is that it be a formal, precise process. To achieve this, the entire process of planning and developing analysis models is modeled in SysML activity diagrams. This formalizes the workflows and roles of the stakeholders. In addition, SysML models are used to define the information that must be captured at each step in the process. Using these models, the token pins modeled in the Activity diagrams are linked to SysML elements that describe their content. These two components provide model users with a formal understanding of the process and instructive models that describe the information that must be generated at each step in the process. Beyond the value of the models themselves, defining this formal process also provided a better understanding of where issues and inconsistencies may arise.

Desired Characteristic:

Provide an intuitive graphical user interface

One drawback of a SysML-based approach is that many users are not yet familiar with SysML. Training users to use SysML can be costly, particularly for large design teams. While SysML models are used wherever practical, in many cases they are not the best option for user interaction. To address this, an Excel-based user interface is used instead.

Most users should be familiar with Excel; it is already one of the most common tools used for vehicle analysis planning at Ford. To make it more intuitive, a template engine is used in Excel to generate dynamic spreadsheets. Additional capabilities are also included, such as context-specific dropdown menus, multi-select capability, and informative tooltips, to make the user interface more user-friendly. These additions are discussed in Section 4.2.3, and provide an impressive amount of functionality within a familiar user interface.

Desired Characteristic:

Minimize opportunity for inconsistencies

As mentioned, there are many potential sources of inconsistency in analysis model planning and development. Several methods are used to address these. First, by defining the formal process of model planning and development, inconsistencies that may arise out of more ad hoc processes can be reduced. Second, using SysML as the master source of information can reduce informational inconsistencies that arise when managing a large number of independent documents. Finally, dependency checking is used within the Excel user interface to ensure that the options selected by each user are valid. These dependencies are maintained in the SysML model, like the rest of the information, so they remain consistent for all stakeholders who interact through the SysML model.

Desired Characteristic:

Support the roles of all stakeholders directly involved

The roles of the many stakeholders are supported in different ways. By formalizing the analysis planning process, those roles become much clearer. In this way, each stakeholder has a better understanding of each other’s responsibilities, as well as their own. If changes are needed, it is easier to determine who has the authority to make those changes.

More practically, the user interface has been designed to be easily tailored to the many different stakeholders involved in analysis model planning—many of which are unlikely to have SysML experience. Because of this, a SysML tool such as MagicDraw would have been inefficient and unwieldy to use by the entire engineering team. To customize the user interface to each stakeholder and their concerns, expertise, and software familiarity, a template engine was created in Excel that is capable of producing spreadsheets from views defined within a SysML model. By creating this interface through intermediate XML files, it also leaves the possibility open to create user interfaces within other tools.

Desired Characteristic:

Add value to the process

Ultimately, a new approach is not worth considering unless it adds value to the existing one. The approach proposed by this thesis is similar in many respects to the existing document-based approach to analysis planning, and was designed to have a minimal impact on current workflows. However, by using MBSE as the foundation for this approach, many of the current inconsistencies and inefficiencies can be reduced. This approach can be used as a stepping-stone from a document-based to a model-based workflow, rather than necessitating abrupt, organization-wide changes.

The tools and models presented were made such that SysML would only need to be used directly by the system engineers; domain engineers can continue to work more or less as they already do. This prevents users unfamiliar with SysML from making invalid changes to the system models and more practically, means that fewer software licenses are needed to support the entire engineering team. However, by using a SysML model as the backbone, the approach proposed in this thesis provides a more informative user interface, better consistency across domains, improved traceability of design decisions, a more formal planning process, and a common set of semantics to improve interdisciplinary communication.

6.2 Contributions

The contributions of this thesis can be divided into two categories. First, there are contributions which address broader, research-related questions. These are summarized as follows. In this research, we:

- **Defined a process for planning and developing complex system analysis models.** The tasks that must be performed by each stakeholder to plan and develop analysis models were defined. This involved not only defining which tasks should be performed, who should perform them, and what information must be exchanged to perform each tasuk.

- **Created a comprehensive SysML model to support analysis model planning.** This SysML model integrates both a descriptive model of the vehicle architecture, as well as models of the analysis model requirements, specifications, and documentation, so that the two these two perspectives can be defined in a single location.
- **Performed validation checks on the process and SysML model by collaborating with Ford engineers.** The process and SysML model proposed by this thesis were created in collaboration with Ford engineers to ensure that the resulting methodology is an improvement on current modeling practices.

In addition to the aforementioned research contributions, there are also more tangible contributions that arose out of the implementation of our approach. These are outlined below.

- **Added functionality to views and viewpoints.** A plugin was created for Excel which is capable of generating unique XML files from SysML using views and viewpoints defined by a user.
- **Developed a template engine in Excel to produce stakeholder-specific user interfaces.** To allow the user interface to be customized for each user, a template engine was written for Excel. This allows domain and system engineers to interact with the SysML model without requiring any formal SysML training, and can be used to make the user interface more intuitive to use.

- **Illustrated the capability of these tools with examples.** Examples were used to make the utility of these previous contributions more apparent. These examples were created to mimic current practices at Ford to demonstrate the utility of our approach for existing workflows.

6.3 Future Work

While the approach proposed in this thesis addresses many of the needs of the planning and development process for system analysis models, there are some limitations to this approach. These limitations, along with some areas of future research, are discussed below.

- **More general support for views and viewpoints is needed.** The Excel user interface, combined with SysML views and viewpoints, is not an all-encompassing solution. While Excel is fairly robust, there may still be a need to support other forms of user interfaces when planning analysis models. A custom XML schema was used for the files that are exchanged between MagicDraw and Excel; this schema should most likely be revisited and more formally defined before utilizing it for other applications. In addition, updates to views and viewpoints in future SysML specifications may render this solution obsolete.
- **The data storage solution chosen should be more scalable.** SysML is not a particularly efficient way to store information. Because this approach relies on the users to manually import and export XML files from SysML, there is also the possibility that the system model could become out of

sync with a domain engineer's user interface. Automatic updates to the SysML model and version control measures could limit these issues. More automated consistency checks are needed throughout the process to ensure that the SysML model remains valid as users update and modify it. In terms of the scalability of this approach, however, a more practical solution may be to use a centralized database as the master source of information, rather than a SysML model.

- **Control signal negotiation should be supported through SysML.** Control signals are not addressed in the approach presented in this thesis. Better support for the negotiation process that occurs when selecting and requesting these signals is an area of future research.
- **The complete Vehicle Reference Architecture needs to be modeled in SysML.** The Vehicle Reference Architecture modeled in SysML is incomplete, because VMA v3 has not been finalized yet. Future work may involve altering the approach presented by this thesis to support VMA v2 until the VMA v3 specification is completed.
- **The "Design Analysis Phase" of analysis model development has not yet been addressed.** Finally, the third phase of analysis model development has not been addressed—the "Design Analysis Phase" shown in Figure 1 and introduced in Section 3.4. Formally defining this phase, as has been done for analysis model planning in this thesis and analysis model development in (Branscomb, 2012), may necessitate changes to the current approach.

APPENDIX A. VIEWS AND VIEWPOINTS

Basic Functionality

A plugin was created for MagicDraw which allows views to be used to graphically select what kind of data should be exported to XML. By selecting the types of elements to export and where those elements are stored, a user can export a very specific set of data to XML.

The base SysML specification includes “view” and “viewpoint” elements, but there is no functionality currently possible with them. Our plugin adds some functionality to them when dealing with many of the elements commonly found in Block Definition Diagrams (BDDs) and Internal Block Diagrams (IBDs).

A view is a package with a <<*view*>> stereotype. It may be related to a single viewpoint by a dependency stereotyped as <<*conforms*>>. That is to say, a view conforms to a viewpoint. A view is used to identify what data it consists of, whether it be a specific element or a package of elements. This is done by linking it to these packages and elements using another dependency relationship, stereotyped as <<*import*>>.

A viewpoint, as defined by the SysML specification, includes information about its purpose, who the stakeholders are, and what their concerns are, among others. A viewpoint, because it is a class, may also inherit from other viewpoints--although this functionality has very little meaning in the base SysML specification. Other than that, however, it serves very little purpose.

Our plugin adds some functionality to these different relationships and uses it to generate XML files with a filtered-down set of data. For an example, refer to Figure 17.

A user starts by first creating a BDD in SysML. From there, in the "Package Diagram" set of tools in MagicDraw, they can create a new view and drop it into the diagram. In order to identify what packages and specific elements should be exported for that view, they should first be dragged into the BDD. Once they've been included, a user can simply create an "import" relationship from the view to the elements they want to filter from. This relationship can be found by either hovering over the package in the BDD (note that this only shows the "Package Import" relationship), or selecting either the "Package Import" or "Element Import" relationships from the "Package Diagram" tools.

Once this is done, a user can then create the viewpoint that the view will conform to. In this case, the viewpoint acts as a filter to identify the elements that should be included in the view. Note that the filter is used to *add* items to the view, rather than delete them. This plays a more important role in the "Advanced Functionality" section.

To select types of elements to include, a user can use one of three options--meta-classes, stereotypes, and "Type Elements". A Type Element is any element which may type another element, such as an interface block used to type a proxy port. In order to add these elements to the filter, simply create a dependency relationship from the viewpoint to the elements to include.

The filtering process works by checking every element chosen to be "imported" into the view against the meta-classes, stereotypes, and types called out by the viewpoint. If an element is valid, the elements that it owns are then checked as well. Note that this means elements are not necessarily checked recursively—if a user doesn't specify that "Packages" should be included in their filter, then any sub-packages will be ignored.

Similarly, properties owned by a block won't be included unless the <<Block>> stereotype has been added to the filter.

Once these elements have been defined, a user can export their view by right-clicking on it in the containment tree and selecting *VMA Analysis Planner > Export View to XML*. Finally, from there, they can name their file and choose where to save it.

Advanced Functionality

To augment the functionality, several additional features were added. One such feature is the ability for a view to import other views. Because a view is just a stereotyped Package, the default "import" relationship in the SysML specification did not need to be modified. Note that these imported views already conform to their own viewpoints, which means that they can act independently as a filter. However, because our process works as an *additive* filter, additional stereotypes can be added to them when they are imported into the new view. This is done by adding stereotypes to the viewpoint to which the new view conforms.

In addition to importing views, functionality was also added for viewpoints to inherit from other viewpoints. Note that the basic "Package" diagram toolbar doesn't include this function in BDDs, but it can be found in the "Block Definition Diagram" toolbar. This allows users to define a base set of viewpoints for other viewpoints to be derived from. For instance, a user may choose to create a viewpoint that would be used to filter out everything but instance data, and then add additional meta-classes to viewpoints which inherit from it.

Although the goal of this views/viewpoints functionality was to allow users to export customized sets of XML data, it was also apparent that it could be useful for other applications as well. One such application is for creating custom views of diagrams. Once a view is defined, a user may right-click on a diagram and select *VMA Analysis Planner > Create custom diagram view*. A copy of the diagram will be created which only includes elements specified by the filter. Note that this functionality has not been fully implemented, but it is a useful example of it could be used in the future.

APPENDIX B. EXCEL TEMPLATE ENGINE

A schema was developed to allow users to auto-generate user forms in an Excel spreadsheet using XML data from a SysML model. The process is somewhat complicated and not entirely bug-free, but this document is meant to give some background into how it works, how it can be used, and ongoing work being done to improve it.

The cornerstone of the entire process is the range where the format of the template is specified. This may be done in either the first row or first column of the spreadsheet (but not both!). The first cell in the document, at A1, should be labeled "TEMPLATE". From there a user can specify commands in the following format:

(Desired Output)::(Reference element)

We'll start with the "Reference Element" portion of the command. The reference element is identified using a pseudo-XML node. A user specifies the type of the element, which attributes to use to find it, and what the value of those attributes should be. Say we want to find a block whose name is "Analysis Application". This block could be found with the following reference element:

<block name="AnalysisApplication"/>

Note that the schema does require that the XML node is a valid XML element—therefore, the forward slash at the end is always required. Users can specify as many attributes as they need to identify the element. However, text content and children elements within the XML node cannot be used. For more information about which XML attributes can be used for each SysML element, refer to Table B1.

Table B1. Valid XML attributes for different SysML elements

XML Element	**XML Attributes**				
Package	id	name	ownerID	comment	
Block	id	name	ownerID	isAbstract	comment
Property	id	name	ownerID	multiplicity	typeName
Property cont.	typeID	redefinedID	comment		
Enumeration	id	name	ownerID	comment	
Literal	id	name	ownerID	comment	
Dependency	id	ownerID	targetID	targetOwnerID	
Instance	id	name	ownerID	classifierID	comment
Slot	id	ownerID	featureID	comment	
Value	id	name	ownerID	instanceID	comment
Generalization	id	ownerID	targetID	targetOwnerID	comment

Attribute values may also be used to reference other cells in the spreadsheet using the following format:

@R(row number)C(column number)

The portions in parentheses should be replaced by the appropriate number for the row or column. It is also possible to refer to only a particular row or column, by omitting either the row or column reference. However, keep in mind that additional hidden columns get added when generating these sheets, so it is important to define static references first in the template, before defining other elements.

The "Desired Output" of the command tells the plugin what to do with the reference element once it finds it. There are several keywords that can be used for this process:

- PACKAGE
- INSTANCE
- FIELD
- VIEW

We'll address them in order. The first keyword, "PACKAGE", is used to identify or create the package to store our data. This is the only case where the reference element may be labeled as "null", rather than as an XML element. If it is labeled as "null", the data will be imported into the package where a user right-clicked in MagicDraw by default. Note that only one "PACKAGE" command may be used.

The second keyword, "INSTANCE", tells the plugin to create an instance of the element identified by the reference element. For this case, the reference element *must* refer to a block. This keyword is also used when importing instance data. In this case, the same reference element is used to identify instances from SysML that can be mapped to our template.

The third keyword, "FIELD", is used to actually create user fields. In this case, "FIELD" must refer to either a "property" or "block" as its reference element. If it refers to a property, it will check to see if that property refers to an enumeration, value type, or block. If it is none of these, it will do nothing. If it's a value type, it will create a field where a user can enter text. If it is an enumeration, it will create a dropdown which consists of the literals owned by that enumeration. In addition, it will check the property's multiplicity to see if the user is allowed to select multiple options from the dropdown. If the property or reference element refers to a block, it will create a dropdown menu of all of the specializations of that block.

The final keyword, "VIEW", is used to display instance data to the user, but disallow them from actually editing that data. It may be used in the same way as the "FIELD" command.

Finally, there is a fifth option when specifying the desired output. A user may specify any XML attribute to display to the user or use as a reference for other cells. For instance, it is often useful to get a block's ID and use it as a reference to find properties whose "ownerID" attribute matches it. An example of this can be seen below.

NOTE: commands cannot be entered in Row 2. Row 2 is reserved for headers used to store additional information about fields, so any information entered in Row 2 can throw off the entire process.

Once these fields have all been specified, the user must then highlight the range (which should be either a single row or single column) and name the range. This can be done by right-clicking on the range and selecting *Define Name*. The name must be "TEMPLATE", and the scope must be set to the name of the worksheet that is currently being worked on.

Once the template range has been created, there are two steps left before creating our user form. First, select the “VMA Tools” tab, and click “Import XML”. The XML file from SysML will be imported into our workbook and mapped to a copy of the

"XMLTemplate" worksheet. This file may be used to refer to a particular property, but should never be modified by the user.

There is an optional step to this process—a user can choose to import existing instance data into the spreadsheet using the "Import Existing Instances" tab. From here, they will be asked whether they want to map the data to multiple ranges. If they select "Yes", each valid instance will be mapped to its own row/column. If they select "No", all of the instance data contained in the XML file will be mapped to the same row or column. This option may be used in some cases, but it is generally best to avoid it—without some experience with the schema and the SysML model, it can be difficult to predict whether certain data will be overwritten.

The final step in the process, if no instance data has been imported, is to generate a new row or column for the user to fill out data. This can be done by clicking the "Generate new row/column" button in the "VMA Tools" tab. A new row or column will then be created in the first empty row that is found.

REFERENCES

[1] Auto-View Generation Working Group Wiki, OMG, http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-autoview:auto-view_generation_working_group.

[2] No Magic's Magicdraw, No Magic, http://www.nomagic.com/products/magicdraw.html.

[3] AUTOSAR, www.autosar.org.

[4] InterCAX, www.intercax.com.

[5] Bajaj, M., Zwemer, D., Peak, R., Phung, A., Scott, A. G., and Wilson, M., 2011, "SLIM: Collaborative Model-Based Systems Engineering Workspace for Next-Generation Complex Systems," *Aerospace Conference, 2011 IEEE*, IEEE, pp. 1-15.

[6] Boddy, M., Michalowski, M., SchwerdFeger, A., Shackleton, H., and Vestal, S., 2011, "Formal United System Engineering Development (FUSED) Language," Air Force Research Laboratory

[7] Branscomb, J. M., 2012, *Supporting Multidisciplinary Analysis Using System Architectures in SysML*, Thesis, Woodruff School of Mechanical Engineering, Georgia Institute of Technology.

[8] Branscomb, J. M., Paredis, C. J., Che, J., and Jennings, M. J., 2013, "Supporting Multidisciplinary Vehicle Analysis Using a Vehicle Reference Architecture Model in SysML," *Procedia Computer Science*, **16**, pp. 79-88.

[9] Claver, C. F., Dubois-Felsmann, G., Delgado, F., Hascall, P., Marshall, S., Nordby, M., Schalk, T., Schumacher, G., and Sebag, J., 2010, "Using SysML for MBSE Analysis of the LSST System," *SPIE Astronomical Telescopes and Instrumentation: Observational Frontiers of Astronomy for the New Decade*, International Society for Optics and Photonics, pp. 77381D-77381D-77310.

[10] Delp, C., Lam, D., Fosse, E., and Lee, C.-Y., 2013, "Model Based Document and Report Generation for Systems Engineering," *2013 IEEE Aerospace Conference*, IEEE, pp. 1-11.

[11] Dumitrescu, C., Mazo, R., Salinesi, C., and Dauron, A., 2013, "Bridging the Gap between Product Lines and Systems Engineering: An Experience in Variability Management for Automotive Model Based Systems Engineering," *Proceedings of the 17th International Software Product Line Conference*, ACM, pp. 254-263.

[12] Eisenmann, H., Basso, V., Fuchs, J., and De Wilde, D., 2010, "ESA Virtual Spacecraft Design," *Proceedings of the 4th International Workshop on System & Concurrent Engineering for Space Applications (SECESA)*.

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

[14] European Space Agency, Virtual Spacecraft Design Project, http://www.vsd-project.org/.

[15] Friedenthal, S., Moore, A., and Steiner, R., 2012, *A Practical Guide to SysML: The Systems Modeling Language*, Elsevier, Waltham, MA.

[16] Graves, H., and Bijan, Y., 2011, "Using Formal Methods with SysML in Aerospace Design and Engineering," *Annals of Mathematics and Artificial Intelligence*, **63**(1), pp. 53-102.

[17] INCOSE SE2 Challenge Team for Telescope Modeling, Docbook Plugin for Magicdraw, http://sourceforge.net/projects/mbse4md/files/.

[18] Ingham, D., Donahue, K., Kennedy, K., and Post, S., 2012, "A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems," *Infotech@Aerospace 2012 Conference*, Santa Ana, CA.

[19] Jennings, M., 2013, "Personal Communication."

[20] Johnson, T., Kerzhner, A., Paredis, C. 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.

[21] Karban, R., Hauber, R., and Weilkiens, T., 2009, "MBSE in Telescope Modeling," *INCOSE INSIGHT*, **12**(4).

[22] Keeney, R. L., 1994, "Creativity in Decision Making with Value-Focused Thinking," *Sloan Management Review*, **35**, pp. 33-33.

[23] Marco, J., and Vaughan, N., 2010, "Integration of Architectural Modelling Using the SysML within the Traditional Automotive CACSD Process," *UKACC International Conference on Control 2010*, IET, pp. 1-6.

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

[25] No Magic Inc., 2013, "Magicdraw Report Wizard User Guide," No Magic, Inc., www.nomagic.com/support/documentation.html.

[26] Nottage, D., and Corns, S., 2012, "Application of Model-Based Systems Engineering on a University Satellite Design Team," *Procedia Computer Science*, **8**(0), pp. 207-213.

[27] Object Modeling Group, 2012, "OMG Systems Modeling Language.", www.omg.org/spec/SysML/1.3/.

[28] Object Modeling Group, 2012, "SysML-Modelica Transformation.", www.omg.org/spec/SyM/1.0/.

[29] Piques, J., and Andrianarison, E., 2012, "SysML for Embedded Automotive Systems: Lessons Learned," *Interfaces*, **3**.

[30] 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*, IEEE, pp. 1306-1311.

[31] Robinson, K., Tramoundanis, D., Harvey, D., Jones, C. M., and Wilson, S., 2010, "Demonstrating Model-Based Systems Engineering for Specifying Complex Capability," *Systems Engineering Test and Evaluation (SETE'10).*

[32] Soyler, A., and Sala-Diakanda, S., 2010, "A Model-Based Systems Engineering Approach to Capturing Disaster Management Systems," *2010 4th Annual IEEE Systems Conference*, IEEE, pp. 283-287.

[33] Spangelo, S. C., Cutler, J., Anderson, L., Fosse, E., Cheng, L., Yntema, R., Bajaj, M., Delp, C., Cole, B., and Soremekum, G., 2013, "Model Based Systems Engineering (MBSE) Applied to Radio Aurora Explorer (RAX) Cubesat Mission Operational Scenarios," *Aerospace Conference, 2013 IEEE*, IEEE, pp. 1-18.

[34] Technical Operations, 2007, "Systems Engineering Vision 2020," INCOSE, www.incose.org/ProductsPubs/products/sevision2020.aspx.

[35] Tiller, M., Bowles, P., and Dempsey, M., 2003, "Development of a Vehicle Model Architecture in Modelica," *3rd International Modelica Conference*.

[36] Walsh, N., 2011, *Docbook 5: The Definitive Guide*, XML Press.