VANDERBILT UNIVERSITY

INSTITUTE FOR SOFTWARE INTEGRATED SYSTEMS

Institute for Software-Integrated Systems

Technical Report

TR#: **ISIS-15-106**

Title: **META Design Space Exploration Using Dynamics**

Authors: **Zsolt Lattmann, James Klingler, Patrik Meijer, Ted Bapty and Sandeep Neema**

This research is supported by the Defense Advanced Research Project Agency (DARPA)'s AVM META program under award #HR0011-13-C-0041.

Table of Contents

List of Figures ii

List of Tables iii

1. Requirements Specification with Test Benches 1

2. Composing Lumped Parameter Dynamics 3

   2.1 Component Curation 4

   2.2 System Model Composition 6

3. Cyber-Physical Dynamics 8

4. Execution of Analysis 9

5. Parametric Exploration 10

6. Design Data Visualization 11

7. Driveline Case Study 12

   7.1 Requirements 13

   7.2 Test Benches 13

   7.3 Single Design Point (Seed Design) 16

   7.4 Discrete Design Space 16

   7.5 Parametric Design Space 22

   7.6 System Robustness 24

   7.7 Surrogate Model and Prediction Profiler 25

   7.8 Conclusion 26

8 Evaluation 28

List of Figures

Figure 1: Dynamics tool architecture diagram 4
Figure 2: Dynamics tool data flow 6
Figure 3: Diagnostics of dynamics models 7
Figure 4: Example simulation result (vehicle speed) 11
Figure 5: Design process 12
Figure 6: Results shown in requirement categories 13
Figure 7: Full Speed Forward Test Bench 14
Figure 8: Set of Test Benches 14
Figure 9: Results shown in requirement categories 15
Figure 10: Single design point for the driveline model 16
Figure 11: Discrete design space for driveline model 18
Figure 12: Example of property constraints 19
Figure 13: Number of possible configurations for design space 19
Figure 14: Number of configurations after all constraints applied 20
Figure 15: Set of Test Benches for the design space 20
Figure 16: Requirement violations for design points 21
Figure 17: Limit violations for design points 21
Figure 18: Ranking of design points 22
Figure 19: Parametric design space exploration model 22
Figure 20: Results of parametric design space exploration 23
Figure 21: Parametric design space exploration models 24
Figure 22: Results of parametric exploration over a discrete design space 25
Figure 23: Prediction profiler and constraint plot 26
Figure 24: Requirement evaluation for the seed design (maps to Configuration 41 in the design space) .. 26
Figure 25: Requirement evaluation for Configuration 30 27

List of Tables

Table 1: Transmission alternatives 17
Table 2: Engine alternatives........ 18
Table 3: CyPhy2Modelica model generation time and overhead 30

1. Requirements Specification with Test Benches

For any design problem, engineers define many requirements which must be fulfilled by the final design. The individual requirements might be grouped based on domains such as performance, safety, spatial/geometric, ergonomic, etc. At this stage of the design process all requirements are written in text, and are frequently stored in a requirements management tool.

In the OpenMETA tools, Test Benches are the executable versions of the requirements and are used to evaluate system designs against specific requirements. Each Test Bench contains a link to a system design (the "system under test" object). The system design can be a crude system mockup at the early stages of the design process, with placeholders for certain subsystems and components whose implementation is not yet clear. The decomposition of the system and its subsystems forms an architecture that can be extended to form a design space by adding new alternative components and subsystems, forming a design space. Even if the original system design is significantly augmented, all associated Test Benches will remain functional and can be used to evaluate all encoded requirements across all point designs generated from the design space. Thus, by defining Test Benches early and executing them periodically, the design space will continually evolve in a positive direction.

In addition to the system under test, each Test Bench has a set of well-defined concepts: analysis tool setup, parameters, context models, metrics, and post processing scripts. A detailed description of each object is presented in this section. In short, the type of requirement dictates the context models and the choice of analysis tool; each analysis tool generates a set of artifacts (e.g., simulation results), from which metrics are extracted using post processing scripts. Metrics are typically numerical values (with SI units) that map to the textual requirements. For instance, "the vehicle must have industry-leading fuel efficiency" would translate to a numerical metric: 40 mi/gal minimum. Lumped Parameter Dynamics Models are used to produce metrics for performance-type requirements.

Lumped Parameter Dynamics (LPD) Models are either represented as Ordinary Differential Equations (ODEs) or Differential Algebraic Equations (DAEs) and are often used at the early stages of the design process to ensure fast turnaround time in analysis execution time and design space exploration. Generally, LPD models are higher abstraction and lower fidelity than Finite Element Analysis (FEA) models, which use Partial Differential Equations (PDEs). Several modeling languages and tools support Lumped Parameter Dynamics modeling, for example: Bond Graphs, 20-Sim, MatLab, Modelica, etc. Some of these tools require causal models, meaning that the flow of data is unidirectional, and pre-specified by the user; others can use acausal models, meaning that the user simply makes connections and the tool resolves the causality of data flow automatically based on the underlying physics of the connected models.

Within the scope of the AVM program, we chose to use acausal modeling languages[1] for Lumped Parameter Dynamics systems. Acausal modeling comes with many benefits:

- models are easier to maintain because they contain a concise equation-based description of the dynamics (i.e., a declarative mathematical description of the underlying physics)
- cause-effect variables are bidirectional (users can quickly compose system models without concern to the direction of data flow)
- it provides a physics-based modeling paradigm (i.e., ensures that all models obey the laws of physics)

The initial OpenMETA implementation utilized Bond Graphs[2], which is a domain independent acausal modeling language. Bond Graphs provided a flexible modeling paradigm and an easy way to understand models, but not all engineers are familiar with Bond Graphs, and even for domain experts it could be challenging to develop new models. Domain experts usually feel more comfortable in a modeling environment where the domains are clearly identified and defined. In the OpenMETA toolchain we finally chose to use Modelica models to capture the dynamics of component models for several reasons:

- Modelica is a powerful object-oriented acausal modeling language
- A Modelica library[3] for ground vehicle design was provided by the AVM program
- The Modelica community has developed and maintains the Modelica Standard Library, which provides domain specific components, connectors, sources, and sensors for many physical domains (e.g., electrical, mechanical, thermal, fluid)
- There are multiple Modelica simulation tools (some listed here), both open-source and commercial, each of which provide several different DAE solvers
- OpenModelica[4] (open-source, free)
- JModelica[5] (open-source, free)
- Dymola[6] (commercial)
- SystemModeler[7] (commercial)
- MapleSim[8] (commercial)

[1]Willems, Jan C. "The behavioral approach to open and interconnected systems." *Control Systems, IEEE* 27, no. 6 (2007): 46-99.
[2]D.C. Karnopp, D.L. Margolis & R.C. Rosenberg, System Dynamics: Modeling and Simulation of Mechatronic Systems (5th edition). Wiley (2012). ISBN: 978-0470889084.
[3] C2M2L was a project within the AVM program which resulted in a library of parametric components targeting the design of an amphibious assault vehicle
[4]D.C. Karnopp, D.L. Margolis & R.C. Rosenberg, System Dynamics: Modeling and Simulation of Mechatronic Systems (5th edition). Wiley (2012). ISBN: 978-0470889084.
[5]https://openmodelica.org/
[6]http://jmodelica.org
[7]http://www.3ds.com/products-services/catia/capabilities/modelica-systems-simulation-info/dymola/
[7]http://www.wolfram.com/system-modeler/
[8]http://www.maplesoft.com/products/maplesim/

2. Composing Lumped Parameter Dynamics

As we have presented in the previous section, a CyPhy Test Bench is the executable form of a system design requirement. Within a Test Bench, the system under test object is an AVM Design Model (ADM). AVM Design Models consist of AVM Component Models (ACMs) and describe the connections between the ACMs and any system-level parameters. An ACM contains one or more domain models (e.g., structural, dynamics, and manufacturability). For the purpose of capturing dynamics behavior each ACM contains one or more dynamics (e.g., Modelica) models. ACMs capture only the interfaces of the included dynamics models but not the implementations thereof (the interfaces are required for composition and instantiation from OpenMETA, but the implementation is stored in an external resource). Component authoring (i.e., curating[9]) can be time consuming process, and since the domain models already define the interfaces, the ability to programmatically create multiple ACMs from a library of existing domain models would be necessary to avoid duplication of effort.

These lumped parameter system models consist of multiple component models, each of which captures the behavior of that component to some specified degree of fidelity, i.e., realism and accuracy. When engineers construct a component behavior model, they consider the typical operational boundary conditions along with the input/output energy and data flows, and guarantee the behavior of the component model only in that 'typical' range of expected conditions. These conditions can then be included with the model as a caveat[10] in the form of required operating condition ranges (which may also be included with the final product[11]). Also, the engineer may choose to encode a particular level of detail into a model depending on the level of fidelity desired in the analysis results.

When setting up an analysis to evaluate a certain design requirement, it is prudent to select a level of model fidelity for component 'C' which is appropriate for C's effect on the outcome of that particular analysis. For example, if the purpose of the analysis is to quantify the performance of a vehicle's suspension system while driving at 10 m/s on a bumpy road, it is computationally wasteful and inadvisable to use a high-fidelity engine model which captures the friction between the engine's cylinder walls and the pistons; in fact, the entire drivetrain might be reduced to a simple torque source without drastically affecting the accuracy of results. However, when measuring the vehicle's coolant temperature to compare the effect of different engine lubricant viscosities, it may be necessary to model the friction between cylinders and pistons to get accurate results (i.e., produce analysis results which have a high degree of fidelity to real-world behavior). To support varying levels of fidelity and enable rapid switching between different fidelity levels, ACMs can contain multiple dynamics behavior models, each targeting a separate

[9] The process of creating a CyPhy/AVM Component Model from one or more domain models.

[10] For instance, a gasoline engine's performance/power profile is valid only for a certain range of fuel octane rating; if the octane rating is too low, fuel detonation can occur, and the engine's power output will suffer.

[11] To follow the example from [3], many gasoline engines specify a minimum octane rating for the fuel.

fidelity level; the designer can quickly switch fidelity levels using an enumeration-type selection menu.

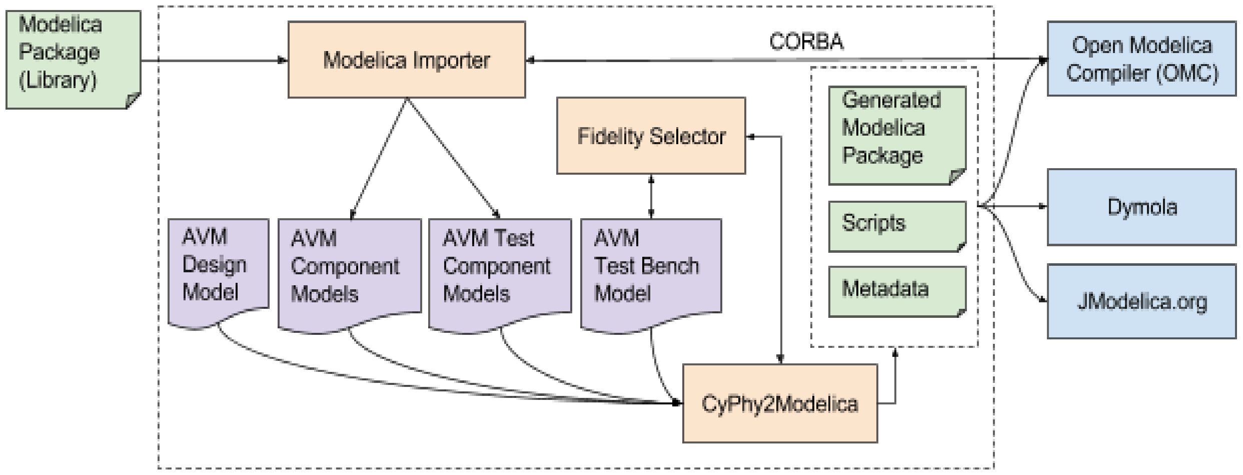


Figure 1: Dynamics tool architecture diagram

2.1 Component Curation

As stated above, the AVM program drove the production of the C2M2L library containing Modelica models of components used in constructing amphibious armored vehicles. For this and other reasons (also stated above), Modelica models were selected to encode the dynamics behavior for ACMs. Thus, OpenMETA developers needed to curate Modelica models into CyPhy Component models in the process of designing, implementing, and testing the OpenMETA toolchain. Unfortunately, Modelica models can be complex to read and understand, even for an experienced user, for several reasons:

- Inheritance - Modelica is an object-oriented modeling language, and allows model Y to *extend* model X, meaning that Y will inherit all the interfaces and implementation from X. The Modelica language specification allows for multiple inheritance, meaning that one model may inherit from several base models. Furthermore, each base model may inherit and accrue several generations of interfaces (and content).
- Restricted data - Modelica has several types of restricted data types (e.g., *protected* and *final*) which should not be 'set' (i.e., modified) when using that model (i.e., instantiating it in a system assembly or referring to it for inheritance).
- Parameter values - There are different methods of setting a parameter when instantiating a Modelica model (e.g., *start*=, *fixed*=), each of which may be invalid in certain situations and may yield different behavior.
- Object types - Modelica has different model types (e.g., *inner*, *outer*, *partial*, etc.), each of which have non-obvious implications.

Given that each model may have multiple *extends* statements indicating inheritance from several models (each of which may also utilize *extends*), and each model in the chain is a specific type and may incorporate some restrictions on its data, understanding all implications and condensing the information to a flat, understandable format is non-trivial. Consequently, quantifying the interfaces of a Modelica model for the purpose of creating an analogous CyPhy representation is tedious and error-prone. The OpenMETA developers experienced this tedium with the component models from the C2M2L Modelica package, which contains complex models with multiple levels of inheritance. Compounding the problem is the fact that in order to build large-scale dynamics system models in CyPhy, users would need to curate entire component libraries from Modelica to the ACM format, and it was apparent that automating the process was critical for facilitating and streamlining the OpenMETA design flow. To address the issue, we created the Modelica Importer tool, along with the py_modelica_exporter utility tool. The Modelica Importer provides a user interface from OpenMETA to py_modelica_exporter to extract the relevant information from the desired Modelica packages/models, and to import the information into the CyPhy/ACM format, including support for creating multi-fidelity component models.

The py_modelica_exporter module utilizes the OMPython module, a Python console interface for the OpenModelica compiler. OMPython opens and maintains an OMShell session (OpenModelica's console application), sends user-specified commands, and gets the response. In many cases, that response is a string, which utilizes commas, parentheses, brackets, and braces to mark up the structure (a structure that is unique to the Modelica language specification). OMPython does have a parser to read the response string and extract Python-typed variables. When the parser fails, regular expressions can be used to convert the response string to a usable format, and py_modelica_exporter contains the implementation for these cases[12]. Also, py_modelica_exporter handles the recursion needed to trace multiple layers of inheritance, and checks all the Modelica data types as it goes, flagging any objects or attributes that should not be included in the CyPhy Component. It can load multiple Modelica packages in a single session and exports both package and model descriptions in a JSON file format, making it portable and useful in situations outside the scope of the AVM program.

To recap, AVM Component Models can refer to one or more Modelica models. In order to associate a Modelica model with an AVM Component Model, the interfaces, connectors, and parameters must be extracted from the Modelica model, which can be time-consuming: multiple layers of inheritance combined with the Modelica language's idiosyncrasies can make the manual task very tedious, even for a person who is familiar with both CyPhy and Modelica. To minimize the labor required for the user to curate a Modelica model and create a CyPhy Component model, the Modelica Importer curation tool was created.

The Modelica Importer tool supports:

- Importing one or more Modelica models from the Modelica Standard Library or a user-defined Modelica library

[12] It is possible to create valid Modelica models which cannot be parsed by the OpenModelica compiler.

- The Modelica models can be imported as AVM Components (including multi-fidelity components) or AVM Test Components depending on the context in which the tool is invoked (Test Components are similar to Components, but are used only in Test Benches to define context)

The Modelica Importer tool provides a usable, succinct interface to get information from Modelica models, and makes it possible to rapidly create multi-fidelity AVM Component Models based on existing Modelica libraries. Early in the OpenMETA development timeline, the amount of time for a veteran user to manually curate a typical C2M2L library component was measured in hours. Using the Modelica Importer, this same task takes a few minutes, a small fraction of the original time. The time savings is magnified for the curation of an entire component library, resulting in an orders-of-magnitude reduction in user effort.

2.2 System Model Composition

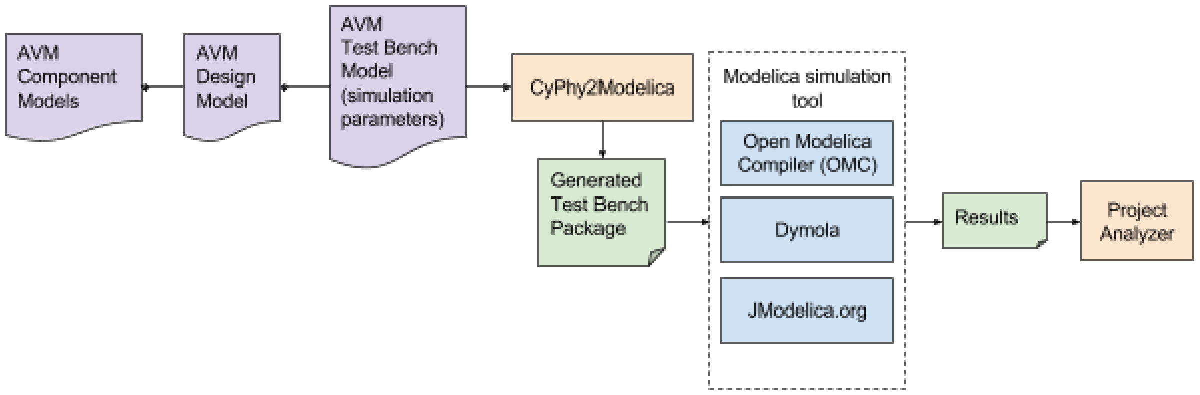


Figure 2: Dynamics tool data flow

Once the user has a library of AVM Component Models, he can proceed to construct system assemblies and Test Benches, and it becomes possible and necessary to execute analyses. The CyPhy2Modelica model translator takes a CyPhy Test Bench model as an input and produces a composed Modelica system model. Prior to the CyPhy model's translation, the CyPhy2Modelica tool checks the structure and content of the model to identify any Modelica syntax violations that can be caught without actually flattening (i.e., compiling) the generated system model. This check utilizes multiple rules, which are based directly on the Modelica language specification and (to some extent) the Modelica Standard Library. In order to save computational time for each CyPhy Test Bench, the user can define for that Test Bench the level of Modelica model complexity to be used when performing the simulation (assuming there are multiple dynamics models associated with the ACMs used in the Test Bench's Top Level System Under Test). To support the Modelica model (fidelity) selection, the CyPhyFidelitySelector tool was implemented. Once the selection is done using the CyPhyFidelitySelector, the CyPhy Test Bench stores the selected fidelity level for each ACM class. Part of the CyPhy2Modelica translation process is to query the fidelity settings from the CyPhy Test Bench and instantiate the appropriate Modelica model implementation within the generated system design. CyPhy2Modelica is a robust and well-tested tool with the following capabilities:

- Any Components and subsystems without dynamics models (e.g., components with a manufacturing model or a 3D CAD model only) are identified and excluded from the generated models.
- Generated models comply with Modelica Specification 3.2.
- Generated models are succinct and light-weight, with minimal overhead.
- A full Modelica package is generated, allowing the user to view all the associated models and sub-packages from a single entry-point.
- When invoked as a standalone interpreter, CyPhy2Modelica exports all component alternatives as part of the generated Modelica package.
- Generated Component models are annotated as *replaceable*, a Modelica keyword which allows the user to easily find and substitute any viable alternative model when viewing the generated system with native Modelica tools (e.g., Dymola).
- Value flows between parameters are preserved in the generated Modelica models as variable aliases, and the translation of simple formulae is fully supported.
- The spatial layout from the CyPhy model is preserved, which aids the user when in-depth debugging of generated models is necessary.
- A traceability map between the CyPhy models and Modelica models is generated.
- The source code of the translation tool was profiled and optimized to provide faster Modelica model code generation.
- Support for versioned Modelica libraries.

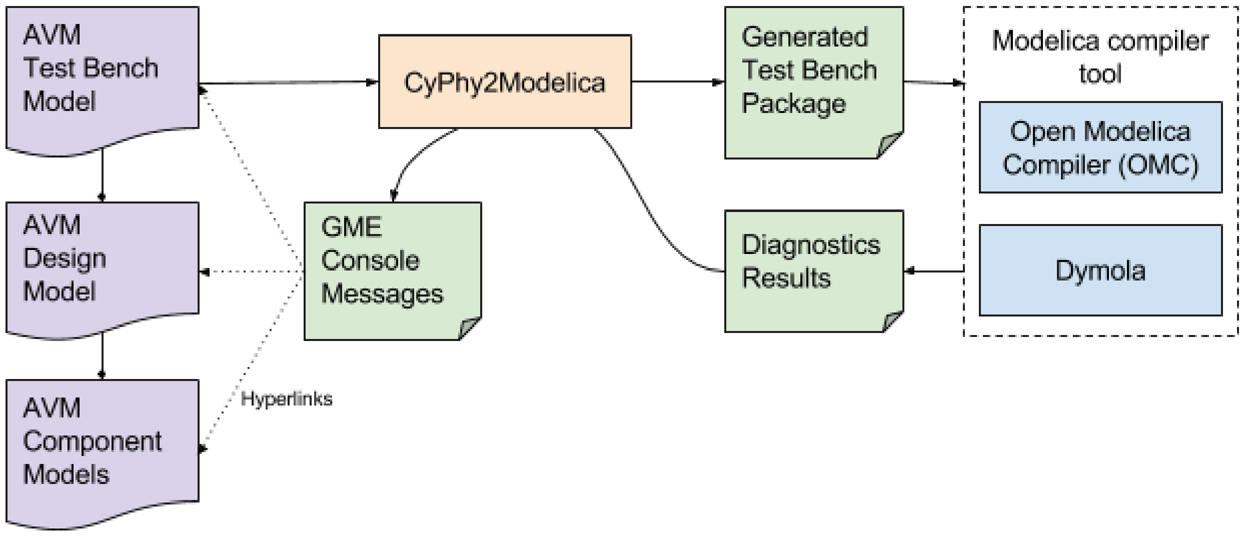


Figure 3: Diagnostics of dynamics models

A model diagnostic utility is provided in the form of a structural model checker including 110 rules; prior to the model translation, the structure of the CyPhy model is evaluated against the checker rules. These rules include violations of Modelica syntax, in addition to semantic errors such as incompatible connections, self-connections, invalid value ranges, loops and type mismatches in value flows, etc. The checker results are summarized in the GME Console for the user based on the severity of the rule violations. As an additional option for pre-execution model diagnostics, generated Test Bench models can be checked for correctness using an external Modelica compiler, and the results are mapped back to the GME Console, with hyperlinks.

3. Cyber-Physical Dynamics

The composition of physical system models from Component models was described in the previous section. The OpenMETA tools also support cyber-physical dynamics system model composition. To augment the existing physical systems with cyber components, controllers can be added to the system designs. Both physical models and controller models are captured by AVM Component Models. Controllers can be implemented and imported from (a) Modelica or (b) Simulink Stateflow.

Modelica controller models are imported and associated with ACMs by the Modelica Importer tool as explained in the previous section. Simulink Stateflow controllers are translated to a Cyber Composition language and packaged as ACMs. The Cyber Composition language and the Simulink Stateflow model import process are presented in-depth in the 'Software Design and Implementation' section. Since these controller types are represented as ACMs in CyPhy, all OpenMETA design/analysis techniques are valid for both physical and controller Component Models, including discrete design space exploration, Test Bench execution, and parametric exploration. This provides an easy path for Cyber-Physical System (CPS) design.

During the composition process of the system design described in the previous section, the cyber code generator (a set of model interpreters) is called for every ACM containing a Cyber Composition (i.e., Simulink Stateflow) model. Each such controller model is transformed into C code, wrapped into a Modelica model as an external C function, and instantiated in the generated system model. User-defined parameters and connections between all controllers and the plant model are generated by the CyPhy2Modelica tool to complete the Cyber-Physical dynamics model. After the composition is completed, simulation and other analyses (e.g., formal verification) can be performed on the generated model.

4. Execution of Analysis

The system model composition of lumped parameter dynamics models for a single design point was presented in the previous sections. These composed system dynamics models can immediately be simulated using a Modelica tool (e.g., OpenModelica or Dymola). The py_modelica Python package is used to script the simulation tasks and to save all simulation results in a consistent way, regardless of which Modelica tool was used. In addition to the raw simulation results, important system-level properties are extracted from the simulation results by user-defined post-processing scripts. If additional logic or another tool is required for executing the simulation or analysis tasks, those extra tools can be registered to the OpenMETA environment and associated with the CyPhy2Modelica model composer. The CyPhy2Modelica model composer applies the analysis tool selection, which is defined in the workflow as part of the Test Bench. Therefore, downstream analysis tools can utilize the generated dynamics models and do further analysis (e.g., structural, simulation, formal verification) on the composed dynamics models. All generated analysis or simulation results are visualized by the Project Analyzer.

The OpenMETA tools allow lumped parameter system models constituting a single design point, frequently called a *seed* design, to be transformed into a design space. Within a design space, components may be replaced with alternative design containers, capturing design choices/trade-offs. For example, in a model of a drive line there is a choice in the selection of the engine and transmission used. These alternatives will yield a range of different design points, where each can be evaluated against the system requirements by executing Test Benches. Starting from a Test Bench, the Master Interpreter utility tool provides automation to repeatedly invoke the CyPhy2Modelica model composer on every selected design point (within the design space). This automation significantly reduces the time to compose dynamics models for simulation or formal verification when compared to traditional design approaches. Moreover, simulations are executed in parallel using the Job Manager[13].

[13] See the Job Manager and Remote Execution section for further details on execution capabilities

5. Parametric Exploration

The design space exploration tool (DESERT) encodes architecture choices and discrete component alternatives for the entire system design. The OpenMETA tools also support parametric design space exploration through the Parametric Exploration Tool, which is presented in-depth in the ‘Parametric Exploration Tool’ section. The purpose of this tool is to facilitate several different parametric design techniques, such as parameter optimization, running a design of experiment, and doing uncertainty propagation for a single design point in the discrete design space. This section briefly describes and presents examples of how the Parametric Exploration Tool capabilities can be used in the dynamics domain.

A Parametric Exploration model contains a CyPhy Test Bench, which has inputs (i.e., parameters) and outputs (i.e., metrics), and a parametric exploration driver object, which can be an optimizer, a parameter study (i.e., design of experiment), or a Probabilistic Certificate of Correctness (PCC) object. The CyPhyPET tool generates an executable experiment from the Parametric Exploration model. CyPhyPET invokes a domain-specific model translator[14] (in this case, CyPhy2Modelica) to generate the executable Test Bench model (in this case, a fully composed Modelica system model). In addition, CyPhyPET generates an executor wrapper (e.g., modelica_executor.py) around the Test Bench for automated execution with different input parameters provided by the selected driver.

In the ‘Examples’ section, some use-cases of these PET experiment types are provided in the context of a driveline design problem.

[14] The domain specific interpreter must implement the *ICyPhyInterpreter* Interface, which is required to make it compatible with automation interpreters such as CyPhyPET, MasterInterpreter, etc.

6. Design Data Visualization

After dynamics system models (Test Benches) have been executed to generate simulation results, the Project Analyzer tool is used to visualize the results. The Project Analyzer provides a unified way to visualize results for all analysis tools, including the lumped parameter model simulations, through the Test Bench Manifest file. In addition to the metric values, simulation plots (i.e., time series of data) are also produced by the dynamics tools for selected variables: (a) metrics and (b) limit violations (examples are shown in the picture below). An in-depth description of the Project Analyzer is given in the 'Project Analyzer' section.

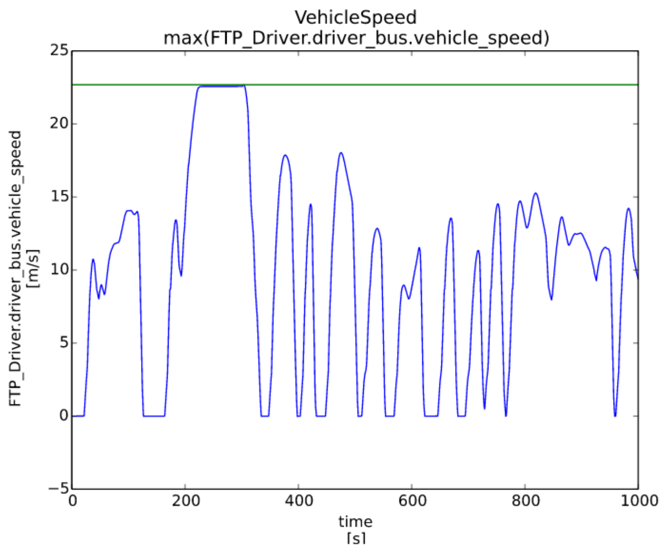


Figure 4: Example simulation result (vehicle speed)

7. Driveline Case Study

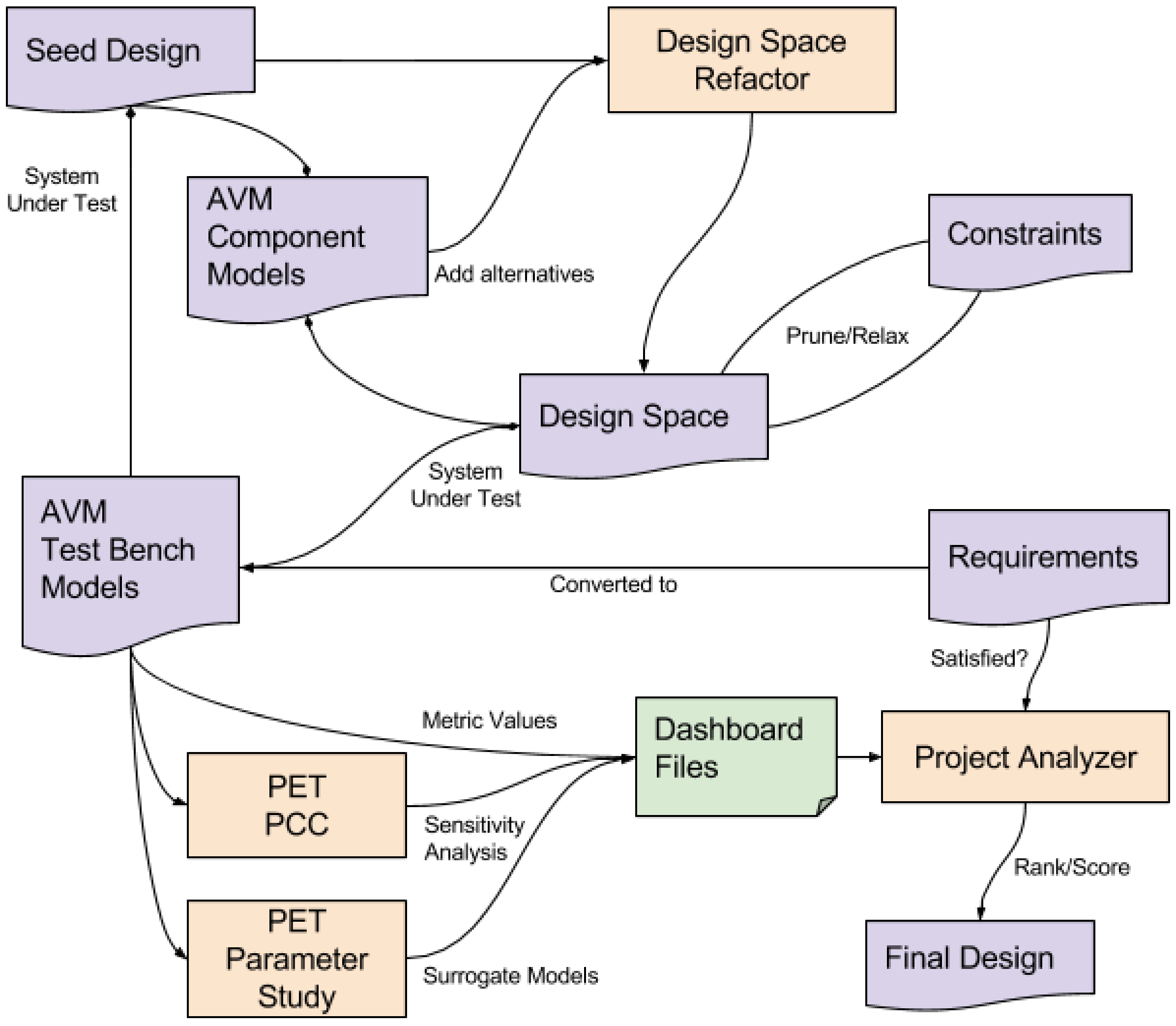
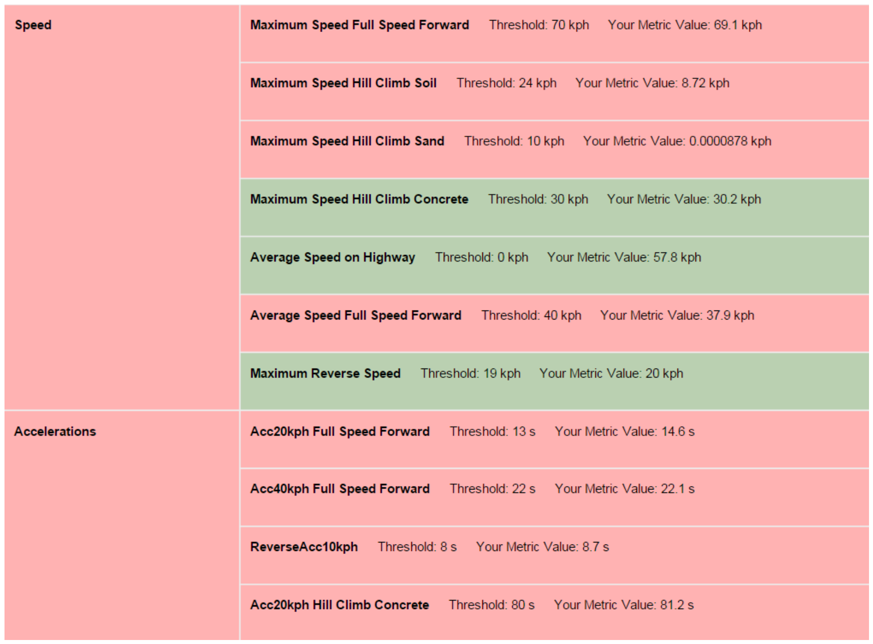


Figure 5: Design process

The 'Use Case/Design Flow/Case Study' section contains examples that span multiple domains and analysis tools. In this section, we present a possible design flow from the dynamics tools point of view, but not limited to the dynamics models. Our case study is the design of a vehicle driveline subject to a set of system requirements. The following sections walk through the stages of the design process for this driveline example to find the best possible design point with respect to the requirements. This use-case does not explore alternative architectures (e.g., hybrid driveline), but our example can easily be extended to include such alternatives.

7.1 Requirements

The driveline model has 19 Automotive Performance requirements that are divided into 5 subcategories: *Speed*, *Acceleration*, *Range*, *Temperature/Cooling*, and *Fuel Economy*. To illustrate the OpenMETA tool concepts, only 11 selected requirements are shown (Figure 6) from two categories: *Speed* and *Acceleration*. The *Speed* requirement category contains 5 maximum speed requirements (forward speed, hill climb on different surfaces, and reverse speed) and 2 average speed requirements (one on highway using US06[15] drive cycle profile and one under full speed forward drive profile). The *Acceleration* requirement category contains acceleration time to 20 km/h and 40 km/h, acceleration in reverse to 10 km/h, and acceleration to 20 km/h during hill climb on a concrete surface.



Speed	**Maximum Speed Full Speed Forward**	Threshold: 70 kph	Your Metric Value: 69.1 kph
	Maximum Speed Hill Climb Soil	Threshold: 24 kph	Your Metric Value: 8.72 kph
	Maximum Speed Hill Climb Sand	Threshold: 10 kph	Your Metric Value: 0.0000878 kph
	Maximum Speed Hill Climb Concrete	Threshold: 30 kph	Your Metric Value: 30.2 kph
	Average Speed on Highway	Threshold: 0 kph	Your Metric Value: 57.8 kph
	Average Speed Full Speed Forward	Threshold: 40 kph	Your Metric Value: 37.9 kph
	Maximum Reverse Speed	Threshold: 19 kph	Your Metric Value: 20 kph
Accelerations	**Acc20kph Full Speed Forward**	Threshold: 13 s	Your Metric Value: 14.6 s
	Acc40kph Full Speed Forward	Threshold: 22 s	Your Metric Value: 22.1 s
	ReverseAcc10kph	Threshold: 8 s	Your Metric Value: 8.7 s
	Acc20kph Hill Climb Concrete	Threshold: 80 s	Your Metric Value: 81.2 s

Figure 6: Results shown in requirement categories

Each requirement has a threshold and an objective value. In order to meet individual requirements, the metric value (i.e., the output of the Test Bench) must meet the threshold; the objective value is used for scoring (i.e., ranking the design points based on the utility function). The threshold defines the minimum value that must be reached and the objective defines the ideal value; exceeding the objective may not have any increased value, if the design point has already maximized the score.

7.2 Test Benches

[15] The US06 is a high acceleration aggressive driving schedule that is often identified as the "Supplemental FTP" driving schedule. (http://www.epa.gov/otaq/emisslab/methods/sc03col.txt)

As discussed above, CyPhy Test Benches are the executable versions of the requirements. Figure 7 depicts a CyPhy Test Bench model containing a driveline design, test components, environmental conditions, driver profile, 4 parameters (i.e., inputs), and 5 metrics (i.e., outputs). This Test Bench evaluates 5 of the requirements for a driveline design point. In order to evaluate other requirements, additional Test Benches are implemented in the OpenMETA tools; the collection of Test Benches that captures the entire set of requirements is defined as Set of Test Benches (SoT) as shown in Figure 8.

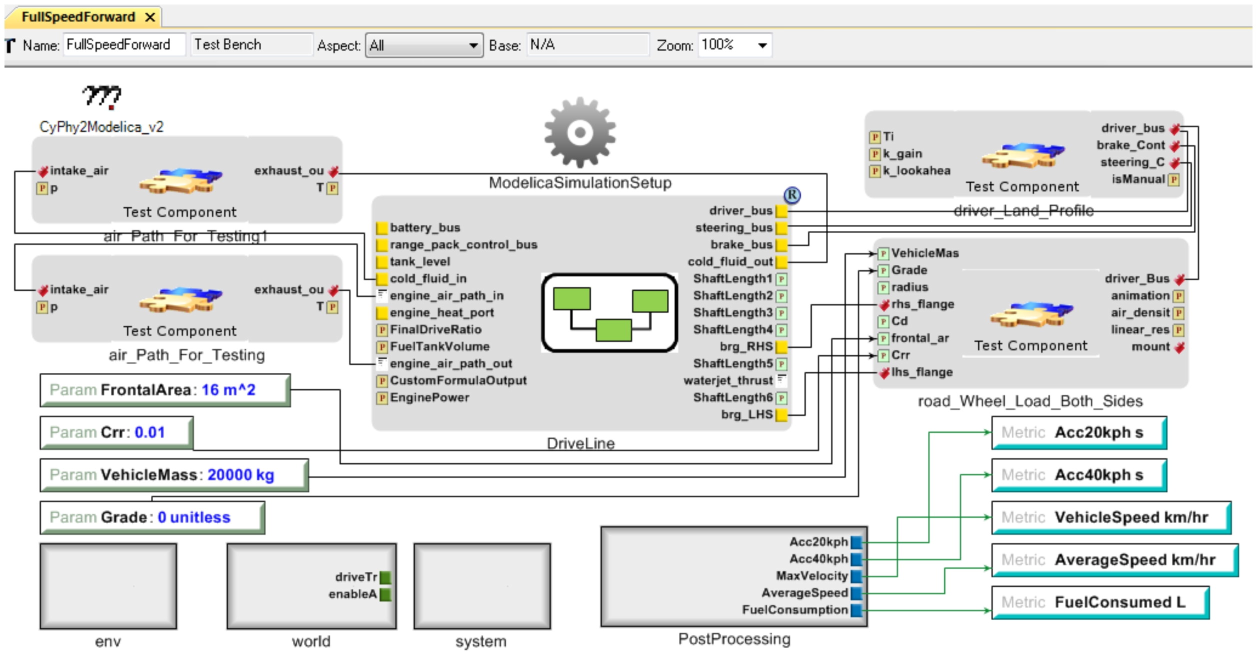


Figure 7: Full Speed Forward Test Bench

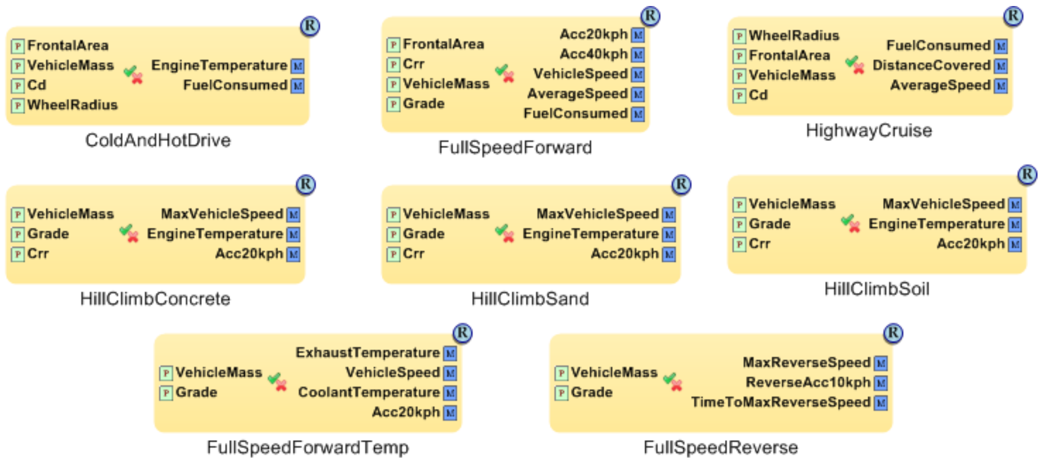
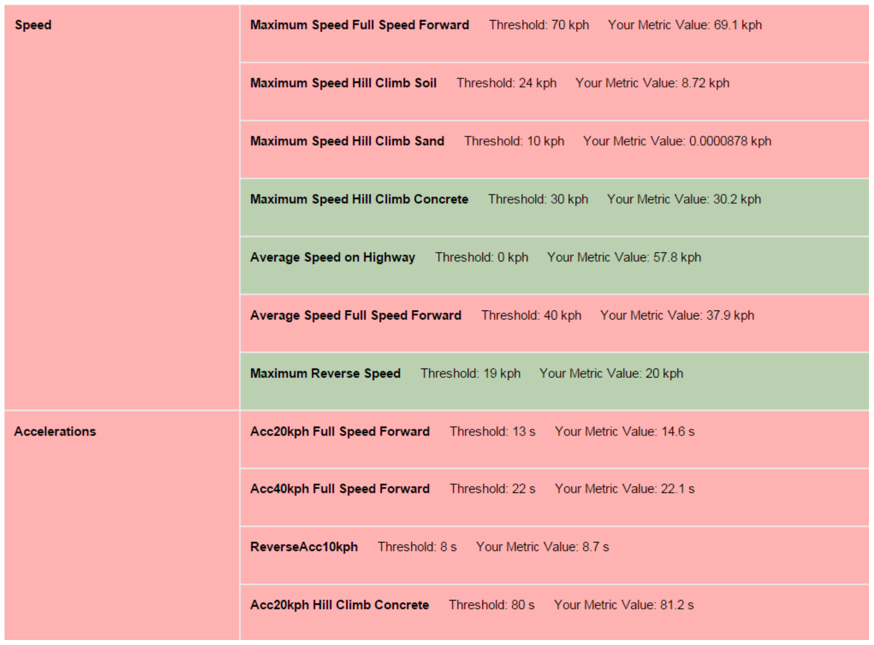


Figure 8: Set of Test Benches

By executing the SoT model, all requirements are evaluated for a single point design. The results of the Test Benches are shown against the requirements and requirement categories. As Figure 9 shows, only 3/11 requirements are met for this single design point.



Speed	**Maximum Speed Full Speed Forward**	Threshold: 70 kph	Your Metric Value: 69.1 kph
	Maximum Speed Hill Climb Soil	Threshold: 24 kph	Your Metric Value: 8.72 kph
	Maximum Speed Hill Climb Sand	Threshold: 10 kph	Your Metric Value: 0.0000878 kph
	Maximum Speed Hill Climb Concrete	Threshold: 30 kph	Your Metric Value: 30.2 kph
	Average Speed on Highway	Threshold: 0 kph	Your Metric Value: 57.8 kph
	Average Speed Full Speed Forward	Threshold: 40 kph	Your Metric Value: 37.9 kph
	Maximum Reverse Speed	Threshold: 19 kph	Your Metric Value: 20 kph
Accelerations	**Acc20kph Full Speed Forward**	Threshold: 13 s	Your Metric Value: 14.6 s
	Acc40kph Full Speed Forward	Threshold: 22 s	Your Metric Value: 22.1 s
	ReverseAcc10kph	Threshold: 8 s	Your Metric Value: 8.7 s
	Acc20kph Hill Climb Concrete	Threshold: 80 s	Your Metric Value: 81.2 s

Figure 9: Results shown in requirement categories

7.3 Single Design Point (Seed Design)

The single design point, often called seed design and used to hash out the initial component choices and general architecture, is built as an AVM Design Model (ADM) in CyPhy. The figure below shows the architecture, components and subsystems, of a single design point for the driveline model. The architecture consists of a cooling system, controllers (Engine Control Unit and Transmission Control Unit), left-hand and right-hand side drive (each includes a drive shaft and a final drive), two surrogate fluid models (air path and fluid sink), a battery, a fuel tank, an engine (power plant), and a transmission (gearbox).

The key components from the *Speed* and *Acceleration* requirements category point of view are the engine and the transmission components. We focus only on these two key components to improve the performance characteristics of the driveline design in order to meet all requirements. The single design point in Figure 10 has a Deutz BFM1015M (290HP) engine and an Allison X200 4A (4 forward gears) transmission. It is also possible to consider different engine and transmission alternatives by using the OpenMETA tools and programmatically turn the single design point into a design space.

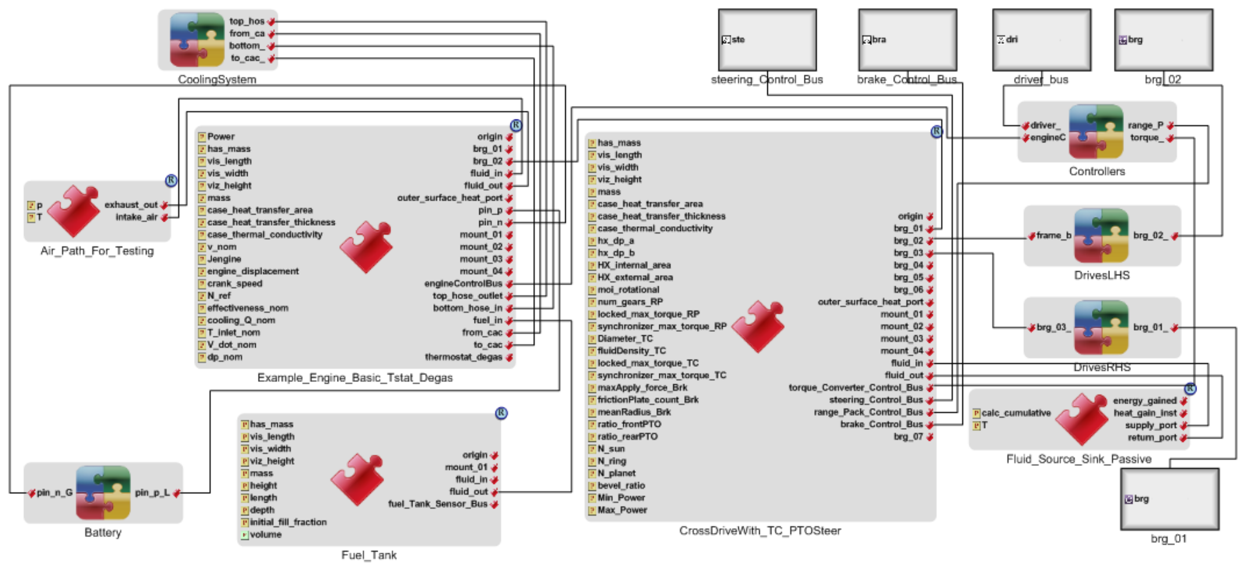


Figure 10: Single design point for the driveline model

7.4 Discrete Design Space

The discrete design space resulting from the single design point has the exact same hierarchical breakdown structure. The discrete design space for the driveline model shows that for the engine and the transmission, users can consider alternative components. In other words, different engine and transmission AVM Component Models (ACM) can be added and all possible design points are encoded by this design space. The tables below summarize some of the engine and transmission options and their important parameters that we considered based on publicly available datasheets from the manufacturers' websites. The highlighted transmission and engine instances are used in the original seed design.

Supplier	Type	Min HP	Max HP	# of forward gears	# of reverse gears	Max RPM
Allison	***X200-4A***	***206.36***	***344.38***	***4***	***1***	***2800***
Allison	X200-4B	239.86	399.32	4	2	2800
Allison	XT1410-4	473.02	787.92	3	1	2400
Allison	XT1410-5A	473.02	787.92	3	1	2400
Allison	XTG411-2A	213.06	355.1	4	2	2300
Allison	XTG411-4	340.36	566.82	4	2	2500

Table 1: Transmission alternatives

Supplier	Type	HP	# of cylinders	Angle	Displacement (l)
Caterpillar	C9 280kW	375	6	inline	8.8
Caterpillar	C11 313kW	420	6	inline	11.1
Caterpillar	C15 444kW	595	6	inline	15.2
Caterpillar	C18 597kW	800	6	inline	18.1
Caterpillar	C27 597kW	800	12	V	27.03
Caterpillar	C32 709kW	950	12	V	32.1
MTU	MT883 644kW	864	12	V-90	27.36
MTU	6V199 261kW	350	6	V-90	11.9
MTU	6V199 335kW	455	6	V-90	11.9
MTU	6V199 430kW	585	6	V-90	11.9
MTU	8V199 530kW	720	8	V-90	15.9
MTU	8V199 603kW	820	8	V-90	15.9

MTU	8VMT881 736kW	1000	8	V-90	18.2
MTU	12VMT883 1103kW	1500	12	V-90	27.4
MTU	6R106Euro3 240kW	325	6	inline	7.2
Deutz	***BF6M1015M group A***	***290***	***6***	***V***	***11.91***
Deutz	BF6M1015MC group A	385	6	V	11.91
Deutz	TCD2015V6M group A	440	6	V	11.9
Deutz	TCD2015V8M group A	600	8	V	15.9
Cummins	QSM 350HP FR20019	350	6	inline	10.8
Cummins	QSM 400HP FR20003	400	6	inline	10.8
Cummins	QSX 400HP FR10581	400	6	inline	15
Cummins	QSX 500HP FR10583	500	6	inline	15

Table 2: Engine alternatives

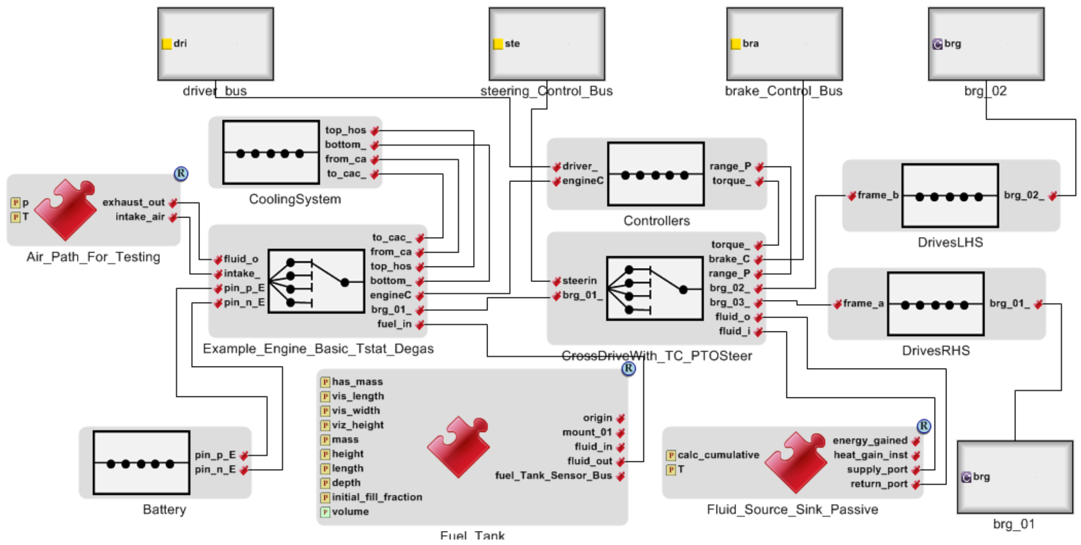


Figure 11: Discrete design space for driveline model

The discrete design space in Figure 11 contains 25 engine alternatives and 8 transmission alternatives yielding 200 configurations, if we explore all *possible* combinations as shown in Figure 13. Adding alternative components for every design container can quickly explode the number of configurations. As a direct result of the large number of configurations, the time required for executing all Test Benches (8 in this examples, but there could be many more) over the entire design space becomes impossible. Elaborating all *possible* configurations is not scalable and is unnecessary, because there are many engine/transmission combinations that cannot be physically realized. Using design space constraints, we can prune the combinatorial design space to a smaller (manageable) set of designs which allows us to elaborate only the *viable* configurations; this is a powerful technique, and saves precious time and computational resources. All constraint types are presented in the 'Constraint-based Architecture Exploration' section. For the driveline model we used two property constraints: the engine output power rating must be within the range of the transmission's minimum and maximum input power rating as shown in Figure 12.

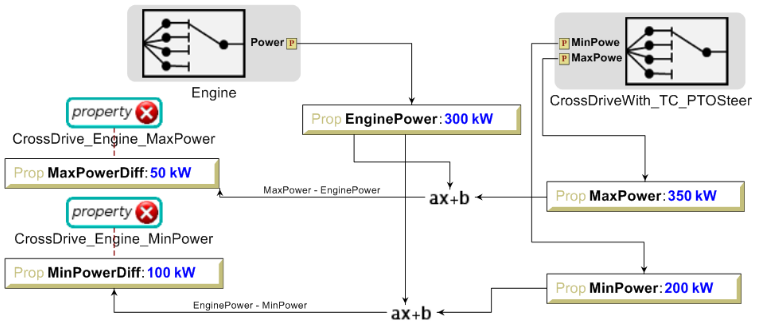
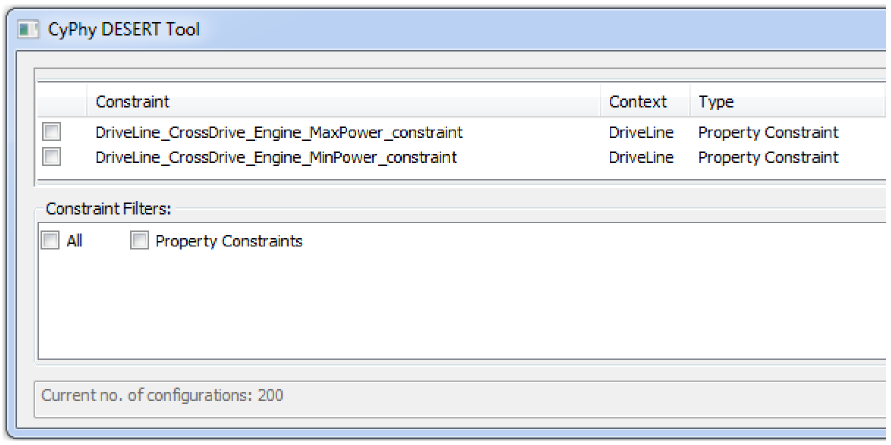


Figure 12: Example of property constraints


CyPhy DESERT Tool

	Constraint	Context	Type
☐	DriveLine_CrossDrive_Engine_MaxPower_constraint	DriveLine	Property Constraint
☐	DriveLine_CrossDrive_Engine_MinPower_constraint	DriveLine	Property Constraint

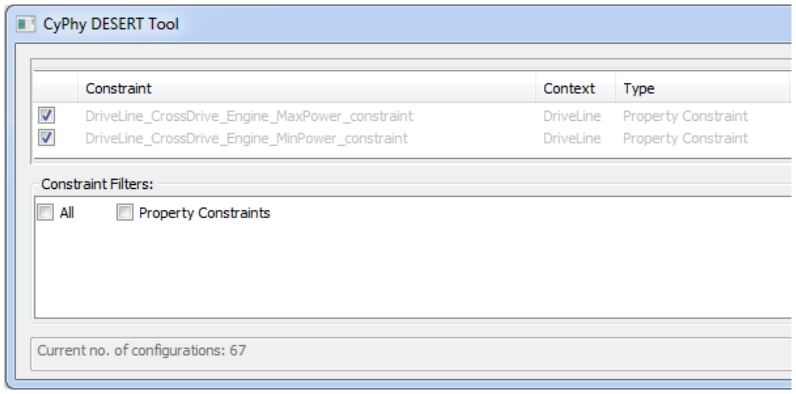
Constraint Filters:

☐ All ☐ Property Constraints

Current no. of configurations: 200

Figure 13: Number of possible configurations for design space

Each constraint type is translated to the Object Constraint Language (OCL). After all constraints are translated to OCL and gathered, the design space exploration tool (DESERT) allows users to apply some or all constraints, including the ability to group constraints and apply such a group. If we apply both transmission input power constraints the *possible* number of configurations (200) drops down to 67 *viable* configurations as shown in Figure 14.


CyPhy DESERT Tool

	Constraint	Context	Type
☑	DriveLine_CrossDrive_Engine_MaxPower_constraint	DriveLine	Property Constraint
☑	DriveLine_CrossDrive_Engine_MinPower_constraint	DriveLine	Property Constraint

Constraint Filters:

☐ All ☐ Property Constraints

Current no. of configurations: 67

Figure 14: Number of configurations after all constraints applied

The next step of this design process is to evaluate all requirements for every *viable* generated design point. 8 CyPhy Test Benches already exist for the original seed design. The OpenMETA tools support the reuse of Test Benches for an entire design space by changing the Top Level System Under Test (TLSUT) object in every Test Bench to point to the newly created design space container. This approach produces Test Bench templates for any design point within the discrete design space. A Set of Test Benches can also be used with a discrete design space to execute all Test Benches as shown in Figure 15 below.

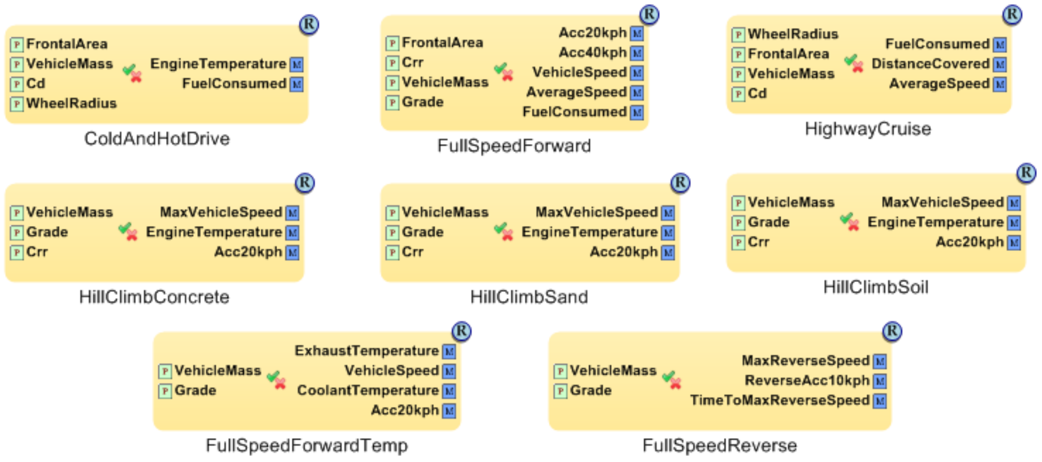


Figure 15: Set of Test Benches for the design space

Running all Test Benches over a design space may take some time, but after all results are available the Project Analyzer tool is used to visualize all Test Bench results (e.g., metrics) across the discrete design space. The Project Analyzer tool helps to understand which designs meet the requirements and which ones do not (and by how much). Figure 16 shows each design as a line on a parallel axis plot and the lines are color coded by green if the design meets the requirement and by red if the design does not meet the requirement. On each vertical axis the threshold values (red) and the objective values (green) are marked for each metric. A filter can be applied between for each metric and a subset of the designs get highlighted. The Project Analyzer supports color coding based on limit violations (see Figure 17) and based on ranking (see Figure 18). All of these capabilities guide the users to make decisions about which design points should be considered for further detailed analyses. In our use case we selected configurations 2, 4, 7, 30, and 43 for parametric design exploration. Note: the original seed design is configuration 41.

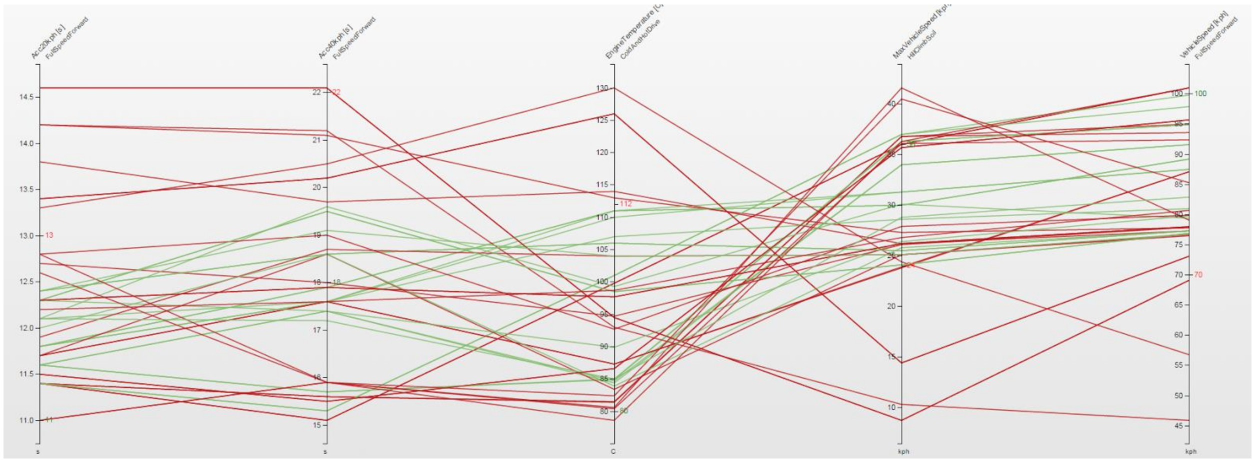


Figure 16: Requirement violations for design points

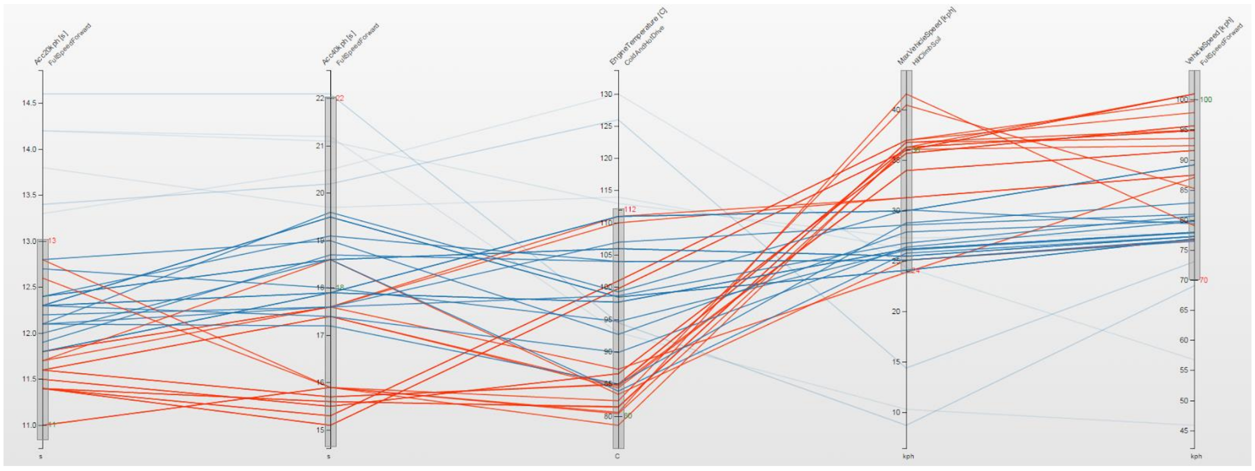


Figure 17: Limit violations for design points

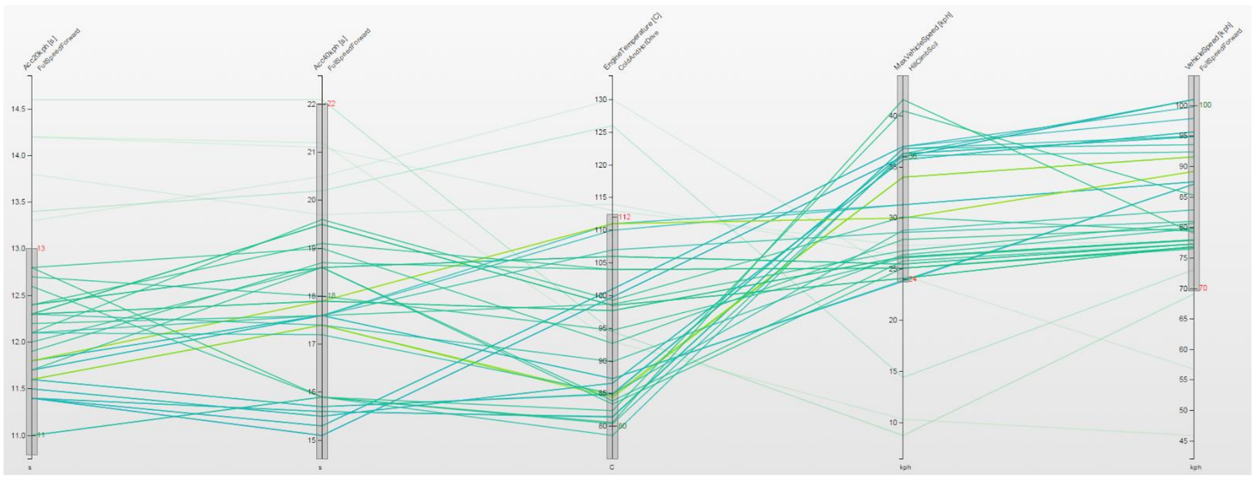


Figure 18: Ranking of design points

7.5 Parametric Design Space

In the previous section the discrete design space exploration was presented in the context of the driveline model. The OpenMETA tools support parametric design space exploration through the Parametric Exploration Tool. The parametric analyses takes significantly more time than the discrete since the Test Benches are executed multiple times for a single design point. These parametric analyses are used to assess the robustness of the designs and to generate surrogate models for the Test Benches. For instance, even if a design meets all requirements using the mean values of component parameters and environment conditions, it is important to see the impact on the metric values (i.e., output of the Test Benches) if parameters (i.e., inputs of the Test Benches) have variations or if they can change within a certain range during normal operating conditions.

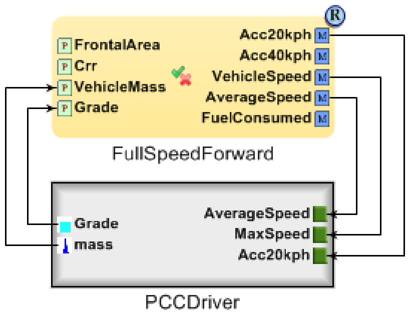


Figure 19: Parametric design space exploration model

A Probabilistic Certificate of Correctness parametric driver is set up for the *Full Speed Forward* Test Bench as shown in Figure 'Parametric design space exploration model'. The *grade* and the *mass* parameters are defined as probability density functions (PDF), where the *grade* has a uniform distribution between a minimum and a maximum value and the *mass* has a normal distribution with a mean and a variance. Figure 20 shows the impact on the outputs (acceleration to 20 km/h, average speed, and vehicle speed) of the Test Bench. The minimum values and the acceptable PCC target values are defined in the parametric exploration model by the user and they are visualized on the plots. The PCC value for each output is the area below the output pdfs within the minimum and maximum range. The results below are shown for a single design point.

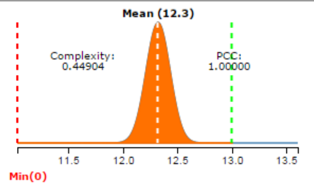
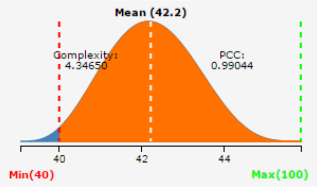
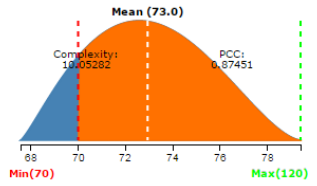
Metric	Probability Distribution Function
Acc20kph	
AverageSpeed	
VehicleSpeed	

Figure 20: Results of parametric design space exploration

7.6 System Robustness

PCC experiments can be defined for multiple Test Benches as shown in Figure ‘Parametric design space exploration models’, which means a system robustness analysis is performed for multiple requirements. For each Test Bench one “joint” PCC value is computed based on the individual PCC values for each metric. The joint PCC is always a real number with a value between 0 and 1.

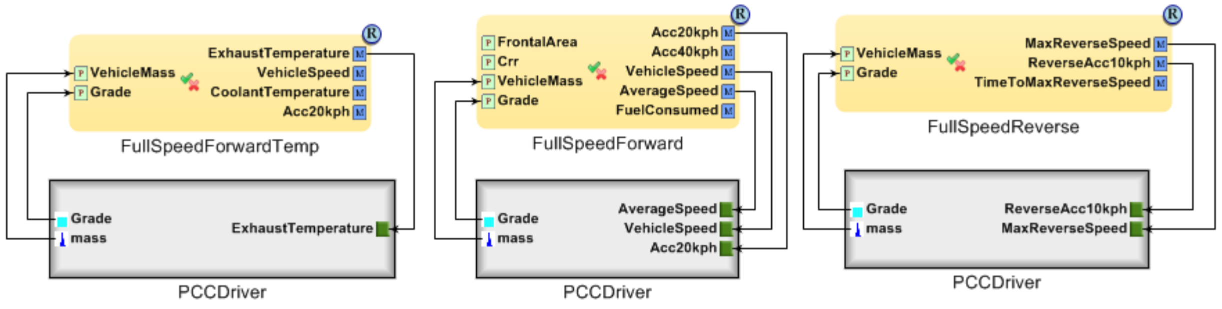


Figure 21: Parametric design space exploration models

The Parametric Design Exploration using PCC drivers can be applied for an entire discrete design space, but these analyses might consume valuable resources to (unnecessarily) perform in-depth analyses on design configurations which do not satisfy design requirements. Based on initial high-level system analyses, we have chosen only three Test Benches and five promising configurations (2, 4, 7, 30, and 43) to further analyze using PCC. The Project Analyzer visualizes the PCC results for multiple Test Benches and configurations in the form of a heat map. Figure 22 depicts the robustness of each design point with respect to each Test Bench. Based on this figure, configuration 30 has the highest accumulated PCC value across all three selected Test Benches, therefore configuration 30[16] is the most robust design in this set.

[16]Configuration 30 uses the Cummins QSM FR20019 (350HP) engine and the Allison XTG411 2A transmission.

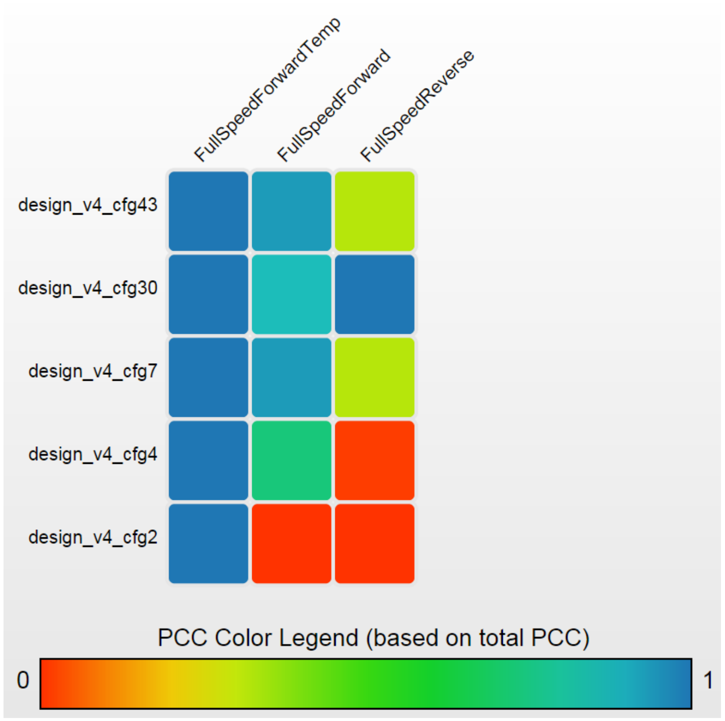


Figure 22: Results of parametric exploration over a discrete design space

7.7 Surrogate Model and Prediction Profiler

The OpenMETA tools support surrogate model generation from a parametric exploration model that contains a parameter study (i.e., design of experiment) driver. Generating surrogate models is computationally intensive, but after a surrogate model is generated, interactive 2D and 3D plots can be generated for the users to predict the metrics within the parametric space where the surrogate model is valid.

Configuration 30 was selected based on the PCC experiments. For configuration 30 a surrogate model was created by the OpenMETA tools. The results are shown in Figure 23. On the left-hand side, two parameters are selected to plot the coefficient of rolling resistance (Crr) and the frontal area of the vehicle. The prediction profiler estimates both the $0 \rightarrow 20$ km/h acceleration time and the maximum vehicle speed based on the selected input parameters. On the right-hand side, a 3D response surface is plotted where $0 \rightarrow 40$ km/h acceleration time is shown w.r.t. the coefficient of rolling resistance and the frontal area of the vehicle. Users can choose which parameters (inputs) and metrics (outputs) are plotted using the Project Analyzer.

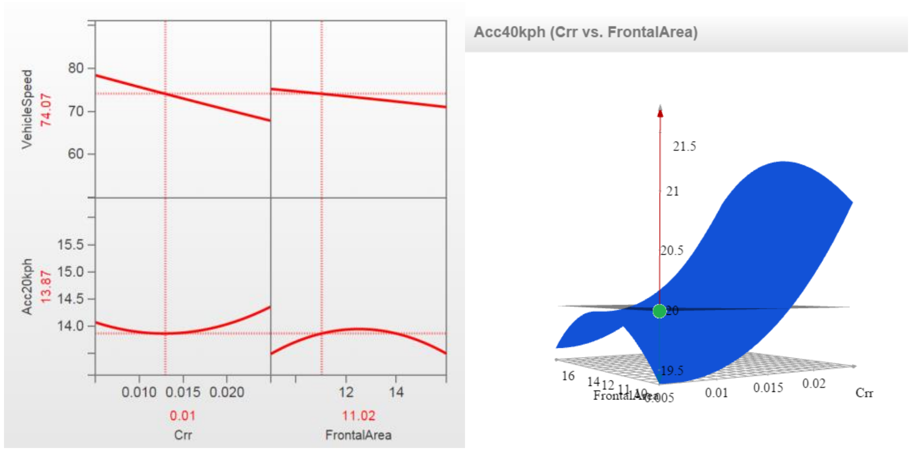
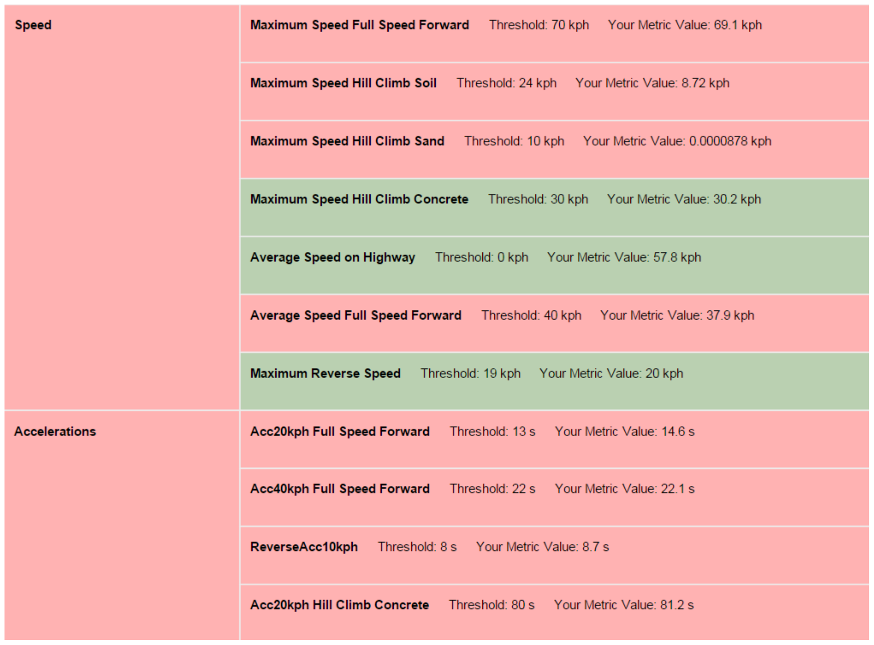


Figure 23: Prediction profiler and constraint plot

7.8 Conclusion

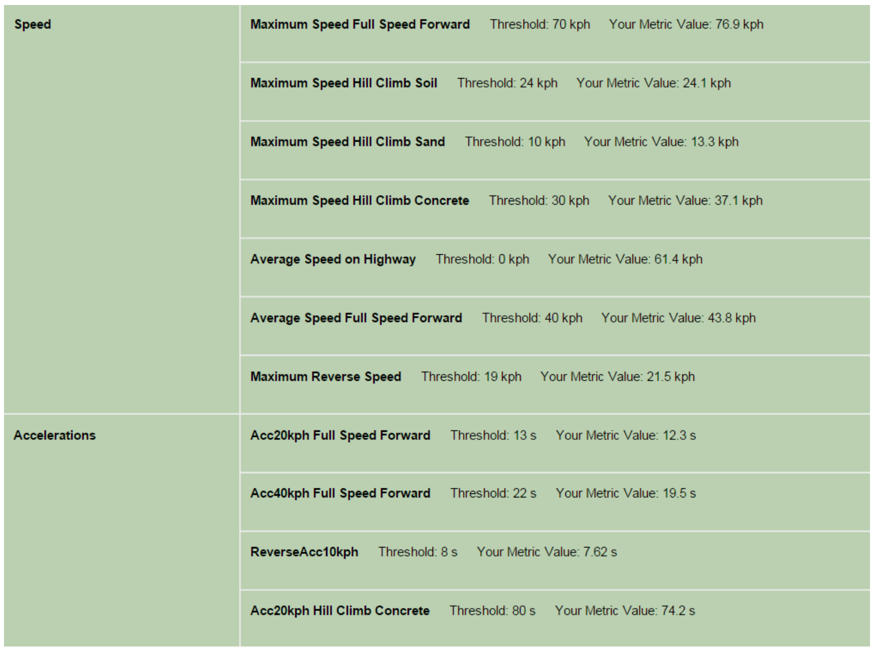
The original driveline design point (seed design) does not meet the requirements as shown in Figure 24. This driveline seed design is turned into a design space and alternative engine and transmission components are added. The discrete design space generates 200 *possible* configurations, and after constraints are implemented 67 *viable* configurations are left in the pruned design space.



Speed	**Maximum Speed Full Speed Forward**	Threshold: 70 kph	Your Metric Value: 69.1 kph
	Maximum Speed Hill Climb Soil	Threshold: 24 kph	Your Metric Value: 8.72 kph
	Maximum Speed Hill Climb Sand	Threshold: 10 kph	Your Metric Value: 0.0000878 kph
	Maximum Speed Hill Climb Concrete	Threshold: 30 kph	Your Metric Value: 30.2 kph
	Average Speed on Highway	Threshold: 0 kph	Your Metric Value: 57.8 kph
	Average Speed Full Speed Forward	Threshold: 40 kph	Your Metric Value: 37.9 kph
	Maximum Reverse Speed	Threshold: 19 kph	Your Metric Value: 20 kph
Accelerations	**Acc20kph Full Speed Forward**	Threshold: 13 s	Your Metric Value: 14.6 s
	Acc40kph Full Speed Forward	Threshold: 22 s	Your Metric Value: 22.1 s
	ReverseAcc10kph	Threshold: 8 s	Your Metric Value: 8.7 s
	Acc20kph Hill Climb Concrete	Threshold: 80 s	Your Metric Value: 81.2 s

Figure 24: Requirement evaluation for the seed design (maps to Configuration 41 in the design space)

All requirements are evaluated over the 67 *viable* configurations in the discrete design space. Many design points meet all requirements, and 5 of them are selected for further parametric design space analysis on system robustness field. Finally, configuration 30 is chosen to generate a surrogate model for the driveline design problem. The surrogate model is used to predict the key performance parameters of the system under different conditions. Figure 25 shows that all requirements are met for this driveline design configuration.



Speed	**Maximum Speed Full Speed Forward** Threshold: 70 kph Your Metric Value: 76.9 kph
	Maximum Speed Hill Climb Soil Threshold: 24 kph Your Metric Value: 24.1 kph
	Maximum Speed Hill Climb Sand Threshold: 10 kph Your Metric Value: 13.3 kph
	Maximum Speed Hill Climb Concrete Threshold: 30 kph Your Metric Value: 37.1 kph
	Average Speed on Highway Threshold: 0 kph Your Metric Value: 61.4 kph
	Average Speed Full Speed Forward Threshold: 40 kph Your Metric Value: 43.8 kph
	Maximum Reverse Speed Threshold: 19 kph Your Metric Value: 21.5 kph
Accelerations	**Acc20kph Full Speed Forward** Threshold: 13 s Your Metric Value: 12.3 s
	Acc40kph Full Speed Forward Threshold: 22 s Your Metric Value: 19.5 s
	ReverseAcc10kph Threshold: 8 s Your Metric Value: 7.62 s
	Acc20kph Hill Climb Concrete Threshold: 80 s Your Metric Value: 74.2 s

Figure 25: Requirement evaluation for Configuration 30

Using the OpenMETA toolchain comes with several benefits including:

- all models are stored and maintained in one environment (CyPhy)
- the component models can capture properties from multiple domains (e.g., dynamics and solid models; this is extremely advantageous when changing a single property value will affect behavior in multiple domains)
- turning a single design point into a discrete design space is a fully automated process
- if a new component type or instance becomes available from a supplier, then it is an easy and straightforward process to add it to the existing discrete design space
- adding alternative components and creating alternative architectures requires minimal effort compared to the traditional design approaches
- all Test Benches defined for a single design point are completely reusable over the resulting design space
- the Test Benches can reference either a single design point or a design space that can evolve over time,
  - OpenMETA tools can automatically execute all Test Benches, and evaluate requirements at any point during the design process

8 Evaluation

In the previous section, an end-to-end demonstration of the OpenMETA design flow was presented for a driveline design problem. Designers begin with a system mockup, which included a power plant (a diesel engine), a gearbox, drive shafts, final drives, and peripheral cooling and electrical components. This is typical design approach, in which a general architecture "prototype" is selected to satisfy the early system architecture requirements, and then the prototype is tested in many scenarios to evaluate the early choices. Those initial tests reveal that the prototype is not satisfactory, so it is augmented with alternative components (for the power plant and gearbox), in addition to constraints relating to the compatibility of the various component combination choices. All tests are executed for all viable configurations, and any configuration that fails to meet the requirements is eliminated from further consideration. Based on outstanding performance, a few configurations are selected for parametric sensitivity analysis (PCC) to evaluate the robustness of their performance. Finally, the most robust configuration is analyzed in a design of experiment using a surrogate model, which shows that configurations expected behavior over a range of possible input parameters in the form of a response surface and constraint plots.

In the example above, it should be noted that the system architecture is identical across all 200 configurations. Also, despite there being several (7-8) distinct component types in the design, only two are augmented with alternative instances; it would be a simple matter to add alternatives for the remaining component types. There are a few more important points to make, which may not be obvious from the description above. First, the initial system mockup does not need to use models of off-the-shelf components, as we did in the example; components or subsystems of the prototype mockup may be represented by crude approximations (e.g., polynomials or lookup tables). Moreover, those component or subsystem placeholders may also be used to represent distinct architecture alternatives. For example, the subsystem composed of gearbox, drive shafts, and final drives may be replaced by an electric drive train consisting of a generator, power converters, power transmission cables, and electric motors, while utilizing the same power plant to turn the generator input. The system requirements can drive the design choices for any type of architecture, allowing designers to compare fundamentally different design alternatives, *without ever building a physical prototype*.

The second point is also powerful: if a component or subsystem is not yet elaborated using an off-the-shelf component instance (or set of instances), the parametric analysis techniques (e.g., design of experiment using surrogate models) may be used earlier in the design process, prior to design space creation or exploration. In some cases, designers may have an idea for a component which currently does not exist or is not yet in mass production. For example, weight requirements on the final design, when combined with cost constraints on certain component-types (e.g., a cost requirement necessitating re-use of previously acquired engines), may result in the conclusion that a new driveshaft and chassis material should be explored. In this way, *the use of surrogate models and parametric prototypes may spark ideas for novel component designs*.

Finally, the driveline example focuses completely on the lumped-parameter dynamics domain models. In a real-world design problem, there is other domain behavior that must be evaluated, in addition to the cross-domain interactions. This brings to light another crucial benefit provided by CyPhy and the OpenMETA toolchain: all the techniques from the example above can be extended to all other domains, and new Test Bench types may target different domain requirements. Using the Project Analyzer, the results from one domain analysis can be viewed alongside results from a separate domain, quickly revealing problems arising from the systems integration process. Additionally, using Sets of Test Benches (SoTs), it is a simple matter to propagate the result from one domain analysis to be used as an input to another domain analysis. In a vehicle design example, a geometric model analysis will reveal the dimensions, mass, and center of gravity (CG) of a driveline design, which can then be automatically used in a dynamics analysis to evaluate the effects of those system properties and answer important questions:

- How will various suspension components perform and affect ride quality?
- Will the vehicle be agile in terms of acceleration, braking, and steering?
- Given its dynamics performance capabilities, how easily will the vehicle roll over?

Such questions can only be answered through cross-domain propagation and analysis, and it should be clear that *design choices in one domain may affect performance in other separate domains in unexpected ways*. The benefit provided by the OpenMETA design tools of automatically detecting and quantifying these cross-domain interactions when they are introduced cannot be overstated.

The choice of Modelica to model the dynamics domain behavior satisfied several AVM-related design objectives. Model and model library versions are easily tracked and updated due it Modelica’s succinct textual representation of physical behavior. The acausal (equation-based) format makes it possible to easily define new component models, and to compose systems of components logically and quickly without close scrutiny as to the directionality of connections. There are several Modelica analysis tools available for simulation execution, both free and commercial. These provide capabilities to verify model validity through strong type checking and through compile-time warning/error messages. Finally, there are many readily available model libraries, both open-source (e.g., C2M2L, Modelica Standard Library) and commercial (e.g., Modelon’s Vehicle Dynamics Library), which can easily be integrated with OpenMETA.

The META design process and the OpenMETA tools can provide significant increases in productivity as presented above; one important contributing design objective was to minimize any overhead on model composition, generated models, and runtime. The Table ‘CyPhy2Modelica model generation time and overhead’ summarizes the time required to generate Modelica models and the overhead on the number of equations compared to the original Modelica implementation for a few designs. Over all, the tools scale well with the size of the models, saving time by reducing laborious model development. The overhead introduced in the dynamics model composition and analysis tools is justified by the benefits to the user in the form of rapid virtual prototyping, model debugging capabilities, and large-scale automation of analysis execution. For example, the increased model generation runtime for the ‘Excavator with

Cyber controllers' model is a result of the running three analysis interpreters in series, including compiling generated C code for the controllers.

Model	# of AVM Components (ACMs)	# of Test Components	Model generation time (s)	# of generated equations	# of equations original Modelica impl.
Rolling Wheel	3	3	0.4	44	44
RI Circuit	4	2	0.6	44	44
Driveline	15	2	1.2	4385	4385
Driveline with design space	15	2	1.4	4431	4385
Driveline with suspension	74	1	3.2	84445	N/A
Excavator	31	9	2	11177	N/A
Excavator with Cyber controllers	55	13	22.2	11839	N/A

Table 3: CyPhy2Modelica model generation time and overhead

The dynamics tools were used extensively by beta testers, gamma testers, FANG competitors, and other AVM performers, in addition to use in the C2M2L Component curation process. Feedback from these use-cases was constantly incorporated by Vanderbilt developers to improve the robustness and usability of the tools.