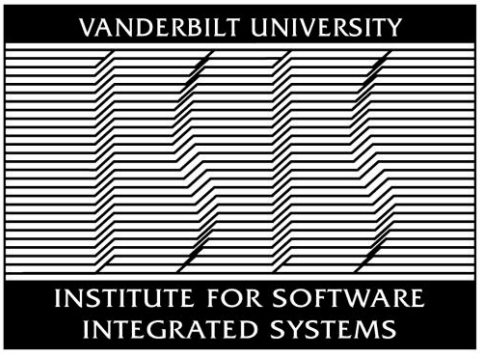

Institute for Software-Integrated Systems

Technical Report

TR#: **ISIS-15-105**

Title: **Architecture Exploration in the META Toolchain**

Authors: **Himanshu Neema, Sandeep Neema and Ted Bapty**

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

List of Tables

List of Acronyms and Abbreviations

CPS	Cyber-Physical System
CyPhyML	Cyber-Physical Modeling Language
DESERT	Design Space Exploration Tool
DSE	Design Space Exploration
DSEM	Design Space Exploration and Manipulation
PET	Parametric Exploration Tool
GME	Generic Modeling Language
DSML	Domain-Specific Modeling Language
CAD	Computer-Aided Design
FEA	Finite-Element Analysis
CFD	Computational Fluid Dynamics
CAE	Computer Aided Engineering
CAT	Component Authoring Tool
CLM	Component Library Manager
MI	Master Interpreter
DS	Design Space
CA	Component Assembly
DC	Design Container
DE	Design Element
CO	Component
COR	Component Reference
DCC	Design Container of Compound type
DCA	Design Container of Alternative type
DCO	Design Container of Optional type
RF	Root Folder
IFV	Infantry Fighting Vehicle
ISG	Integrated Starter Generator Assembly

1.0 Summary

Cyber-Physical Systems (CPS) [1] are engineered systems that require tight interaction between physical and computational components. Designing a CPS is highly challenging [2] because these systems are inherently complex, need significant effort to describe and evaluate a vast set of cross-disciplinary interactions, and require seamless meshing of physical elements with corresponding software artifacts. Moreover, a large set of architectural and compositional alternatives must be systematically explored and evaluated in the context of a highly constrained design space. The constraints imposed on the selection of alternatives are derived from the system's functional, performance, dimensional, physical, and economical objectives. Furthermore, the design process of these systems is highly iterative and requires continuous integration of design generation with design selection and manipulation supported by design analyses.

To enable the iterative design process for CPS-s, we have developed a design tool chain, OpenMETA [4] [9], built around a Domain-Specific Modeling Language (DSML) [3], called the Cyber-Physical Modeling Language (CyPhyML). In this report, we describe the elements of OpenMETA that deal with Design Space Exploration and Manipulation (DSEM) for CPS-s. These parts of the tool chain collectively provide modeling methods and tools for exploration and visualization of designs and design spaces, solving complex design constraints, and effective management of the design spaces and designs.

In particular, the report will cover the key language aspects dealing with design space construction and specification of design constraints arising from component interactions and due to functional and practical system requirements. Also, we discuss our approach to effectively manage large-scale design spaces that are pervasive in META design problems. We provide details of key design space tools that were developed in META to support the above process. We also describe the iterative design process that is crucial for designing complex real-world systems and highlight how META's design space tools support it. Further, we discuss how designs are evaluated and how design space is evolved using the analysis results with the help of design space tools. We also present a detailed case study to exemplify our approach and usage of the design space tools.

1.1 Overview

Design Space Exploration for complex CPS cannot be realized as a closed form analytical search procedure and requires multiple techniques, at multiple abstractions and fidelity, and involves complex iterations. Existing engineering practice follows the classical systems engineering design process model, which includes requirements and design iterations, with system analysis and controls development going in parallel. However, in the context of large-scale CPS-s, the re-design cycles require tremendous amount of work, time, and resources. This is further exacerbated by pushing the verification and validation to the testing phase. Not only does that introduce errors much later in the design phase, it also lends the design process itself

highly inflexible to changes in requirements and implementation technologies, which are commonplace in the modern continually evolving globally connected world. What is needed is a design process that is not only able to easily adapt changes in the design requirements, but is also tied to an integrated testing framework that allows for continual design verification and evolution of the design space itself to produce better and faster designs.

The OpenMETA tool chain provides unique capabilities in this respect and incorporates a comprehensive suite of methods for design space exploration such as discrete combinatorial design space exploration, parametric design analysis, simulation based design metric evaluation, and dashboard for design space metric visualization.

1.2 Goal

Traditional design processes employ the waterfall approach where the requirements definition and design phase precedes the testing and evaluation of designs. The main problems with the traditional approach are:

- Designs developed are purposed for use of a particular use-case and need substantial re-work if the design needs to be adapted for other applications.
- Problems are discovered much later during the system integration phase and so the optimal system architecture and components may not be determined earlier in the design process.
- Requirements often change during the design process, which often requires re-designing the systems.

Our goal for design process in META is to provide a comprehensive set of tools that enable easier design exploration and adaptation, support integrated testing and evaluation, and enable near-continuous design evolution for updates in requirements as well as using feedback from design analyses.

1.3 Summary of Approach

Designing of complex cyber-physical systems is not a straightforward process from requirements specification to a full design; rather it is an iterative process where the generation of design configurations must interplay with design requirements as well as design analyses.

Our approach to solve this problem is to provide tools for designers to effectively explore the design spaces and manage design requirements that may change from day-to-day. Our architecture design language, Cyber-Physical Modeling Language (or CyPhyML for short) allows for modeling design spaces and specifying design constraints. Our design space exploration tool allows for systematic exploration of design spaces to prune design configurations that do not satisfy design constraints. This tool allows for the selective application of design constraints to provide detailed feedback of design constraints on validity of design choices in the design spaces. Further, we provide a set of tools to manipulate the designs and design spaces in various ways to help designers to evolve the design spaces as the requirements

change. Also, we provide integrated testing framework for analyzing and visualizing designs and various tools to incorporate the results of those analyses to adapt the design spaces.

Tel (615) 343-7472 Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

2.0 Introduction

2.1 Design Space Exploration of Cyber Physical Systems

Cyber-Physical Systems (CPS) [1] are systems that require tight interaction between physical and computational components. These systems span several engineering domains such as mechanical, electrical, thermal, and cyber. CPS design is highly challenging [2] because these systems are inherently complex, need significant effort to describe and evaluate a vast set of cross-domain interactions, and require seamless meshing of physical elements with corresponding software artifacts. CPS-s involve a large set, or design space, of architectural and compositional alternatives that must be systematically explored and evaluated. The design spaces are formulated with these alternatives and design constraints that limit the selection of design elements to those that satisfy them. These basic constraints are derived from the system's functional (e.g., gas, electric, or hybrid drivetrain), performance (e.g., minimum torque of engine), dimensional (e.g., maximum height or capacity), physical (e.g., weight, join structures), and economical (e.g. cost) objectives.

Good system design must further consider several factors such as manufacturability, stability, complexity, reliability, risks, time-to-market, etc. However, we consider these factors as secondary, coming after the basic set of constraints is satisfied. The discrete selection of compositional and architectural alternatives based on these constraints form the initial Design Space Exploration (DSE). The generated configurations are subjected to dynamic analyses for evaluation against the secondary requirements. The result of these detailed system analyses in terms of valid design selections and reformulations must be incorporated into the original design space, which must be re-explored to generate a new set of valid design configurations. This iterative nature of the design process with strong bidirectional coupling between design activities and system analysis and verification is a key requirement for CPS design.

2.2 Design Space Exploration in META Tool chain

To enable the iterative design process for CPs-s, we have developed a design tool chain, that we call *OpenMETA* [4] [9] (see Figure 1), built around a Domain-Specific Modeling Language (DSML) [3], called the Cyber-Physical Modeling Language (CyPhyML). The CyPhyML captures integration interfaces of system components across multiple design domains (e.g., *Cyber*, *CAD*, *and FEA*) as well as generic assembly rules given in terms of compositional and architectural alternatives and *hard* design constraints for the final assembly. OpenMETA supports multi-level and multi-fidelity exploration of system-level architectural and parametric tradeoffs. These tools facilitate the iterative design process by integrating formal qualitative reasoning methods, DSEM, and automated dynamics and structural analyses of designs. In this paper, we focus only on tools that support design space exploration and manipulation for CPS-s.

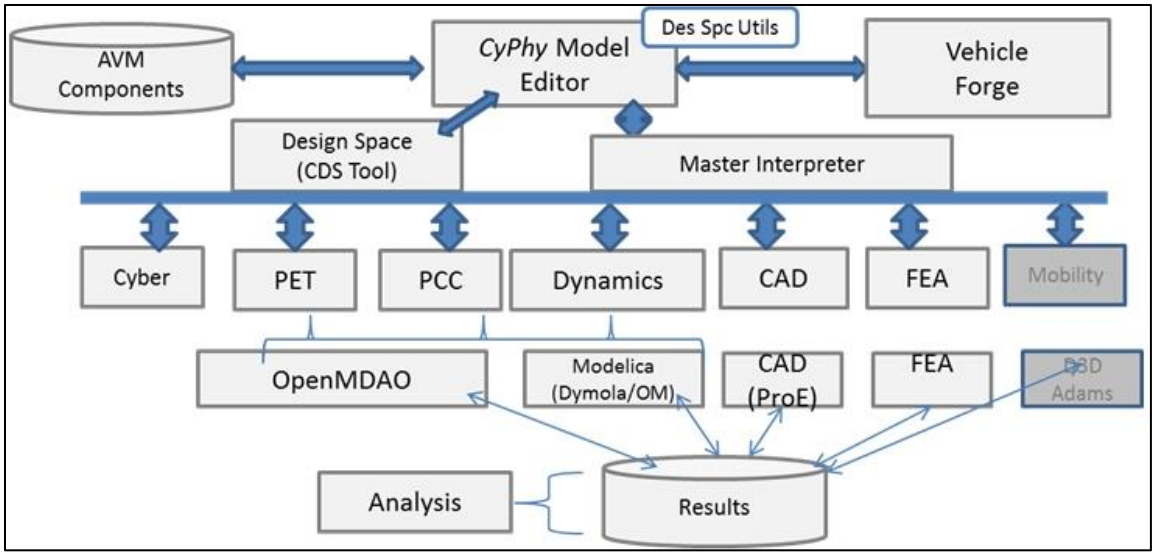

Figure 1: Design Space Tools in OpenMETA

3.0 Design Space Modeling

3.1 Design Space Concepts

In this section we provide definitions of some of the key concepts related to design space exploration and manipulation:

Component: A Component is a self-contained, reusable entity that can be used in a design. A component can be anything from an electrical resistor, a spring, or a transmission. Components are the basic building blocks of a design space.

Component Model: A Component Model captures everything that we know about a component. Component models are self-contained and can include schematic, geometric, physical, and behavioral models of the component. Additionally, component models can provide data sheet and interface information. As the component models are self-contained, they are reusable across all META designs.

Property: A property is a way to capture fixed values of characteristics of a component. Examples of properties could be length of a drive shaft or the maximum output power of a transmission.

Parameter: A parameter is similar to a property except that the value of a parameter can be varied by the designers. Examples of parameters could be length of a spring, or allowable torque range of a transmission.

Component Assembly: A Component Assembly is a fully specified design or part of a design including all the components, their connections with other components in the assembly, and the value of their properties and parameters. Final designs in META are Component Assemblies.

Design Space: The Design Space allows the user to model optionality and composition of multiple components and component assemblies.

Design Element: The key objects that a design space is composed of are Design Elements. Design Element could also be a child/sub Design Space. Other types of Design Elements are components, component assemblies, and design containers.

Design Container: Design Container represents a partially specified part of the design or sub-system and contains a set of components and constraints. There are three types of design containers: Compound, Alternative, or Optional. All elements of a compound design container must be part of the final system design. Alternative design containers are used to capture design choices/trade-offs. The final assembly will include only one of the choices from alternative design containers. Optional design containers are similar to alternative, but also allow ‘none’ as an option.

Design Constraint: Design Constraints are used to specify requirements of the design that are not different from hierarchical composition of components. Examples of design constraints could be the minimum power requirement of an engine, or maximum speed of a vehicle. Also, these constraints can specify compatibility restrictions such as a component cannot be chosen if another component is part of the design. We discuss different types of design constraints supported in META in the later section.

Seed Design: Seed designs are used to integrate a design flow in META that starts with an existing system design. This is a highly practical approach as industry typically has a design that

is currently in use and needs to start from it instead from scratch and work toward improving it or make it amenable to changing design requirements.

Design Flow: A design flow refers to ways in which designers proceed with designing systems. As mentioned above, one of the design flows could be to start from a seed design. Other example would be to design in a hierarchical top-down method that start from designing the system as a whole first and then populating it with sub-systems and so on.

Design Space Refactoring/Manipulation: This refers to modifying a Design Element in a Design Space. The tools in META allow modifying design elements in various ways and provide refactoring choices based on contextual information of the type and location of the Design Element.

Design Space Refinement: This refers to the process of selecting a set of well-tested and verified designs and creating a new design space from these designs. This is used by engineers to finalize parts of the designs as well as to refine/explore them further. The process helps designers to reduce design space complexity and effectively manage design spaces.

3.2 Design Space Modeling Language

The CyPyML uses Model-Integrated Computing (MIC) [3] techniques to support design-time integration of vast number of system-level design aspects and methods, and the automated exploration and manipulation of design spaces. Model-Integrated Computing (MIC) is the core technology on which CyPhyML and its tools are built. MIC focuses on the formal representation, composition, analysis, and manipulation of models during the design process. It places models in the center of the entire life-cycle of systems, including specification, design, development, verification, integration, and maintenance. The Generic Modeling Environment (GME) is a meta-programmable toolkit that enables definition and use of Domain-Specific Modeling Languages (DSMLs) [3] such as CyPhyML. In MIC, DSMLs are configured through metamodels, expressed as UML class diagrams, specifying the modeling paradigm of the application domain. Metamodels characterize the abstract syntax of the DSML, defining which objects (i.e. boxes, connections, and attributes) are permissible in the language. Simplistically, DSML is a schema or data model for all possible models that can be expressed by a language. A DSML for finite state machines would consist of states, and transitions, from which any valid state machine can be realized. The inherent flexibility and extensibility of GME via metamodels make it an ideal platform for CPS design and analysis using CyPhyML.

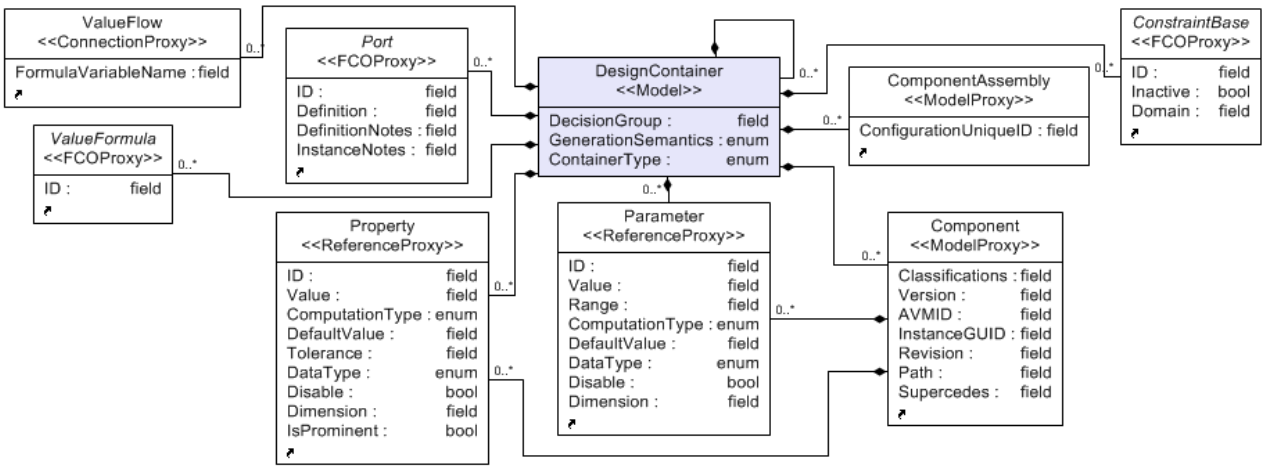

Figure 2: Simplified Metamodel of the Design Space

CyPhyML captures the concepts of design models in various CPS domains, specifies how these concepts are organized and related, and specifies the rules governing their composition. OpenMETA consists of a number of model interpreters and analysis tools, which can be used to generate system and analysis artifacts from system designs, and perform various structural and dynamic analyses.

Any DSML requires precise specification of the language's syntax and semantics. Figure 2 provides a simplified view of the design space part of the CyPhyML metamodel. As shown, the central modeling element in the language is called a *DesignContainer*. The key attribute of a design container is *ContainerType*, which can have one of the following three values: *Compound*, *Alternative*, or *Optional*. All elements of a compound design container must be part of the final system design. Alternative design containers are used to capture design choices/trade-offs. The final assembly will include only one of the choices from alternative design containers. Optional design containers are similar to alternative, but also allow 'none' as an option.

A key element of this language is that design containers can contain child design containers. This enables construction of a hierarchical AND/OR design space. The concrete elements of the design are *Component* and *ComponentAssembly* (CA). A CA is a system that can only contain components and child CAs (i.e. subsystems), and represents a system that has been fully explored, analyzed, and finalized.

As can be seen in Figure 2 components, component assemblies, and design containers have special elements called *Property* and *Parameter*. A property represents a static property of a component or a component assembly. Properties cannot directly be changed at design time during design space model construction. Examples of properties are engine's power rating, or a driveshaft's mass. A property of a design container may correspond to a basic property at that level in the design space or it may be a representative property that is calculated based on the chosen sub-elements of the design container. Alternatively, CyPhyML parameters can be used to specify a range of acceptable values. A key element of our tools is that parameters are

automatically translated into design constraints to ensure that the values of the parameters generated for selected configurations lie within the ranges specified.

CyPhyML also supports combining the properties and parameters using *ValueFormula*. The *ValueFlow* connections are used to connect properties and parameters to value formulae. A simple example could be to calculate mass of the system by adding the masses of its sub-components. CyPhyML supports two different kinds of value formulae: *SimpleFormula* and *CustomFormula* (not shown in the figure). A *SimpleFormula* is used for basic arithmetic operations on incoming *ValueFlow* properties, while a *CustomFormula* is used in situations when a derived property needs to be calculated using complex operations, e.g., *Cosine()* and *Sqrt()*.

For the specification of high-level as well as fine-tuned system requirements, CyPhyML also provides a large number of constraint types to support design constraints arising from component interactions and due to functional and practical system requirements. We provide details of the supported constraint types in the next section.

3.3 Design Space Constraint Types

Design Constraints are key elements of Design Spaces. These constraints specify functional and practical requirements of the design. These could include performance requirements, parameters, and even constraints that are qualified based on certain conditions. The constraint types supported by Design Space Tools are presented below. These include Contextual Non-linear OCL constraints, Visual constraints, Parameter constraints, DecisionGroup constraints, Property constraints, and Conditional Property constraints.

3.3.1 Contextual Non-linear OCL Constraints

Contextual Non-linear constraints are written textually in OCL format and are associated with the container it contains in the context with which it must be satisfied. Figure 3 depicts an example of the Context constraint. This constraint is basically specified to ensure that the IFV drivetrain is capable enough to accelerate on a 20-degree uphill at an acceleration of 2 m/s^2.

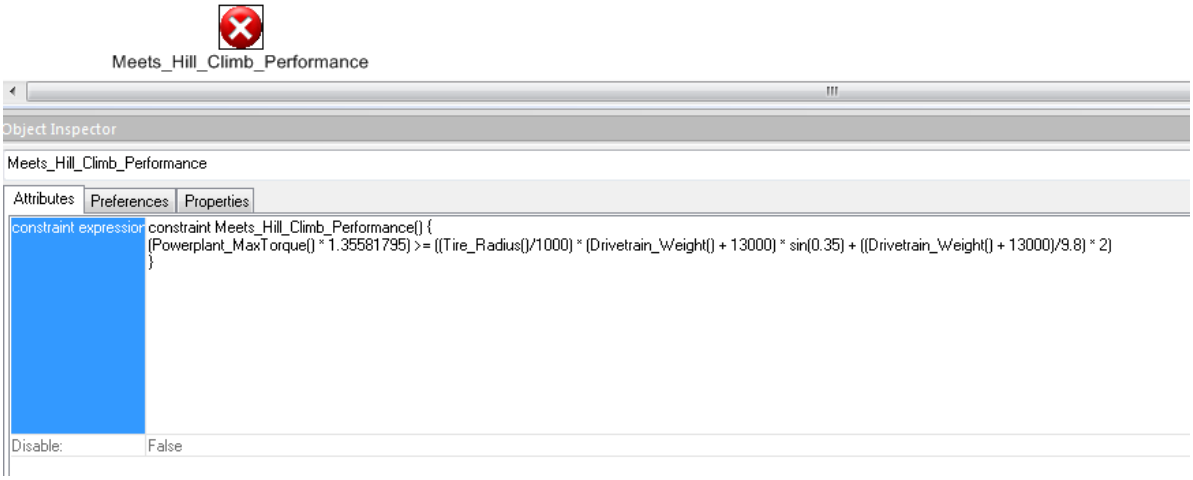

Figure 3: Context constraint to ensure IFV meets hill climb performance

Additionally, the language supports the use of trigonometric functions. The Table 1 below lists all the mathematical functions that are supported for the OCL constraints.

Table 1: Mathematical functions supported for specifying design constraints

S. No.	Mathematical Function	Description
1.	*sin*	SINE of a given number in radians
2.	*cos*	COSINE of a given number in radians
3.	*tan*	TANGENT of a given number in radians
4.	*asin*	INVERSE SINE of a given number in radians
5.	*acos*	INVERSE COSINE of a given number in radians
6.	*atan*	INVERSE TANGENT of a given number in radians
7.	*sinh*	HYPERBOLIC SINE of a given number in radians
8.	*cosh*	HYPERBOLIC COSINE of a given number in radians
9.	*tanh*	HYPERBOLIC TANGENT of a given number in radians
10.	*asinh*	INVERST HYPERBOLIC SINE of a given number in radians
11.	*acosh*	INVERSE HYPERBOLIC COSINE of a given number in radians
12.	*atanh*	INVERSE HYPERBOLIC TANGENT of a given number in radians
13.	*log2*	LOGARITHM TO THE BASE 2 of a given number
14.	*log10*	LOGARITHM TO THE BASE 10 of a given number
15.	*ln*	NATURAL LOGARITHM (i.e., to the base ‘e’) of a given number
16.	*exp*	EXPONENTIAL FUNCTION (i.e., e^x)
17.	*sqrt*	SQUARE ROOT of a given number
18.	*sign*	SIGNUM FUNCTION (returns -1 for –ve, 0 for 0, and +1 for +ve number)
19.	*rint*	MATHEMATICAL ROUND FUNCTION returning double value
20.	*abs*	ABSOLUTE VALUE FUNCTION

3.3.2 Visual Constraints

Visual constraints are helpful in specifying compatibility constraints. It allows the designer to group Design Element references in *AND* & *OR* groups connecting them with implied relationships. As an example, Figure 4 below shows how Visual constraints can be used to specify that Non-C7 Engines are only compatible with larger transmission CX31.

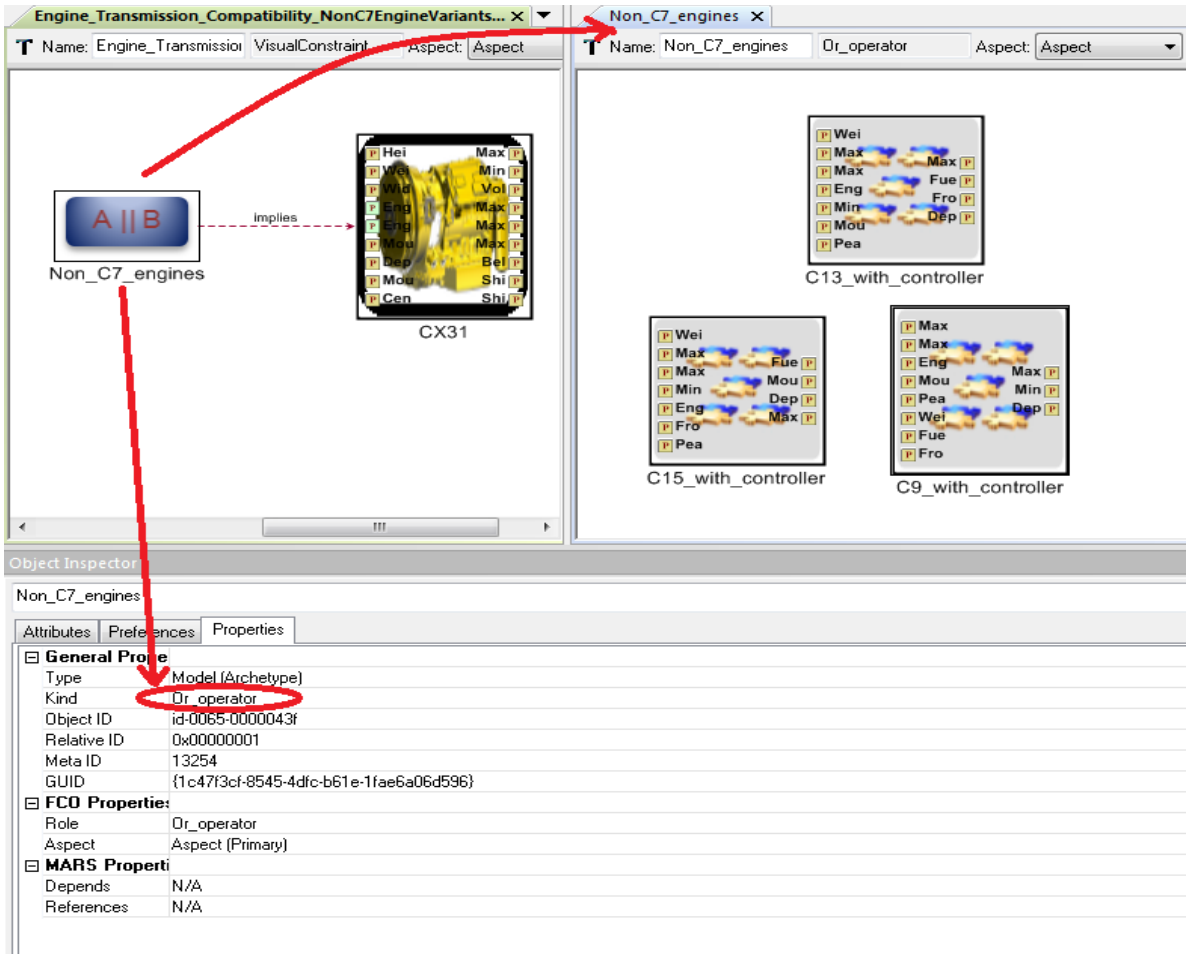

Figure 4: Visual constraint specifying an engine-transmission compatibility requirement

3.3.3 Parameter Constraints

Parameter constraints are constraints that get automatically generated at runtime depending on the parametric ranges specified by the user for values of certain parameters of design elements. For example, the Figure 5 below shows that the *sprint_constant* parameter was specified by the user to have a value between 0.75 and 45. The default value was given as 45. This leads to the automatic creation of a parameter constraint that forces the value of the parameter *spring_constant* to satisfy the valid parametric range provided by the user.

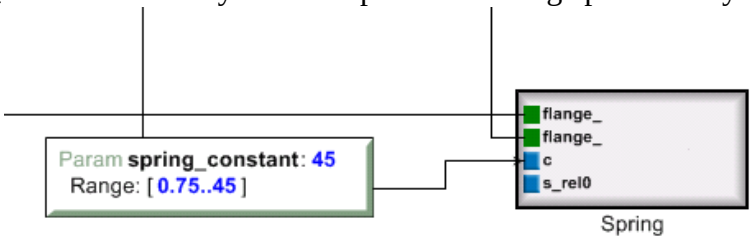

Figure 5: Parameter range specification leading to generation of a parameter constraint

3.3.4 DecisionGroup constraints

DecisionGroup constraints are a good utility available for the users to be able to succinctly specify a set of compatibility constraints for selection of design elements. For example, if the system design contains the use of 4 tires and each tire has a choice from 3 different types, the user can specify a set of Visual constraints that make sure that if a particular tire type is chosen for one of the 4 tire choices in a configuration, then the other 3 tires must also be of the same type. However, this is not sufficient because it must be specified for all available tire types. Moreover, the user also needs to specify these constraints in reverse directions (i.e., both A → B and B → A). As such, it quickly becomes an arduous task of specifying several visual constraints manually. To ease this, DesignSpaceHelper provides an easy way to specify such constraints where choice (decision) is made for one of the alternative design container must also be chosen for all other design containers present in a DecisionGroup parent model. For example, Figure 6 below specifies that same tire type (viz. A, B, or C) is chosen for all 4 tire choices (viz. TireFrontLeft, TireFrontRight, TireRearLeft, TireRearRight).

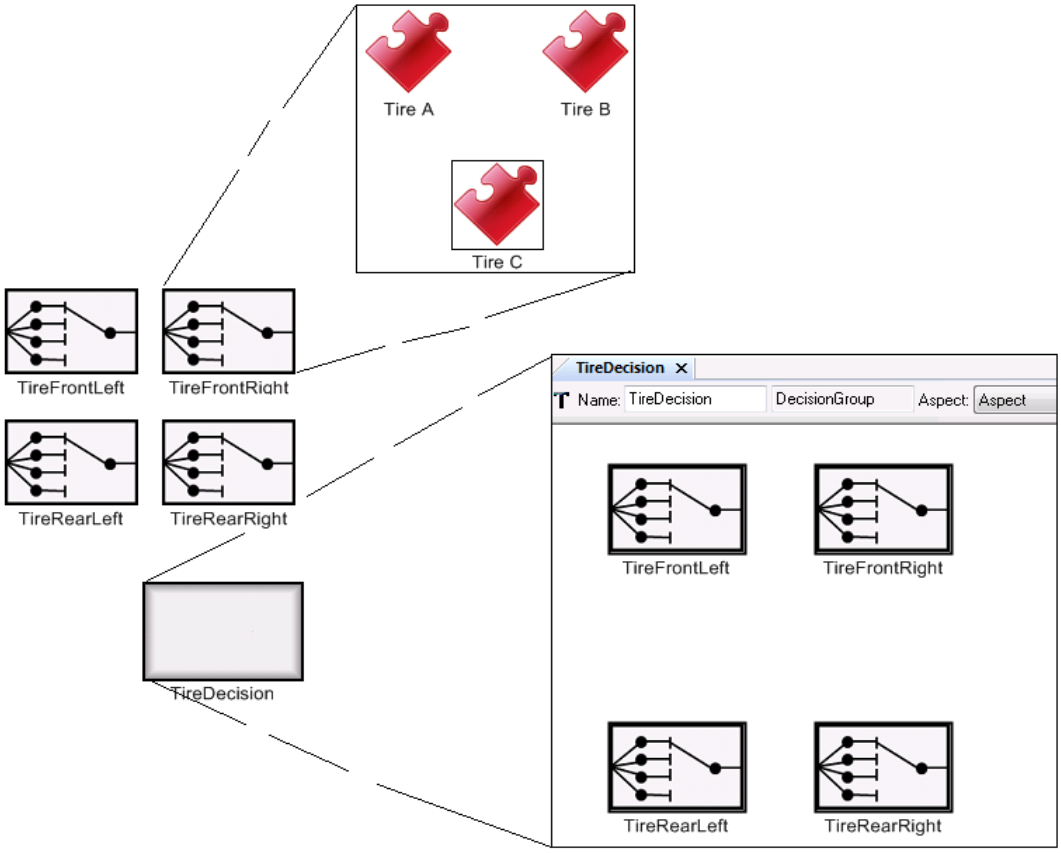

Figure 6: Decision Group constraint to specify a set of compatibility constraints

3.3.5 Property Constraints

Property constraints are recent additions in the CyPhy language. Given a target value for a property, a property constraint allows a user to specify if the value for that property in any of the

generated configurations should be less than, less than or equal to, equal to, greater than or equal to, or greater than the target value. Figure 7 below shows an example property constraint.

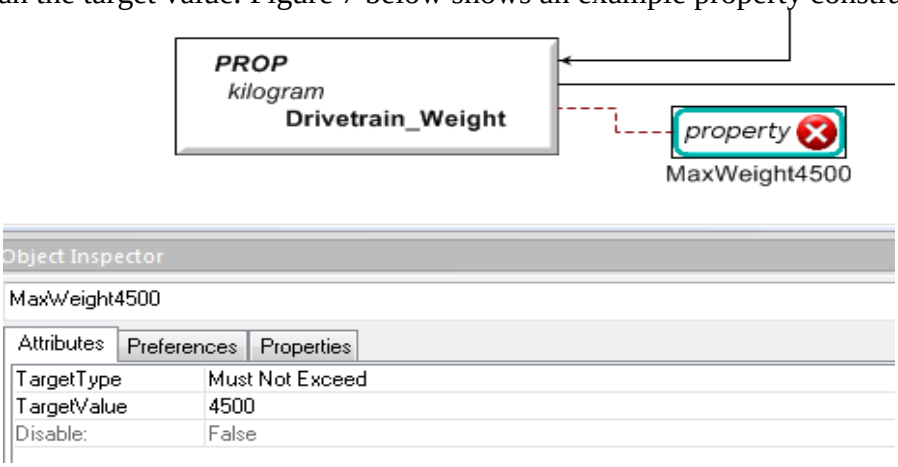

Figure 7: Property constraint restricting total drivetrain weight

3.3.6 Conditional Property Constraints

Conditional Property constraints are similar to Property constraints but are conditioned on a given condition. This condition is specified by a Visual constraint using the AND/OR operators and Implies connections as shown in Figure 4. The condition may represent existence of a group of components or any arbitrarily complex constraint that must hold for the Conditional Property constraint to be valid. Once the Property constraint and Visual constraint for the condition are modeled, the two can be tied by adding a reference of the Property constraint inside the Visual constraint. This transforms the Property constraint into a Conditional Property constraint as shown in Figure 8 below.

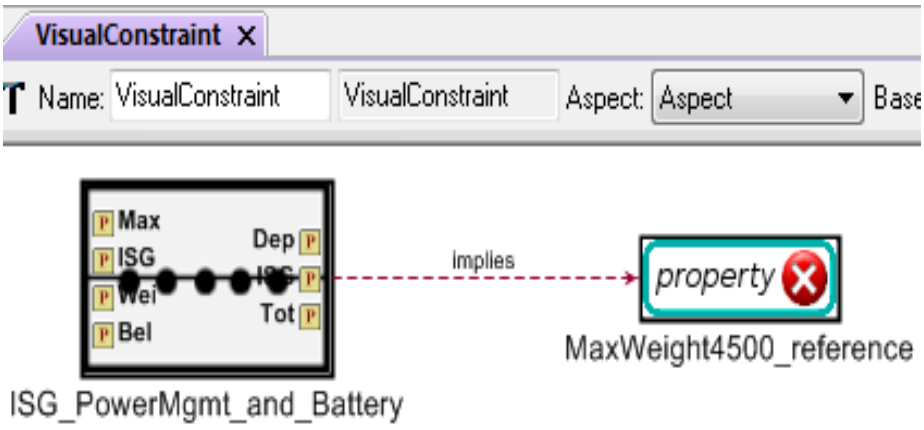

Figure 8: Conditional property constraint

4.0 Overview of Design Space Tools

OpenMETA provides a collection of modeling methods and tools for exploration and visualization of designs and design spaces, solving complex design constraints, and effective management of the design spaces and designs. A brief report of these tools is given below. Examples of tool usage and analyses are given later sections.

4.1 Design Space Exploration Tool (DESERT)

The Design Space Exploration Tool is our key tool for design space exploration. It uses symbolic constraint satisfaction for design space exploration using Ordered Binary Decision Diagrams (OBDD-s) [5]. The choices in the design space come from the AND-OR-LEAF tree structure as well as from the variability of values that can be bound to properties and parameters. The design space is a cross product of all possible choice outcomes. The process begins with a binary encoding of the design space, including the AND-OR-LEAF tree and the design constraints. Each node in the design space tree is assigned a unique integer identifier (ID). These IDs are then translated into BDD variables such that the encoding reflects the design container containment semantics. The properties and constraints are also handled symbolically as BDDs [6]. For handling variable properties, we extended BDDs to Multi-Terminal BDDs (MTBDDs), which enables values other than 0 and 1 as terminals of BDDs [6]. With this encoding, the constraint satisfaction amounts to the composition of design constraints and the symbolic design space representation. The resultant BDD represents the pruned design space. The symbolic representation has been proven to handle very large design spaces consisting of up to 10^{80} design configurations [6].

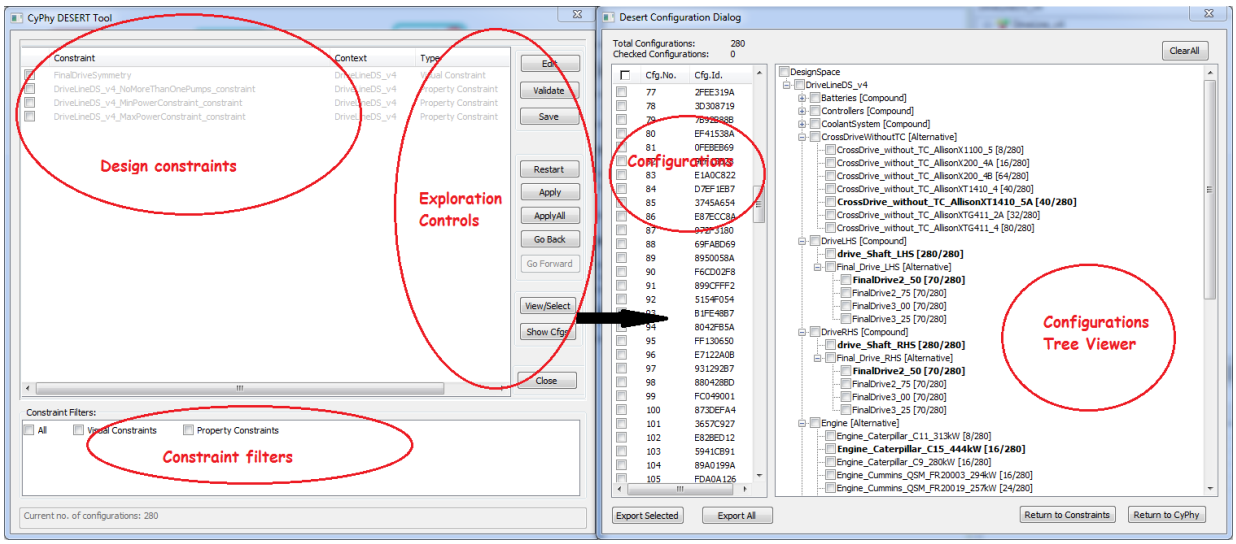

Figure 9: Design Space Exploration Tool (DESERT)

The key elements of DESERT are shown in Figure 9. The exploration controls allow the users to manage constraints and enable them to selectively apply them to explore design spaces. The number of viable constraints may reduce or increase as new constraints are applied or

reverted respectively. It also permits grouping constraints using their types and domains for selective constraint application. Further, design elements can also be selected to be included in all configurations.

The right-hand side of Figure 9 shows the configurations in a tree view. The left panel lists the configurations and the right panel shows the corresponding design space tree with selection frequency for each design element. Users can select configurations and corresponding design elements are highlighted. Users can also select a particular alternative element and corresponding configurations on the left become checked. The selected configurations can be exported back to design space.

As shown in Figure 9, the main panel of the Design Space Exploration Tool shows the list of constraints defined in design space model. Here are brief descriptions of the key buttons on the tool:

- ***Edit:*** Edit the expression of the checked constraint. Valid function list is provided.
- ***Validate:*** Check the validity of all the constraints. If there is any error, the error information will be shown.
- ***Save:*** Save all the changes of the constraints back into model.
- ***Restart:*** Goes to initial state when no constraints were applied.
- ***Apply:*** Apply the checked constraint(s) to design space.
- ***ApplyAll:*** Apply all constraints to design space.
- ***View/Select:*** This can be used to down-select certain components. This is useful when we know that we must certain components for some of the choices in the design space. This can greatly reduce number of configurations that needs to be evaluated.
- ***Show Cfgs:*** The generated configurations are displayed.
- ***Close:*** Exit the DesignSpaceHelper tool.

The constraints can be applied incrementally. The "Go back" and "Go forward" buttons can be used to navigate backward or forward between design space results from applied constraints.

Moreover, the bottom panel shows various constraint-filters available for the constraints listed in the top panel. In the models, users can specify a constraint domain for all constraints. Additionally, the constraints have their types as one of the domains already listed. As such, a constraint can have more than one domain. Using these domains, the constraints are grouped and the filters are provided so that user can choose to selectively apply constraints belonging to the domains of interest.

Furthermore, in the right side of Figure 9, the dialog shows the list of generated configurations by applying the constraints from previous step. The right panel shows the hierarchical (tree) view of the design space. A user can click on the configuration and see all selected components highlighted. For example, in Figure 9, the configuration 1 is selected. As soon as this configuration is selected, the components that form this configuration in the design space tree are highlighted in bold face. This dialog also allows users to select key options that they want in all of the exported configurations. Checking appropriate checkboxes in the right hand side design tree completes this step. When options are selected, the appropriate configurations are automatically selected in the left side panel. The default behavior is to select only those configurations (intersection) that include all of the chosen options in the design tree.

However, when multiple design options of a single Alternative container are chosen, all configurations that include any of the selected options are selected (union). The dialog also contains several command buttons, which work as follows:

- ***Export Selected:*** Export the selected configuration(s).
- ***Export All:*** Exports all configurations.
- ***Return to Constraints:*** Goes back to the prior constraint dialog (shown on the left side in Figure 9).
- ***Return to CyPhy:*** Closes all dialogs and goes back to GME.

The selected configurations are exported as *Configurations With Constraints* (CWC) models and placed in a folder of type *Configurations* in the top-level design space for which they were generated. A CWC model stores the references of the selected components in the configuration and references to the constraints that were applied to arrive at these configurations. Additionally, it contains a reference to the Design Space for which the configurations were generated. These are useful aids in traceability of the generated configuration.

It is important to know that a CWC model contains the references to design elements of the design space without any connections or hierarchy. This is because, for any realistic design space, there are thousands of possible configurations at the static analysis stage. Using fully elaborated models can quickly deplete all available system memory. Another tool, described in the next section, called Design Space Component Assembly Exporter (CAExporter), is used to fully elaborate chosen configurations (CWC models). A fully elaborated ComponentAssembly will include all of the connections between components and subsystems and preserves the hierarchy of the design space.

4.2 Design Space Component Assembly Exporter (CAExporter)

The DESERT process exports "configurations", each one representing a set of decisions encapsulating a feasible design point. As mentioned in previous section, these configurations contain references to the design elements of the design space without any connections or assembly hierarchy. The reason for this is that a large design space may contain hundreds, even thousands, of design configurations that need to be further investigated to analyze their usefulness. Elaborating all of these design configurations in a fully-specified form (with connections and hierarchy) can quickly consume all available system memory. Hence, we have created a separate tool to convert the design configurations generated by DESERT to their fully-specified form. We can also use this tool to convert design configurations one-by-one, run through system analysis, and keep/discard them depending upon their usefulness. This is useful way to deal with system memory constraints. This is also the way the Master Interpreter (MI), as described in later sections, uses the design space tools in an automated manner.

The tool can be used to convert a single *Configuration With Constraints* (CWC) model, or a group of CWC configuration models, or even all CWC models contained in a *Configurations* folder depending what is selected prior to invoking the tool. The Figure 10 below shows a group of exported CWC models for a design space.

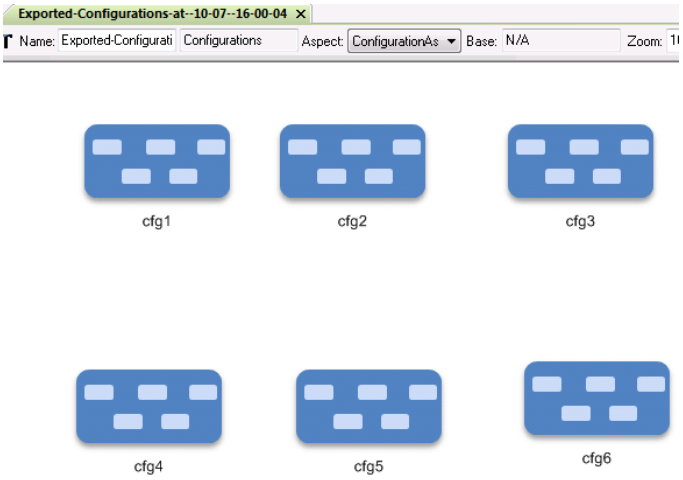

Figure 10: Sample exported design configurations

Once a configuration (or a group of configurations) is selected and CAExporter is invoked, it creates a fully-specified assembly model for the chosen configuration. This fully-specified component assembly includes the full hierarchy of the models as well as all appropriate connections between components and sub-systems. At the same time, the tool also creates references of the generated component assembly models and places them next to the CWC models that were used for generating them and connects them so that the user can easily navigate to the corresponding generated component assembly from the configuration for which it was generated. As an example, the Figure 11 below shows the updated *Configurations* folder that now includes references to the corresponding generated component assemblies.

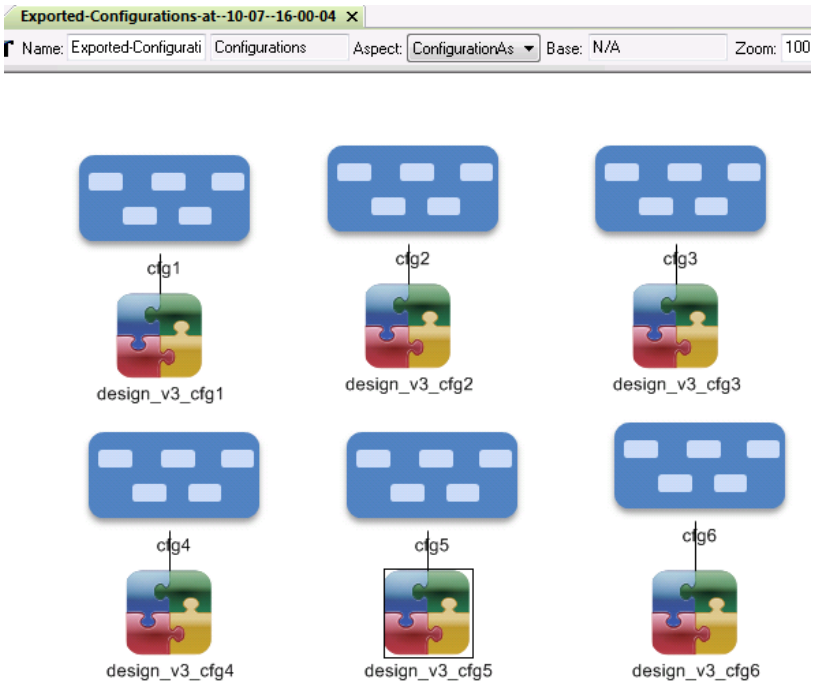

Figure 11: Design configurations converted to component assemblies

These exported designs represent the pruned sub-set of all possible designs that satisfies the design-time constraints. User can use other types of analysis tools provided in OpenMETA to verify suitability of the designs and further prune/rank-order these configurations.

4.3 Design Space Manipulation Tool (DSRefactorer)

When a design space is being constructed, a user often needs to change design space elements. For example, when multiple vendors of a component become available, the static component in the design space could be replaced with an *Alternative* design container with multiple choices for the component. Another example could be when a sub-system has been explored and finalized; it could be replaced with a component assembly. Similarly, there are a number of use-cases where the user needs to manipulate the design space elements in order to update the design space. Manual manipulation is doable, but requires a lot of unnecessary tasks like recreating ports in parent/child design elements, redrawing connections, etc. Furthermore, this is also susceptible to errors when done manually. The Design Space Manipulation Tool (DSRefactorer) helps avoiding these errors and remarkably increasing design efficiency by automatic all of these tasks.

Additionally, after the initial Design Space Exploration (DSE) has been completed, the generated configurations are subjected to dynamic analyses for evaluation against the secondary requirements. The result of these detailed system analyses in terms of valid design selections and reformulations must be incorporated into the original design space, which must be re-explored to generate a new set of valid design configurations. This iterative nature of the design process with strong bidirectional coupling between design activities and system analysis and verification is a key requirement for CPS design. This necessitates re-working the original design space in

various ways, where the Design Space Manipulation tools becomes highly useful in automating the various design space manipulation tasks as described below.

The main application of this tool is for converting existing components, component assemblies, or design containers into a new design container, or component assembly that can now include the new parts in it. In this process, a user first selects a set of components, component assemblies, or design containers and then invokes DSRefactorer. The tool will create new design elements depending on what was selected for refactoring. If more than one option is available, then the user is presented with a dialog to make the choice and then appropriate refactoring will be carried out according to user's choice. The tool will perform the refactoring while preserving (or creating where needed) all connections and ports. It also maintains the overall hierarchical structure of the design space while manipulating its constituent parts according to user choices. Moreover, the tool also intelligently utilizes contextual information of where the manipulated design space element resides within the design space and what type it is to determine the set of refactoring actions that are applicable.

Below is a summary of what options are applied or choices are presented to the user depending on the context of the design elements chosen for which the refactoring was applied. For brevity the Table 2 below uses the following acronyms: Component (CO), Component Assembly (CA), Design Element (DE), Design Space (DS), Design Container of Compound type (DCC), Design Container of Alternative type (DCA), Design Container of Optional type (DCO), Component Reference (COR), and Root Folder (RF).

Table 2: Use-cases of Design Space Manipulation

Use-case Name (*Abbreviation of the refactoring use-case*)	**Context** (*where the refactoring was invoked from, i.e. inside which design element*)	**Selection** (*what was the refactoring invoked on, i.e. the selected design elements before invoking the refactorer*)	**Action** (*how the selected elements were refactored*)
InCA.0	CA	None	No dialog shown; New DS created (as a DCC) under RF
InCA.1.CA	CA	Single CA	Dialog shown with choices (a) Extract elements of CA (b) Convert CA to a new CA (c) Convert CA to a new DCC
InCA.1.COR	CA	Single COR	Dialog shown with choices (a) Convert CO to a new CA (b) Convert CO to a new DCC

InCA.>1 .COR	CA	Combination of CAs and CORs	No dialog shown; New child CA inside current CA (parent) created; Child CA contains all selected CAs and/or CORs; Additional ports created for connection to objects in parent CA
InDC.0	DC	None	No action; Usage information shown
InDC.1. CA	DC	Single CA	Same as InCA.1.CA
InDC.1. COR	DC	Single COR	Same as InCA.1.COR
InDC.1. DCC.0.DC	DC	Single DCC (not containing any DC)	Dialog shown with choices (a) Extract elements of selected DC (b) Convert selected DC to a new DCA (c) Convert selected DC to a new CA
InDC.1. DCC.>1.DC	DC	Single DCC (that also contains a DC)	Dialog shown with choices (a) Extract elements of selected DC (b) Convert selected to a new DCA
InDC.1. DCA	DC	Single DCA	No dialog shown; New DCA created with selected DCA placed inside as child
InDC.1. DCO.1.DE	DC	Single DCO containing only 1 DE	Dialog shown with choices (a) Convert DCO into mandatory (i.e. extract its contained DE out), (b) Convert DCO to a DCA
InDC.1. DCO.>1.DE	DC	Single DCO containing > 1 DE	Same as InDC.1.DCA
InDC.1. CA	DC	Single CA	Dialog shown with choices (a) Convert selected CA to a new DCA with selected CA in it (b) Convert selected CA to a new CA with selected CA in it
InDC.1. COR	DC	Single COR	Dialog shown with choices (a) Convert selected COR to a new DCA with selected COR in it (b) Convert selected COR to a new CA with selected CA in it
InDC.>1 .DE.0.CA.0. COR	DC	>1 DE selected, none of which is a DC	No dialog shown; New DCA created with selected DEs in it
InDC.>1 .DE.>0.CA. or.>0.COR	DC	>1 DE selected, one of which is either a CA or a COR	Dialog shown with choices (a) Convert to a new DC (with selected DEs in it), (b) Convert to a new CA (with selected DEs in it)

The Design Space Manipulation Tool is very useful for the iterative design process in OpenMETA. In conjunction with the Design Space Refinement Tool (described in the next section), it enables users to perform continuous design and analysis in an integrated and efficient manner allowing users to complete the feedback loop. This helps with evolving the design space as the system requirements, resources change, and as more analysis results become available.

4.4 Design Space Refinement Tool (DSRefiner)

One of the critical requirements of Design Space exploration and configuration generation is the capability to perform coarse-grained exploration and constraint satisfaction on some parts of the design space and when satisfactory configurations have been generated, do deeper refinement of those parts on the selected results. Such a capability is provided by the Design Space Refinement Tool.

If a user endeavors to completely specify the entire design space down to individual nuts and bolts, not only does the design space becomes unmanageable, but the analysis tools of varying capabilities are difficult, cumbersome, and time-consuming to apply. It is even possible that the design space becomes so huge that it allows for generation of billions of configurations some of which only differ in a very small way such as color of the dashboard meter! Even if the user manages to reduce the number of configurations by using an appropriately chosen set of design constraints, using all the analysis tools (for all domains we need to analyze such as CAD, Thermal, Electrical, etc.) at such a detailed level becomes highly arduous and time-consuming.

As such, we need to be able to specify design space at a level of detail that we are comfortable reasoning with. As there are constraints that are applicable at this level itself, it is desirable to make use of them and eliminate huge chunks of design space that are clearly infeasible. User then proceeds with generation of constraints at this coarser-level of design space. It is important to note that it is not necessary to specify all parts of the design space at the coarser level and it all depends on the level of detail for each part of the design space that user is comfortable reasoning with and have analysis tools available. The next step is, of course, to use the generated configurations and run various static and dynamics analysis. Once the analysis is over, a few configurations are down-selected that satisfy all static and dynamics design constraints.

This is the point where the Design Space Refinement Tool can be invoked on a set of selected design configurations (CWC models generated by the Design Space Exploration Tool). The design space refinement tool will take these selected design configurations and convert them into a refined design space that can be reasoned with in the same way as the original design space. Even the generated refined design space looks very much like the original design space. However, the key difference is that any component, component assembly, or design container that is not part of any of the selected design configurations is not included in the refined design space. Secondly, the original static constraints are removed and a new *visual* constraint is added that directly encodes the configurations that were selected for design refinement. The reason for this is that the initial configurations were selected only after a detailed constraint satisfaction and dynamics analysis of the original coarse-level design space and there is no need for re-doing that work.

Note that for all components which are still part of the refined design space, all of their connections, ports, and properties are preserved in the generated refined design space. In fact, if Design Space Exploration Tool is invoked again on the refined design space, the configurations generated are exactly the same as those that were selected for refinement from original design space – with same look and behavior!

The new refined design space is leaner and contains a direct representation of the originally selected design configurations. However, as it is still a design space, the user can freely refine and expand this design space as different parts of the design space are now included or some parts of the design space are further elaborated into greater detail. As described in the previous section, the Design Space Manipulation Tool becomes very useful here to convert existing components, component assemblies, or design containers into a new design container that can now include new parts in it.

Below we provide an example to illustrate the design space refinement process. Figure 12 shows the top-level view of a sample Infantry Fighting Vehicle (IFV) drivetrain design space.

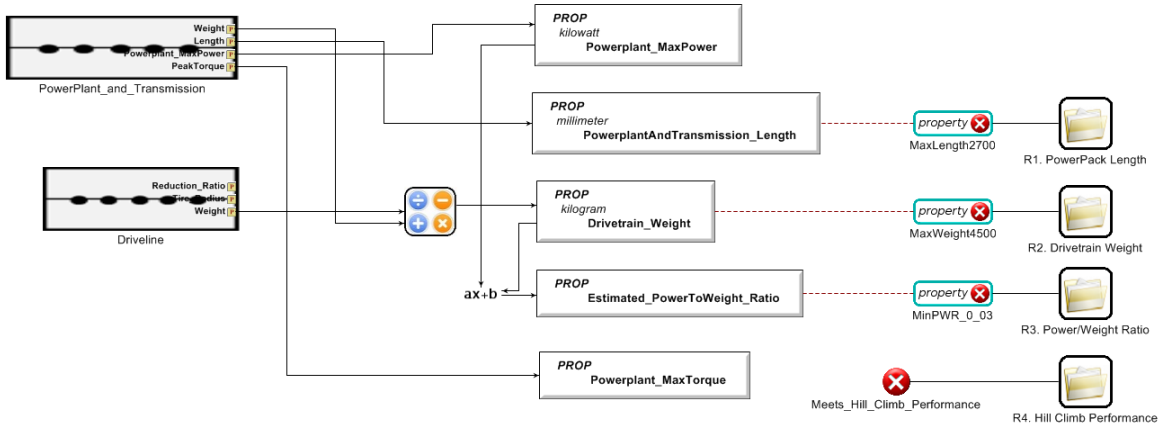

Figure 12: Top-level view of example design space for IFV drivetrain

When Design Space Helper tool is invoked on this design space, the tree viewer of the configurations shows that there are a total five configurations that satisfy all of the design constraints that were specified in the design space. Figure 13 shows the tree viewer of the tool. It can be seen in Figure 13 that both VU_ISG_V2 and VU_ISG_V3 are used in some of these five configurations. Also, both transfer cases are used, viz. 455 and 484.

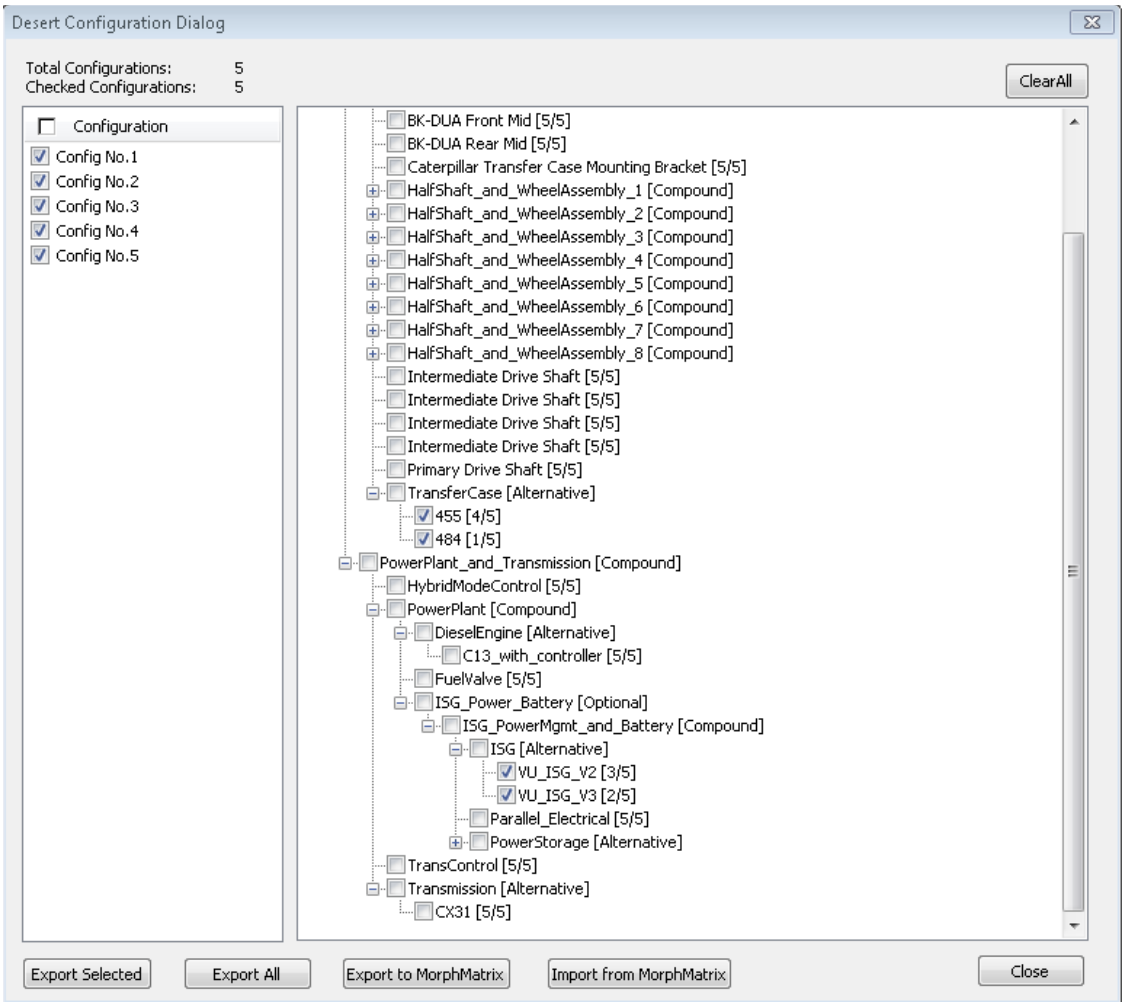

Figure 13: Design configurations for example IFV design space in a tree-viewer

When all of the five configurations are exported, each of them is then elaborated from CWC models (containing only design element references with no connections or hierarchy) to fully-specified component assemblies with hierarchy and connections. Figure 14 shows the five generated configuration models and a fully-specified component assembly corresponding to configuration #3.

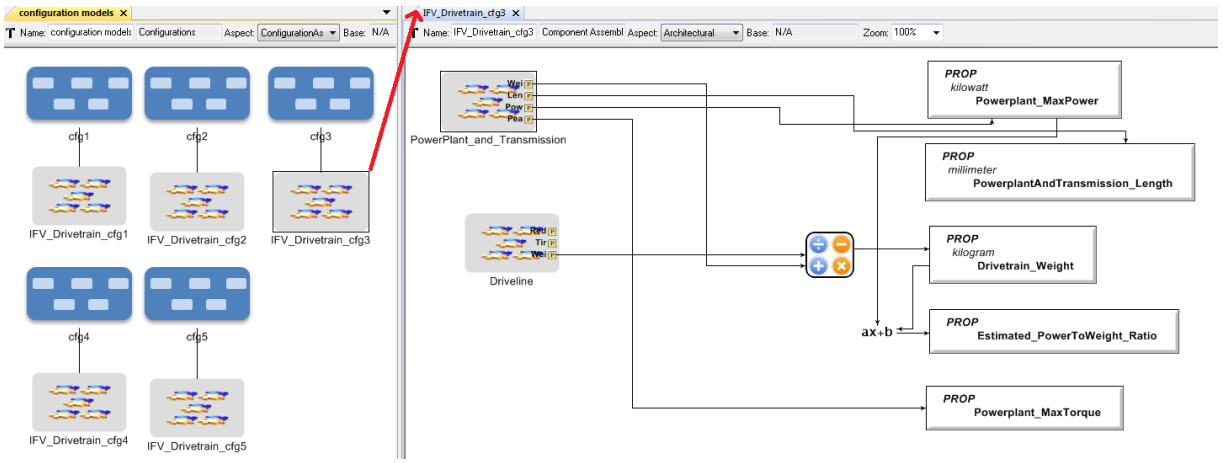

Figure 14: Generated configurations (with component assembly) for example IFV design space

Next the detailed analysis is performed for these fully-specified component assemblies. Let's assume that after analysis, configurations #2 and #3 were selected. Next, we select the cfg2 and cfg3 CWC configuration models in GME and invoke the Design Space Refinement Tool to generate a new refined design space that includes only these two design configurations. Figure 15 shows the top-level view of the refined design space. Notice that this looks exactly like the original design space, except that a new reference to the original design space and a new visual constraint is added to the refined design space.

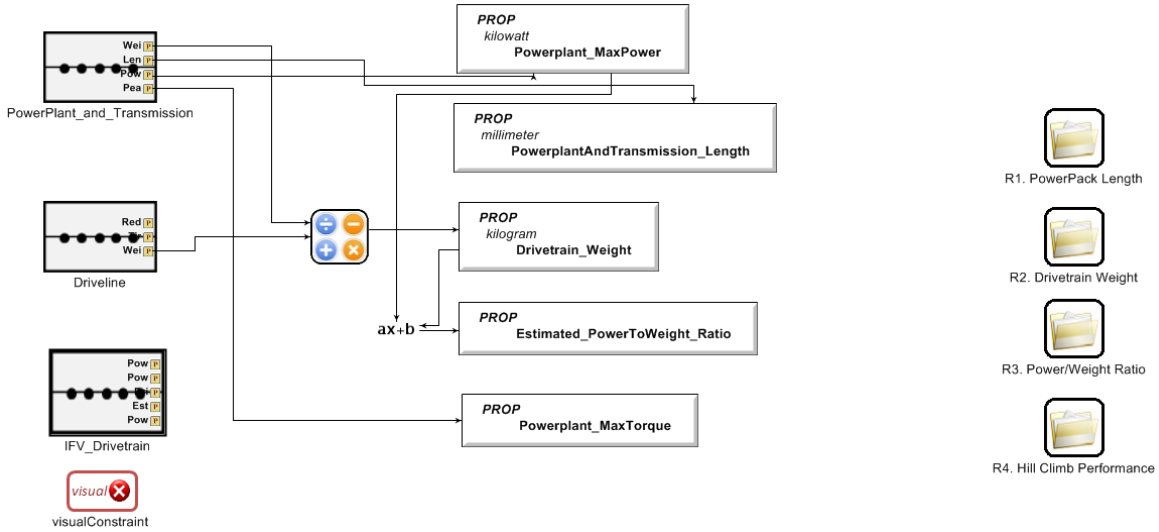

Figure 15: Refined design space for chosen configurations of IFV design space

However, when we look deeper into the ISG design alternatives in the refined design space (see Figure 16), we can see that now it contains only VU_ISG_V3 ISG (and only 455 transfer case) as the constituent component. The detailed path to ISG container can be seen in the title of the GME window in the Figure 16.

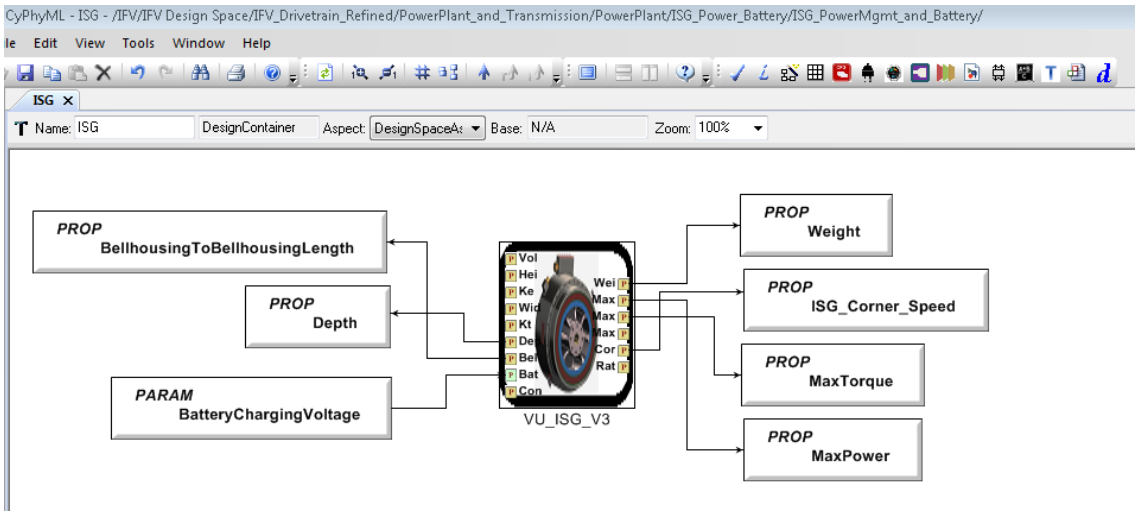

Figure 16: Refined (ISG) design container in refined IFV design space

At this point, user can safely edit the refined design space for further elaboration or refinement as is normally done during design space modeling. Newer design constraints can also be added along with this refinement. Moreover, if more alternatives need to be added at the same level as an existing design element (i.e., a component, a component assembly, or a design container), then the Design Space Manipulation Tool can be used.

In summary, the Design Space Refinement Tool allows the user to select a subset of configurations of a design space and generate a refined design space using these configurations. The refined design space has the exact same hierarchical structure, ports, and connections as are in the original design space, but omits design elements from the original design space that are not part of the selected configurations. Further, original design constraints are removed, but a new *visual* constraint is added that ensures that when Design Space Exploration Tool is run on the refined design space, the exact same set of configurations are generated. This avoids repetition of the analyses that were done in the original design space. Thus, the Design Space Refinement Tool is highly useful for gradually building design spaces, performing coarser-grained analyses, and incorporating the results for refining and manipulating the design space.

4.5 Design Space Criticality Meter (DSCriticalityMeter)

The Design Space Criticality Meter is an important tool for informed use of Design Space Manipulation and Design Space Refinement Tools. During the design space refinement process, user selects a set of configurations based on the outcome of static and dynamic analysis of all configurations for further refinement. Using the Design Space Refinement Tool, user converts the selected configurations into a newly created refined design space. This new refined design space is leaner and a direct representation of the originally selected design configurations with all connections, ports, and properties preserved. The newly created refined design space can be freely refined and expanded for further design space exploration and refinement. Also, the Design Space Manipulation Tool is used to convert existing components, component assemblies, or design containers into a new design container that can now include new parts in it. For both of these tools to be used with greater information, the Design Space Criticality Meter can be used to

determine the usefulness of refining or manipulating a particular component, component assembly, or a design container.

One example of the key criticality metrics that is of immediate help to the designer is the number of configurations a particular component, component assembly, or a design container appears in. Depending on the design space, user may choose to refine a design element that is included in all or some reasonable number of configurations. It is important to note that at this stage only those design elements will appear in the refined design space that are part of at least one of the chosen configurations for refinement. In general, design elements appear only in a subset of all design configurations depending on how many alternatives were designed in the design space. As such, it is important to know how many configurations include a particular design element. This is even more important in vendor selections and reliability analysis. This metric is also a parameter of the overall design complexity of the design space.

The Design Space Complexity Meter currently shows the number of configurations for all components, component assemblies, and design containers. This is shown as an attribute of design elements called *NumAssociatedConfigs*. In the future, more metrics related to the complexity of design elements can also be appended. The criticality metric is illustrated in Figure 17. It shows that ISG-1 wasn't selected in any design configurations, whereas ISG-2 was selected in 3 of 5 design configurations. This example is for original IFV drivetrain design space.

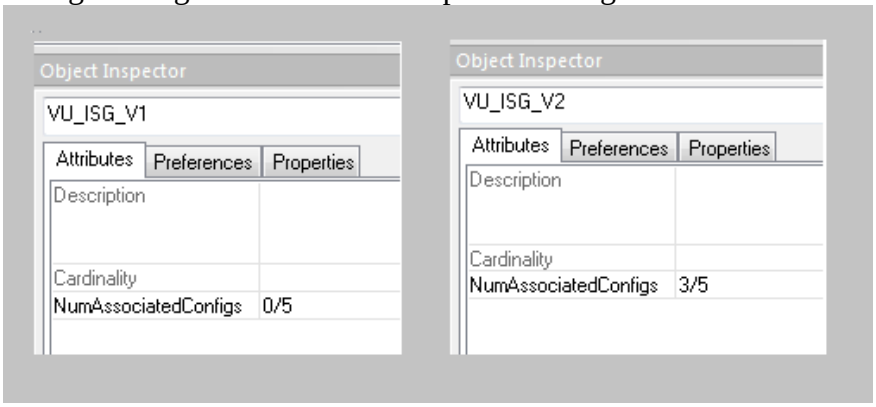

Figure 17: Criticality metrics generated by Design Space Criticality Meter

Also, when invoked, the tool calculates these numbers for all design spaces that are in the GME model. The tool internally runs the Design Space Exploration Tool and applies ALL constraints of design space and updates the *NumAssociatedConfigs* attribute for all of the design elements in the design space. The tool runs in batch mode such that NO user selection dialogs are presented (e.g. to select only a few constraints in a design space).

4.6 Supporting META Tools

Several other interpreter components exist in OpenMETA that are associated with DSE. The *Component Authoring Tool* provides importing capability from various domains (e.g. CAD, Modelica) into OpenMETA tools. After the component library is populated the *Component Library Manager* helps to discover and insert different instances of the same component types into an alternative design container. Once components and subsystems are composed in a design space and design configurations are exported the *Master Interpreter* automates the translation of

all designs into executable domain specific models. After the model transformations, it transfers the generated executable models (analysis packages) to the *Job Manager*, which executes the analyses using domain specific tools, e.g., Dymola, Creo, etc.

5.0Iterative Design Process in META

CPS design is a major integration problem because of their inherent complexity and unexpected component interactions. As shown in [7], the design process must allow for the continuous existence of an executable system, with a concrete architecture, well-defined interfaces, and an executable form. This allows designers to analyze their designs earlier during the design process and obtain useful feedback. This facilitates less error-prone designs, saved manpower, and manageable design spaces.

To enable the iterative design process for CPS-s, OpenMETA supports three key design flows. As shown in Figure 18, the first design flow is a classic top-down design space construction. In this case, the user begins with the top-level design container and adds design elements to it and constraints on those elements according to design requirements. A second design flow involves starting from a single seed design. This design flow is highly applicable for the real-world design use-cases, where there are existing designs and design processes. Starting from the seed design, the user extends the design space (using the tools mentioned above) to add alternatives in place of concrete design elements such as components or component assemblies. In this fashion, the user grows a larger design space from that seed design. Another supported design flow is when the user does not have concrete design elements or assemblies to work with. In this case, the user can use *surrogate equations* in place of design space elements. The design space can still be explored and analyzed. These surrogates can then be replaced with more accurate models as they become available. Surrogates are also helpful for performing coarse-grained analyses, the results of which can be used to refine the design space.

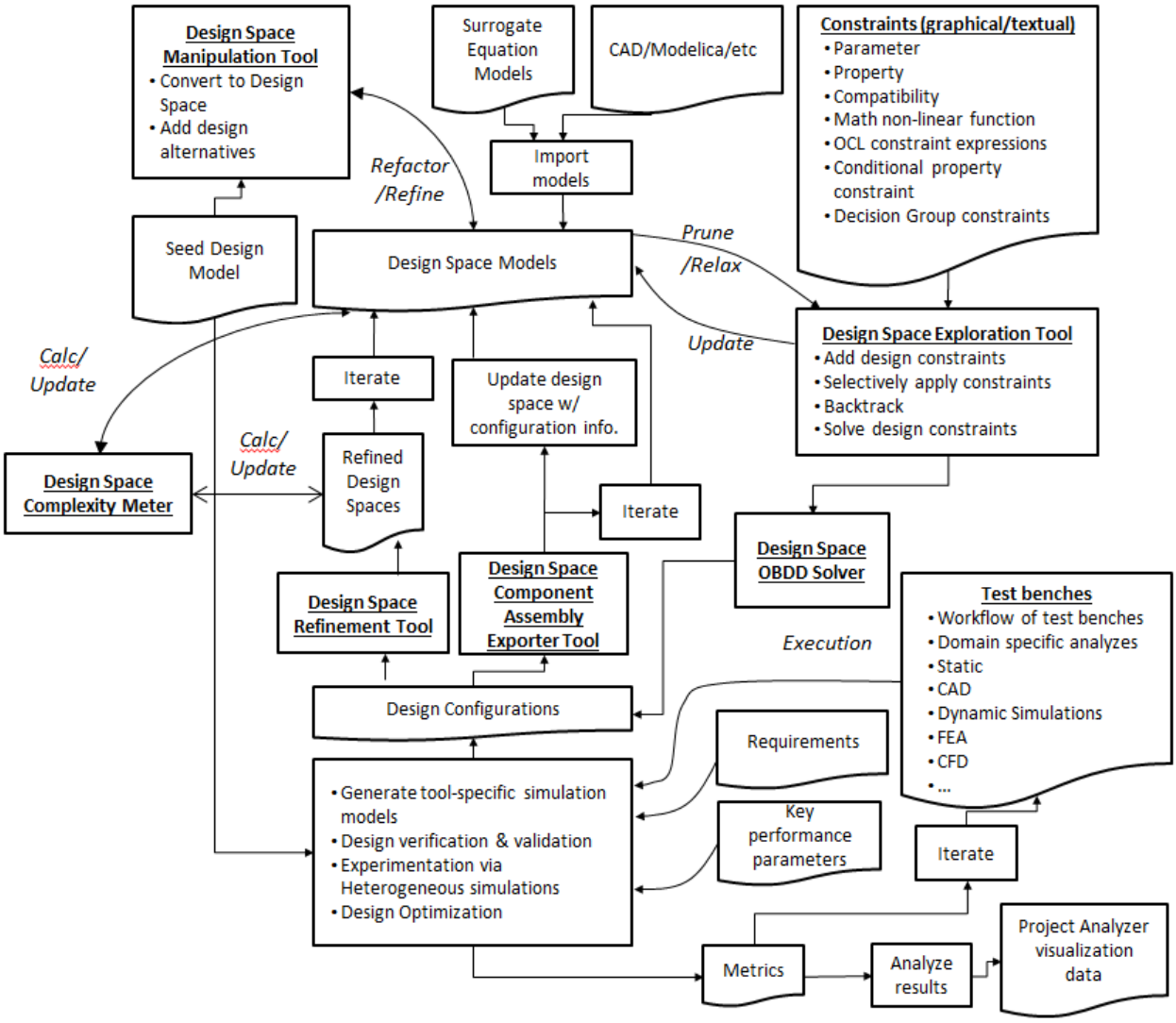

Figure 18: Iterative design process in META

It is important to note that while there are several design flows that users can exercise in OpenMETA, all of the design space tools, such as DSRefactorer and DSRefiner, are equally applicable. Different design flows do not eliminate the need to continually evolve design spaces using a closed-loop integration of design and analysis activities. As shown in Figure 18 and described previously, OpenMETA provides several supporting tools to build, test, and analyze design configurations.

6.0 Case Study

In this section we present a simplified drive line model as a case study for design space exploration. Many CAE tools (e.g., CAD, FEA, CFD, and Modelica) have great capabilities to analyze a design, but creating/extending design in CAE tools often takes significant time, and invariably requires subject matter expertise. To improve the process, a seed design point needs to be altered with alternative component instances to evaluate which combination performs better. We use DSEM tools in OpenMETA to capture architectural alternatives and specify design constraints, generate configurations that satisfy these constraints, and then apply a set of model transformations to generate composed executable models for the CAE tools.

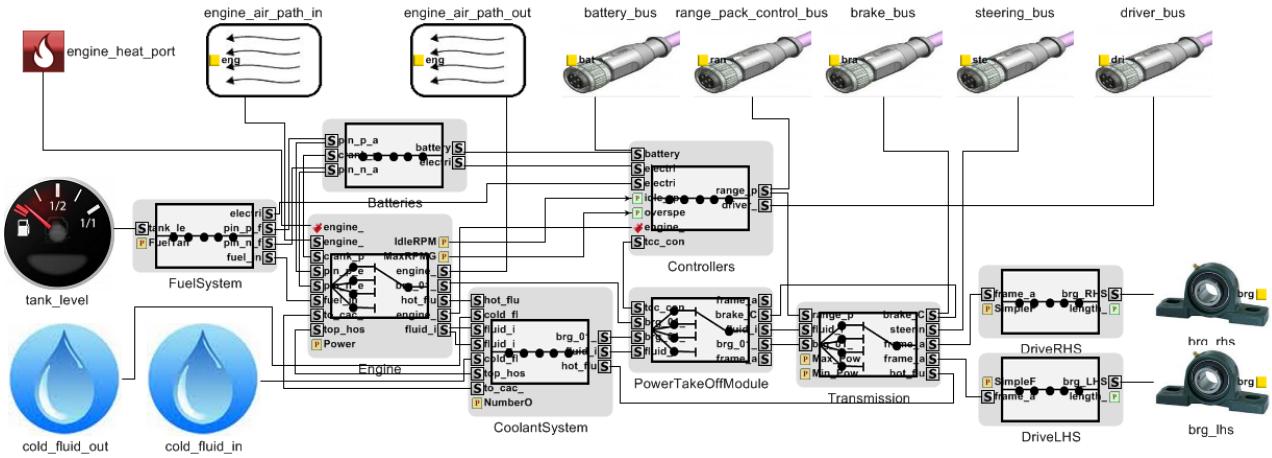

Figure 19: Drive-line design space model

Figure 19 shows the design space of a simplified drive line that contains several subsystems. The architecture and composition of the design space was derived from a single design point built in Modelica. The design space is extended with additional engine, power take-off module, transmission, final drives, hydraulic fan, and hydraulic pump alternatives. This leads to a significantly large design space – 15456 configurations – many of which are not viable due to design constraints. Figure 20 shows 4. *FinalDriveSymmetry* constraint ensures that the left and right drives have the same gear ratios. *MinPower* and *MaxPower* constraints assert that the engine's nominal power lies between the transmission's minimum and maximum power ratings. *NoMoreThanOnePumps* ensures the design uses no more than one hydraulic pump. Application of these constraints reduces the viable number of designs to 47 – a manageable set that users can analyze.

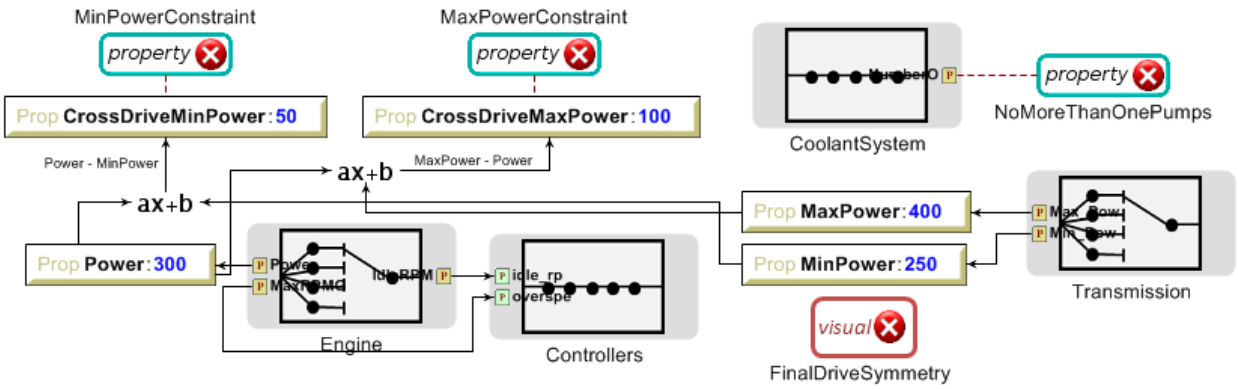

Figure 20: Constraints used in the drive-line design space

Configurations are composed across various domains, but here we briefly illustrate the composed Modelica models, simulation execution, and visualization of results for completeness. Please note that further details on model composition, simulation execution and visualization are presented in other chapters of the META final report.

We analyzed the behavior of the selected 47 configurations using the Modelica simulation tools and collected results are shown in Figure 21. It shows two visualization capabilities of the Project Analyzer (a) parallel axis plot and (b) multi-attribute decision analysis. The parallel axis plot has vertical axis for each variable of interest from the analysis and each colored plot represents a design configuration. The requirement objective and threshold values are shown with green and red colors respectively. The multi-attribute decision analysis widget shows an ordered list of configurations based on the user's specified weighting of each variable of interest. This is an interactive widget that helps to quickly identify differences between designs and choose the best design based on user preferences.

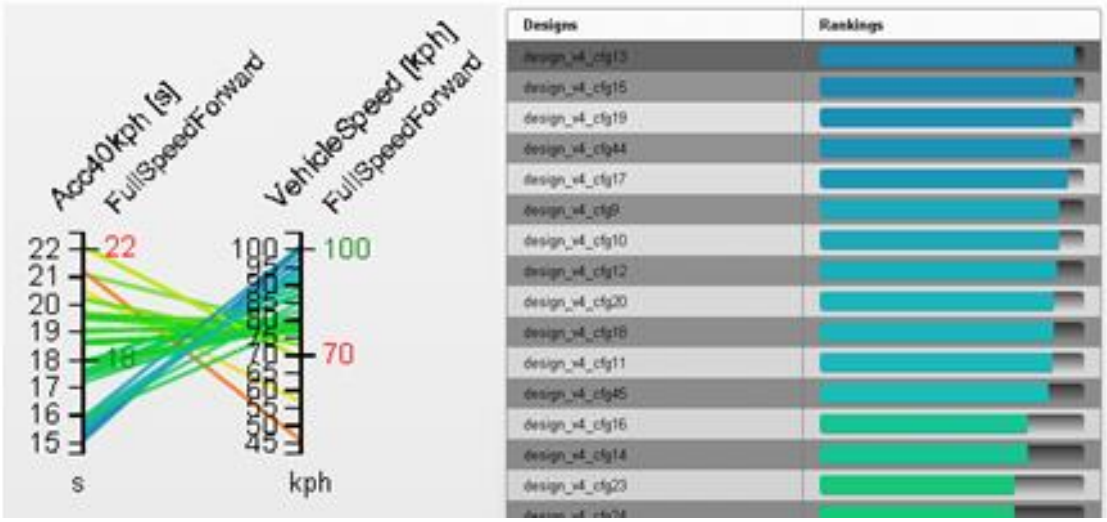

Figure 21: Project Analyzer showing parallel axis plot and multi-attribute decision analysis

7.0 Conclusion

Cyber-physical systems are tremendously hard to design and analyze due to continuous interaction between different domains of the system as well as the use of cyber infrastructure to facilitate information flows between system components. This becomes even more challenging when the designer is faced with ever-changing system requirements and vendor management. Consequently, designs are never fixed in stone and are subject to continual development. However, adapting designs involves a huge penalty in terms of time and resources to make sure that the design meets system constraints as well as satisfies the new requirements for which the system design is being adapted. This is where the strength of OpenMETA tool chain for constraint-driven design space exploration and manipulation is highly useful. The variety of design space tools discussed in the sections above provide for a complete tool-suite to enable the iterative design process of cyber-physical systems. This not only enables highly efficient design process, but lends the designs themselves to be highly analyzable. Furthermore, the additional supporting tools, as discussed above, provide deeper insights into the designs and provide full support for their evaluation, ranking, and visualization of key system metrics.

8.0Future Work

In the future, as the opportunity arises, we plan to further extend our tool chain to increase its capabilities. In particular, we plan on adding:

1. Adding additional design space manipulation tools,
2. Increasing the library of supported design constraints including those that get automatically generated from specifications and ones that facilitate succinct representations of existing methods, and
3. Increasing feedback messaging from analysis tools into design space exploration and manipulation.

Bibliography

1. Sztipanovits J.: Composition of cyber-physical systems. In: 14th Annual IEEE Int'l. Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07), Washington, DC, USA. IEEE Computer Society, 2007, pp. 3–6.
2. Lee E.: Cyber physical systems: Design challenges. In: Proc. of the 11th IEEE Int'l. Symposium on Object Oriented Real-Time Distributed Computing (ISORC '08), May 2008, pp. 363–369.
3. Sztipanovits J., Karsai G.: Model-Integrated Computing. In: IEEE Computer 30, 1997, pp. 110-112.
4. Wrenn R., Nagel A., Owens R., Yao D., Neema H., Shi F., Smyth K., van Buskirk C., Porter J., Bapty T., Neema S., Sztipanovits J., Ceisel J., Mavris D.: Towards Automated Exploration and Assembly of Vehicle Design Models. In: ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC '2012), Chicago, Illinois, USA, August 12-15, 2012. Volume 2: 32nd Computers and Information in Engineering Conference, Parts A and B, pp. 1143-1152. doi:10.1115/DETC2012-71464.
5. Bryant R.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, vol. C-35, pp. 677-691, 1986.
6. Neema S., Sztipanovits J., Karsai K.: Constraint-based design-space exploration and model synthesis. In EMSOFT, 2003, pp. 290–305.
7. Karsai, G., Sztipanovits, J.: Model-Integrated Development of Cyber-Physical Systems. In: Proceedings of the 6th IFIP WG 10.2 international workshop on Software Technologies for Embedded and Ubiquitous Systems, October 01-03, 2008, Anacarpi, Capri Island, Italy. doi: 10.1007/978-3-540-87785-1_5.
8. DARPA Adaptive Vehicle Make Program. www.darpa.mil/Our_Work/TTO/Programs/Adaptive_Vehicle_Make__(AVM).aspx.
9. Lattmann Z., Nagel A., Scott J., Smith K., van Buskirk C., Porter J., Neema S., Bapty T., Sztipanovits J., Ceisel J., Mavris D.: Towards Automated Evaluation of Vehicle Dynamics in System-Level Designs. In: ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Volume 2: 32nd Computers and Information in Engineering Conference, Parts A and B, Chicago, Illinois, USA, August 12–15, 2012, pp. 1131-1141. doi:10.1115/DETC2012-71378.
10. Manolios P., Subramanian, G., Vroon D.: Automating component-based system assembly. In: ISSTA 2007, pp. 61-72.
11. Gries M.: Methods for evaluating and covering the design space during early design development. In: Integration, 38(2):131{183, 2004.