META-X DESIGN FLOW TOOLS

FINAL REPORT

Ted Bapty Sandeep Neema, Janos Sztipanovits

Institute for Software Integrated Systems, Vanderbilt University

March 21, 2013

Contract Number: FA8650-10-C-7082

Agency: AFRL

CONTENTS

LIST OF FIGURES

1 SUMMARY

The META language and tool flow has been developed to support model-based, component-centric development of complex cyber-physical systems. This report describes the basic concepts driving the approach, the language implementation, and the tools developed to implement the design flow.

The overall process is described, showing how components are used in a successive refinement of design spaces to converge upon a set of feasible designs. The core concepts and semantic foundations of the language are described, along with an overview of the language. Design space exploration is presented as implemented in the DESERT tool. Composition of models to supported analysis tools is described, along with the concept of executable requirements in the form of Test Benches.

Mechanisms to support multi-fidelity/multi-abstraction representation and analysis of system models are described, along with the tools implementing the balance between accuracy and cost of computations. Analysis of system dynamics using Modelica and Bond Graphs is described for lumped parameter analysis. Geometric analysis tools using automated analysis of CAD models are described, along with analysis using Finite Element Methods.

An overall execution infrastructure was developed to manage execution of computationally intensive analyses on parallel computers, along with visualization techniques. Verification methods are described.

Finally, experiences using the tools in FANG and the user threads are described.

2 INTRODUCTION

The META-X Toolset has been developed to support component-based design of complex cyberphysical systems. These systems include military vehicles, as exemplified by the systems defined by the requirements for the Fast Adaptable Next-Generation Ground Vehicle (FANG) Competition.

The tools implement a set of concepts formulated under the META and AVM program to dramatically reduce the cost and schedule required to achieve the first limited production of a target vehicle. The primary concepts are:

- Component-Based Design is defined as a deep, complete set of components being developed via the C2M2L program, to include full, multi-domain, producible components. The components are, by design, composable to produce subsystems and systems that can be analyzed, simulated, assembled in 3D, and verified.
- Domain-Specific Modeling Languages (DSML) and modeling tools (developed outside this SOW) to support construction of designs and design spaces to represent cyberphysical systems in terms of architectures of interconnected components, multi-physical interactions, and multi-dimensional spaces of design options. As there are three primary options for META-X tools and design data need to be communicated with the AVM manufacturing tools, an interchange format has been defined to support translation of designs and components between vendors.
- Design flow tools, to support composition of the component-based designs for a variety of analyses, including:
    - Constraint-based design space exploration,
    - Dynamics simulation,
    - 3D structural/thermal analysis with finite element analysis (FEA).
- Leverage off-the-shelf open source and commercial tools.
- Support scoring of designs by computing Key Performance Parameters on the system and scoring designs against requirements and stakeholder preferences (via MAUF).
- Complexity analysis tools help the designer rapidly assess the structural and parametric complexity of the design, providing comparison of architectures for future developmental success. Lower complexity leads to less design effort, reduced risk, and fewer unintended problems.
- Verification tools help to identify potential problems prior to build. Multiple approaches are being applied to system verification, ensuring scalability and coverage. Qualitative and Relational abstractions are used to explore system behavior. Probabilistic state machines are used for evaluation of potential system faults and culprit analysis. Probabilistic verification techniques are used to create a Probabilistic Certificate of Correctness, identifying the impact of component property variation on the ability of the system to meet requirements. The PCC is intended to help reduce the level of testing needed to establish confidence in delivered system performance.
- Manufacturing composition tools produce design snapshots for evaluation of manufacturability by iFAB/Foundry, and the final Technical Data Package needed to produce

the target system. The META design tools are evolving to produce an increasing fraction of the data needed to manufacture the system and integrate manufacturability tests in the design flow

- Vehicle Forge interfaces permit the seamless interaction with the repository, which houses the global component library, as well as the project collaboration for teams of designers. The META tools must import curated components and export new/experimental components for curation to incorporate and produce components. For project collaboration, the tools must upload and download designs (intermediate and complete) for intra-team collaboration and exchange of results. For scoring, the META tools must upload analysis results in a controlled manner to provide relative comparison of competing designs.

This report describes the development of the META tools under the META Design Flow FA8650-10-C-7082 contract.

3 META X DESIGN FLOW AND TOOLS

The META X Design Flow methodology and tools that were developed to support this methodology are described in the sections below.

3.1 META DESIGN FLOW AND DESIGN ABSTRACTIONS

META design flow supports analysis of target designs over a range of domains and abstractions.

- Physical domains are selectable in the META composition process, to suit the needs of an analysis:
    - Understanding and evaluation of CyberPhysical designs requires analysis in many physical domains. As there are interactions between domains (e.g., Mechanical friction generates heat), multiple physical domains must be evaluated concurrently.
    - Any specific evaluation is toward a purpose, computing metrics on a design to a needed fidelity and range of interactions. Evaluating against a greater set of physical domains typically results in greater computation, taking more time in the best case, or resulting in intractable analyses at worst.

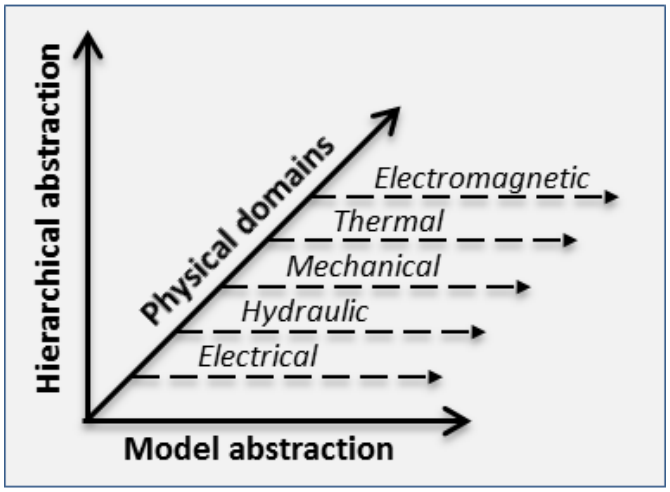


FIGURE 1: META TARGET DOMAINS AND ABSTRACTIONS

- Abstractions are supported on axes of the evaluation space
    - Model abstraction permits the user to select the appropriate level of detail in the component to achieve an analysis goal. Modeling a vehicle suspension is an example. For rough performance and speed across a nominal terrain, the suspension can be abstracted to a simple rolling resistance. To compute the power transmitted from the terrain or obstacle, a fidelity level capturing spring-damper responses of the suspension is required.

- Hierarchical abstractions permit using a single model representing the combined behavior of all subsystems, when feasible. This can result in greatly simplified computation and can be used to support in-the-loop computation.

Figure 2 illustrates multiple levels of abstraction, representative size of the problem, and mechanisms for support of the abstraction under CyPhy/META design flow tools.

Traversing along the Physical Domains, the models represent behavior and interfaces appropriate to the component and the level of detail needed for a range of applications. For example, motor generates mechanical power, uses electrical energy, and produces thermal energy as waste heat. Simple analysis may ignore heat for 1st order analysis.

Traversing along Model Abstraction, a model can use a variety of mathematical representations for capturing the behavior of the component. The figure below shows a few of these for the electrical motor, from a Qualitative representation of basic input/output and states, thru ordinary differential equations, to full geometrical/distributed structural, force, and electromagnetic field representations. Each of these abstractions is appropriate for certain evaluations, at a certain computational cost.

		Size of State Space	Compute Complexity	CyPhy Tool Support Automating Analysis Composition
Electrical Motor Model Abstraction	R_a = Motor internal resistance L_a = Motor internal inductance r = Current/Torque Ratio J = Internal rotational inertia μ = Shaft angular damping			
Quali-ative	Legend: i_a wrt. Steady state, J_m wrt. Steady state	2 Disc. Vars	Abstraction $O(n)$ Worst-Case QE $O(3^n)$	10s QR Testbench + Modelica Comp + PARC QR Mapping
Quantitative/ Relational	$\dot{x} = Ax + Bu$ $y = Cx$ $D^T x' - D^T x \le b^T u_k$ $D = f_{R,D}(A,B)$ $b = f_{R,b}(A,B)$	2 Cont. Vars	Abstraction $O(n^3)$ Worst-Case BMC $O(c^{n \cdot k})$	10s RA Tstbch + + SRI QR Mapping
ODE / Bond Graph + *Cyber Discrete Event*	Se, I:L_a, r, GY, I:J, R:R_a, R:μ	2 Cont Vars	$O(n \cdot T)$ n = f(# states) T = # time samples	5s Dyn TB Map to Simulink & Modelica (10s) + ESMOL & TrueTime
PDE → Finite Element Solution		Inf → 5,000 Pts * 2 *4	Super-computer, Iterative Solution, Single Point in time	2m FEA TB Map to CAD (33s) + Gridding & Constraint/Forcing Func + NASTRAN Deck Gen

FIGURE 2: EXAMPLE ELECTRIC MOTOR WITH META ABSTRACTIONS

Typically, an electrical motor within a context is represented by the schematic on the top line of the table. The motor is treated as a 3 terminal component, conducting electrical power thru inductive and resistive loads, and producing a torque with inertia and rotating resistance.

Qualitative representations of the motor are concerned with the direction and acceleration of the device, using two state discrete variables. The qualitative computational complexity is order N. With this approach, we can explore the discrete space of the system.

Relational abstractions linearize the system dynamics, again using two variables, but continuous representations. Equations can be computed with a worst-case Order N cubed, with simplification possible. We can rapidly explore the continuous state space of the system.

Ordinary Differential Equation/Hybrid System Models capture the nonlinear behavior of the system with a lumped parameter model. These equations are amenable to simulations, which, with proper stimuli, can explore trajectories of system behavior over timeframes related to the inertial time constants of the system

Finally, a Partial differential Equation formalism captures the full three-dimensional (3-D) electromagnetic current and flux interactions of the motor's stators and rotors and windings. Forces can be computed at specific angular positions, and geometrical parameters can be evaluated. Much higher resolution models are required, along with complex gridding and spatial solvers. The complexity of this calculation is orders of magnitude greater than the others.

META also supports hierarchical abstraction of a system, capturing system/subsystem, part-whole hierarchies. In addition to allowing gradual refinement of systems at design time, the computation time and accuracy of a system analysis can be controlled by representing peripheral subsystems at an aggregated level, while critical systems are analyzed in detail. The META Language and Tool Flow projects have been developed to fully support these abstractions, both in representing a system and in automatic composition of analysis of these systems in a computationally efficient manner.

META design flow support of these abstractions and phenomena is described in the detailed sections below.

3.2 META AND THE AVM DESIGN FLOW

The conceptual design flow for AVM is shown in Figure 3.

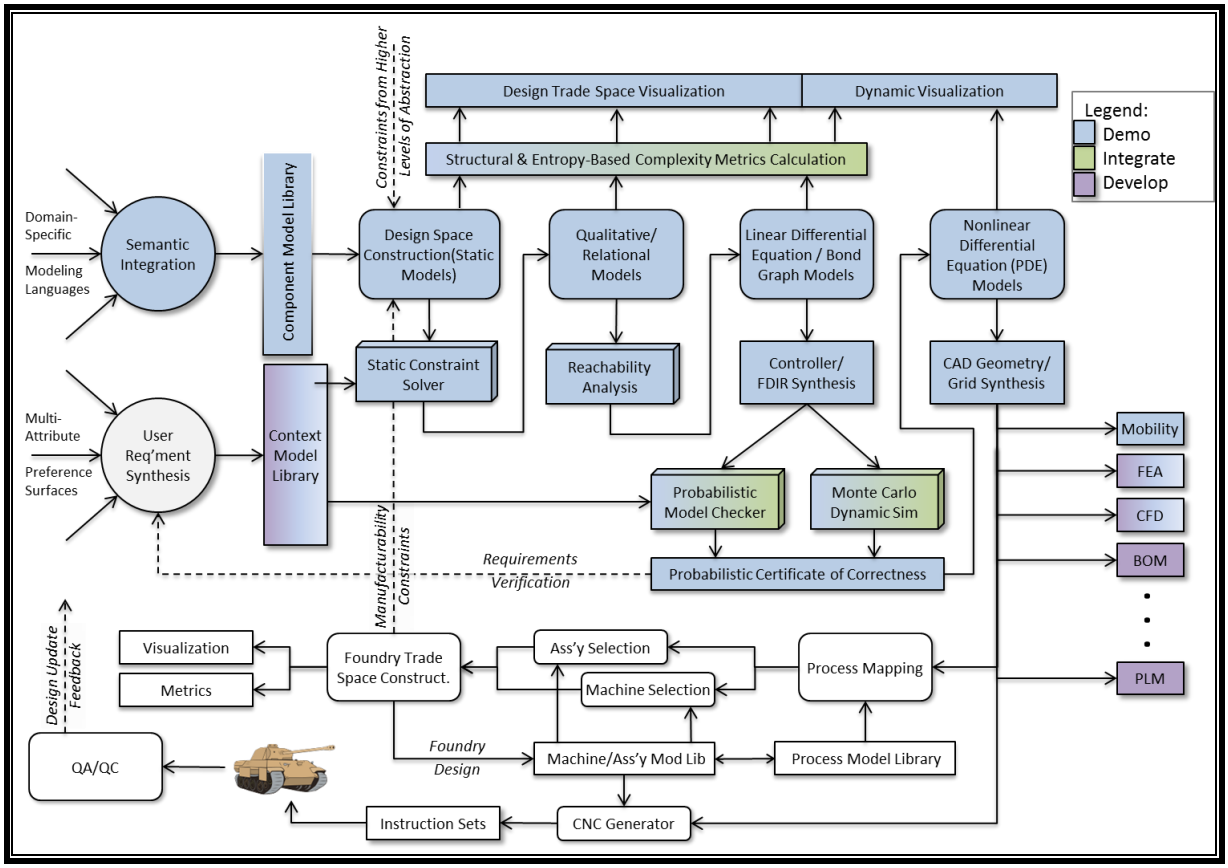


FIGURE 3: AVM OVERALL INFORMATION FLOW DIAGRAM

This flow diagram was conceived by DARPA with input from META and other AVM participants. The diagram has been color coded to show the state of the META tools in supporting the AVM goals.

Semantic Integration is a task of the META Design Language, which is used by the Design Flow. It is currently developed, however in a flexible manner to allow expansion as new requirements evolve. META Design Flow is closely coupled with this effort, as composition of analyses relies on a full and strict alignment with model semantics.

Component and Context models are the physical instantiation of the model semantics, capturing component multiphysics behavior. These are integrated and continue to evolve and expand in content, but formats and semantics are integrated into the flow.

Design space exploration tools allow rapid evaluation and constraint-based reduction of large design spaces into feasible sets. DESERT implements a static constraint solver using BDD techniques, the implementation under META Design Flow (described below), are fully integrated into the tools.

Qualitative Reasoning explores the system's performance space with the envisioned qualitative model and has been integrated into the design flow. These tools have been developed and adapted to

AVM by PARC. While the tools are integrated, implementation of C2M2L models prevented the widespread use of these tools in FANG 1.

Relational Abstractions, also explores the system's performance space, but using an employing relational model abstraction, has been integrated into the design flow. These tools have been developed and adapted to AVM by SRI. While the tools are integrated, implementation of C2M2L models prevented the widespread use of these tools in FANG 1.

Ordinary Differential Equation-based analysis/Dynamics Analysis forms the core of the dynamics analysis capabilities implemented in META and used in FANG 1. These capabilities have progressed thru two phases during the META design flow project. Initially, Bond Graph-based dynamics simulations were used for acausal component behavior modeling. Models were translated to a Simulink-based execution platform. As a result of C2M2L component supplier decisions, an additional Modelica-based execution platform was added. This had impacts on both language and tools. Both were developed and integrated into the design flow.

The Controller modeling and simulation capability was integrated into the META design flow, supporting both state machine and signal flow specification of controllers, with a code-synthesis and co-simulation with the dynamics simulation. This is integrated and used in FANG.

Nonlinear Analysis, in the form of Finite Element Analysis has been integrated into the tools, with the ability to compute static stress for a CyPhy system model. This tool has been integrated and demonstrated, however no FANG 1 requirements needed this analysis for the competition. This capability implements a composition of geometry files, with gridding and composition of Abaqus/NASTRAN input files and post-processing of results.

CAD composition tools have been developed and integrated into the META tool flow. These permit creation of an assembled 3D model of the system geometry, using the geometry of the components. This tool has been heavily used in the FANG competition.

Mobility Simulation has been developed and integrated in an experimental mode. The use in FANG was limited by component capabilities, primarily the ability to compose kinematic joints from the models. The tool permits co-simulation of the 3D physics with dynamics models.

Complexity Metrics were developed and integrated into the tools. Two types of complexity metrics were created under subcontracts with MIT. A structural complexity metric computes the graph-energy of a design, combining component complexity with interaction graph-arc strength from Oli DeWeck's team. An information uncertainty-based complexity metric uses the uncertainty encountered in simulation of the system. The information uncertainty metric was developed by Karen Wilcox's team at MIT.

3.3 META DESIGN FLOW AND INTEGRATION OF LANGUAGES, DESIGN PROCESSES, AND TOOLS

The META Tool flow maps to a variety of tools and technologies. Figure 4 summarizes these concepts.

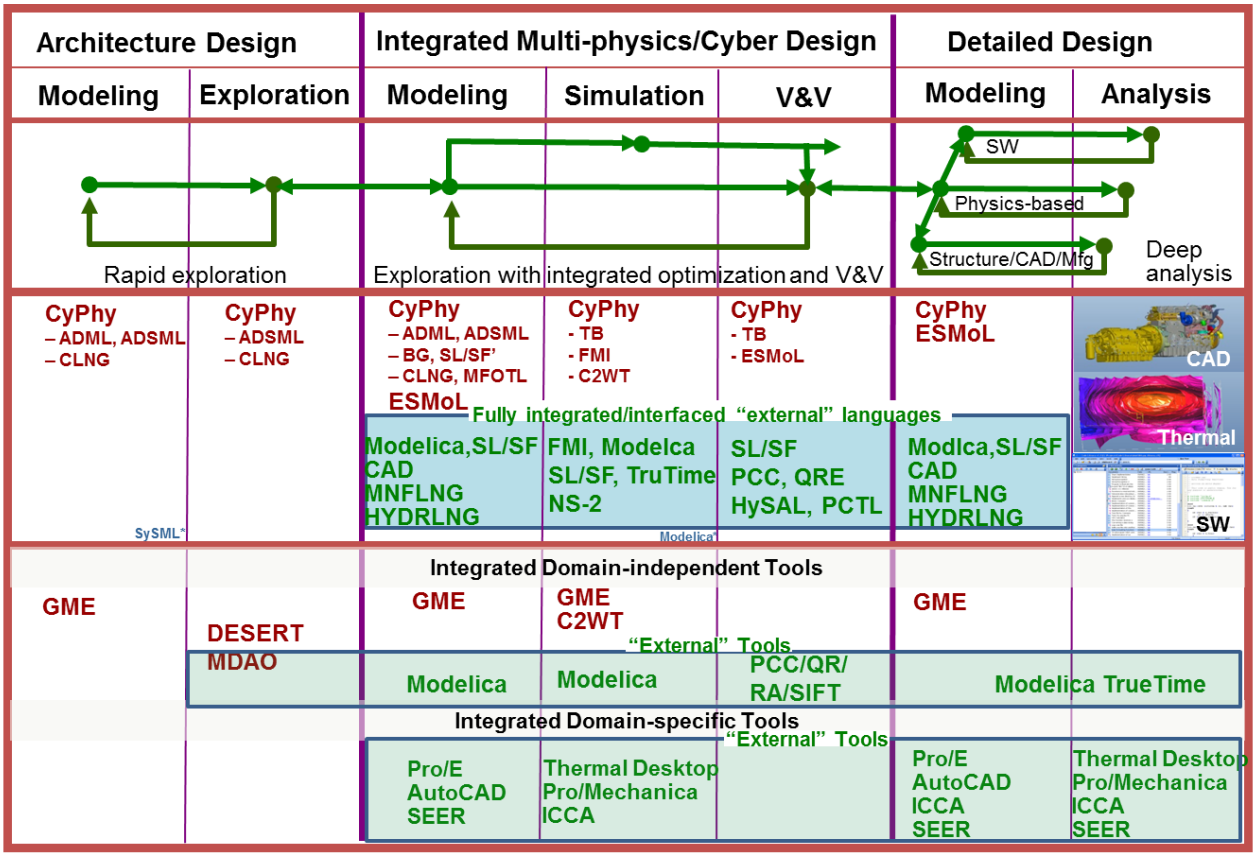


FIGURE 4: MAPPING OF LANGUAGE, DESIGN FLOW, AND TOOLS

META Design Flow leverages many concepts and tools from research and industry. It also strives to support the conventional phases of design flow, from conceptual/architectural to detailed design flow. Below is a brief summary of the concepts expressed:

- Architecture design or Conceptual phase uses CyPhy and its sublanguages to express components used and component/architectural/parametric design space options. Exploration occurs based on static evaluation of component and system properties. (e.g., weight/parts costs, interface compatibility, et cetera). The GME/CyPhy and DESERT tools support these operations.
- Integrated Multi-Physics/Cyber Design phases implement a modeling/ simulation/ Verification & Validation loop focused on refining designs to achieve target system requirements with a satisfactory design. Specific tools and associated languages include:
    - Design Modeling/Specification: CyPhy/GME for system architecture, CAD (ProE) for geometry, Bond Graph & Modelica (Dymola, OpenModelia) for dynamics Behavior, SEER for costing, StateFlow, SignalFlow Language for control algorithms (ESMOL)

    - Simulation Analysis: Test Bench (CyPhy), FMI for coupled multi-sims, C2WindTunnel (C2WT) for simulation integration, along with other domain tools.
    - Verification and Validation leverages the dynamics simulations to compute Probabilistic Certificate of Correctness using OSU probabilistic computations in dynamics simulation models, under the runtime of OpenMDAO. Other exploration of performance state space is accomplished with reduced fidelity models of Qualitative Reasoning, Relational Abstraction (for continuous dynamics) and the SIFT tools for probabilistic models of failure.
- Detailed Design completes the design process and computes deeper domain analyses, with the goal of providing more accuracy and discovery of unintended interactions or black swans, that is, the "extreme impact of certain kinds of rare and unpredictable events (outliers) and humans' tendency to find simplistic explanations for these events retrospectively" (Wikipedia). Specific tools include:
    - Modelica/StateFlow/SignalFlow Language for coupled dynamics with detailed TrueTime computational simulations, and Finite Element-Based computations for structural, thermal, and Fluids.

4 META DESIGN FLOW TOOL ARCHITECTURE

The META Design Flow tools for CyPhy computational architecture is shown below. Each block is a subsystem that integrates with models, and/or domain tools.

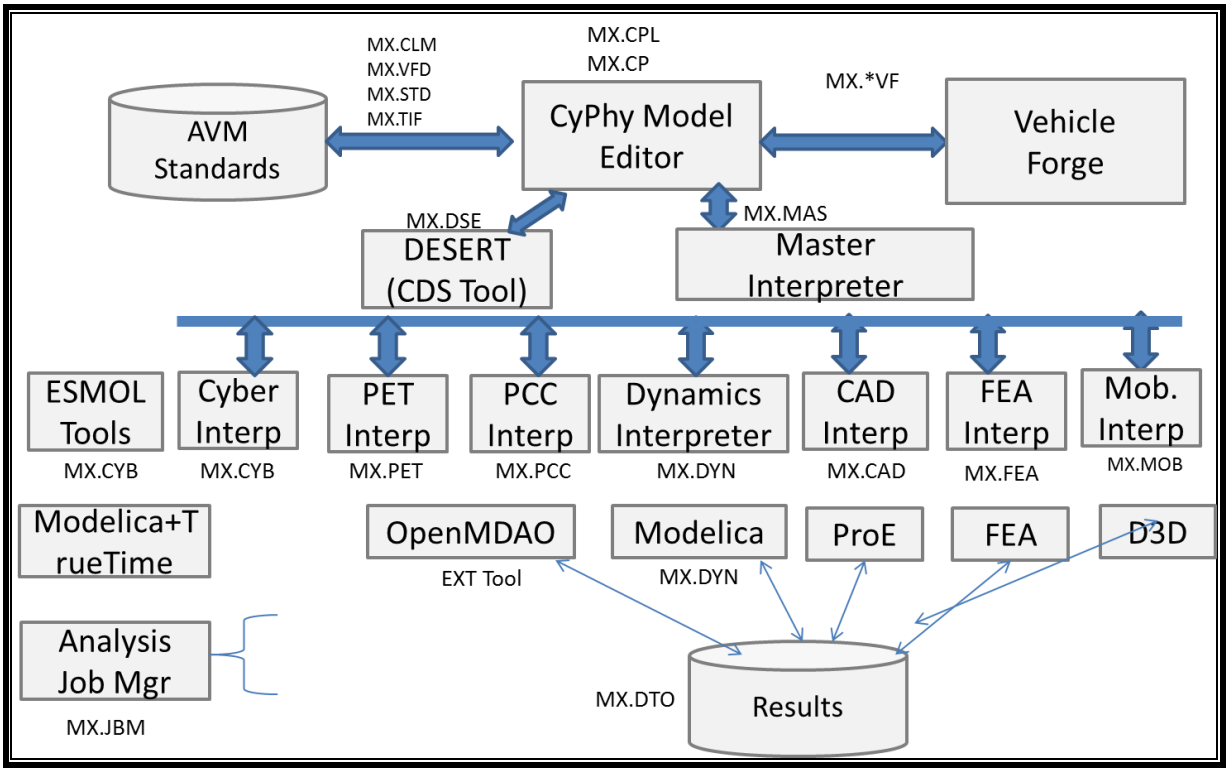


FIGURE 5: META DESIGN FLOW COMPUTATIONAL ARCHITECTURE

The external tools are shown on the top line of the diagram. They were not developed under META Design Flow, but are shown for context. Abbreviations (e.g., MX.CLM) are references to the internal design process task tracking/schedule, and can be ignored for the purpose of this report.

The CyPhy Model Editor provides the user interface, model repository, and backplane for tool integration. It provides a structured, programmatic interface to models, and a mechanism to enforce domain-specific language semantics on the models. The editor provides bindings for several languages for model access and manipulation, which are used by the various composition engines (C#, C++, Python, Java)

Desert operates on the CyPhy model, converting from a design space alternative structure to a set of design points based on constraint satisfaction. This tool performs in-place modification of the CyPhy model, adding a specification that

The Master Interpreter executes design flow operations on models and DESERT generated constrained design configuration models, coordinating execution of tools in the correct order. Typically, the component tools are not manually accessed by the user. In addition to coordinating execution, the Master Interpreter manages the flow of results and the coordination of test bench results into a logical, consistent structure.

The Dynamics Interpreter converts the design architecture and component models as a system under test (SUT) along with the test bench containing scenarios, environments, and post processing into an executable Modelica model with surrounding metrics extraction operators. The targets for the Dynamics interpreter are OpenModelica and Dymola, using Python 2.7 for post processing.

The CAD interpreter evaluates models for their structural connections and produces a connectivity specification file. Another service interprets the connectivity and constructs the target CAD file via the automation interface of Creo/ProE to apply assembly constraints, produce standard CAD file outputs, and compute a set of metrics for evaluations such as bounding box/transportability and center of gravity.

Finite Element analysis composition leverages CAD composition to create geometric representations of a design, followed by preparation of the finite element analysis (FEA) input deck (Grids, Boundary conditions, Forcing functions). FEA codes are executed and results extracted (e.g., max Von Mises stress). This tool leverages NASTRAN, Abaqus, and Calculix.

The PET/PCC interpreter evaluates PET models and constructs a configuration for execution under OpenMDAO. OpenMDAO is an open-source toolkit for implementing Analysis chains under the control of a DOE or parameter optimization service. For the purposes of the META design flow tools, the OSU PCC methods have been incorporated into the OpenMDAO framework to support computation of Probabilistic Certificate= of Correctness data.

4.1 DESIGN SPACE EXPLORATION

The DESERT Tool is a highly scalable mechanism for managing large-scale design spaces, such as those that can be easily represented in the CyPhy language. Design space is expanded by including structural alternatives, via CyPhy language constructs. The nominal semantics of a component or assembly alternative is to include the all permutations of all choices in the model. Consequently, with even a few component alternatives, design spaces can grow well beyond the capability to simulate or even elaborate design options.

DESERT uses the technique of Multi-Terminal Binary Decision Diagrams to compactly represent the design alternatives. Once represented, constraints can be defined as operators on the MTBDD to reduce the open space of the design. The design space does not need to be elaborated until after all constraints are applied, and design point instances are needed for further analysis.

The design space representation is captured in the schema below. The design architecture and alternatives are represented in Spaces and Elements, with associated properties and values.

Constraints are captured in relations, sets, and formulas.

The DESERT engine converts these into internal BDD representations, and provides facilities for applying constraints to spaces.

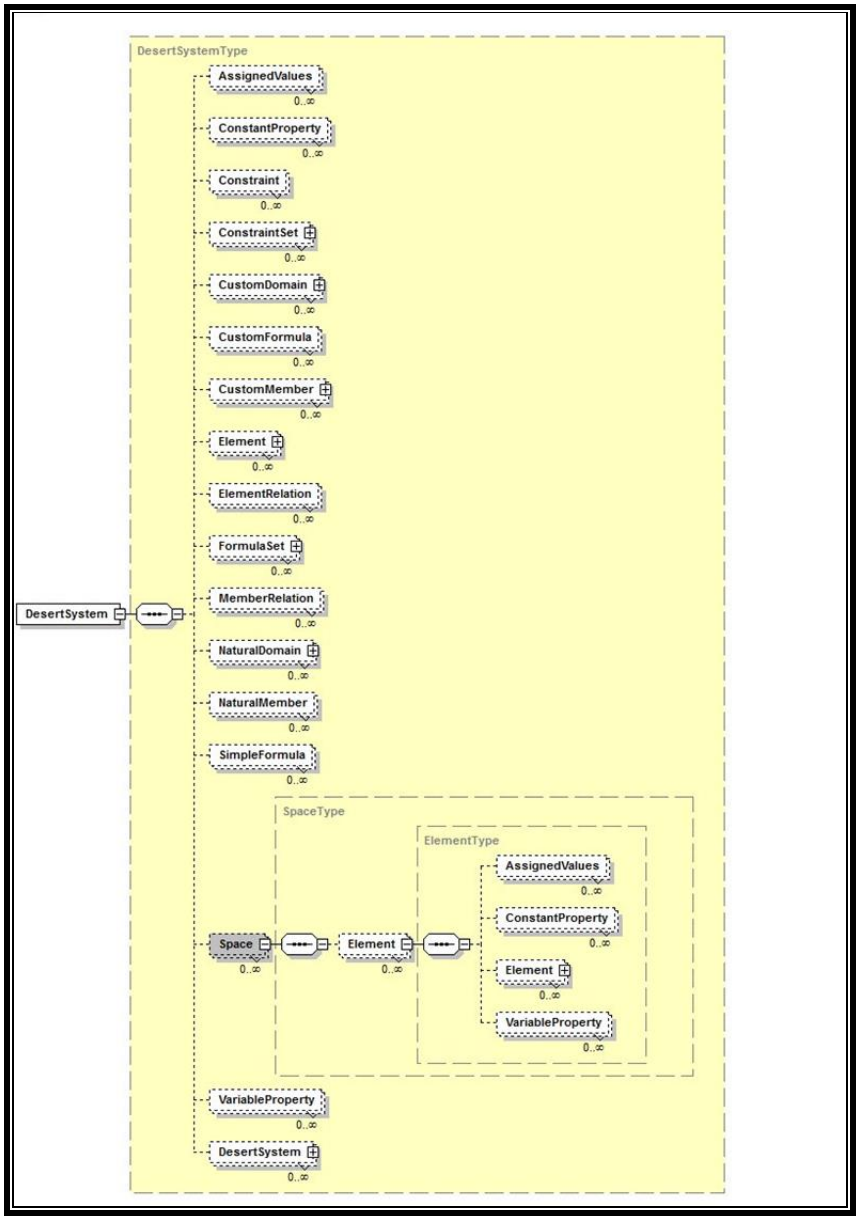


FIGURE 6: DESERT CONSTRAINT ENGINE INPUT

The output of DESERT is a set of configurations, which contains constrained architectures of the system after iterative application of system constraints.

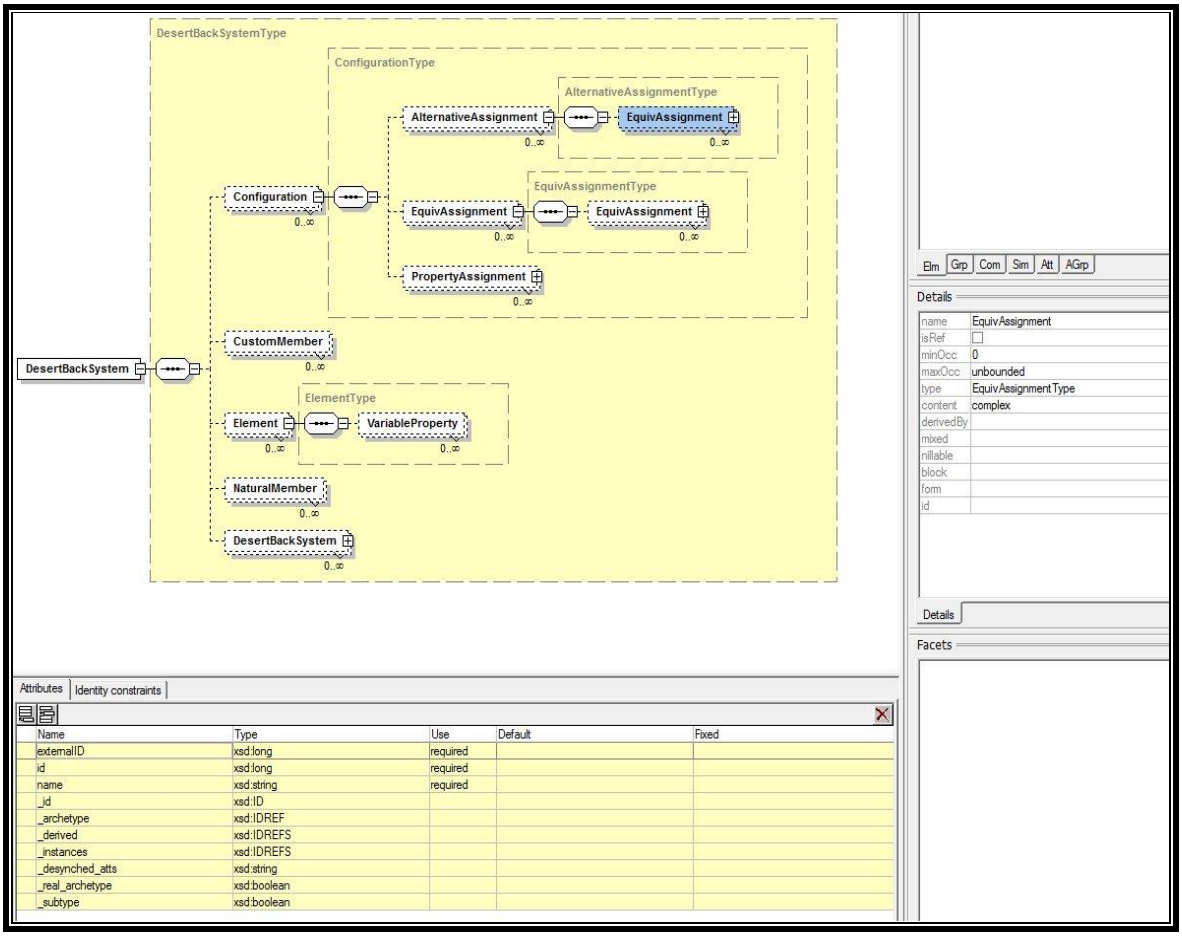


FIGURE 7: DESERT TOOL OUTPUT SCHEMA

The tool is integrated into the CyPhy editor. The figures below show an example execution of the DESERT tool.

The first figure shows a set of constraints available for application. Constraints can be one of several types (See CyPhy Language for description of constraint models):

- User Defined Constraints: these can be formulas, with simple operators on component properties (e.g., Sum on Mass)
- Relationship Constraints: relating the selection of one option to a constraint on another. For example, a symmetry constraint would require left and right components to match.
- Compatibility Constraints: automatically generated by DESERT, requiring a property on one side of an interface to match the other side. For example, this is used to ensure that two mechanical interfaces are compatible.
- Property Constraints: automatically generated by DESERT, requires parameters of a component to be within the stated range of those properties.

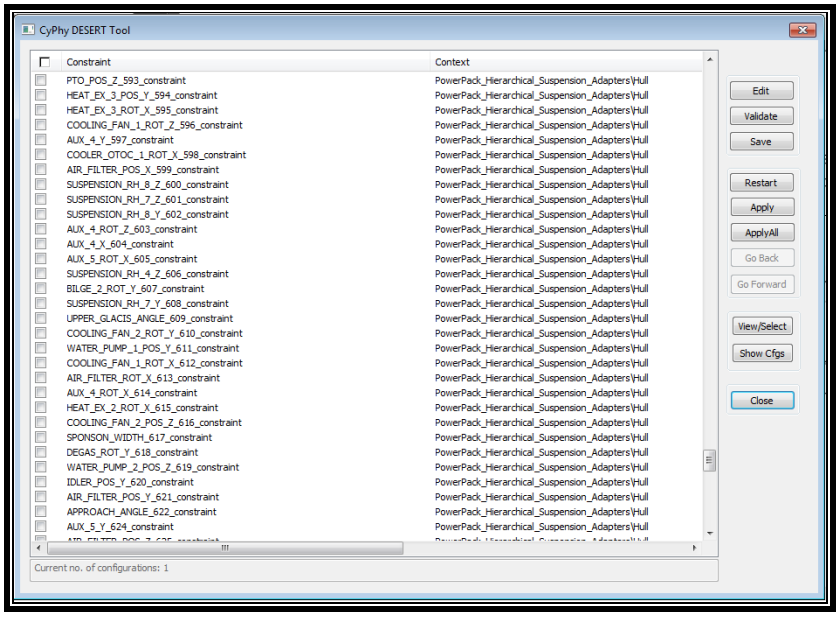


FIGURE 8: EXAMPLE SYSTEM CONSTRAINTS

These constraints can be individually selected and applied, or applied all at once. The tool allows constraint application to be rolled back, to support exploratory application of individual constraints or sets of constraints.

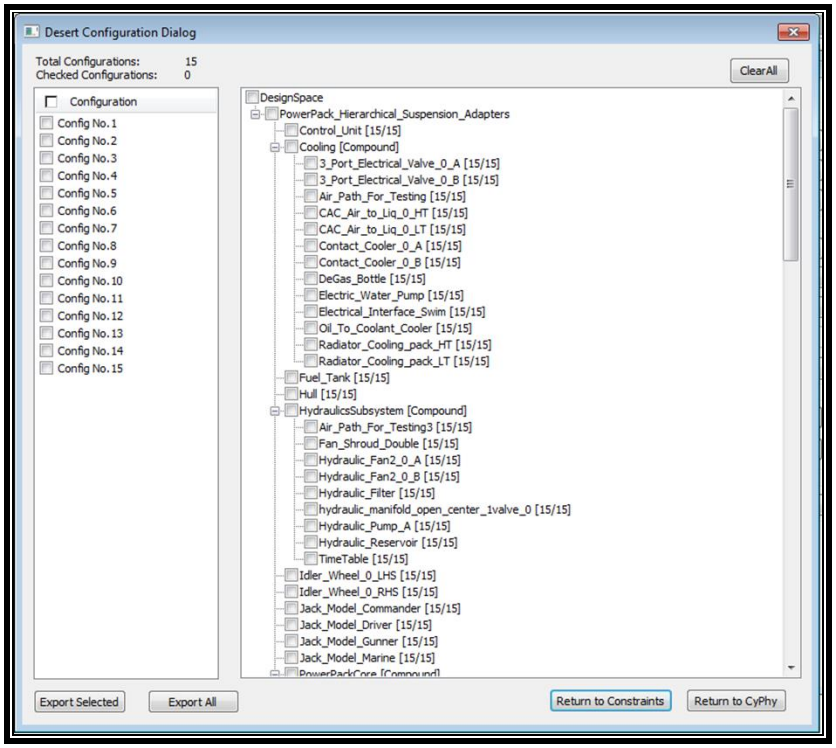


FIGURE 9: VIEWING AND EXPORTING CONSTRAINED DESIGN SPACE CONFIGURATIONS

4.2 COMPOSITION OF SYSTEM MODEL ANALYSES

The META design flow tools focus on automated composition of analyses from a common model. The figure below summarizes the span of analyses implemented during the META X project. The common model is CyPhy, developed under a parallel META X effort, which supports multi-physics, multiple levels of abstraction modeling of components, assemblies of components into systems, and design spaces of components, assemblies, and parameters.

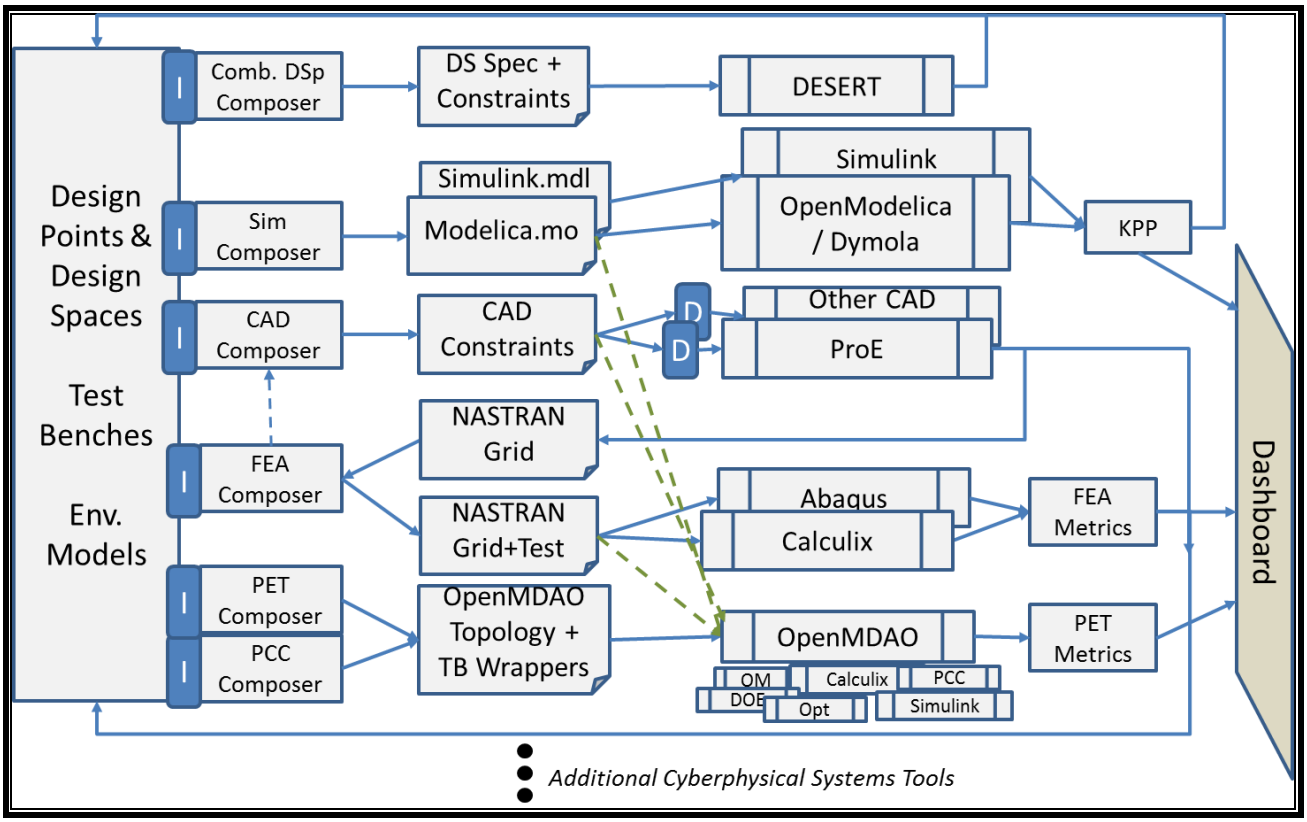


FIGURE 10: COMPOSITION OF ANALYSES SUPPORTED BY META

META Design Flow composition paths are shown in the figure above, along with the intermediate representations and the external tools that are leveraged to accomplish the specific domain analysis.

Note that one of the major objectives with META Design Flow is to build a completely Open Source tool flow. This goal is sometimes at odds with tool quality in terms of scalability, performance, and/or compatibility with the component space. For this reason, multiple similar target tools have been integrated to support open/free software and robust solutions. Ideally, these two coincide, or can be encouraged to converge with additional development. This has been the case with Modelica, in particular. The commercial Dymola package is often able to execute a larger fraction of the Modelica Standard Library and C2M2L library. The open source OpenModelica, while behind in terms of MSL support, is rapidly catching up, especially considering the modest resources applied under this contract.

Support for open source CAD packages has not progressed as rapidly, due to the advantage in required features for the FANG analyses available in PTC Creo vs. open source (e.g., OpenCascade). Specifically, these include constraint managers, gridding and integration support with FEA, and ability to export/import from other commercial packages.

FEA packages have been integrated, with the commercial solvers supporting advanced capabilities such as adaptive gridding, but experiments have found good accuracy with Calculix and OpenFOAM.

Early experiments with optimization infrastructures such as iSight and ModelCenter showed disadvantages in terms of openness and integration support when compared to OpenMDAO. As a NASA-supported tool, with a growing user base, we anticipate OpenMDAO to have a better long-term growth path.

It should be noted that some of these techniques rely on other composition tools. The META Design Flow supports user defined series of test benches, allowing complex computations to be specified and executed. Arbitrary topologies of analyses can also be implemented in a PET optimization or DOE loop.

4.3 TEST BENCH CONCEPTS

The test bench concept was created for the META Design Flow project and implemented in a semantically well-founded manner within the CyPhy Language. In general, a Test Bench is an executable specification of a requirement. Test benches are used throughout the system as shown in the figure below.

Test Benches in META are a reusable, succinct, complete, and executable representation of an analysis specification. Test Benches Contain:

- SUT - This is a reference (link) to a design OR design space of the system or subsystem to be tested. The system will typically have a standard set of interfaces and parameters to allow different designs to be placed in the test bench for reuse of the test bench
- Drivers and Boundary Conditions - This is the set of signals that stimulate the system to set conditions under which the system models will be measured, or drive the system through a state trajectory of interest. These drivers are Test Components, following many of the same semantics that are used for META components.
- Environment Specification - This specifies any environmental conditions under which the system will be evaluated.
- Metrics, Requirements, and Evaluation - These are components that process system outputs to compute quantities of interest (e.g., time-to-accelerate, power absorbed, average temperature, maximum stress. Metrics identify the quantities of interest, and requirements are the links to the system requirements tested by the test bench.
- Analysis Tool Settings - These set the parameters for the analysis, such as simulation time, solver method, maximum time step, etc.

The test benches are tied to specific workflows. Currently, CyPhy/OpenMETA implements test benches for:

- *Dynamics* - using a lumped parameter model executed in the Modelica language. Dynamics covers mechanical, electrical, hydraulic, and thermal domains.
- *Structural* - using 3D CAD assemblies to evaluate the physical compatibility of parts, locate potential interferences, and compute physical properties including center of gravity, bounding box, and assembled location of specific points on the system.
- *Finite Element* - using Finite element techniques to compute stress/strain, thermal propagation, computational fluid dynamics, etc.
- *Mobility* - using the NATO Reference Mobility Model (NRMM) to predict vehicle mobility based on aggregate system properties,
- *Cyber* - co-simulating dynamics with a time-based software, processor and network simulation.
- *Manufacturability* - creating the 3D CAD file, a set of properties for each manufactured join between parts, and an electronic bill of materials. From this design package, iFAB can predict a cost and schedule to manufacture the system.
- *Complexity* - evaluating the graph-energy complexity of the system based on its component complexity and structure of its connections. The complexity metric will correlate with system cost and schedule.

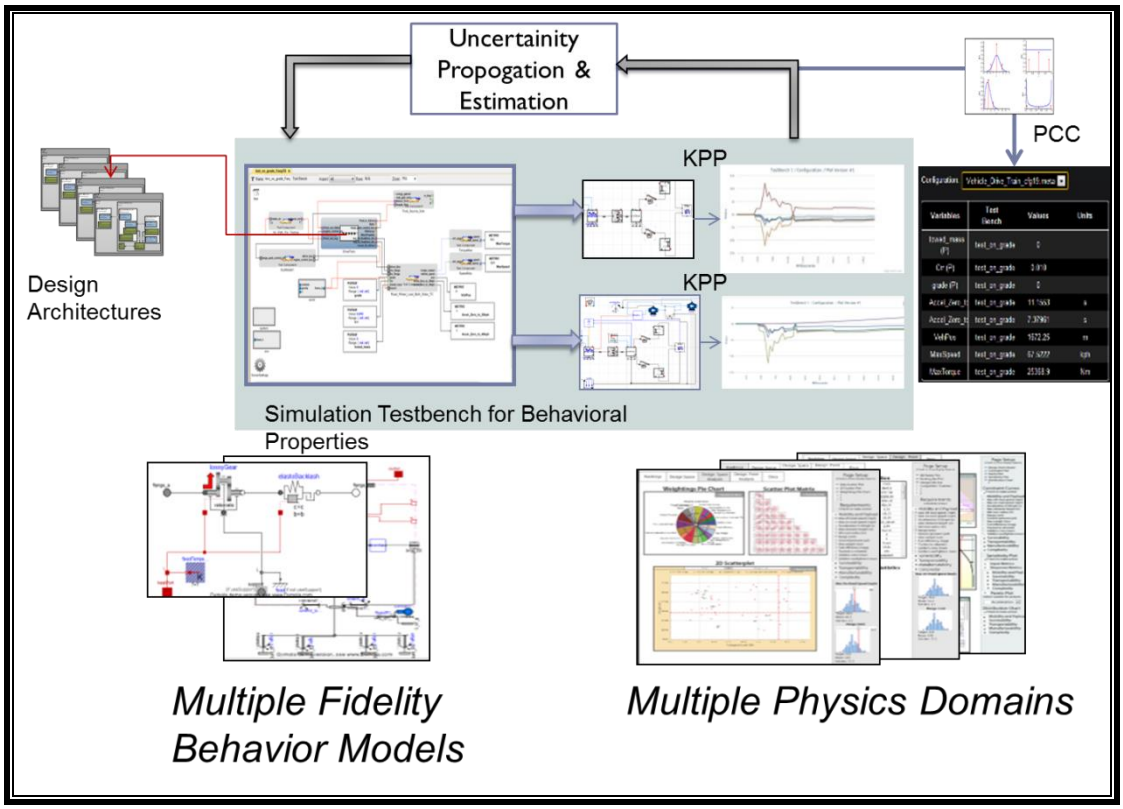


FIGURE 11: TEST BENCH APPLICATIONS WITHIN META DESIGN FLOW

The test bench core supports an analysis topology, focusing on a System Under Test, SUT. The SUT can be a single design point, or more importantly, a design space.

Test benches are used to compute specific metrics, which can be linked to system requirements. In the use case of FANG, these can be Key Performance Parameters (KPPs) or any other priority requirements. Test bench results can be visualized via the SimViz tool (described below).

Test benches can be used in isolation, connected in a workflow a Suite of Testbenches (SOT) tool, or employed by the Parametric Exploration Tool and/or PCC tool under OpenMDAO.

While all test benches follow a general pattern, test benches are customized to the analysis domain, supporting the analysis domain concepts and, where necessary, details of the tools.

4.4 MULTIPLE ABSTRACTION SIMULATION CONTROLS

As a Test Bench is created to evaluate a specific requirement, the test bench must capture the required physics domains and level of abstractions of a component behavior model. This capability is controlled via the Fidelity Selector form below.

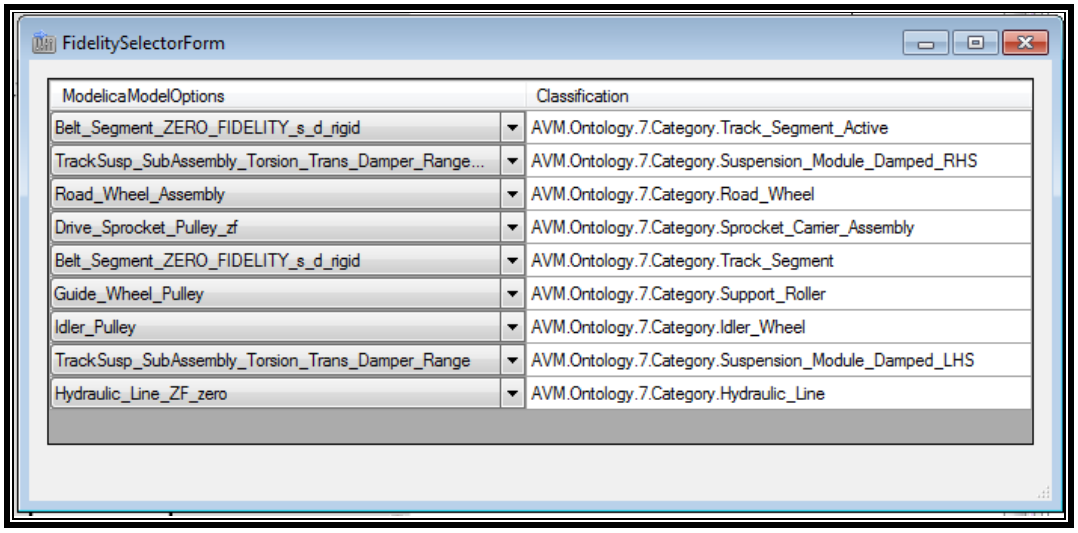


FIGURE 12: SELECTION OF MODEL FIDELITY/COMPONENT ABSTRACTION

The fidelity selector locates all components within a design which have multiple fidelity options. It presents the test bench designer with a form to allow selection of a component fidelity/phenomena representation, by component class.

The abstraction/phenomena selection is stored with the test bench, so that all future executions of the test bench will be composed with the specified fidelity and resultant accuracy/phenomena. Note that this is supported only for the Dynamics composition, but is planned for the PDE-based analyses in a future version.

4.5 DESIGN FLOW MASTER INTERPRETER

The design flow Master Interpreter (MI) is an integration driver for all system test benches, and test bench suites. The MI supports the following tasks in orchestration of a test bench execution:

- Elaboration of design space within a test bench: This effectively walks through all valid design points, and prepares an “Instance Model”, where design space concepts are replaced with specific component selections.
- Execution of precursor interpreters, such as the Formula Evaluator, which resolves any mathematical dependencies between component properties and parameters or system parameters.
- Preparation of results templates, where the test benches store metrics results, and various index files maintaining test bench status and history.
- Preparation of any files in the execution directory, including pre/post processing of results.

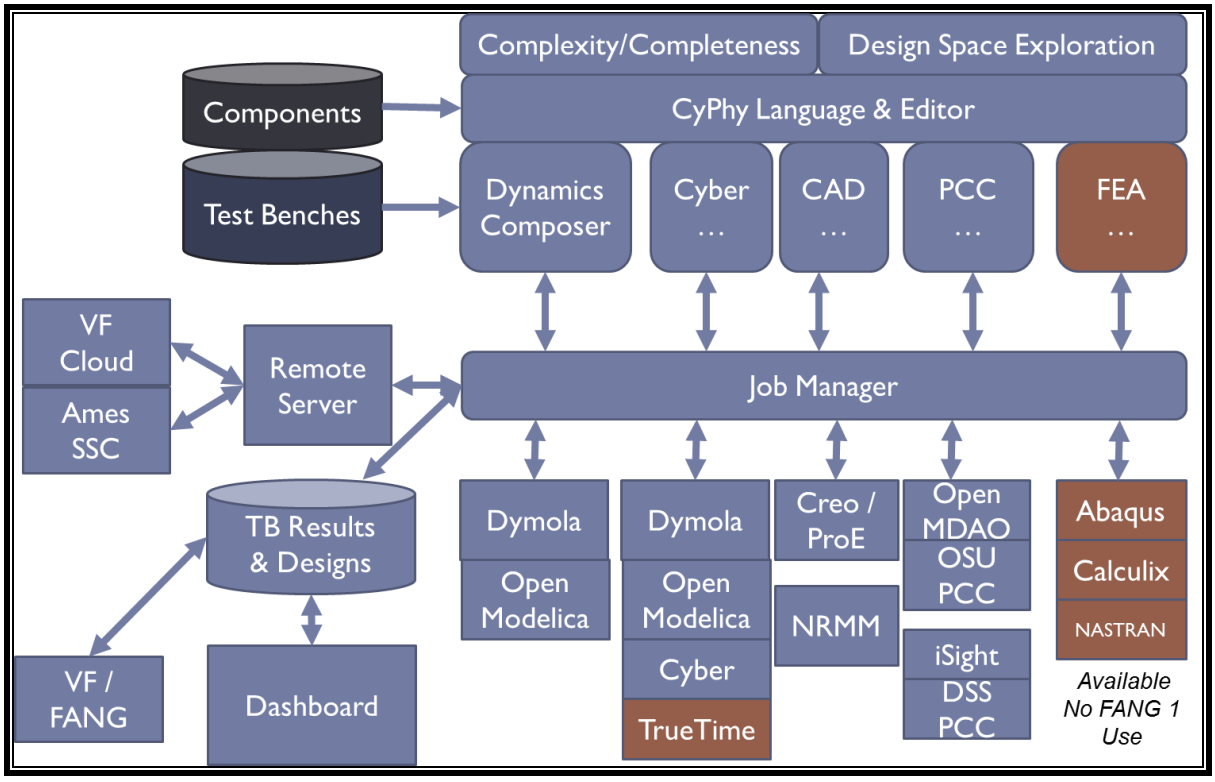


FIGURE 13: OVERALL TOOL ARCHITECTURE

4.6 DYNAMICS COMPOSITION

Dynamics composition is a core capability of the META Design Flow tools. It is an extreme method of evaluating a system under a specified set of conditions. The system and specification of simulation conditions are defined in a test bench. An example of a dynamics test bench is shown in the figure below.

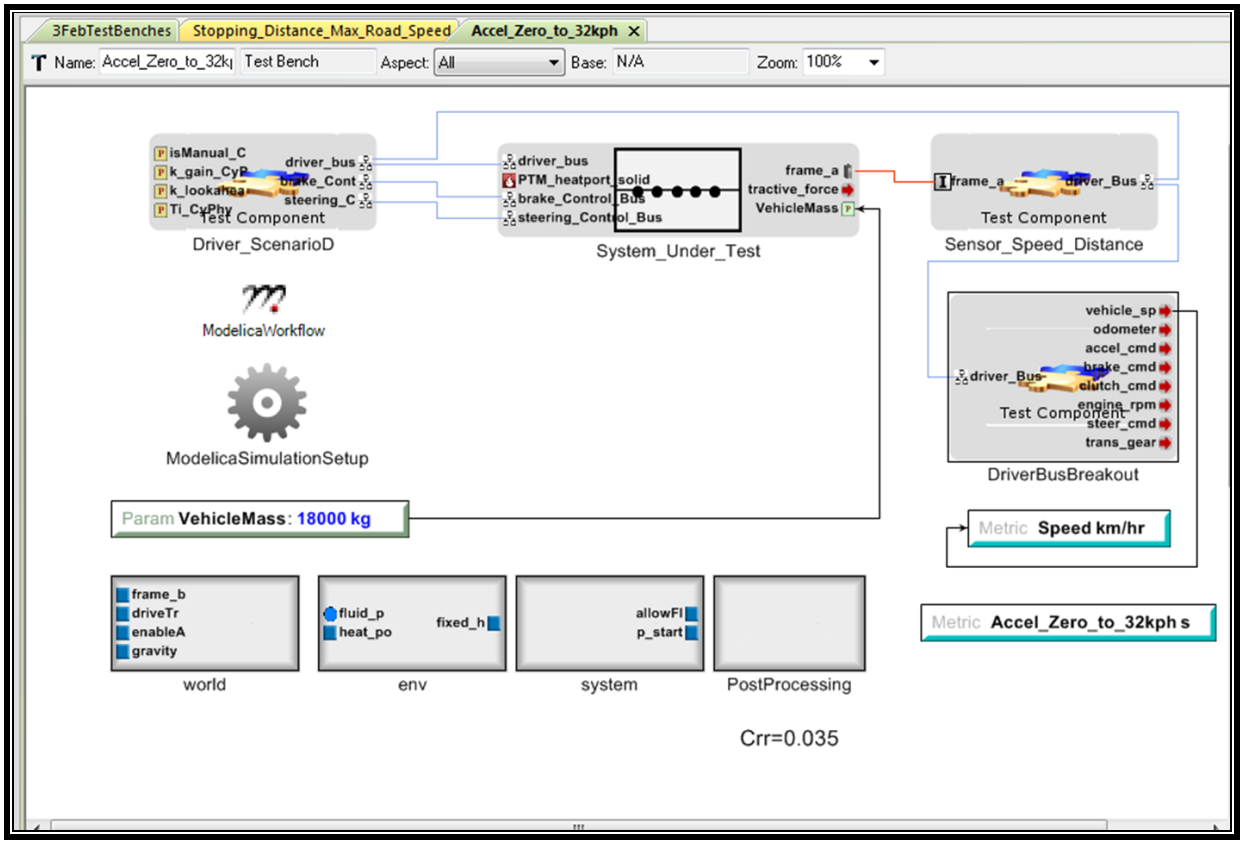


FIGURE 14: EXAMPLE DYNAMICS TEST BENCH

The key components of the dynamics test bench are:

- The SUT is typically at the center of the model. It describes a design or design space model, with any parameters that can control the system, exposed as ports for manipulation by the test bench or external tool. In the example above, the system under test is a powertrain/suspension subsystem. The mass of the other parts of the system (Hull, weapons, stores, etc.) are passed in as a VehicleMass parameter.
- The scenario controls the test (in this case, max acceleration from 0 to 32 kilometers per hour). Driver_ScenaroD provides a simulated driver to set the target speed, control brakes, and control transmission mode. For any test bench, it is the responsibility of the test scenario to excite the system into the desired state.
- Post-processing services monitor the outputs of the system under test and computes metrics from the inputs and outputs of the system. In this example, Speed Sensor Distance calculates the distance the system has travelled, and DriverBusBreakout extracts signals from a control bus.
- Metrics are the outputs of the test bench, the purpose for executing the simulation. These can be tied to requirements specs, which state the threshold and objective values of the metrics. For FANG, the requirement management is done outside META, but the metrics are

coordinated: computed in META and accumulated externally in the GT/ASDL MAUF scoring function.

- All limits specified in the CyPhy model are also checked as a post-processing task. Limit conformance or exceptions are noted in the results file.

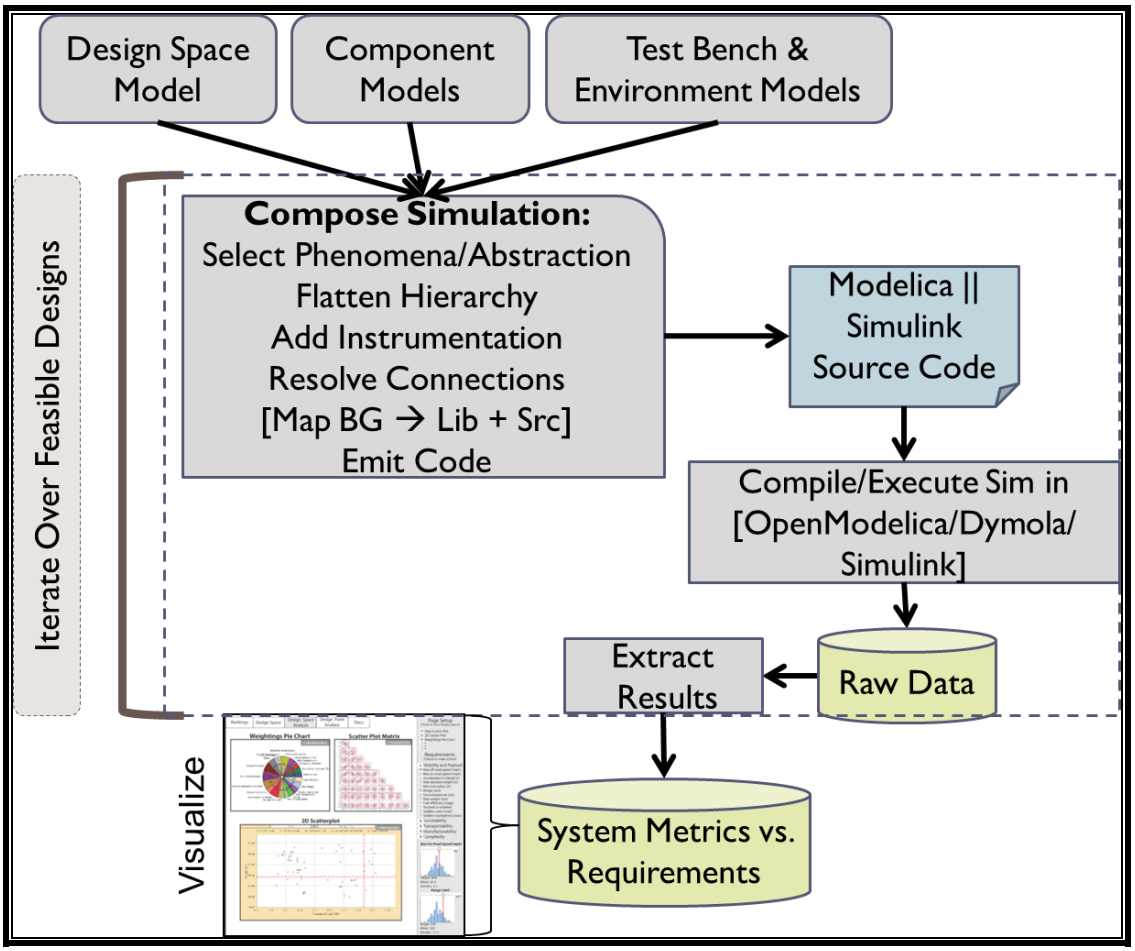


FIGURE 15: COMPOSITION OF SIMULATIONS AND CALCULATION OF METRICS

4.6.1 BOND GRAPHS AND SIMULINK TARGET

Control design involves two distinct paradigms: the discrete specification of the controller and continuous processes governed by the laws of physics. While a discrete controller can naturally be modeled as signal flows, the key to modeling physics is the use of an acausal modeling framework [13]. Using causal models (e.g., signal data flows) to represent interactions between components that share physical variables can be complex. Typically, acausal physics models have power ports, which represent a simultaneous, bidirectional energy exchange between components [10] [8]. A well-formed model in an acausal framework represents a well-formed set of dynamic equations. Acausal models typically must interface with causal models to represent the integration of a controller function into a physical system. This requires carefully directed variable sharing between cyber and physical system

components (e.g., through sensors and actuators). This is one of the key issues of this paper. In the following, we discuss the two most important acausal modeling paradigms.

4.6.2 HYBRID BOND GRAPH MODELING

Bond Graphs [8] are a physics-based, domain-independent graphical notation for describing the behavior of components and systems which can be modeled using differential algebraic equations. Bond graphs generically model the energy exchange between different types of energy storage and con- version components, analogously to a circuit diagram in the electrical domain. Bond graphs are composed of the following primitive elements: source of effort (Se), source of flow (Sf), resistor (R), capacitor (C), inertia (I), transformer (TF), gyrator (GY), one-junction (1), and zero-junction (0). These primitive elements are connected through junctions, which correspond to either common flow (one-junction) or common effort (zero-junction). For example, in electrical circuits, one-junctions (common flow) represent series connections and zero-junctions (common effort) represent parallel connections. The connections between the primitive Bond Graph elements and the junctions are called bonds, each of which represents an effort and a flow variable. The product of the effort and flow variables is the power flowing between the connected elements.

In our previous work, we have extended Bond Graphs in multiple ways to include modulated elements, domain- specific power ports, and hierarchical modeling support [8]. Domain-specific power ports (e.g., electrical power port) connect quantities in one component with another, and each includes two variables: a domain specific effort (e.g., Voltage) and a domain specific flow (e.g. current). Power ports can be connected to either a one-junction or a zero-junction only. Bond Graphs easily and uniformly represent electrical, rotational, translational, thermal, and other types of power domains. Input signals are either control parameters (e.g., Modulate an effort or a flow source) or directly influence the system behavior through functions on the physical variables (i.e., Determine the parameter value of a modulated element). The Hybrid Bond Graph Language (HBGL) includes the ability to resolve causality and create a Simulink model from a Bond Graph model. HBGL also supports domain specific power ports for valid component composition.

4.6.3 MODELICA TARGET

Modelica is a modeling language for dynamic systems that is equation-based and uses signals to express physical constraints imposed by physical connections in the system [10] [2]. Modelica is an object-oriented mathematical modeling approach to systems modeling. The building blocks of the models are stereotyped classes, of which the most important constructs are models, blocks, and connectors. Models can describe hybrid models, which are composed of discrete and continuous variables. Blocks are similar to models with a restriction that they can only expose those connectors that are tagged as input or output. Connectors are ports representing causal/acausal signal variables. The behavior of the building blocks is defined by equations. Modelica does not strive for the uniformity of representation that Bond Graphs provide, but provides a library of standard components for each physical modeling domain called Modelica Standard Library (MSL). Also, Modelica simplifies connecting physical variables by its interconnection model. Interconnections

among components are made using connections (i.e., Connect statements) between connectors, which directly represent physical connections (e.g., Attaching a wire to a pin of an electronic device), enabling the compositional definition of system behaviors. Each connector that represents a physical interface has the same number of flow and potential variables. For instance, an electrical pin connector has voltage (potential) and current (flow) variable. For a well- formed model, Modelica compilers translate all of the model subsystems and connections into equations suitable for simulation or analysis. Unlike Bond Graphs, the Modelica language is an international standard that has well-supported commercial tools. Modelica is an open-source language and has some level of open-source compiler support as well as an open-source standard library (MSL).

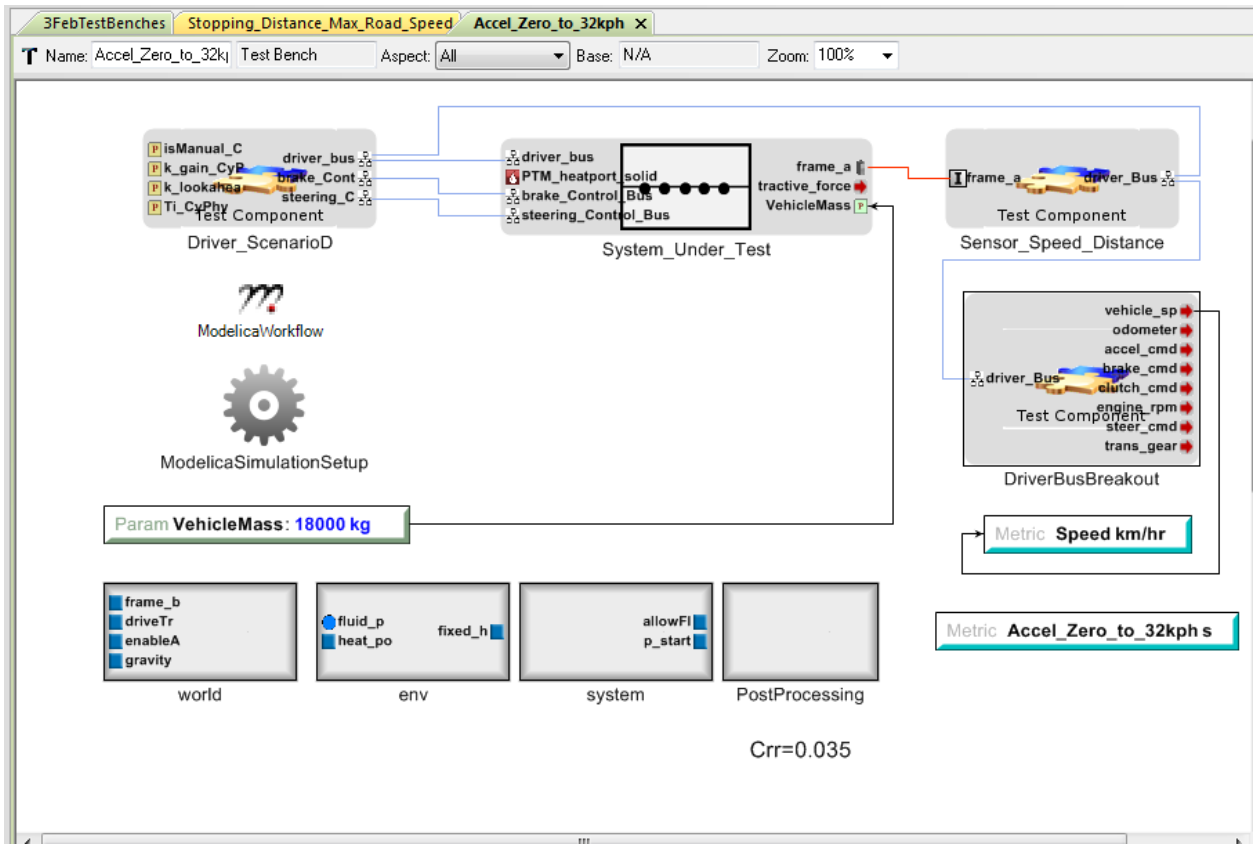


FIGURE 16: EXAMPLE TEST BENCH FOR DYNAMICS EVALUATION

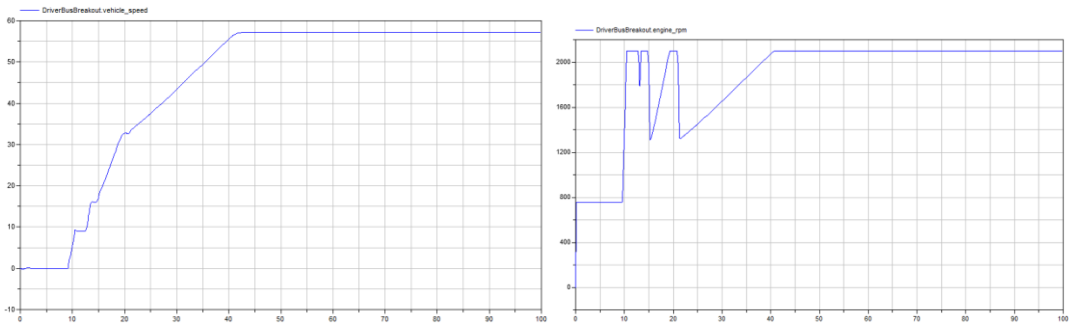


FIGURE 17: EXAMPLE DYNAMICS OUTPUT

4.6.4 OPENMODELICA MATURATION

While the initial goal for FANG use, Dymola was the only available simulator that was capable of executing C2M2L models. Mid-contract, an effort was started to expand the maturation of the OpenModelica compiler/simulator. The full report on this effort is in the META Language report, but the primary accomplishments are shown here for reference:

- Achieved simulation of more than 90 percent of MSL 3.2.1 example models.
- Achieved flattening of whole MSL 3.2.1 library including the Fluid library.
- Achieved simulation of more than 70 percent of the AVM test cases.
- Achieved much more efficient simulation compared to OpenModelica 1.8.1.
- Fluid flattening achieved.
- More than 90 percent MSL 3.2.1 example models achieved.
- Significantly improved simulation performance.

4.6.5 CYBER/CONTROLLER COMPOSITION

The CyPhy language and design flow supports co-design of physical and controller systems. While controller design software is commonly available, the META design flow offers the ability to produce high fidelity simulations of the physical system, in addition to high-fidelity computer hardware/software simulations. This co-simulation allows a much higher level of developmental testing and software/system interaction discovery over a design-to-spec approach common in many software approaches. Typically, high-fidelity testing cannot occur until brassboard hardware and/or physical test rig is available.

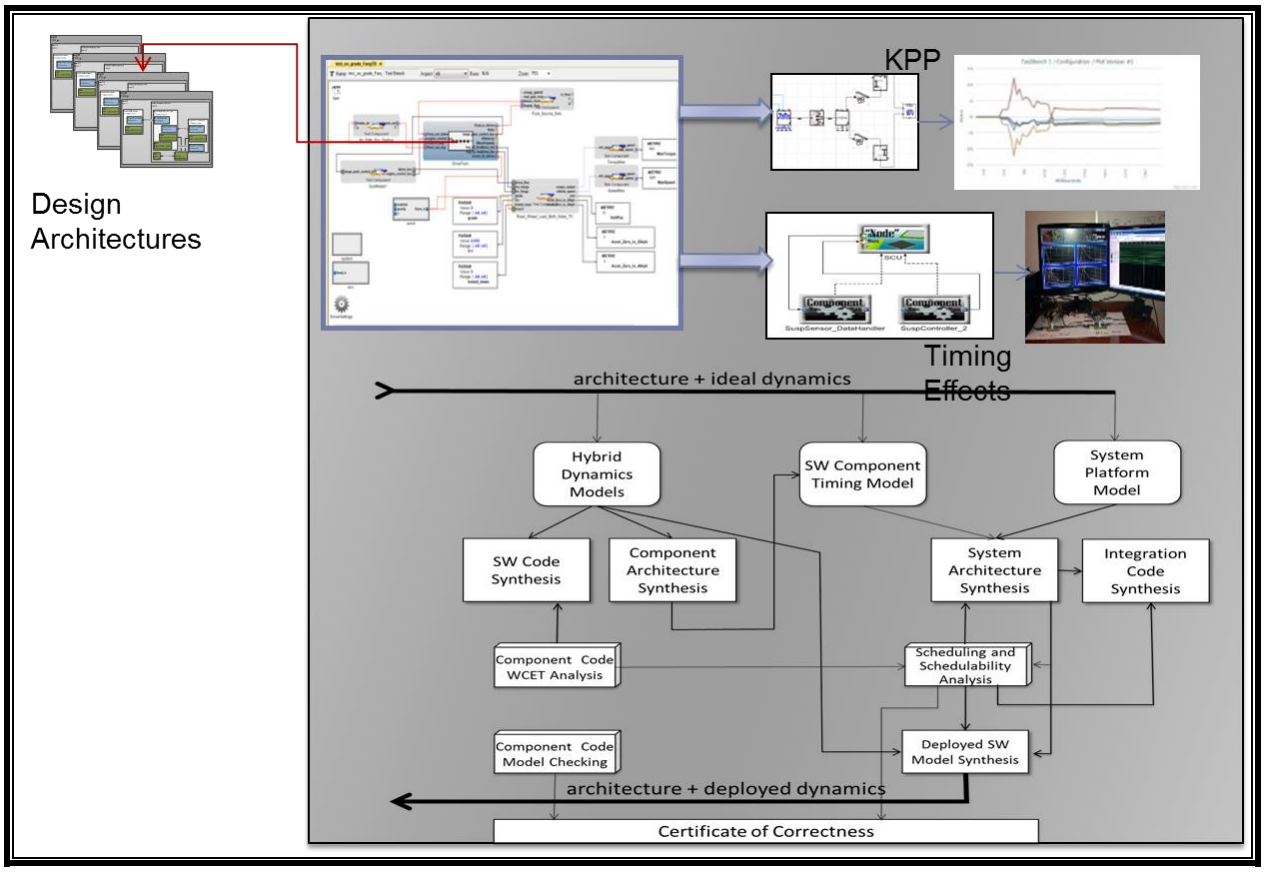


FIGURE 18: DESIGN FLOW FOR CYBER/CONTROLLERS

The figure below shows the META design flow of a cyber/controller test bench. The specification of a controller is described in the META Design Language, consisting of state-transition diagrams for discrete state controllers, and a signal flow paradigm for continuous signal control (e.g., PID controllers).

Cyber analysis uses the same test bench structure as a Dynamics test bench, and is invoked automatically for any system that contains a cyber-controller component. The design flow is shown in the figure above.

Cyber models are extracted (Hybrid Dynamics Models) and synthesized into executable code. Worst case execution time is computed for use in scheduling and schedulability analysis.

Synthesized software can be integrated with a Modelica dynamics model for behavioral checkout. Under this mode, software timing allocations based on aggregate WCET's and system dynamics needs to execute at an idealized sample rate. Under this mode, the algorithm and system dynamics can be tested, assuming idealized real-time scheduling and communication.

The next level of abstraction uses the system platform model to include compute resource limitations, real-time scheduling effects, and communications latencies. The deeper abstractions models can be simulated using the TrueTime simulator coupled with Modelica dynamics model to validate behavior prior to a hardware build. These model are also planned to be used with verification techniques

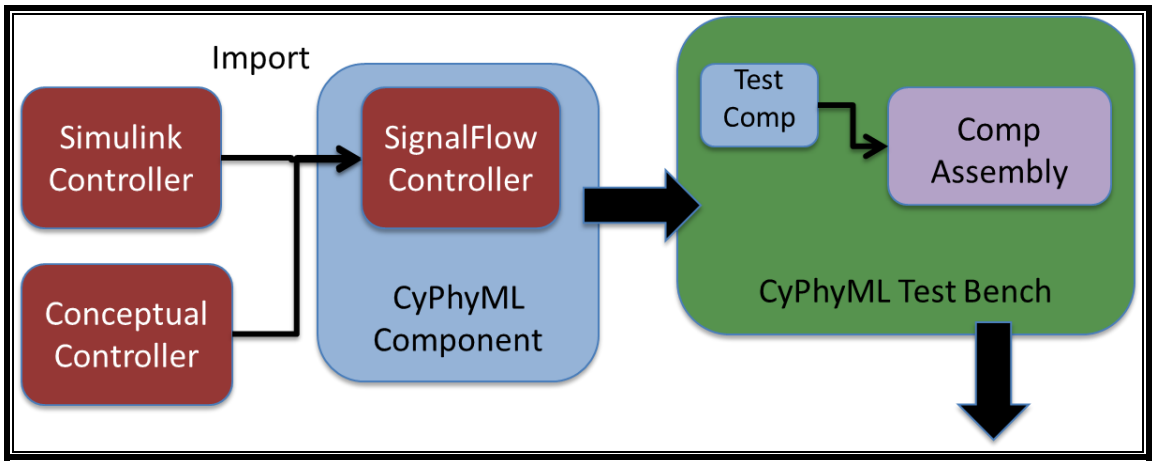


FIGURE 19: TOOL FLOW FOR CYBER ANALYSIS TEST BENCHES

4.7 CAD COMPOSITION

CAD composition forms a key capability in composing the 3 dimensional representation of the system that maintains consistency with the dynamics model. In addition, coupled with the design space representation and exploration, the CAD composition allows wide scale evaluation of alternative geometries and physical constraints on system assemblies.

Using the CAD assembly design flows, a user can execute a CAD assembly operation and specify a geometric measurement set and reasoning on that geometry via a test bench. An example test bench is shown below.

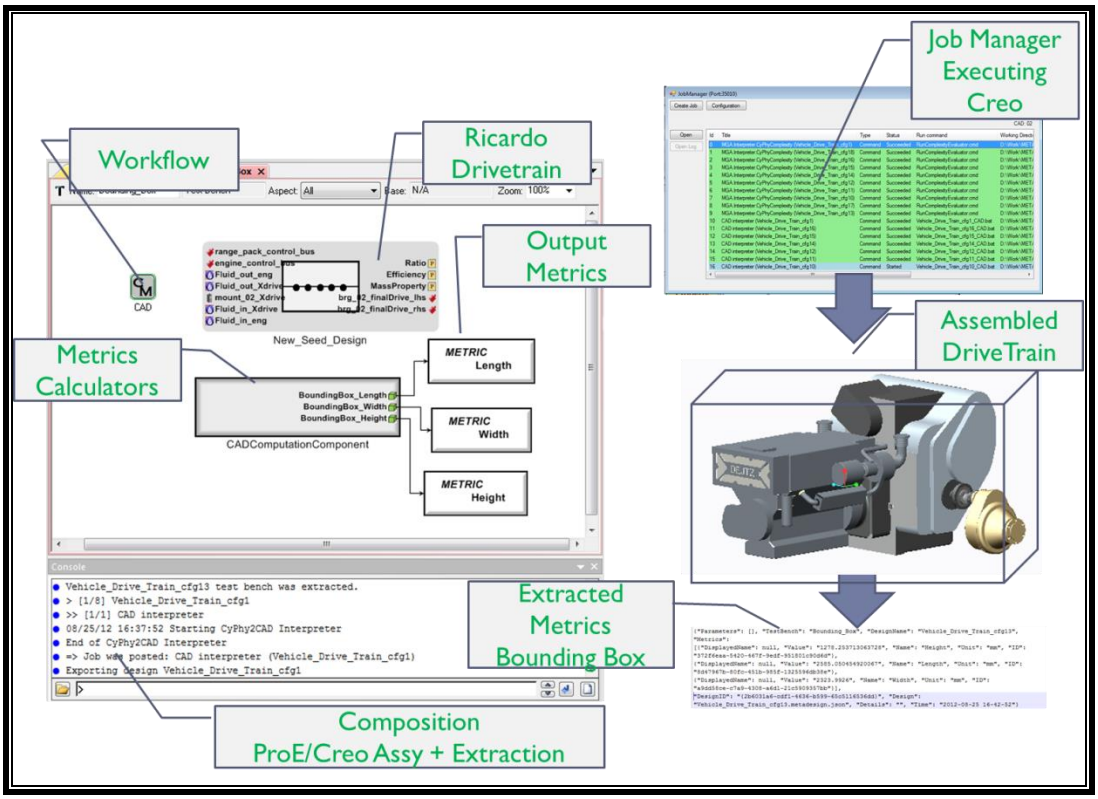


FIGURE 20: EXAMPLE CAD COMPOSITION TEST BENCH

The example test bench references a system design space as the system under test. The CAD workflow in the test bench specifies that a geometric assembly is required. The FANG Drivetrain is the SUT. A CAD Computation Block specifies geometric calculations are required, in this case bounding box length, width, and Height. The test bench below computes the FANG metric for Well Deck Transportability:

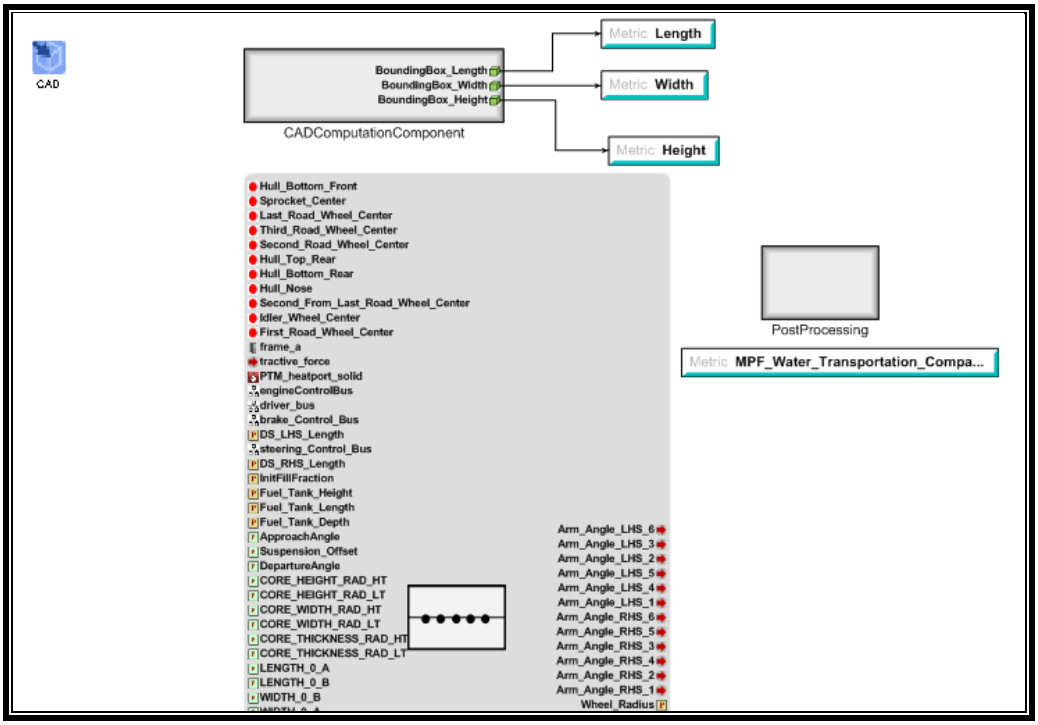


FIGURE 21: WELL DECK TRANSPORTABILITY REQUIREMENT EVALUATION

As an intermediate result, the test bench generates a 3-D CAD model in a variety of formats. The highest level of detail and model content retention is with the ProE/Creo output options. Figure 22 shows the resulting CAD model of the FANG seed design.

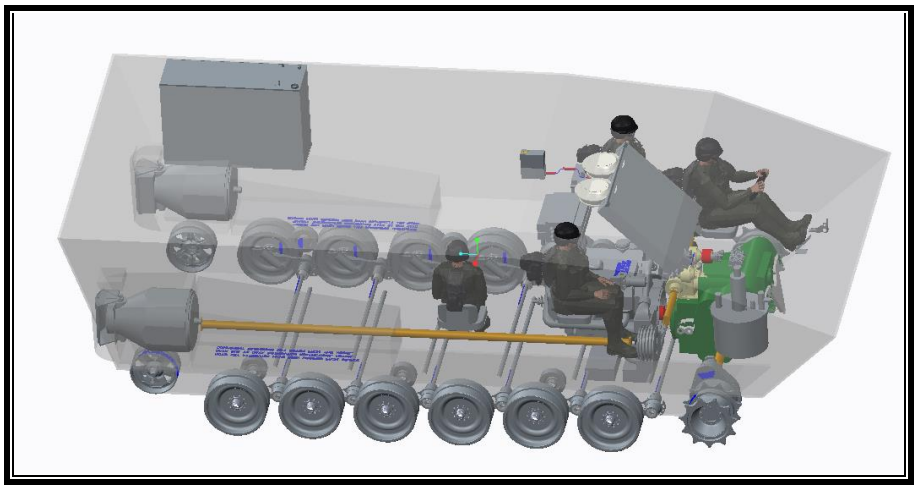


FIGURE 22: CAD ASSEMBLY RESULTS

Note that CAD composition requires all interfaces to be compatible between attached components. For the purposes of design space exploration, a wild-card and adapter capability was created. These allow a character-by-character matching relaxation, to support selective matching. The above model includes several adapters, (e.g., PTM to Hydraulic Pump) which appear as red cylinders.

The CAD composition uses CyPhy structural interfaces to associate connections. The CyPhy structural interfaces reference datum within the CAD file (planes, Axes, Points, Coordinate Systems). Within the CAD composition, algorithms are encoded to break constraint loops, detect islands, and support under-constrained joints. CAD tool drivers enforce constraints between these datum to enable PTC Creo to properly position parts and assemblies.

4.8 FEA COMPOSITION

FEA composition design flow is shown in the figure below. The testbench contains the system under test (which can be the entire system, or a part (e.g., Suspension A-Arm). For FEA, the test bench language allows the system under test to expose geometric handles: points, areas, and volumes. The test bench operators are (for structural) forcing functions and constraints. Forcing functions can be applied to any of the geometric handles, and can represent the force of gravity on an attachment point, the forces of a weapon firing, etc. Constraints also apply to the handles and will be applied at the physical geometry referenced by the handles.

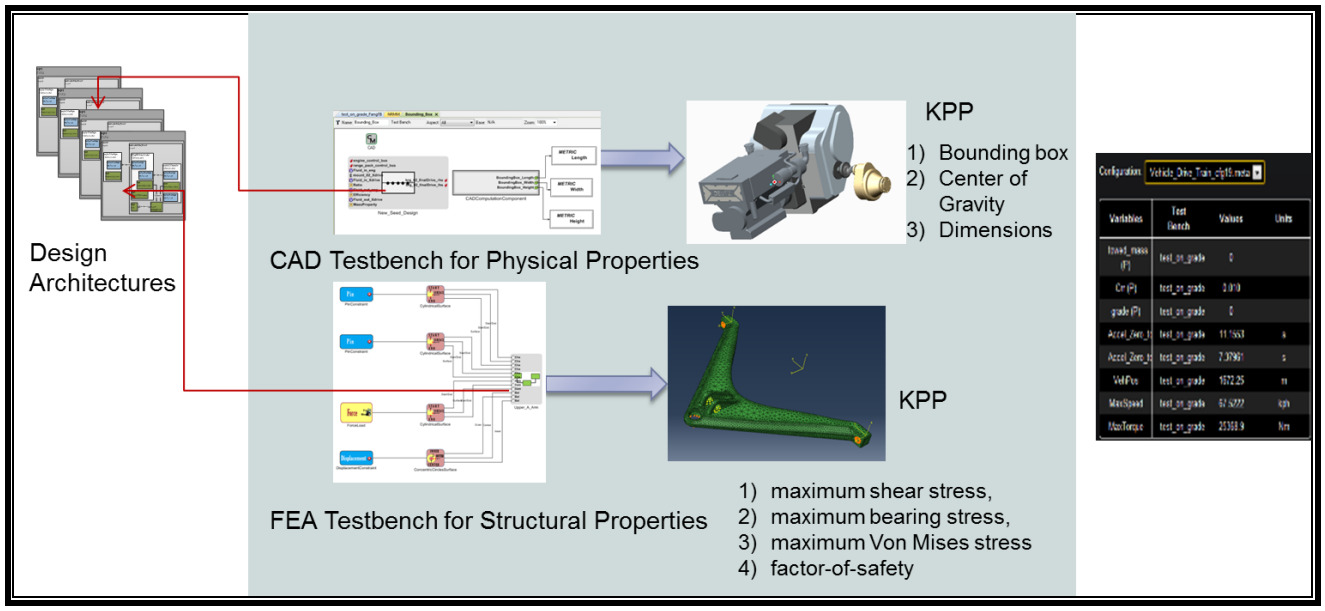


FIGURE 23: TEST BENCH CONCEPT FOR FEA

The figure above shows a single forcing function (Yellow) applied to a surface of the A-Arm, with 3 constraints holding the arm at its attachment points. The computation will calculate maximum shear stress, maximum bearing stress, maximum Von Mises stress, and apply an overall factor-of-safety to the assembly based on the materials properties of the component.

The basic META Design flow is shown in the figure below.

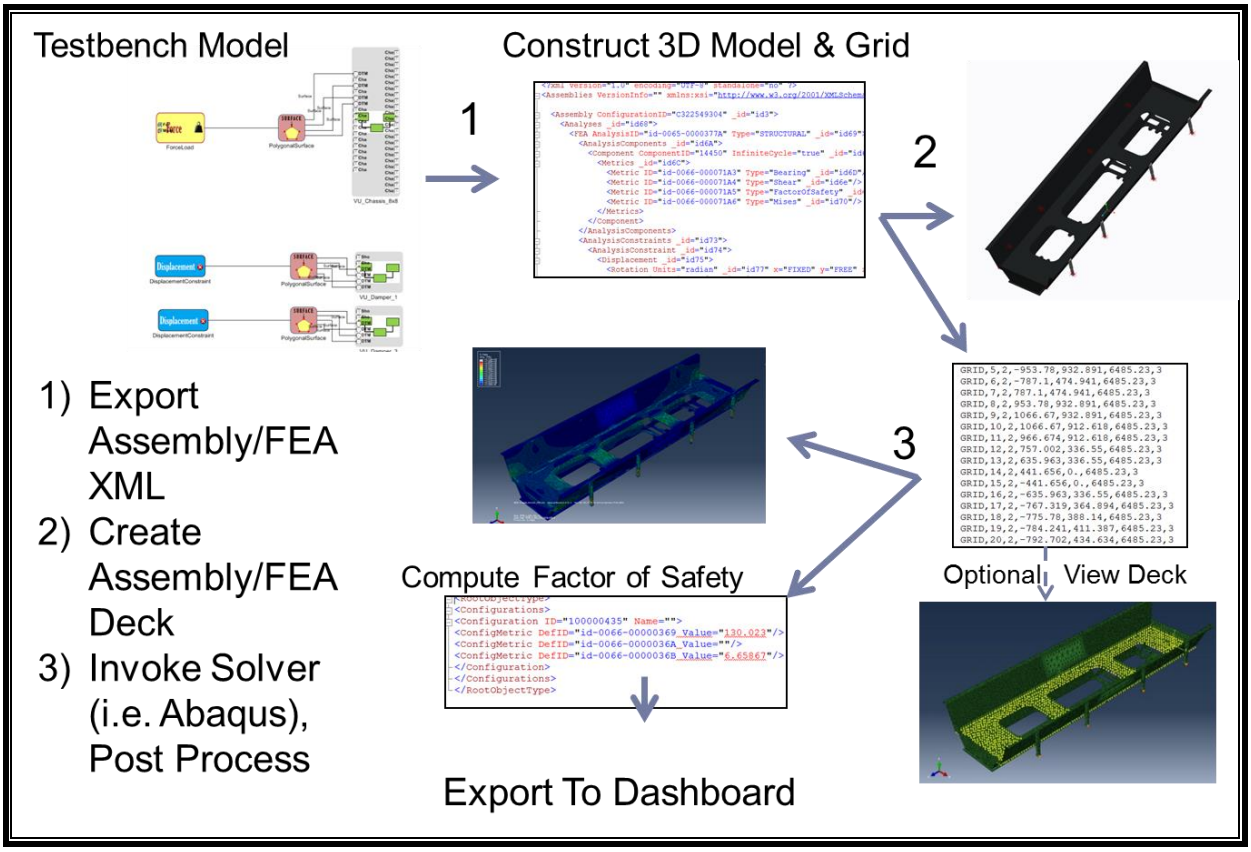


FIGURE 24: INFORMATION FLOW FROM FEA TEST BENCH

The basic steps are:

- A META CAD assembly operation occurs, using the CAD assembly tool.
- The META Tools use the CAD tool to create a grid of the system under test.
- Based on the test bench forcing functions and constraint locations, the grid objects are located geometrically by the META Tools.
- Constraints and forcing functions are applied to the grid objects (surfaces, points). The grid deck is modified with these annotated objects by the META tools.
- The FEA tool is called on the modified deck to compute stresses, and the result files are generated.
- The META tools post process the results files, extracting the requested metrics. Metrics are stored in an AVM compatible results file.

FEA Stress analysis has been the primary focus of the META FEA tools, and statics are the most developed tool.

Thermal analysis experiments have also been done, determining the heat profile for an engine-transmission model.

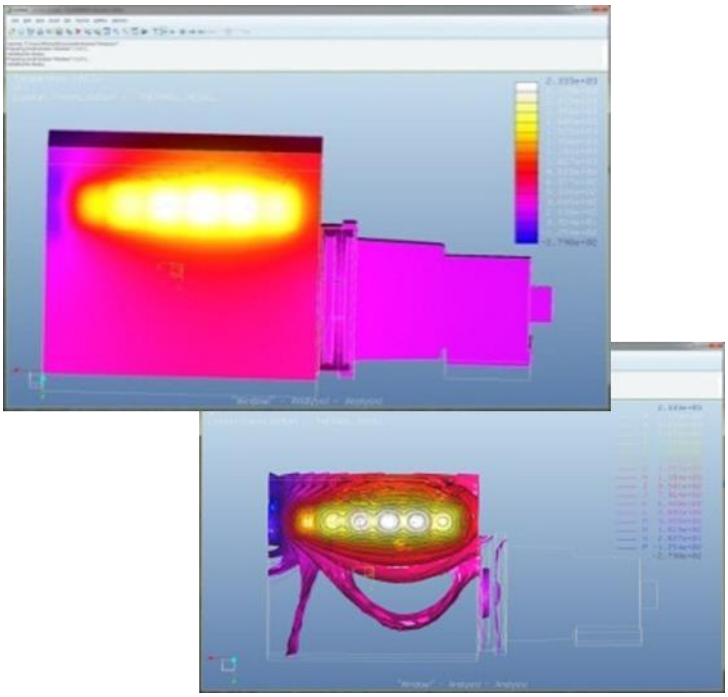


FIGURE 25: EXAMPLE THERMAL ANALYSIS

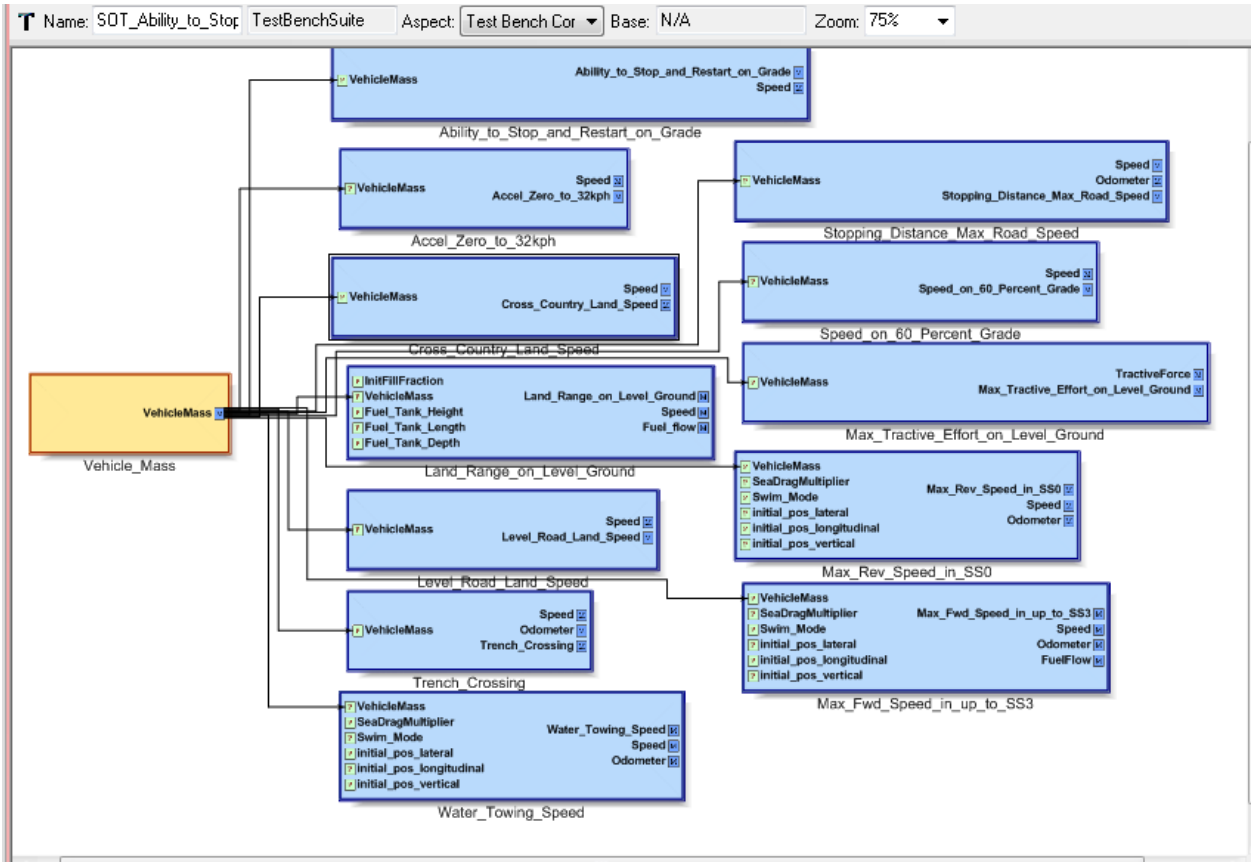
4.9 STATIC CALCULATIONS WITH CYPHYPYTHON

A python-based facility allows simple calculations to be implemented. The CyPhyPython tool allows a python code to traverse the model, accessing models, connections, attributes, or other parts of the model. These can calculate results based on the structure, or can modify the model itself.

For FANG, test benches were implemented to calculate nominal vehicle weight, and special-purpose completeness metric.

4.10 SUITE OF TEST BENCHES

The Suite of Test Benches supports automation of composite analyses. For instance, a static test bench using the CyPhyPython facility calculating mass can drive the dynamics test benches for acceleration, speed, etc. This example is shown in the figure below.



4.11 EXECUTION INFRASTRUCTURE

The META tool flow can support automatic composition of a large number of test benches, across a large number of design alternatives.

While powerful, and labor saving in terms creating the executable analyses, large computational tasks can be created. These tasks can easily overwhelm an engineering workstation. Additionally, manually managing these job runs and organizing the results can be a difficult task.

To address these needs, the Job Manager was developed. The job manager has several functions:

- Interface with the Master Interpreter to receive tasks as they are composed.
- Manage the tasks received, keeping track of their state and displaying that state to the user
- Launching tasks to the compute resources
    - Local resources: starting tasks that can leverage all the processors and hyperthreads on the local workstation (Typical laptops can execute 8 simultaneous tasks with little loss in performance)
    - Remote resources:
        - negotiate with a remote job server (e.g., Jenkins META Compute service deployed on any compute farm or cloud),
        - create jobs and upload all necessary information required for that job,

- Start jobs
- Monitor progress
- Download job results and maintain results in the proper locations for other tools (Visualization and results analysis)
- Collect and visualize the state of the job servers.

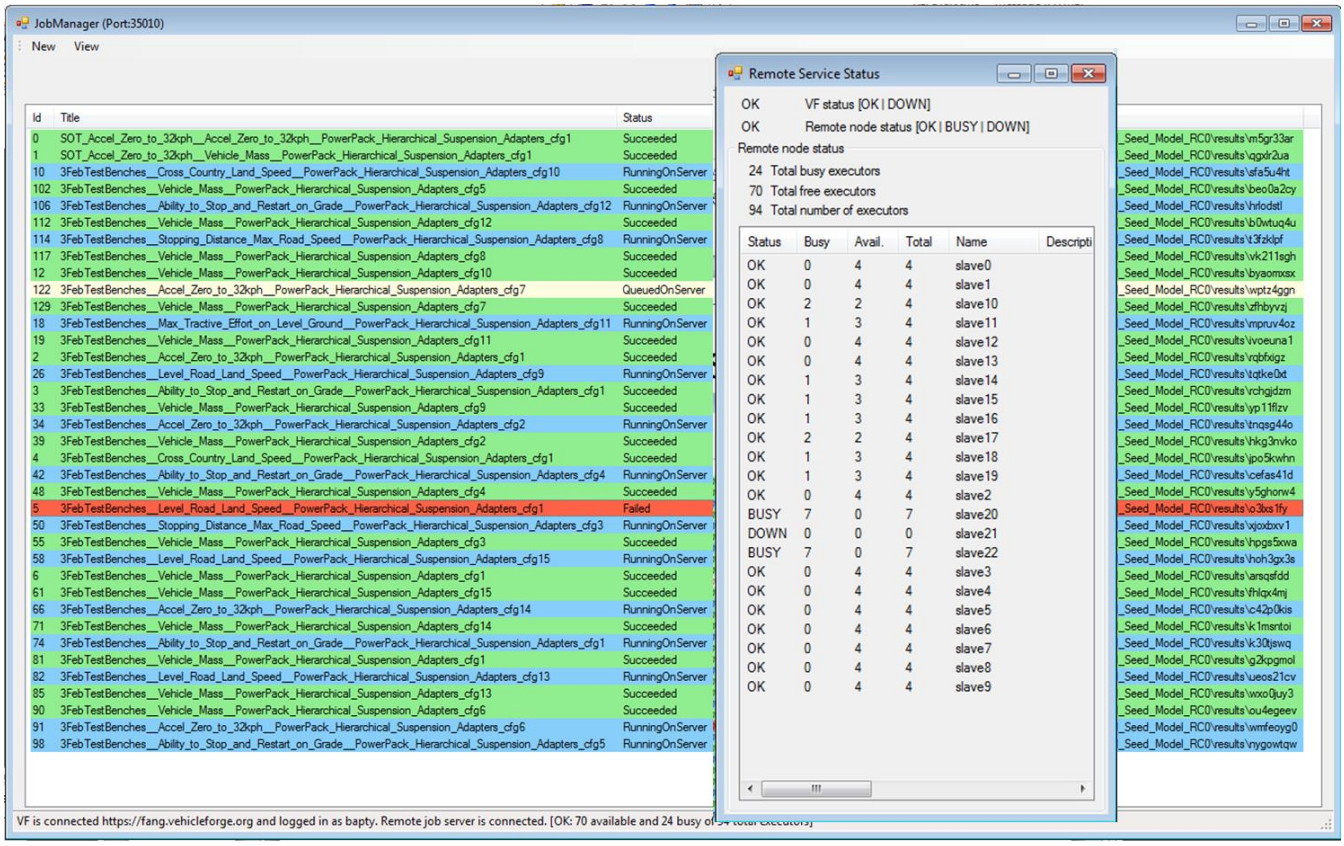


FIGURE 26: META DESIGN FLOW JOB MANAGER

The job manager UI is shown above, with a list of jobs (Green = succeeded, Blue = in progress, Red = Failed)

4.12 VISUALIZATION METHODS

As a result of the automation to analyze multiple metrics across large design spaces, a large amount of data can be generated across many different designs. The META dashboard has been designed and implemented to help understand the analysis results and the span of the design spaces.

The full report for the dashboard is detailed in the subcontractor report from Georgia Tech ASDL.

In summary, the dashboard consumes all the metrics from all designs and visualizes these results to help locate the best designs. Several key plots are supported.

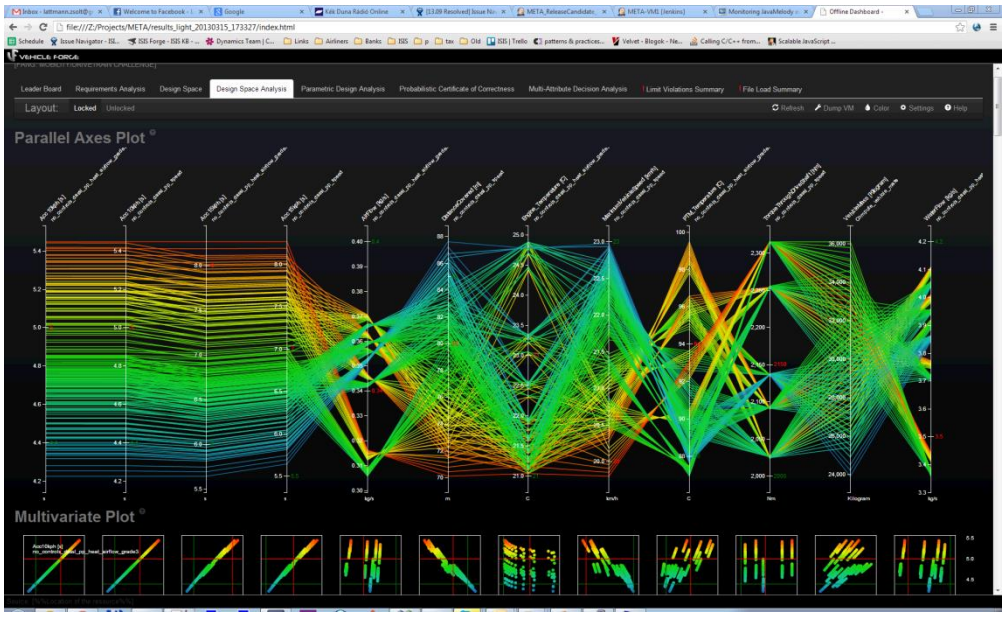


FIGURE 27: PARALLEL AXIS PLOT, COLORS BY RANK

The Parallel Axis Plot shows individual designs as a line that traverses horizontally across a series of metrics. These lines can be colorized by the weighted rank, set in another panel.

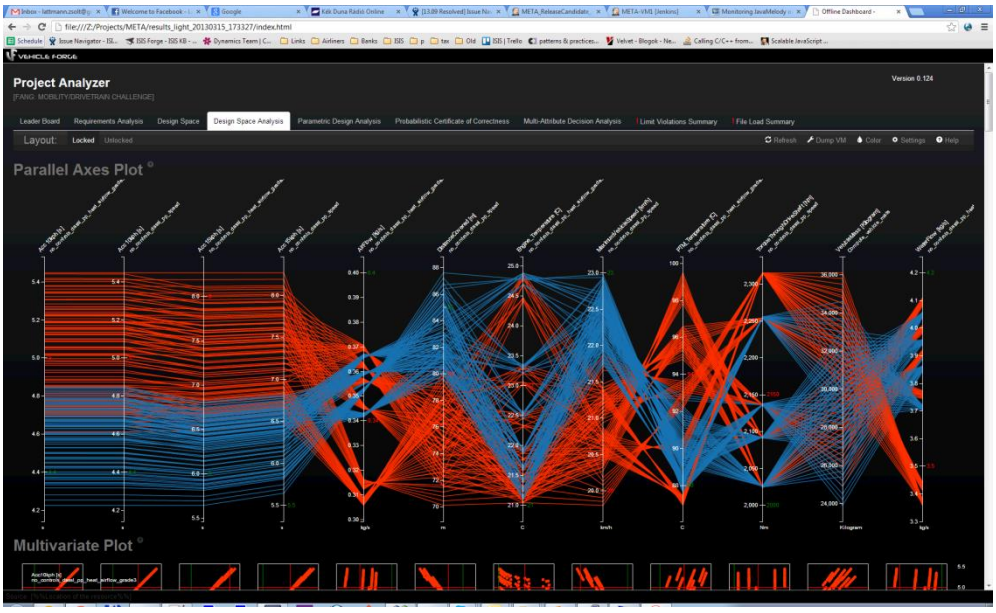


FIGURE 28: PARALLEL AXIS PLOT, RED=LIMIT EXCEEDED

The same plot can be visualized, with design axes colored Red where limits were exceeded during dynamics simulations

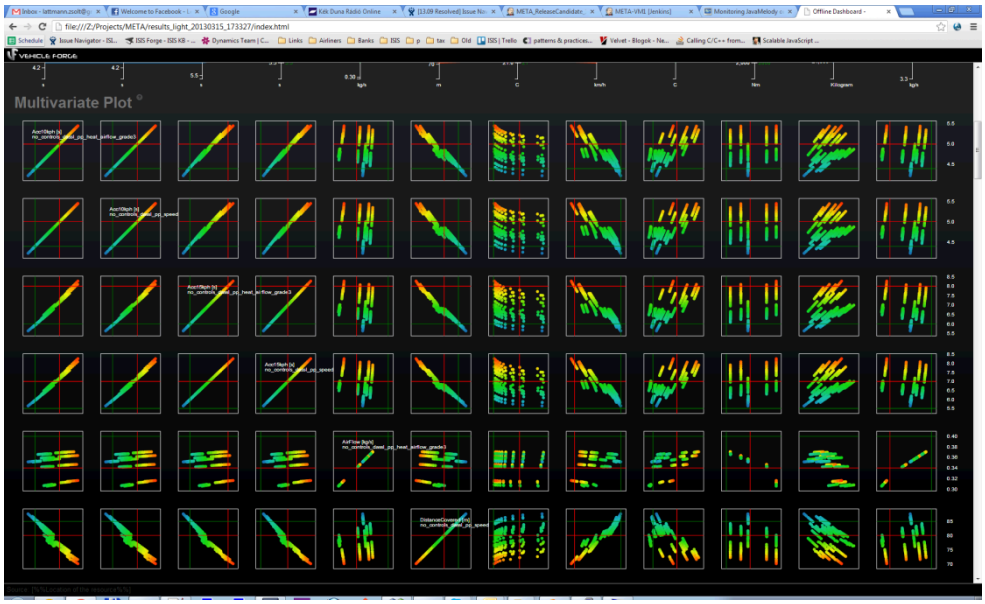


FIGURE 29: PAIR-SIZE METRICS PLOTS

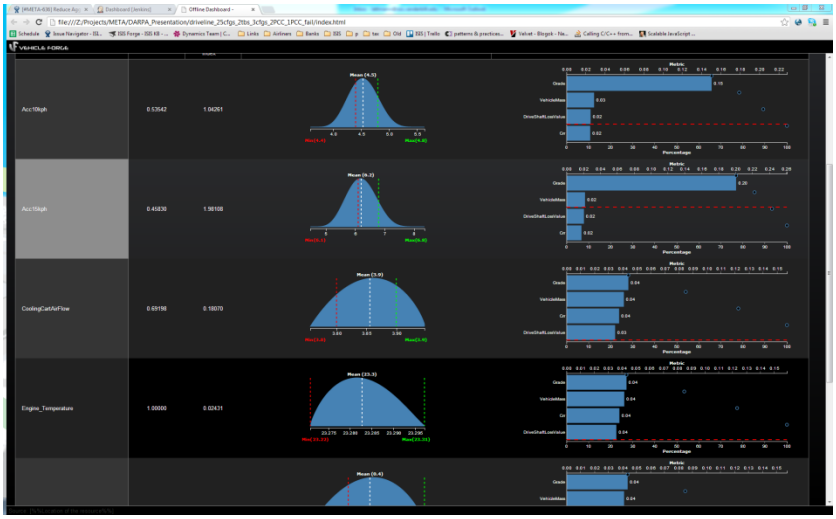


FIGURE 30: DASHBOARD VISUALIZATION OF PCC RESULTS

4.13 COMPLEXITY METRICS

Complexity metrics are also calculated via a test bench. The test bench is shown in Figure 31.

Technical details for the Complexity metrics are shown in Appendix B.

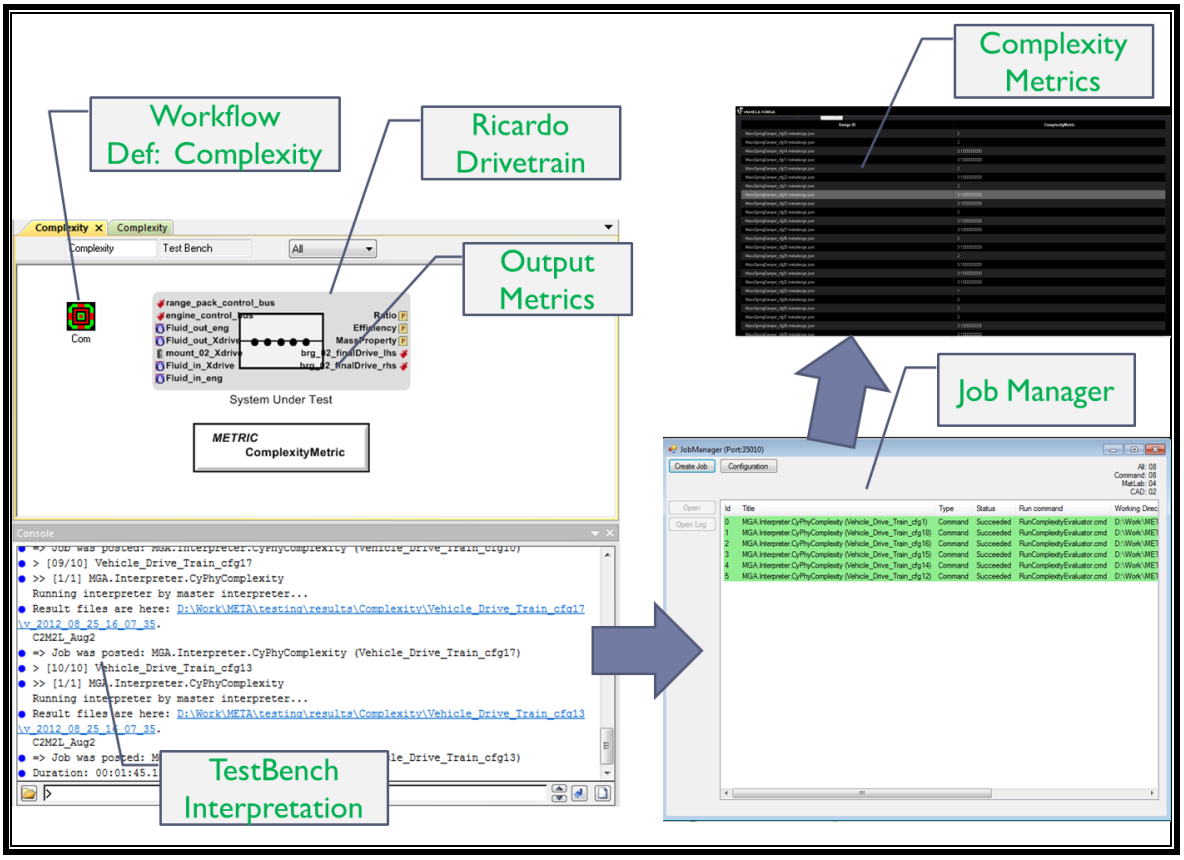


Figure 31: Complexity Test Bench

4.14 VERIFICATION METHODS

The correct-by-construction of the META design process is the key to the AVM approach. Verification techniques are integrated with the CyPhy/OpenMETA system. Currently, the primary method is a simulation-based, probabilistic certificate of correctness.

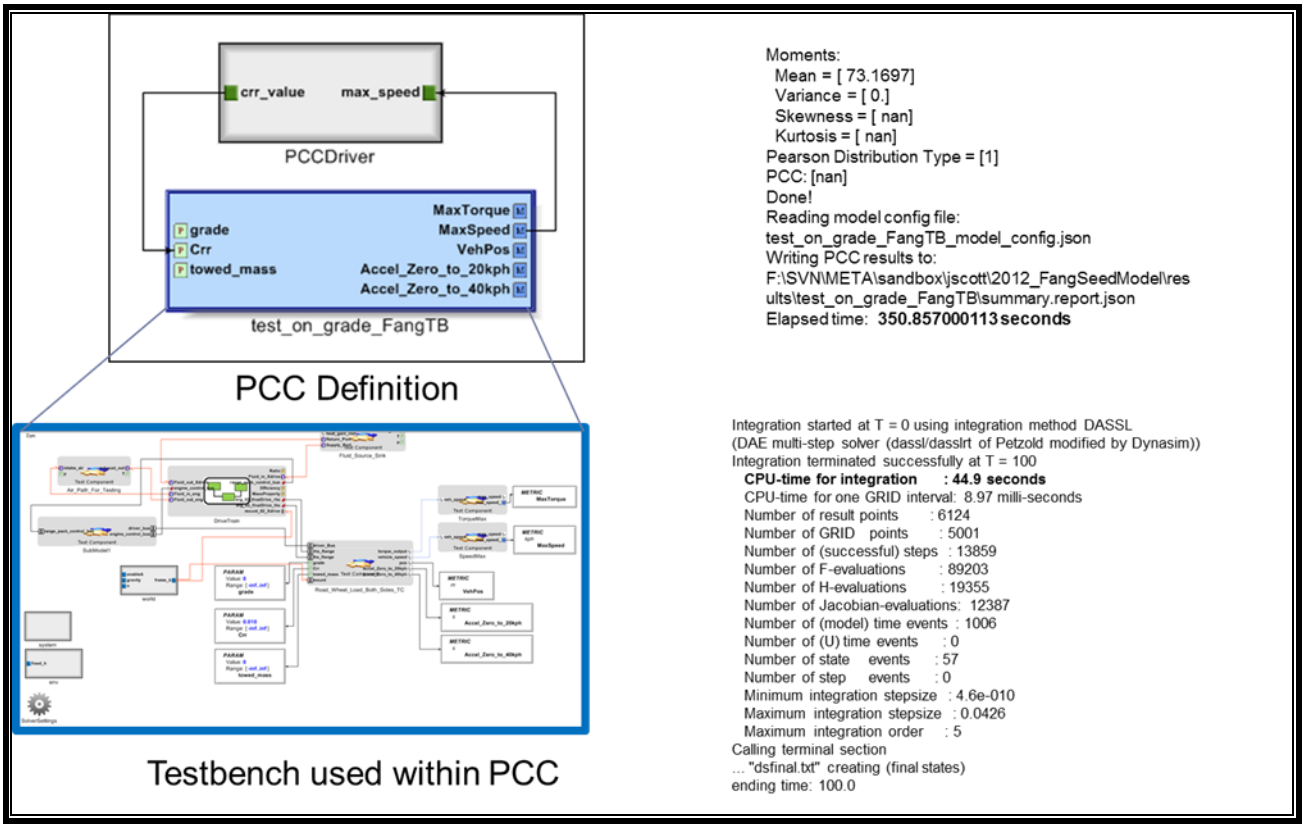


FIGURE 31: EXAMPLE PROBABILISTIC CERTIFICATE OF CORRECTNESS

The Probabilistic Certificate of Correctness, or PCC, is configured by the model shown in the figure above. The PCC model builds upon a testbench, typically of the dynamics type. The parameters of a test bench will map to a system variable, such as environmental or component property that has manufacturing variations. The PCC calculation will modify all specified parameters while doing a statistical analysis of input vs. output metrics. The results of all experiments can be combined to compute parametric sensitivities and an overall probability that system metrics will stay within the allowable ranges. PCC uses Monte Carlo techniques, as well as more sophisticated methods to reduce the required number of samples.

A related tool, the Parametric Exploration Tool (PET), allows the designer to explore a range of numbers to help find acceptable values of adjustable parameters. Using Design of Experiment techniques, CyPhy can help to find good values of these parameters for a single architecture.

Qualitative Reasoning

See appendix B for QR.

5 RESULTS AND DISCUSSIONS

5.1 EXECUTION THREADS FOR META DESIGN FLOW AND FANG

META Tools form the core of activities in the FANG 1-3 competitions. The figure below shows the relationships between META and other AVM primary activities.

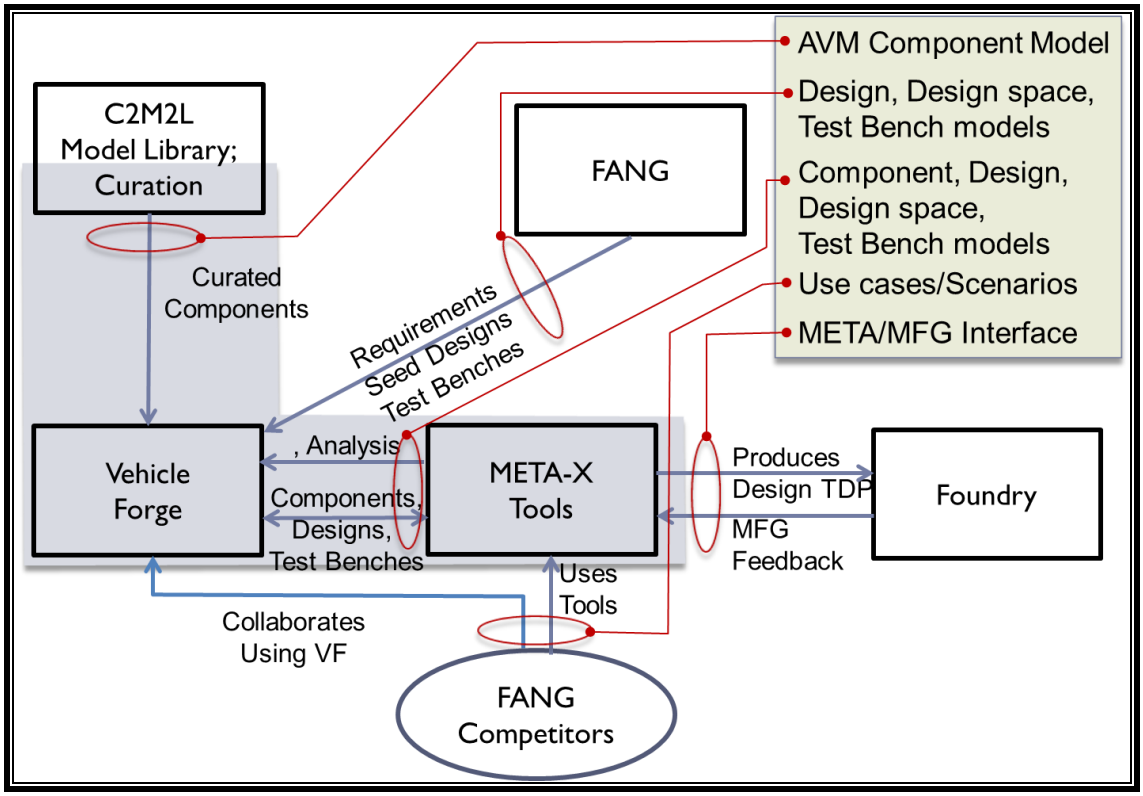


FIGURE 32: META TOOL INTERACTIONS WITHIN FANG COMPETITION FLOW

The Competitor is the focal point, and primary target for support, who executes the competition activities. The competitor interacts directly with two of the primary entities:

- Vehicleforge is the host web site for secure collaboration, hosting facilities for issue reporting tickets (similar to TRAC/Redmine/JIRA), forums for collaboration, repositories (SVN/GIT), and a repository to allow searching and downloading of components. Other FANG-specific services include hosting a design scoring function and a gateway to the iFAB servers. VF also hosts cloud resource used by META compute servers.
- META tools implement the design flow, described in the vignettes and threads below, providing the capabilities described in this report.
    - Via META, a designer produces designs and design spaces. These can be shared via the SVN/GIT repositories (hosted on VF) between team members.

- Via META, a designer composes analyses for execution on META compute servers hosted on the VF cloud, also providing access to proprietary/license-locked software (Dymola & ProE) and receives results.
- Results can be visualized locally or on a META visualizer hosted on VF.
- Results can be submitted to the scoring facility.
- META composes queries for Manufacturability Analysis, which is serviced by the iFAB Foundry manufacturability analysis.

Other Entities include:

- C2M2L produces components for integration with META. Components flow thru curation and onto the VF component repository.
- The FANG performer creates the requirements, requirement evaluation specifications, and seed designs, along with documentation and competition rules, guidance, and oversight.

The role of META in FANG is described in more detail in the figure below:

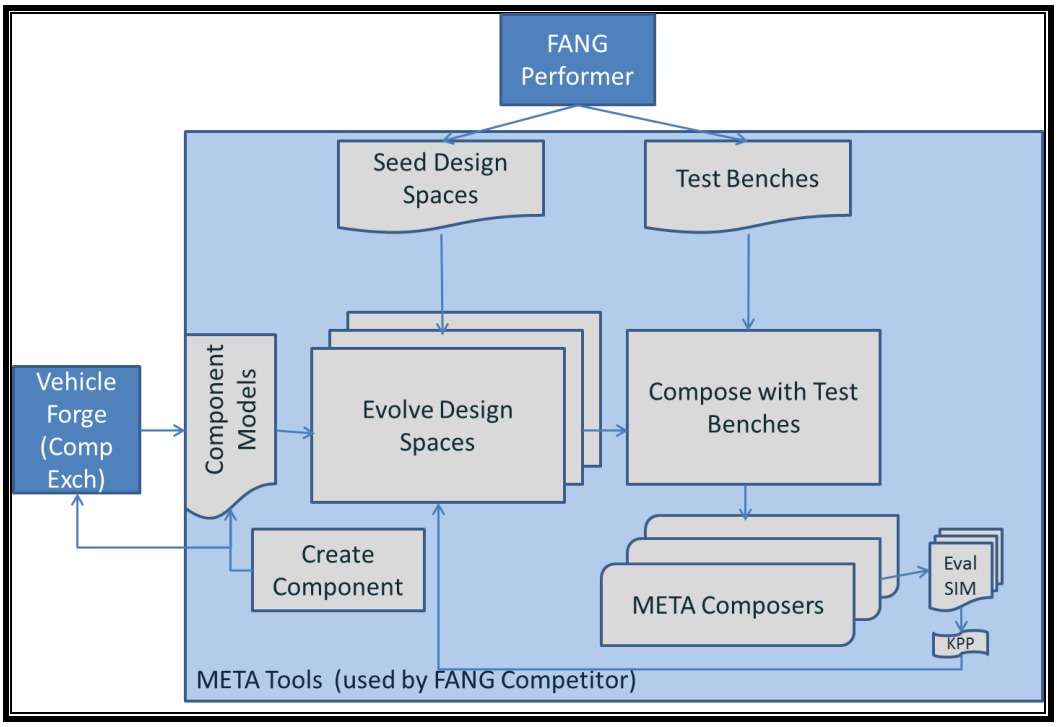


FIGURE 33: DETAILED META-RELATED FANG COMPETITOR ACTIVITIES

5.2 META TOOL CAPABILITY PLANNING VIA EXECUTION THREADS

In preparation for the FANG competition, a set of competitor threads was created to ensure capabilities would be sufficient to support all design activities. The process of allocating capabilities with development activities is described in the figure below.

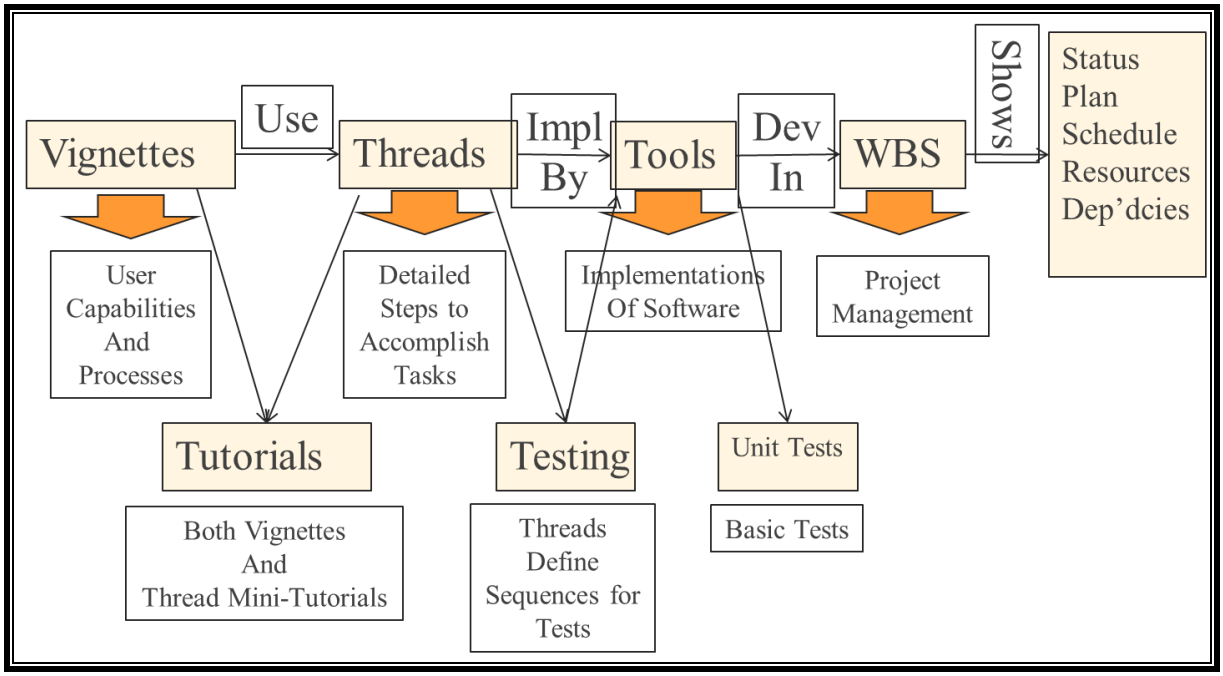


FIGURE 34: CAPABILITY MAPPING PROCESS

Vignettes were created to accomplish the main tasks that the competitors will require for the competition. These are broken down into individual threads that describe step-by-step tasks within the tools. These tasks are mapped to tools and tool capabilities. Tools and tool capabilities are allocated resources and assigned a schedule in the WBS.

The individual products in this chain are also used to define and drive tutorials (Vignettes and threads) and testing (Threads X Tools).

A full set of vignettes were delivered at the Preliminary Design review at Camp Pendleton. An example of a thread is shown below.

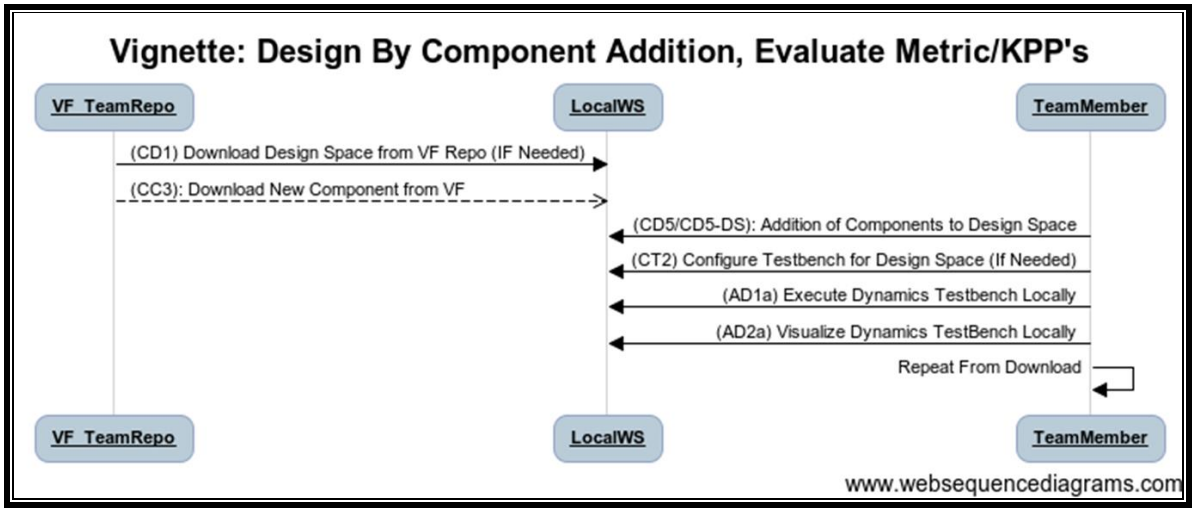


FIGURE 35: EXAMPLE THREAD DIAGRAM

6 CONCLUSIONS & TOOLS COMPLETED

The following tools have been completed, delivered to several targets, and used in Beta Test and the FANG Competition. See the Tool Data Sheets in the attached appendix for more information on each tool.

6.1 DESERT DESIGN SPACE EXPLORATION TOOL

Integrated into the META design flow tool, supporting the full CyPhy language, with user specified constraints and derived constraints (e.g., component limits, structural compatibility)

Delivered on: ALL RELEASES

6.2 MASTER INTERPRETER TOOL

Master interpreter manages the execution of all types of test benches across all selected configurations in a design space, greatly automating an analysis process.

Delivered on: ALL RELEASES Post Aug 2012

6.3 DESIGN SPACE ELABORATOR

Design space elaborator converts a design space with a set of selected configurations to a set of fully elaborated design point models, suitable for test bench composition tools below.

Delivered on: ALL RELEASES

6.4 FIDELITY SELECTOR TOOL

Fidelity Selector Tool allows specification of the fidelity/abstraction of components within a design space and saves settings to the testbench. It also allows configuration of fidelity/abstraction per-test bench.

Delivered on: ALL RELEASES Post December 2012

6.5 JOB MANAGER TOOL

The Job manager executes test benches locally and remotely, and manages result files.

Delivered on: ALL RELEASES Post August 2012.

6.6 DYNAMICS COMPOSITION TOOL (CYPHY2MODELICA)/ (CYPHY2SIMULINK)

The Dynamics Composition Tool takes a dynamics test bench with a design point system under test and composes a simulation job which can be executed locally or remotely. The job consists of scripts

to execute dymola, post-processing of results to create metrics, and general management of the job sequence. An earlier version created Simulink executable jobs.

Delivered on: ALL RELEASES Post March 2012

Uses: MODELICA (OpenModelica, Dymola)

Deprecated (SIMULINK/STATEFLOW)

6.7 CAD COMPOSITION TOOL

Constructs CAD in Creo and STEP, computes geometric properties of the assembled model

Uses: ProE/Creo

6.8 CYBER COMPOSITION AND RUNTIME

Constructs combined cyber controller and dynamics simulation

Delivered: All Releases Post March 2012

Uses: Modelica, TrueTime

6.9 PET TOOL

Executes parametric optimization tasks on dynamics simulation testbenches, running on OpenMDAO.

Delivered: All Releases, Post March 2012

Uses: OpenMDAO

6.10 PCC TOOL

Executes Probabilistic Certification on dynamics test benches, running on OpenMDAO. Supports several OSU algorithms, ranging from a general but inefficient Monte Carlo to an extremely efficient algorithm that computes PCC's for systems obeying certain conditions.

Delivered: All Releases, Post March 2012

Uses: OpenMDAO

6.11 COMPLEXITY TOOL - STRUCTURAL

Computes structural complexity on CyPhy models based on graph Energy (MIT DeWeck).

Delivered: All Releases, Post March 2012

6.12 COMPLEXITY TOOL - UNCERTAINTY

Computes complexity on CyPhy models based on simulation uncertainty. (MIT – Alaire/Wilcox)

Delivered: All Releases, Post March 2012

6.13 COMPLETENESS TOOL

Computes metrics associated with connecting matching structural/power ports, fully connecting structural supports, and minimizing adapters in a design.

Delivered: All Releases Post Feb 2013

Uses: CyPhyPython.

6.14 QR COMPOSITION TOOL AND QR ENVISIONMENT RUNTIME TOOL

QR Tools from PARC were integrated into META Design Flow for simplified component models.

Delivered: All Releases Post Mar 2012

Uses: PARC Quantitative Envisionment Tools.

6.15 RA COMPOSITION AND RA RUNTIME

RA Tools from SRI were integrated into META Design Flow for simplified component models.

Delivered: All Releases Post Mar 2012

Uses: SRI Relational Abstraction Tools.

6.16 SIMVIZ

SimViz is a collaborative visualization tool for analysis and understanding of simulation results.

Delivered: All Releases Post Nov 2012

6.17 DASHBOARD

Dashboard is a design space visualization tool that portrays the range of results from multiple testbenches across a design space.

Delivered: All Releases Post July 2013

Requires: Georgia Tech/ASDL Dashboard Software

APPENDIX A: SUBCONTRACTOR FINAL REPORT - OREGON STATE UNIVERSITY META X REPORT

Oregon State University Meta X Report

For Period Through Sept 30, 2012

Christopher Hoyle
PI Irem Tumer PI

1. Summary

The deliverables during this period consisted of a methodology for computing the Probabilistic Certificate of Correctness (PCC) for a given design, methods for Sensitivity Analysis (SA), and integration of the methods into the Vanderbilt GME tool. The deliverables are summarized as follows:

1.1. Methods for PCC and SA

Eight methods (originally written in Matlab) have been implemented in order to enable PCC and SA to be conducted on an arbitrary system design: Monte Carlo Simulation (MCS), Taylor Series Method (TSM), Most Probable Point (MPP), Full Factorial Numerical Integration (FFNI), and Univariate Dimension Reduction (UDR) as Uncertainty Propagation methods (for PCC computation), and Sobol Method (SOBOL), Fourier Amplitude Sensitivity Test (FAST), and Extended Fourier Amplitude Sensitivity Test (EFAST) as Sensitivity Analysis methods. Simple test models and results published by other researchers studying these methods were utilized to verify correct implementation of the methods. These methods are described in more detail in Section 2.

1.2. Conversion to Python and Implementation as an OpenMDAO Driver

The eight PCC and SA methods have been converted from Matlab to Python (2.7) in order to comply with the open source requirements of the project and enable integration into the Vanderbilt GME. Simple test models and results published by other researchers studying these methods were utilized to verify correct implementation of the methods. To ensure consistency across the entire project, as well as allow different modeling languages and/or simulation software to be used (Dymola), the entire module was implemented as an **OpenMDAO** driver. Georgia Tech helped in the development of a template for converting the PCC/SA code to an OpenMDAO driver. Thus, the verification methods themselves no longer call, for instance, OpenModelica; they instead rely upon OpenMDAO to facilitate communication between

programs. This has greatly simplified the integration with the Vanderbilt GME tool since OpenMDAO was already integrated with GME and thus effort to integrate the PCC/SA tools was greatly reduced. This also allows the PCC/SA tools to be applied to models simulated in a variety of software packages, such as Dymola, using the OpenMDAO wrappers.

1.3. Integration with MIT Complexity Measure

We have integrated the MIT complexity measure within our code (and thus within the OpenMDAO driver) since the MIT complexity measure also requires quantification of uncertainty and thus could be integrated with the PCC/SA methods.

1.4. Integration with the Georgia Tech Dashboard

The input/output is now integrated with the Georgia Tech Dashboard. This was enabled by changing the input/output format to JSON. For example, previously uncertainty distributions were stored in SQL database; they are now handled within the JSON format. This enables all input/output to be displayed in the Dashboard. More details regarding the specific inputs and outputs of the methods are provided in Section 3.

2. Performance Verification Methodology Overview

This section provides more detail on the methodology. The purpose of the Performance Verification module is to estimate how well a component meets a set of requirements. The Performance Verification does this by estimating a Probability of Correctness for the component.

2.1. Probability of Correctness Computation

For mission-critical design applications, a key consideration is the ability of the designed system to meet the specified performance requirements. In the META X project, the estimation of the PCC is enabled using methods for uncertainty propagation (UP), which is then used to verify the correctness of the proposed designs with respect to a set of specified requirements. In general, the goal of each UP method is to determine the probability that the performance function, $g(x)$ is less than (or greater than) the requirement, c. This can alternately be stated as ensuring the limit state function $z(x)$ is less than or equal to zero (all requirements are converted to format):

$$(x) \leq c \equiv (x) - c \leq 0 \quad (1)$$

As seen in Eq. (1), the limit-state function is $z(x) = g(x) - c$. To compute the PoC, the goal is to estimate the multidimensional integral over the set of input variable distributions:

$$PoC = \int_{\Omega} \ldots \int f(x)(X)dx \quad (2)$$

Where is the joint probability density function, x is the set of random inputs, and Ω= {x | z(x) = 0}. Because the integration cannot typically be performed analytically due to the number of stochastic inputs, **x**, the form of $g(x)$, or because the performance function is embedded in a black box simulation in which only inputs or outputs are known, numerical methods are used to approximate the integral. (These methods are generally classified as methods for uncertainty propagation.) As part of our project, we have implemented and compared six UP methods for PCC estimation: Monte Carlo Simulation (MCS), Taylor Series Method (TSM), Most Probable Point (MPP), Full Factorial Numerical Integration (FFNI), and Univariate Dimension Reduction (UDR). For brevity, in this report, we only provide a short summary of our study of different UP methods and subsequent PCC estimation. The system models are built in Modelica; however, because Modelica is a system modeling language as opposed to a programming language, Python is used to code the various UP methods. The Python scripts call the black box Modelica model using OpenMDAO, requiring only that the input and output variables be known from the Modelica model, but not the functional relationships coded in Modelica. The first step in the performance verification process to compute the PCC of each requirement individually (i.e., the marginal probability of correctness of each requirement). The second step in the performance verification process is to compute the joint probability of meeting the complete set of requirements. Note that the joint probability cannot be obtained by simply multiplying the two marginal probabilities. Instead, we need to compute the covariance matrix for the marginal probabilities.

$$\Sigma = cov(Y_i, Y_j) = E\left[(Y_i - \bar{Y}_i)(Y_j - \bar{Y}_j)\right] \tag{3}$$

If the marginal probabilities in each dimension are normal, we can use a multivariate normal distribution to compute the PPC. In the case in which the marginal distribution is not normal in each dimension, we use the Gaussian Copula function to approximate a true multivariate distribution. Using the Copula function, we can join different types of distributions (normal and beta, etc.)

$$\begin{aligned} C_Y(u_1, u_2, \ldots, u_M) &= \Phi_{\mathcal{N}(0,\Sigma)}(\Phi^{-1}_{\mathcal{N}(0,1)}(u_1), \ldots, \Phi^{-1}_{\mathcal{N}(0,1)}(u_M)) \\ u_M &= F_{Y_M}(Y_M) \end{aligned} \tag{4}$$

As mentioned in the previous section, the methods for UP can be classified into four broad categories as follows:

1. Simulation-based methods such as Monte Carlo simulation (MCS).
2. Local expansion-based methods like the Taylor series method (TS) or perturbation method.
3. The most probable point (MPP)-based methods. The first-order reliability method (FORM) and second-order reliability methods are two popular methods in this category.
4. Numerical integration-based methods, where the statistical moments are first calculated by direct numerical integration, and then the probability density or the tail region probability is approximated using an empirical distribution system based on the calculated moments. The two methods considered from category five in this work are Full Factorial Numerical Integration (FFNI) and Univariate Dimension Reduction (UDR).

In the early phase design, MCS is used when only qualitative or hybrid qualitative-quantitative models are available because MCS is the only method compatible with qualitative models. If quantitative models are available, first-order MPP is proposed at this stage when the number of stochastic input variables is small to moderate. If, for the quantitative models, the set of stochastic inputs is large, TS can be utilized, but only if the system can be reasonably quantified with a linear approximation at the failure surface.

In later phases of design, more advance methods can be utilized. A consideration in this stage is the number of stochastic inputs. For a small number of inputs, the FFNI is recommended due to its accuracy and its ability to handle correlated inputs and interaction effects. The output distribution can be characterized using the Pearson system, and is therefore not limited to a single parametric distribution type, such as a normal distribution. For a moderate to large number of stochastic inputs, the UDR or second-order MPP method is recommended. While neither method can handle correlated inputs, the UDR method has the advantage that neither inputs nor outputs are required to follow a normal distribution; the second-order MPP method requires all inputs and outputs be normally distributed but accounts for input interactions better than UDR.

For the final verification stage, MCS is recommended because it can handle both parametric and non-parametric input uncertainties and makes no assumptions on the output distribution. A key issue with MCS is the computation expense: it is assumed that there are very few or a single system design to evaluate at this phase of the design process. A summary of the methods is provided in Table 1.

Table 1: Summary of PoC Methods

Method	Scalabili	Accuracy	Model	Function	# Inputs	Input f(x)	Output
MCS		Varies	Flexible	Flexible	High	Flexible	Flexible
TS	O(n)	Low	Quant	Low Order	High	Normal	Normal
MPP	O(n)	Med	Quant	Low Order	Med	Normal	Normal
FFNI	O(m^n)	High	Quant	Flexible	Low	Parametric	Parametri
UDR	O(n)	Med	Quant	Flexible	High	Parametric	Parametri

In the MCS method, samples of input variables **x** are generated based on their probability density functions. The system performance function, $g(\mathbf{x})$ is then evaluated at each x_i sample. The CDF of $g(\mathbf{x})$ at requirement limit, c, is estimated by the frequency of $g(\mathbf{x})$ samples less than c. MCS is flexible for any type of input distribution and any form of model function. Neglecting the algorithmic error caused by simulations, if a sufficient number of simulations n is used, MCS results in solutions with a high accuracy. Compared with other numerical methods, MCS has a desirable feature that its computational cost does not generally depend on the dimension of the random model input variables (for a given number of simulations n); however, if there are rare events as a result of interactions, this may necessitate the use of a greater number of simulations.

The Taylor Series (TS) method approximates the performance function, $g(\mathbf{x})$ with a p-order Taylor series truncation. Typically, the Taylor series is truncated at the first or second order terms to create first and second order Taylor Series approximation, respectively. Once the TS approximation of $g(\mathbf{x})$ is computed, the first two moments can be computed to estimate

the mean and variance of $g(\mathbf{x})$. The first-order Taylor Series method is typically utilized for uncertainty propagation due to its straightforward implementation.

The MPP method was originally developed in the field of reliability analysis. The MPP is formally defined in a coordinate system of an independent and standardized normal vector. The input variables **x** (in the original design space) are transformed into the standard normal space **u**. The MPP is defined as the shortest distance from the origin to a point on the limit-state surface in **u** space. Mathematically, finding the shortest distance is a minimization problem with an equality constraint: The solution **u** of this minimization problem is called the most probable point (MPP). At the MPP, the joint probability density function on the limit-state surface has its highest value; therefore, the MPP in the standard normal space has the highest probability of producing the value of limit-state function z (**u**). The MPP is the point on z (**u**) that contributes the most to the integral for probability estimation.

The FFNI method performs the numerical integration of Eq. (2) using the Gaussian quadrature numerical integration technique. This method is used to compute the moments of the $g(\mathbf{x})$ output distribution, which are then used to construct a parametric distribution of the output to compute the PCC. With this approach, the statistical moments of the performance function $g(\mathbf{x})$ are calculated through direct numerical integration using an appropriate quadrature formula. In numerical analysis, a quadrature formula is an approximation of the definite integral of a function, usually expressed as a weighted sum of function values at specified points in the domain of integration. These sampling points are called the nodes and the weighting factors are the weights.

The UDR method is similar to the FFNI method in that it utilizes numerical integration; however, it approximates the multivariate function with multiple univariate functions used to calculate the multivariate statistical moments. This method reduces computational cost by approximating the performance function $g(\mathbf{x})$ by a sum of univariate functions, which depend on only one random variable with the other variables fixed to their mean values.

2.2. Hierarchal Sensitivity Analysis

Global Sensitivity is a variance-based method to quantify the amount of variance that each input factor contributes with on the unconditional variance, V, of the output response. This analysis assumes a model of the form $\mathbf{Y} = f(\mathbf{X})$, where **Y** is the output and $\mathbf{X} = (X_1, X_2, \ldots, X_m)$ are m independent input factors, each one varying as defined by a probability density function. Unlike conventional global sensitivity analysis, in this proposal we consider that the output **Y** is a set of n output responses $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ and therefore the goal is to determine the amount variance and covariance each input factor contributes to the unconditional variance-covariance matrix, V_{ij}, of the n output responses. The goal then of the analysis is a ranking of the input factors according to the amount of variance that would disappear, if we knew the true value of a given input factor X_i. The methods utilized for this analysis allow computation of contribution of both main effects, i.e., the effect of the individual X_i, as well as the contribution of interaction effects, i.e., $X_i \cdot X_j$, $X_i \cdot X_j \cdot X_k$, etc.

Three methods for global sensitivity analysis have been developed and implemented as part of the Meta II project. The methods are summarized in Table 2.

Table 2: Summary of Sensitivity Methods

Method	Scalabilit	Accuracy	Model Type	Function	# Inputs	Input f(x)	Output
Sobol'		Varies	Flexible	Flexible	High	Flexible	Flexible
FAST	O(n)	Med	Quant	Flexible	High	Parametric	Parametric
EFAST	O(n)	Med	Quant	Flexible	Low	Parametric	Parametric

Sobol' introduced the first order sensitivity index by decomposing the model function *f* into summands of increasing dimensionality. The approach has been expanded by subsequent researchers to include computation of the total sensitivity index. The integrals utilized in the analysis can be computed with Monte Carlo methods.

The main idea underlying the FAST method is to convert the k-dimensional integral into a one dimensional integral. Each uncertain input factor is related to a frequency ω and transformed by $X(s) = G_i(\sin(\omega_s))$, where G_i is a suitably defined parametric equation which allows each factor to be varied in its range, as the parameter s is varied. The set $\{\omega_1,\ldots, \omega_k\}$ are linearly independent integer frequencies.

In 1999, researchers proposed an improvement of the FAST method. They called it the Extended Fourier Amplitude Sensitivity Test (EFAST). With this method they could estimate the total effect indices, as in the Sobol method, by estimating the variance in the complementary set. This is done by assigning a frequency ω for the factor X (usually high) and almost identical frequencies to the rest ω_i (usually low).

The performance verification also provides the capability of hierarchical sensitivity analysis to decompose the complex system design into separable subsystems based on its hierarchy. A built- in example of ramp system can be followed for conducting hierarchical sensitivity analysis.

3. Requirements

3.1. Inputs (all inputs specified through the Dashboard)

- System model: a black-box type of model (i.e., Modelica) which has outputs of system performance based on component inputs parameters.
- Stochastic component input parameters: these are the list of input parameters of the system model to be varied and their uncertainty distributions.
- Component output requirements: the list of output parameters of the system model and their lower and upper limits based upon a requirements document.

3.2. Outputs (all inputs specified through the Dashboard)

- Marginal PCC for each requirement
- Joint PCC for the set of requirements
- First Order (and optionally Total) Sensitivity of the output variance to input variances.
- Graphical representation of results.

3.3. Format

- System model: Constructed in the Generic Modeling Environment (GME)
- Stochastic component input parameters: Specified in the Parametric Exploration Tool in GME
- Component output requirements

3.4. Running Performance Verification

PCC/SA can be run from within GME. See GME documentation for running PCC/SA from within GME.

4. Limitations

4.1. Implementation

1. The methods have not been tested extensively on complex models (i.e., Ricardo Dymola Models).
2. Distribution fitting is only based on the first four statistical moments.

4.2. Method Selection

It is important to use appropriate method for uncertainty analysis. The most accurate method in uncertainty propagation is the Monte Carlo Simulation for any system independent of its complexity or nonlinearity. However, the computational cost increases directly with the complexity, nonlinearity and the number of design variables. Therefore for extremely large scale and complex system often the designer have very few samples for the system. This limitation makes it even harder to study the behavior of the system and also in deriving an accurate probability of failure. There are several improvements for lowering the cost of MCS for more complex systems. These improvements include both importance sampling and also meta-models as surrogate systems to be used instead of the original systems

In the local expansion based methods such as Taylor Series, perturbation, FORM and SORM the accuracy is highly dependent on the degree of nonlinearity of the limit state function. But these methods will be reliable for simpler systems with small number of design variables with relatively acceptable convex limit state function.

The UDR and FFNI methods are appropriate when the moments of the system exist. The limit state functions (i.e., Modelica system model) can be nonlinear. In case of heavy tail phenomena, the moments do not exist for the system and it is recommended that MCS to be used instead.

In general the appropriate method in uncertainty propagation highly depends on the dynamic

of the system, existence of heavy tail phenomena, and the degree of nonlinearity and also complexity of the system. Therefore it is recommended for the designer to have an understanding of such concepts.

5. Proposed Future Work: Tail Study

Calculating the probability of failure is a challenging task in a large scale and complex system. It is mainly because of large number of random variables and their interactions. The common methods to calculate probability of failure are sampling methods (MCS), Most Probable Point (MPP) based methods (FORM, SORM), Taylor Series (TS) methods, Full Factorial Numerical Integration method (FFNI), Univariate Dimension Reduction (UDR) method, polynomial Chaos Expansion (PCE) method.

The exhaustive Monte Carlo Simulation method is very expensive and almost impossible for a complex system. The rest of these methodologies are based on calculating the moments of the limit state function. However these moments might not exist in reality. Therefore the question arise how confident the results are in these methods.

In order to answer this question, we focus more on the tail of the distribution of the limit state function. Since the probability of failure is defined as the area of the tail of the distribution. At first, we test if we have heavy tail phenomena at the tail. One well-known graphical method is using QQ-plot. In this plot, the quantiles of the limit state function is compared to the quantiles of the standard normal distribution. If the quantiles of the data are above the standard normal
line, then the real tail is heavier than the normal distribution. However, if the quantiles are below the $y = x$ line, then the standard normal tail is heavier than the real tail .

Thus by using the conventional methods, we might have underestimated the probability of failure or also over-estimated the value for some of the design variables based on the analysis of their tail. It is notable to mention that for a complex system these plots are based on a very small sample of the limit state function. This might be the case for large scale and complex systems where each attempt to propagate uncertainties will be costly and almost impossible.

We propose developing methods which quantify the tail of the output distribution and use this information in both the fitting of the output distribution as well as to place bounds on the estimated PCC.

APPENDIX B: SUBCONTRACTOR FINAL REPORT – QUALITATIVE SIMULATION

GUIDING AND VERIFYING EARLY DESIGN USING QUALITATIVE SIMULATION

ABSTRACT

Design of a system starts with functional requirements and expected contexts of use. Early design sketches create a topology of components that a designer expects can satisfy the requirements. The methodology described here enables a designer to test an early design qualitatively against qualitative versions of the requirements and environment. Components can be specified with qualitative relations of the output to inputs, and one can create similar qualitative models of requirements, contexts of use and the environment. No numeric parameter values need to be specified to test a design. Our qualitative approach (QRM) simulates the behavior of the design, producing an envisionment (graph of qualitative states) that represents all qualitatively distinct behaviors of the system in the context of use. In this paper, we show how the envisionment can be used to verify the reachability of required states, to identify implicit requirements that should be made explicit, and to provide guidance for detailed design. Furthermore, we illustrate the utility of qualitative simulation in the context of a topological design space exploration tool.

INTRODUCTION

The field of qualitative reasoning has its roots in capturing human reasoning about the physical world. Such reasoning about the interactions of connected elements is at the heart of an early design process, where a designer is attempting to achieve some desired overall behaviors, and avoid unwanted interactions. Consider the drivetrain model in Figure 1. A qualitative analysis will show that it can move smoothly up through the gears, increasing speed over level terrain. But it will also show that the engine may stall because of excessive load for certain combinations of design parameters, and driving and terrain patterns. This qualitative analysis can provide guidance for parameter and component selection during detailed design. By linking our qualitative reasoning system to a standard tool for interactive graphical design (Open Modelica, http://www.openmodelica.org/), we are enabling designers to use qualitative analysis as part of their standard work practice.

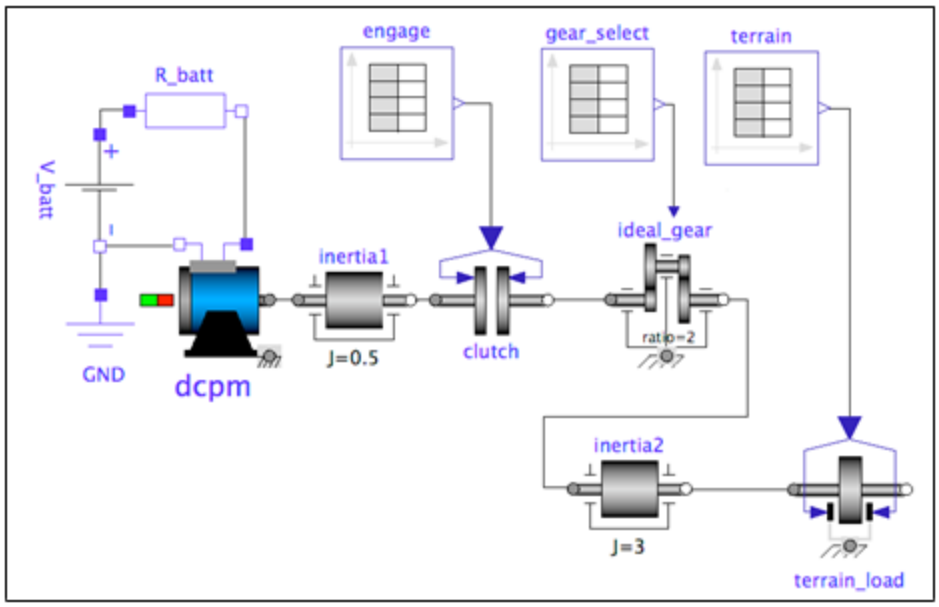


FIGURE 1: MODELICA MODEL OF A VEHICLE DRIVETRAIN

While developing models of systems with fully specified parameters, engineers frequently have to determine whether their numerical results conform to expected behaviors or are in fact errors in their modeling or simulation. This process relies on an understanding of all constraints on possible dynamics of the system (e.g., when the engine is running, and the vehicle is in a forward gear it should not go backwards, it is possible for the engine to stall, etc.). Qualitative reasoning automates this form of reasoning.

Many new designs of systems are instantiations of previous successful designs that leverage new components and/or capabilities of materials. For innovative results, it is useful to explore a larger space of designs including new topologies of components. Doing detailed parametric design for each element of the space is costly; qualitative verification helps prune the space by efficiently analyzing component topologies without the need to specify all component parameters needed for numeric simulation. Qualitative modeling supports the rapid exploration of designs that are only specified using the mathematical form of the relationships between a component's inputs and outputs. The systems do not need to be piece-wise linear; non-linear models are fine. Given a model, qualitative simulation generates all possible behavioral trajectories of the system's variables. Analyzing these trajectories can determine whether with appropriate parameter selection, a design could satisfy the requirements, or whether it can never fulfill certain requirements.

This paper begins by introducing qualitative simulation, its representation and semantics. We then discuss our design architecture QRM and our approach to creating models. We illustrate this using an example of a door system based on an infantry fighting vehicle, and highlight how qualitative simulation verifies requirements and guides detailed design by identifying implicit failures. In a second example dealing with electric circuits, we show how qualitative simulation drastically prunes a design search space. We close with a discussion of related approaches, scaling and future work.

QUALITATIVE SIMULATION

Qualitative simulation [1][2][3] or envisioning, is the process of projecting forward, from an initial situation and a model, all possible qualitative states that may occur. Qualitative representations of continuous quantities (e.g., the voltage across a diode) are central to this process. In our familiar Newton- Liebnitz calculus we use variables to represent quantities that can take any value from the real number line, and vary with time. Variables can have arbitrarily many higher-order derivatives. Likewise, in qualitative reasoning, these variables and their derivatives take on values – except that the values are qualitative. Each variable (or derivative) has a quantity space consisting of an ordered set of landmark values representing important points for understanding the behavior of the model (e.g., the turn-on voltage for a diode). A qualitative value is either a landmark or the open interval denoted by two adjacent landmarks. For a door, there are two landmark values: Closed and Open. The doors position can be at one of these two landmarks, or between the (Closed, Open). The qualitative value also has a direction (a qualitative derivative) of increasing, decreasing or steady. The most common quantity space uses just the sign of the real quantity. We represent the interval x<0 as Q-, x=0 as Q0, and x>0 as Q+).

A qualitative state is an assignment of qualitative values to variables in the model. We represent equations as qualitative constraints. Consider the equation governing a resistor, V=I*R, where voltage, V, and current, I, are quantities and R is a fixed parameter with a positive value. The resulting multiplication constraint ensures that the qualitative product of I and R is V. Because R is a positive constant value, if I is a negative value, then V must also be a negative value. Furthermore, their derivatives must also match. Figure 2 defines qualitative addition and multiplication for sign values.

+	Q-	Q0	Q+
Q-	Q-	Q-	?
Q0	Q-	Q0	Q+
Q+	?	Q+	Q+

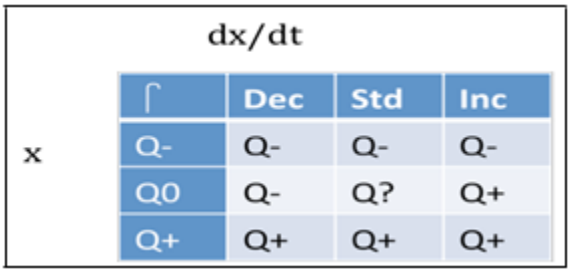
*	Q-	Q0	Q+
Q-	Q+	Q0	Q-
Q0	Q0	Q0	Q0
Q+	Q-	Q0	Q+

FIGURE 2: QUALITATIVE ARITHMETIC TABLES

One of the most significant consequents of the coarseness of qualitative values is that variables may be qualitatively constant for long periods of time (perhaps infinite). Hence, qualitative simulation need only consider the instants of time at which there is a possible change in qualitative value,. The passage of time is represented as an alternating sequence of instants and intervals. A qualitative state can either describe an instant or an interval. Qualitative simulation determines all trajectories through the qualitative state space from an initial state. Given a state, qualitative simulation computes possible successors for each quantity value and uses constraints to determine how they may be combined to form a next, if any, state or states. The rules for generating successor values and directions are based on the mean value theorem from calculus [4].

Consider a position quantity that was between open and closed and moving toward closed. There are four possible successors for this quantity. Its value may remain in the interval or reach the

closed landmark and it may continue increasing or become steady (its derivative stays positive or becomes Q0). Figure 3 illustrates the qualitative integration rule for an instant to the following interval where are variables are continuous.



x	dx/dt		
∫	Dec	Std	Inc
Q-	Q-	Q-	Q-
Q0	Q-	Q?	Q+
Q+	Q+	Q+	Q+

FIGURE 3: CONTINUITY OF NEXT VALUES FROM AN INSTANT

From basic calculus if a variable is non-zero at an instant, it will remain at that qualitative value in the following interval. If the variable is 0, it will have the qualitative value of its derivative over the following interval. There is one ambiguous case: if a variable and its derivative are both 0, the qualitative value on the following interval is ambiguous (but the variable and its derivative must be qualitative equal during the interval). Consider x=t^2 when t=0. The qualitative values x and dx/dt are both Q0, but x=Q+ on the following interval.

Cyber-physical systems include dynamics that are discrete as well as continuous (e.g., an input signal to open the door, the changing of gears in a drive train, a diode switching from off to on). We model such changes through modes, which include an entry condition, initial values for variables, and equations that are valid within that mode. During simulation, discrete changes occur at instants when mode entry conditions are satisfied. The initial values and equations govern the behavior of quantities in the following interval and subsequent states. Modes are different than the operating regions in that they allow for the modeling of hysteresis.

QUALITATIVE SIMULATION SEMANTICS

For qualitative reasoning to be useful for verification it must have a well-defined semantics. One can prove a theorem: Given a qualitative model with the appropriate abstractions for the ODE's used in, say Modelica, to define continuous behavior for a numeric simulation, the qualitative simulation will contain a path which describes the trajectory of the numeric simulation [3].

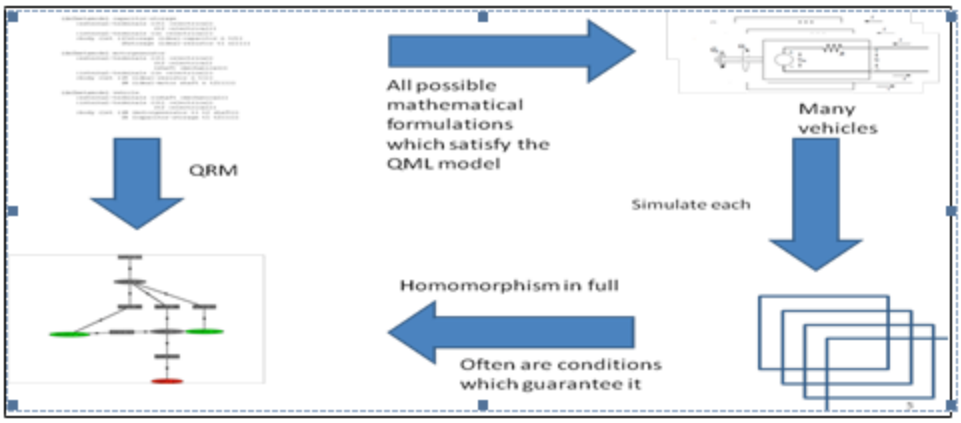


FIGURE 4: QUALITATIVE SIMULATION SEMANTICS

DESIGN ARCHITECTURE

In this section, we describe our view of an automated, or semi-automated, design process (shown in Figure 5). A human designer or an automated Design Space Exploration tool starts with a high level functional specification of the desired system to be designed. This search produces tentative topologies for analysis. These topologies are expressed in the Modelica connection language only specifying components and their connections; it need not contain any parameter values. This is illustrated more concretely in Figure 4. This particular example is of an electric vehicle, but the details of the model are irrelevant here. Qualitative simulation produces envisionments from qualitative models (left downarrow). An envisionment corresponds to a real system in the following way. First, the qualitative models describe an infinite number of possible systems (all possible numerical assignments to parameters as well as all possible conventional component models which satisfy the qualitative model, including non- linear ones). Each of those systems will have a particular behavior (right downarrow). Each such behavior will map to a sequence of qualitative states (leftarrow). Each possible real behavior occurs in the envisionment. Hence, if a desired behavior does not appear in an envisionment, it cannot occur in with any possible assignment of parameters to this system. This is an extremely important property.

One would also like the converse to be true: that every state in the envisionment can actually occur in a real system. Although this property often holds, and there are complex conditions under which it holds. However, we cannot guarantee it in general [6]. Elimination of spurious transitions and states has been an active research area in the qualitative reasoning community.

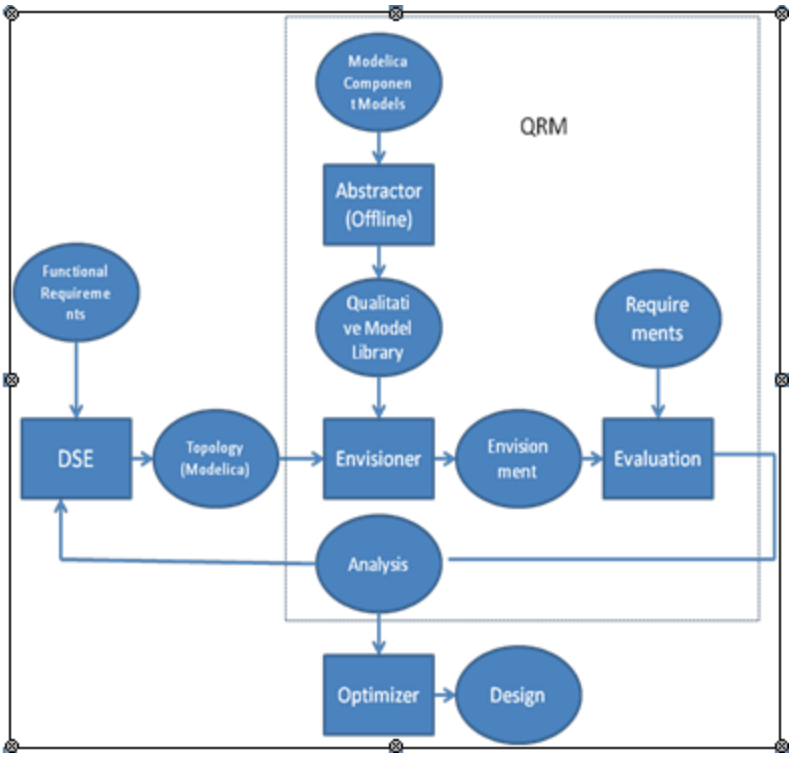


FIGURE 5: QRM SYSTEM ARCHITECTURE

Given the qualitative models and the topology the envisioner constructs the envisionment of the system. The requirements (converted to qualitative terms) are evaluated against the envisionment. Some requirements may be met, some may not. If the requirements are not satisfied, this analysis identifies which requirements fail to be met and why. This is then presented to the designer or the automated Design Space Exploration tool. If the requirements are adequately met, subsequent analysis selects parameter values which optimize the requirements. This optimization may discover no assignment of values to parameters meets the numerical requirements in which case this analysis will be fed back to the designer or DSE.) This paper focuses on the fully implemented qualitative reasoning aspect of this process which we call the Qualitative Reasoning Module (QRM).

The design space explorer uses component models from a standard library. The vast majority of our qualitative model library is obtained from Modelica models which are abstracted only once, and comprise 'well written' Modelica models abstract directly. More complex Modelica models require human intervention. Inclusion of Modelica function blocks, algorithm blocks or complex conditionals are difficult to translate automatically. These abstractions need be done only once and form the qualitative component model library.

BUILDING QUALITATIVE MODELS

Component-based modeling is becoming increasingly popular in industry (e.g., Modelica [5]) due to savings incurred by reusing existing models for new applications. Component modeling efforts take lots of resources; therefore, we align our models as much as possible with Modelica to facilitate our ongoing automatic translation efforts. The composition of models occurs through connections that are domain specific (e.g., electrical pin). The composition of the models creates additional constraints on the flow and effort variables of the models governed by Kirchhoff's current and voltage laws. One area where we differ from Modelica representation concerns our use of modes instead of conditional equations. Modes offer the following advantages: (1) they localize the definition of hybrid behavior for the component, and (2) they provide a natural way to model various faulty behaviors.

To illustrate our modeling approach, Figure 6 contains our definition of an ideal-diode. We present this here in our internal S-expression syntax which highlights this localization. This model is a subclass of the electrical one port model, which defines two electrical connections, a positive pin and a negative pin, and variables for the current and voltage of the diode. The redefinition of the voltage variable v is essential to define the quantity space including Q0, representing 0V, and OnVoltage, representing the turn on voltage for the diode. The diode has two modes, off and on. The component is in a mode until the entry conditions for another mode have been satisfied, in this case, if the diode was off, the equation stating that no current was passing through the diode would be enforced. This persists until the instant when the voltage transitioned to OnVoltage, at which point the equation holding the voltage constant would be enforced and current would be allowed to flow through the diode.

```
(defprototype ideal-diode :extends (one-port)

:variables ((v voltage :landmarks (Q0 OnVoltage)))

:mode (off :entry ((= i Q0))

:equations ((= i Q0)))

:mode (on :entry ((= v OnVoltage))

:equations ((= v OnVoltage))))
```

FIGURE 6: DIODE MODEL WITH TWO MODES

We have defined a standard template for expressing modes that is acceptable to current Modelica compilers, but do not include it here.

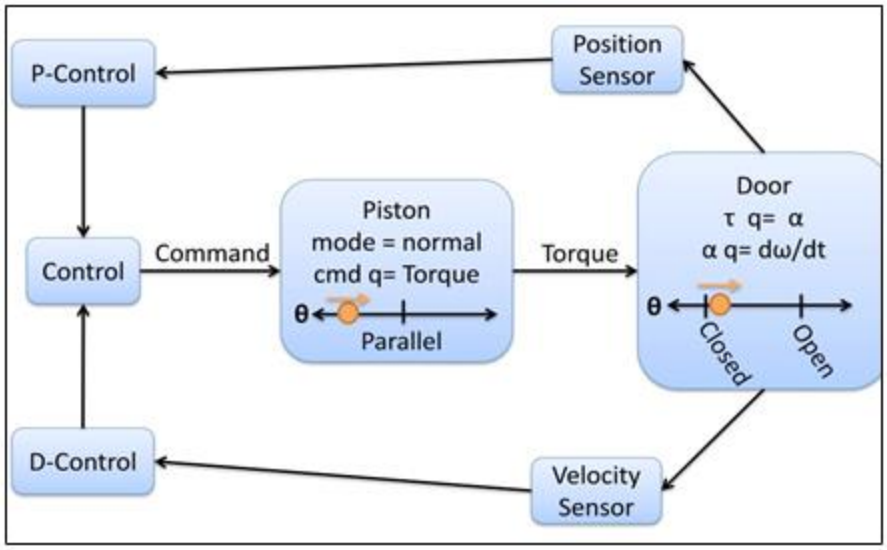


FIGURE 7: ARCHITECTURAL OF THE DOOR SYSTEM

QUALITATIVE SIMULATION FOR VERIFICATION

In addition to specifying a topology of connections between qualitative component models, it is necessary to encode requirements in a formal language. We work with a variety of temporal logic specifications [7]. While Linear Temporal Logic is common in verification, Computational Tree Logic is an extension of LTL that is better suited to qualitative verification of requirements over multi-trajectory envisionments. Requirements may be evaluated over an individual qualitative state in the envisionment (e.g., a variable should never exceed a particular level); requirements may also take into account a sub path of a trajectory.

Success and failure conditions for our envisionment algorithm can terminate simulation along a trajectory when one of the conditions is met. The requirement, "the door shall not overshoot the closed position" a, can consider a state a failed terminal state if the door's position is below the closed landmark.

After the envisionment graph has been created, QRM provides the following analysis. If none of the trajectories violate requirements, then all possible numeric values for the system parameters will satisfy all requirements (recall the completeness guarantee of the envisionment graph). If some trajectories violate requirements and others do not, then the design may satisfy the requirements with appropriate constraints on parameter values. In this case, detailed design is required to determine an assignment of parameter values that will satisfy the requirements. If all of the trajectories violate requirements, detailed design is not necessary because no set of parameter values will satisfy the requirement.

VERIFICATION EXAMPLE: VEHICLE DOOR LINKAGE

To illustrate qualitative simulation consider the door system shown in Figure 7. The architectural model shows quantity spaces for the positions of the piston that moves the door, and the door itself. The system consists of a PD controller, which uses position and velocity sensors from the door, a piston, whose linear motion applies a torque on the door, and finally

the door slab itself. An input signal to the controller specifies the desired position for the door. In this case, the door has two landmarks in the angular position.

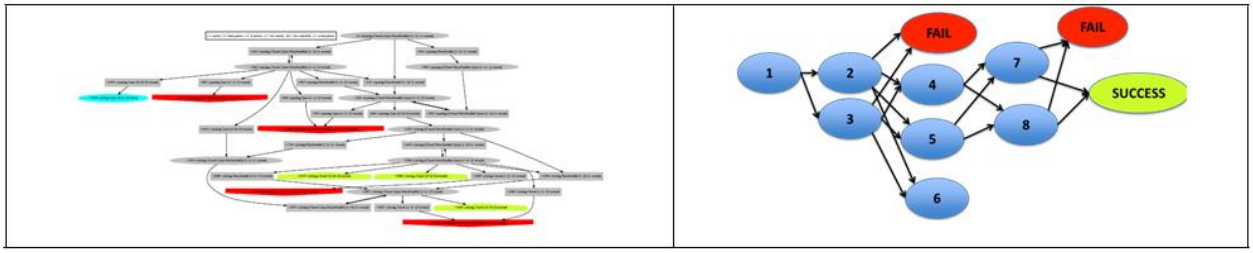


FIGURE 8: THE ENVISIONMENT GRAPH FOR THE DOOR SYSTEM MODEL

THE ENVISIONMENT COMPUTED BY QRM IS ON THE LEFT, AND A SIMPLIFIED VERSION ON THE RIGHT

quantity space, closed and open, and the piston has one landmark on the linear position quantity space, piston parallel, representing the position where the piston acts in parallel with the hinge. We will evaluate this design against the requirements that the door should always be able to be closed, the door's position should operate between the door open and door closed position inclusively. For a context of use, from an initial situation in which the door is closed, we will consider two discrete transitions: (1) the command is given to open the door, and (2) when the door has reached the open position, the command will be given to close the door.

QRM produces the envisionment (shown in Figure 8) providing the following feedback to the designer. The design may reach a successful situation (shown in green). Each of the requirements that may be violated is shown in red. Therefore, appropriate parametric assignment will be needed to ensure that trajectory for each failed state is avoided. A metric for estimating how difficult it will be to verify the design is the ratio of successful states to terminal states, in this case 1/3.

Further analysis of the envisionment provides additional guidance for the detailed design. There is a terminal situation, 6, that does not satisfy the success or failure conditions of the system. This dead-end state implies the need for additional requirements to guide the designer to avoid this state. In this case, this situation results from a kinematic singularity in the piston door connection. That is, when the acting angle of the piston is parallel to the angle of the door, the piston produces no torque. While this is part of the piston component model, it is only leads to a quiescent (terminal) state if the door is stationary at this point. To identify this risk requires simulating the system with a use case where the door first opened and then closed. This analysis happens very early in the design process, when alternative system topologies are being considered. In the next section, we illustrate how this process could be used within an automated design space exploration system.

QUALITATIVE VERIFICATION IN TOPOLOGICAL DESIGN SPACE EXPLORATION

Innovative design exploration searches for configurations of existing components (new topologies) to achieve specified functionality. Consequently, this search space is exponentially large in the number of components in the design. Qualitative verification prunes the design space in two ways. The first is use of qualitative models of components, where the component models capture only significantly different behaviors of the models. The second is use of a qualitative simulation to identify bad topologies from which no choice of parameters will satisfy the requirements, and to guide parameter selection in detailed design. Qualitative simulation graphs are much smaller than those that explore parameter spaces. Therefore, qualitative verification can eliminate designs for large parts of the parameter space.

Consider the following example of designing a system that turns on a light after a short delay of a switch being flipped. If the available components include batteries, switches, resistors, capacitors, inductors, and diodes, the topological design space includes every configuration of these components. To illustrate the utility of qualitative verification, we will consider a design space exploration tool that searches the design space by taking one of the following design actions: adding a component in parallel or series with an existing component, removing a component, or flipping a component in the circuit. Figure 9 illustrates the starting design, which includes just a battery, switch and diode.

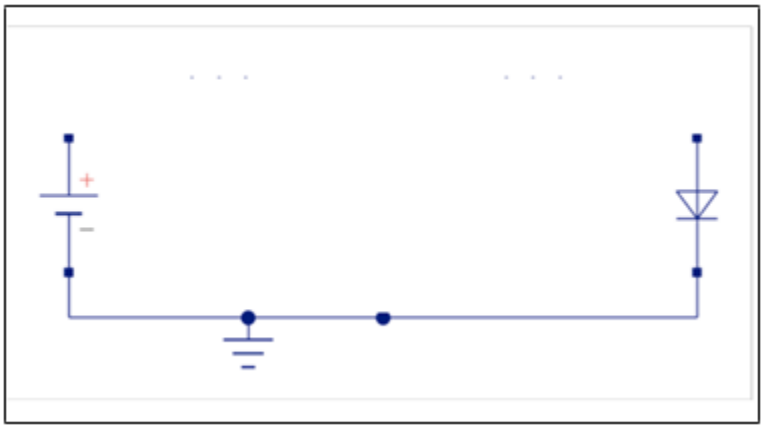


FIGURE 9: STARTING POINT FOR TOPOLOGICAL DESIGN SPACE EXPLORATION

After each design action, we attempt to build a qualitative model and simulation for the current design candidate. Now many of these candidates are actually shorted or open circuits and QRM identifies them because their initial conditions are inconsistent. If the design candidate has consistent initial conditions, QRM generates an envisionment and analyzes the results. Consider a circuit with a resistor in place of the cloud in Figure 9. The envisionment of this will begin with both the switch and diode off, and has two trajectories for the instant the switch is turned on. In one, the diode is on, and, in the other, the diode is off. The trajectory of the actual system depends on the ordinal relationship between the on voltage for the diode and the battery's voltage. Because neither of these trajectories satisfies the requirement that there exists a delay before the light turns on, qualitative verification eliminates this topology without considering all possible combinations of battery voltages, resistances and on voltages.

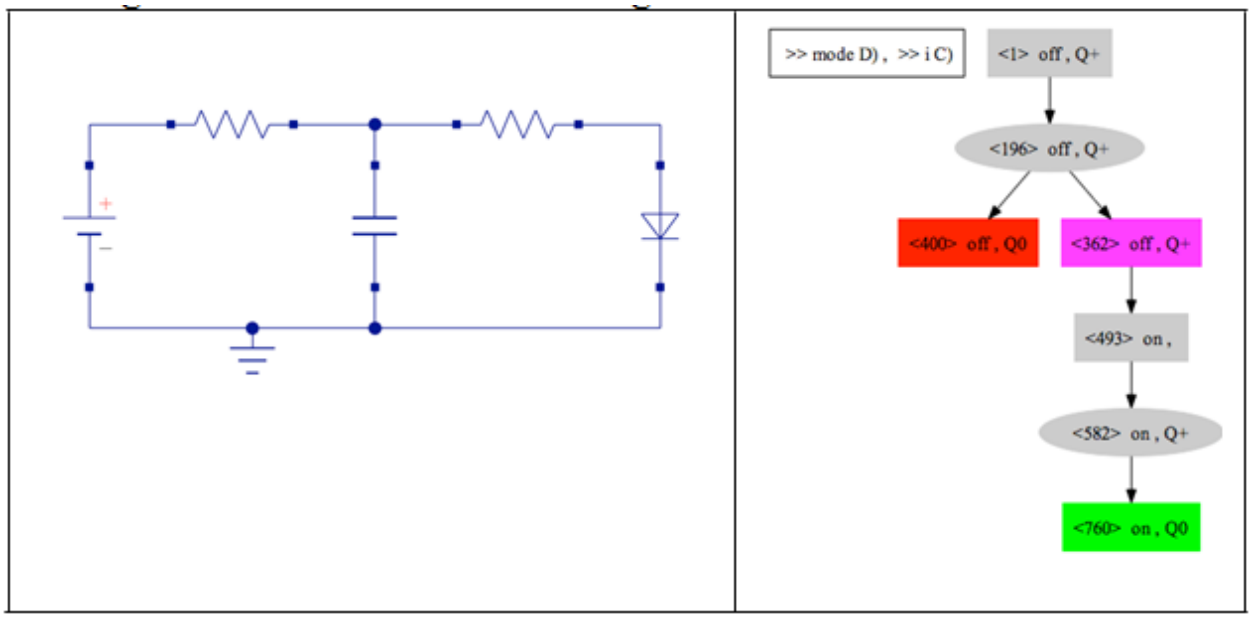


FIGURE 10: THE ENVISIONMENT ON THE RIGHT PROVES THAT THE TOPOLOGY ON THE LEFT CAN SATISFY THE DESIGN REQUIREMENTS

Now consider the design in Figure 10. QRM produces an envisionment with two trajectories. They are identical in the interval after the switch is turned on, the capacitor is charging and the voltage across the diode is increasing. This interval terminates in one of two instants: (1) the current ceases flowing into the capacitor and the system reaches a steady state, and (2) the voltage across the diode reaches the on voltage landmark causing a mode transition (shown in magenta) resulting in the diode turning on. This second trajectory satisfies the requirement. Therefore, this topology is a candidate for parameter selection and care should be taken that the battery voltage should be greater than the turn on voltage of the diode.

SCALING

One of the promises of Qualitative Reasoning applied to design is its performance. By answering simple questions, requirements can be evaluated for surprisingly complex systems very quickly. We draw on decades of experience on building fast qualitative envisioners. In particular, we draw on advances developed in the recent DARPA Deep Green program [8].

Qualitative Reasoning has the advantage it only needs to address qualitative distinctions – that alone often severely contracts the search space. The complexity of QRM is not driven by the number of variables in the system – it is more determined by the dynamics of the system. If the dynamics are simple, analysis will be simple. We can successfully analyze systems of tens of thousands of variables in seconds. On the other hand, we can construct a pathological example with a few

dozen components that cannot be solved (e.g., the voltage across a series of unsynchronized oscillators).

One important way QRM improves its performance (first developed in Deep Green) is to include requirement evaluation during envisioning. If a state or a combination of states do not meet the requirements, QRM immediately cuts off generation of any subsequent states: after all there is no necessity to analyze the consequences of states that do not meet requirements.

QRM also includes a qualitative solver which determines when qualitative variables are locked (state dependent) together and thus can be completely eliminated. For this, it uses a form of qualitative algebra. This greatly reduces the complexity of most analyses.

COMPARISON TO OTHER APPROACHES

There is a broad literature on formal verification of hybrid systems. However, almost all approaches require quantitative models and numerical parameters. Such information is often not available in early. In contrast to our approach of constructing a model from components, verification with HybridSAL[10] begins with a set of equations, with numeric values chosen. HybridSAL [10], is also limited to linear models.

HybridSAL has the advantage of being able to answer quantitative questions about a design (e.g., will the vehicle reach 30 mph in 6 seconds). Answering such queries for fully specified designs is an important part of our future work; we believe that the QRM envisionment can improve the efficiency of our version of this analysis. It is an open question what classes of non-linear equations can be analyzed.

Other researchers have explored the use of PRISM [9] to perform verification of cyber-physical systems. PRISM models have the advantage that they can consider probabilistic state transitions. Probabilistic state transitions make PRISM particularly useful for verifying requirements about the likely reliability of systems given failure rates of components (e.g., "what is the probability that vehicle will be able to operate continuously for 570 hours"). A challenge for doing this analysis is that there is no automatic way to move from equations specifying components to the models used by PRISM.

As we have shown in this paper, even when we know all the models and values, QRM can help verify requirements very quickly. Almost all formal verification tools are very general and their performance scales very poorly with number of variables or components. For more complex systems, QRM can verify requirements when formal methods cannot. QRM has the advantage of algorithms specifically tuned to continuous systems developed over decades in the AI community.

DISCUSSION

We have presented our QRM approach for early design verification using qualitative simulation. In particular, we have illustrated how envisionments can verify requirements and guide detailed design by identifying implicit requirements. Furthermore, we have shown that qualitative

verification using QRM is able to eliminate large areas of the intractable search space of design from components.

Our initial explorations have opened a number of promising directions for future work. As stated earlier, automatically incorporating available quantitative information about parameters would allow us to verify a large set of requirements. We intend to build on existing work on semi- quantitative simulation [11]. Another important future direction concerns the interaction between design space exploration and qualitative simulation. In the case of design flaws, QRM could use the envisionment to produce diagnoses guiding topological search. In the case of potentially successful designs, the envisionment could provide guards, or inequalities, to guide parameter selection.

ACKNOWLEDGMENTS

Peter Bunus (Linkoping) for helping us understand the nuances of Modelica models. This work was sponsored by The Defense Advanced Research Agency (DARPA) Tactical Technology Office (TTO) under the META program. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

REFERENCES

[1] de Kleer, J. and Williams, B.C. 1992. Special volume on Qualitative Reasoning about Physical Systems II, Artificial Intelligence. Elsevier.

[2] Forbus, K. 1984. Qualitative process theory. Artificial Intelligence, 24. Elsevier 85-168.

[3] Kuipers, B. 1994. Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. Cambridge, MA: MIT Press.

[4] Williams, B. 1984. The Use of Continuity in Qualitative Physics. In Proceedings of the National Conference on Artificial Intelligence, Austin, TX, pp. 350-354.

[5] Fritzson, P. 2004. Principles of Object Oriented Modeling and Simulation with Modelica 2.1. Piscataway, NJ: IEEE Press.

[6] Say, C. and Akin, H. 2003. Sound and complete qualitative simulation is impossible, Artificial Intelligence, v.149 n.2, p.251-266.

[7] Konur, S. 2010. A survey on temporal logics. CoRR, abs/1005.3199.

[8] Hinrichs, T. R.; Forbus, K. D.; de Kleer, J.; Yoon, S.; Jones, E.; Hyland, R.; and Wilson, J. 2011. Hybrid qualitative simulation of military operations. In Proceedings of Innovative Applications of Artificial Intelligence. San Francisco.

[9] Kwiatkowska, M., Norman, G., and Parker, D. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585-591, Springer.

[10] Tiwari, A. 2008. Abstractions for hybrid systems. Formal Methods in Systems Design, 32:57–83,

[11] Berleant, D. And Kuipers, B. 1997. Qualitative and quantitative simulation: bridging the gap. Artificial Intelligence 95(2): 215-255.

STOCHASTIC PROCESS DECISION METHODS FOR COMPLEX CYBER-PHYSICAL SYSTEMS

K. Willcox, D. Allaire, J. Deyst, C. He, A. Babuscia, and E. Clements

Massachusetts Institute of Technology

December 2012

Final Report

1.0 SUMMARY

The primary objective of our effort is to develop a fundamental theory to quantify the inherent uncertainties and risks in complex system design and development processes. These theoretical developments will help enable the achievement of the META goal of devising, implementing, and demonstrating in practice a radically different approach to the design, integration/manufacturing, and verification of complex systems. Our approach to meeting this objective is: to adapt the entropy concepts of information theory to create a metric for system complexity; to apply estimation theory to characterize inherent uncertainty in system development processes; and to utilize this theoretical base to develop efficient methods for resource allocation so as to manage uncertainty and mitigate risk in complex system developments.

Our specific innovative claims for this project, building on our previous DARPA META project, are as follows:

1. Viewing system development as a problem of Bayesian estimation leads to a theoretical framework for complex system development.

2. Quantifying complexity in terms of information theoretic concepts permits the treatment of the complexity metric with the tools of estimation theory. This enables a systematic approach to quantitative modeling of system development as a resource investment procedure in the presence of uncertainty.

3. A stochastic model for system development facilitates quantification of the uncertainty reduction that is necessary for success and can be used as a tool to monitor the actual development process.

4. Our proposed theoretical framework for uncertainty quantification provides the bedrock upon which the methods and tools, enabling orders of magnitude improvement in complex system developments, can be built.

In this research we achieved our objectives through further development and demonstration of the complexity metric defined under our previous META project. This includes demonstrating our approach on the Vanderbilt University bond graph model of an infantry fighting vehicle, establishing a correspondence between complexity-based sensitivity analysis and variance-based sensitivity analysis for additive functions with Gaussian distributions, creating a compositional UQ methodology, creating an expert elicitation procedure for model discrepancy quantification, and creating a resource allocation methodology for redesign and refinement decisions. Some of the material found in this report may also be found in Ref. 40.

2.0 INTRODUCTION

Over the years, engineering systems have become increasingly complex, with astronomical growth in the number of components and their interactions. With this rise in complexity comes a host of new challenges, such as the adequacy of mathematical models to predict system behavior, the expense and time to conduct experimentation and testing, and the management of large, globally-distributed design teams. These obstacles contribute uncertainties to system design, which can have serious, often disastrous, implications for program outcome. A notable example is the Hubble Space Telescope which, when first launched, failed its resolution requirement by an order of magnitude. A Shuttle repair mission, costing billions of additional dollars, was required to remedy the problem [1]. The V-22 Osprey tilt-rotor aircraft is another example: over the course of its 25-year development cycle, the program was fraught with safety, reliability, and affordability challenges, resulting in numerous flight test crashes with concomitant loss of crew and passenger lives [2]. More recently, the Boeing 787 Dreamliner transport aircraft program has experienced a number of major prototype subsystem test failures, causing budget overruns of billions of dollars and service introduction delays of about three years. One major source of blame for Boeing's setbacks is its aggressive strategy to outsource component design and assembly, which created heavy program management burdens and led to unforeseen challenges during vehicle integration [3].

In these cases and numerous others, the design program was unaware of the mounting risks in the system, and was surprised by one or more unfortunate outcomes. Although these examples are extreme, they are suggestive that current system design practices are unable to recognize performance, cost, and schedule risks as they emerge. Such unanticipated or emergent behavior is often attributed to the complexity of the underlying system [4]. This has led to a desire to measure system complexity in a manner that will enable design trades and improve parameterization of cost and schedule. Thus, our objectives are to quantitatively define system complexity in terms of system quantities of interest and to formulate a complexity-based sensitivity analysis. The resulting methodology identifies the key contributors to system complexity and provides quantitative guidance for resource allocation decisions aimed at reducing system complexity.

We define system complexity as the potential for a system to exhibit unexpected behavior in the quantities of interest. A background discussion on complexity metrics, uncertainty sources in complex systems, and related work presented in Section 3.0 We measure this complexity as the exponential information entropy of the probability distribution of the quantities of interest associated with a given system. Exponential entropy has been established by Ref. [5] as a rigorous measure of the extent of a probability distribution and is described in more detail in Section 3.0, which also includes the development of our sensitivity analysis procedure, which may be used to direct a design refinement process [6]. We apply our methodology to a design of an infantry fighting vehicle (IFV). The quantity of interest for the application is the range of the vehicle. The application is described in more detail in Section 4.0. A demonstration of the use of our methodology is presented in Section 4.0 as well, where two IFV options are considered and general conclusions are drawn in Section 5.0.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

Complexity in system design is an elusive concept for which many definitions have been proposed, though none formally adopted. Early work in the field of complexity science by Warren Weaver posited complexity as the nebulous middle ground between order and chaos, a region in which problems require "dealing simultaneously with a sizeable number of factors which are interrelated in an organic whole" [7]. Another interpretation of this idea considers a set of "phase transitions" during which the fundamental features of a system undergo drastic changes [8]. As an illustrative example, consider the phase transitions of water [9]. On one end of the spectrum, water is frozen into a simple lattice of molecules whose structure and behavior are straightforward to understand. At the other extreme, water in gaseous form consists of millions of molecules vibrating at random, and the study of such a system requires methods of statistical mechanics or probability theory [10]. In between the two lies the complex liquid state, wherein water molecules behave in a manner neither orderly nor chaotic, but at once enigmatic and mesmerizing, which has captured the imagination of fluid dynamists past and present.

Though the above example makes the idea of complexity relatable to a large audience, the debate over its definition still persists. However, many researchers agree that there are several properties that complex systems tend to share [11, 12, 13, 14, 15] (1) they consist of many parts; (2) there are many interactions among the parts; (3) the whole exceeds the sum of the parts, that is, the parts in combination produce synergistic effects that are not easily predicted and may often be novel, unexpected, even surprising; and (4) they are difficult to model and to understand.

In addition to qualitative descriptions of complexity, there have also been many attempts to explain complexity using quantitative measures. These definitions can be classified into two general categories, structure-based metrics and process-based metrics. Structure-based metrics quantify the complexity associated with the physical representation of a system [16]. They typically involve counting strategies: in software engineering, the source lines of code (SLOC) can be used to describe a computer program [17]; in mechanical design, analogous measures include the number of parts [18], functions [19], or core technologies [20] embodied in a product. Though appealing, these counting metrics may be susceptible to different interpretations of what constitutes a distinct component - depending on the level of abstraction; a component may be as high-level as an entire subsystem, or as basic as the nuts and bolts holding it together. More sophisticated structure-based metrics also attempt to address the issue of component interactions through an analysis of the topology and connectivity of the system [21, 22]. For example, Thomas J. McCabe proposed the idea of cyclomatic complexity in software engineering, which uses graph theory to determine the number of control paths through a module [23]. Numerous others have also recommended approaches to estimate system complexity by characterizing the number, extent, and nature of component interactions, which govern the interconnectedness and solvability of the system [24, 25, 26]. Overall, structure-based complexity metrics are usually easy to understand and to implement, but they may not be meaningful except in the later stages of design, after most design decisions have been made, and the system is well-characterized [27].

A second class of complexity metrics quantifies system uncertainty in terms of processes required to realize the system. One metric in this category is algorithmic complexity, or

Kolmogorov complexity, which measures the compactness of an algorithm needed to specify a particular message [28, 29, 30]. Similar definitions include the number of basic operations required to solve a problem (computational complexity), or the amount of effort necessary to design, modify, manufacture, or assemble a product [31, 32]. Another possible interpretation of complexity is related to the information content of a system. The concept of information entropy was originally proposed by Claude E. Shannon to study lossless compression schemes for communication systems [33]. Information entropy, or Shannon entropy, measures the uncertainty associated with a random variable. It also has an intuitive and appealing analogy to entropy in the thermodynamic sense, as a measure of a system's tendency toward disorder [34]. In this work, we propose a complexity metric based on exponential information entropy, which is described in the next section. It is important to note here that there are many different metrics of complexity and each can be useful in different ways and thus, all are important.

We intend our complexity metric to be used in simulation-based design activities where limited information is known about quantities of interest relevant to the design of a complex system. Given our context, our metric is based on the information content in our estimates of quantities of interest. Thus, our metric reflects a correspondence between uncertainty in a system and the complexity of the system, as consistent with our complexity definition stated in Section 2. This correspondence does not exist for many of the other complexity metrics noted, particularly the structure-based metrics.

3.1 Complexity Metric Theoretical Development

In this section we define our complexity metric and develop a quantitative measure of it. We then develop a sensitivity analysis procedure designed to identify the key contributors to system complexity in an effort to identify how to best allocate resources for complexity reduction.

3.1.1 Complexity Metric.

We define complexity as the potential of a system to exhibit unexpected behavior in the quantities of interest, which are the quantities characterizing the performance, cost, schedule and other relevant attributes of the system. Thus, we wish to characterize the amount of knowledge we have with respect to our quantities of interest. To measure this amount of knowledge, or level of information, we define a metric of complexity based on exponential information entropy. For a discrete random variable Y with probability mass function $p(y)$, the information entropy of Y is defined as

$$H(Y) = -\sum_i p(y_i) \log p(y_i), \quad (1)$$

Where $y_1, y_2, \ldots$ are values of y such that $p(y) \neq 0$. For a continuous random variable X with probability density function $f_X(x)$, the differential information entropy of X is defined as

$$h(X) = -\int_{-\infty}^{\infty} f_X(x) \log f_X(x) dx, \quad (2)$$

where the integrand is taken to be zero when $f_X(x) = 0$. Our work here focuses on continuous random variables. For both the discrete and continuous case the base of the logarithm is chosen by the user. We will deal exclusively in this work with the natural logarithm. Thus, our quantitative metric of system complexity is given as

$$C(Q) = \exp\left\{-\int_{-\infty}^{\infty} f_Q(q) \ln f_Q(q) dq\right\} \quad (3)$$

Where Q is the random variable associated with a quantity of interest of a given system.

The exponential entropy of a uniform random variable can be interpreted as the length of the support of the random variable (and area, volume, and hyper volume for 2, 3, and n-dimensional jointly distributed uniform random variables). To this end, the exponential entropy of any arbitrarily distributed random variable can be related to the length of the support of an information-entropy-equivalent uniform distribution. In this sense it has some similarities to Kolmogorov complexity.

3.1.2 Background Material on Information Entropy. Here we present some brief background material on information entropy. For the discrete case, consider a random variable Y with probability mass function p(y). The entropy of Y is then defined as

$$H(Y) = -\sum_i p(y_i) \log p(y_i), \tag{4}$$

Where $y_1, y_2, \ldots$ are the values of y such that p(y) does not equal zero. For the continuous case, consider a random variable X with probability density function $f_X(x)$. The differential entropy of X is then defined as

$$h(X) = -\int_{\Omega_X} f_X(x) \log f_X(x) dx, \tag{5}$$

Where Ω_X is the support of X. Examples of the information entropy for typical distributions are as follows:

Normal Distribution: $$h(N(\mu, \sigma^2)) = \frac{1}{2} \ln(2\pi e \sigma^2), \tag{6}$$

Uniform Distribution: $$h(U[a,b]) = \ln(b-a), \tag{7}$$

Triangular Distribution: $$h(T(a,b,c)) = \frac{1}{2} + \ln\left(\frac{b-a}{2}\right), \tag{8}$$

where μ is the mean and σ^2 is the variance of the normal distribution, a is the minimum and b is the maximum of the uniform distribution, and a is the minimum, b is the maximum, and c is the mode of the triangular distribution.

3.1.3 Complexity Estimation. Defining complexity in terms of exponential entropy implies that we are concerned with uncertainty associated with quantities of interest. In modeling a potential system, which is typically done with numerical simulation models, there are many potential sources of uncertainty that can impact quantities of interest, and thus system complexity. Among these are parametric uncertainty, parametric variability, code uncertainty, observation error, and model inadequacy. Following Ref. [35], parametric uncertainty refers to uncertain inputs or parameters of a model, parametric variability refers to uncontrolled or unspecified conditions in inputs or parameters, code uncertainty refers to the uncertainty associated with not knowing the output of a computer model given any particular configuration until the code is run, observation error is uncertainty associated with actual observations and measurements, and model inadequacy relates to the fact that no model is perfect. For the application considered here we do not incorporate experimental data, therefore, our focus is on parametric variability, parametric uncertainty, code uncertainty, and model inadequacy.

A simulation model, or simulator, is a function $g(\cdot)$ that maps inputs $\mathbf{x}$ into an output $q = g(\mathbf{x})$. In our work, we incorporate the presence of simulator model inadequacy by adding noise to simulator output. Thus, the true value of a quantity of interest that has been estimated by a simulator is in the form

$$q = g(\mathbf{x}) + \epsilon(\mathbf{x}), \tag{9}$$

where $\epsilon(\mathbf{x})$ is additive noise that is permitted to vary throughout the input space. In the demonstrations provided in Section 4, we notionally account for model inadequacy by assuming normally distributed noise. The purpose of this is to ensure that we are taking into account some form of model inadequacy in the complexity estimation process and the sensitivity analysis methodology. However, our approach does not require that the model inadequacy term be normally distributed. The need to quantify model inadequacy in simulation models was originally addressed in Ref. [35]. More general approaches to the quantification of model inadequacy that incorporate both data and expert opinion is an important topic of future work.

When analyzing quantities of interest with computer models, it is often necessary to approximate the input/output relationships of expensive simulators using less expensive surrogate models. For this, we employ the well-known technique of Gaussian process regression [36, 37, 38, 39]. Gaussian process regression is based on emulating a simulator with a stochastic process model. Emulating with a stochastic process ensures there is a complete statistical approximation of the simulator, which enables the code uncertainty associated with the use of the emulator in place of the simulator to be quantified. This is essential for situations where the code uncertainty of the emulator is a key driver of complexity.

When using an emulator, the true value of a quantity of interest is in the form

$$q = G(\mathbf{x}) + \epsilon(\mathbf{x}), \tag{10}$$

Where $G \sim gp(m(\mathbf{x}), k(\mathbf{x},\mathbf{x}'))$, $m(\mathbf{x})$ is the mean function of the Gaussian process $gp(\cdot,\cdot)$, and $k(\mathbf{x},\mathbf{x}')$ is the covariance kernel of the Gaussian process. A Gaussian process emulator is built with a set of training runs of the simulation model. This training set is treated as data that are used to estimate the simulation model. An example of one-dimensional Gaussian process regression is shown graphically in Figure 36, where three data points from a simulator have been used as training points for the emulator.

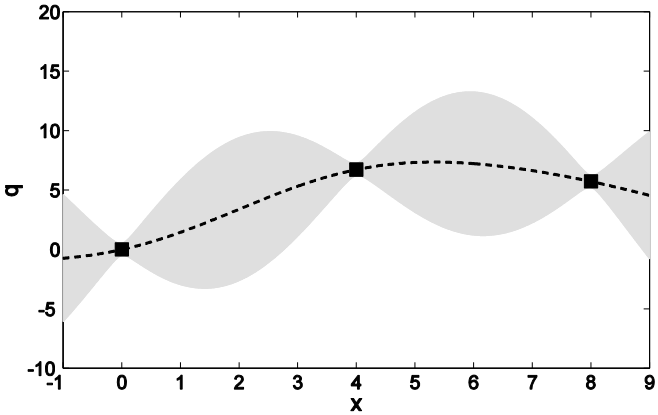


FIGURE 36. EXAMPLE OF GAUSSIAN PROCESS EMULATION WITH THREE TRAINING POINTS.

The emulator itself is a stochastic process, which is represented on the figure as a mean function (dashed line) and plus and minus two standard deviation bounds (grayed area). The grayed area is a representation of the code uncertainty associated with the use of this emulator in place of the underlying simulator. The fitting of such an emulator is a machine learning task that involves the estimation of several hyperparameters. Details on how this may be accomplished can be found in Ref. [38].

To estimate complexity with respect to a quantity of interest, we require an estimate of the probability density function of the quantity of interest. We estimate this using Monte Carlo simulation followed by kernel density estimation. We then discretize this density to estimate the entropy given in Equation 5. For situations where an emulator must be used in place of a simulator to compute quantities of interest, the complexity estimate must also account for code uncertainty. In this case, the procedure described in the preceding paragraph is conducted for each sample of the emulator stochastic process. To be conservative we take the maximum complexity estimate of the emulator samples as the overall complexity.

3.1.3 Sensitivity Analysis. For situations where the system complexity is large, it is desirable to identify factors of the system, which include inputs, parameters, components, subsystems, simulators, and emulators that are the largest contributors to the complexity.
Thus, we have developed a rigorous sensitivity analysis procedure for identifying the most significant factor contributors to the system complexity associated with the quantities of interest.

The approach taken here is similar to that of variance-based sensitivity analysis as described in Ref. 41. In the variance-based case the goal is to apportion the variance of a quantity of interest

among its various factor contributors. This apportionment is based on the law of total variance, which for a given quantity of interest Q and a given factor X_i is written as

$$var(Q) = E[var(Q|X_i)] + var(E[Q|X_i]). \quad (11)$$

From this, a main effect sensitivity index, S_i, for factor X_i can be written as

$$S_i = \frac{var(E[Q|X_i])}{var(Q)}, \quad (12)$$

Which is the expected fraction of the variance of Q that is removed if the true value of X_i was known. In analogous fashion, we consider the expected complexity of the system that would remain if the true value of some factor X_i was known. This quantity is given as $E[C(Q|X_i)]$, where the random variable associated with the quantity of interest for the system is Q. Thus, to identify the expected fraction of complexity that can be removed if the true value of a given factor X_i is known, we define complexity-based sensitivity indices as

$$\eta_i = \frac{C(Q) - E[C(Q|X_i)]}{C(Q)}, \quad (13)$$

Where here the uncertainty associated with X_i is attributable to either parametric variability or parametric uncertainty.

The information gained from our sensitivity analysis procedure can be used as part of a resource allocation strategy aimed at reducing system complexity. It is important to note here that the system complexity we are referring to is that of our proposed definition based on the potential for unexpected behavior. For other definitions of system complexity different means should be taken for complexity reduction. For example, if structural complexity is a concern for particular design, increased modularity could be a viable means for complexity reduction. In this work, we deal exclusively with our proposed definition, and hence aim to increase knowledge of the system quantities of interest via identification of key sources of uncertainty in the system.

4.0 RESULTS AND DISCUSSION

We demonstrate the use of the complexity metric and sensitivity analysis developed in Section 3.0 on a simulation-based design of an infantry fighting vehicle using models developed at Vanderbilt University. The quantity of interest for this demonstration is the range of the vehicle.

4.1 IFV Simulation Emulators

A single simulation of an IFV design for the range calculation takes approximately 1500 seconds on a standard laptop computer. The estimation of the complexity metric and the subsequent sensitivity analysis involves the estimation of several potentially high dimensional integrals, which could require thousands of function evaluations if Monte Carlo simulation is employed.

Thus, for the IFV application, we wish to generate Gaussian process models of the candidate IFV design to emulate the simulation of the vehicle. The Gaussian process model of the potential IFV design constructed here is shown in Figure 37. The Gaussian process was trained with 20 training points from the bond graph simulation model.

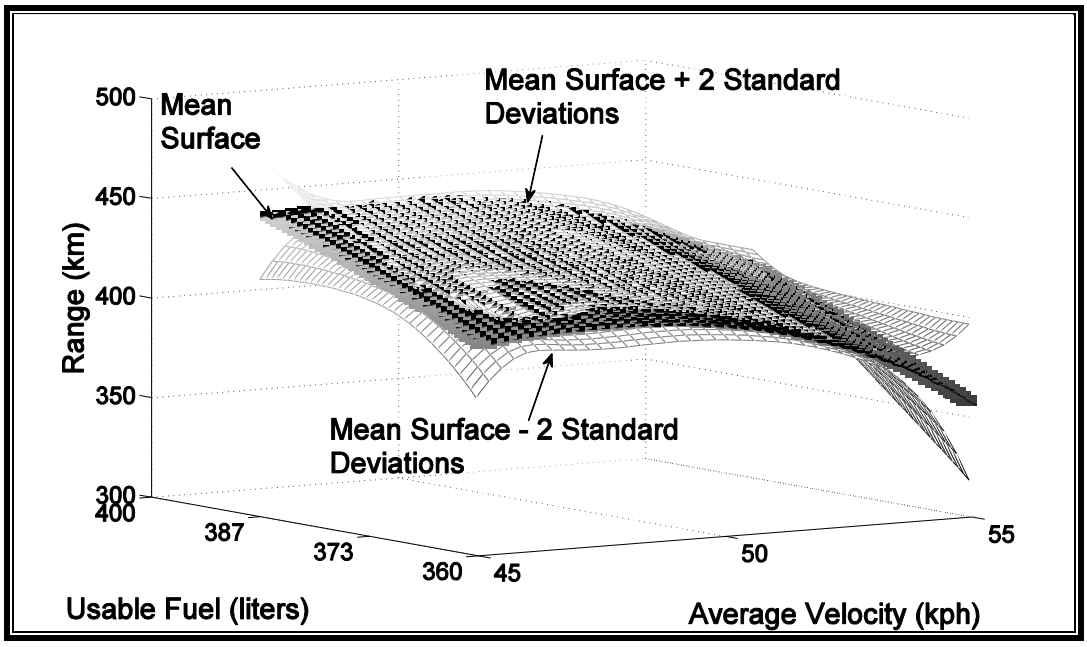


FIGURE 37: EMULATOR OF A CANDIDATE IFV DESIGN

4.2 IFV Sources of Uncertainty

As noted previously, there are many sources of uncertainty that affect estimates of quantities of interest for a complex system. For the IFV range application, we are considering parametric uncertainty, parametric variability, code uncertainty, and model inadequacy. Thus, for the stages of complex system design that involve computer simulation models, we have included all sources of uncertainty.

The parametric uncertainty we consider here is the result of an uncertain amount of trapped fuel that cannot be used by the IFV. The uncertainty in the amount of trapped fuel is captured by considering the available fuel at the beginning of the mission to be uniformly distributed from 360 to 400 liters. Thus, we are assuming between 0 and 10% of the fuel will be unusable. In general, such information should be obtained from expert opinion or historical data [42]. Here we have assigned the distribution for demonstration purposes only. The parametric variability we consider here is the result of different possible human operators of the IFV driving at different speeds. We assume that each operator is attempting to operate the tank at 50 kph, however, each operator may be more or less skilled at achieving this objective. To account for this, we allow the target velocity of the vehicle to be uniformly distributed between 45 and 55 kph. If this

uncertainty is found to be a major contributor to complexity, an obvious next step in the design process is to ensure adequate feedback information to the operator to ensure the operator is capable of maintaining the vehicle at the target velocity. The model inadequacy we consider here is assumed to be normally distributed with mean 0 and a standard deviation of 10 km. This uncertainty is added to the output of the emulator. We have assumed that the model inadequacy is constant throughout the input space. The code uncertainty we consider here is captured by the variability between training points in the Gaussian process model. There are of course many other parameters that would be uncertain at an early stage of the design of a complex vehicle such as the IFV considered here. However, our goal is to demonstrate our methodology rather than perform a complete complexity analysis of the IFV design.

4.3 Complexity Estimation

Following the procedure outlined in Section 3, we estimate the complexity of the IFV design. The result is a complexity of 104 km. Distributions of the range of the IFV is shown in Figure 38. Here, two distributions are shown in solid black lines that were estimated using two different samples of the Gaussian process emulator shown in Figure 37. The dashed gray lines are the output distributions from the same two samples of the Gaussian process emulator, however, for these distributions, model inadequacy has been included.

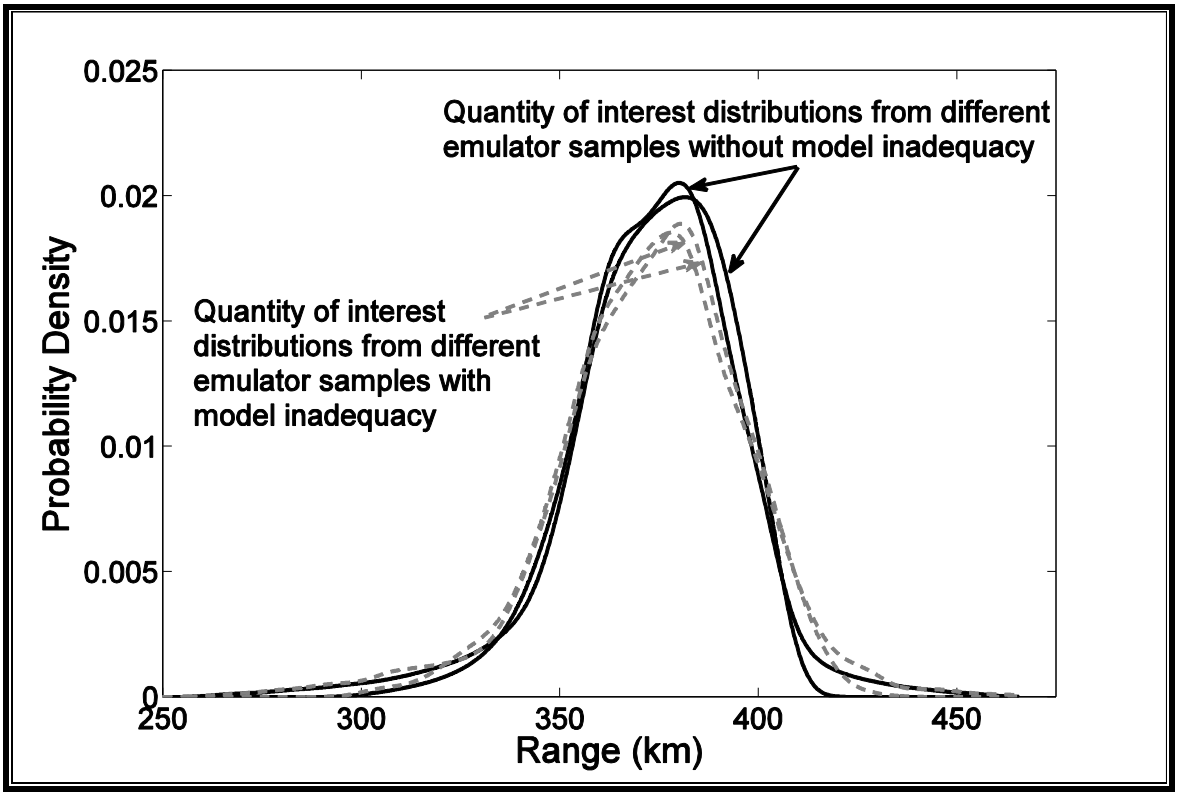


FIGURE 38: PROBABILITY DENSITY FUNCTIONS OF THE RANGE OF THE IFV

4.4 IFV Sensitivity Analysis

Following the procedure outlined in Section 3, we estimate the sensitivity indices of the average velocity, usable fuel, model inadequacy, and code uncertainty with respect to the quantity of interest, IFV range. The results of the sensitivity analysis are shown in Figure 39. As shown on the figure, the sensitivity indices are $\eta_{AV} = 0.46, \eta_{UF} = 0.44, \eta_{MI} = 0.15, \eta_{CU} = 0.16$ for the average velocity, usable fuel, model inadequacy, and code uncertainty respectively.

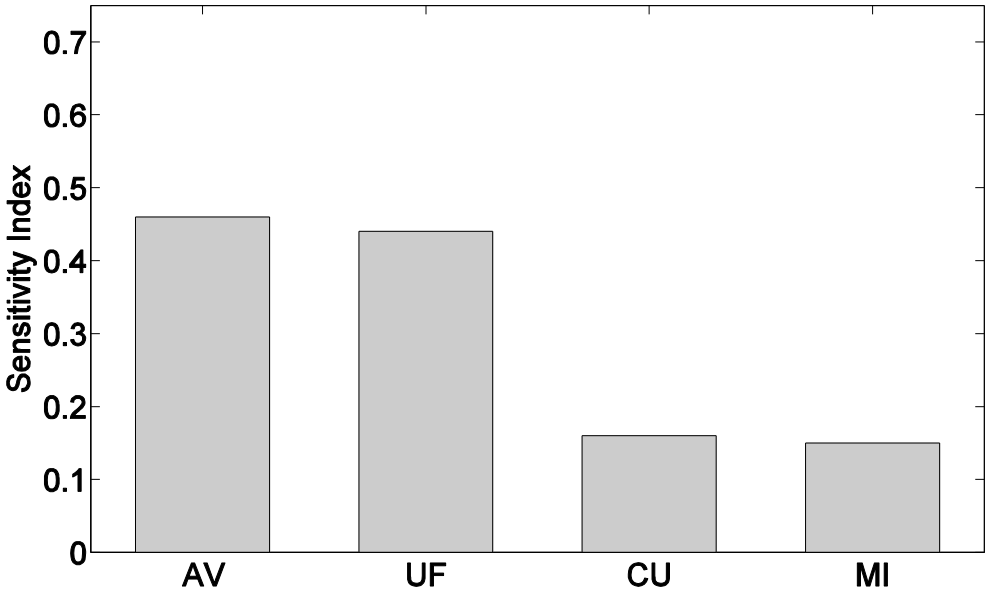


FIGURE 39: SENSITIVITY INDICES FOR IFV EXAMPLE

5.0 CONCLUSION

We have developed an demonstrated a methodology for estimating system complexity with respect to quantities of interest, as well as estimating sensitivity indices designed to indicate key contributors to system complexity. Our complexity metric can be used to compare and rank different candidate designs of complex systems with respect to quantities of interest. In situations where designs are too complex, our sensitivity analysis methodology can be used to identify key contributors to the complexity, which may then be used to inform a resource allocation process. The incorporation of model inadequacy in our approach ensures that complexity arising from the use of low fidelity models be accounted for, and provides direction, in a resource sense, for a multifidelity approach to complex system design. The incorporation of code uncertainty ensures that uncertainty associated with the use of inexpensive surrogate models be accounted for, and

the sensitivity index associated with code uncertainty can potentially be used in the future as part of an adaptive approach to train the emulators. The work described here assumed the existence of quantified uncertainty in the form of parametric variability, parametric uncertainty, model inadequacy, and code uncertainty. In general, it is critical in the design of complex systems that these uncertainties be rigorously quantified. Systematic methods for achieving this goal are an important topic of future work. Once such methods exist, the use of metrics such as the complexity metric described here, as well as the sensitivity analysis developed here, can be used as part of a design verification strategy aimed at producing probabilistic certificates of correctness for designs through simulation.

6.0 REFERENCES

1. Allen, L., Angel, R., Mangus, J. D., Rodney, G. A., Shannon, R. R., and Spoelhof, C. P., 1990. The Hubble Space Telescope optical systems failure report. Tech. Rep. NASA-TM-103443, National Aeronautics and Space Administration.

2. Bolkcom, C., 2005. V-22 Osprey tilt-rotor aircraft: Congressional Research Service Report for Congress. Tech. rep., Congressional Research Service, Jan. 7.

3. Hiltzik, M., 2011. "787 Dreamliner teaches Boeing costly lesson on outsourcing". Los Angeles Times, Feb. 15.

4. Brown, O., and Eremenko, P., 2006. The value proposition for fractionated space architectures. AIAA Space 2006, San Jose, CA, Paper No. AIAA-2006-7506.

5. Campbell, L., 1966. "Exponential entropy as a measure of extent of a distribution". Z. Wahrscheinlichkeitstheorie verw. Geb., 5, pp. 217–225.

6. Takeda, H., Veerkamp, P., Tomiyama, T., and Yoshikawa, H., 1990. "Modeling design processes". AI Magazine, 11(4), pp. 37–48.

7. Weaver, W., 1948. "Science and complexity". American Scientist, 36(536).

8. Huberman, B. A., and Hogg, T., 1987. "Phase transitions in artificial intelligence systems". Artificial Intelligence, 33, pp. 155–171.

9. Johnson, G., 1997. "Researchers on complexity ponder what it's all about". The New York Times: Technology, May 6.

10. Balestrini-Robinson, S., 2009. "A modeling process to understand complex system architectures". PhD thesis, Georgia Institute of Technology, Atlanta, GA.

11. von Bertalanffy, L., 1950. "An outline of general system theory". The British Journal for the Philosophy of Science, 1(2), pp. 134–165.

12. Eriksson, D., 1997. “A principal exposition of Jean-Louis Le Moigne’s systemic theory”. Cybernetics & Human Knowing, 4(2).

13. Corning, P. A., 1998. “Complexity is just a word!”. Technological Forecasting and Social Change, 59, pp. 197–200.

14. Bankes, S. C., 2002. “Tools and techniques for developing policies for complex and uncertain systems”. In Proceedings of the National Academy of Sciences of the United States of America, Vol. 99, pp. 7263–7266.

15. Summers, J. D., and Shah, J. J., 2010. “Mechanical engineering design complexity metrics: Size, coupling, and solvability”. Journal of Mechanical Design, 132(2), pp. 021004–1–11.

16. Braha, D., and Maimon, O., 1998. “The measurement of a design structural and functional complexity”. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 28(4), pp. 527–535.

17. Albrecht, A. J., and J. E. Gaffney, J., 1983. “Software function, source lines of code, and development effort prediction: A software science validation”. IEEE Transactions on Software Engineering, SE-9(6), pp. 639 – 648.

18. Murmann, P. A., 1994. “Expected development time reductions in the German mechanical engineering industry”. Journal of Product Innovation Management, 11(3), pp. 236 – 252.

19. Griffin, A., 1997. “The effect of project and process characteristics on product development cycle time”. Journal of Marketing Research, 34(1), pp. 24–35.

20. Meyer, M. H., and Utterback, J. M., 1995. “Product development cycle time and commercial success”. IEEE Transactions on Engineering Management, 42(4), pp. 297–304.

21. Braha, D., and Bar-Yam, Y., 2004. “Topology of large-scale engineering problem-solving networks”. Physical Review, 69(1), pp. 016113–1–016113–7.

22. Braha, D., and Bar-Yam, Y., 2007. “The statistical mechanics of complex product development: Empirical and analytical results”. Management Science, 53(7), pp. 1127–1145.

23. McCabe, T. J., 1976. “A complexity measure”. IEEE Transactions on Software Engineering, 2(4), pp. 308–320.

24. Tatikonda, M. V., and Rosenthal, S. R., 2000. “Technology novelty, project complexity, and product development project execution success: A deeper look at task uncertainty in product innovation”. IEEE Transactions on Engineering Management, 47(1), pp. 74 –87.

25. Novak, S., and Eppinger, S. D., 2004. “Sourcing by design: Product complexity and the supply chain”. Management Science, 47(1), pp. 189–204.

26. Kim, J., and Wilemon, D., 2009. “An empirical investigation of complexity and its management in new product development”. Technology Analysis & Strategic Management, 21(4), pp. 547–564.

27. Holtta, K. M. M., and Otto, K. N., 2005. “Incorporating design effort complexity measures in product architectural design and assessment”. Design Studies, 26(5), pp. 463–485.

28. Kolmogorov, A. N., 1965. “Three approaches to the quantitative definition of information”. Problems of Information Transmission, 1(1), pp. 1–7.

29. Chaitin, G. J., 1969. “On the simplicity and speed of programs for computing infinite sets of natural numbers”. Journal of the ACM, 16(3), July, pp. 407–422.

30. Cover, T. M., and Thomas, J. A., 1991. Elements of Information Theory. John Wiley & Sons, Inc., New York, NY.

31. Suh, N. P., 1990. The Principles of Design. Oxford University Press, New York, NY.

32. Rodriguez-Toro, C. A., Tate, S. J., Jared, G. E. M., and Swift, K. G., 2003. “Complexity metrics for design (simplicity+ simplicity = complexity)”. In Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, Vol. 217, pp. 721–725.

33. Shannon, C. E., 1948. “A mathematical theory of communication”. The Bell System Technical Journal, 27, pp. 379–423 and 623–656.

34. Lloyd, S., and Pagels, H., 1988. “Complexity as thermodynamic depth”. Annals of Physics, 188, pp. 186–213.

35. Kennedy, M., and O’Hagan, A., 2001. “Bayesian calibration of computer models”. J.R. Statist. Soc. B, 63(3), pp. 425–464.

36. O’Hagan, A., 1978. “Curve fitting and optimal design for prediction”. Journal of the Royal Statistical Society B, 40, pp. 1–42.

37. O’Hagan, A., 2006. “Bayesian analysis of computer code outputs: A tutorial”. Reliability Engineering & System Safety, 91, pp. 1290–1300.

38. Rasmussen, C., and Williams, K., 2006. Gaussian Processes for Machine Learning. M.I.T. Press, Cambridge, Massachusetts.

39. Sacks, J., Welch, W., Mitchell, T., and Wynn, H., 1989. “Design and analysis of computer experiments”. Statistical Science, 4(4), pp. 409–435.

40. Allaire, D., He, X., Deyst, J., and Willcox, K., 2012. “An Information-Theoretic Metric of System Complexity with Application to Engineering System Design, Journal of Mechanical Design, Vol. 134, 100906-1-100906-10. [DOI: 10.1115/1.4007587]

41. Homma, T., and Saltelli, A., 1996. “Importance measures in global sensitivity analysis of nonlinear models”. Reliability Engineering and System Safety, 52, pp. 1–17.

42. Oakley, J., and O’Hagan, A., 2007. “Uncertainty in prior elicitations: A nonparametric approach”. Biometrika, 94, pp. 427–441.

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

C (Q) complexity of a system with quantity of interest Q
H(Y) information entropy of random variable Y
IFV infantry fighting vehicle
P probability measure
Q quantity of interest
T (a, b, c) triangular distribution with minimum a, maximum b, and mode c
U [a, b] uniform distribution with minimum a, maximum b
d design variable vector
h (Q) differential entropy of the distribution of the quantity of interest Q
p(y) probability mass function of random variable Y
y (**d**) true output
z (**d**) **model** output
F sigma field

Acronym	Description
AVM	Adaptive Vehicle Make
BDD	binary decision diagram
C2M2L	Component, Context, and Manufacturing Library
C2WT	C2 Wind Tunnel
CAD	Computer Aided Design
CyPhyML	Cyber-Physical Modeling Language
DOE	Design of Experiments
DSM	Design Structure Matrix
FANG GV	Fast Adaptable Next-Generation Ground Vehicle
FEA	Finite Element Analysis
GME	Generic Modeling Environment
HBGL	Hybrid Bond Graph Language
HDM	Hybrid Dynamics Model
iFAB	Instant Foundry Adaptive through Bits
MAUF	Multi-Attribute Utility Function
MDAO	Multi-Domain Analysis and Optimization
MI	Master Interpreter
MSL	Modelica Standard Library
NRMM	NATO Reference Mobility Model
PCC	Probabilistic Certificate of Correctness
PDE	pulse detonation engine
PET	Parametric Exploration Tool
PID	proportional-integral-derivative
PTM	phonetically tied mixture
SUT	System under Test
WBS	Work Breakdown Structure