AFRL-RQ-WP-TR-2013-0119

META II: MULTI-MODEL LANGUAGE SUITE FOR CYBER PHYSICAL SYSTEMS

Ted Bapty, Sandeep Neema, and Janos Sztipanovits

Vanderbilt University

MARCH 2013
Final Report

AIR FORCE RESEARCH LABORATORY
AEROSPACE SYSTEMS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

AFRL-RQ-WP-TR-2013-0119 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

*//Signature//
DOUGLAS R. RANEY
Project Engineer
Mechanical and Thermal Systems Branch
Power and Controls Division

//Signature//
JACK VONDRELL, Chief
Mechanical and Thermal Systems Branch
Power and Controls Division
Aerospace Systems Directorate

//Signature//
JOHN NAIRUS, Chief Engineer
Power and Controls Division
Aerospace Systems Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE ***(DD-MM-YY)*** March 2013	**2. REPORT TYPE** Final	**3. DATES COVERED** ***(From - To)*** 30 September 2010 – 31 March 2013

4. TITLE AND SUBTITLE META II: MULTI-MODEL LANGUAGE SUITE FOR CYBER PHYSICAL SYSTEMS	**5a. CONTRACT NUMBER** FA8650-10-C-7075
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62303FF
6. AUTHOR(S) Ted Bapty, Sandeep Neema, and Janos Sztipanovits	**5d. PROJECT NUMBER** 3000
	5e. TASK NUMBER N/A
	5f. WORK UNIT NUMBER Q0MM
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Vanderbilt University Institute for Software Integrated Systems (ISIS) The Division of Sponsored Research, 110 21st Avenue S., Suite 937 Nashville, TN 37203-2416	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Aerospace Systems Directorate Wright-Patterson Air Force Base, OH 45433-7542 Air Force Materiel Command United States Air Force Defense Advanced Research Projects Agency/Tactical Technology Office (DARPA/TTO) 3701 North Fairfax Avenue Arlington, VA 22203-1714	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** DARPA AFRL/RQQM
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RQ-WP-TR-2013-0119

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
PA Case Number: 88ABW-2013-2996; Clearance Date: 20 Jun 2013. This report contains color.

14. ABSTRACT
The automatic vehicle monitoring metaheuristics (AVM META) projects have developed tools for designing cyber physical (or Mechatronic) Systems. These systems are increasingly complex, take much longer to design and build, and are increasingly costlier. The vision of the AVM program is to revolutionize the design methodology.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Douglas R. Raney
Unclassified	Unclassified	Unclassified	SAR	28	**19b. TELEPHONE NUMBER** *(Include Area Code)* N/A

Table of Contents

List of Figures

1.0 INTRODUCTION

The Adaptive Vehicle Make (AVM) program "META" projects have developed tools for designing cyber physical (CPS) (or Mechatronic) systems. Exemplified by modern amphibious and ground military vehicles, these systems are increasingly complex, take much longer to design and build, and are increasingly costlier. The vision of the AVM program is to revolutionize the design methodology of such systems and reduce the design time to one-fifth (1/5) of the traditional systems engineering V methodology (MIL – STD 499).

The META tools realize this vision by advancing a novel design flow geared around the following core concepts:

- Component-Based Design enables design cycle compression by reuse of existing technology and knowledge, encapsulated in integrated and customizable components that can be rapidly used in a design. Components in CPS are heterogeneous, span multiple domains (physical – thermal, mechanical, electrical, fluid, and computational – software, computing platforms), and require multiple models to soundly represent the behavior, geometry, and interfaces, at multiple levels of abstractions. The META Language allows creating multi-model multi-domain representation of CPS components that are composable by design.

  Design Space Construction is supported by the META Language using concepts to represent design choices and parameterized components. These constructs in META enable a designer to systematically engineer a flexible and comprehensive design space for sub-systems and system that can be explored for satisfying product-specific requirements. The design spaces for subsystems and systems are assets that encapsulate design knowledge, which can be reused in a context different for which it was originally created.
- Multi-Scale Design Space Exploration incorporates multiple methods that trade accuracy with computation time for exploring the large design spaces. A combinatorial design space exploration tool (DESERT), rapidly prunes design space using highly scalable constraint satisfaction methods over static properties of components and designs (i.e., weight, power, cost, etc.). Higher fidelity, higher computation time methods such as qualitative reasoning, ODE simulations, are used to further explore and reduce the design space, iteratively converging over to solutions of interest, given a set of requirements.
- Test-benches for Design Evaluation capture requirements in a form that can be automatically evaluated for a system-under-test, using a large set of domain-specific analyses – ranging from hybrid dynamics simulation, software platform timing simulation, geometric parameter evaluation, finite element analysis, probabilistic certificate of correctness, qualitative simulation, among others. The model composition tools included in META operate over defined test-benches and synthesize artifacts necessary for executing analysis in domain tools such as Dymola, Pro-e/Creo, Truetime, Qualitative Envisionment, Abaqus, etc.

The META design flow and tools are built around the META Language, labeled the Cyber Physical Modeling Language (CyPhy). CyPhy is a model integration language which integrates models from different domains in a semantically sound manner that enables reasoning for correctness of models and modeling languages. This report describes the development of the CyPhy and related specifications and tools under the META Language contract (FA8650-10-C-7075).

2.0 PROJECT OVERVIEW

The charter of project, as envisioned in the proposal, was to develop a multi-modeling language suite for design and synthesis of Cyber Physical Systems.

2.1 META Language Enabled META Design Flow

The META design flow (separate contracted effort) articulates a design methodology and the associated tool flow for the CPS system design. The figure below summarizes the key elements of the META design flow, and motivates the key functionality that was needed in META language to enable the design flow.

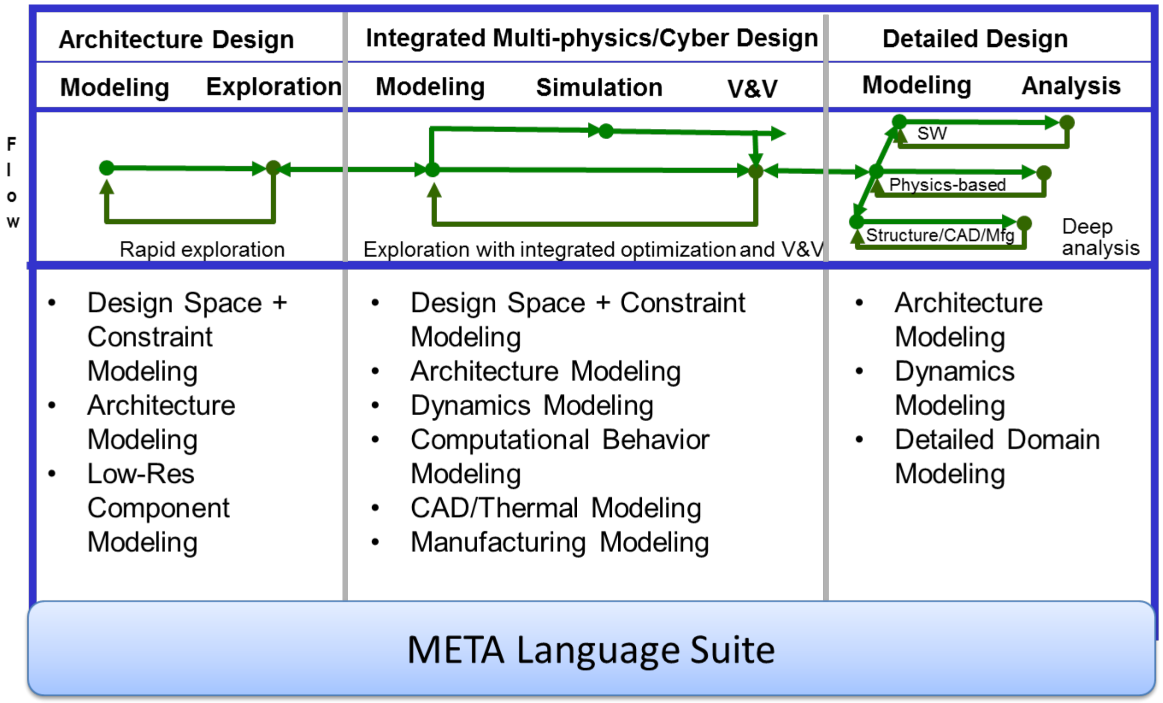


Figure 1: Meta Language Suite

The META design flow involves three core groups of activities:

1. Initial Architecture design involves modeling and rapid Exploration of early design space sketched out with the system requirements. These activities in design flow require that the META language includes concepts for modeling system design space and constraints, enable representing the key architectural variants that can broadly support the customer requirements. The early architecture exploration also requires low compute intensity methods that can allow examination of lots of design options. The META language needs to support modeling low-resolution components to enable coarse grain exploration.

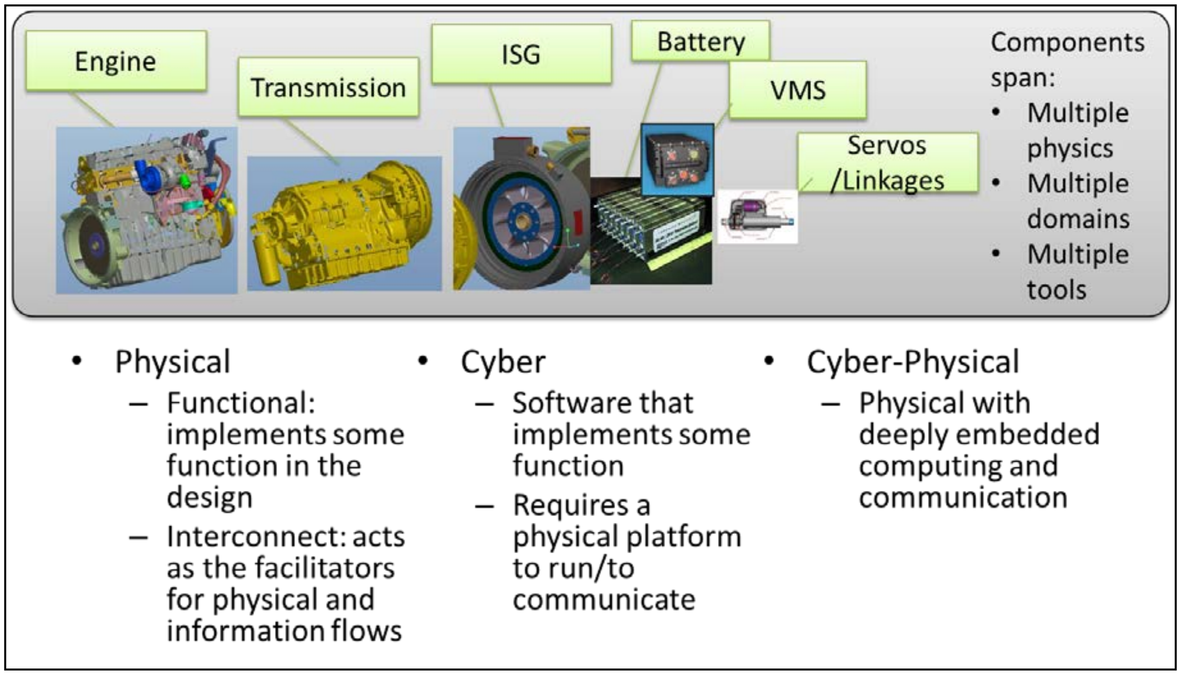
2. The Integrated Multi-Physics and Cyber Design stage expands upon the broadly identified architecture, and refines them with integrated design of physical and cyber

components and conducts relevant tradeoffs. These activities in addition to modeling of the design space and constraints require dynamics modeling for progressively refined performance simulation of the system. This phase also requires support for computational modeling to analyze the behavior and interaction of software with physical components. Geometry and geometry-driven analyses are central to the Physical nature of CPS, and consequently the modeling language suite needs to support CAD and derivative analysis such as thermal, FEA, and allow evaluation of designs for manufacturability.

3. The Detailed Design stage involves further refinement, and analysis, of designs leading towards production, which shifts the emphasis of modeling capabilities from design space to domain model elaboration.

2.2 META Language Multi-Model Multi-Domain Components

In addition to the diversity of the design and modeling activities, the META Language Suite needs to facilitate representation of a diverse range of cyber physical components.



Components span:
- Multiple physics
- Multiple domains
- Multiple tools

- Physical
  - Functional: implements some function in the design
  - Interconnect: acts as the facilitators for physical and information flows
- Cyber
  - Software that implements some function
  - Requires a physical platform to run/to communicate
- Cyber-Physical
  - Physical with deeply embedded computing and communication

Figure 2: Cyber/Physical Components

The components that constitute a typical cyber physical system such as a military ground vehicle span a broad range from commodity physical components such as nuts and bolts, to large complex dynamical components like Engines, Transmissions, Sensors, Actuators, and Controllers. These components can generally be categorized as:

1. Physical – components consist purely of electro-hydro-mechanical elements with little or no programmability. Examples of such components include transmissions, differentials, gears, clutches, starter-generators, servos, among others. These can be further

categorized as: functional – implementing a function in the design, or interconnect – that act as facilitator for physical energy flow or provide linkages such as nuts, bolts, pipes, and tubes.

2. Cyber – components are software components that require a computer processor to run, and implement some function such as the Vehicle Management Software, or controller algorithms implemented in software
3. Cyber-Physical – components cross-cut cyber and physical domains, such that they are physical and implement some function, however, contain deeply embedded computing and communication functions that enable configuration and control of the designed function. Modern combustion engines are a good example of cyber-physical components, in that they include programmable controllers that will interface with rest of the vehicle management system over communication buses (such as Controller Area Network (CAN) and TT/FlexRay), and allow optimizing torque delivery by controlling air/fuel mixture and valve timing for optimal combustion.

Figure 2 depicts the fact that META components span across different energy and physics domains. A combustion engine, for example, turns chemical energy into rotational mechanical energy while a battery delivers electrical energy from stored chemical energy, or an integrated starter generator (ISG) delivers mechanical rotational energy from electrical energy.

Moreover, the components depicted in figure require multiple models to describe and analyze their behavior. A combustion engine has a Computer Aided Design (CAD) model which represents the physical geometry including mass distribution, center-of-gravity, a dynamics model will describe the performance of the engine as a function of the driver and torque demand, a thermal model will describes heat generation, distribution, and dissipation as a function of the driver and torque demand.

Furthermore, these different models are often developed in different domain tools i.e. ProE/CREO or SolidWorks© tools are used for CAD modeling, while Dymola and Simulink© might be used for modeling dynamics. Often these models constitute an asset base of different engineering organizations, and have been developed with significant time and resource investment.

These motivating and constraining factors had a strong impact on the design of the META Language. The META Language had to be designed to represent components that are "Heterogeneous, Multi-Physics, and Multi-Model", in such a way that it could leverage and integrate existing model assets in domain tools.

2.3 META Language a Model Integration Language

The consequence of these factors was that we developed the META Language that we call CyPhy as a *Model Integration Language*. A Model Integration Language is a thin layer wrapping language that wraps the domain models and exports only the key interface and parameters that are relevant for integration. The wrapping maintains the link to the domain model – to allow integration in the domain tool. The integration language has a very small set of native modeling constructs by design. The native construction includes concepts such as hierarchical ported modules and interconnects, structured design spaces, and includes a variety of meta-model composition operators which enables systematic integration across different domain modeling languages.

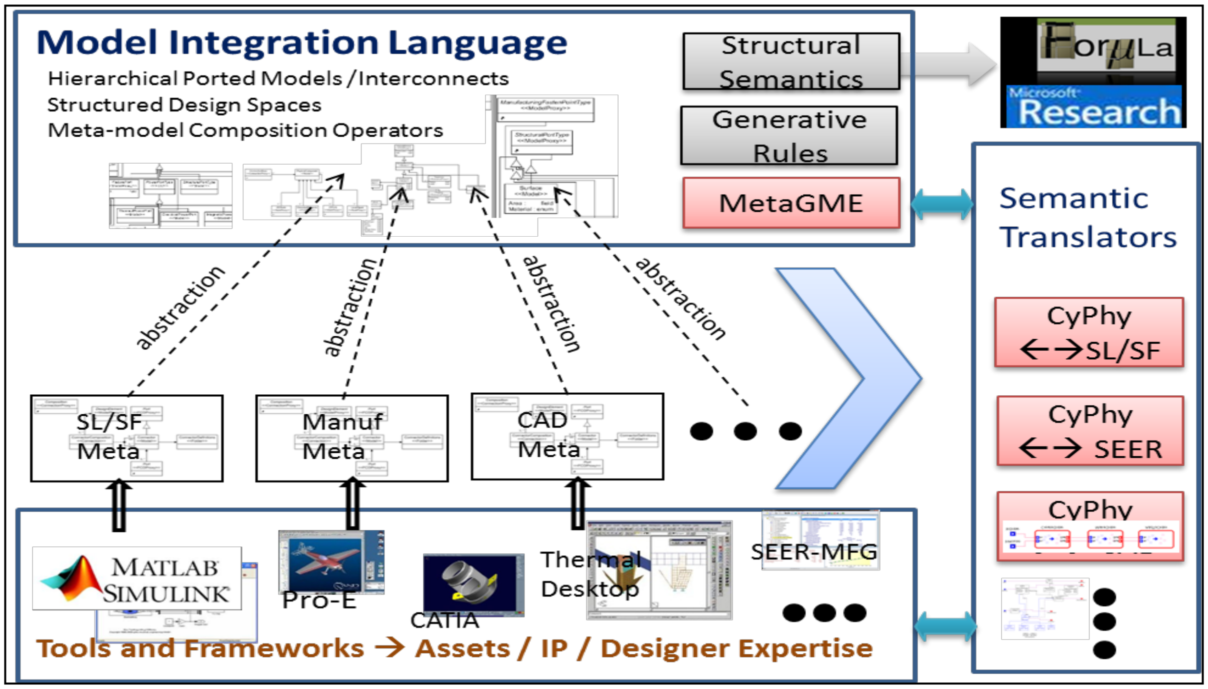


Figure 3: Integration of Domains within Modeling Languages

The integration is done in a manner that abstracts the key properties and interfaces from the domain models that are relevant for integration across domains. These constitute the key variables, or design parameters that must be reasoned about in a multi-domain context. For example, when modeling system architecture the detailed and exact geometry may not be important, however, the key concepts of relevance are the join interfaces, surfaces and constraint with which components must be physically attached to each other. A systematic linkage of the abstractions and modeling concepts automatically enables the projection from architecture models back into the domain models.

The Domain Tools and Frameworks depicted in the figure above are rich engineering infrastructures that were developed with significant investment, and have accumulated a large volume of Design Assets, Intellectual Property, Designer Expertise. The Model Integration Language approach enables reuse of these assets in the form of a META Component Library, and when systems are built using the components, the Model Integration Language approach allows the projection of the integrated models back into to the Domain Tools and Frameworks to analyze, visualize and refine the design.

A model integration language approach also allows opportunistic linking and adds new design languages on demand enabling an open language framework, that allows for adapt languages to accommodate evolving needs of design flows.

2.4 META Language Compositional Semantics

A major challenge in realizing a model integration approach, relates to the heterogeneous semantics of the modeling languages integrated together. The project addresses this challenge by formally specifying the semantics of the integrated domain languages, as well as formally specifying the composition semantics. The figure below illustrates the compositional semantics of integrated the Dynamics Modeling Language with the Architecture Modeling Language (AML).

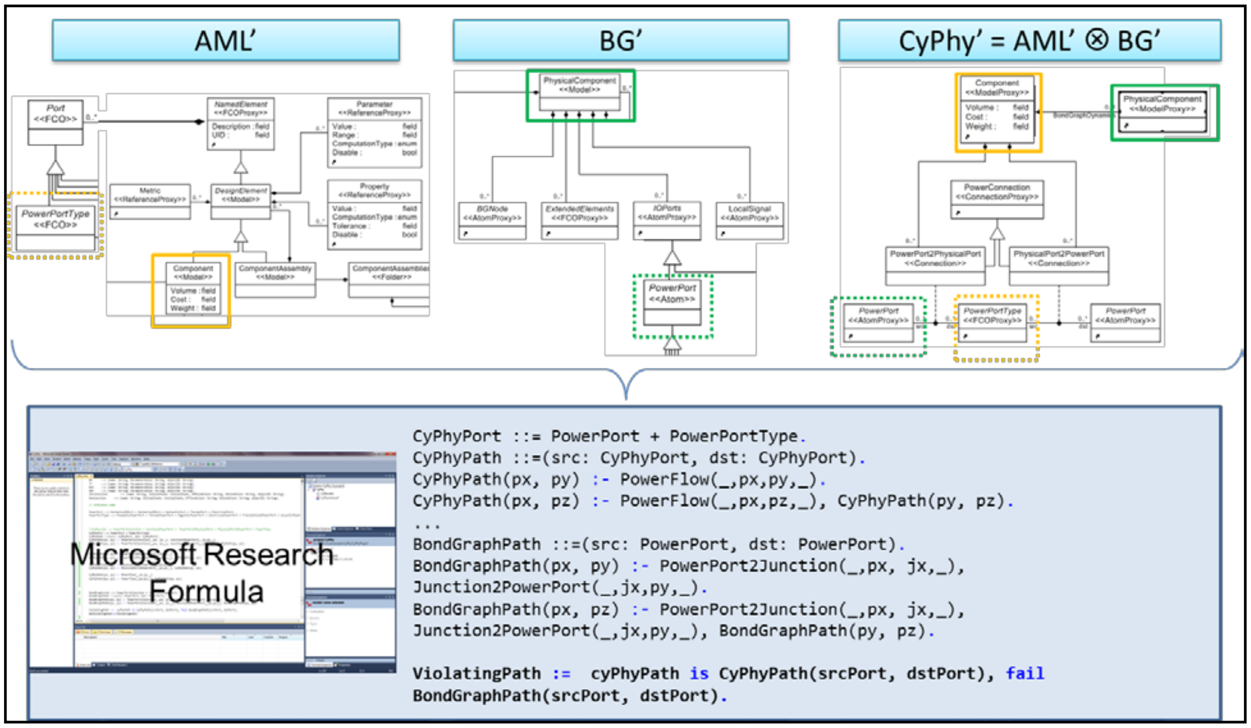
Figure 4 depicts a subset of the meta-model (meta-models are definition of modeling languages using a UML class diagram notation) of the AML. The figure also shows a subset of the Bond Graph language, which is a multi-physics modeling language. The complete Bond Graph language is pretty large and complex, however, for integration with architecture language the core concepts that are relevant for integration are abstracted out. The *PhysicalComponent* is a term in Bond Graph notation referring to a component, and *PowerPort* are the interface of BondGraph components. In the integration language the composition is accomplished as follows:

a) A *PowerPortType* is defined in the AML, and is inherited from the *Port* concept in the AML. This allows for creation of power ports in architecture component model, and

b) a containment relation is established between the AML Components and Bond Graph components, which allows embedding Bond Graph components in architecture components, and

c) The *PowerPort* of Bond Graph is linked with *PowerPortType* in AML

However, while this enables drawing Bond Graphs within Architecture model, several crucial question remains regarding the semantics and well-formed compositions. In the META language project these questions are addressed by use of FORMULA, a constraint logic programming environment, developed at Microsoft Research. The FORMULA tool allows specification and reasoning over well-formed domain composition.


AML'
BG'
CyPhy' = AML' ⊗ BG'
Port <<FCO>>
PowerPortType <<FCO>>
NamedElement <<FCOProxy>>
Parameter <<ReferenceProxy>>
Metric <<ReferenceProxy>>
DesignElement <<Model>>
Property <<ReferenceProxy>>
Component <<Model>>
ComponentAssembly <<Model>>
ComponentAssemblies <<Folder>>
PhysicalComponent <<Model>>
BGNode <<AtomProxy>>
ExtendedElements <<FCOProxy>>
IOPorts <<AtomProxy>>
LocalSignal <<AtomProxy>>
PowerPort <<Atom>>
Component <<ModelProxy>>
PhysicalComponent <<ModelProxy>>
PowerConnection <<ConnectionProxy>>
PowerPort2PhysicalPort <<Connection>>
PhysicalPort2PowerPort <<Connection>>
PowerPort <<AtomProxy>>
PowerPortType <<FCOProxy>>
Microsoft Research Formula

```
CyPhyPort ::= PowerPort + PowerPortType.
CyPhyPath ::=(src: CyPhyPort, dst: CyPhyPort).
CyPhyPath(px, py) :- PowerFlow(_,px,py,_).
CyPhyPath(px, pz) :- PowerFlow(_,px,pz,_), CyPhyPath(py, pz).
...
BondGraphPath ::=(src: PowerPort, dst: PowerPort).
BondGraphPath(px, py) :- PowerPort2Junction(_,px, jx,_),
Junction2PowerPort(_,jx,py,_).
BondGraphPath(px, pz) :- PowerPort2Junction(_,px, jx,_),
Junction2PowerPort(_,jx,py,_), BondGraphPath(py, pz).

ViolatingPath :=  cyPhyPath is CyPhyPath(srcPort, dstPort), fail
BondGraphPath(srcPort, dstPort).
```

Figure 4: A Meta-Model

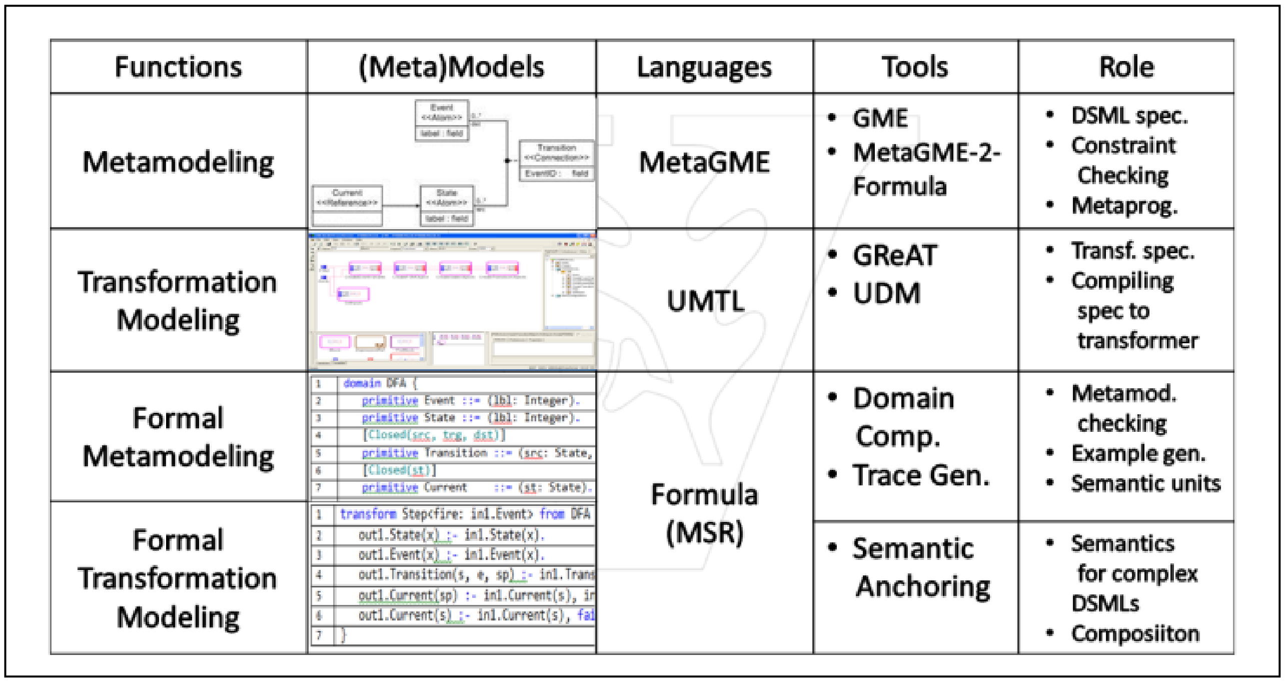
3.0 RESULTS

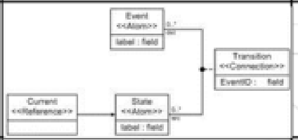
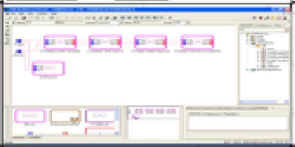
The outcome of this research was the integration of some modeling as well as the development of META modeling suite of software.

3.1 META Semantic Backplane

The Semantic Backplane includes modeling languages, models and tools for the semantic integration of Domain Specific Tool Chain (DSTC) configurations. The semantic integration is performed by

1. *Metamodeling* - defining structural and behavioral semantics of domain specific modeling languages
2. *Metamodel Analysis and Verification* - composing and relating DSTC-level domain specific modeling languages) and
3. *Metagenerators* - automatically generating model translators from formal specification of relationship between modeling languages.



Functions	(Meta)Models	Languages	Tools	Role
Metamodeling		MetaGME	• GME • MetaGME-2-Formula	• DSML spec. • Constraint Checking • Metaprog.
Transformation Modeling		UMTL	• GReAT • UDM	• Transf. spec. • Compiling spec to transformer

Formal Metamodeling	`domain DFA {`
	`primitive Event ::= (lbl: Integer).`
	`primitive State ::= (lbl: Integer).`
	`[Closed(src, trg, dst)]`
	`primitive Transition ::= (src: State,`
	`[Closed(st)]`
	`primitive Current ::= (st: State).`
		Formula (MSR)	• Domain Comp.
			• Trace Gen.
				• Metamod. checking
				• Example gen.
				• Semantic units

Formal Transformation Modeling	`transform Step<fire: in1.Event> from DFA`
	`out1.State(x) :- in1.State(x).`
	`out1.Event(x) :- in1.Event(x).`
	`out1.Transition(s, e, sp) :- in1.Trans`
	`out1.Current(sp) :- in1.Current(s), in`
	`out1.Current(s) :- in1.Current(s), fai`
	`}`
			• Semantic Anchoring	• Semantics for complex DSMLs
				• Compositon

Figure 5: Semantic Backplane Overview

Tools and methods developed for the Semantic Backplane are not targeting the general engineering users: these are for a relatively small group of specialized experts responsible for the semantic integrity of the evolving domain specific tool chains.

An essential element of the Vanderbilt MIC tool suite is that most of the Semantic Backplane tools are "metaprogrammable" and used both in the Semantic Backplane and DSTC levels. In the followings we summarize the delivered components.

Meta-modeling provides the formal specification of the semantics of the META modeling language suite.

3.1.1 Metamodeling Languages

The meta-modeling languages listed below are part of the deliverables. We expect that the meta-modeling languages will continue to evolve beyond this project as an overall consolidation in the practical use cases for semantics. We are also investigating other alternatives such as *BIP* (developed by Joseph Sifakis – 2008 Turing Award Laurate) for capturing interaction semantics among cyber components.

The meta-modeling languages are:

1. *MetaGME++:* the mature meta-modeling language MetaGME (a variant of UML class diagrams and OCL) extended with generative constructs. MetaGME++ is used as meta-modeling language for all meta-programmable tools. It has well established relationship with various standards.
2. *FORMULA:* constraint logic programming language developed by Microsoft Research. FORMULA is used as formal language for defining the structural semantics of *MetaGME++* and domain specific modeling languages defined using *MetaGME++*. (MSR and Vanderbilt ISIS collaborates in evolving FORMULA; e.g. current work expands the logic used in FORMULA with metric first order linear temporal logic).
3. *ASML:* a language variant for the Abstract State Machine (ASM) formal framework. We use ASMs as common semantic domain for specifying discrete behavioral semantics of modeling languages. ASML was selected because of its availability in the Visual Studio tool suite. (We expect that in the future we migrate to FORMULA as the supporting theory evolves). ASML-based behavioral semantics are operational specifications (as opposed to denotational) therefore they are executable and suitable for generating reference traces.
4. *DE:* lumped parameter differential equations as a common denotational semantic domain for a wide range of continuous time dynamics. We use a syntactic form that can be easily transformed. DEs provide a bridge toward symbolic mathematics tools developed for order reduction. The provided semantics for continuous dynamics is independent from simulation algorithms.

3.1.2 Meta-models

Meta-models are models of domain specific modeling languages describing the use of meta-modeling languages. Their goal is to capture the formal structural and behavioral semantics of modeling languages. The Semantic Backplane includes the *CyPhy Meta-model Library* that integrates semantic aspects of a given configuration of the META DSTC.

Being a model integration language, CyPhy includes a core set of language constructs for model and design space integration as well as an evolving suite of abstracted languages imported from various META tools. The abstracted sublanguages are the simplest possible well-formed subsets of the domain specific modeling languages of constituent META tools – still sufficient for capturing cross-domain interactions (structural and behavioral). Abstracting sublanguages for multi-model integration from bloated and complex domain languages is an important step toward making META DSTC-s practical.

At this point, the *CyPhy Metamodel Library* includes metamodels for the following sublanguages:

1. *ADML (Architecture Design Modeling Language)*: represents hierarchical component architectures and typed interfaces. Precise relationship is being defined between ADML and component modeling sublanguages of various standards or frequently used modeling languages, such as SySML (in progress), AADL (planned) and SL/SF. This relationship is defined as model transformation in GReAT (the MIC tool suite graph model transformation specification language) – and in some cases in FORMULA.
2. *Architecture Design Space Modeling Language (ADSML):* extends the design modeling languages with constructs for design space modeling, allowing traditional design languages to capture design spaces instead of just point designs. The extensions come in the form of introducing design containers with model structure variability such as Alternatives, Optional, and variable cardinality containment, as well as Parameterization of design elements. Introduction of these design space extensions at all levels within the design hierarchy provides a powerful and compact mechanism of representing very large design spaces.
3. *Bond Graph Modeling Language (BGML) (Extended):* is a multi-domain (energy and physical) formalism for representing lumped parameter dynamics of physical systems. A Bond Graph represents energy flow across systems in an energy domain neutral manner. Hybrid Bond Graphs are also able to represent hybrid dynamics with the aid of switched junctions, and support derivation of causality relation across systems.

Beyond the core model and design space integration language elements, CyPhy has been complemented with the following abstracted sublanguages imported from integrated tools:

1. *Simulink/Stateflow Interface Language:* The cyber aspects, specifically the controller design, are captured using Simulink/Stateflow models. The CyPhy meta-model integrates an abstracted Simulink/Stateflow meta-model, capturing the input, output, and parametric interface of Simulink models and defines associations with CyPhy components and component interfaces.
2. *Embedded Systems Modeling Interface Language (ESMoL):* The ESMoL language defines software components, computation and communication platform, and allocation

of software components on platform. CyPhy meta-model defines the relation of software components with CyPhy components and allows defining the sensing, actuation, and control path by specifying associations between energy interface of physical components, sensors and actuators with data interface of software components.

3. *CAD Constraint ML:* represents geometrical constraints (axial alignment, surface placement, between CAD components (linked into CyPhy components) and allow derivation of CAD assemblies with a network of geometric constraints

4. *Manufacturing (Cost) ML:* represents manufacturing cost drivers for buy and make parts. These drivers include factors such as parts types, complexity, and counts, join types, complexity, and counts for part assemblies. The Manufacturing ML is integrated within CyPhyML allowing associating manufacturing cost parameters with CyPhy components.

5. *Hydraulics ML (in progress):* is an abstraction of hydraulics systems modeling primitives as used for modeling Hydraulic systems in COTS tools (Boeing ICCA). These abstractions are being linked into the Fluid aspect of the CyPhy component model.

The meta-models are represented in MetaGME++ and translated for verification and validation to FORMULA. (We do not expect the verification step fully completed by the end of September. Rather, we expect that the CyPhy Meta-model Library will continue evolving during the tool maturation period and beyond following the evolution of the META tool chain.)

3.1.3 Meta-modeling Tools

1. *Generic Modeling Environment (GME)*: Vanderbilt's meta-programmable modeling tool is the modeling environment for MetaGME++. Except the newly implemented support for the generative extension of MetaGME, the tool is mature and has been tested in major academic and industrial projects. GME is open source and distributed for research as well as commercial use.

2. *Unified Data Model (UDM)*: is a meta-programmable API tool that provides API-s to programmatically manipulate domain-specific models built using GME (persisted in GME's native format or conformant XML). UDM is open source, has multiple programming language support (Java, C++, .net, Python), is mature and tested in various academic and industrial projects.

3. *The* Graphical modeling environment (and associated toolset) for formally defining/ modeling Model Transformations as Graph Rewriting specification over Domain Meta Models (GReAT). The model transformations defined with GReAT can be interpretively executed for rapid prototyping, or compiled into executable specifications for performance. The formal definition provides opportunities for verifying the transformation, and allows for systematic evolution of the model transformation as the domain meta-models evolve.

3.2 Cyber Physical Modeling Language (CyPhy)

The CyPhy Modeling Language is defined using Meta-models (described earlier). The CyPhy Meta-model is provided as a deliverable of this project. This section documents the core concepts of CyPhy using sections of CyPhy meta-models.

3.2.1 Components

Components in Cyphy are the basic units for composing system design. Components are self-contained models representing a physical or software part of the system. As an atomic component, they are not intended to be further subdivided at the level of representation in CyPhy, but can be used as a standalone part.

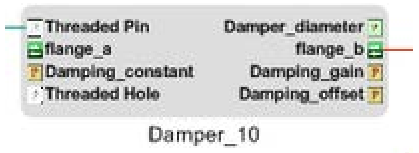


Figure 6: Example Component Model

The component model represents several things about the actual component, including its physical representations and connections, its dynamic behavior, and numerical properties. The component in figure 1 shows several connections for structurally connecting (Threaded Pin & Hole), dynamically connecting (flange_a/b), and parameters (Damping Constant,...). These aspects are: physical implementation, Dynamic and Cyber.

Physical implementation: The component will have a 3D shape, and various physical properties, such as mass, center of gravity, 3D geometry (CAD). As the components are interconnected into *assemblies*, subsystems and systems, the *interfaces* are carefully defined to permit *composition* of models. The physical properties of the model are shown in the *Structural Aspect* of the model.

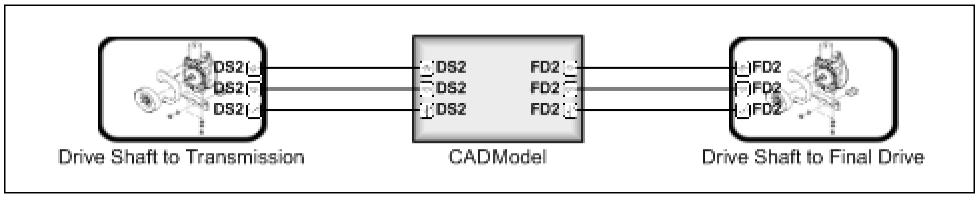


Figure 7: Physical – Inside the Structural Aspect

Dynamics are the component behaviors in one or more domains (e.g., Electrical, Thermal, Mechanical-Rotational, Mechanical-Translational, Hydraulic, etc. Dynamics is expressed in the Modelica language, which uses a mix of Causal (directional input or output is assigned to each port) and Acausal (power flows either direction based on its context, as in most physical systems).

Cyber is the software is a critical part of the cyber-physical system design, with many components having a physical, dynamic, and software implementation. The Cyber aspect captures the software representation. The Cyber aspect is intended primarily for specifying controller logic for the system. Controllers can be specified in a combination of state diagrams and signal flow. Software is automatically generated from these models.

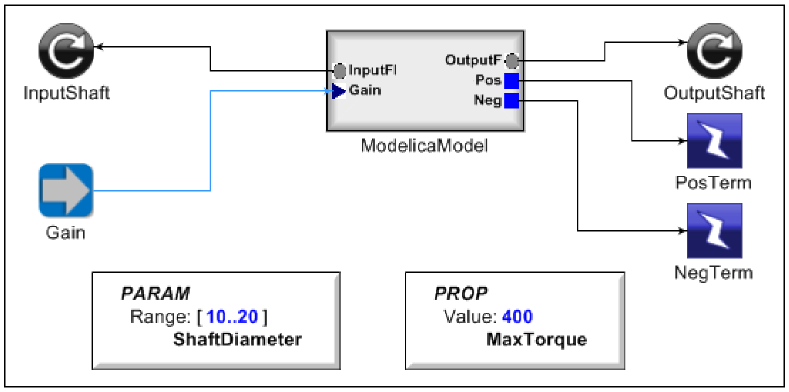


Figure 8: Example Dynamics Model

In summary, components are multi-domain and multi-model, include interfaces for composition, have properties for informational and analytical evaluation, and can be parameterized.

3.2.2 Design Spaces

Using components and assemblies allows the designer to capture a single design architecture, with a single choice of components. This has several drawbacks:

Requirements often change during the design process, sometimes necessitating a redesign.

Component and subsystem behavior is discovered during the design process, and the best choice of architecture and components may not be apparent until late in the design process.

The design is applicable to a single target use, and can require substantial rework for other applications.

Instead, CyPhy/OpenMETA offers the concept of a Design Space. The design space allows the models to contain multiple alternatives for components and assemblies. Any component or assembly can be substituted for another component or assembly with the same interface.

The editor offers a simple syntax for capturing design options. A design alternative container is created with an interface matching a component and the component is placed inside and wired to the external interfaces (there is a tool to automatically do this). Additional alternative components (or assemblies) are added to the alternative design container.

The semantics of this construct are to choose one of the internal components in place of the alternative container.

The design space is the combination of all options of all alternatives. Consequently, the design space can get very large (i.e. Design space size is # Alt1 * # Alt2 * # Alt3 *...). While this is a powerful mechanism to expand the range of designs under consideration, a mechanism is needed to limit the design space to a manageable size. For this purpose, design space constraints can be specified, and used by the Design Space Exploration Tool (DESERT).

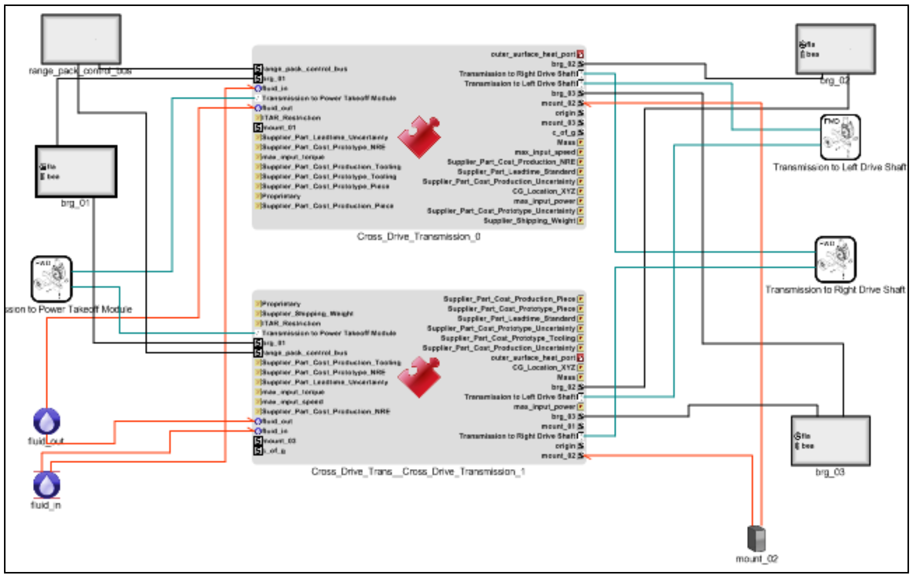

range_pack_control_bus
brg_01
sion to Power Takeoff Module
fluid_out
fluid_in
Cross_Drive_Transmission_0
Cross_Drive_Trans__Cross_Drive_Transmission_1
brg_02
Transmission to Left Drive Shaft
Transmission to Right Drive Shaft
brg_03
mount_02
outer_surface_heat_port
Transmission to Power Takeoff Module
STAR_Restriction
Supplier_Part_Cost_Production_Piece
Supplier_Shipping_Weight
CG_Location_XYZ
Mass
max_input_power
origin

Figure 9: Example Design Space Alternative

Design space constraints are simple, static operations/equations that can be specified on the properties or identities of components or assemblies in the design alternative space. Operations on the properties such as total weight and cost, thresholds on a component property (e.g. TRL > 3), or identity (e.g., all wheel types must match – do not mix and match).

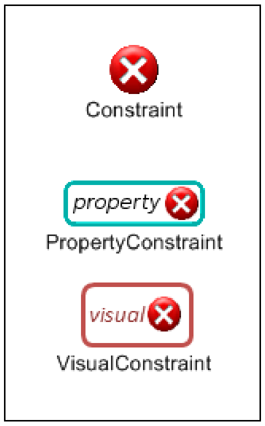
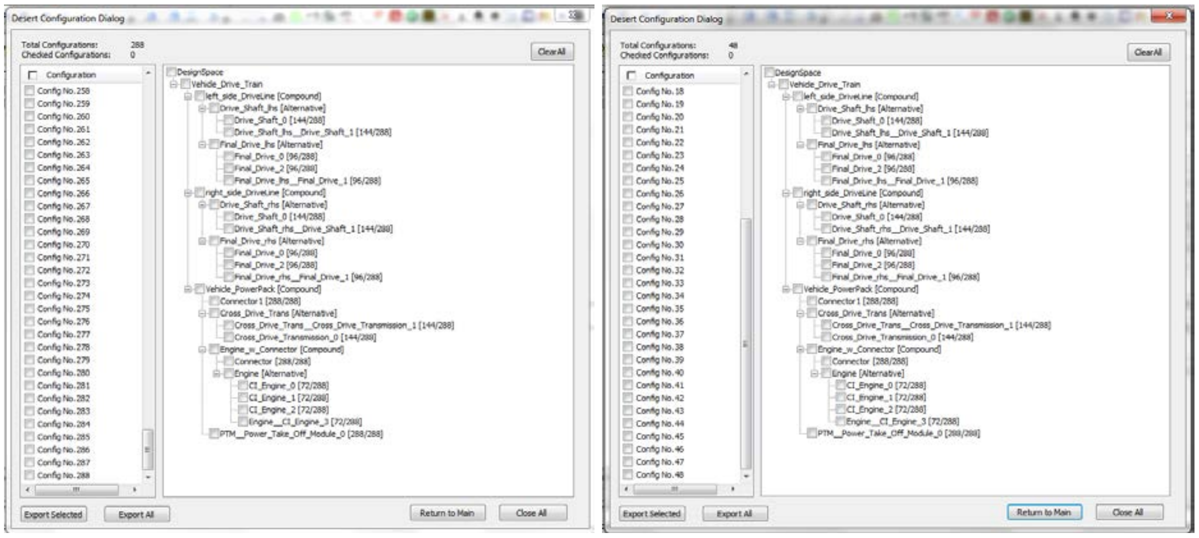


Figure 10: Example Constraints

The DESERT Tool uses scalable techniques to apply these constraints to very large design spaces to rapidly prune the design space to a manageable size. The figure below shows the design space for the simple drivetrain. Prior to applying constraints, there are 288 configurations. After, there are 48. Typical design spaces can easily reach 10B configurations.

After proper constraint application, these can be reduced to 1000s. Design space creation and exploration is a process of expansion and contraction of the design space. It can be a powerful tool to build adaptable, flexible designs.



```
Total Configurations:    288
Checked Configurations:  0

Configuration
Config No.258
Config No.259
Config No.260
Config No.261
Config No.262
Config No.263
Config No.264
Config No.265
Config No.266
Config No.267
Config No.268
Config No.269
Config No.270
Config No.271
Config No.272
Config No.273
Config No.274
Config No.275
Config No.276
Config No.277
Config No.278
Config No.279
Config No.280
Config No.281
Config No.282
Config No.283
Config No.284
Config No.285
Config No.286
Config No.287
Config No.288

DesignSpace
  Vehicle_Drive_Train
    left_side_DriveLine [Compound]
      Drive_Shaft_lhs [Alternative]
        Drive_Shaft_0 [144/288]
        Drive_Shaft_lhs__Drive_Shaft_1 [144/288]
      Final_Drive_lhs [Alternative]
        Final_Drive_0 [96/288]
        Final_Drive_2 [96/288]
        Final_Drive_lhs__Final_Drive_1 [96/288]
    right_side_DriveLine [Compound]
      Drive_Shaft_rhs [Alternative]
        Drive_Shaft_0 [144/288]
        Drive_Shaft_rhs__Drive_Shaft_1 [144/288]
      Final_Drive_rhs [Alternative]
        Final_Drive_0 [96/288]
        Final_Drive_2 [96/288]
        Final_Drive_rhs__Final_Drive_1 [96/288]
    Vehicle_PowerPack [Compound]
      Connector1 [288/288]
      Cross_Drive_Trans [Alternative]
        Cross_Drive_Trans__Cross_Drive_Transmission_1 [144/288]
        Cross_Drive_Transmission_0 [144/288]
      Engine_w_Connector [Compound]
        Connector [288/288]
        Engine [Alternative]
          CI_Engine_0 [72/288]
          CI_Engine_1 [72/288]
          CI_Engine_2 [72/288]
          Engine__CI_Engine_3 [72/288]
      PTM__Power_Take_Off_Module_0 [288/288]
```

```
Total Configurations:    48
Checked Configurations:  0

Configuration
Config No.18
Config No.19
Config No.20
Config No.21
Config No.22
Config No.23
Config No.24
Config No.25
Config No.26
Config No.27
Config No.28
Config No.29
Config No.30
Config No.31
Config No.32
Config No.33
Config No.34
Config No.35
Config No.36
Config No.37
Config No.38
Config No.39
Config No.40
Config No.41
Config No.42
Config No.43
Config No.44
Config No.45
Config No.46
Config No.47
Config No.48

DesignSpace
  Vehicle_Drive_Train
    left_side_DriveLine [Compound]
      Drive_Shaft_lhs [Alternative]
        Drive_Shaft_0 [144/288]
        Drive_Shaft_lhs__Drive_Shaft_1 [144/288]
      Final_Drive_lhs [Alternative]
        Final_Drive_0 [96/288]
        Final_Drive_2 [96/288]
        Final_Drive_lhs__Final_Drive_1 [96/288]
    right_side_DriveLine [Compound]
      Drive_Shaft_rhs [Alternative]
        Drive_Shaft_0 [144/288]
        Drive_Shaft_rhs__Drive_Shaft_1 [144/288]
      Final_Drive_rhs [Alternative]
        Final_Drive_0 [96/288]
        Final_Drive_2 [96/288]
        Final_Drive_rhs__Final_Drive_1 [96/288]
    Vehicle_PowerPack [Compound]
      Connector1 [288/288]
      Cross_Drive_Trans [Alternative]
        Cross_Drive_Trans__Cross_Drive_Transmission_1 [144/288]
        Cross_Drive_Transmission_0 [144/288]
      Engine_w_Connector [Compound]
        Connector [288/288]
        Engine [Alternative]
          CI_Engine_0 [72/288]
          CI_Engine_1 [72/288]
          CI_Engine_2 [72/288]
          Engine__CI_Engine_3 [72/288]
      PTM__Power_Take_Off_Module_0 [288/288]
```

Figure 11: Design Space Exploration. Before and After Constraint Application

3.2.3 Design Evaluation (Test Bench)

While application of constraints can eliminate design alternatives based on simple, static properties, much of the system behavior (desirable and undesirable) emerges from the dynamic interaction between components. These interactions occur across and between any and all of the physical domains within the spectrum of the model coverage.

Evaluation of a model configuration can be done vs. requirements imposed on a system design. Requirements are expressed in terms of Metrics that can be computed on the system models. Metrics might include speed, maximum towing force, or acceleration time. Requirements are tests on these metrics. Typically, the conditions and scenario will be specified for a requirement CyPhy support, e.g. Level ground, Pavement, and the scenario of Driver Throttle at 100%. For example, “the vehicle must accelerate from 0 to 60 MPH in less than 8 Seconds”.

System performance evaluation is specified via a Test Bench. A test bench is an executable specification of a requirement analysis. The parts of a Test Bench are:

- Test Drivers, reproducing the stimulus to the system
- Wraparound environment, providing the interfaces at the periphery of the system (e.g. the ground interface with the tires, the external air, ...)
- Metrics evaluation, taking measurements of the system properties and converting into a value of interest. The metrics are also tied to requirements, which can convert the metric to a design “score”. And,

- The system under test – either a point design or a design space. In the case of a design space, the test bench can be applied over the entire set of feasible designs.

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 cover mechanical, electrical, hydraulic, and thermal domains.
- Structural, using 3D CAD assemblies to evaluate the physical compatibility of parts, locate potential interference, and compute physical properties such as 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 to predict vehicle mobility based on aggregate system properties,
- Cyber, co-simulating dynamics with a time-based software/processor/network simulation.
- Manufacturability; creating the 3D CAD files containing the properties of 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.

Test bench also has a set of limits associated with part minimum/maximum parameters, (such as maximum torques on a drive shaft), design limits associated with an assembly or the use of a part in a system (such as minimum allowed battery charge). The limits are automatically evaluated with each evaluation of a test bench. If limits are exceeded, a test bench result can be ignored or otherwise modified or treated with less trust.

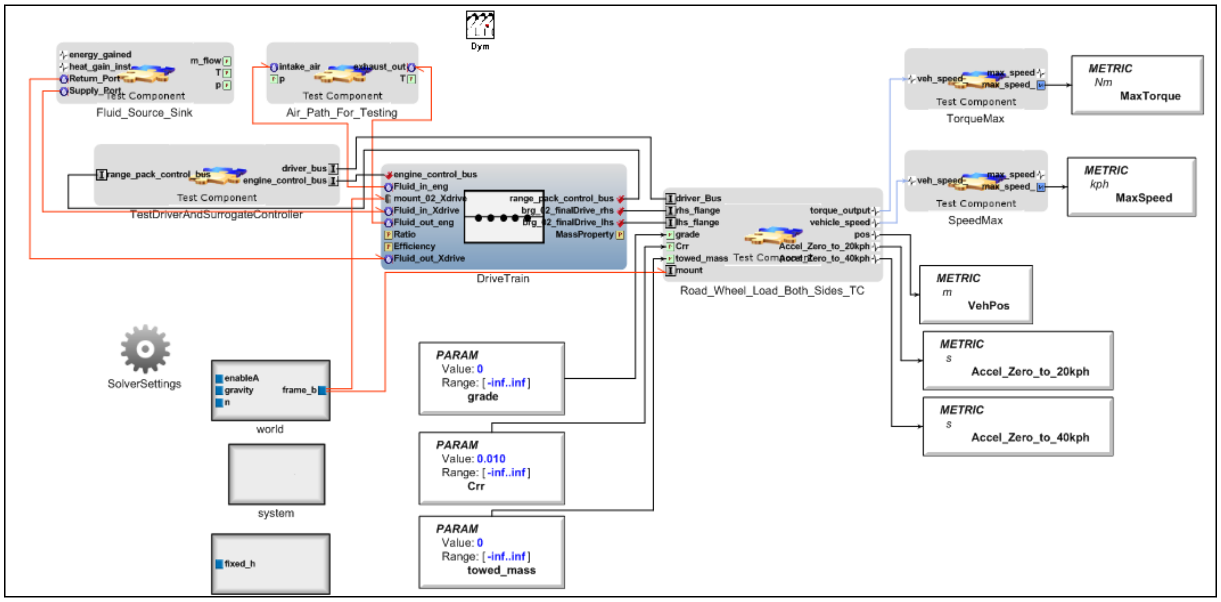


Figure 12: Example Test Bench

4.0 PUBLICATIONS

Neema S., Bapty T., Karsai G., Sztipanovits J., Corman D., Herm T., Stuart D., and Mavris D., “A Multi-Modeling Language Suite for Cyber Physical Systems,” Proceedings of the 4th International Workshop on Multi-Paradigm Modeling.

Lattman Z., et al., “Towards Automated Evaluation of Vehicle Dynamics in System-Level Design,” Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2012, August 2012, Chicago, IL, USA.

Wrenn R., et al., “Towards Automated Exploration and Assembly of Vehicle Design Models,” Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2012, August 2012, Chicago, IL, USA

Simko G., et al., “Foundation for Model Integration: Semantic Backplane,” Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2012, August 2012, Chicago, IL, USA

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

ACRONYM	**DESCRIPTION**
AVM	Adaptive Vehicle Make
AML	Architecture Modeling Language
CPS	Cyber-Physical Systems
CAN	Controller Area Network
CAD	Computer Added Design
ISG	integrated starter generator
DSTC	Domain Specific Tool Chain
DESERT	Design Space Exploration Tool
META	META is not an Acronym. This is the program name as presented by DARPA/TTO AVM
OCL	Object Constraint Language
UML	Unified Model Language