Proceedings of the ASME 2012 International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
IDETC/CIE 2012
August 12-15, 2012, Chicago, IL, USA

DETC2012-71464

TOWARDS AUTOMATED EXPLORATION AND ASSEMBLY OF VEHICLE DESIGN MODELS

Ryan Wrenn Adam Nagel Robert Owens
Di Yao Himanshu Neema Feng Shi
Kevin Smyth Chris vanBuskirk Joseph Porter
Ted Bapty Sandeep Neema Janos Sztipanovits

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee 37212
Email: rwrenn@isis.vanderbilt.edu

Johanna Ceisel Dimitri Mavris

Aerospace Systems Design Laboratory
Georgia Institute of Technology
Atlanta, Georgia 30332-0150
Email: jceisel@asdl.gatech.edu

ABSTRACT

We describe the use of the Cyber-Physical Modeling Language (CyPhyML) to support trade studies and integration activities in system-level vehicle designs. CyPhyML captures integration interfaces across multiple design domains for system components, and generic design assembly rules given in terms of architecture alternatives. The CyPhyML tools support automated exploration of system-level architectural and parametric tradeoffs using a suite of design exploration tools that can be applied to models at different levels of fidelity and scale. Our overall approach includes exploration over the space of potential designs by evaluating structural combinations and then comparison of designs by simulating the dynamics of systems and subsystems with varying degrees of detail. In that flow, we use the DESERT toolkit for design tradeoffs that can be evaluated from the structure of the design model (i.e. interconnections of components and their parameters). DESERT extends a graphical modeling language with concepts and relations to define structural alternatives and constraints on system properties. Alternatives are given in an abstract way, decoupled from the details of the encoding of the combinatorial design space problem for the underlying binary decision diagram (BDD) solver. For desirable design instance models, the tools automatically assemble complete vehicle or subsystem computer-aided design (CAD) models from the associated component CAD models, and likewise create detailed finite-element structural analysis (FEA) models for selected component assemblies. Evaluation results are presented in a dashboard display which provides comparisons between different valid designs over all specified design metrics.

INTRODUCTION

The DARPA AVM project aims to reduce the typical specification, design, analysis, construction, and manufacturing time and cost for a new military vehicle design by a factor of five. Two significant challenges that impact development cost and schedule are 1) system-level integration and 2) evaluation of design alternatives. Integration, or the resolution of details between design domains and subsystems in an engineered system is frequently cited as a principal source of schedule and cost overruns [1]. Tackling integration issues early in the design cycle goes a long way towards producing the expected gain. Second, the number of potential vehicle designs which could be built from a set of existing components is very large, even when considering only macro-component choices such as engines, transmissions, hull, and chassis options. Evaluation of possible candidate designs against the system requirements enables the decision-making necessary to converge to a final design candidate. Automated design space exploration (DSE) has been identified as a key problem in Model-Based System Engineering (MBSE) [2]. As for integration, the early pruning of candidate designs dramatically reduces the effort required to complete the end-to-end design and development processes. However, early design candidates usually lack the details necessary to effectively prune the design space, suggesting an iterative approach.

In this work, we describe the Cyber-Physical Modeling Language (CyPhyML) and tools that use model-integrated computing techniques to support the design-time integration of numerous aspects of a system-level vehicle design, and the automated exploration of design alternatives. CyPhyML integrates models from existing engineering design and analysis tools to seamlessly support design work across multiple domains.

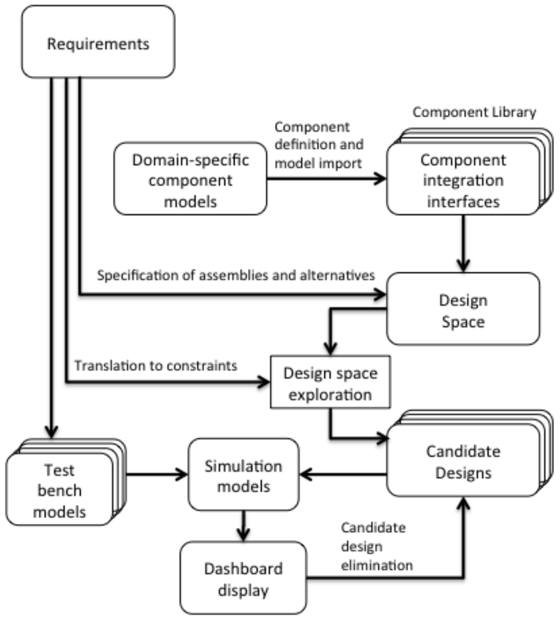
Requirements
Domain-specific component models
Component definition and model import
Component Library
Component integration interfaces
Specification of assemblies and alternatives
Design Space
Translation to constraints
Design space exploration
Candidate Designs
Test bench models
Simulation models
Candidate design elimination
Dashboard display

FIGURE 1. DESIGN SPACE REFINEMENT

CyPhyML supports integration and evaluation efforts using a few key concepts:

1. Model-Integrated Computing (MIC) is the core technology on which CyPhyML and its tools are built [3].
   The use of MIC in system-level design moves many of the detailed design decisions normally resolved during lengthy integration (and often redesign) phases forward to the design and analysis stage of a system design, or often even prior to design activities. These details are resolved by an up-front concerted modeling effort which captures concepts and relations from design models in different design and analysis domains, representing each domain in a formal language description known as a metamodel. These metamodel descriptions are then integrated into a single language (in this case, CyPhyML) that precisely represents the relationships between the metamodels for the different domains. Model interpreters automatically import language and analysis artifacts from existing design tools into models in the integrated language, and can also be created to generate required design models. Both CyPhyML and its integrated domain-specific metamodels are defined using the Generic Modeling Environment (GME). GME also provides tools and APIs to develop the model interpreters required to realize the benefits of the MIC approach.
2. Components in CyPhyML correspond (in most cases) to real physical components. CyPhyML aggregates the different domain models associated with each component using carefully designed interface abstractions for each domain, so that larger design models can be assembled automatically using the multi-domain component definitions and descriptions of acceptable interconnections. We say that CyPhyML is *compositional* with respect to well-formed domain models. CAD models comprise one design domain that is integrated by CyPhyML. Designers create component specifications which link to CAD models stored in a repository, and annotate the component description in CyPhyML with interface ports which represent the possible interconnection points for that component. Similar modeling structures are used to define compatibility and assemble designs.
3. CyPhyML supports automated exploration of possible design alternatives, and the automated realization of selected configurations as complete design models. CyPhyML encodes valid design alternatives by specifying component assemblies in terms of multiple candidate components and rules for their assembly. Integrated tools allow the automated exploration of valid candidates based on structural information available in the component integration model. The design process supports additional pruning of the design space by generating more detailed simulation and analysis models which can be evaluated against test cases to determine suitability and performance of candidate designs.
4. CyPhyML allows users to specify test bench models to simulate model behavior or perform detailed model analysis. A test bench provides a context in which the details of a vehicle or particular subsystem can be evaluated. Designers can assess correctness and compare different design configurations with respect to design metrics, which are the key performance parameters for the design.

Figure 1 illustrates the general process of defining a design space to identify and compare design candidates which satisfy the requirements, and then the refinement steps to narrow the set of candidate designs down to a manageable set of alternatives. First, domain-specific component models (such as CAD parts/assemblies, dynamics simulations, etc...) are captured in a data repository, and their interfaces are represented in the corresponding CyPhyML component library. System requirements drive the specification of component assemblies and design alternatives. Requirements also serve as constraints during automated DSE processes. The constraints eliminate some designs from the set of feasible design candidates, leaving a smaller set of designs for more detailed evaluation. Repeated simulation and analysis steps driven by test specifications further reduce the set of possible design candidates, while providing metric values to compare between remaining candidates. The summary metrics are displayed in a dashboard to visualize the trade-offs.

Many design requirements can be expressed as constraints on the structure of design models. The importane of structural constraints in the design flow is that constraint-based design space exploration tools can efficiently restrict the design space using computationally inexpensive techniques, relying solely on details readily available in component and design specifications. As a simple example, if each component and assembly in the component library is annotated with metadata which allows the tools to estimate its cost, then the total estimated cost can be computed for any candidate design. Comparing estimated design costs against budget thresholds and between candidate designs allows the designers to eliminate candidates which are clearly out of range for a particular project. This determination is based solely on information computed from properties stored directly in the model, without executing simulations of model behavior.

As opposed to structural constraints, behavioral constraints such as performance requirements call for behavioral analysis of designs using simulation or other analysis tools. Since simulation for large models is expensive, it is a good strategy to restrict the design space first by means of structural constraints before exploring the remaining (smaller) candidate designs using simulation (e.g., as shown in Fig. 1).

We will cover a specific subset of the CyPhyML design flow which relates to the specification and evaluation of physical models in the form of CAD and associated structural analysis tools.

BACKGROUND

Model-Integrated Computing

In model-based design, systems are described by models expressed in domain specific modeling languages (DSML). Formally, a DSML is a five-tuple of concrete syntax (C), abstract syntax (A), semantic domain (S) and semantic and syntactic mappings (M_S, and M_C):

$$L = < C, A, S, M_S, M_C > \quad (1)$$

The *concrete syntax C* defines the specific (textual or graphical) notation used to express models, which may be graphical, textual or mixed. The *abstract syntax A* defines the concepts, relationships, and integrity constraints available in the language. Thus, the abstract syntax determines all the (syntactically) correct "sentences" (in our case: models) that can be built. (It is important to note that the abstract syntax includes semantic elements as well. The integrity constraints, which define well-formedness rules for the models, are frequently called "structural semantics.") The *semantic domain S* is usually defined by means of some mathematical formalism in terms of which the meaning of the models is explained. The mapping $M_C : A \rightarrow C$ assigns syntactic constructs (graphical, textual or both) to the elements of the abstract syntax. The semantic mapping $M_S : A \rightarrow S$ relates syntactic concepts to those of the semantic domain.

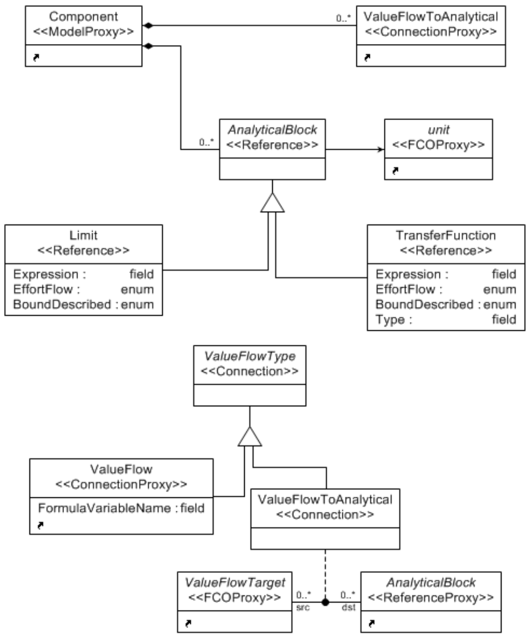

FIGURE 2. METAGME EXAMPLE

Any DSML requires the precise specification (or modeling) of all five components of the language definition. The languages which are used for defining components of DSMLs are called meta-languages and the concrete, formal specifications of DSMLs are called metamodels [3].

The models and languages we will describe were created using the ISIS Generic Modeling Environment tool (GME). GME allows language designers to create stereotyped Unified Modeling Language (UML) class diagrams defining metamodels. The metamodels are instantiated into a graphical language, and metamodel class stereotypes and attributes determine how the elements are presented and used by modelers. The GME metamodeling syntax may not be entirely familiar to the reader, but the syntax is well-documented [4]. Class concepts such as inheritance can be read analogously to UML, as shown in Fig. 2. Class aggregation represents containment in the model editing environment, though an aggregate element can be flagged as a port object. In the model editing environment a port object will be visible at the next higher level in the model hierarchy, and available for connections. The dot between the *ValueFlowTarget* class and the *AnalyticalBlock* class, together with the *ValueFlowToAnalytical* association class represent a line-style connector in the modeling environment. When a connection is drawn in the model, the corresponding object is an instance of the association class connected via the dashed line to the dot (in this case, the class ValueFlowToAnalytical).

GME and related tools facilitate the model-integrated computing [3] approach to system-level design. We build up custom domain-specific modeling languages (DSMLs) in GME by creating metamodels for individual domains, whether describing a component interface language or the language required by an existing analysis tool. These metamodels capture the model structure and relationships to formally represent concepts and relations in each particular domain. Model integration occurs when we use those specific metamodels as sublanguages to build up a larger language to support system integration activities, engineering design flows, and multi-domain analysis. GME allows a DSML designer to use existing language constructs as building blocks in the new language, and to relate those language constructs to others to define their proper relationship and use in an integrated model. As a simple example, consider the metamodel fragment shown in Fig. 2 which describes concepts for parameter calculation and propagation during formula evaluation. The metamodel relates the class Component (which is used in many domains) to child elements Limit and TransferFunction from the Bond Graph domain for defining dynamics. Each of these class types is also a reference (denoted by the arrow) to an object of type *unit*, indicating that each can have a physical units specification as well.

Once the integration language has been defined to associate models in the various design domains, GME provides application programming interfaces (APIs) which allow developers to write software that manipulates, analyzes, or transforms models conforming to the language. We refer to these software components as *model interpreters*.

AUTOMATED DESIGN SPACE EXPLORATION

Reuse of existing model components in the design process significantly decreases the cost of constructing models in MBSE. Model structures for vehicle design are specified as refinements of a generic architecture. Each top-level model component, such as engine, transmission, chassis, etc..., has many alternative realizations with different parameters and internal structure - arranged in a refinement hierarchy. Accordingly, components at any level in the refinement hierarchy may have alternative implementations. We call components with alternative implementations *templates*. At the leaf nodes of the refinement hierarchy are the *primitive* model components, which may also be templates with alternative (primitive) implementations. In our specific application context, the primitive model components are either individual physical components or integrated assemblies of physical components. The template concept has a significant impact on the structural semantics of models. Without templates, each model m is unique and represents a single point design. By introducing templates in the specification of the modeling language, each model including templates defines a set of models M_D which we call a design space. A model instance in the design space ($m \in M_D$) is defined by binding the component templates to one of the alternative implementations and by the binding the parameter values of the parameterized components to a specific value.

The DEsign Space ExploRation Tool (DESERT) encodes the structures and constraints into Binary Decision Diagram (BDD) models of the component design space. The BDD model represents the set of candidate designs. Exploration is performed as the BDD solver applies constraints to prune the space [5].

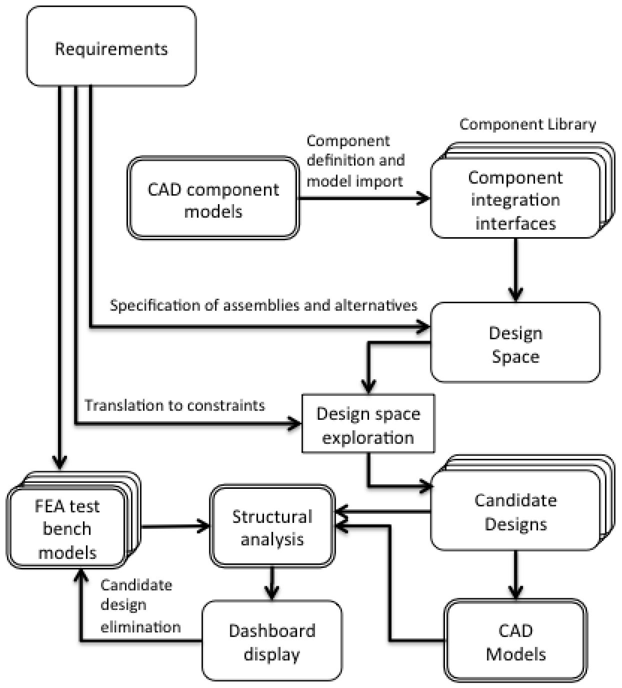

FIGURE 3. DESIGN FLOW SUPPORTING DESIGN SPACE EXPLORATION AND CAD SYNTHESIS

DESIGN USING CYPHYML

Design Flow Supporting CAD Assembly

For the subset of the CyPhyML language and tools covered in this article, we present design qualities and features that can be determined from details specified in CAD models. The interface models for elements of the component library include the details required to compose CAD models and to translate CAD assemblies into structural finite element analysis (FEA) models. FEA of vehicle subsystems is driven by test bench models. Figure 3 highlights details of the CAD integration features of CyPhyML into the overall design evaluation flow shown in Fig. 1. In the sequel we use two examples from different parts of the vehicle design in order to illustrate different concepts. A single example would have been preferable, but the availability of detailed models for different parts of the design was limited.

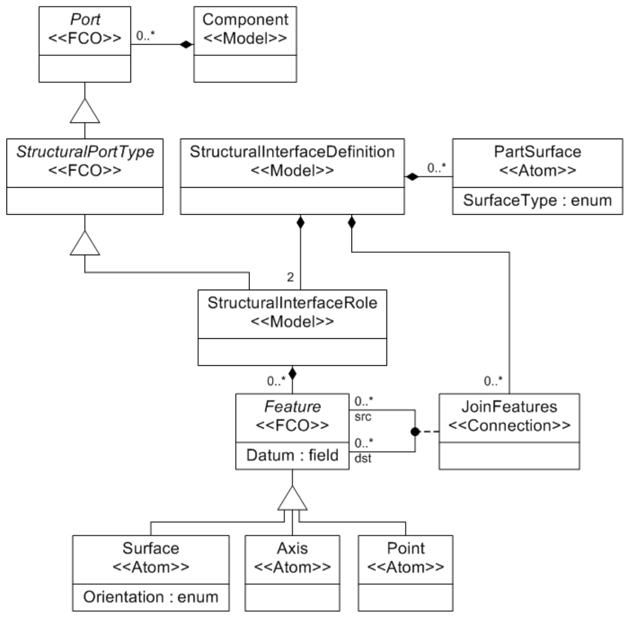

FIGURE 4. METAMODEL FOR COMPONENT CAD INTEGRATION INTERFACE

Integration Using Component Models

A CAD modeler specifies feature points which define the integration details between components in the CAD model. These features are stored as metadata in the CAD model and are referenced in CyPhyML as the component integration interface. As shown in Fig. 4, structural interfaces are included with component definitions, and in the structural aspect of the model the feature points are visualized as ports on a component block. Compound assemblies are built up by selecting parts and connecting points to describe how the parts fit together physically. For FEA, additional structural features may be added to define surfaces or regions to be evaluated. These are called out in a test bench model as depicted in Fig. 10.

A CAD modeler would uniquely name geometry features (i.e. points/axes/surfaces) within the CAD model, and those named features would be used to specify how parts could be positioned relative to one other in the CAD assembly. For example, three orthogonal planes in one part could be mated to three orthogonal planes in another part. A combination of the named geometry features sufficient to position a part is termed a structural interface role as shown in Fig. 4. A structural interface role in CyPhyML would be composed of the names of the CAD geometry features along with orientation information where appropriate. The above discussion applies equally well to multilevel assemblies, where sub-assemblies would be oriented relative to each other. A CAD assembly would be defined within CyPhyML by specifying components along with lines connecting ports on the components, where the ports would be structural interface roles. For FEA, additional structural features may be added to define surface regions to be evaluated. These are called out in the test bench as shown Fig. 10.

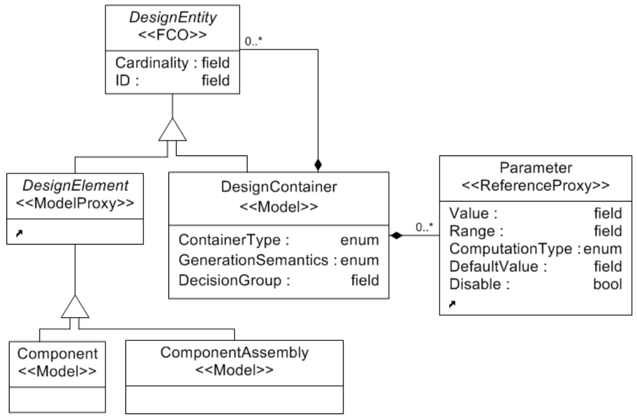

FIGURE 5. METAMODEL DEFINING DESIGN SPACE

Definition of Component Assemblies and Design Space

Fig. 5 depicts the CyPhyML mechanism for defining design alternatives. Anywhere in a CyPhyML model that a component can appear, we can also substitute a design container. A design container contains other components and specifies the relationship between the included components and the design space. The *ContainerType* attribute indicates its interpretation in the design space – A *Compound* container simply aggregates components and design containers, an *Optional* container indicates that the contents may be excluded, and an *Alternative* container indicates that any one (and only one) of the contained components may be used.

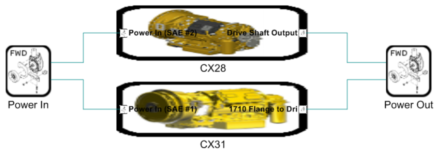

FIGURE 6. TRANSMISSION ALTERNATIVES - STRUCTURAL ASSEMBLY ASPECT

The alternative containers are key to the design space exploration process. For initial design space pruning, we use lower-fidelity parameterized component models which can share common physical and parametric interfaces. The common interfaces allow the generation tools to operate orthogonally to the design space exploration process, so that any feasible model from the design space can be generated to either CAD or to simulation accordingly.

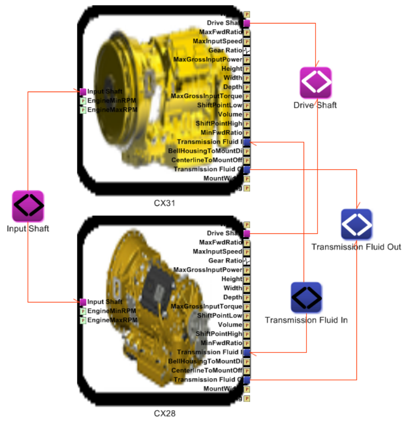

FIGURE 7. TRANSMISSION ALTERNATIVES - DYNAMICS ASPECT

AUTOMATED DESIGN SPACE EXPLORATION

Modeling Alternatives in CyPhyML

DESERT uses an abstract domain-independent meta-model, which separates its internal algorithms from domain-specific constructs. The Design-Space Abstraction tool provides two-way model translation between the Design-Space Models in CyPhyML and the abstract design-space models in DESERT.

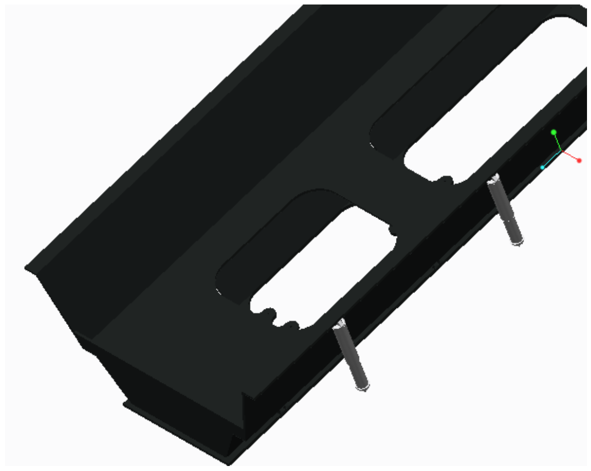

FIGURE 8. AUTOMATED CAD ASSEMBLY

Figures 6 and 7 show two aspects from the alternative design container for transmission components. The container holds two transmissions, *CX28* and *CX31*. The CyPhyML *Structural* aspect shown in Fig. 6) defines the common interface required to compose CAD models for either component. The *Power In* and *Power Out* objects shown in the figure are presented as connectable ports at the level of hierarchy above the model shown. These ports encode structural compatibility between either transmission component and other drive train components which could be physically connected to a transmission. In this aspect of CyPhyML, a connection drawn between the *Power In* port of the transmission and a *Power Out* port of an engine component would represent the CAD-model details of the physical coupling between two components. Such a connection is only allowed if the structural information in the two ports is compatible.

Likewise, the CyPhyML *Dynamics* aspect (Fig. 7) specifies a behavioral interface for either transmission alternative. Each has two rotational power connections (purple port objects outside the component) and two fluid power connections (blue port objects). Each of these power objects will appear as ports at the next higher level of the model hierarchy. The ports encode behavioral compatibility between physical components. Continuing the example, if the *Input Shaft* port of a transmission is connected to the rotational power output port of the engine, then the connected structure would represent an acausal, simultaneous power-sharing connection between the two components. This means that for the connected shaft, torque and angular velocity variables are considered common between the two components. Their respective equation sets are merged to represent a single mutual set of behaviors.

Because alternative components in the design space have common multi-aspect interfaces, model operations such as parameter calculation (e.g., for weight) and simulation generation are orthogonal to the specification of the design space. In CyPhyML we can define an assembly generically, representing all of the possible component combinations among the parts needed to define the assembly. Note that this interface commonality is only strictly possible for low-fidelity component models, but that is exactly the objective of the first stages of design space reduction. Low-cost, low-fidelity models are used to narrow the set of candidates down to a small set for the high-cost, high-fidelity simulation and analysis required for detailed design.

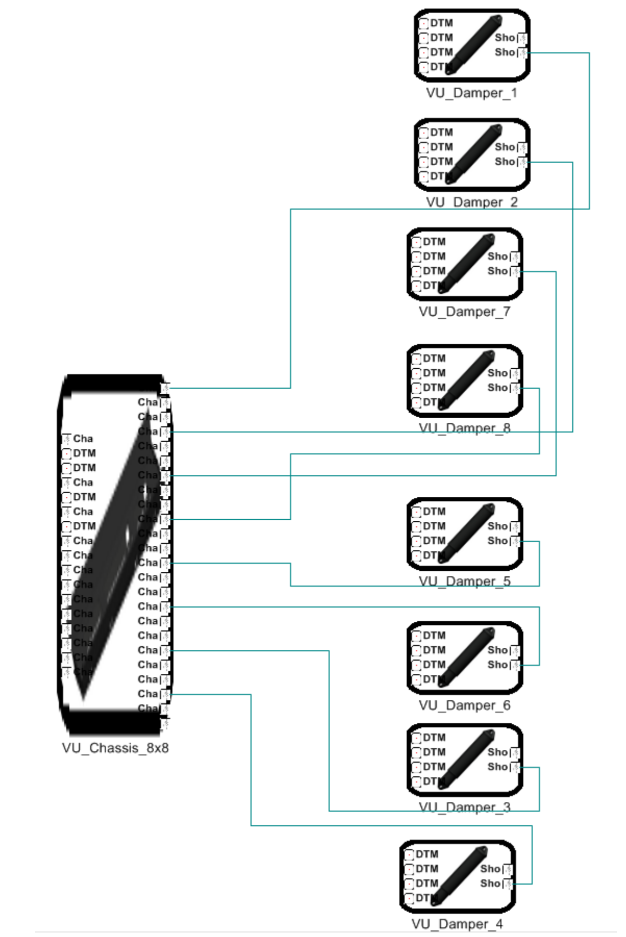

FIGURE 9. DESIGN SPACE FOR CHASSIS AND DAMPERS

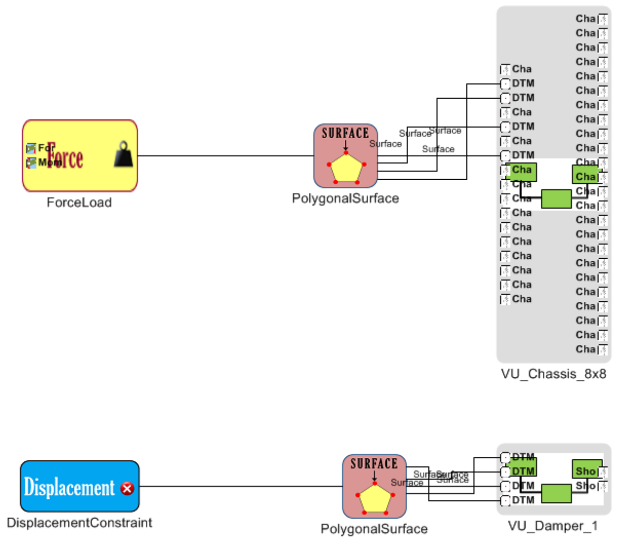

FIGURE 10. CAD TEST BENCH EXAMPLE

MODEL SYNTHESIS AND ANALYSIS

CAD Model Assembly

Once DSE is complete, the CyPhyML tools elaborate the abstract high-level system design model instances with CAD model details and construct the CAD assembly model. The key to achieving development schedule gains is in the ability of the CyPhyML tools to automate the processes of creating assemblies and synthesizing detailed analysis models. For example, building a CyPhyML model for a single assembly would be more laborious than building the single assembly via the native CAD system; however, once the initial setup was completed, hundreds of similar assemblies could be built programmatically with little human intervention. Likewise, once the CAD assembly information has been modeled in CyPhyML, building a FEA test bench would require a similar level of effort as using a native CAD package to compose a FEA analysis. The major advantage is that once the CyPhyML test bench model was built, hundreds of analyses could be run with minimal human intervention.

Figure 9 contains the design space specification for the chassis and damper model assembly. In this example the design is a singleton as each component has only a single alternative in the component library. The connections are structural (as in Fig. 6), so the component models and their connections represent the physical assembly of any compatible parts in the design space. A ChyPhyML model interpreter automates this assembly process. Fig. 8 shows an assembled chassis model from Creo together with damper components attached at the proper points.

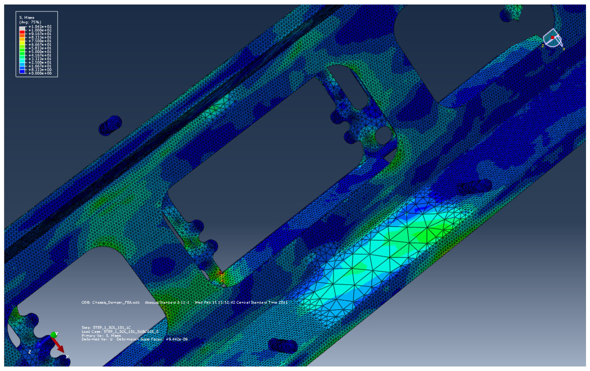
FIGURE 11. STRUCTURAL ANALYSIS RESULTS

Structural Analysis

Figure 10 shows a test bench model for defining structural analyses. The test bench model calls out specific surfaces (selected by the modeler), selects a shape for those surfaces (in this case the shapes are polygonal), and specifies either forces to be applied to or physical constraints for those surfaces. The tested components (VU_Chassis_8x8 and VU_Damper_1 – only one damper is shown to save space) each represent the entire design space for those types of components, as any of the alternative components may be substituted into the design. After the automated DSE step, a new test bench model is created from this test bench model. The components in the new test bench are assembled according to the design space assembly rules for CAD models, but in the new test bench the generic components have been substituted for specific component instances. This process of automatically specializing generically defined test bench models to specific design candidates is key to the refinement process supported by the CyPhyML language and tools (see Figs. 1 and 3).

The basic process for producing a FEA model from the component CAD models and the CyPhyML integration and design space data is as follows:

1. From CyPhyML, export to an XML file the hierarchy and structural interfaces of the CAD assembly along with the FEA constraints and loads information.
2. Programatically invoke the CAD application and build the assembly, create a FEA mesh of the assembly, and add constraints and loads to the mesh resulting in a completely defined FEA input deck.
3. Submit the FEA input deck to a solver.
4. Post process the results for consumption by the dashboard.

Evaluating Design Alternatives

Table 1 displays values from the dashboard that can be determined by component properties, the assembly rules, and the selection of a particular design candidate. For space we only include two design candidates. Note that these values are only representative.

Providing access to relevant predictive data is not sufficient to enable users to make informed decisions. Decisions are hard to make on poorly-visualized raw data. A multi-disciplinary trade-off tool that dynamically varies design requirements and quickly filters cost and performance metrics puts this predictive power into the hands of not only design engineers, but of Integrated Product Teams and stakeholders across multiple disciplines.

In order to use the data to its full potential, AVM developed a trade-off tool that possesses the capability to display and compare several different concepts side by side through relevant visualizations. Graphs, diagrams and constraint plots help the engineer evaluate their decisions comparatively, enabling them to make more informed decisions. Visualizations tend to provide a higher bandwidth flow of information: communicating more in a short period of time that traditional raw data ever could. Visual communication also grants the ability to communicate advanced engineering concepts and their effects to a non-technical audience. An image of the overall dashboard is shown in Fig. 12.

TABLE 1. DASHBOARD RESULTS FOR STRUCTURAL PROPERTIES

Design ID	C012	C019
Vehicle Height (m)	3.41807	3.84590
Vehicle Length (m)	5.32698	6.70708
Vehicle Width (m)	3.72523	3.69449
Vehicle Weight (kg)	14705.9	14602.3
Tire Radius (m)	0.60809	0.61524
Final Gear Ratio	3.74	4.34
Diesel Engine	C7_Diesel	C13_Diesel
Fuel Weight (kg)	266.059	227.953
No. of crew	3	3
No. of passengers	9	9
Hull	Steel/Welded	Composite/Bonded
Transmission	CX28	CX31
Transfer Case	484	484

RELATED WORK

State of Current MBSE Tools

Reichwein and Paredis offer a canonical overview of concepts, history, and directions for the field of Model-Based Sys-

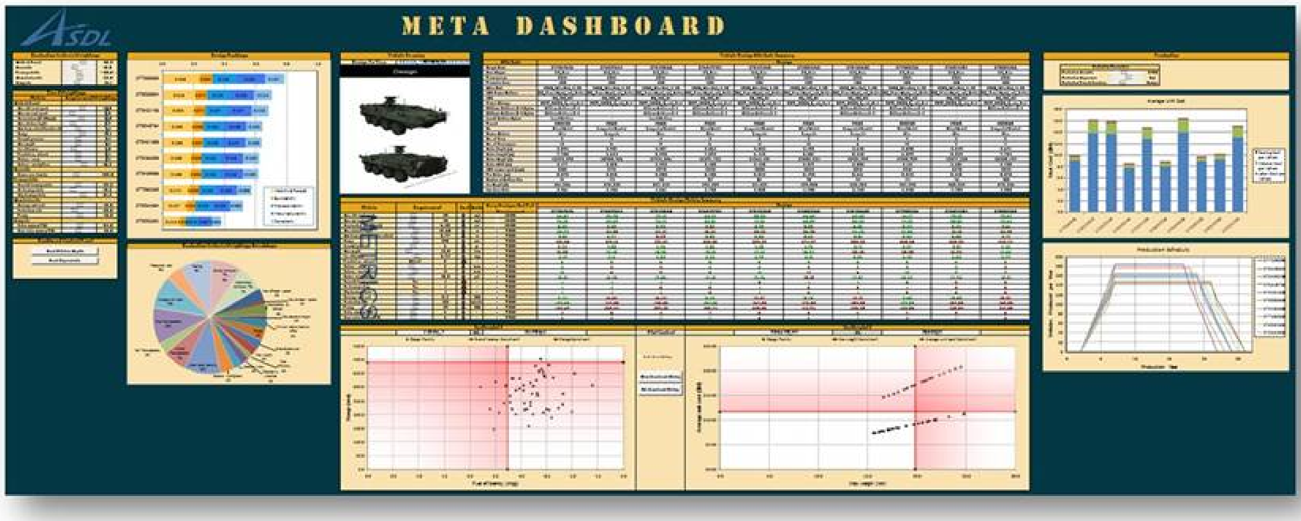

FIGURE 12. DESIGN SPACE VISUALIZATION

tems Engineering (MBSE) [2] - their work surveys some of the basic terminology and fundamental issues. Our work has much in common with current research efforts in SysML and related tools.

In contrast to current MBSE approaches, our integration methods aim towards a more strict 'glue' role, which does not prescribe particular technologies/standards, but picks and chooses [and integrates] from best of breed alternatives for attacking particular aspects of the problem. We use abstractions for each of the integrated capabilities (component definition, DSE, and generation) so that users can work in different modeling tools and integrate their existing component libraries.

The Core Product Model (CPM) from NIST is a standard which covers many of the same design issues covered by CyPhyML and its associated tools [6] [7]. CPM defines a meta-model to capture components and their features, as well as the design of assemblies using those components. CPM is multi-domain, and seeks to separate the specification of function (i.e., intended behavior), form (i.e., physical realization), and behavior (i.e., implemented behavior) within design models. CPM and CyPhyML are both component-based and support different levels of model refinement. The Open Assembly Model (OAM) for CPM defines component interfaces for creating physical assemblies (i.e. for CAD and FEA models) from models for components [8], in the same way that CyPhyML allows designers to specify physical assembly interfaces. One key difference between CPM and CyPhyML is that CPM seems to be designed to support the evolution of a component definition within the context of an evolving design, where CyPhyML relies on a semi-static component library where component variants are versioned and typed, and stored as separate components in the library.

CAD Integration

The Initial Graphics Exchange Specification (IGES) and the Standard for the Exchange of Product Model Data (STEP) are generic formats for CAD models or assemblies, and are used to translate CAD models between systems. An example would be translating a Creo model to IGES and reading that IGES file with SolidWorks. For part models, the IGES file contains graphical entities (e.g. surfaces, lines, points...) and for assemblies, IGES contains a list of part/sub-assembly names along with positioning information (i.e. transformation matrix). CyPhyML is similar to an IGES assembly in that it contains the list of part/sub-assembly names along with positioning information, where the positioning information is based on aligning features of parts. Additionally, the CyPhyML contains parametric information (e.g. thickness of hull, length of passenger compartment...) about parts, which would not be part of an IGES file. In general, the strength of our approach is not about the representation format (CyPhyML/IGES), but that we can programmatically build and analyze CAD assemblies based on synthesized designs.

Gross et al describe the use of UML to integrate CATIA CAD models with Matlab/Simulink simulations for satellites [9]. The presented CAD integration approach captures many details for each part and integrates parts using constraint classes. Matlab serves as an algebraic engine to calculate part properties between reference frames. The Matlab interface also provides configuration parameters to an existing Simulink simulation, and controls the invocation of the simulation. Model and analysis workflows are captured directly in UML activity diagrams, allowing the evaluation of multiple geometric configurations of parts in the system design for a particular workflow.

Design Space Exploration

Shah et al discuss the integration of nonlinear solvers with a SysML-based design process to explore the sizing of hydraulic components in a mechanical design [10]. Selection among alternatives seems to be based on size. Their approach creates a metamodel describing the constraints and objectives in the generic GAMS optimization language, and calls out the solvers to be used for particular models. The paper gives a design example of sizing components for a hydraulic log splitter. The approach presented here is generic in the sense that it can use different solvers, but constraints and objectives are limited to those that can be modeled in GAMS and addressed by the available solvers (which is true for any integrated solver-based approach). The sizing problem is constrained by different quantities, but the constraint model as displayed in the SysML essentially contains the elements of a script.

ACKNOWLEDGMENT

This work was supported by DARPA under contracts FA8650-10-C-7082 and FA8650-10-C-7075.

REFERENCES

[1] Sangiovanni-Vincentelli, A. L., 2007. "Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design". *Proc. of the IEEE,* ***95***(3), March, pp. 467–506.

[2] Reichwein, A., and Paredis, C. J., 2011. "Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering". In Proc. ASME 2011 International Design Engineering Technical Conf. & Computers and Information in Engineering Conf.

[3] Karsai, G., Sztipanovits, J., Ledeczi, A., and Bapty, T., 2003. "Model-integrated development of embedded software". *Proc. of the IEEE,* ***91***(1), Jan, pp. 145–164.

[4] Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., IV, C. T., Nordstrom, G., Sprinkle, J., and Volgyesi, P., 2001. "The generic modeling environment". *Workshop on Intelligent Signal Processing*, May.

[5] Neema, S., Sztipanovits, J., Karsai, G., and Butts, K., 2003. "Constraint-based design-space exploration and model synthesis". In *Embedded Software*, R. Alur and I. Lee, eds., Vol. 2855 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 290–305.

[6] Fenves, S., 2001. A Core Product Model for Representing Design Information. Tech. Rep. NISTIR 6736, NIST.

[7] Fenves, S., Foufou, S., Bock, C., and Sriram, R., 2008. "CPM2: A Core Model for Product Data". *Journal of Computing and Information Science in Engineering,* ***8***(1).

[8] Rachuri, S., Han, Y.-H., Feng, S., Roy, U., Wang, F., Sriram, R., and Lyons, K., 2003. Object-oriented representation of electro-mechanical assemblies using UML. Tech. Rep. NISTIR 7057, NIST, Oct.

[9] Gross, J., Reichwein, A., Rudolph, S., Bock, D., and Laufer, R., 2009. "An Executable Unified Product Model Based on UML to Support Satellite Design". In AIAA SPACE 2009 Conference & Exposition.

[10] Shah, A. A., Paredis, C. J., Burkhart, R. M., and Schaefer, D., 2010. "Combining Mathematical Programming and SysML for Component Sizing of Hydraulic Systems". In Proc. ASME International Design Engineering Technical Conf. & Computers and Information in Engineering Conf. (IDETC/CIE 2010).