INVITED PAPER

Model and Tool Integration Platforms for Cyber–Physical System Design

In this paper, the authors address the heterogeneity in CPS model libraries and design tools with two integration platforms. The model integration platform enables precise representation of semantic interfaces among modeling domains, while the tool integration platform features automated design space exploration and formal verification.

By Janos Sztipanovits, *Fellow IEEE*, Ted Bapty, Xenofon Koutsoukos, *Fellow IEEE*, Zsolt Lattmann, Sandeep Neema, and Ethan Jackson, *Member IEEE*

ABSTRACT | Design methods and tools evolved to support the principle of "separation of concerns" in order to manage engineering complexity. Accordingly, most engineering tool suites are vertically integrated but have limited support for integration across disciplinary boundaries. Cyber–physical systems (CPSs) challenge these established boundaries between disciplines, and thus, the status quo on the tools market. The question is how to create the foundations and technologies for semantically precise model and tool integration that enable reuse of existing commercial and open source tools in domain-specific design flows. In this paper, we describe the lessons learned in the design and implementation of an experimental design automation tool suite, OpenMETA, for complex CPS in the vehicle domain. The conceptual foundation for the integration approach is platform-based design: OpenMETA is architected by introducing two key platforms: the model integration platform and the tool integration platform. The model integration platform includes methods and tools for the precise representation of semantic interfaces among modeling domains. The key new components of the model integration platform are model integration languages and the mathematical framework and tool for the compositional specification of their semantics. The tool integration platform is designed for executing highly automated design-space exploration. Key components of the platform are tools for constructing design spaces and model composers for analytics workflows. The paper concludes with describing experience and lessons learned by using OpenMETA in drivetrain design and by adapting OpenMETA to substantially different CPS application domains.

KEYWORDS | Cyber–physical systems (CPSs); design automation; model integration; tool integration

I. INTRODUCTION

Cyber–physical systems (CPSs) are engineered systems where functionality emerges from the networked interaction of computational and physical processes.

The tight interaction of physical and computational components creates new generations of smart systems whose impacts are revolutionary; this is evident today in emerging autonomous vehicles and military platforms, intelligent buildings, smart energy systems, intelligent transportation systems, robots, or smart medical devices. Emerging industrial platforms such as the Internet of Things (IoT), industrial Internet (II), fog computing, and Industrie 4.0 are triggering a "gold rush" toward new markets and accelerating the development of societal-scale systems such as connected vehicles, which, in addition to the synergy of computational and physical components, involve humans (H-CPS).

Manuscript received October 8, 2017; revised February 27, 2018; accepted April 29, 2018. Date of publication June 26, 2018; date of current version September 14, 2018. This work was supported in part by the Defense Advanced Research Project Agency under Awards #FA-8650-10-C-7075, #FA8650-10-C-7082, and #HR0011-12-C-0008; and by the National Science Foundation under Award # CNS-1035655. *(Corresponding author: Janos Sztipanovits.)*

J. Sztipanovits, T. Bapty, X. Koutsoukos, and **S. Neema** are with the Institute for Software Integrated Systems, Vanderbilt University, TN 37212 USA (e-mail: janos.sztipanovits@vanderbilt.edu).

Z. Lattmann is with A_3 Airbus, CA 95113 USA (e-mail: zsolt.lattmann@airbus-sv.com).

E. Jackson is with Microsoft Research, WA 80305 USA (e-mail: ejackson@msr.com).

Digital Object Identifier: 10.1109/JPROC.2018.2838530

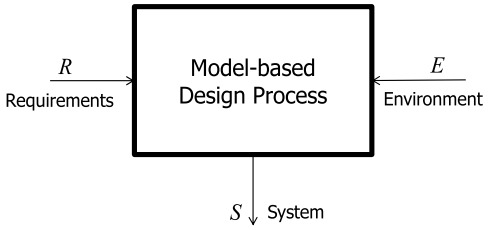

Fig. 1. ***Model-based design.***

As for most engineered systems, CPS design flows are dominantly model based. The model-based design process results in a formal description (model) of the system S, using as input the set of requirements R that must be satisfied in some environment E (Fig. 1).

If S, R, and E can be represented formally, the system design can be framed as a synthesis process [1] that yields S such that when composed with the environment E, it satisfies the requirements R, or $S \parallel E \vDash R$. The automation of the design process assumes the availability of precise, formal models for R and E restricted to a semantic domain where the synthesis process is computationally feasible. However, in CPSs, there are significant challenges with deep impact on model-based design flows.

- Heterogeneity: It is an essential property of CPSs. Many of the advantages of CPS design are expected to come from the synthesis of traditionally isolated modeling domains (creating novel modeling platforms in design flows [2], [3]) and from the explicit representation of interdependences across modeling domains, which is frequently neglected if the separation-of-concerns principle [4] is used incorrectly.
- Modeling uncertainties: In model-based design, R, E, and S are models of some referent systems (conceptual, logical, or physical) or their properties. The model is always a simplification, preserving specifically chosen features of the referent. The relationship between models and their referents plays key role in the effectiveness of the design process. In cyber domains, this relationship is usually an abstraction/refinement relation where property preservation between an abstract model and its referent (e.g., a software implementation) is an achievable goal. In physical domains, this relationship is fundamentally different. There is always some level of uncertainty between a model and its physical referent that may be aleatoric (irreducible) originating from the underlying physics, or epistemic arising from the lack of knowledge [69]. Measuring, expressing, and propagating uncertainties in CPS design models (DMs) is an important challenge.
- Complexity and scalability: Complexity and scalability of verifying properties of models are significant concerns for all model-based design approaches that target real-life systems. Selecting modeling abstractions with scalable symbolic or simulation-based verification methods is a key approach on the cyber side of CPSs and may result in significant complexity reduction [1]. However, on the physical side, adjusting the level of abstractions used in modeling without considering the underlying physics inevitably leads to increase in epistemic uncertainty that may make the verification unsound and/or incomplete.

This paper is based on our experience with constructing a fully integrated model-based design tool chain prototype for the "make" process of complex CPSs, as part of the Defense Advanced Research Project Agency (DARPA) Adaptive Vehicle Make (AVM) program [5]. The resulting design automation tool chain, OpenMETA [6], was expected to contribute to the following goals of the program.

- Improved design productivity: Shortening development times for complex CPSs was a primary goal of the program. OpenMETA contributed to achieving this goal with the following technical approaches: 1) extending model-based design to component- and model-based design; 2) pursuing correct-by-construction design methods; and 3) executing rapid requirements tradeoffs across design domains.
- Incorporation of manufacturing-related constraints into the design flow to help moving toward foundry-like manufacturing capability for CPSs, decreasing the need for lengthy and expensive design–manufacture–integrate–test–redesign iterations.
- Support of crowdsourced design by providing a web-based collaboration platform for *ad hoc*, geographically distributed design teams and access to dominantly open source, cloud deployed tool configurations for very low cost.

The project focused on integrating and testing an automated, model-based design flow for the power train and hull of the fast adaptable next-generation ground vehicle (FANG) [7], [8] using primarily open source tool components. Given the program focus, our team explored the fundamental challenges of CPS design automation emerging from a practical constraint: domain-specific CPS design flows are constructed to support synergistically a design methodology but need to be implemented by integrating a wide range of heterogeneous tool components. This leads to, what Sangiovanni-Vincentelli and Broy call, the "tyranny of tools" in design automation [9], when not the design problems, but available tools dictate the abstractions that designers should use in problem solving.

The significance of the problem has long been recognized by industry. In fact, the challenge of creating end-to-end integrated domain/product-specific tool chains is considered to be a primary impediment for the faster penetration of model-based design in CPS industry. Large system companies face immense pressures to deliver safe and complex systems at low cost. Tools are at the heart of their engineering process covering the full spectrum of requirements, design, manufacturing and operations support. The internal tool landscapes of large aerospace and automotive

companies contain ~3000–5000 distinct tools totaling several hundreds of millions of dollars in internal investments [10]. End-to-end tooling for these complex CPS product lines spans too many technical areas for single tool vendors to fully cover. In addition, a significant part of the companies' design flow is supported by in-house tools that are proprietary and capture high value design intellectual property (IP). In many areas, such as drivetrain electronics in the automotive industry, production tool suites include a combination of in-house and commercial-off-the-shelf (COTS) tools in the approximate ratio of 70% and 30%, respectively. The development and use of in-house tools is not necessarily the result of deficient COTS offerings, but, rather, it is an essential part of the innovation process that yields competitive advantage via improved product quality and productivity. The primary technology barrier that slows down this process and makes integration of in-house tools with third party tools extremely expensive and error prone is the lack of modern tool integration and deployment platforms.

Seamless integration of end-to-end tool chains for highly automated execution of design flows is a complex challenge of which successful examples are rare, even after massive investment by companies. Vendors provide limited integration, primarily of their own tools, with a few cross-vendor integrations for particularly dominant tools (e.g., integration with DOORS, Matlab, Word, or Excel). This limitation results in design flows that consist of islands of integrated tool subchains, bridged by various *ad hoc*, semi-automated, or manual stopgaps. These stopgaps impose a variety of costs: additional work in performing manual transformations, additional work in guarding against divergence between multiple representations, and forgone analysis opportunities, to name just a few.

Truly transformational impact requires an approach for composing an end-to-end integrated tool chain from a heterogeneous collection of COTS, open source, and proprietary tools. The ideal solution would support tools from multiple vendors, and allow companies themselves to include the most closely guarded proprietary tools. Such a truly integrated toolset would yield significant improvements in productivity and decreases in design time by eliminating the unnecessary work associated with the existing integration mechanisms and shortening the learning curves associated with diverse, un-integrated tool suites.

In spite of its large significance, integrating design tool chains today relies on *ad hoc* methods: engineering processes are defined, tools are selected covering parts of the process, and the emerging isolated islands in the design flow are manually bridged with huge recurring cost, or patched together with opportunistically constructed data interchange mechanisms that cannot be maintained and evolved.

The main contributions of this paper are the integration technologies developed for CPS design automation tool chains. This integration technology is structured around three horizontal integration platforms: the model integration platform, the tool integration platform, and the execution integration platform. The primary focus of this paper are the model and tool integration platforms representing key technology components of model-based design. These platforms incorporate domain agnostic methods and tools for comodeling CPS artifacts and engineering processes that can be instantiated in domain-specific contexts. We use the OpenMETA design flow that we developed for the FANG ground vehicle challenge as an example for exposing the integration challenges. We argue that establishing model- and tool-integration platforms for CPS design automation helps to create decoupling between domain-specific, and frequently proprietary, engineering processes of system companies from their actual implementation incorporating a large suite of tools and extensive IT infrastructure. Since our primary focus is integration technologies for design automation systems, we do not intend to provide an exhaustive survey of specific CPS design methodologies, but leave it to the excellent papers in the literature [1]–[4], [11], [12], [23].

In Section II, we provide an overview of the model-based design process and the integration architecture in the context of the OpenMETA design flow, and identify the role of the model- and tool-integration platforms. Section III focuses on the model-integration platform and provides details on the methods and tools used for semantic integration. Section IV focuses on the tool-integration platform. Section V summarizes the lessons learned and describes current research directions.

II. OVERVIEW OF THE MODEL-BASED DESIGN PROCESS

As described before, a model-based design process receives requirement models R and environment models E, and synthesizes a system model S such that $\| E \vDash R$. Naturally, system design from "scratch" would not be practical, therefore the structure of the design process must allow for reusing as much design knowledge—models, processes, and tools—as possible. There are two kinds of design knowledge where reuse is critical: system models and testing/verification methods with related tools. These considerations are reflected in the conceptualization of the design process shown in Fig. 2.

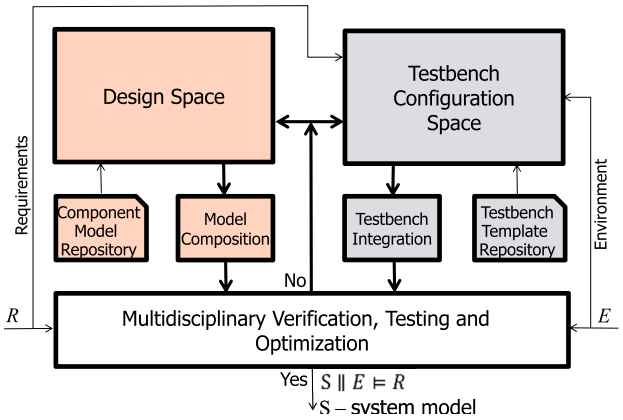

Fig. 2. ***Conceptualization of the design process.***

On the modeling side, model-based design is transformed into model- and component-based design, where previous design knowledge for the system to be constructed is available in a component model (CM) repository. The repository includes a set of parameterized CMs $C = \{C_i(x, p)\}$, each including a parameter vector x and typed ports p representing the component interface. The CMs are instantiated in specific architectures by setting their parameter values. For a system model S, $C_S = comptypes(S)$ denotes the set of CM types instantiated in S (possibly multiple times) and $C_{SI} = comps(S)$ yields the set of instantiated CMs. The architecture of a system S is defined by a labeled graph G_S, where the vertices are the ports of its components and the edges represent interactions corresponding to the type of the connected ports. An architecture is well formed if G_S satisfies a set of composition constraints $\mathbf{\Phi}$ over G_S derived from the semantics of the interaction types.The sets of component types C and composition constraints $\mathbf{\Phi}$ define a design space

$$D \stackrel{\text{def}}{=} \{S \mid G_S \vDash \mathbf{\Phi}, comptypes(S) \subseteq C\}$$

that includes all possible system architectures that can be composed from the instances of the parameterized component library subject to the composition constraints. The design process needs to synthesize an $S \in D$ design model (DM), for which $S \parallel E \vDash R$.

As Fig. 2 illustrates, the design process that should yield an acceptable DM is conceptualized as a design space exploration process that—by using some exploration strategy—incrementally narrows the size of the initial design space until (one or more) satisfying design is found. The core components of this process are the multidisciplinary verification, testing, and optimization activities that evaluate design points against different requirements in R while operating in targeted environments E. Reusing analysis tools and processes is a significant goal, therefore we use modeling and model-based integration for the design and implementation of the design automation process as well. From the point of view of the design automation process, these evaluations are performed by testbenches that incorporate specific configurations of tools for implementing an analysis flow over DMs. The goal of the testbenches is to make a decision if a design satisfies a subset of the requirements. Testbenches are linked to requirement categories (e.g., mobility requirements that are tested using a dynamic simulation testbench) and well suited for model-based integration. Testbench models that incorporate the model of an analysis flow and tool interfaces can be templatized and placed in a testbed template repository. A testbed template is instantiated by configuring it with specific requirement models, and with a suite of DMs required by the incorporated analysis tools. The testbench integration process deploys a configured testbench for execution.

The last essential condition for improving reuse in the design automation process is to enable decoupling between the system modeling and analysis sides by introducing a model composition process (see Fig. 2) that composes integrated, analysis-specific system models from the model of the system architecture G_S and the set of instantiated CMs C_{SI}.

A. Overview of an Example Tool Architecture: OpenMETA

The design process described above was prototyped and tested in the OpenMETA tool suite in designing the power train and the hull of an infantry amphibious vehicle. This particular challenge was focused on the design of the drivetrain and associated mobility subsystems of an infantry vehicle (fast adaptable next-generation ground vehicle FANG). Besides the physical (CAD) and functional (CyPhy/Modelica) requirements of the drivetrain and mobility subsystems they had to be packaged inside a predefined volume in the form of a 3-D surrogate hull. The designs were evaluated for driving performance on land and water as well as for manufacturing lead-time and anticipated unit manufacturing cost.

Prototyping the conceptual design flow described above required several categories of tools that had to be integrated to allow execution of a highly automated workflow. The integrated tools can be broadly categorized as: 1) authoring tools that would allow definition of CMs, and design space models (DSMs), and to populate component and design repositories that can be used/reused by system engineers for designing a specific system; 2) model composition tools that transform, compose, and derive inputs for domain-specific analysis tools from the system DMs; 3) domain-specific analysis testbenches and tools for analysis and evaluation of the candidate system using models of progressively deepening fidelities; and 4) analytics and visualization tools for visualization of analysis results as well as for interactively conducting design trades.

We will briefly exemplify the usage of these tools in context of a subset of the FANG vehicle challenge noted above. The simplified view of the tool configuration is shown in Fig. 3.

1) Requirements: In OpenMETA, the requirements inform the testbenches that are developed and used to assess the candidate design. The drivetrain model of the FANG vehicle had 19 performance requirement types that are divided into five subcategories: speed, acceleration, range, temperature/cooling, and fuel economy. The speed requirement category contains five maximum speed requirements (forward speed, hill climb on different surfaces, and reverse speed) and two average speed requirements. Each requirement has a threshold and an objective value. The threshold is the pass/fail mark, and the objective represents the ideal outcome. A score can be assigned to each metric by comparing it to the threshold and objective; exceeding the objective will not necessarily have increased benefit. A design's overall (weighted) score is computed based on the requirement structure and analysis results, and represents the quality of the design, which facilitates the quantitative comparison of different design solutions.

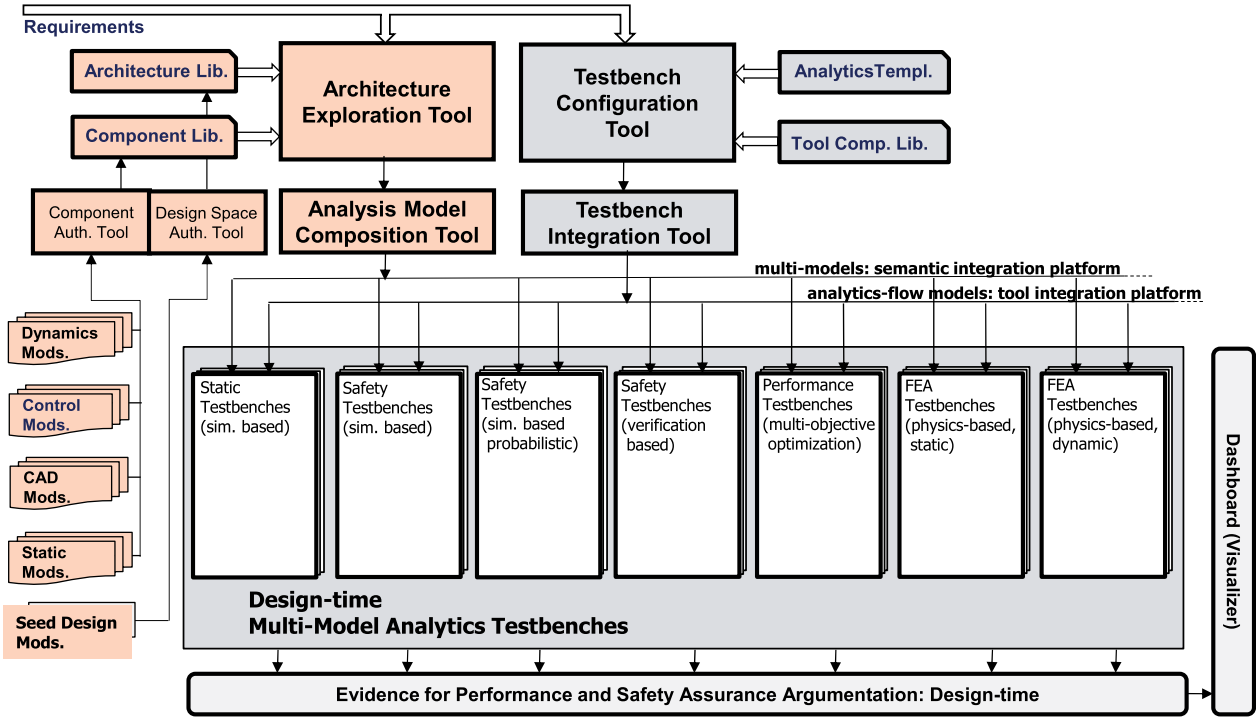

Fig. 3. ***OpenMETA tool architecture.***

2) Component Authoring and Repository: The cyber–physical CM used in OpenMETA is a novel construct that was designed to address challenges unique to the cyber–physical design. The heterogeneity inherent in cyber–physical components implied that a singular formalism and model would not be adequate to represent components. In the FANG vehicle design, for example, the engine was one of the most critical components in terms of its impact on the design requirements. The power curve of the engine (a multiphysics model) models its ability to deliver torque at different engine speeds, and corresponding fuel consumption, and plays a primary role in assessing the gradeability (i.e., whether the vehicle will be able to climb a slope at a desired speed) requirements of the vehicle design. The physical geometry of the engine (a CAD geometry model) is a primary determinant of the 3-D placement, manufacturability, and serviceability requirements of the vehicle design. The second challenge was that these different models are typically developed in mature engineering tools (such as Dassault System's Dymola[1] tools, or PTC's Creo[2]) with significant investment, and represent the knowledge capital of an organization.

The cyber–physical CM had to leverage these artifacts without requiring a redesign in a new formalism. The third challenge was that the CMs, as the primary unit of reuse, and their authoring being decoupled from use, had to be comprehensively characterized, described, packaged, curated, and cataloged according to a taxonomy, such that it would allow a systems engineer to later use the components in a system design. These challenges were addressed by developing a component specification language, a component container format (described in [13]–[15]) to author and package the components, and a CM exchange to repository and organize curated components. In OpenMETA, we leveraged GME, a metaprogrammable modeling environment [16], [17], and customized it with the component specification language for authoring components.

Table 1 shows an enumeration of different engine components (and their key parameters) authored in the component specification language (described later) and included in the component repository for use in the vehicle design.

3) Design Space Authoring and Repository: Typically in engineering organizations, seed designs are baseline architectures that have been proven through prior usage, and constitute the starting point from which new designs are derived by introducing variations in architectural topologies and innovations in the component technologies. In OpenMETA, this conceptual process is systematized by formally representing the baseline architectures using a cyber–physical systems modeling language (CyPhyML) (described in [13]). The (partial) seed design for the vehicle consists of a drivetrain (engine and transmission), left-hand- and right-hand-side drive (drive shaft and final drive), cooling system, fuel tank, batteries, and software controllers (engine control unit and transmission control unit),

[1]https://www.3ds.com/products-services/catia/products/dymola/latest-release/

[2]https://www.ptc.com/en/cad/creo

Table 1 Engine Components in the Repository

Supplier	Type	HP
Caterpillar	C9 280kW	375
Caterpillar	C11 313kW	420
Caterpillar	C15 444kW	595
Caterpillar	C18 597kW	800
Caterpillar	C27 597kW	800
Caterpillar	C32 709kW	950
MTU	MT883 644kW	864
MTU	6V199 261kW	350
MTU	6V199 335kW	455
MTU	6V199 430kW	585
MTU	8V199 530kW	720
MTU	8V199 603kW	820
MTU	8VMT881 736kW	1000
MTU	12VMT883 1103kW	1500
MTU	6R106Euro3 240kW	325
Deutz	**BF6M1015M group A**	**290**
Deutz	BF6M1015MC group A	385
Deutz	TCD2015V6M group A	440
Deutz	TCD2015V8M group A	600
Cummins	QSM 350HP FR20019	350
Cummins	QSM 400HP FR20003	400
Cummins	QSX 400HP FR10581	400
Cummins	QSX 500HP FR10583	500

organized in a topology that represents the interactions (energy flow, signal flow, etc.) between different subsystems and components.

In OpenMETA, these seed designs can be systematically extended and turned into a design space by introducing alternative components and parameterization. For example, the key components of the vehicle design that impact the speed and acceleration requirements are the engine and the transmission components. The seed design is augmented by replacing the specific engine component [Deutz BFM1015M (290HP)], with the set of alternative engines shown in Table 1, and similarly the single transmission component is replaced with a set of transmission components from the repository. The discrete design space resulting from the single design point has the same topological structure, however, now encapsulates 200 (25 engines, 8 transmissions) candidate designs.

4) Design Space Exploration: The OpenMETA design flow is implemented as a multimodel composition and testing/verification process that incrementally shapes and refines the design space using formal, manipulable models [14], [15]. The model composition and refinement process is intertwined with testing and analysis steps to validate and verify requirements and to guide the design process toward the least complex, therefore, the least risky and least expensive solutions. The design flow follows a progressive deepening strategy, starting with early design-space exploration covering very large design spaces using abstract, lower fidelity models and progressing toward increasingly complex, higher fidelity models and focusing on a rapidly decreasing number of candidate designs.

The META design flow proceeds in the following main phases.

1) Combinatorial design space exploration using static finite domain constraints and architecture evaluation (static exploration testbench).
2) Behavioral design space exploration by progressively deepening from qualitative discrete behaviors to precisely formulated relational abstractions and to quantitative multiphysics, lumped parameter hybrid dynamics models using both deterministic and probabilistic approaches (simulation-based dynamics testbenches and symbolic safety verification testbenches).
3) Geometric/structural design space exploration coupled with physics-based nonlinear finite element analysis of thermal, mechanical, and mobility properties.
4) Cyber design space exploration (both HW and SW) incorporated in dynamics testbenches.

We gained the following insights with the implementation of OpenMETA.

1) In a model-based design automation process, both component modeling and analysis flow modeling are important sources of reusability, therefore affordability. In addition, model composition and transformation technology is essential for composing integrated analysis models in the form required by the various analysis tools.
2) The complexity of the synthesis process is fundamentally influenced by the size, granularity, and richness of the model library and the restrictiveness of composition constraints. In the extreme case, when the component library includes a single parameterized component that can potentially satisfy all the requirements, the design process is reduced to verifying if a satisfying parameterization exists and to optimizing the system parameters for some performance metrics by means of the multidisciplinary verification, testing, and optimization process.

B. Need for Horizontal Integration Platforms

Automated execution of the design-space exploration process requires the seamless integration of heterogeneous models and tools. A core technical challenge in creating OpenMETA was complementing the traditional, vertically structured, and isolated model-based tool suites with horizontal integration platforms for models, tools, and executions as shown in Fig. 4.

The functions of the integration platforms are summarized as follows.

1) *Model integration platform:* As shown in Fig. 4, a CPS design-space exploration process cuts across

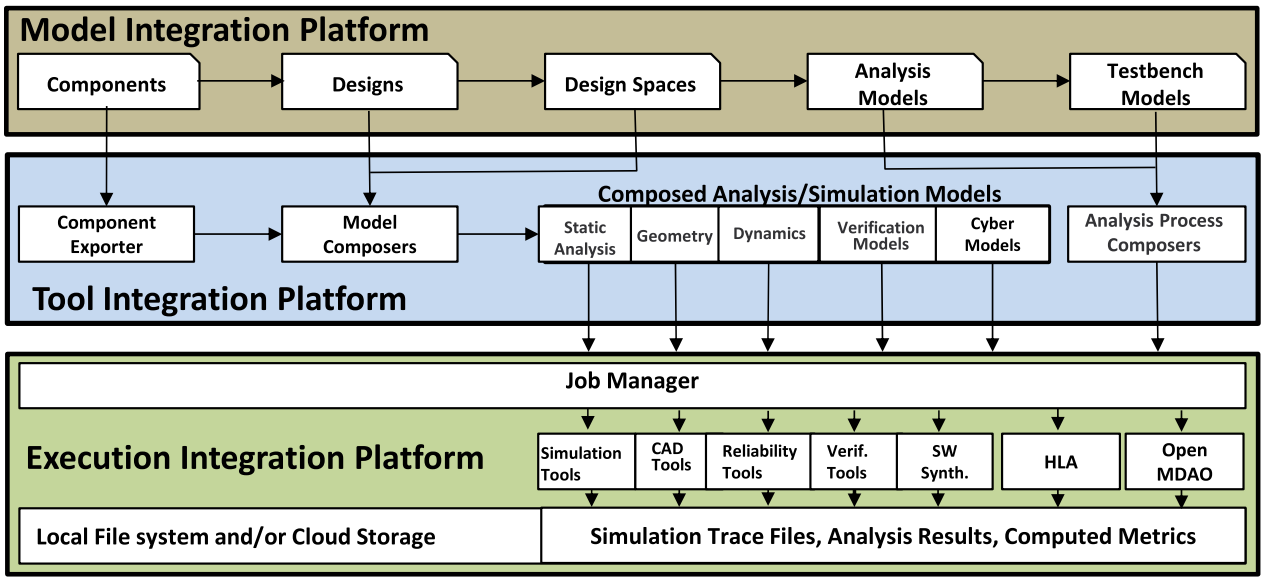

Fig. 4. ***Horizontal integration platforms for CPS design automation.***

traditionally isolated design domains supported by a wide range of tools and valuable model libraries that are accessible in popular COTS or open source tools. Many of the languages are complex, not necessarily because of the innate domain complexity, but auxiliary complexities caused by an insatiable push for generality and various incidental tool functions. The CPS design process not only utilizes these different modeling languages but also it needs to exploit their interaction expressed as cross-domain models. Given the semantic complexity of integrating and relating heterogeneous models and modeling languages, we considered model integration as a fundamental functionality that needs to be supported by specific methods and tools. The model integration platform includes languages and tools for defining cross-domain model integration languages, formally modeling their semantics and the semantics of model transformations.

2) *Tool integration platform:* Referring to Fig. 3, the design-space exploration process incorporates a suite of testbenches that implement analysis processes to test/verify satisfaction of requirements. The specification and integration of testbenches is also model based, therefore the tool integration platform incorporates both model composers for generating the tool-specific product models for analysis and analysis process composers generating executable version of testbench models. The executable testbench models link the tools incorporated in the analysis flow, the corresponding product models, and the integration models, and the appropriate version of the tool integration framework used: open multidisciplinary design analysis and optimization (MDAO) optimization tool [18] for parametric optimization, or the high level architecture (HLA) for distributed simulation [19].

3) *Execution integration platform:* The automated execution of analysis processes invokes a variety of tools that need to receive and produce models, and need to be scheduled as the analysis flow requires. The execution integration platform incorporates the services and mechanisms required for running the analyses using deployed tools and generating the analysis artifacts. The first test of the openMETA tool suite was the FANG-1 challenge prize competition [7], [8] focusing on the drivetrain and mobility subsystems. Since the competition was also a crowdsourcing experiment with over 1000 individuals grouped in over 200 design teams [8], access to the design tools and model libraries by the competitors across the country was in itself a major challenge. Our execution integration platform (VehicleForge) provided web-based delivery platform for the integrated design tools, supported their cloud-based deployment through a software-as-a-service-delivery model and incorporated a range of collaboration services for design teams. While the execution integration platform is an important part of design automation systems, we do not discuss it in this paper. For further information, see [21].

The horizontal model, tool, and execution integration platforms adopt the philosophy and key characteristics of platform-based design [22], [23].

1) Construction of the integration platforms represent distinct, domain-independent challenges.

2) The integration platforms are not just conceptual but include specific sets of methods, services, and tools that are domain agnostic.
3) We believe that adoption of generic integration platforms accelerates the construction of design automation tool chains from reusable components.

In the following sections, we will focus on the model integration platform and the tool integration platform, present our solutions developed first in OpenMETA, and use examples for demonstrating the integration process.

III. MODEL INTEGRATION PLATFORM

We believe that the single most important change to achieve the correct-by-construction design is the introduction and systematic use of cross-domain modeling. However, creating design tool chains that cover all potentially relevant CPS modeling abstractions and satisfy the needs of all application domains is unrealistic. In addition, tool chains that are highly configurable to specific application domains are not available. Consequently, our objective with the introduction of the model integration platform was to enable the rapid construction of domain-specific, end-to-end tool suites for CPS design domains by supporting the modeling of cross-domain interactions in heterogeneous modeling environments.

In a naïve approach, the challenge of creating an integrated CPS design automation tool chain crossing different design domains is considered to be a tool interoperability problem that can be taken care of with appropriate data conversions and tool APIs. In complex design domains, these approaches inevitably fail due to the rapid loss of control over the semantic integrity of design flows. The primary reason for this is that the key stakeholders—tool vendors on one side and systems companies on the other side—need to respond to different pressures that drive their product lifecycles. The interest of system companies is to control and evolve their product data and engineering processes and use the best-of-breed tools for implementing their design flows. Tool companies are interested in expanding the relevance and reusability of their tools across product categories and capturing essential parts of end-user design flows.

The role of the model integration platform is to facilitate decoupling between these two sides by explicitly modeling their relationship. In the model- and component-based framework of OpenMETA (see Figs. 2 and 3), we structured the model integration platform in the following two layers.

1) Tool agnostic model integration layer that incorporates a model integration language (CyPhyML) for representing a) CMs $C(x,p)$; b) designs $S = \langle C_{SI}, G_S \rangle$; c) design spaces D; d) architecture G_S and composition constraints Φ; e) cross-domain interactions; f) data model interfaces for tools; g) models of engineering processes; and h) model transformations for composing analysis models. The model integration layer is supported by metaprogrammable modeling and model transformation tools [16] for rapidly defining and evolving model integration languages and transformations. Details of the model integration layer will be discussed in Sections III-A–III-C.
2) The semantic integration layer provides methods and tools for formally representing the semantics of the elements of the model integration layer. The primary reason for introducing formal modeling (metamodeling) of the integration languages, data models, and model transformations is that model integration languages are designed to be evolvable: as product categories, engineering processes, and tool interfaces are changing, the integration models need to be evolved without compromising semantic integrity. Details of the semantic integration layer will be discussed in Sections III-D and III-E.

The primary benefit of this approach is that model integration languages, expressing a tool agnostic view of product architectures and related engineering processes, can be designed for simplicity: they do not incorporate all the semantic details present in various domain-specific tools, but focus on cross-domain interactions and composition.

In the next parts of this section, we provide highlights from the model integration language, CyPhyML, that was developed for the FANG design challenge in OpenMETA and summarize key features of the selected formal framework FORMULA [26], [27]. However, we emphasize that the essence of the OpenMETA approach is flexibility; the choices in the design of CyPhyML were driven by the specific circumstances of the FANG challenge problem.

A. Developing a Model Integration Language: CyPhyML

The CyPhyML model integration language, as noted earlier, was designed to serve the needs of performing automated design-space exploration for the drivetrain and mobility subsystems of the FANG vehicle [28] to be followed by a hull design challenge. Consistently with the goals of the competition, the design process targeted the system-level design, roughly corresponding to conceptual and preliminary design phases in the usual systems engineering process, with a path toward a detailed design. The targeted granularity was on the subsystem level, including commercial-off-the-shelf (COTS) components such as engines. While the design space included embedded controllers, the decomposition hierarchy was strongly dominated by physical structure.

The key considerations in the design of CyPhyML were as follows:

1) abstraction layers for integrated design flow;
2) granularity and coverage in the CM library;

3) principles used for constructing the design space;
4) scope and depth of the requirements and key performance parameters.

For example, evaluation of a drivetrain design for the mobility requirement "maximum speed hill climb sand" requires a testbench that simulates the lumped parameter dynamics of the drivetrain model S composed with appropriate terrain data E. For a given drivetrain architecture composed of a selected suite of components, the OpenMETA model composer for lumped parameter dynamics accesses the dynamics models of the individual components in the architecture and composes them into a system model that can be simulated by a simulation engine (OpenMETA used OpenModelica [29], [30] as the primary simulation engine). The CMs and the composition mechanism must be flexible enough to enable the use of CMs of different levels of fidelity (even represented in different modeling languages, e.g., Modelica models, Simulink/Stateflow models, FMUs, or bond graph models). The $S \parallel E$ composition of the drivetrain model S and the terrain model E is specified in the testbench model by modeling the interaction between the terrain and the drivetrain. The testbench links the models to the simulation engine to gain an executable for the evaluation of the "maximum speed hill climb sand" performance parameter. Since all design points in the overall design space have the same interface, the testbench model can be linked to a design space with many alternative, parameterized architectures. Using the open MDAO optimization tool [18], a multiobjective parametric optimization can be performed if the exploration process requires it.

These type of considerations led to the detailed design of model integration languages, specifically the cyber–physical system modeling language (CyPhyML). CyPhyML is the composition of several sublanguages as follows.

- CMs that incorporate: 1) several domain models representing various aspects of component properties, parameters, and behaviors; 2) a set of standard interfaces through which the components can interact; 3) the mapping between the component interfaces and the embedded domain models; and 4) constraints expressing cross-domain interactions.
- DMs that describe system architectures using assemblies, components and their interconnections via the standard interfaces.
- DSMs that define architectural and parametric variabilities of DMs as hierarchically layered alternatives for assemblies and components.
- Analysis models (AMs) that specify data and control interfaces for analysis tools.
- Testbench models (TMs) that 1) specify regions in the design space to be used for computing key performance parameters; 2) define analysis flows for computing key performance parameters linked to specific requirements; and 3) provide interfaces for visualization.

B. Component Modeling in CyPhyML

Components in CyPhy are the basic units for composing a design. The CMs represent several things about the actual component, including its physical representations, connections, its dynamic behavior, properties, and parameters. To achieve the correct-by-construction design, the system models are expected to be heterogeneous multiphysics, multiabstraction, and multifidelity models that also capture cross-domain interactions. Accordingly, the CMs, in order to be useful, need to satisfy the following generic requirements.

1) Elaborating and adopting established, mathematically sound principles for compositionality. The semantics of composition frameworks are strongly different in physical dynamics, geometric structure, and computing that need to be precisely defined and integrated.
2) Inclusion of a suite of domain models (e.g., structural models, multiphysics lumped parameter dynamics, distributed parameter dynamics, manufacturability) on different levels of fidelity with explicitly represented cross-domain interactions.
3) Specification of component interfaces required for heterogeneous composition. The interfaces need to be decoupled from the modeling languages used for capturing the embedded domain models. This decoupling ensures independence from the modeling tools selected by the CM developers.
4) Established bounds for composability expressed in terms of operating regimes where the CMs remains valid.

The CPS CM must be defined according to the needs of the design process that determines: 1) the type of structural and behavioral modeling views required; 2) the type of component interactions to be accounted for; and 3) the type of abstractions that must be utilized during design analytics. We believe that it does not make sense to strive for a "generic" CPS CM, rather, CMs need to be structured to be the simplest that is still sufficient for the goal of the "correct-by-construction" design in the given context.

The CyPhyML component model (CCM) was designed to integrate multidomain, multiabstraction, and multilanguage structural, behavioral, and manufacturing models and to provide the composition interfaces for the OpenMETA model composers consistently with the needs of power train and hull design [15]. In Fig. 5, we illustrate the overall structure of a CCM for an engine. The main elements of the CM are the port-based component interfaces, the related embedded domain models, and constraints representing cross-domain interactions.

Based on the needs of system level integration in FANG, we chose the following CM elements.

Parameter/property interfaces include a set of parameters/properties ports P_{pp} characterizing the components using established ontologies. Properties represent

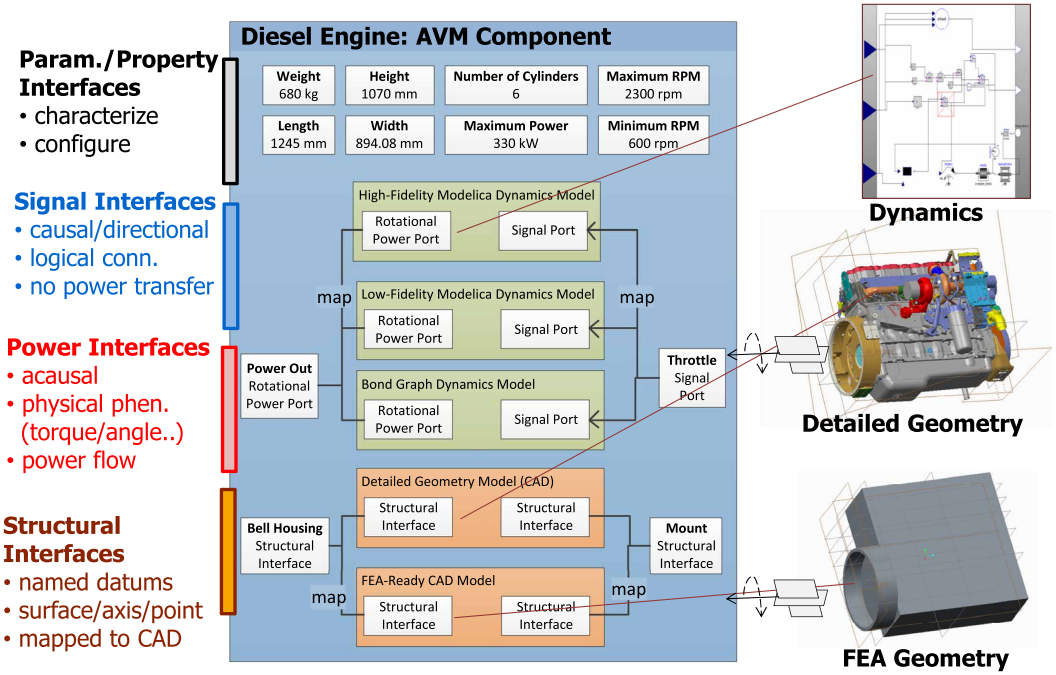

Fig. 5. ***Illustration of the structure of the CCMs.***

component attributes such as material, mass, or an attribute associated with its maturity, e.g., TRL. Parameters are variables that can be changed as part of the design process. Properties can be fixed, or could admit an uncertainty bound with a distribution. For parameterized components (see parameters below), properties can be variable and algorithmically associated with parameters. Properties can also be scalar or vector and have a unit associated with them. CyPhyML employs a standard unit system based on NASA's QUDT unit ontology. Fig. 5 shows examples of the properties characterizing a diesel engine. Parameters are mechanisms to capture component variability, allowing customization for a specific instantiation or use. For example, a variable-length drive-shaft component can be represented with a length parameter that can be configured for a specific use. The parameters are largely similar to properties from a specification standpoint and inherit from a common base class, are associated with a unit, and have a specified default value as well as a validity range. Design space exploration tools use component parameters for tuning or optimization of a design, or adaptation of a component for a specific use [13], [31].

Signal interfaces are defined by the P_{in} set of continuous-time input signal ports, and the P_{out} set of continuous-time output signal ports. Interactions via signal interfaces are causal and defined by models of computations [32], [33]. Models of dynamics implemented by embedded controllers are represented using causal modeling approaches using modeling languages such as Simulink/Stateflow [34], ESMoL [35], functional mockup units [36], or the Modelica Synchronous Library [37].

Power interfaces represent physical interactions using power flows. With the emphasis on power flows and the ensuing distinction between physical variables, we follow the port-Hamiltonian approach to physical system modeling. From the modeling point of view, the approach models the systems as energy storing and dissipating components that are connected via ports to power conserving transmissions and conversions. Accordingly, the power interfaces are represented using acausal modeling framework [38]–[40] to achieve compositionality in modeling physical dynamics. CyPhyML incorporates the following port types: P_{rotMech} is a set of rotational mechanical power ports; $P_{\text{transMech}}$ is a set of translational mechanical power ports; $P_{\text{multibody}}$ is a set of multibody mechanical power ports; $P_{\text{hydraulic}}$ is a set of hydraulic power ports; P_{thermal} is a set of thermal power ports; and P_{electric} is a set of electrical power ports. Each power port is associated with two physical variables (effort and flow), whose product yields power. In this approach, dynamic models are represented as continuous-time differential algebraic equations (DAEs) or hybrid DAEs. Since model libraries may come from different sources, CMs are potentially expressed in different modeling languages such as Modelica [37] or bond graphs error (although we dominantly used Modelica-based representations). The use of multifidelity models is important in assuring scalability in virtual prototyping of systems with a large number of complex components.

Structural interfaces provide interaction points to the geometric structure of components usually expressed as CAD models of different fidelity. Geometry is a fundamental aspect of the CPS design. The structural interface

incorporates geometric ports P_{geom}, where component geometries can be connected and provide the geometric constraints from which the composed geometry of an assembly is calculated. They are the basis for deriving geometric features of larger assemblies and performing detailed finite element analysis for a range of physical behaviors (thermal, fluid, hydraulics, vibration, electromagnetic, and others [15]).

Domain models capture model artifacts specific to supported domain tools. The domain models in CyPhyML are a wrapping construct that refers to the domain model in its native representation language and tool. The domain model exposes the core interface concepts that are used to map into the component interfaces. For example, the component power interface maps directly into Modelica power ports. The supported set of domain models are a point of expansion for the language. In many OpenMETA transition projects, several new domain models have been added. For the FANG-1 competition, supported domain models included Modelica models, bond graph models, Simulink/Stateflow models, STEP-compliant CAD models, and restricted manufacturing models representing cost, lead time, and process information. CCM allows multiple domain models for each domain thereby enabling a multifidelity representation of the component. In case there are multiple domain models, they are tagged with fidelity tags. The fidelity tags are kept freeform to allow users', component modelers', and tool developers' flexibility in specification and usage, since there is currently no universally accepted taxonomy of model fidelity.

Formulas are modeling constructs for specifying parametric interdependences across domain models. They are used in conjunction with ValueFlow that establishes the dependency relation while formulas define the mathematical transformation between the parameters. They are used extensively as a method for cross-domain modeling.

The construction of CCMs from a library of domain models (such as from the Modelica models representing lumped parameter dynamics of physical or computational behaviors, the CAD models, the models of properties and parameters and cross-domain interactions, and the mapping of domain modeling elements to component interfaces) is time consuming and error prone. The most important challenge is that the CCM defines interface semantics (e.g., acausal power interfaces) that represent restrictions over the modeling languages used in the behavior and structural models embedded in the components. These restrictions need to be enforced, otherwise a semantic mismatch is created between the CCM and the embedded domain models.

To solve this problem, OpenMETA includes a full tool suite for importing domain models (such as Modelica dynamic models), integrating them with standard CCM interfaces, automatically checking compliance with the standard, and automatically checking model properties, such as restrictions on the types of domain models, well-formedness rules, executability, and others. Based on our direct experience, the automated model curation process resulted in orders-of-magnitude reduction in a required user effort for building CM libraries.

In summary, CPS CMs are containers of a selected set of domain models capturing those aspects of component structure and behavior that are essential for the design process. While the selected modeling domains are dependent on CPS system categories and design goals, the overall integration platform can still be generic and customizable to a wide range of CPSs.

C. Design and Design Space Modeling

The $S = \langle C_{SI}, G_S \rangle$ generic DM is refined in CyPhyML into the tuple

$$S = \langle C_{SI},\ A,\ F,\ P,\ \mathit{Contain},\ \mathit{portOf},\ E_P,\ E_S,\ E_G,\ E_V \rangle$$

with the following interpretation:

- C_{SI} is a set of component instances;
- A is a set of component assemblies;
- F is a set of formulas;
- $B_e = C_{SI} \cup A \cup F$ is the set of design elements;
- P is the union of ports included in the component interfaces;
- $\mathit{contain}: B_e \rightarrow A^*$ is a containment function, whose range is $A^* = A \cup \{\mathit{root}\}$, the set of design elements extended with a special root element *root*;
- $\mathit{portOf}: P \rightarrow B_e$ is a port containment function, which uniquely determines the container of any port;
- $E_P \subseteq P_P \times P_P$ is the set of power flow connections between power ports;
- $E_S \subseteq P_S \times P_S$ is the set of information flow connections between signal ports;
- $E_G \subseteq P_G \times P_G$ is the set of geometric structure connections between structure ports;
- $E_V \subseteq P_{PP} \times P_{PP}$ is the set of value flow connections between parameter and property ports.

The restrictions over composing models by allowing the formation of only four types of flows E_P, E_S, E_G, E_V represent the composition constraints $\mathbf{\Phi}$ defined earlier. Even with these constraints, if we adopted the design space as the set of all possible combination of components via well-formed flows

$$D \stackrel{\text{def}}{=} \{S \mid G_S \vDash \mathbf{\Phi}, \mathit{comptypes}(S) \subseteq C\}$$

we would get design space that incorporates almost exclusively useless design points. To restrict the design space, we used a common heuristics that starts all new designs from modifying an old one. Accordingly, we provided for designers seed designs S_s that are functional, fully instantiated, and tested architectures together with a set of operations that they could use to expand the S_s design points into a design space by introducing structural alternatives (alternative,

compatible components and assemblies) and parametric ranges. While this approach rapidly converges to new designs, it certainly biases initial thinking in some direction. We believe that design space construction is a hugely important area of design automation that deserves to receive significant attention in the future.

From the point of view of the design of the CyPhyML model integration language, the central issue is how the selected component modeling and design modeling concept impacts the semantics of the integration language. It is important to note that we do not intend here to extend the considerations to the details of modeling languages used for CPSs; we leave this to the many excellent articles [1]–[4], [12], [33], [41].

The relationship between the semantics of the domain modeling languages and the model integration language is illustrated in Fig. 6. As the figure shows, the model integration language provides a precisely controlled level of decoupling between the frequently vastly complicated domain modeling languages (such as Modelica for lumped parameter dynamics or STEP/IGES for CAD) using semantic interfaces. The primary goal for their design is simplicity: select semantics that is sufficient for the integration of DMs and engineering process models. To preserve flexibility without losing precision, the semantics of the model integration language and that of the semantic interfaces need to be explicitly defined. These semantic specifications are collected in the semantic backplane in OpenMETA. There are two kinds of semantic specifications required.

1) Structural semantics (or static semantics) [24] that specifies constraints that all model instances of the language need to satisfy. In the case of CyPhyML, this is particularly important, since the composition rules for creating models from components have many type restrictions for obtaining valid power flows and a geometric structure. A more detailed description is available in [42].

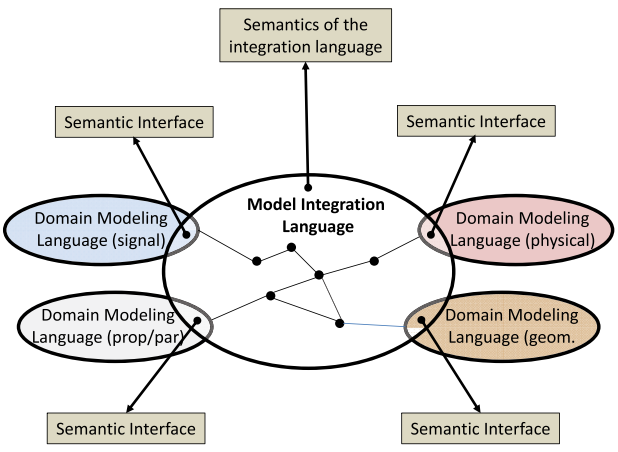

Fig. 6. ***Semantic interface between the integrated languages and the model integration language.***

2) Behavioral semantics for the composition operators that connect interface ports. Several examples for the specification of denotational semantics for composing power flows as constraints over effort and flow variable pairs (Kirchoff laws) are presented in [43] and [44].

D. Formal Framework for Semantic Integration: FORMULA

The "cost" of introducing a dynamic model integration language is that mathematically precise formal semantics for model integration had to be developed. Because no single tool uniformly represents all aspects of a CPS, a holistic CPS engineering process must include a method to define, compose, store, and evolve models in a tool-independent fashion. The only truly unambiguous and tool-independent method is a mathematical one, which defines: 1) a standard representation for models that is suitable for many domains; 2) a formalism for separating valid models from invalid ones with an acceptable degree of precision; and 3) a class of functions for transforming one model into another. These definitions provide a semantic backplane supporting CPS model composition and evolution throughout the engineering lifecycle.

Armed with this semantic backplane, tools may be implemented that automate its application. For example, a tool might be implemented that efficiently checks if a CPS model is valid (e.g., does not violate certain cross-domain electromechanical constraints) according to unambiguously stated formal validation rules. One implementation may be more (or less) efficient for specific operations, such as the transformation of large existing models or the synthesis of new models from scratch. Most importantly though, models, model validation, and model transformations remain as formal concise tool-independent objects.

For the remainder of this section we describe our experiences developing a semantic backplane for CPS, which led to the implementation of the FORMULA 2.0 system. Before discussing specific design decisions, it is important to emphasize a "no silver bullet" principle for semantics backplanes: A semantic backplane cannot capture the formal semantics of all modeling domains with perfect fidelity while simultaneously having practical implementations. For example, consider a semantic backplane that could validate vehicle models for complex errors in their physical dynamics. Any implementation of such a backplane most likely subsumes the functionality of Matlab or Modelica, which already have hugely complex implementations. Also, consider that a vehicle likely has embedded software written in a C-like language. A semantic backplane that could detect hard-to-find bugs in this code via a perfect understanding of C semantics most likely subsumes a C compiler, which again is another complex implementation. There is no silver bullet and no utopian formalism that has perfect formal

semantics while being practically implementable. Instead, a pragmatic approach needs to be taken. The backplane focuses on distinguishing models that are likely to be valid with an emphasis on cross-domain validation rules. It also supports model transformations that can create detailed projections of complete system models, and these can be further analyzed by domain-specific tools (e.g., ODE solvers, finite-element analysis tools, or software verification tools) without reinventing the entire wheel.

A semantic backplane must provide a convenient formalism for representing, validating, and transforming models. Each of these goals implies a nontrivial set of design decisions that balance convenience, expressiveness, and efficiency. In fact, many of the design decisions are related to ongoing debates in many fields of computer science.

- Model representation: CPS models take the form of graph-like structures (e.g., Matlab/Simulink), code-like structures (e.g., systems ODEs and C code), and relational structures (e.g., maps of model parameters to parameter values). Should these all be reduced to relations over atomic data types (e.g., alloy, logic programs without function symbols, relational databases)? Should they all be expressed as trees of data (e.g., JSON, XML, Coq)? Should they be some mixture of the two (NoSQL, document databases, graph databases)? Decisions at this level impose classic burdens on the comprehensibility of models.
- Model validation: How should the users express rules which separate (likely) valid from invalid models? Classic examples are first-order logic, graph constraints, (finite-) automata, and functional and procedural programs. Again, tradeoffs of expressiveness, comprehensibility, and efficiency occur here.
- Model transformation: How should functions be formalized that transform one model to another? Classic examples are string and tree transducers, term rewrite systems, functional and procedural languages, and text templates. Should this be a distinct formalism from model validation?

The approach we have taken represents models as sets of tree-like data (similar to modern NoSQL databases) allowing a uniform encoding of tree-like, graph-like, and relational structures. We have unified model validation and transformation using open-world logic programming (OLP), which allows both validation and transformation operations to be formalized as an extension of first-order logic with fixpoint operations. Axioms written in this logic can be dually understood as executable programs, providing engineers with an additional mental model for comprehending their specifications.

Our specifications are highly declarative and easily express complex domain constraints, rich synthesis problems, and complex transformations. Automated reasoning is enabled by efficient symbolic execution of logic programs into quantifier-free subproblems, which are dispatched to the state-of-the-art SMT solver Z3 [45]. FORMULA 2.0 has been applied within Microsoft to develop modeling languages for verifiable device drivers and protocols [46]. It has been used by the automotive embedded systems industries for engineering domain-specific languages [47] and design-space exploration [31] under hard resource allocation constraints. It has been used to develop semantic specifications for complex CPSs.

FORMULA 2.0 is released under an open-source license and can be found at https://github.com/Microsoft/formula.

Domains. The structure of models and validation rules for models are specified using algebraic data types (ADTs) and OLPs, as shown in the example below.

```
 1:  domain Deployments
 2:  {
 3:   Service  ::= new (name: String).
 4:   Node     ::= new (id: Natural).
 5:   Conflict  ::= new (s1: Service, s2 Service).
 6:   Deploy   ::= fun (s: Service => n: Node).
 7:
 8:   conforms no { n | Deploy(s, n), Deploy(s', n),
 9:                    Conflict(s, s') }.
10:  }
```

The `Deployments` domain formalizes the following cross-domain problem: There are services, which can be in conflict, and nodes, which can run services. Services must be deployed to nodes such that no node executes conflicting services. Lines 3–6 introduce data types to represent the entities of the abstraction. The conformance rule (lines 8–9) forbids conflicting tasks to run on the same node. This is an example of an OLP rule, which can be interpreted either as a logical axiom on valid models, or as a program that searches over a model checking for violations of this rule. Constructing a valid model for a fixed set of tasks, conflicts, and nodes is NP-complete. It is equivalent to coloring the conflict graph with nodes, demonstrating that model construction can be difficult for humans and machines.

Domain Composition. FORMULA 2.0 provides a composition of domains so abstractions can be composed. For example, `Deployments` can be constructed by gluing together two independent domains, and then adding additional validation rules and data types. Here is an example of a refactoring `Deployments` into three separate domains.

```
1:  domain Deployments extends Services, Nodes
2:  {
3:   Conflict   ::= new (s1: Service, s2: Service).
4:   Deploy    ::= fun (s: Service => n: Node).
5:
6:   conforms no { n | Deploy(s, n), Deploy(s', n),
7:                    Conflict(s, s') }.
8:  }
```

The domains `Services` and `Nodes` contain the data types and validation rules for a valid set of services and nodes. This composition semantically merges data types (or produces an error) and conjoins inherited constraints with new constraints. In this way, large complex domains can be built up from smaller pieces. The FORMULA 2.0 system provides a formal meaning for this composition and its implementation checks for many logical inconsistencies automatically, including semantically conflicting definitions of data types, unsatisfiable rules, and rules that produce badly typed inferences.

Models. Models are represented simply as sets of well-typed trees created from domain data types. Model modules hold the set of trees. Note that by allowing for a set of trees, it is possible to naturally represent the full spectrum of models from purely relational, to graph like, to fully tree like, and FORMULA 2.0 rules interact easily with this spectrum of representations.

Formally, a domain D is an OLP. A model M closes D with a set of facts, written $D[M]$. A fact is just a simple rule stating that a data element is provable, and so a model is simultaneously a set of facts. The properties of a model M are those properties provable by the closed logic program $D[M]$. In the example below, a model of an automotive system contains two services, one for voice recognition and one for handling the car's dashboard user interface. Due to computational constraints, these two services cannot be placed on the same compute node.

```
 1:  model Undeployed of Deployments
 2:  {
 3:    sVoice is Service("In-car voice recognition").
 4:    sDB    is Service("Dashboard UI").
 5:    n0     is Node(0).
 6:    n1     is Node(1).
 7:    Conflict(sVoice, sDB).
 8:  }
 9:  model Good of Deployments extends Undeployed
10:  {
11:    Deploy(sVoice, n0).
12:    Deploy(sDB, n1).
13:  }
14:  model Bad of Deployments extends Undeployed
15:  {
16:    Deploy(sVoice, n0).
17:    Deploy(sDB, n0).
18:  }
```

Model composition is like domain composition, and allows sets of trees to be combined. Each of these related models differs in the properties that can be derived from their contents (i.e., closed logic program).

- Deployments [Undeployed] does not satisfy conforms, because services are not deployed to nodes. (Violates validation rule in line 6 of `Deployments` domain, which requires an instance of the Deploy data type for every instance of Service data type.)
- Deployments [Good] satisfies conforms, because all services are deployed and all conflicts are respected.
- Deployments [Bad] does not satisfy conforms, because its deployments violate conflicts.

Partial Models. Partial models partially close domains. A partial model P is solved by a model M if all facts contained within P and all requires clauses of P are provable in D[M]. In this way, partial models describe many problem instances, and solving a model is equivalent to synthesis. The partial model below describes a specific deployment problem, and there are infinite set of models that solve it.

```
 1:  partial model SpecificProblem of Deployments
 2:  {
 3:    requires Deployments.conforms.
 4:
 5:    sVoice is Service("In-car voice recognition").
 6:    sDB    is Service("Dashboard UI").
 7:    n0     is Node(0).
 8:    n1     is Node(1).
 9:    Conflict(sVoice, sDB).
10:  }
```

The assertions in lines 3–9 must hold in a solution. Requires clauses state more complex requirements on solutions, e.g., line 3 requires models to conform to the Deployments domain. The Good model is a manually constructed solution to this partial model. However, with modern constraint solver, such as Z3, it is sometimes tractable to automatically synthesize complete models. Tractability depends on the size of the partial model, structure of constraints, and degrees of freedom that need to be resolved. In the AVM program, such synthesis has been successfully applied to perform aspects of design-space exploration.

Transformations. Transforms are OLPs that transform models between domains. They are useful for formalizing changes in abstractions (such as compilers) and for projecting large integrated models into consistent submodels that can be fed to domain-specific tools. Below is a simple example that compiles Deployment models into configuration files.

```
 1:  transform Compile (in::Deployments)
 2:  returns (out::NodeConfigs)
 3:  {
 4:    out.Config(n.id, list):-
 5:      n is in.Node,
 6:      list = toList (out.#Services, NIL,
 7:                     { s.name | in.Deploy(s, n) }).
 8:  }
 9:  domain NodeConfigs
10:  {
11:    Config ::=
```

```
12:    fun (loc: Natural ->
13:      list: any Services + { NIL }).
14:    Services::=
15:      new (name: String,
16:           tail: any Services + { NIL }).
17:  }
```

Models of the `NodeConfigs` domain contain node configuration files (lines 11–16). Each file lists the services that run on a node. These are modeled using recursive ADTs (i.e., utilizing the full power of tree-like representations). The `Compile` transform takes a `Deployments` model called in and produces a `NodeConfigs` model called out. This is accomplished by the rule in lines 4–7. This rule converts every node into a configuration file containing a list of services. It employs the built-in `toList` function which joins a set of small trees into on large tree (in this case in the form of a list). Notice that models, domains, and transforms all use the same data types and logical rules that describe their behavior. They can all be composed. In fact, the `Compile` transform has a copy of `Deployment` and `NodeConfigs` domains within it, and its rule can refer to data types and have access to derivations performed by rules in these domains.

E. OpenMETA Semantic Backplane

The second component of the semantic integration layer is the OpenMETA semantic backplane. Its primary role is the collection of all model and tool integration related semantic specifications as formal models of all key aspects of the domain-specific integration solution.

As shown in Fig. 7, there are two semantic dimensions that need to be considered in a domain.

1) Epistemic semantics that capture shared conceptualization using ontologies. These ontologies incorporate standards such as QUDT[3] ontology for units of measure, quantity kinds, dimensions, and data types originally developed for the NASA Exploration Initiatives Ontology Models (NExIOM) project, and many domains, even company-specific ontologies defined for improving interoperability of engineering processes.
2) Formal metamodels (models of modeling languages [17]) of domain specific modeling languages (DSMLs) defined for representing integration models such as the model integration language (in the example CyPhyML) and its sublanguages, semantic interfaces to domain models, and modeling languages specifying design flows in testbenches. Formal metamodeling in FORMULA-2 also enables the specification of model transformations in model composers extensively used in the tool integration platform (see Section IV-A).

[3]http://www.qudt.org/

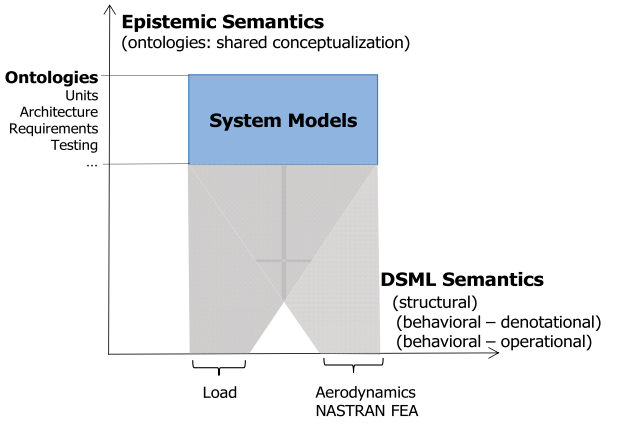

Fig. 7. ***Dimensions of semantics.***

These two dimensions of semantics intersect, such that (ideally) the various DSMLs created for supporting model integration utilize accepted ontologies—standards or locally defined. The OpenMETA project followed this principle as much as it was practical, given the project time frame.

The semantic backplane is a collection of all formal metamodels and specification of model transformations accessible via a web interface to inspect, update, and execute them on the FORMULA-2 tool. The semantic backplane was close to 20 000 lines of FORMULA-2 code, of which about 60% was autogenerated from MetaGME-based graphical metamodels [16]. Examples and more details of the formal metamodels for OpenMETA are available in several papers [14], [42]–[44].

We believe that the introduction of semantic backplane is a new approach to constructing complex component- and model-based design tool chains. It is an essential tool for those who design and evolve domain-specific tool chains and is responsible for the overall integrity of the model and tool configurations used in the design process. Its importance was proven in the following use cases.

1) As in all areas of engineering, mathematical modeling helped designing and evolving modeling languages, composition semantics, and model transformations. It was invaluable in finding and correcting inconsistencies, identifying incompleteness problems, and fixing errors in the semantic foundations of the tool chain.
2) The FORMULA-2-based executable specifications of model transformations were used for generating reference traces and served as abstract prototypes for constraint checkers and production-level transformations used throughout the tool chain.
3) The CyPhyML reference manual was autogenerated from the formal specifications.

In summary, the tool-agnostic model integration layer that incorporates a model integration language (CyPhyML)

(with sublanguages for representing CMs, designs, design spaces, cross-domain interactions, composition constraints, data model interfaces for tools, models of engineering processes, and model transformations for composing analysis models) is complemented by the semantic integration layer including the FORMULA-2-based semantic backplane. Since the model integration language is designed for evolution, defining formal, mathematical semantics of its components is essential for keeping tight control over the semantic integrity of the design process.

IV. TOOL INTEGRATION PLATFORM

The OpenMETA tool integration platform (see Fig. 4) comprises a network of composers implemented as model transformations that compose models for individual tools (e.g., Modelica models from CyPhyML DMs and CMs) invoked by testbenches and deploy model-based design flows on standard execution platforms such as MDAO or HLA. Model transformations are used in the following roles.

1) Packaging: Models are translated into a different syntactic form without changing their semantics. For example, CMs and DMs are translated into standard design data packages for consumption by a variety of design analysis, manufacturability analysis, and repository tools.
2) Composition: Model- and component-based technologies are based on composing different design artifacts (such as DAEs for representing lumped parameter dynamics as Modelica equations [37], input models for verification tools [48]–[54], CAD models of component assemblies [15], DSMs [55] [61], and many others) from appropriate models of components and component architectures.
3) Virtual prototyping: Several test and verification methods [such as probabilistic certificate of correctness (PCC)] require testbenches that embed a virtual prototype of the designed system executing a mission scenario in some environment (as defined in the requirement documents). We found distributed, multimodel simulation platforms to be the most scalable solution for these tests. We selected the HLA as the distributed simulation platform and integrated FMI cosimulation components with HLA [19].
4. Analysis flow: Parametric explorations of designs (PETs), such as analyzing effects of structural parameters (e.g., length of vehicle) on vehicle performance, or deriving PCC for performance properties, frequently require complex analysis flows that include a number of intermediate stages. Automating design space explorations require that Python files controlling the execution of these flows on the OpenMDAO platform (that we currently use in OpenMETA) be autogenerated from the testbench and parametric exploration models (Fig. 4).

A. Model Composers

The model composers are the bridge between the model, tool, and execution integration platforms by 1) composing testbench (and incorporated tool) specific analysis models from designs models $S_D = \langle C_{SI}, G_S \rangle, S_D \in D$ that are included in the D design space using the information in the CMs $C(x, p)$; 2) integrate those with the testbench models to obtain executable specification; and 3) map the fully configured testbench models on the execution integration platform [55].

Model composers work on a specific view of models and implement a logic that relies on the structural semantics of a domain or domains associated with the specific view. The model composers extract relevant portions of the model from a selected view. Once the specific view is selected, a model transformation is performed to another model representation, while the semantic specifications are preserved. The transformed models are executable or analyzable by simulation, static analysis, or verification tools.

There are several approaches to implement model composers:

1) using manually implemented and maintained source code in a programming language;
2) using a combination of manual implementation of a source code in conjunction with automatically generated application programming interfaces (APIs) for the structural semantics of each model representation;
3) using model-based rewrite rules, where no or minimal logic is captured by a manually implemented source code [16];
4) using a formal specification of the model composer (e.g., FORMULA).

All implementation approaches starting from 2) have progressively stronger linkage and relationship to the semantic backplane. This provides significant benefit on verification of ensuring semantic consistency across the multiple representations and guarantees that all concepts are kept consistent. However, our experience shows that as we move toward more formal specification of model composers, the runtime of model composers gets larger. Model composers observe scalability issues, when dealing with large-scale complex CPS models along with rich domain-specific languages containing thousands of concepts.

These model composers also work on design spaces and on the instantiated testbenches from the testbench template library. This capability provides a high degree of automation, when a design space is elaborated to hundreds or thousands of configurations and the model composers

extract the configuration and view-specific content and perform the model transformation to simulation, analysis, or verification tools.

B. Example Model Composer for PCC Testbench

As an example for a model composer, we discuss the PCC testbench and its PCC model composer, which generates a PCC experiment setup for the execution integration platform. Each testbench is applicable for design spaces, thus the PCC experiment is applicable over a set of configurations generated by the design space exploration tool. The purpose of these experiments is to find the most robust design with respect to changes and uncertainties in the input parameters to the workflow. A distribution function is fitted on each output. Each output has three parameters defined: a minimum value, a maximum value, and a target percentage. The area under the output distribution function within the defined minimum and maximum values gives the PCC value, a number between 0% and 100%. If this PCC value is higher than the defined target PCC value in the model, it means that the output parameter falls within our required limit. Often this target PCC value is directly linked to a system or subsystem requirement. For each design, a joint PCC value is derived from all outputs.

The PCC testbench model contains a definition of a workflow of analysis tools and a PCC driver module model as depicted in Fig. 8. The workflow model elements refer to other testbenches, which use the execution integration platform to generate analysis results. A dependency between the different testbenches, most likely spanning vastly different domains, is contained in the PCC testbench model. The PCC driver model defines a set of input parameters for the workflow and their probability distribution functions (pdfs) and a set of instrumentation points, which variables are recorded throughout the multiple execution steps. In addition, the PCC driver model defines a sampling method for the input parametric space, which determines the number of iterations for this experiment.

A PCC model composer was implemented to map the PCC testbenches to the execution integration platform. Fig. 9 depicts the logic of the PCC model composer. Each model composer is reusable, functionally complete, and composable. For example, model composer 1 is reused "as is" for

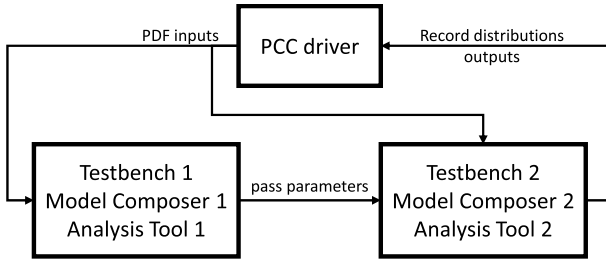

Fig. 8. ***PCC testbench model.***

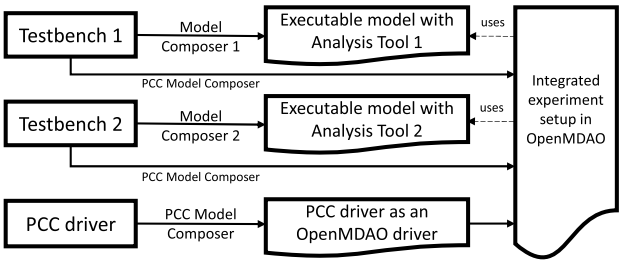

Fig. 9. ***Example for a model composer (PCC).***

generating executable models for analysis tool 1. We utilize the capabilities of OpenMDAO to perform the execution. The PCC model composer invokes all required model composers and generates additional OpenMDAO component wrappers for each testbench type around the executable models. In addition to the OpenMDAO components, the PCC driver model is transformed to an OpenMDAO driver. The driver module is a parametric OpenMDAO driver developed by Oregon State University [53]. Finally, the integrated experiment is set up by instantiating all components including the driver and the workflow specification according to the PCC testbench model.

In order to effectively manage the large demand of simulations and analyses, a job manager and a dispatch server were created in the execution integration platform. The detailed description of the job manager is outside the scope of this paper.

As the example illustrates, a significant part of the overall integration complexity, and complex transformation logic, is concentrated on the model composers. This underlines the significance of explicit modeling of the semantics of model composers using tools of the semantic backplane.

C. Static Design-Space Exploration Testbench

Arguably, the ability to construct, shape, and explore design spaces is critical in the design of CPSs, which have to fulfill a large number of—often conflicting—performance objectives, as well as satisfy a variety of implicit and explicit constraints. The design space representation in CyPhyML, described earlier, using hierarchically layered assembly and component alternatives and parameterization is quite powerful in its ability to compactly represent extremely large design spaces. However, the size and dimensionality of the design space poses a significant challenge, rendering it infeasible to exhaustively enumerate and evaluate all candidate designs encapsulated in the design space representation.

OpenMETA design flow, as we articulated earlier, employs a progressive deepening strategy, i.e., evaluates the candidate designs using lower fidelity models and progresses toward increasingly complex, higher fidelity models and focuses on the rapidly decreasing number of candidate designs. However, even with this pragmatic strategy, a major challenge was that the initial combinatorial design

space (sometimes with billions of candidate designs) is simply infeasible even to enumerate, let alone evaluate, irrespective of the model fidelity. Therefore, we had to employ powerful analytic techniques that could manipulate entire design spaces without exhaustively enumerating them. Inspired by the success in model checking [56], our prior work on design space exploration in hardware/software codesign [57], [58] resulted in development of DESERT [59]–[61], a tool for constraint-guided pruning of design spaces using symbolic methods.

The static design space exploration testbench utilizes and integrates DESERT into the OpenMETA design flow. The static design space exploration testbench is uniquely distinct (compared to other testbenches in OpenMETA), in that it transforms an entire design space (rather than design configurations) into a symbolic representation suitable for DESERT, and then inverse transforms the (severely) trimmed design space from DESERT.

We refer to this as "static design space exploration," since DESERT solves constraints over static (time-invariant) properties of design elements (e.g., MaxTorque, or MaxRPM of an engine) and their analytic composition over architecture variants. It is often tempting to trivialize the type of low-fidelity analytics employed in static DSE and their role in understanding complex CPS designs. However, we learned that several common classes of design errors ("sizing mismatch," "part compatibility," "manufacturability") that require expensive redesign and design iterations in industrial design processes can be eliminated by encoding as constraints over static properties to navigate away from the erroneous architecture variants. For example, a "gradeability" requirement (e.g., vehicle climbing a 20° hill can accelerate at 2 m/s^2) translates to an engine sizing problem and can be represented as the following constraint:

```
(Powerplant_maxTorque() * 1.3558) >=
((Tire_radius()/1000) *(Vehicle_weight() +
13000) *sin(0.35) + ((Vehicle_weight() +
13000)/9.8) * 2).
```

Each of the four named properties referred to in the constraint above is a "variable" and is nontrivially dependent upon the component selections. Satisfying this constraint in static DSE enforces that all "pruned-in" designs are conformant to the requirement, and will not result in "sizing mismatch" type of design errors.

DESERT uses an AND–OR–LEAF tree structure to represent design spaces in a generic manner, where AND nodes represent hierarchically inclusive containers, OR nodes represent exclusive or choice containers, while LEAF nodes represent design primitives. DESERT also allows properties associated with LEAF nodes that can have a unique value or a range of value assignment. DESERT supports a number of different types of constraints, specified using an extended object constraint language (OCL) [62]. Functions supported in constraints include all logical, trigonometric, and logarithm functions as well as *sign*, *rint*, *sqrt*, and *abs*. DESERT's internal operation and user interface is outside the scope of this paper, however, it is included in [61].

The static design space exploration testbench maps CyPhyML design spaces to DESERT's AND–OR–LEAF representation with the following mapping: component assemblies are mapped to AND nodes, alternative containers are mapped to OR nodes, and components are mapped to LEAF nodes. Constraints, as exemplified above, are represented in CyPhyML using multiple graphical and textual notations amenable to end users. In the mapping process, these constraints are translated into the extended OCL notation supported by DESERT. In CyPhyML, intercomponent and intracomponent property relations are represented with "value flows," while in translation to DESERT, these "value flows" are mapped into constraints.

DESERT accepts as input the design space representation as an AND–OR–LEAF tree, a set of constraints, and generates an enumeration of satisfying configurations. These configurations represent a binding of choice elements (OR nodes) to the specific selection (either an AND node or a LEAF node). In the inverse transformation, these configurations are elaborated back into hierarchical component assemblies in CyPhyML's native representation, where they become suitable for subsequent evaluation by other testbenches.

Design space is a foundational construct in OpenMETA, and our experience indicated the need to robustly support the construct throughout the design process with additional tools. For example, designers in OpenMETA after an iteration through the design flow often wanted to understand the critical component selections, i.e., components that occur in most viable design configurations. We also observed that the design flow is not linear even in the sense that the design space is not fully defined (depth wise or breadth wise) all at once in the beginning, but rather follows an iterative path that can best be articulated as alternately narrowing (by exploration and pruning), deepening (by further elaborating and adding details in subset of component hierarchies), and widening (by adding additional component selections and parameterization in subtrees). These observations led us to improve support for design space management by creating several helper tools: the design space refiner, the design space refactorer, and the design space criticality meter [61]. Finally, DESERT as a symbolic constraint satisfaction tool represents a powerful capability, however, it is constrained by the underlying BDD and MT-BDD backend and limited in terms of algebraic domains supported. The recent development in SMT solvers, particularly with an expansive support library of domain theories, offers a unique opportunity to augment DESERT with an SMT-solver-based backend.

D. Verification Testbenches

Verification is a key enabler to the development of advanced CPSs by improving system assurance with

compressed qualification times [63]. Verification aims at analyzing functional and behavioral correctness in the presence of multiple sources of nondeterminism and has a crucial role in model-based design. The ultimate objective is to facilitate the "correct-by-construction" design of systems by reducing the need for expensive and unavoidably incomplete experimental prototyping and testing. Verification methods involve analysis and reasoning using models of the system in each step of the design to ensure that it satisfies the requirements. Such methods are gaining increased attention because they allow the decomposition of the design process into an iterative progression from requirement models to implementation models using the repeated step of model construction, verification, and transformation [2], [3]. In this process, verification techniques can focus on the semantics of a suite of abstract system models that are incrementally refined into implementation models.

Verification of CPSs is a very hard problem due to heterogeneous, highly nonlinear, and nondeterministic dynamics, scalability due to a large number of components, and complexity of the formal framework needed to express and reason about requirements. These challenges are addressed typically by considering simplified abstractions and/or approximations of system models. A fundamental gap is the lack of understanding of the interrelations among different abstractions as well as the lack of methods for the automated composition of multiabstraction and multifidelity models that can be adapted to the property to be verified. A significant objective in our work is the development of methods and tools for automated generation of suitable abstractions from detailed systems models that enable the use of formal verification. An additional objective is the integration of verification methods with simulation that is achieved through model integration and the CyPhyML modeling integration language.

Formal verification methods are integrated in OpenMETA using verification testbenches. The verification testbenches include 1) system models in some mathematical domain (e.g., DAEs [29], qualitative automata [49], and hybrid systems) [48]; 2) requirements expressed as model properties in some logic framework [64]; 3) information about ranges of inputs, model parameters, context, and initial conditions; and 4) algorithms proving that the models satisfy the properties over the defined domains or generating counterexamples demonstrating that the properties are violated.

The main modeling formalism used in the lumped parameter dynamics abstraction level in OpenMETA is hybrid systems [50]. A significant challenge is the automated translation from composed simulation models represented in the Modelica language to mathematical models that can be used for verification and formal analysis. In OpenMETA, this problem of automated translation is decomposed to multiple subproblems that are addressed by different tools.

1) The first step is the construction of a CM library consisting only of declarative models that can be represented with mathematical equations. We developed a fully equation-based, symbolic version of the FANG Modelica component libraries that the formal verification tools are able to process.
2) The second step is the use of tools within the Modelica compiler for generating the mathematical equations of complex system models with interconnections of multiple components. These models are used as inputs to model translators that generate formal models suitable for various verification techniques. Although the coverage of automated translation is not complete, the models and tools developed demonstrate the feasibility of the approach for a large and important class of system dynamics.

The use of verification methods in the design flow starts with the construction of a system model that is adapted to the property that needs to be verified. In the context of OpenMETA, examples of such properties express vehicle requirements that can be associated with mobility operational range, payload, braking, and other performance requirements. In addition to performance, safety requirements comprise another group of properties to be verified. For example, a typical requirement for military vehicles is that the vehicle will meet all performance targets for all load conditions without exceeding component manufacturers' limits.

Such safety properties are associated with subsystems and components and need to be derived during the system design. Verification methods require encoding such properties in a formal logic framework. In OpenMETA, linear temporal logic (LTL) is used to express performance and safety properties. LTL formulas use combinations of temporal and logic operators to express complex properties that must be satisfied by the system behavior [50]. Writing the informal system requirements as formal temporal logic formulas is a challenging task especially for nonexperts. The project developed a template-based tool that allows users to write properties using an intuitive interface, which then are translated to LTL formulas that are used by the verification tools. Reliability requirements comprise another class of system requirements that need to be addressed. When a system is deployed, faults occur in the field due to environmental conditions, poor maintenance, and attacks on equipment. Therefore, it is crucial to reason about fault behavior and failure impact as part of the early system correctness analysis and verification efforts, and determine the survivability and continued utility of equipment in the presence of faults [51], [52].

The tool chain includes three verification methods for complementing simulation-based analysis: 1) qualitative reasoning; 2) verification based on relational abstractions; and 3) reliability analysis using fault-augmented models. These methods provide different but synergistic capabilities, demonstrate how verification methods can be integrated in

the design tool chain, and how they can be used for reducing the need for prototyping and testing.

Qualitative reasoning (QR) aims to automate verification of continuous dynamics by generating qualitative descriptions of physical systems and their behaviors [49]. QR can identify feasible designs, suggest parameter changes that improve design quality, and highlight potential problems due to poor parameter choice relying on qualitative abstractions of the system dynamics. In OpenMETA, Modelica hybrid system models are abstracted into constraint networks that are used to generate a multitrajectory simulation, or envisionment, of all possible qualitative behaviors of the abstraction. This envisionment enables qualitative verification where behavioral/safety requirements can be proved to be satisfied for all, some, or no choice of the parameters in the underlying Modelica model. Qualitative analysis can be performed early in the design process before all parameters have been selected. In the case the model may not meet the requirements for a specific choice of design parameters, QR provides guidance about how to modify the model parameters. We were able to analyze models with more than 100 equations and integrate the analysis into the tool chain to evaluate various system requirements. The main limitation is scaling up to more complex models consisting of thousands of equations that are typical of complete vehicle system models.

The tool suite uses the HybridSal relational abstraction as a formal verification tool for verifying safety properties of CPSs [50]. The main objective is verifying control algorithms while taking into account the behavior of the physical plant. The controller is modeled using Matlab, Simulink, and Stateflow, while the physical plant is modeled in Modelica using components from the declarative component library. Physical plants are represented using DAEs. The desired behavior of the controller is encoded in LTL formulas focusing on safety properties. Integration is based on 1) a slicer which aims at identifying a small subset of the set of DAEs of the Modelica model that correctly describe the dynamics of a given set of variables which are selected as the input variables of the controller models; 2) a model translator from Modelica to the HybridSal verification tool; 3) a tool that takes as input a Matlab controller model and outputs the same controller in HybridSal's input language; and 4) the HybridSal relational abstraction-based tool for verifying composed controller and plant models. Examples of verification testbenches include the following:

1) a drivetrain model with transmission controller in order to determine the range of grades of the road where gears provably do not chatter for all driver throttle positions;
2) a model of an engine, which takes as input the torque value and outputs engine RPM, in order to determine the range of inputs that guarantee that the engine RPM is bounded. The main limitations are related to the coverage of the slicer and model translator of the physical plant.

The goal of the reliability analysis testbenches is to evaluate a vehicle design using various reliability metrics. The testbenches use fault-augmented CMs that are developed to simulate the effects of component faults and their propagation to system-level behavior [51]. The starting point of the reliability analysis is a design with fault-augmented CMs [52]. Components are associated with applicable damage parameter maps that provide the probability distributions of damage parameters (e.g., wear fraction for friction wear in a clutch) with respect to usage as well as other component-specific parameters (e.g., geometric constant and maximum normal force for a clutch). The design is then analyzed and the damage-parameter maps are interpolated to generate time/usage-dependent probabilistic transitions from the nominal to faulty state for the system components. Subsequently, the design is run through a suite of reliability testbenches. These testbenches evaluate the design against several key reliability requirements as they affect system-level performance after extended usage.

In order to complement formal verification, a suite of methods for performing probabilistic analysis for a given performance requirement are also developed and integrated. While deterministic methods seek to offer a yes/no answer to verification questions, probabilistic methods provide a PCC using methods of uncertainty quantification (UQ). The goal is to determine the probability that the performance function is less than (or greater than) the requirement. PCC methods extend simulation methods by assuming that inputs, outputs, and model parameters are uncertain but can be specified using probability distributions. In general, they are more scalable than formal verification methods but they require additional information in order to quantify the uncertainty in the model. PCC testbenches use Modelica system models extended to capture uncertainty in the system inputs and parameters. As part of our project, we implemented six UQ methods for PCC estimation: Monte Carlo simulation (MCS), Taylor series method (TSM), most probable point (MPP), full factorial numerical integration (FFNI), univariate dimension reduction (UDR), and polynomial chaos expansion (PCE) [53]. In addition, we implemented sensitivity analysis methods to quantify the amount of variance that each input factor contributes to the PCC of the performance requirement (Sobol, FAST, EFAST [54]).

In conclusion, we developed and integrated in the OpenMETA toolchain two formal verification methods, one reliability analysis method, and methods for UQ. The developed tools have been demonstrated using various testbenches representing realistic vehicle models. Although the model complexity, heterogeneity, and scale create considerable challenges, verification methods provide complementary capabilities for addressing performance, safety, and reliability requirements at different phases of the design flow. These capabilities facilitate the "correct-by-construction" design and can be used to complement simulation methods.

V. LESSONS LEARNED

Creating end-to-end design automation tool chains for the model- and component-based design of CPSs is a significant challenge that extends to the fundamentals of compositionality in heterogeneous domains, formulating abstractions with established relationships, modularization of models to create reusable CM libraries, verification methods, and scaling to real-life systems. The OpenMETA project that was part of DARPA's AVM program offered unique opportunity for conducting a large-scale integration experiment with extensive testing of the results using a design competition [8]. In the following, we summarize some of the most important lessons learned.

A. Horizontal Integration Platforms

Approaching the design automation challenge by establishing horizontal model, tool, and execution integration platforms was a necessity. The key insight for us was that the tradeoffs among using standard-based, or tool/vendor-specific, or locally defined ontologies and DSMLs as model integration languages is an interesting problem of its own and the answer depends on the context: characteristics of the CPS product line, size of the company, and complexity of the engineering process. However, independently from the context, we did not conclude that the ultimate solution would be the emergence of a standard, universal model integration language (e.g., SysML,[4] or other), simply because these languages need to respond rapidly to changes in the product line and in the engineering processes, and the cost of evolving domain-specific modeling languages is not a prohibiting factor due to the appearance of metaprogrammable modeling and model management tools such as WebGME,[5] EMF,[6] and others. We found that the use of model and tool integration platforms can provide an increased level of decoupling between the product and engineering-process-specific view of the systems' companies and the more generic view of tool vendors.

B. Availability of Reusable Component Model Libraries

In model- and component-based design, the key productivity factors depend on the availability of reusable, componentized model libraries on different domains and on the feasibility of fully automated model composition during design space exploration. Both of these factors were much harder to obtain than we expected. There are excellent examples for existing, highly successful model libraries, both in crowdsourced or COTS form: DOE's EnergyPlus[7] is an open-source model and simulation library for energy; the Modelica Standard Library (MSL)[8] is a crowdsourced, multiphysics lumped parameter dynamics library developed and maintained by the OpenModelica Consortium; Modelon's Vehicle Dynamics Library[9] is a COTS component library on the top of the Modelica Standard Library; and many others. We believe that domain-specific model libraries will continue emerging both in open-source and COTS form and will become one of the engines in the progress of component and model-based design. We believe that our CCMs should accelerate progress in creation of multidomain CM libraries.

C. Automated Model Composition

This is frequently missing in physical domains due to the perception that useful physical models need to be handcrafted for specific phenomena, and consequently, the modeling abstractions used are often decoupled from the physical architecture of the system. One explanation for this is the frequent use of modeling approaches that do not support generic compositionality. Our CCM places strong emphasis on compositional semantics to resolve this problem. However, we have identified several challenges, ranging from technical to fundamental that had to be addressed or still open according to our knowledge.

Target language challenge. While CyPhyML incorporates an extensive set of checks on the well-formedness of the composed architecture models (as defined in the structural semantics of CyPhyML), when the model composers generate the individual analysis models, there are additional considerations that are specific to the target language and do not belong to the CyPhyML structural semantics. For example, the composed Modelica models need to satisfy a range of best practices and conventions that need to be guaranteed by component authoring guidelines and curation checks. These guidelines include rules such as using potential rather than flow as boundary condition in physical models, preference of using declarative constructs instead of imperative, and many others. Even with these guidelines, extensive testing and debugging of CMs in various system contexts were required to achieve acceptable composability.

Component validity challenge. Models of physical components have (occasionally implicitly) an operating regime where they are valid, i.e., reproduce accurately the behavior of the referent (component or subsystem). In lumped parameter dynamic models, this region of validity is usually expressed as constraints over the state variables. The challenge to composability is that whether CMs remain valid during simulation runs depends on their interaction with other components. It is possible that a computed trajectory becomes invalid simply because one or more components temporarily leave their region of validity, or in other words, they lose

[4]http://www.omgsysml.org/
[5]http://webgme.org
[6]http://www.eclipse.org/modeling/emf/docs/
[7]http://apps1.eere.energy.gov/buildings/energyplus/energyplus_addons.cfm
[8]http://www.modelica.org
[9]http://www.modelon.com/products/modelica-libraries/vehicle-dynamics-library/

their composability. We addressed this problem by making the regions of validity explicit, i.e., part of the CMs, and by actively monitoring validity violations during these simulation runs.

Complexity challenge. A particularly hard problem of automated composition of system models from multiphenomenon CMs is that it can easily produce very complex, high-order models that the solvers cannot handle. We experimented with supporting multiple abstractions, and multiple fidelity levels in component libraries, and with adapting the selected level of abstraction and component fidelity level to the system property being analyzed. While this approach has proved to be promising, the cost of creating multiple fidelity models has remained an open challenge. Deriving surrogate models combined with machine learning [67], [68] was a possible solution but validation of surrogate models was still costly, and composability of surrogate models is not well understood.

Time resolution challenge. Complex, multiphysics models frequently incorporate dynamics with highly different frequency ranges. For example, in drivetrains, the time constant of mechanical processes is significantly smaller than that of thermal processes. Composition of the system level Modelica model for a drivetrain yields a large number of equations for which the simulation with a single Modelica simulator may be extremely slow because the step size of the solver is determined by the fastest processes. We experimented with phenomenon-based slicing of multi physics models by first composing the system-level model using CMs, followed by the decomposition of the model, but not along the component/subsystem boundaries, rather along physical phenomena (mechanical processes and thermal process) so that we can separate the fast and slow dynamics. This decomposition led to two models that can be cosimulated using the cosimulation testbench, so the recomposition of the system-level simulation occurs in a different semantic domain (distributed cosimulation). We have demonstrated the feasibility and impact of the approach in [66]. However, the construction of model libraries that can be automatically sliced by physical phenomena is an open problem.

D. Semantic Backplane

While our experience with constructing a formal semantic model for key integration entities (model integration language, ontologies, semantic interfaces, model composers) has proved to be extremely valuable to keep the overall integration effort consistent, the fact that the "production tools" in OpenMETA [the metaprogrammable generic modeling environment (GME), its metamodels, and the code of the model composers] were separate from the FORMULA-2-based formal specifications created the risk of divergence between the implemented integration components and their formal models. To mitigate this risk, a tight coupling needs to exist between the metaprogrammable tools and FORMULA-2, which is an active research effort.

E. Verification

In spite of significant progress, verification of complex CPSs remains a very hard problem and there are major challenges that hinder the use of verification methods: 1) automated translation of complex component-based models to suitable abstractions; 2) mapping of significant correctness requirements to typical safety/reachability properties; and 3) integration of effective and usable verification methods into the design flow. The fundamental problems for addressing such challenges can be categorized along the following four dimensions: 1) complexity due to highly nonlinear and nondeterministic dynamics; 2) heterogeneity of components and behaviors; 3) scalability due to large number of components; and 4) epistemic uncertainty due to neglecting certain effects in the models because of knowledge gaps. Furthermore, these challenges are coupled in modern systems, and this coupling magnifies the problem. Another fundamental gap is the lack of understanding of the interrelations among different abstractions and the lack of methods for the automated composition of multiabstraction and multifidelity models that are adapted to the property to be verified.

F. Code Complexity

The dominant challenge in developing OpenMETA was integration: models, tools, and executions. The OpenMETA integration platforms included over 1 million lines of a code that is reusable in many CPS design contexts. In the AVM project, OpenMETA integrated 29 open-source and eight commercial tools, representing a code base estimated to be two orders of magnitude larger than OpenMETA [6] itself. The conclusion is that integration does matter. It is scientifically challenging and yields major benefits. This is particularly true in design automation for CPSs, where integrated design flows are still not a reality.

G. Evaluation and Transitioning

The OpenMETA tool suite evolved along parallel evaluation efforts performed first in the FANG 1 design competition focusing on the drivetrain and mobility aspects of the vehicle and later on the hull design. A comprehensive summary of the FANG 1 competition and related evaluation is available in [8]. While FANG 1 was relatively early in the project (after two years of effort), we believe that it has proved early the feasibility and validity of the overall integration architecture we discussed in this paper. The competition experience also provided our team with useful insights and feedback regarding the maturity of the dominantly open-source simulation and verification component technologies, and in a more general sense, advantages and limitations of the model- and component-based design approach for CPSs (the most essential ones described above).

The OpenMETA transitioning process has started in 2014 through various transitioning programs,[10] partially by spinoff companies pursuing a wide range of domains including the integration of tool chain pilots for electronic design, robotics, aerospace systems, and automotive systems. Since OpenMETA addressed integration technologies that were motivated by the needs of CPS design, their applicability remains broad. Most of the transitioning efforts utilize two open-source repositories in Github: one for OpenMETA-core[11] and one for OpenMETA-extensions.[12]

VI. CONCLUSION AND FUTURE DIRECTIONS

The OpenMETA integration platforms addressed the following problems of component- and model-based design for CPS: 1) composing system-level models from reusable, heterogeneous CM libraries; 2) extending the limits of the correct-by-construction design by supporting heterogeneity and automation; 3) applying multiple levels of abstractions in design flows for CPSs; 4) executing rapid design tradeoffs; 5) defining an interface between the design and manufacturing for CPSs; and 6) creating an open framework for reusing open-source tool assets. The project gave the OpenMETA developers unique opportunity not only to understand the limits of the current state of the art in the context of a real-life DoD challenge problem but also to push the limits in several areas. We believe that the program also provided opportunity for the developers and the research community in general to better understand open problems and their impact on the broad applicability of model-based design technologies. Based on this experience, in the following, we summarize some of the open challenges and opportunities.

1) Product and manufacturing process codesign: Merging the relatively isolated activities in product and manufacturing process design into an integrated codesign process promises the largest benefits and truly revolutionary advantages. This would be particularly important with the increased use of composites in manufacturing, in which the interdependence of product models and manufacturing process models is large and not understood well. While the OpenMETA design space exploration process incorporated feedback from manufacturability analysis, full integration was not achieved yet.
2) Configurable domain-specific design environments: The OpenMETA horizontal integration platforms emerged as key enablers for the tool chain development effort. The primary end users of OpenMETA were the vehicle designers. Consequently, the implemented automations and user interfaces served the designer community. However, the development of the model, tool, and execution integration platforms—the core contributions of OpenMETA—created opportunity for automation and improved user interfaces for another category of users, those whose goal is to integrate domain-specific CPS design tool chains. To achieve progress in this area is one of our goals.
3) Integrating model-based and data-driven design: A fundamental limitation of model-based design processes is that models of physical components and environments always have epistemic uncertainties. Epistemic uncertainties originate in the lack of knowledge or data. In complex CPSs, their weight and potential risk are significant because they decrease predictive properties of the DMs and limit the confidence level of the assurance arguments. Decreasing this uncertainty is expensive and, with the rapidly growing size and openness of important categories of CPSs, it is becoming unfeasible. A promising approach of addressing this challenge is the incorporation of data-driven machine learning methods in the design process that will require fundamental rethinking of its key elements from modeling to verification and to assurance argumentation. ■

Acknowledgment

The authors would like to thank P. Eremenko for his vision and leadership and who conceived and led the program through the crucial first two years. They also would like to express their appreciation to Dr. N. Wiedenman for steering the program through the FANG-1 design challenge and Dr. K. Massey for guiding the program to its conclusion. The OpenMETA project integrated technology components provided by researchers from Vanderbilt University, Georgia Tech, MIT, Oregon State University, SRI, PARC, and Modelon. The authors are also grateful for the significant contributions of A. Nagel, Metamorph Inc., to the transitioning efforts and latest extensions of OpenMETA. The project benefited tremendously from the discussions, feedback, and ideas the authors received from their team of collaborators, and from Dr. M. Lowry, Chief Engineer, and Dr. K. Bellman, Chief Scientist of the AVM program. The authors would also like to recognize the importance of the guidance from the Senior Strategy Panel including A. Sangiovanni Vincentelli (University of California Berkeley), Prof. J. Sifakis (VERIMAG), D. Frey (MIT), K. Hedrick (University of California Berkeley), O. De Weck (MIT), and J. Sztipanovits (Vanderbilt University).

[10]https://www.pddnet.com/news/2014/02/darpa-begins-early-transition-adaptive-vehicle-make-technologies
[11]https://github.com/metamorph-inc/meta-core
[12]https://github.com/metamorph-inc/openmeta-mms

REFERENCES

[1] S. A. Seshia, "Combining induction, deduction, and structure for verification and synthesis," *Proc. IEEE*, vol. 103, no. 11, pp. 2036–2051, Nov. 2015.

[2] S. A. Seshia, S. Hu, W. Li, and Q. Zhu, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1421–1434, Sep. 2017.

[3] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proc. IEEE*, vol. 103, no. 11, pp. 2104–2132, Nov. 2015.

[4] J. Sifakis, "System design automation: Challenges and limitations," *Proc. IEEE*, vol. 103, no. 11, pp. 2093–2103, Nov. 2015.

[5] E. Paul, "Philosophical underpinnings of adaptive vehicle make," Tech. Rep. DARPA-BAA-12-15, Dec. 2011, vol. 1.

[6] J. Sztipanovits, T. Bapty, S. Neema, X. Koutsoukos, and E. Jackson, "Design tool chain for cyber-physical systems: Lessons learned," in *Proc. 52nd ACM/EDAC/IEEE DAC*, San Francisco, CA, USA, vol. 15, Jun. 2015, pp. 1–6.

[7] A. Spencer. (2013). "This is the million-dollar design for DARPA's crowdsourced swimming tank." [Online]. Available: https://www.wired.com/2013/04/darpa-fang-winner

[8] O. L. de Weck and E. S. Suh, "Complex system design through crowdsourcing: DARPA FANG–1 as a case study," *Ind. Eng. Mag.*, vol. 21, no. 4, pp. 32–37, 2014. [Online]. Available: http://www.dbpia.co.kr/Article/NODE06085424

[9] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless model-based development: From isolated tools to integrated model engineering environments," *Proc. IEEE*, vol. 98, no. 4, pp. 526–545, Apr. 2010.

[10] J. D'Ambrosio, T. Bapty, E. Jackson, J. Paunicka, and J. Sztipanovits, "*Comprehensive transitioning model for AVM tools*," Boeing, GM, Microsoft Research, MetaMorph Inc., Vanderbilt, White Paper DARPA-SN-14-04, Oct. 2013.

[11] A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone, and M. Sgroi, "System level design paradigms: Platform-based design and communication synthesis," *ACM Trans. Design Autom. Electron. Syst.*, vol. 11, no. 3, pp. 537–563, Jun. 2004.

[12] E. A. Lee, "CPS foundations," in *Proc. 47th Design Autom. Conf. (DAC)*, Anaheim, CA, USA, Jul. 2010, pp. 737–742.

[13] Z. Lattmann *et al.*, "Towards automated evaluation of vehicle dynamics in system-level designs," in *Proc. ASME Int. Design Eng. Techn. Conf. Comput. Inf. Eng. Conf. IDETC/CIE*, Chicago, IL, USA, Aug. 2012.

[14] G. Simko *et al.*, "Foundation for model integration: Semantic backplane," in *Proc. ASME Int. Design Eng. Techn. Conf. Comput. Inf. Eng. Conf. IDETC/CIE*, Chicago, IL, USA, Aug. 2012.

[15] R. Wrenn *et al.*, "Towards automated exploration and assembly of vehicle design models," in *Proc. ASME Int. Design Eng. Techn. Conf. Comput. Inf. Eng. Conf. IDETC/CIE*, Chicago, IL, USA, Aug. 2012.

[16] G. Karsai, A. Ledeczi, S. Neema, and J. Sztipanovits, "The model-integrated computing toolsuite: Metaprogrammable tools for embedded control system design," in *Proc. Comput. Aided Control Syst. Design Int. Conf. Control Appl., Int. Symp. Intell. Control*, Oct. 2006, pp. 50–55.

[17] A. Sangiovanni-Vincentelli, S. K. Shukla, J. Sztipanovits, G. Yang, and D. A. Mathaikutty, "Metamodeling: An emerging representation paradigm for system-level design," *IEEE Design Test Comput.*, vol. 26, no. 3, pp. 54–69, May/Jun. 2009.

[18] [Online]. Available: http://openmdao.org

[19] *Standard for Modeling and Simulation High Level Architecture–Federate Interface Specification*, Standard IEEE 1516.1-2010.

[20] *Standard for Modeling and Simulation High Level Architecture-Framework and Rules*, Standard IEEE 1516-2010, 2010, pp. 1–38.

[21] L. Juracz *et al.*, "VehicleFORGE: A cloud-based infrastructure for collaborative model-based design," in *Proc. 2nd Int. Workshop Model-Driven Eng. High Perform. Cloud Comput. (MDHPCL), 16th Int. Conf. Model Driven Engineering Languages and Systems (MODELS)*, vol. 1118, 2014. [Online]. Available: http://www.isis.vanderbilt.edu/sites/default/files/MDHPCL%202013%2004-paper.pdf

[22] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? Reasoning about the trends and challenges of system level design," *Proc. IEEE*, vol. 95, no. 3, pp. 467–506, Mar. 2007.

[23] P. Nuzzo *et al.*, "A contract-based methodology for aircraft electric power system design," *IEEE Access*, vol. 2, pp. 1–25, Dec. 2013.

[24] E. Jackson and J. Sztipanovits, "Formalizing the structural semantics of domain-specific modeling languages," *J. Softw. Syst. Model.*, vol. 8, no. 4, pp. 451–478, Sep. 2009.

[25] K. Chen, J. Sztipanovits, and S. Neema, "Compositional specification of behavioral semantics," in *Design, Automation, and Test in Europe: The Most Influential Papers of 10 Years DATE*, R. Lauwereins and J. Madsen, Eds. New York, NY, USA: Springer-Verlag, 2008.

[26] E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Reasoning about metamodeling with formal specifications and automatic proofs," in *Model Driven Engineering Languages and Systems*, vol. 6981, J. Whittle, T. Clark, and T. Kühne, Eds. Berlin, Germany: Springer-Verlag, 2011, pp. 653–667.

[27] [Online]. Available: http://research.microsoft.com/formula

[28] M. R. Myers, R. E. Gregory, J. E. Hite, S. Neema, and T. Bapty, "Collaborative design in DARPA's FANG I challenge," in *Proc. ASME Int. Mech. Eng. Congr. Expo., Syst. Design*, San Diego, CA, USA, vol. 12, Nov. 2013, pp. 15–21.

[29] P. Adrian, S. Martin, A. Adeel, F. Peter, and C. Francesco, "Integrated debugging of Modelica models," *Model. Identif. Control*, vol. 35, no. 2, pp. 93–107, 2014.

[30] [Online]. Available: https://openmodelica.org

[31] E. K. Jackson, G. Simko, and J. Sztipanovits, "Diversely enumerating system-level architectures," in *Proc. EMSOFT*, Montreal, QC, Canada, Sep./Oct. 2013.

[32] J. Eker *et al.*, "Taming heterogeneity—The Ptolemy approach," *Proc. IEEE*, vol. 91, no. 1, pp. 127–144, Jan. 2003.

[33] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems—A Cyber-Physical Systems Approach*, 2nd ed. 2015. [Online]. Available: http://leeseshia.org

[34] Simulink. [Online]. Available: http://www.mathworks.com/products/simulink

[35] J. Porter *et al.*, "The ESMoL language and tools for high-confidence distributed control systems design. Part 1: Design language, modeling framework, and analysis," Vanderbilt Univ., Nashville, TN, USA, Tech. Rep. ISIS-10-109, 2010.

[36] Functional Mock-Up Interface. [Online]. Available: http://www.fmi-standard.org

[37] (Mar. 24, 2010). *Modelica Association: Modelica—A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.2*. [Online]. Available: http://www.modelica.org/documentas/ModelicaSpec32.pdf

[38] D. C. Karnopp, D. L. Margolis, and R. C. Rosenberg, *System Dynamics: A Unified Approach*. Hoboken, NJ, USA: Wiley.

[39] J. C. Willems, "The behavioral approach to open and interconnected systems," *IEEE Control Syst.*, vol. 27, no. 6, pp. 46–99, Dec. 2007.

[40] S. Dai and X. Koutsoukos, "Model-based automotive control design using port-Hamiltonian systems," in *Proc. Int. Conf. Complex Syst. Eng. (ICCSE)*, Univ. Connecticut, Nov. 2015.

[41] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *Eur. J. Control*, vol. 18, no. 3, pp. 217–238, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0947358012709433

[42] G. Simko and J. Sztipanovits, "Model integration in CPS," in *Cyber Physical Systems*, R. Rajkumar, D. de Niz, and M. Klein, Eds. Boston, MA, USA: Addison-Wesley, 2017, pp. 331–360.

[43] G. Simko, D. Lindecker, T. Levendovszky, S. Neema, and J. Sztipanovits, "Specification of cyber-physical components with formal semantics—Integration and composition," in *Proc. ACM/IEEE 16th Int. Conf. Model Driven Eng. Lang. Syst.*, 2013.

[44] G. Simko, D. Lindecker, T. Levendovszky, E. Jackson, S. Neema, and J. Sztipanovits, "A framework for unambiguous and extensible specification of DSMLs for cyber-physical systems," in *Proc. IEEE 20th Int. Conf. Workshops Eng. Comput. Based Syst. (ECBS)*, Apr. 2013, pp. 30–39.

[45] N. Bjørner, K. L. McMillan, and A. Rybalchenko, "Higher-order program verification as satisfiability modulo theories with algebraic data-types," in *Proc. Workshop Higher-Order Program Anal. (HOPA)*, 2013.

[46] A. Desai, G. Vivek, E. Jackson, E. Qadeer, S. Rajamani, and D. Zufferey, "P: Safe asynchronous event-driven programming," in *Proc. ACM SIGPLAN Conf. Programm. Lang. Design Implement.*, 2013.

[47] E. K. Jackson, "Engineering domain-specific languages with FORMULA 2.0," in *Proc. ACM SIGADA Annu. Conf. High Integr. Lang. Technol. (HILT)*, 2014, pp. 3–4.

[48] P. J. Antsaklis, "Special issue on hybrid systems: Theory and applications a brief introduction to the theory and applications of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 879–887, Jul. 2000.

[49] D. S. Weld and J. De Kleer, *Readings in Qualitative Reasoning About Physical Systems*. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[50] A. Tiwari, "HybridSAL relational abstracter," in *Computer Aided Verification*. Berlin, Germany: Springer-Verlag, 2012.

[51] J. de Kleer, "Fault augmented Modelica models," in *Proc. 24th Int. Workshop Princ. Diagnosis*, 2013, pp. 71–78.

[52] T. Honda *et al.*, "A simulation and modeling based reliability requirements assessment methodology," in *Proc. ASME Int. Design Eng. Techn. Conf. Comput. Inf. Eng. Conf.*, 2014.

[53] C. Hoyle, I. Y. Tumer, T. Kurtoglu, and W. Chen, "Multi-stage uncertainty quantification for verifying the correctness of complex system designs," in *Proc. ASME Int. Design Eng. Techn. Conf.*, Washington, DC, USA, 2011.

[54] A. Saltelli, S. Tarantola, and F. Campolongo, "Sensitivity analysis as an ingredient of modeling," *Stat. Sci.*, vol. 54, no. 4, pp. 377–395, 2000.

[55] Z. Lattmann, "An analysis-driven rapid design process for cyber-physical systems," Ph.D. dissertation, Vanderbilt Univ., Nashville, TN, USA, 2016.

[56] R. E. Bryant, "Symbolic manipulation with ordered binary decision diagrams," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-92-160, Jul. 1992.

[57] T. Bapty, S. Neema, J. Scott, J. Sztipanovits, and S. Asaad, "Model-integrated tools for the design of dynamically reconfigurable systems," *VLSI Design*, vol. 10, no. 3, pp. 281–306, 2000.

[58] S. Mohanty, V. K. Prasanna, S. Neema, and J. R. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," *ACM SIGPLAN Notices*, vol. 37, no. 7, pp. 18–27, 2002.

[59] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts, "Constraint-based design-space exploration and model synthesis," in *Proc. Int. Workshop Embedded Softw.*, Philadelphia, PA, USA, Oct. 2003, pp. 290–305.

[60] H. Neema *et al.*, "Design space exploration and manipulation for cyber physical systems," in *Proc. IFIP 1st Int. Workshop Design Space Exploration Cyber-Phys. Syst. (IDEAL)*, Berlin, Germany: Springer-Verlag, 2014.

[61] Z. Lattmann *et al.*, "Verification and design exploration through meta tool integration with OpenModelica," in *Proc. 10th Int. Modelica Conf.*, Lund, Sweden: Lund Univ., 2014, pp. 353–362.

[62] J. C. Gogolla, "Object constraint language (OCL): A definitive guide," in *Proc. 12th Int. School Formal Methods Design Comput. Commun. Softw. Syst.*, Bertinoro, Italy, vol. 7320, M. Bernardo, V. Cortellessa, and A. Pierantonio, Eds. New York, NY, USA: Springer-Verlag, Jun. 2012.

[63] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11th IEEE Int. Symp. Object Oriented Real-Time Distrib. Comput. (ISORC)*, May 2008, pp. 363–369.

[64] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*. Cambridge, MA, USA: MIT Press, 1990, pp. 99–1072.

[65] J. Sztipanovits, T. Bapty, Z. S. Lattmann, and S. Neema, "Composition and compositionality ion CPS," in *Handbook of System Safety and Security: Cyber Risk and Risk Management, Cyber Security, Threat Analysis, Functional Safety, Software Systems, and Cyber Physical Systems*, E. Griffor, Ed. Amsterdam, The Netherlands: Elsevier, 2017.

[66] H. Neema *et al.*, "Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems," in *Proc. 10th Int. Modelica Conf.*, Lund, Sweden, Mar. 2014, pp. 235–245.

[67] D. Gorissen, P. Demeester, T. Dhaene, and K. Crombecq, "A surrogate modeling and adaptive sampling toolbox for computer based design," *J. Mach. Learn. Res.*, vol. 11, pp. 2051–2055, 2010.

[68] S. T. Ounpraseuth, "Gaussian processes for machine learning," *J. Amer. Stat. Assoc.*, vol. 103, no. 481, p. 429, 2008, doi: 10.1198/jasa.2008.s219 .

[69] Y. Ben-Haim, "Order and indeterminism: An Info-Gap perspective," in *Error and Uncertainty in Scientific Practice*, M. Boumans, G. Hon and A. Petersen, Eds. London, U.K.: Pickering and Chatto Publishers, 2014, pp. 157–175.

[70] [Online]. Available: http://openmdao.org

ABOUT THE AUTHORS

Janos Sztipanovits (Life Fellow, IEEE) graduated from the Technical University of Budapest, Budapest, Hungary, in 1970 and received the Ph.D. degree from the Hungarian Academy of Sciences, Budapest, Hungary, in 1980.

He is currently the E. Bronson Ingram Distinguished Professor of Engineering at Vanderbilt University, Nashville, TN, USA, and founding Director of the Institute for Software Integrated Systems. His current research interest includes the foundation and applications of model and component-based design of cyber–physical systems, design automation methods, and systems-security codesign technology.

Dr. Sztipanovits is an external member of the Hungarian Academy of Sciences and holds the title of John von Neumann Professor of the Technical University of Budapest. He was founding chair of the ACM Special Interest Group on Embedded Software (SIGBED).

Ted Bapty received the B.S. degree in electrical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1985 and the Ph.D. degree in electrical engineering from Vanderbilt University, Nashville, TN, USA, in 1995.

He is currently a Research Associate Professor at the Institute for Software Integrated Systems, Vanderbilt University. His research includes model-based tools for design space exploration and validation of complex cyber–physical systems, tools and standards for aviation mission computing, and adaptive computing. He has published over 60 journal and conference papers and four patents. He is one of the founders of Metamorph Software Inc, a company formed to transition CPS technology to industry, and has served as Captain in the USAF, where he developed high speed real-time analysis for aerospace testing.

Xenofon Koutsoukos (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, USA, in 2000.

He is a Professor at the Department of Electrical Engineering and Computer Science and a Senior Research Scientist at the Institute for Software Integrated Systems (ISIS), Vanderbilt University, Nashville, TN, USA. He was a Member of Research Staff at the Xerox Palo Alto Research Center (PARC) (2000–2002), working in the embedded collaborative computing area. His research work is in the area of cyber–physical systems with emphasis on formal methods, data-driven methods, distributed algorithms, security and resilience, diagnosis and fault tolerance, and adaptive resource management. He has published more than 250 journal and conference papers and is coinventor of four U.S. patents.

Prof. Koutsoukos was the recipient of the National Science Foundation (NSF) Career Award in 2004, the Excellence in Teaching Award in 2009 from the Vanderbilt University School of Engineering, and the 2011 NASA Aeronautics Research Mission Directorate (ARMD) Associate Administrator (AA) Award in Technology and Innovation.

Zsolt Lattmann received the M.Sc. and Ph.D. degrees in electrical engineering from Vanderbilt University, Nashville, TN, USA.

Currently, he is a Project Executive at A^3 focusing on advanced design and manufacturing. Most recently, he was Head of Research, Development, and Product Strategy at MetaMorph Inc., a spinout from Vanderbilt University's Institute for Software Integrated Systems. He spent three years as one of the lead developers on a DARPA's Adaptive Vehicle Make initiative, aimed at reducing drastically the development times for complex cyber–physical systems.

Sandeep Neema received the B.Tech. degree from the Indian Institute of Technology, New Delhi, India, in 1995, the M.S. degree from Utah State University, Logan, UT, USA, in 1997, and the Ph.D. degree from Vanderbilt University, Nashville, TN, USA, in 2001, all in electrical and computer engineering.

Currently, he is a Research Professor of Electrical Engineering and Computer Science at Vanderbilt University, and a Senior Research Scientist at the Institute for Software-Integrated Systems at Vanderbilt University. He has over 15 years of experience in model-based design tools and cyber–physical systems (CPSs). His research interests span model-based design and manufacturing tools for CPSs, design space exploration and constraint programming, applications of mobile computing, distributed real-time systems, and embedded signal processing.

Ethan Jackson (Member, IEEE) received the Ph.D. degree in computer science from Vanderbilt University, Nashville, TN, USA.

Currently, he is a Senior Researcher and Director in the Clinical Sensing & Analytics Group at Microsoft NExT, Redmond, WA, USA. His research focuses on intelligent robotic platforms that monitor the environment to make people—and the ecosystems they inhabit—healthier. Ethan applies his background in cyber–physical systems, program verification, and programming languages to develop trustworthy, robust, and field-deployable platforms. He leads Project Premonition, an advanced health project which aims to detect the movements of potential pathogens in the environment, before they cause outbreaks in humans. He is the creator of the FORMULA system for building domain-specific programming languages and enabling formal analysis of complex software, which has been used in large academic and industrial settings. He is also the co-creator of the P programming language which allows developers to specify complex systems of communicating asynchronous components, and has been used to design critical components of Microsoft Windows.