Electronic Communications of the EASST
Volume X (2010)

Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)

A Multi-Modeling Language Suite for Cyber Physical Systems

Sandeep Neema, Ted Bapty, Gabor Karsai, Janos Sztipanovits, David Corman, Thomas Herm, Douglas Stuart, Dimitri Mavris

X Pages

Guest Editors: Vasco Amaral, Hans Vangheluwe, Cécile Hardebolle, Lazlo Lengyel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Multi-Modeling Language Suite for Cyber Physical Systems

Sandeep Neema, Ted Bapty, Gabor Karsai, Janos Sztipanovits,[(1)]
David Corman, Thomas Herm, Douglas Stuart, [(2)]

(1) Institute for Software Integrated Systems,
Vanderbilt University, Nashville, TN, USA

(2) The Boeing Co.,
St. Louis, MO, USA

Abstract: The design of cyber-physical systems is complicated by the fact that not only the physical and computational aspects of the systems are critical, but so are the interactions between the two. The paper introduces a concept based on multi-modeling and a multitude of modeling languages to address these issues in an advanced design flow. The main ideas are described, together with notional tool architecture for a 'language suite'. The material presented in this paper is work in progress.

Keywords: Cyber Physical Systems, Multi-Modeling, Domain-Specific Modeling Languages

1 Introduction

The verifiable and producible design of cyber-physical systems (like autonomous vehicles) is a challenge to the entire engineering community. Existing engineering practice follows the classical systems engineering design process model, which includes requirements and design loops, with parallel system analysis and controls development. The model has worked well for 50+ years, but it has not prevented costly re-design cycles. Furthermore, it has left much of the verification of the system to the testing phase, and it did not cope well with variations and advances in implementation technology. The appearance of cyber-physical systems (that cannot function without computing) has further compounded the problem because interactions between the computational and physical aspects of the system were often poorly understood.
As the complexity of systems increases, the missions expand, and the design cycle compresses, we need better design processes to succeed. We need methods to manage the complexity of the systems while maintaining their adaptability to potential future missions.
The main sources of problems that hurt productivity of current design flows are as follows:
Complex interactions across engineering domains: Cyber-physical systems involve many diverse interactions (designed and accidental) among components, in different physical domains (mechanical, thermal, electromagnetic, electrical, hydraulic, etc.) and across the interface between computational and physical. Interactions can be expressed as dependencies at design time or as physical power flows and information flows at run time. If designers and their design process are not prepared to recognize and manage such interactions, developments can suffer lengthy design iterations, and, at worst, failure due to unintended interactions discovered late in the design cycle, at integration time.
Implementation and platform constraints: While the consumer electronics industry has produced highly complex system designs based on solid state technology, existing cyber-physical systems engineering practice for complex (e.g. electromechanical) platforms still lags

behind. The main difference between the two industries is the existence of a well-defined separation between the design layer and the implementation layer: i.e. the platform. For digital electronics the 'platform' is provided by the semiconductor technology that (1) guarantees specific abstractions (e.g. digital logic levels, noise immunity, timing properties) given that (2) the designs created for it obey the technology constraints. In other words, any design that follows the rules, will work on that platform. This is due to significant design margins in digital logic, and the high degree of isolation between subsystems on a chip. In other engineering disciplines, for instance aerospace vehicles, the degree of component/subsystem interaction is much higher. Even so, opportunities are missed, e.g. an embedded software 'platform' abstraction is not used. Hence, design becomes very complex as the design flow has to continuously iterate across different levels of weakly defined of abstractions to track the subsystem interactions and its resultant complexity. Platform constraints and detrimental interactions are discovered too late, which lead to costly iterations. Needed are tools that can help the designer avoid complex systems where possible, using design tools to find less complex solutions.

Problems detected during system integration: Conventional systems engineering and classical software engineering (with the most egregious example of the waterfall model) places integration at the end of the engineering process: when designs are locked-down and implemented, only then do we integrate the engineering products. This approach works only if the interfaces -- both the explicitly designed and accidental ones -- are precisely defined and implemented, and thus all the interactions among components are understood and considered well ahead of integration time. Typically, interfaces are designed first and frozen. Late changes in related subsystems and feature creep cause increases in interface (and interaction) complexity thus interfaces change. When interfaces are often underspecified or not considered, their late discovery is a major cause of integration problems. We need a better way of integrating systems, such that (1) interfaces and interactions are understood at design, (2) integration happens early and often to verify these interactions, and (2) potential problems at integration time are minimized by detection and rectification much earlier.

Verification through exhaustive testing: Testing is the ultimate answer to system validation and verification (V&V) in the industry today. There is a good reason for this: there is no other, universally accepted method for V&V to build confidence in the design and the implementation for highly complex systems. But testing is expensive, especially if it is done rigorously and systematically (i.e. test every component, then test every subsystem, then test the entire system, through its entire region of operation, etc.). Too-high costs leads to incomplete testing and thus verification, resulting in undiscovered systems problems. To reduce total design cycles we need reduction in testing and greater effective verification coverage.

Clearly, a multi-pronged approach is needed to address these challenges[1]. One of the core challenges is the need for a common design representation language, which covers the diverse disciplines involved in the design of a complex CPS. Arguably, such a design language can

[1] Karsai, G. and Sztipanovits, J. 2008. Model-Integrated Development of Cyber-Physical Systems. In Proceedings of the 6th IFIP WG 10.2 international Workshop on Software Technologies For Embedded and Ubiquitous Systems (Anacarpi, Capri Island, Italy, October 01 - 03, 2008). U. Brinkschulte, T. Givargis, and S. Russo, Eds. Lecture Notes In Computer Science, vol. 5287. Springer-Verlag, Berlin, Heidelberg, 46-54. DOI= http://dx.doi.org/10.1007/978-3-540-87785-1_5

capture the system design in an integrated manner; expose the factors leading to complexity, and enable adaptability, and verification. The flexibility to span all necessary domains and the power to capture details and interactions are critical. In the rest of this paper we present arguments for a Domain-Specific Modeling based approach, and our proposal for a Multi-Modeling Language Suite (referred to as the META Language suite in the rest of this paper).

2 Motivation for Domain-Specific Modeling Languages for CPS

In all approaches to model-based design, modeling languages play fundamental role. This is reflected by the large number of modeling languages that have been promoted during the past decade. From the point of view of their intended role, they fall into the following three categories:

1. Unified (or universal) modeling languages (such as UML/SysML[2] and Modelica[3]) that are designed with goals similar to programming languages. They are optimized to cover a broad domain and advocate the advantage of being able to remain inside a single language framework independently from the domain and system category. Necessarily, the core language constructs are tailored more toward an underlying technology (e.g. object-oriented modeling) rather than to a particular domain - even if extension mechanisms, such as UML profiling, allow some form of customizability.
2. Interchange languages (such as the Hybrid System Interchange Format - HSIF developed in DARPA's MoBIES program) are designed for sharing models across a group of analysis tools. Interchange languages are optimized for providing the simplest modeling language that can express the semantics of an analysis domain (e.g. hybrid system). Using the interchange language as a common semantic platform, tools can communicate by providing two-way translators between their native modeling language and the interchange language.
3. Domain-specific modeling languages (DSMLs) are tailored to the particular concepts, constraints and assumptions of application domains. While the universal language can also be considered as DSMLs, the language design approach is different. In DSML-based approaches the languages are optimized to be focused: the modeling language should offer the simplest possible formulation that is still expressive enough for the modeling tasks. The biggest technology difference is that model-based design frameworks that aggressively use DSMLs, need to support the construction and evolution of modeling languages without sacrificing semantic precision.

Heterogeneity of CPS products makes language design a challenging problem. Selection of a "universal" modeling language could be a comforting idea with the implicit promise that standards- sooner or later – lead to COTS tool suites. Unfortunately, this argument has the following weaknesses: (a) Feasibility of creating and end-to-end tool chain demands the use of established tools (e.g. Simulink/Stateflow for control system design) to take advantage of their extensive existing investment in libraries and the familiarity with their use. (b) Many examples show that standards developed for "universal modeling language" must be constantly changed to keep the language relevant. This change starts eroding the primary argument of their adoption: having a stable language environment where tools can develop for a longer time.

[2] Object Management Group, "SysML v.1.1", November 2008

[3] Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification V. 3.2, March 2010

This issue can be observed in much of the broad-based standardization efforts during the past decade, such as the AUTOSAR standard developed for automotive software. After 8 years, a number of Tier-1 manufacturer-specific versions have emerged. It is hard to predict bounds for this divergence. (c) Progress in all aspects of model-based design makes the modeling domain open: innovations bring new analysis and synthesis methods that need to be adopted and integrated in design flows. While most "universal language" approaches offer some extension mechanism, these are considered secondary from the language design point of view. For example, UML stereotypes are free of semantics that leads the loss of fundamental advantages of model-based design, such as automated well-formedness testing of models and tool supported composition of metamodels.

This paper advocates the use of DSML-based language design. The main properties of our approach are the following:

1. The language suite is open and evolving. The component languages include user defined abstraction layers and modeling languages of established tools that become part of the CPS design flow.
2. Tool interoperability is established through the explicit representation of semantics for the constituent DSMLs through metamodeling. Metamodeling will be extensively used also for describing the formal relationship across DSMLs as specification of model transformations.
3. The language suite includes a wide-spectrum integration language that supports high-level design on different levels of abstraction, has facilities for expressing design choices (and thus design spaces), and serves as a conceptual integration tool across modeling domains and functional subsystems.

The proposed DSML-based approach is supported by Vanderbilt's Meta-Level Tool Architecture shown in Figure 1. Domain-specific modeling, modeling languages and tool chains (top layer) are specified and integrated using the Meta-Level methods and tools. The Meta-Level includes metamodels of component DSMLs, and metaprogrammable tools for modeling (Generic Modeling Environment – GME), model data management (Unified Data Model tool – UDM), model transformation (Graph Rewriting and Transformation tool - GReAT), and tool integration (Open Tool Integration Framework – OTIF). These tools are part of Vanderbilt's open source Model-Integrated Computing tool suite[4] that have been developed over two decades and have been matured in a wide range of applications from chemical to automotive, manufacturing to defense. The Meta-Level tools

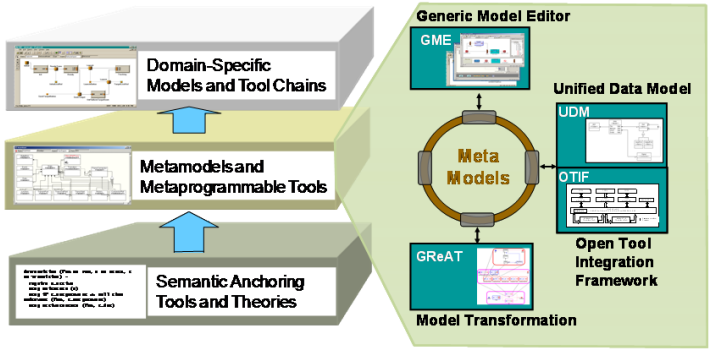

Figure 1: Meta-Level Tool Architecture

[4] G. Karsai, A. Ledeczi, S. Neema, and J. Sztipanovits. The model integrated computing toolsuite: Metaprogrammable tools for embedded control system design. In *Proceedings of the IEEE Joint Conference CCA,ISIC and CACSD*, Munich, Germany, 2006.

constitute a "language engineering environment where DSMLs and tool chains can be rapidly designed and evolved. This process is supported by metamodel libraries and tools for composing and migrating modeling languages and models. The Meta Level is built on the Semantic Level that includes the theories methods and tools required for making the overall architecture sound.

3 Precision Component Multi-Modeling

Multi-Modeling in simplest terms is an ensemble of models representing physical and computational artifacts. For example, a hydraulic actuator is described with fluid dynamics models that relates the pressure and flow and interfaces with fluid transports, and the same hydraulic actuator is also described with structural dynamics models that relate the structural load, forces, acceleration on its mechanical components, etc. These two models are not alternative descriptions – both are required to represent the complete behavior of the hydraulic actuator. This example might rhyme with "multi-domain" modeling – as it is referred to in Modelica (and in Simulink). Our use of term multi-modeling, however, spans beyond multi-domain modeling, and encompasses multi-resolution (ex: lumped parameter vs. an ODE

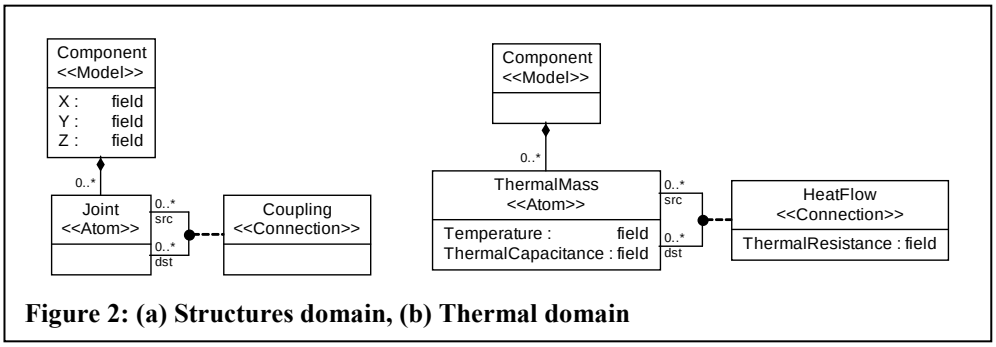

Figure 2: (a) Structures domain, (b) Thermal domain

dynamics model of the same artifact), and multi-abstraction (ex: finite state machine vs. digital circuit representation of a digital device) modeling.

Nevertheless, multi-modeling (even beyond multi-domain) as described above is already the state of the practice. Domain engineers in Aerospace (and other CPS systems), routinely use multiple models and tools to describe various systems, component, and sub-systems, and a single subsystem shows up in multiple models. See for example Figure 6, which shows some of the major subsystems of an aircraft, and their associated modeling considerations. What is significantly lacking in current practices however is that the models are at best loosely connected, and largely treated as "orthogonal aspects". Real-world physics, however, dictates that these modeling aspects or domains are not truly orthogonal: they are interdependent through interactions or physical laws creating constraints across domains. Unmodeled and unmitigated interactions between system components result in emergent behaviors and loss of performance and predictability leading to major difficulties during system integration or worse yet, following deployment. Remediation this late in the development process results in large cost and schedule overruns.

The key innovation that we propose is, therefore not the status-quo of multiple models, but rather Precision Multi-Modeling – which advances multi-modeling from a mere "ensemble" of

models to a formally and precisely integrated, mathematically coupled suite of models. The precise integration, in simple terms, means that we:

a) define formal DSML-s for the different modeling domains (such as Thermal, Electrical, Software, …) along with their structural and behavioral semantics;
b) we define cross-cutting interactions between domains as explicit relations between different DSML-s. These relations take two forms: 1) structural interactions expressed directly as metamodel composition; 2) behavioral interactions expressed as metamodel mapping;
c) we provide modeling environment (using GME) in which domain modelers can create integrated models, and we provide tools for integrating (via active linking – change in one model is automatically transformed and propagated to another model) models created within different tools. The effect is that cross-cutting interactions are explicitly represented (possibly automatically derived via transformations).

We illustrate this idea with a simple pedagogical example: consider the structures domain where components are described along with their physical layout and coupling, and also consider the thermal domain where components are described with their heat capacitance, temperature, and thermal resistance. Simple metamodels of DSML-s for the two domains are shown in Figure 2. The diagram reads as follows: (a) in the Structures domain there are *Components* (properties: *x, y, z* coordinates) with *Joints* that are joined via *Coupling*, and (b) in Thermal domain there are *Components* with *ThermalMass (*properties: temperature and capacitance*)* and joined via *HeatFlow (*property: *resistance)*. If we intend to model structural interactions we need to compose the two DSMLs (Figure 3). The result of the composition is an integrated DSML, which has *Components* with *Joints* and *ThermalMass* that are joined via *Coupling* and *HeatFlows*.

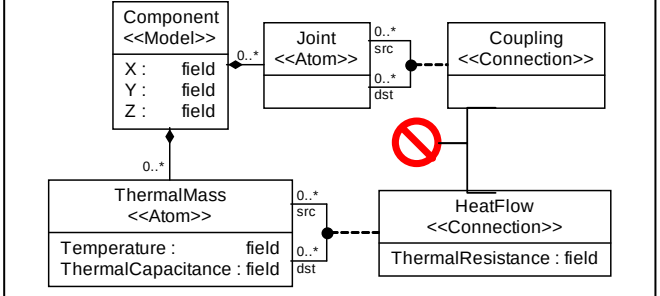

Figure 3: Structures + Thermal domain composition

The composed DSML allow-s integrated modeling of Structures and Thermal domains of a component. However, notice that the integrated DSML contains a constraint – a well-formedness rule (stop-sign symbol in Fig) - with the restriction that for every Coupling between two Components, there is a corresponding *HeatFlow*. Models that do not satisfy this constraint are rejected by the GME modeling tool.

Behavioral interactions are defined by a mapping that relies on behavioral semantics of domains. In this example we see that components that are in proximity (defined by location) will also have convective heat flows across them – even though there is no direct link. We define a mapping that creates a *HeatFlow* connection across components when they are in proximity (distance < K), and sets the *ThermalResistance* property of the *HeatFlow* based on the convection properties of the medium.

The implications are significant. Even with this simple example we demonstrated that metamodel composition captures cross-domain interactions (as constraints) that can be validated (with lightweight automated constraint checkers). We also demonstrated that cross-

domain interactions can be automatically derived (with concept mappings). Furthermore, since interactions are a primary source of complexity, the ability to express and deduce interactions, is a major step towards driving complexity metrics.

Precision Component Multi-Modeling is at the core of our approach, and the Language suite realizes a (significantly) larger version of the pedagogical example. It is important to note that using our Meta Level Tool Architecture approach, we can support Precise Component Multi-Modeling even if different modeling domains are captured by different (possibly COTS) tools. In this case we create a model integration language that includes metamodels of the constituent modeling tools. The integration language composes these metamodels and explicitly captures the cross-domain constraints and mapping across the modeling domain as described above.

4 META Language Suite

The META Language suite is a Precision Multi-Modeling Language that incorporates META-

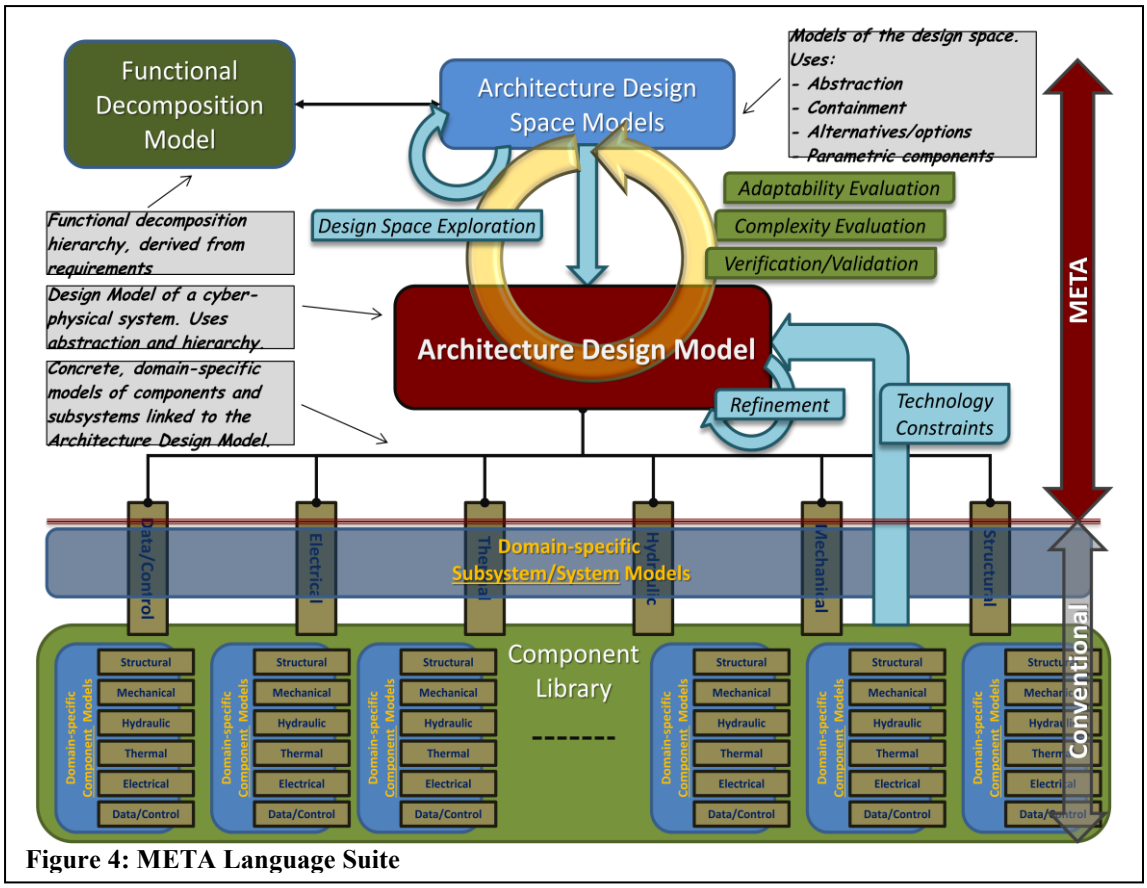

Figure 4: META Language Suite

specific novel languages, as well as integrates established Domain Languages. Figure 4 shows the META Language stack identifying the novel META-specific language as well as "Conventional" languages (those supported by existing domain tools – listed in Figure 6 below).

The vision for the META-specific design language is that of a wide-spectrum integration language that supports high-level design on different levels of abstraction, has facilities for expressing design choices (and thus design spaces), serves as a conceptual integration tool

across modeling domains and functional subsystems, and most importantly incorporates abstractions and attributes to drive various complexity and adaptability metrics for design. The design language has its own semantics but in itself it is not suitable to express detailed designs (or actual physical or software components), rather it serves as an abstraction layer above concrete, detailed models of components that is capable of capturing cross-domain and cross-system interactions, both in the physical and the cyber domains. The language is also capable of expressing implementation technology constraints as they have impact on the design.
We envision the META design language having two layers: (1) the Architectural Design Space Model (ADSM) layer, and (2) the Architectural Design Model (ADM) layer. The ADM layer contains system-level models which are abstract functional, structural, and behavioral models of the system and its components. The ADM supports design through modeling and low-fidelity analysis of the entire system, with interaction across all domains. Note that the models here are 'high-level': they are more abstract, less detailed than component models or subsystem models but they capture salient properties of and interconnects (i.e. potential interactions) among components. An ADM model acts as a high-level global blueprint for the system that captures the high-level physical and computational interactions across the components.

The ADSM layer adds an abstraction layer on top of the ADM layer to support (1) *design space modeling* and (2) *product line architectures* with variants. In contrast to the ADM models (that represent concrete 'point' designs) , ADSM models capture entire families of designs, with potential variants or parametric 'skeleton designs' for subsystems and components. The use of ADSM allows the designers to (1) execute design space exploration and (2) build and work with product lines. We envision that the ADSM layer supports compositional modeling of design spaces and product-line architectures that can be subjected to design exploration activities, which will produce ADM point-designs. The ADM models are linked to their original design spaces through appropriate constructs in the language, so designers can keep track of what design space / product line their point-designs belong to, and what exploration process and parameter settings lead from a set of designs to a specific design.

The ADSM and ADM models are also linked to a functional decomposition of the system. Functional decomposition, i.e. the mapping of requirements to system and subsystem functions is the first part of a design process and provides a useful initial starting point in the design process. It also provides support for validation: it can be used to check that all desired functions are assigned to at least one component in the final architecture, and similarly, all components contribute to at least one function. It is important to maintain these links between the elements of the functional decomposition and the design artifacts because as requirements change, we need to track the impact of that change on the design.

The ADM models are linked to detailed domain-specific models that are models for the actual implementation. An ADM model is a model of an abstract component, an abstract assembly, an abstract subsystem, or the abstract system. ADM models capture the structure of the complex system: its ingredients and all material, energy, and information flows among those ingredients. These flows are concrete, but they are also implicitly capture all potential interactions, including power exchanges and data exchanges, but also potentials for fault effect propagations. A critical problem in design is the management and understanding of such interactions. The first step towards this goal is to express and model such interactions.
The domain-specific models include component models and models for component assemblies (up to subsystems). A component model is typically multi-faceted where each facet represents

an engineering domain or interface type. For instance, a hydraulic actuator has a 'hydraulic interface', a 'mechanical interface', and a 'data/control interface' that may be represented on one model (with three facets) or with three dependent models. On the ADM level the component/assembly is represented by a single, abstract model with the three interfaces. This ADM model component/assembly is then linked to (potentially many) domain-specific, concrete models of the component. For instance, a fuel system for an aircraft includes various components (tanks, pipes, pumps, control valves, etc.), and the fuel system can be simulated by composing models of the components. This 'fuel system model' is a high-fidelity, fine-grain model of the entire subsystem.

We are developing tools that allow the tight linking and automated mapping of the ADM models to domain-specific component models and component assembly (or subsystem models). Expressing and maintaining such links is essential for the design process, with dependencies that can be automatically tracked and analyzed during the design process. The Figure 5 below shows a first cut at the high-level organization of the ADM, ADSM, and Functional modeling language, expressed as a UML diagram that shows the metamodel of the language.

The functional decomposition model lists all the system functions in a hierarchical structure, possibly linked to each other through non-hierarchical, cross-cutting dependencies. The architecture design space models contain various model elements, including alternative and optional elements, while the specific architecture design models contain only assemblies and components. In both cases design elements interact through various flows. Functions are explicitly mapped to design elements in both the design space and in the specific designs (horizontal dashed lines with the tag <<mapsTo>>). The bottom part of the diagram sketches

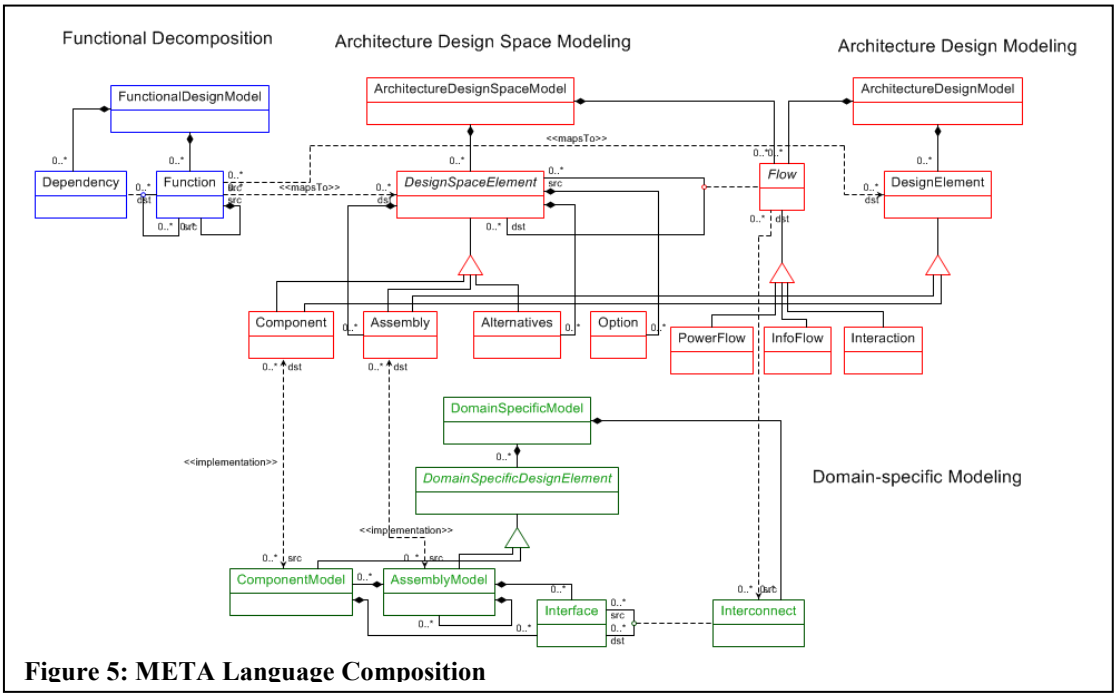

Figure 5: META Language Composition

the organization of the domain-specific models (which are stored in third-party, domain specific tools) – these are concrete models using components and assemblies, interconnected

through interfaces. The abstract design elements (components, assemblies, and flows) are linked to these implementation elements (vertical dashed lines, with the tag <<implementation>>).

The design modeling language will support several techniques for model organization, including object-orientation, and port- and interaction-based composition for both physical and computational components. The use of classification hierarchies and type systems have proven very successful in organizing knowledge, data, and software systems, hence we need to be able to express inheritance hierarchies in the models. Model library elements shall be defined as types (and the modeler will be able to define new types), and specific designs can use the instances of these model types. The advantage of this approach is that if the type definition changes, then all model instances of those types will be updated.

Components connectivity is captured in heterogeneous ports: physical 'ports' that represent physical interfaces where some generalized 'power flow' takes effect; 'signal ports' represent information flow. The physical manifestations of power include: mechanical, fluid, thermal, electrical, etc. Information flows ('signals') and power flows can interact (e.g. sensor convert power-related quantities into signals, actuators use signals to modulate power flows, etc.). The design models need be able to capture all these aspects.

Domain Model	Concepts	Systems Modeled	Tool Examples
Structural	3D/CAD Structural Geometry, Shapes, Materials, CAE, FEA, Gross Weight, FEM	Fuselage, Empennage & Wings, External & Internal, Shapes Dimensions & Materials	Solid Works, NASTRAN, PATRAN, V5 Catia, ENOVIA, NX
Electrical	Generation, transport & consumption of electrical power.	APU, TE Generator, Electrical Acutators, other Electrically Powered Systems	Common Electrical/Electronic Data System
Electronic	Internal transmission and processing of all digital information.	Flight Control, Munition Management, Sensor Management and Processing, Displays, Comms, etc	Cadence Tool Suite
Aerodynamic	3D Models --> Grids, Shapes, Grid Sampling	Wings, Fuselage, Tail, Nacelle, Stores	CFD Tools, Grid Tools, Wind Tunnels, MultiSurface Aerodynamics
Fluid Transport Models (Hydraulic, Fuel, Pneumatic, Etc.)	Fluid flow, Pumps, Valves, Pressure, Flow Rate, Hyro-circuit Schematics	Hydraulic Actuators, Fuel Storage and Distribution, Coolant Delivery	ICCA Hydralics
Electro-Magnetic Emissions and Interference	E&M Fields, Antennas, Frequencies, Radiation, Shielding, Lightning	Radar, Electronics and Electrical Switching Systems	EM Explorer, EMS Plus, Rsoft, SPEAG
Thermal	Heat flow, conduction, convection, radiance, Thermal Mass, Heat Capacity, Heat Input/Heat Sin	Thermal Management Systems, Fuel Thermal Mgt System, Power Thermal Mgt System, Electronics, ...	THERMOD, NETHeat, ThermalModel, PowerTherm, Bond graph
Guidance/Nav/Control	Hybrid Dynamics, Discrete Event, TTA, ...	Flight Control,	Matlab/Simulink,
Propulsion	Thrust, Weight, ...Physical layout of solid state models including interfaces to fuel, hydraulic, electrical and control systems	Emgines, Nacelle	ECAP, PARA, AEDsys, PERF, COMPR, INLET
Mission	Sensor Payload, Range, Objectives, Peformance Reqts, Mission Computer, etc.	Airframe, Requirements,	AMP, STK
Manufacturing	Virtual Prototyping, Evaluation of design concepts,Supplier-s, Assembly Blueprints, Bill of Material, Work Instructions, Maintenance Instructions	All Components and Subsystems	IPPD, ACCEM, P-BEAT, SEER-MFG
Observability	Thermal Radiance, Radar Cross Section, Electromagnetics	Airframe, Engines, Paints, Electronics/Wiring	EMAP, ELMER, NEC2++, T-MATRIX

Figure 6: Modeling Domains, Concepts, and Tools

Analysis of ADM Models
Based on our previous work on cyber-physical system modeling we plan to use the extended bond-graph (BG) formalism for the integrated analysis of the ADM models. BGs provide a language for spanning domains (each domain being a specific branch of physics, e.g. mechanics, fluid dynamics, thermodynamics, etc.). Extended BG-s have capabilities to model switched interconnects among the elements, non-linear behaviors, and the interactions between information flows and physical elements. On previous projects we successfully used these techniques to model aircraft fuel systems and their controller, as well as aircraft landing gear systems. Extended BGs are natural for component-based design: component models can be combined into assemblies by connecting their (power and signal) interfaces. Furthermore, dynamic system simulations can be automatically derived from the models, as well as the precise causality propagations and dependencies between components can be computed via an algorithm, allowing very modeling of sophisticated interactions among cyber-physical system components. Note that extended bond-graphs can model both continuous time and hybrid (discrete switching) systems, and these models can be co-operated with synchronous (clock-driven) models in an appropriate simulation framework.

Domain-Specific Models
Figure 6 represents a first-cut at a set of domain design modeling areas, the concepts addressed, example aircraft systems modeled, and example openly available tools. The table captures a starting point for the combined, detailed domain models added to the META-specific languages identified here.

ADM/Domain-Specific Model Integration
Information flow from ADM to Domain-Specific (DS) models is needed to:

- Process the elaborated ADM models and instantiate DS models as the starting point for analysis.
- Update parameters from ADM to DS after using Design Space Optimizations.
- Update ADM parameters with results from DS analysis.
- Add links to ADM based on interactions discovered in DS analysis

Each of the domain-specific tools and their models must integrate into the overall system. Specifically, for each component/subsystem/system in the ADM, we must capture the linkages to the matching entities (components, connections, parameters) in the Domain-Specific tools. This is not a 1 to 1 mapping of parameters and components between ADM and the DS models. Consequently, the semantic mappings must be defined to create/update DS entities from ADM, and to update/back-annotate ADM entities from the DS models.
The semantic mappings can range from trivial (e.g. an ADM component parameter is the same as a DS component parameter) to very involved (e.g. mapping domain concepts and structure into a simplified abstraction, converting detailed phenomena like aggregate airflow thru a complex geometry to a consolidated convection heat flow). This can be handled with multiple approaches. Simple transforms can be implemented directly within the tools. Moderate complexity can be addressed through complex and validated model transformations developed for high-payoff, frequently used paths. For complex, problems, designer intervention is required.

5 Related Research

Based on our evaluation of the research literature, there are many candidates that aim at providing a design language however we found shortcomings in each of these. SysML[5] is proposed by OMG for system-level design and it supports a number of relevant concepts. However, it does not support type refinement (i.e. classification hierarchies), does not support physical dimensions on parameters, and it has too much overlap (and possible confusion) with UML. AADL[6] is an architecture modeling language, mostly for embedded computing systems, modeling both hardware and software aspects -- not the physical elements, and their interactions. Modelica is a universal simulation language that has sophisticated libraries for various physical domains, but it is unclear how software behaviors, software components, and connections to the physical elements can be modeled with it. The HRC (Heterogeneous Rich Component) model of the SPEEDS project[7] comes close to our concepts, but on closer inspection it turns out that they use a synchronous model of computation (not sufficient for continuous time systems) and they lack the physical component modeling aspect.
One common theme missing in all approaches evaluated was the lack of openness and the concept of linking the models to other, third-party models and functional decomposition. We believe the design language should allow links to the domain-specific models (of components and assemblies) and should act as an 'umbrella' model that connects and integrates the related models. If such a design model is available, one can execute heterogeneous, cross-domain analyses – for instance via integrated simulations. If the model dependencies are precisely mapped out, changes on the design (leading to costly design iterations currently) can be analyzed and the iteration kept limited.

6 Summary

This paper presents the need for and a proposal for a multi-modeling language suite to support design of complex Cyber Physical Systems. The heterogeneity and multi-domain nature of CPS design requires multi-modeling language that can capture integrated representations of components spanning multiple domains. We advocate an approach labeled Precision Component Multi-Modeling for integrated modeling of multi-domain components. The integrated modeling language suite consists of an Architecture Modeling Language that integrates multiple Domain models. The linkages are defined via meta-model composition and are implemented as model transformations. One principal challenge lies in cross-domain analysis of such integrated multi-domain models. We are proposing to use Hybrid Bond Graph as a common, high-level semantic domain to support multi-domain analysis. The proposed language suite will be linked into a larger context that develops a comprehensive design flow that spans the entire gamut of CPS design ranging from requirements, design, analysis, synthesis, verification, and manufacturing.

[5] www.sysml.org

[6] www.aadl.info

[7] www.speeds.eu.com