Modeling for Structural Adaptation: Lessons Learned from Model-based Design

Janos Sztipanovits

Institute for Software Integrated Systems
Vanderbilt University
Email: janos.sztipanovits@vanderbilt.edu

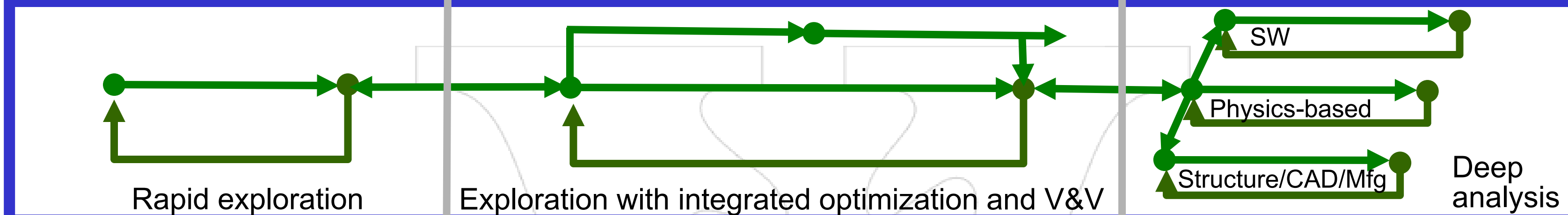
Caltech
1 August 2012

Overview

- Model-Based Design for CPS in META
  - Design flow and design infrastructure
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Thoughts on Resilience

CPS Design Flow

Architecture Design		Integrated Multi-physics/Cyber Design			Detailed Design	
Modeling	Exploration	Modeling	Simulation	V&V	Modeling	Analysis



Architecture Design

- Rapid architecture exploration
- Engineered design space
- Static constraints (multiphysics)
- 100s of “interesting” configurations remains

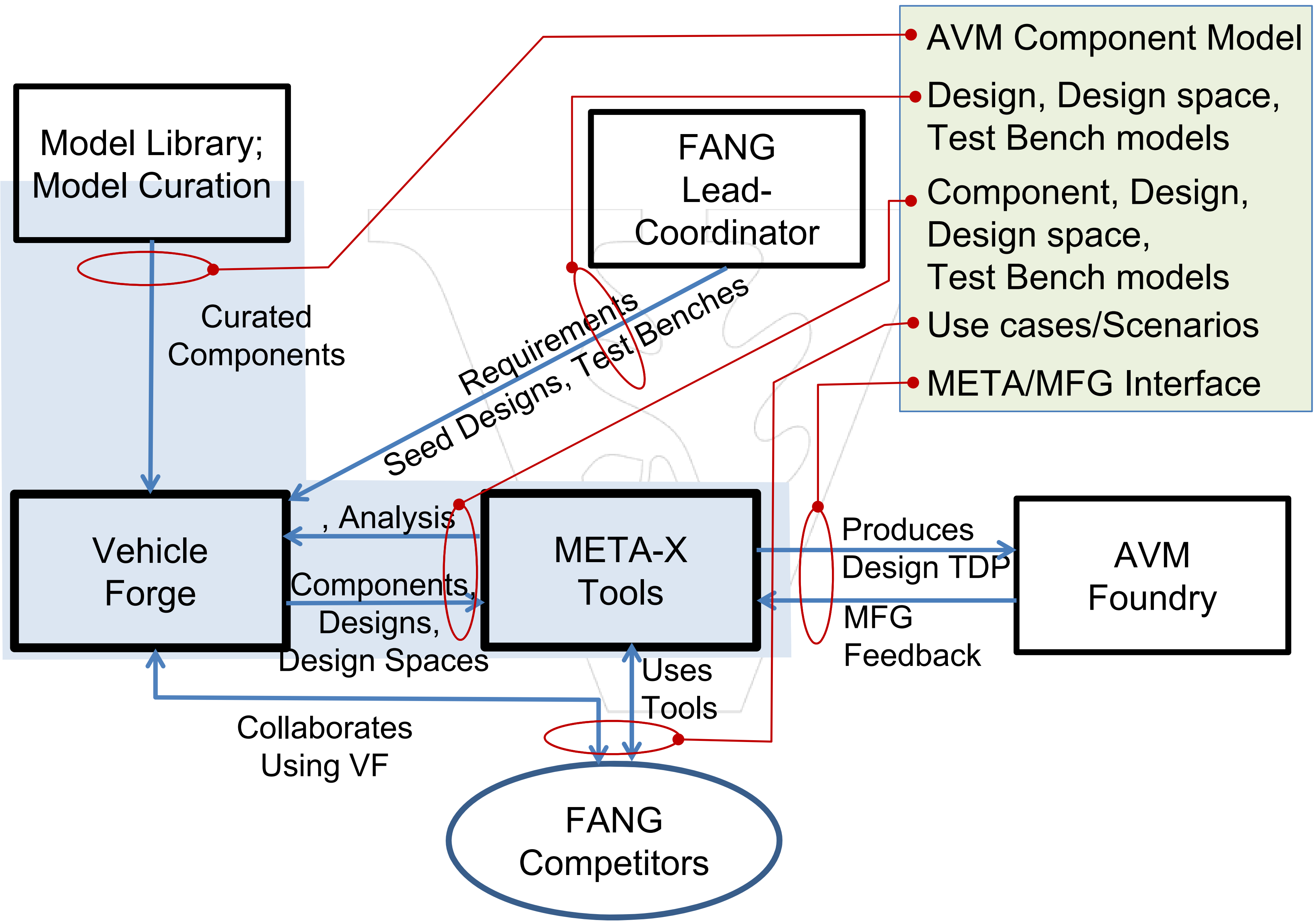
Integrated Multi-physics/Cyber Design

- Behavior exploration using lumped parameter dynamics
- Basic structure synthesis (CAD)
- Manufacturing Constraint Checking
- Composition with testbenches, PCC
- Parametric design tradeoffs
- Selecting few target configurations

Detailed Design

- “Deep”Domain Analysis
  - CAD
  - FEA; thermal, fluid…
  - Surrogate gen.
- Detailed Mfg. modeling
- RT SW modeling, code generation and testing

Interaction Among Major Program Components

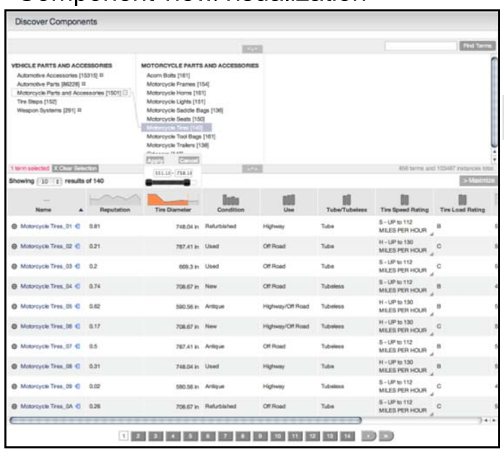


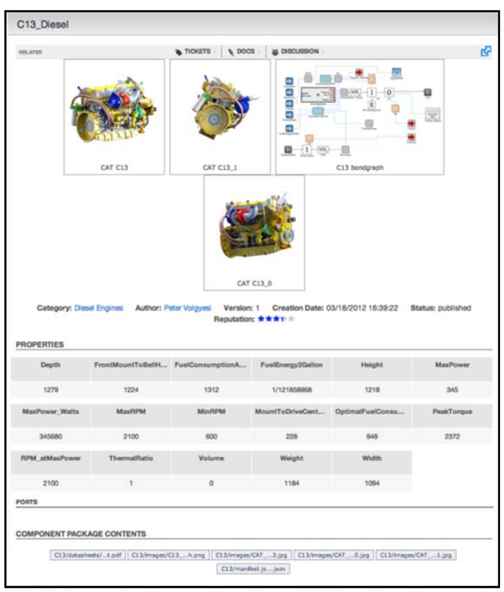
Vehicle.Forge

(or Mechanism for “Horizontal Transfer”)

Components

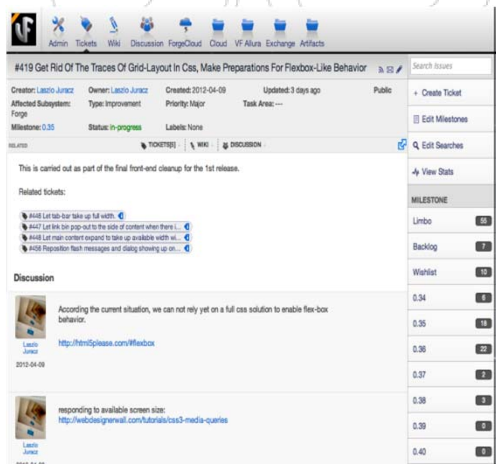
• Component discovery interface based on taxonomical- and faceted search
• Component view/visualization





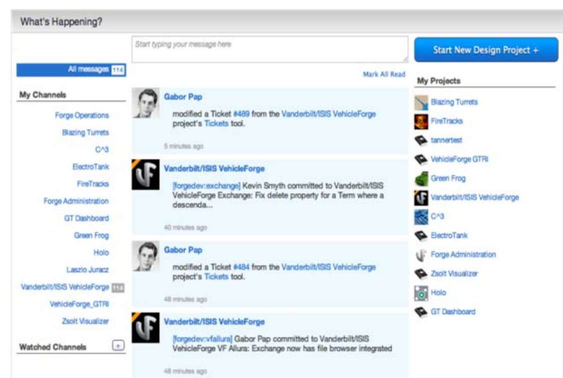
Design Projects

• Self-provisioned collaboration tools
  • Wiki,
  • Discussion Forum,
  • Issue tracking for managing team work.
• Git/SVN repositories for design artifacts
• Project and tool-based permission control
• Notification and Messaging system (in e-mail or as Dashboard messages)
• Set of available tools is extensible



Designers

• Public profile to show recent activities and involvement in design projects
• Designer portfolio publishing résumé and for self-promotion
• Find designers based on expertise and résumé
• Private profile for customizing account and notification settings
• User dashboard showing feeds of activities from projects, public/private messages from other users, announcements from forge-message channels



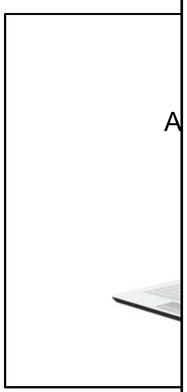
Vehicle.Forge Services

Guided

- Project templat
- Built-in visualiz
- META

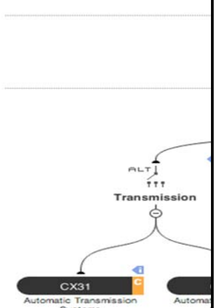
Compo

- Curated
- Taxom discove
- Drag-a subscri
- Chang
- Proven



Design

- Sharing artifacts reposito
- VF uses forge art
- VF agen e.g., by s processi
- Designe individu reposito

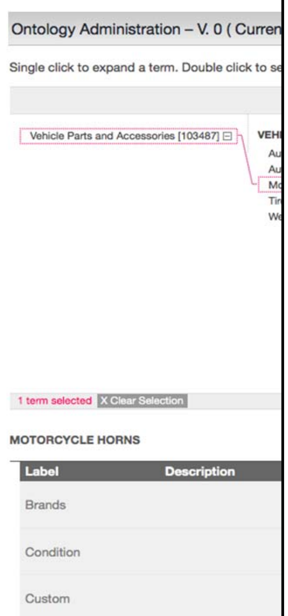


Design W Support

- VF provide tools for pr discussion
- Designers with Electr

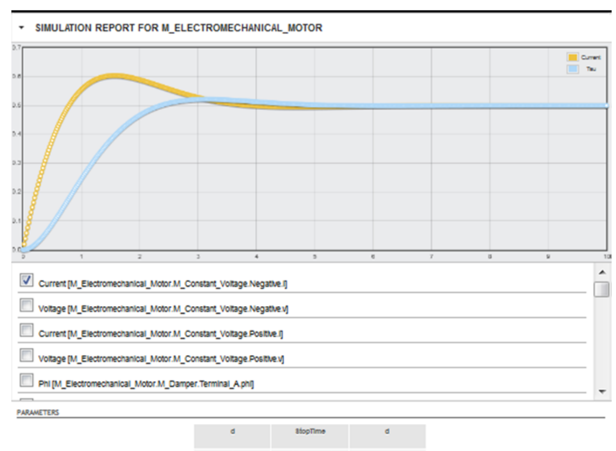
Ontology Administ

- Version-c definition
- Exportabl for tools



Continuous Integration Platform

- Cloud-based analysis job scheduling
- Presentation using built-in and custom web visualizers



Main Research Areas

- Modeling and model integration
  - Heterogeneity of physics and abstraction layers
  - Modeling language design
  - Modeling languages and semantics
- Design space construction and exploration
  - “Good architectures” as parameterized templates
  - Characterization for adaptability; decoupling techniques
  - Layered exploration algorithms
  - V&V tools and Probabilistic Certificate of Correctness (PCC)
- Collaboration platform for design
- Composition platforms
  - TTA, DDS, others...

Main Research Areas

- **Modeling and model integration**
  - **Heterogeneity of physics and abstraction layers**
  - **Modeling language design**
  - **Modeling languages and semantics**
- Design space construction and exploration
  - “Good architectures” as parameterized templates
  - Characterization for adaptability; decoupling techniques
  - Rapid exploration algorithms
  - V&V tools and Probabilistic Certificate of Correctness (PCC)
- Collaboration platform for design
- Composition platforms
  - TTA, DDS, others...

Overview

- Model-Based Design for CPS in META
  - Design flow and design infrastructure
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Thoughts on Resilience

Model-Based Design

Domain Specific Design Automation Environments:

- *Automotive*
- *Avionics*
- *Sensors…*

Tools:

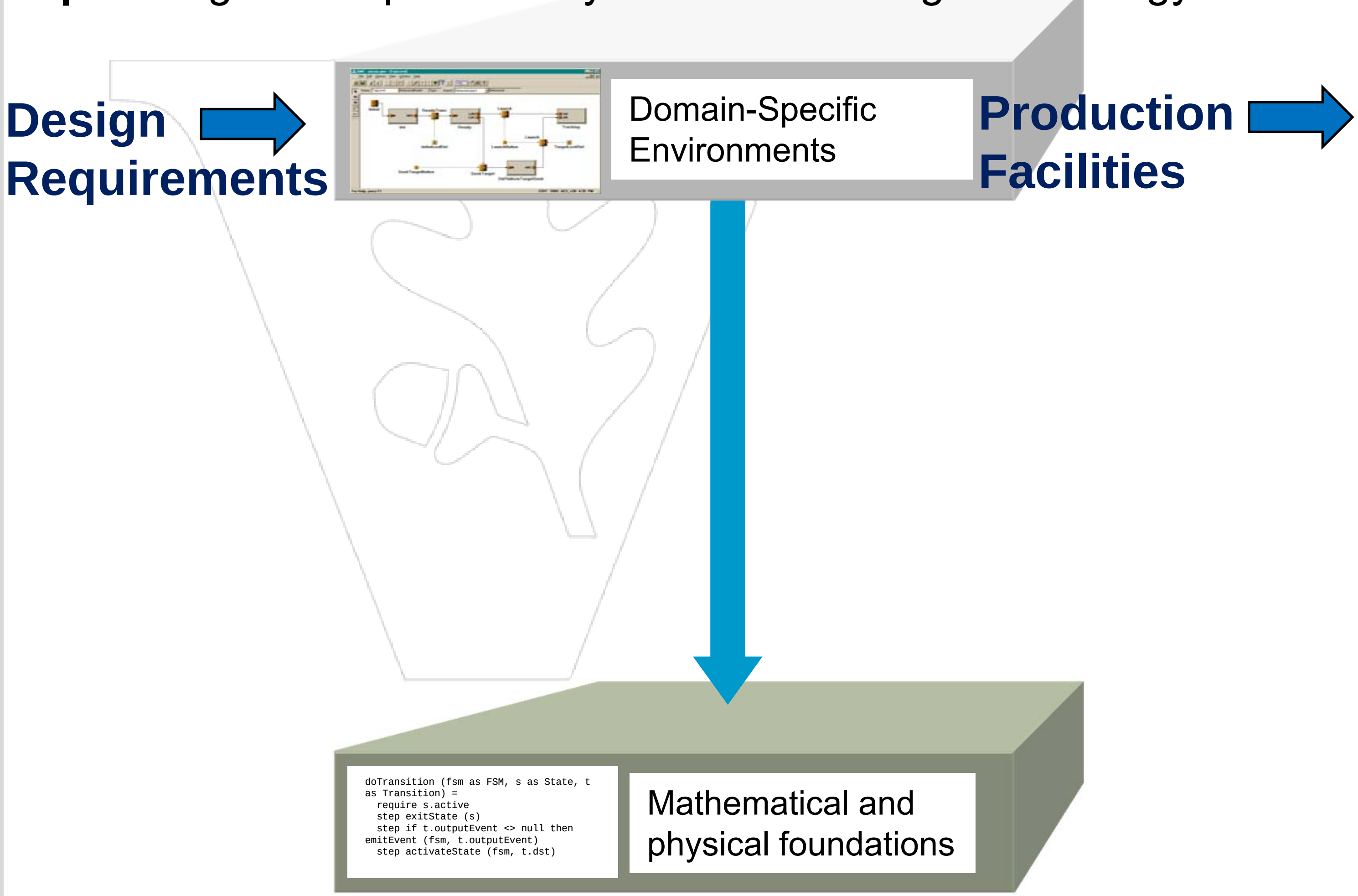
- *Modeling*
- *Analysis*
- *Verification*
- *Synthesis*

Challenges:

- *Cost of tools*
- *Benefit only narrow domains*
- *Islands of Automation*

Key Idea: Use models in domain-specific design flows and ensure that final design models are rich enough to enable production of artifacts with sufficiently predictable properties.

Impact: significant productivity increase in design technology



Metaprogrammable Design Tools

Domain Specific Design Automation Environments:

- *Automotive*
- *Avionics*
- *Sensors…*

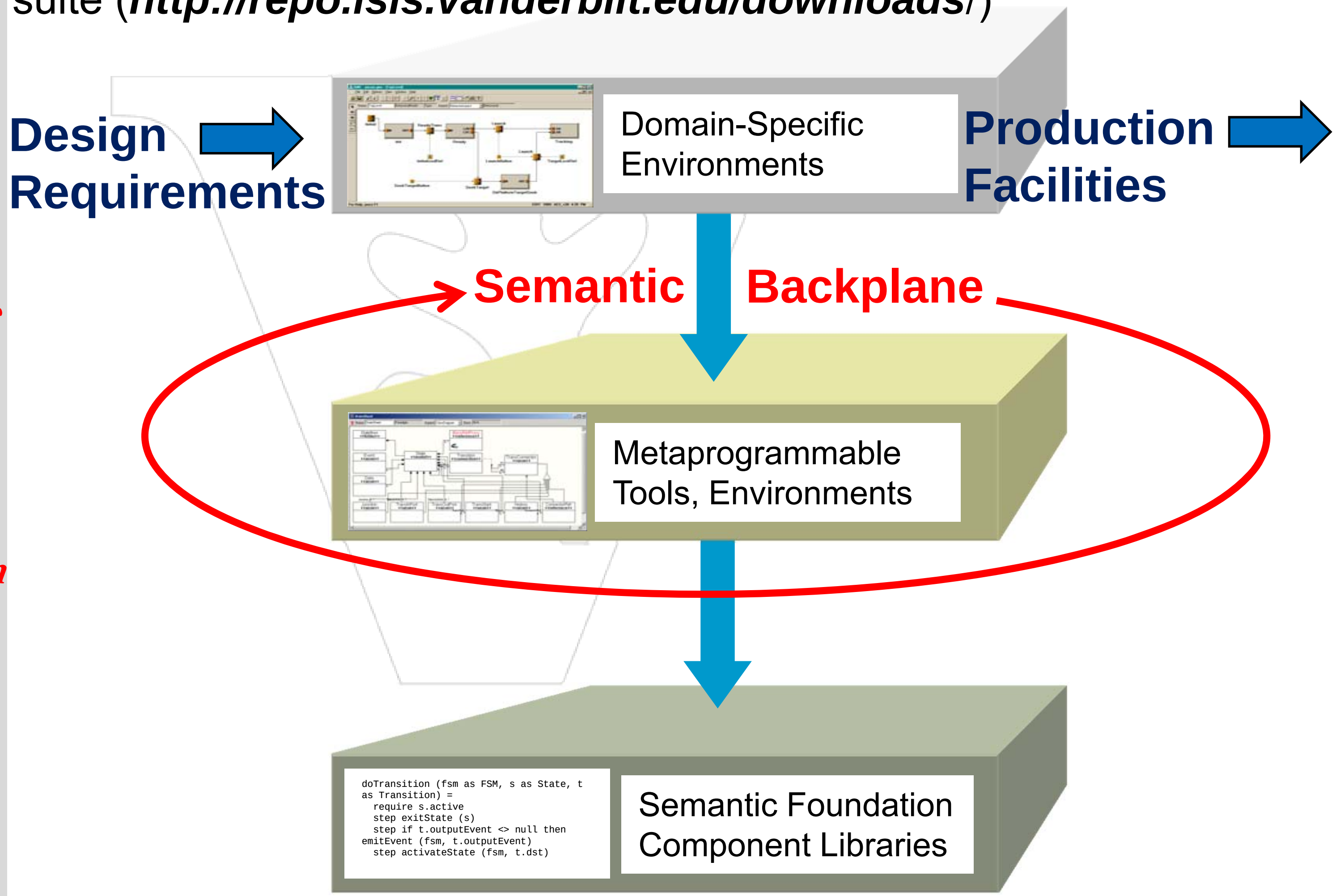
Metaprogrammable Tool Infrastructure

- *Model Building*
- *Model Transf.*
- *Model Mgmt.*
- *Tool Integration*

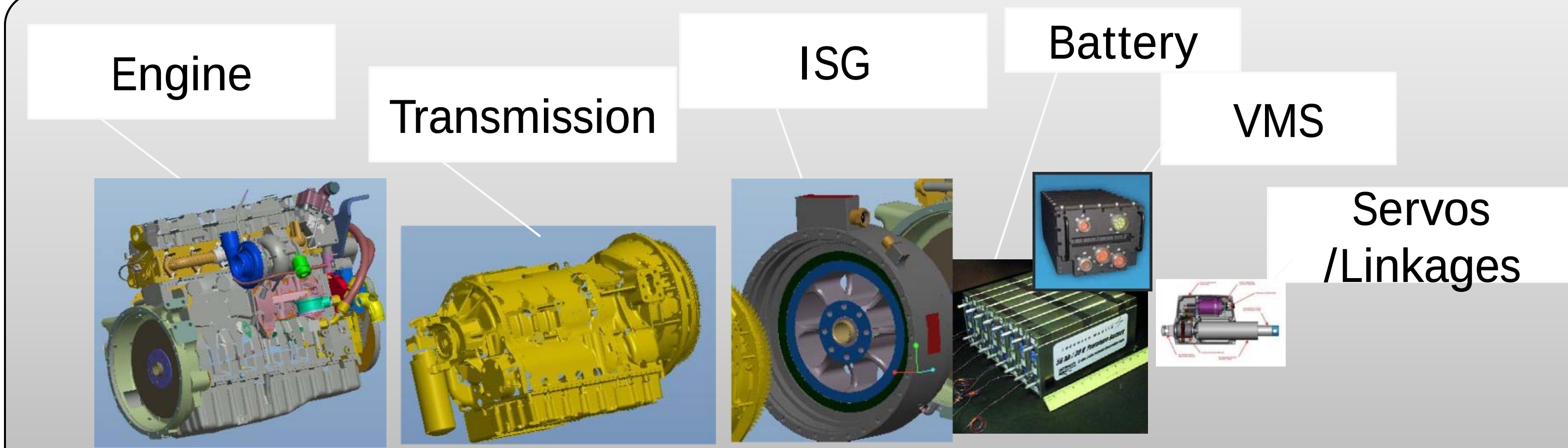
Explicit Semantic Foundation

- *Structural*
- *Behavioral*

Key Idea: Ensure reuse of high-value tools in domain-specific design flows by introducing a metaprogrammable tool infrastructure.
VU-ISIS implementation: Model Integrated Computing (MIC) tool suite (***http://repo.isis.vanderbilt.edu/downloads/***)



Components of a CPS



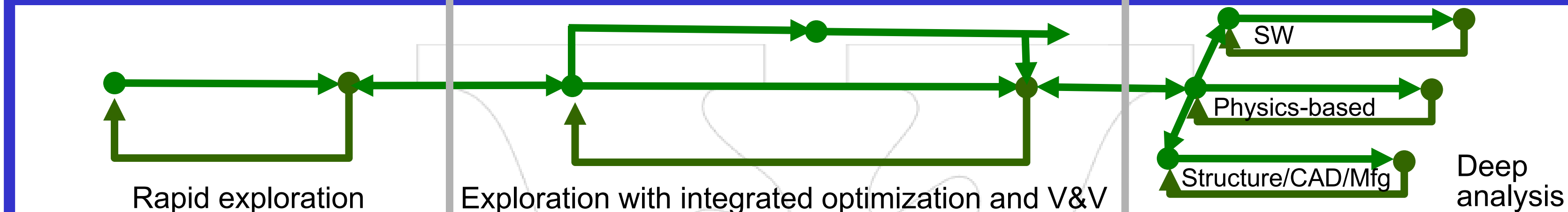
Components span:

- Multiple physics
- Multiple domains
- Multiple tools

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

CPS Design Flow Requires Model Integration

Architecture Design		Integrated Multi-physics/Cyber Design			Detailed Design	
Modeling	Exploration	Modeling	Simulation	V&V	Modeling	Analysis



Architecture Design

- Design Space + Constraint Modeling
- Architecture Modeling
- Static Component Modeling (multiphysics)

Integrated Multi-physics/Cyber Design

- Design Space + Behavioral Constraint Modeling
- Architecture Modeling
- Dynamics Modeling (multiple abstractions and multiphysics)
- CAD/assembly modeling
- Coarse Manufacturing Constraint Modeling

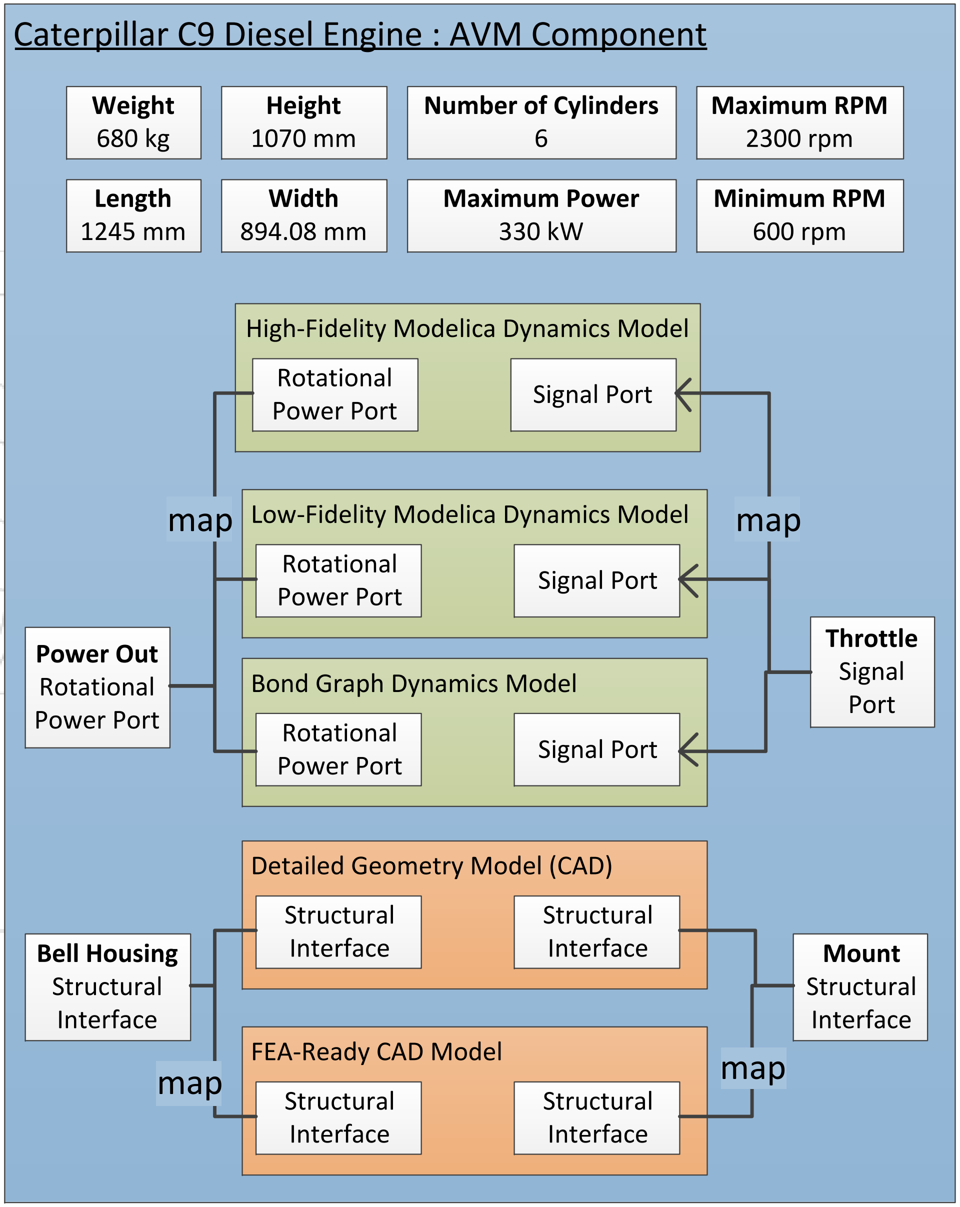
Detailed Design

- Architecture Modeling
- Detailed Domain Modeling
  - CAD
  - FEA; thermal, fluid…
  - Surrogate gen.
- Detailed Mfg. modeling
- RT SW modeling

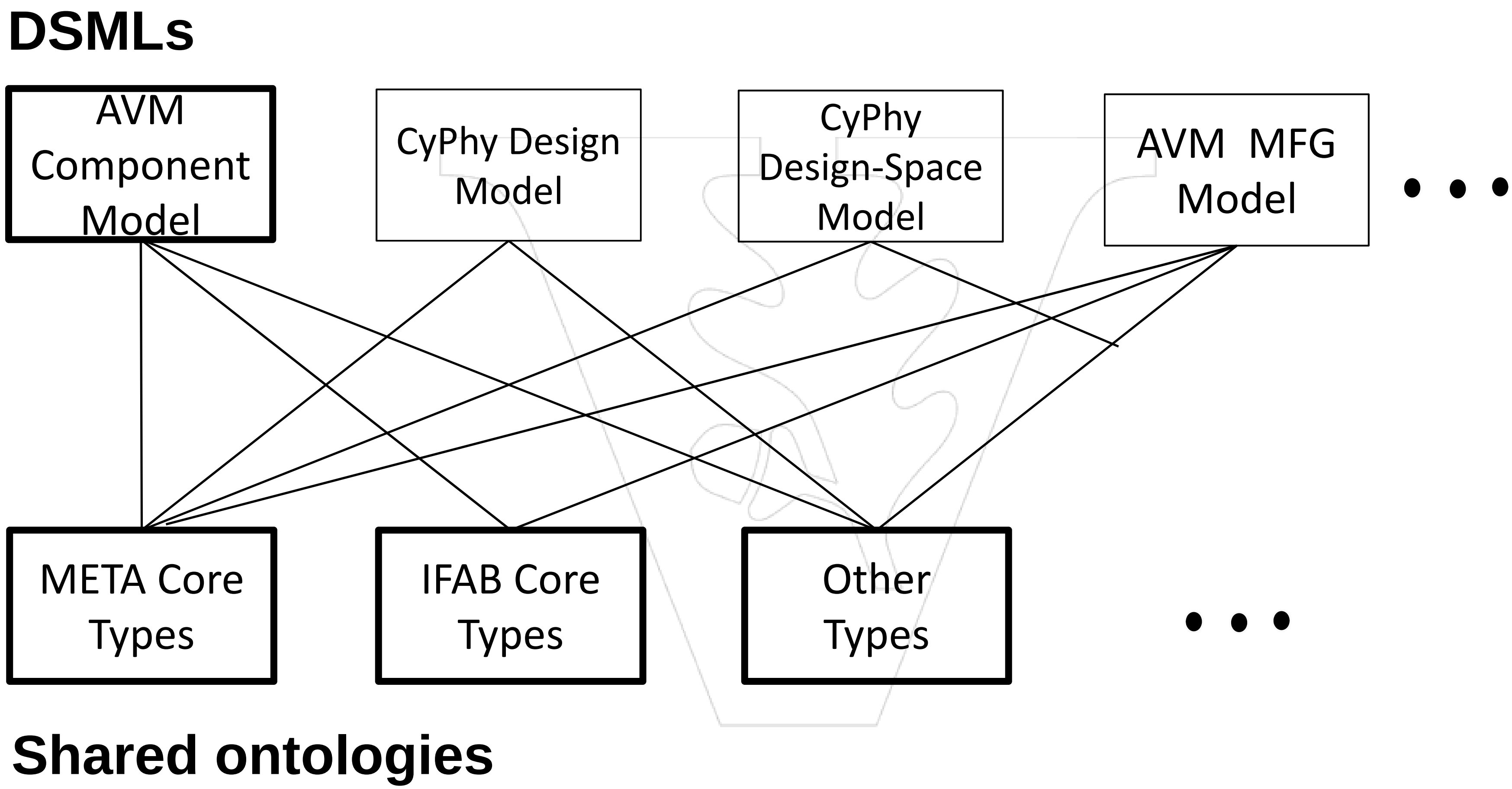
Domain Specific Modeling Languages

AVM Components

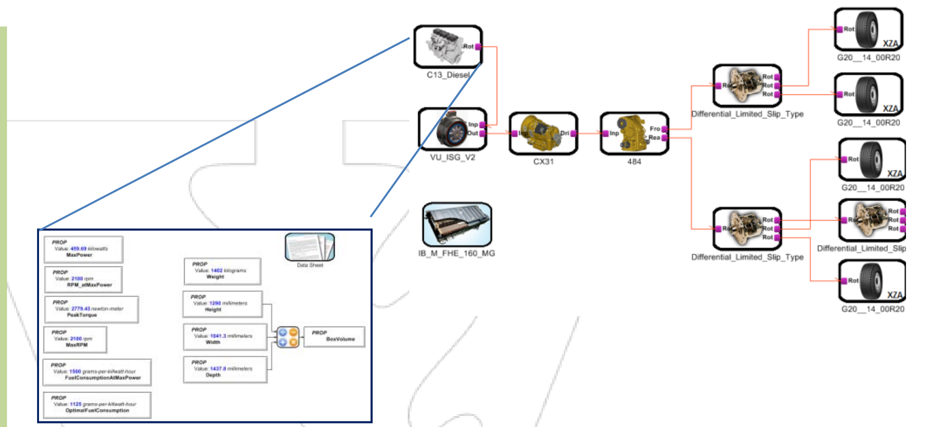
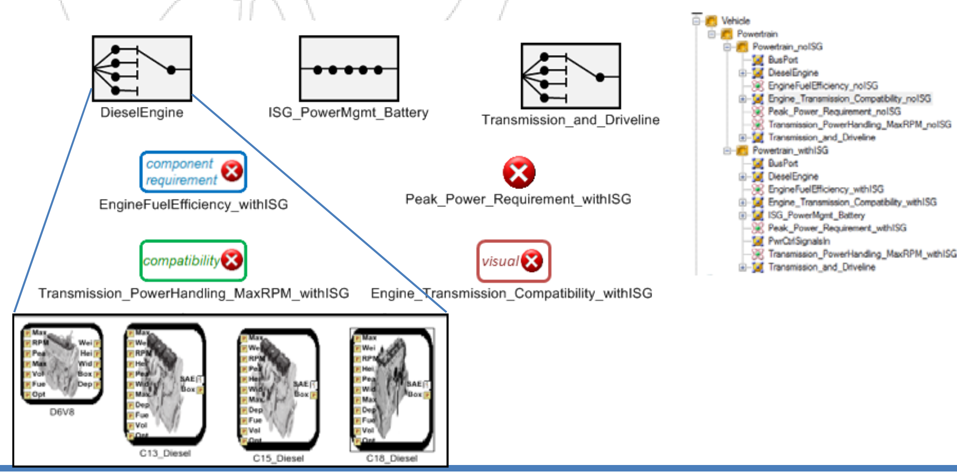
- An **AVM Component** is a well-defined, self-contained entity that can be used in a design.
- Defines the key **Properties** of the component.
- Serves as a wrapper for aggregating **detailed domain models** of the component.
  - Models can vary in level of fidelity and purpose
  - Each model independently represents the component behavior, geometry, etc.
- Aggregates the individual interfaces of these models into a **single set of component interfaces**.



Ontologies and DSMLs



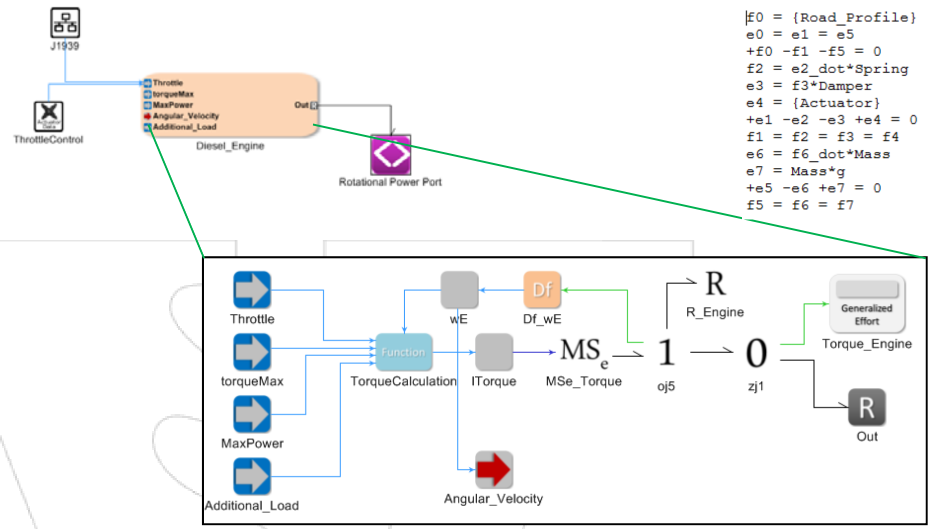
Example: Architecture Modeling

<table>
<tr><th>Sublanguage / Capability</th><th>Formalism, Language Constructs, Examples</th><th>Usage</th></tr>
<tr><td>Architecture Modeling</td><td>Hierarchical Module Interconnect
- Components
- Interfaces
- Interconnects
- Parameters
- Properties

</td><td>Systems Architect
- Explore Design Space
- Derive Candidate Designs</td></tr>
<tr><td>Design Space Modeling</td><td>Hierarchically Layered Parametric Alternatives
- Alternatives/ Options
- Parameters
- Constraints

</td><td>Systems Architect
- Define Design Space
- Define Constraint</td></tr>
</table>

Example: Dynamics Modeling

Physical Dynamics Modeling

Hybrid Bond Graphs
- Efforts, Flows,
- Sources, Capacitance, Inductance,
- Resistance,
- Transformers Gyrators,

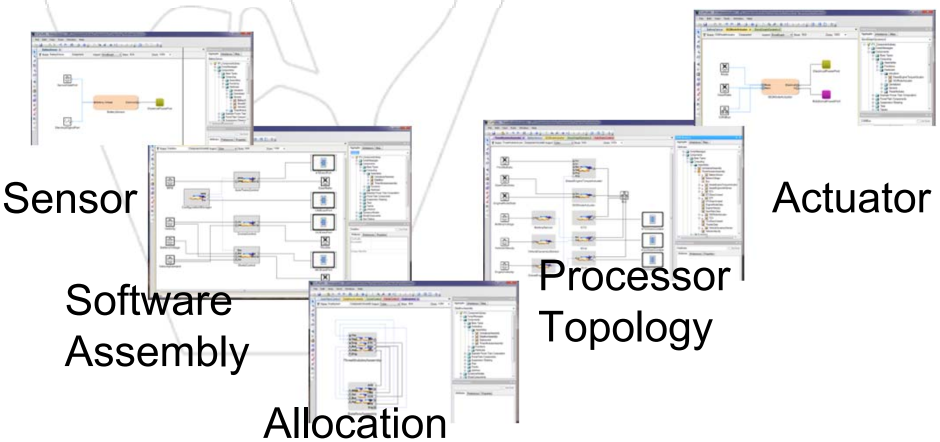


Component Engineer
- model dynamics with Hybrid Bond Graphs

System Engineers
- Compose system dynamics

Computational Dynamics Modeling

Dataflow + Stateflow + TT Schedule
- Interaction with Physical Components
- Cyber Components
- Processing Components



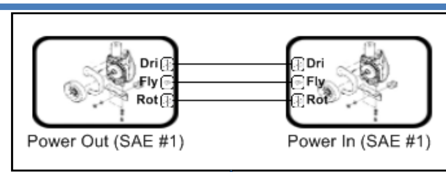
Domain Engineers
- design controller

System Engineers
- Processor allocate
- Platform Effects

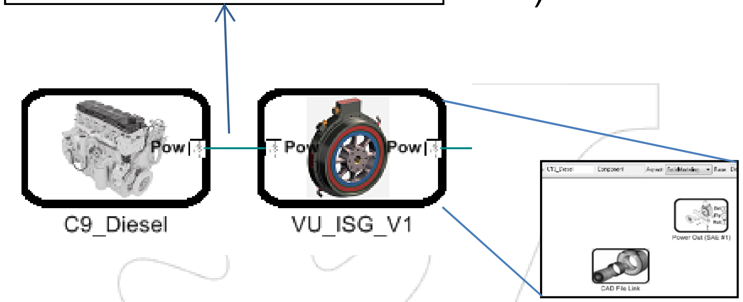
Example: Physical Structure and Manufacturing Modeling

Solid Modeling (CAD / Geometry)

Structural Interfaces
- Defined with Peer Roles:
  - Axis
  - Point
  - Surface
- CAD Links



Standard Structural Interfaces (ex: SAE #1)



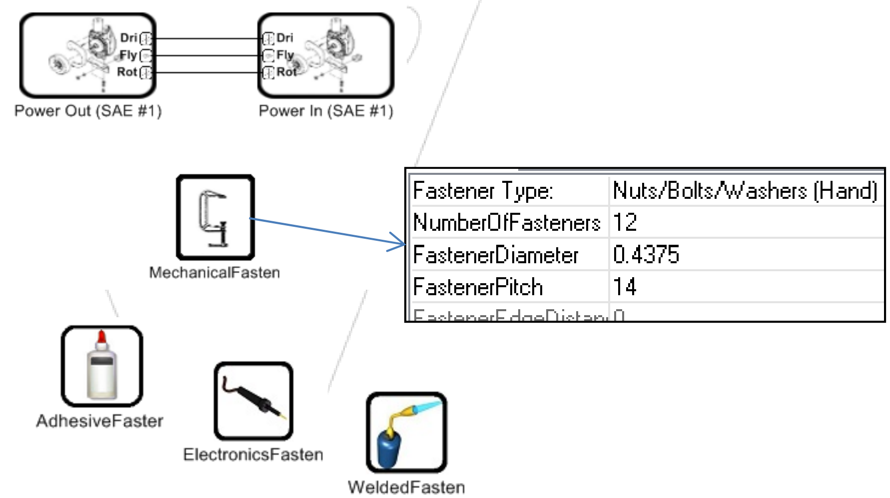
Component Engineer
- Defines Structural Interface

System Engineer
- Defines Architecture

Manufacturing Modeling

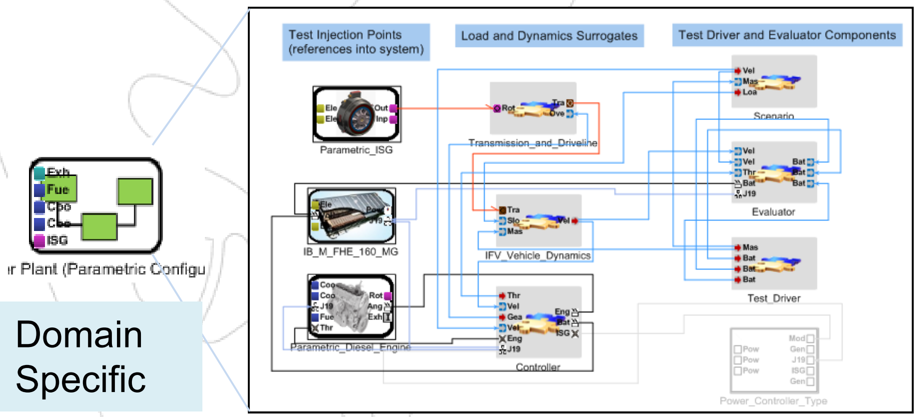
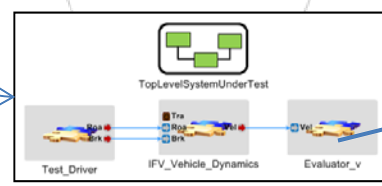
Component Manuf. Cost
- Make
  - Material
  - Fab Proc
  - Complxity
  - Shape/Wt
- OTS: Cost/unit

Structural Interfaces
- Fastener Types, Num# …

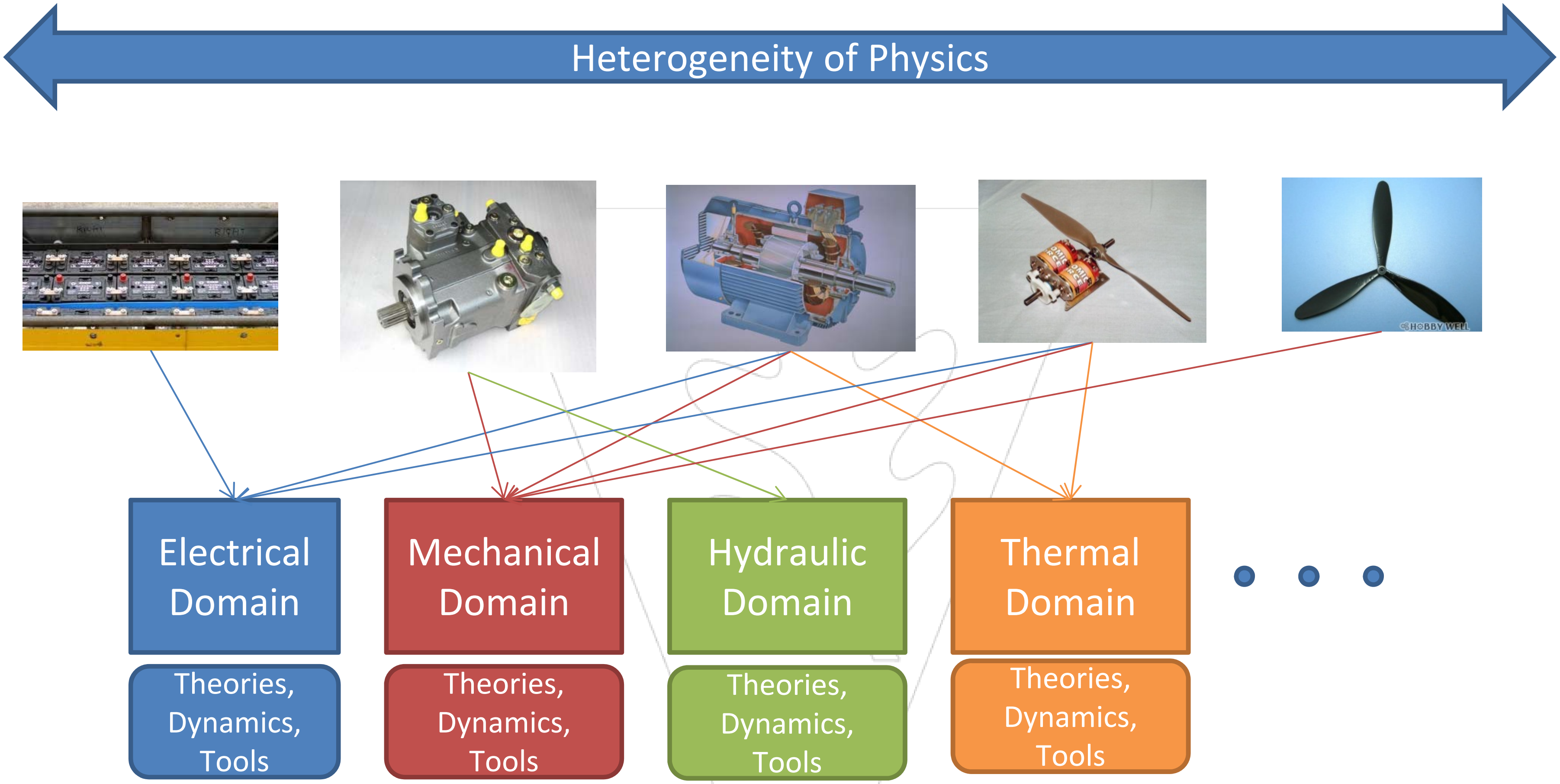


Component Engineer
- Defines Part Cost
- Defines Structural Interface, Fastener

Example: Requirement Modeling

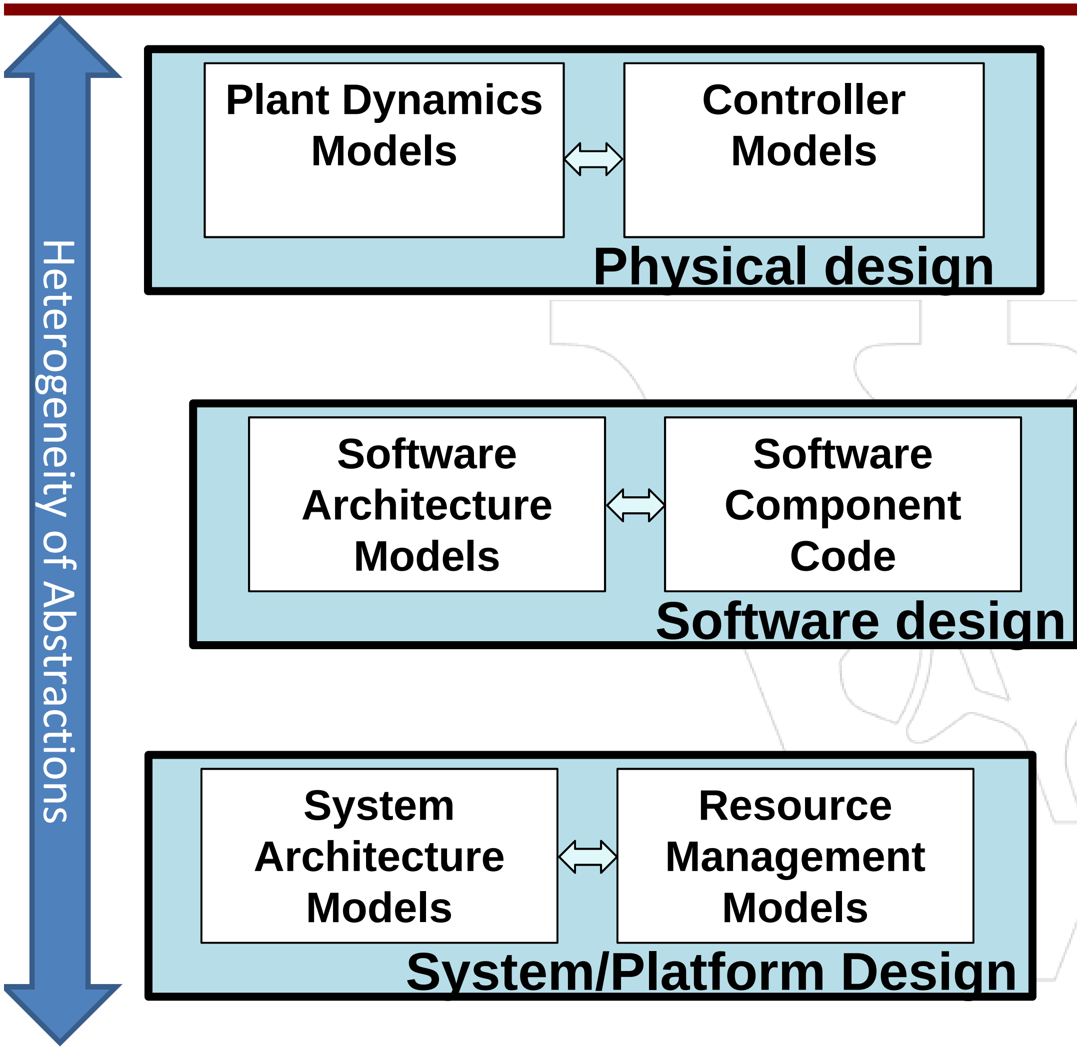
Requirements	DOORS SysML	1.2 | Soldiers.Warfighters | Carry a squad of 9 (6ft, crew-weight-table) | Infantry squad wearing personal Land Warrior 2 3.3 | acceleration | 0-30mph under 8 seconds | 0-20mph under 8 seconds | Derived from M1 Abrams	Customer
Executable Requirements (Constraints)	DESERT Constraint Language	constraint ForceOnGround_TracksArchitecture_withISG() { ((25000 + (self.parent().Weight() * 2.204623)) <= (17 * children("TrackPack").ContactPatch())) } MTBDD Solver	Systems Engineer - Requirement decomposition
Executable Requirements (Test Bench)	Domain-Specific Instantiations - Context - System Configuration - KPP Evaluation	 Domain Specific	Systems Engineer Domain Engineer - Requirement decomposition
Executable Requirements (Verification Test Bench)	- Context - Statistical Parameters - Temporal Monitors	Bayesian Statistical Model Checker (Clarke et al)  MFOTL ~F[0,5.5s](f1 <= 0 && f2 <= 180)	Verification Engineer - Requirement decomposition

Model Integration Challenge: Physics



Physical components are involved in multiple physical interactions (multi-physics)
Challenge: How to compose multi-models for heterogeneous physical components

Model Integration Challenge: Abstraction Layers



Dynamics: $B(t) = \kappa_p(B_1(t), \ldots, B_j(t))$

- *Properties*: stability, safety, performance
- *Abstractions*: continuous time, functions, signals, flows,…

Software : $B(i) = \kappa_c(B_1(i), \ldots, B_k(i))$

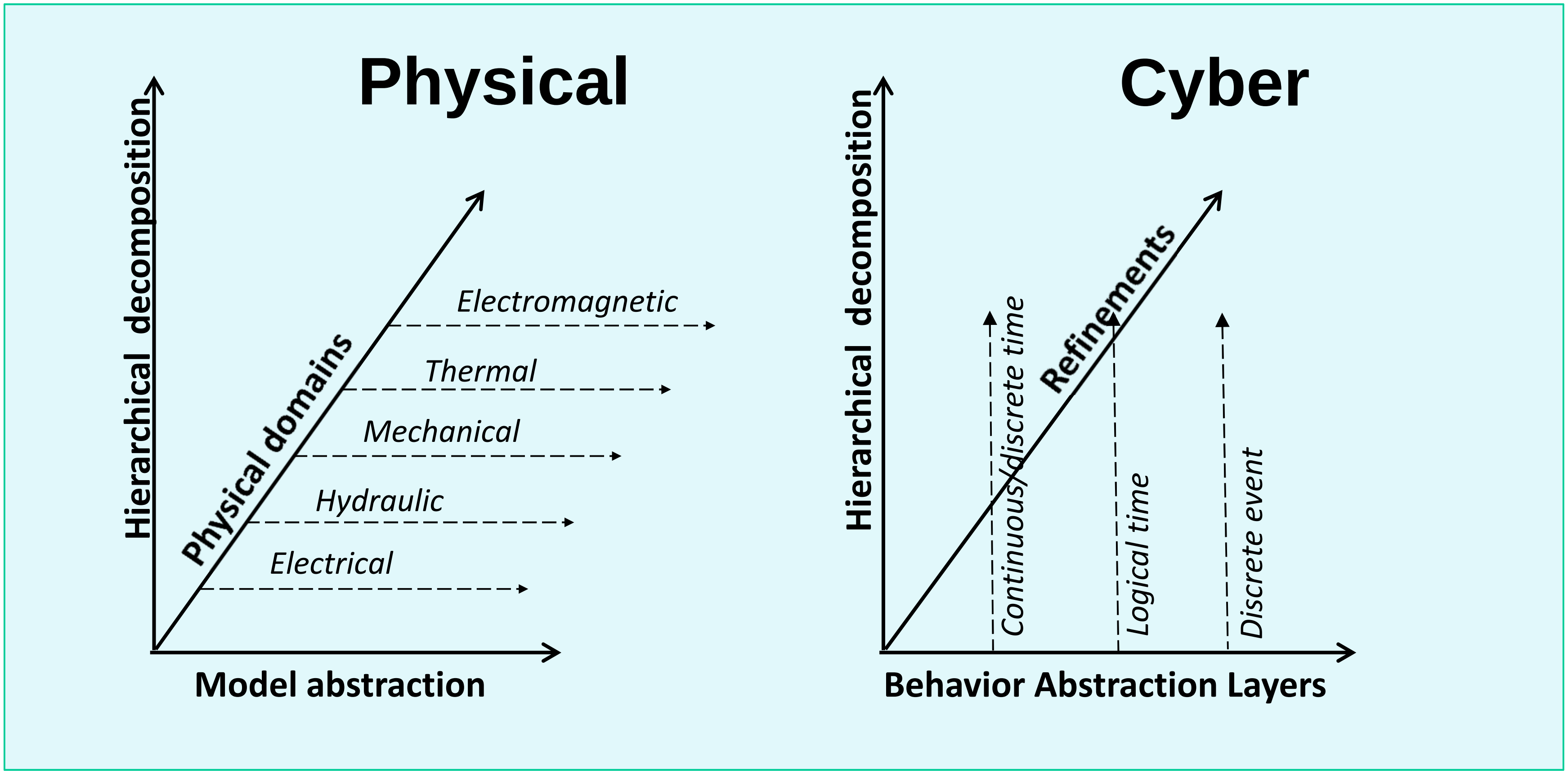
- *Properties*: deadlock, invariants, security,…
- *Abstractions*: logical-time, concurrency, atomicity, ideal communication,..

Systems : $B(t_j) = \kappa_p(B_1(t_i), \ldots, B_k(t_i))$

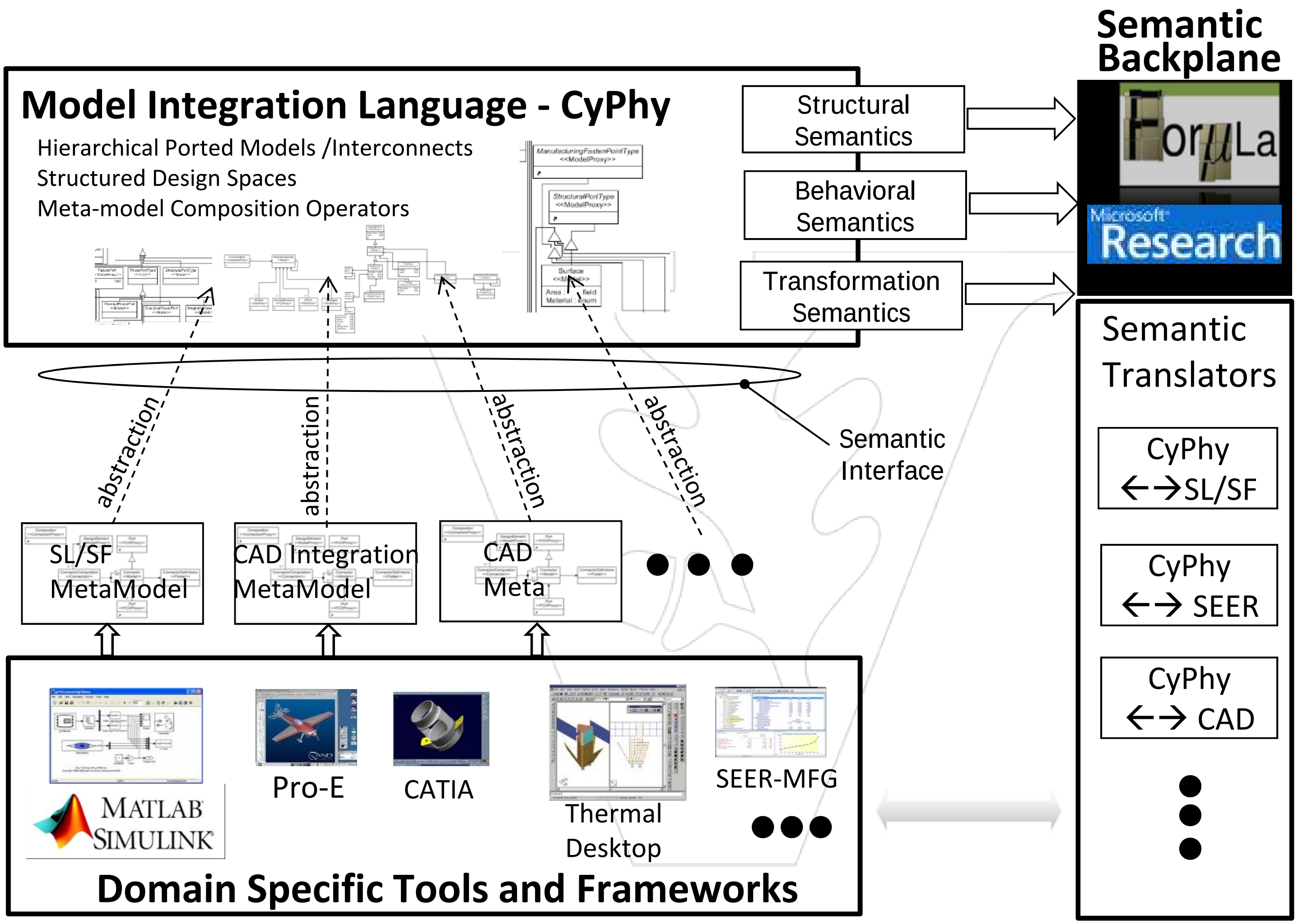
- *Properties*: timing, power, security, fault tolerance
- *Abstractions*: discrete-time, delays, resources, scheduling,

Cyber-physical components are modeled using multiple abstraction layers
Challenge: How to compose abstraction layers in heterogeneous CPS components?

Dimensions of Composition in CPS



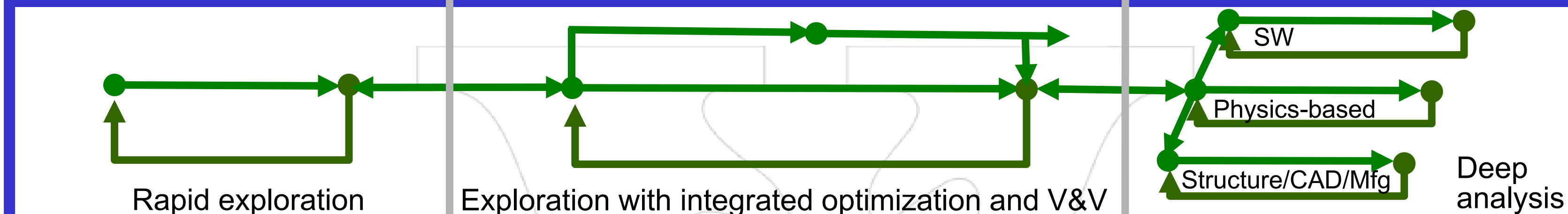
Case for Model Integration Languages…



Impact: Open Language Engineering Environment → Adaptability of Process/Design Flow → Accommodate New Tools/Frameworks , Accommodate New Languages

Example for Model Integration

Architecture Design	Integrated Multi-physics/Cyber Design	Detailed Design
Modeling · Exploration	Modeling · Simulation · V&V	Modeling · Analysis

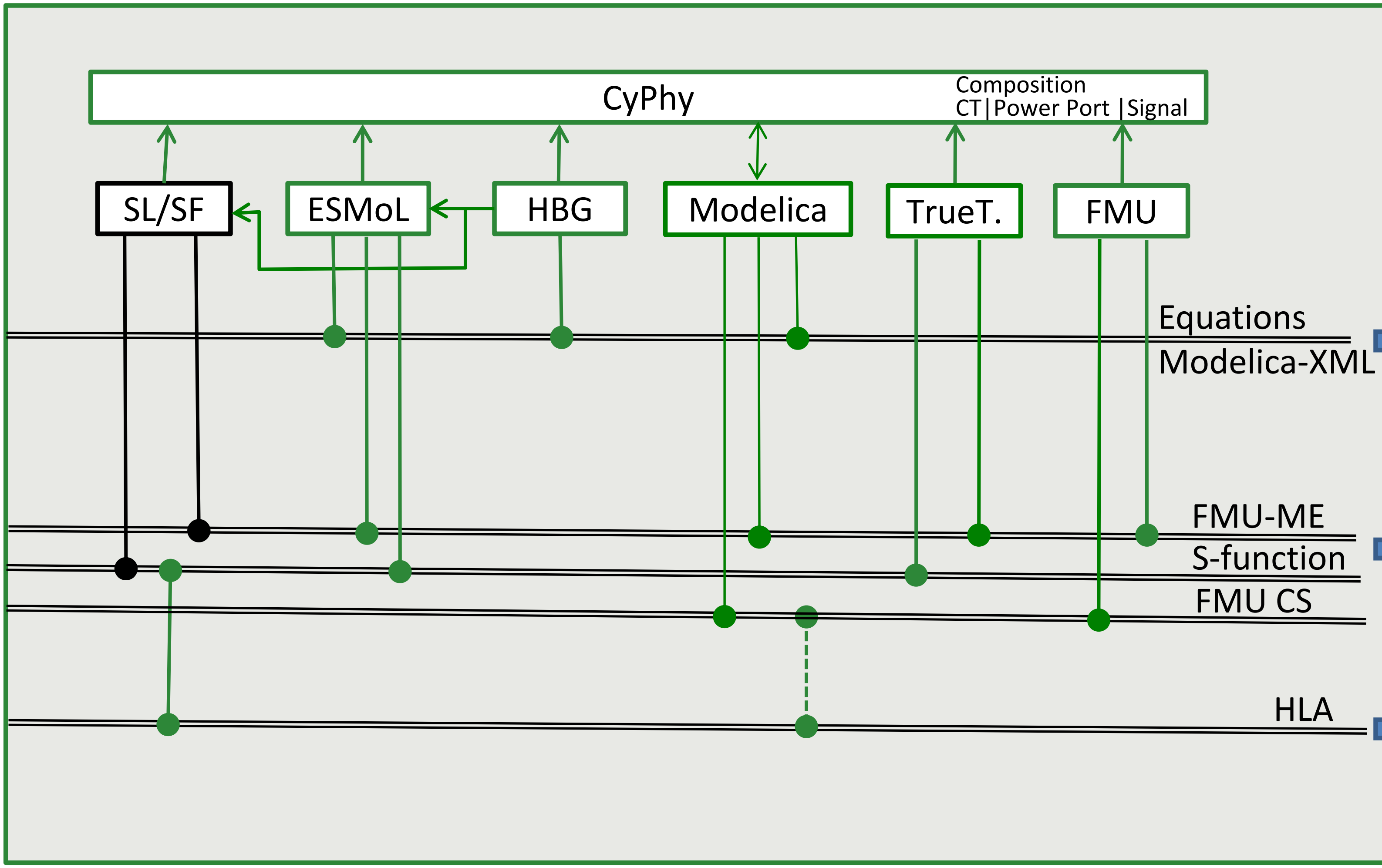


Architecture Design	Integrated Multi-physics/Cyber Design	Detailed Design

• Design Space + Constraint Modeling
• Architecture Modeling
• Low-Res Component Modeling
	• Design Space + Constraint Modeling
	• **Architecture Modeling**
	• **Dynamics Modeling**
	• Computational Behavior Modeling
	• CAD/Thermal Modeling
	• Manufacturing Modeling
		• Architecture Modeling
		• Detailed Domain Modeling
		  – CAD
		  – FEA; thermal, fluid…
		  – Surrogate gen.
		• Mfg. modeling
		• RT SW modeling

Domain Specific Modeling Languages

Example: Model Integration for Lumped Parameter Dynamics

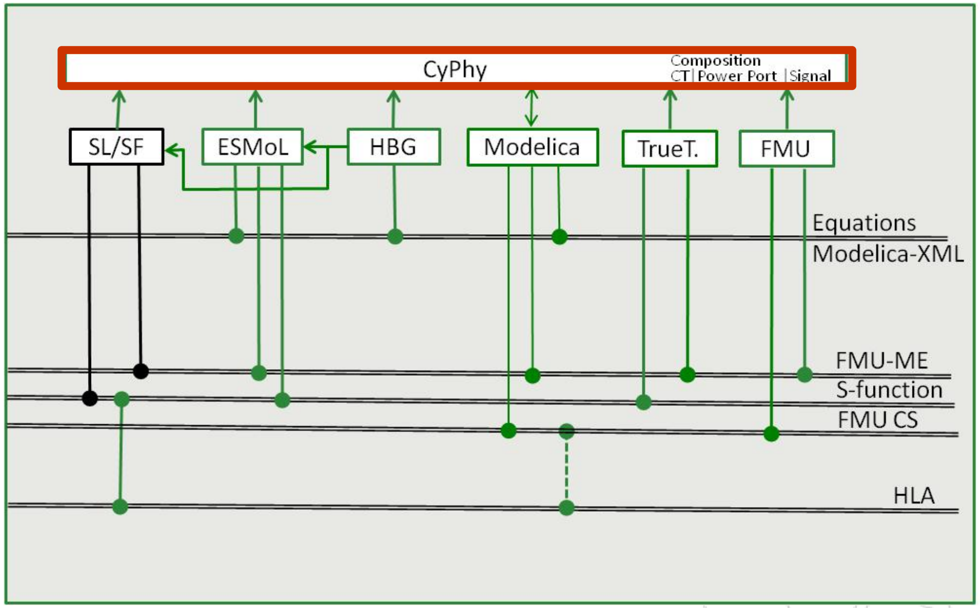


Verification Tools
- QR
- RA
- Analysis

Multimodel Simulation Tools

Distributed Multimodel Simulation
- NS3
- OMNET
- Delta-3D
- CPN

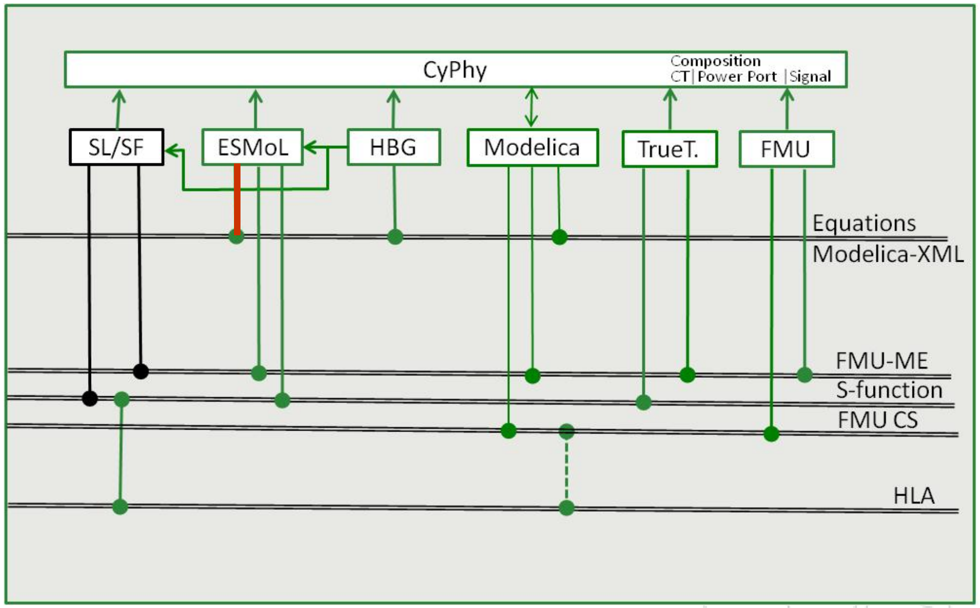
CyPhy Component and Design Interchanges



Specification obligations

- **CyPhy structural semantics**
  - Component Interchange Model
  - Design Interchange Model

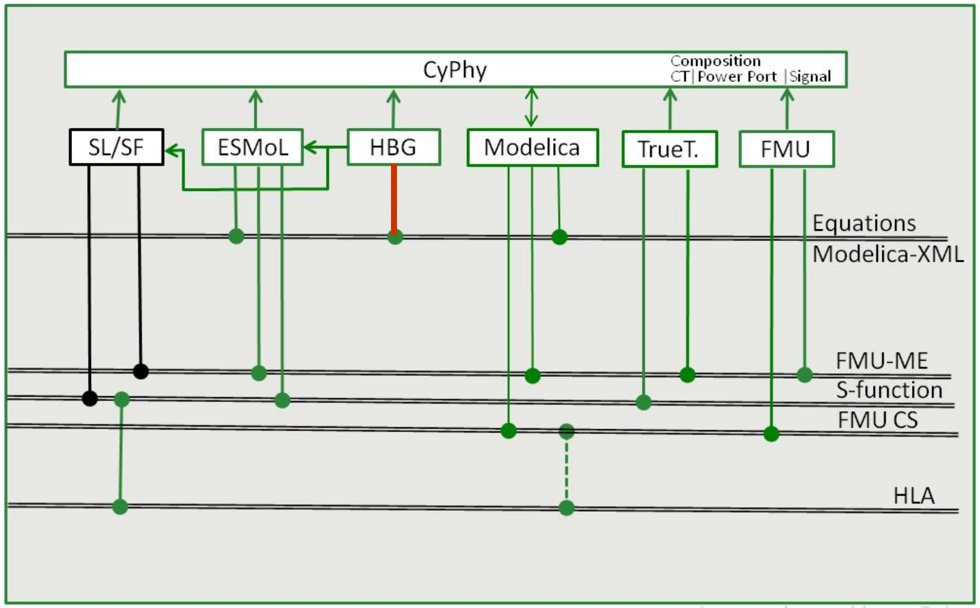
ESMoL Denotational Semantics



Specification obligations

- **CyPhy structural semantics**
- **ESMoL denotational semantics**
  - Signal Flow
  - Statechart

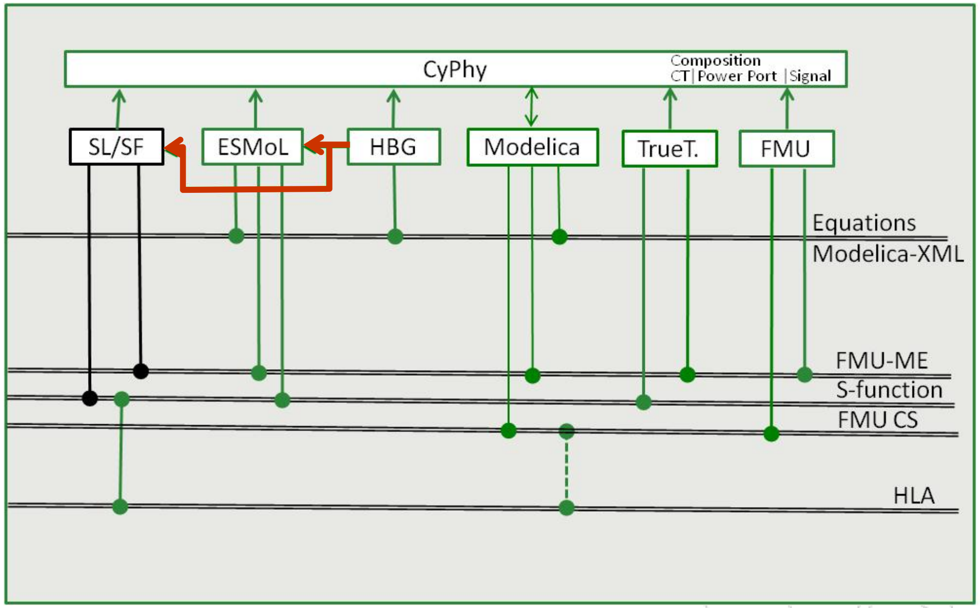
Hybrid Bond Graph Denotational Semantics



Specification obligations

- **CyPhy structural semantics**
- **ESMoL denotational semantics**
- HBG denotational semantics

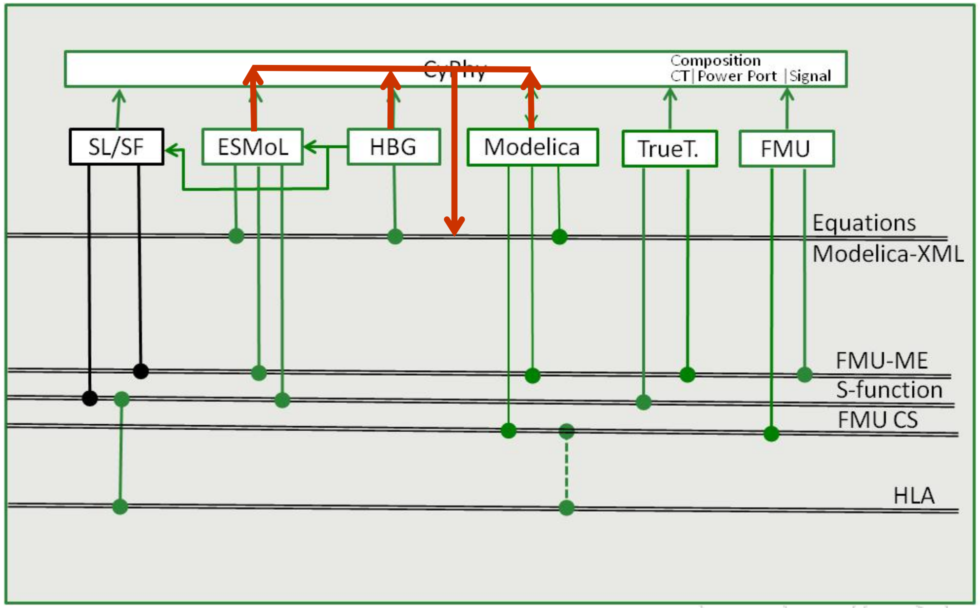
HBG Behavioral Semantics



Specification obligations

- **CyPhy structural semantics**
- **ESMoL denotational semantics**
- HBG denotational semantics
- HBG behavioral semantics

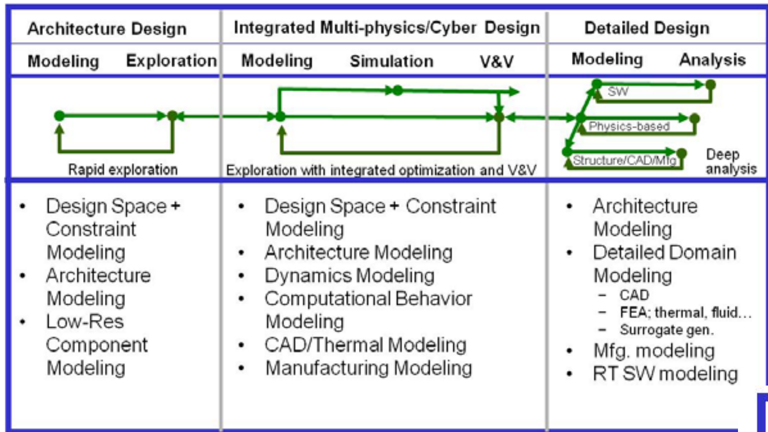
CyPhy Denotational Semantics for Composition (Design Interchange)



Specification obligations

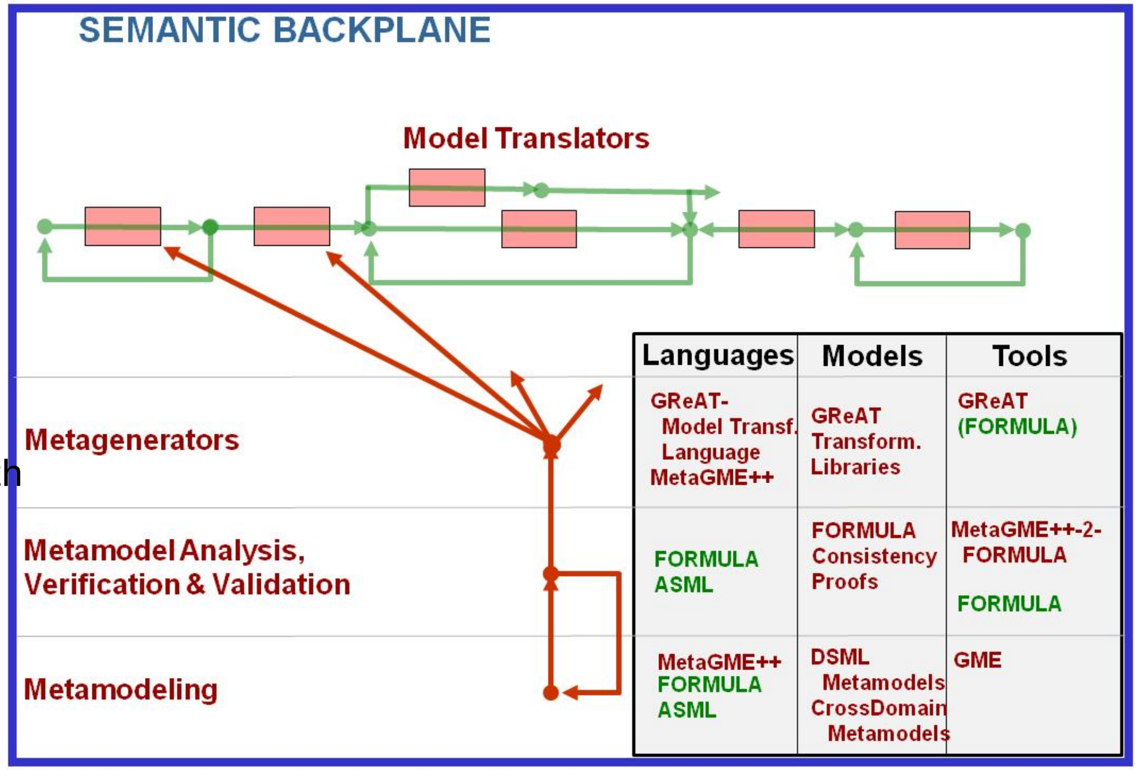
- **CyPhy structural semantics**
- **ESMoL denotational semantics**
- HBG denotational semantics
- HBG transformational semantics
- **CyPhy denotational composition semantics**

Cost of Model Integration Languages: "Semantic Backplane"



- Tight integration from architecture modeling to physics-based modeling
- Integrated multi-physics modeling
- Bridging gap between computation and physics-based domains
- Tight integration of structural and behavioral models
- Emphasis is on automation and scaling
- *The META tool suite must be designed for rapid evolution*

- Agility is achieved by introducing a ***Semantic Backplane***
- The Semantic Backplane is implemented by:
  - tools and methods for modeling language specification, validation and transformation
  - tools and methods for explicit representation of and computation with structural and behavioral semantics
  - metamodel and transformation libraries
  - metaprogrammable tools



Languages	Models	Tools
GReAT- Model Transf. Language MetaGME++	GReAT Transform. Libraries	GReAT (FORMULA)
FORMULA ASML	FORMULA Consistency Proofs	MetaGME++-2-FORMULA FORMULA
MetaGME++ FORMULA ASML	DSML Metamodels CrossDomain Metamodels	GME

FORMULA: http://research.microsoft.com/formula

Overview

- Model-Based Design for CPS in META
  - Design flow and design infrastructure
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Thoughts on Resilience

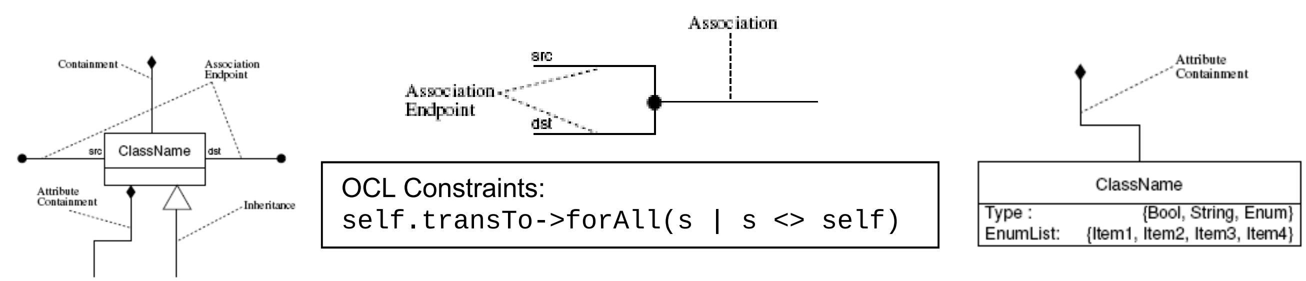
Specification of Domain-Specific Modeling Languages

Abstract syntax of DSML-s are defined by metamodels.

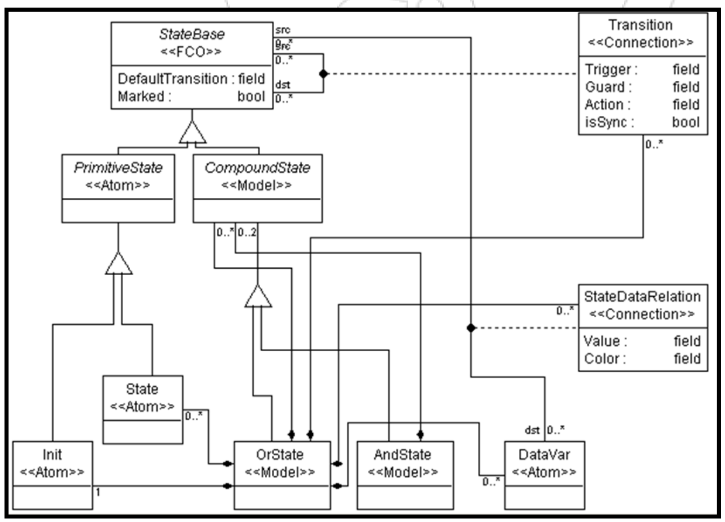
A metamodeling language is one of the DSML-s.

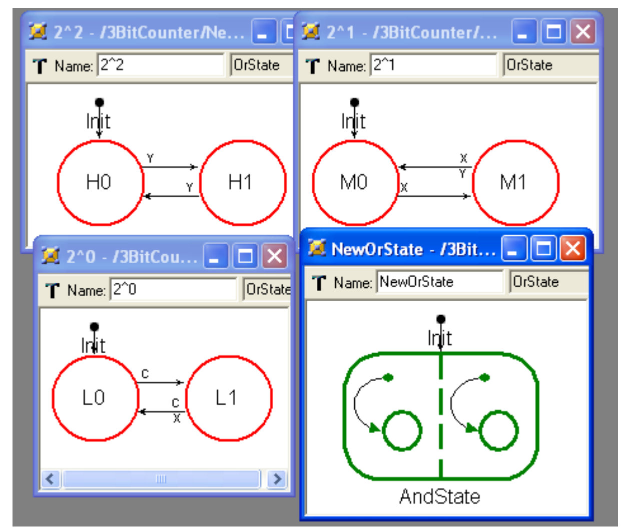
Semantics of metamodeling languages: structural semantics.

<u>Key Concept</u>: Modeling languages define a set *of well- formed models* and their *interpretations*. The interpretations are mappings from one domain to another domain.



Basic metamodeling notation: UML Class Diagram + OCL





MetaGME metamodel of simple statecharts

Model-editor generated from metamodel

Formalization of Structural Semantics

$L = \langle Y, R_Y, C, ([\ ]_i)_{i \in J} \rangle$
$D(Y,C) = \{ r \in R_Y \mid r \models C \}$
$[\ ] : R_Y \mapsto R_{Y'}$

Y: set of concepts,
R_Y: set of possible model realizations
C: set of constraints over R_Y
$D(Y,C)$: domain of well-formed models
[ ]: interpretations

Jackson & Sz. '2007
Jackson, Schulte, Sz. '2008
Jackson & Sz. '2009

Key Concept: DSML syntax is understood as a constraint system that identifies behaviorally meaningful models. ***Structural semantics provides mathematical formalism for interpreting models as well-formed structures*.**

Structural Semantics defines modeling domains using Algebraic Data Types and First-Order Logic with Fixpoints. Semantics is specified by Constraint Logic Programming.

Use of structural semantics:

- Conformance testing: $x \in D$
- Non-emptiness checking: $D(Y,C) \neq \{nil\}$
- DSML composing: $D_1 * D_2 | D_1 + D_2 | D' \ includes\ D$
- Model finding: $S = \{ s \in D | s | = P \}$
- Transforming: $m' = T(m); m' \in X; m \in Y$

Microsoft Research Tool: FORMULA

- Fragment of LP is equivalent to full first-order logic
- Provide semantic domain for model transformations.

Behavioral Semantics

- Given a DSML

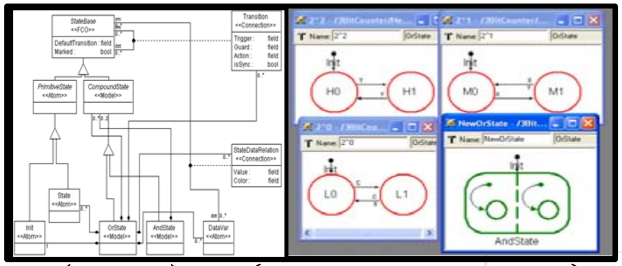
$$L = \langle Y, R_Y, C, ([\ ]_i)_{i \in J} \rangle$$

$$D(Y, C) = \{ r \in R_Y \mid r \models C \}$$

$$[\ ] : R_Y \mapsto R_{Y'}$$

- Behavioral semantics will be defined by specifying the transformation between the DSML and a modeling language with behavioral semantics.

Explicit Methods for Specifying Behavioral Semantics 1/2

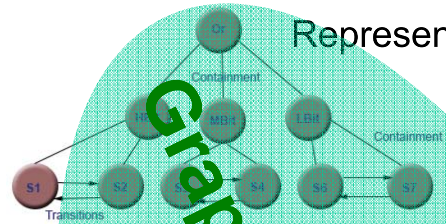


$$D(Y,C)=\{r\in R_Y \mid r \models C\}$$

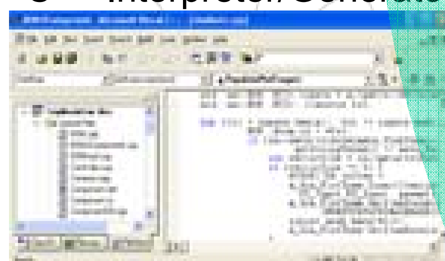
$$[\;]: R_Y \mapsto R_{Y'}$$

$$D(Y',C')=\{r\in R_{Y'} \mid r \models C'\}$$
$$[\;]: R_{Y'} \mapsto R_{Y''}$$

Representation as AST

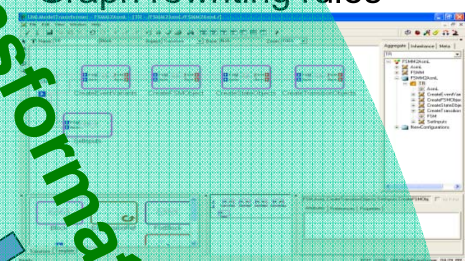


C++ Interpreter/Generator



Explicit

Graph rewriting rules



Graph transformation semantics

Executable Model (Simulators)

Executable Code

Executable Specification

Heterogeneous math domain;
Operational semantics

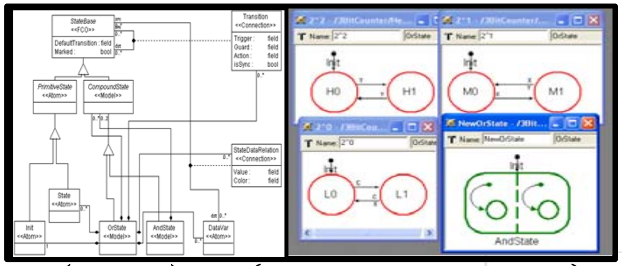
Reasonable tool support;
Easy to understand

Convergence in Formal Framework: FORMULA

- History: Foundations for Embedded Systems ITR; Ethan Jackson at VU 2005-2008
- Microsoft Research (Bellevue & Aachen); Z3 Satisfiability Modulo Theory Solver (Z3); VS distribution
- *http://research.microsoft.com/formula*

- Foundation: Algebraic Data Types (ADT) and First-order logic with fixpoints (FPL)
- Parameterized with background theories (bit vectors, term algebras, etc.
- Semantics is defined by constraint logic programming (CLP)
- Evolving structures; temporal logic

Explicit Methods for Specifying Behavioral Semantics 2/2



$$D(Y,C)=\{r \in R_Y \mid r \models C\}$$

```
domain AcausalBG_elements
{
  primitive Sf ::= (id: String).
  primitive Se ::= (id: String).
  primitive R  ::= (id: String).
  //…
primitive TF ::= (id: String).
  primitive GY ::= (id: String).
  primitive ZeroJunction ::= (id: String).
  primitive OneJunction  ::= (id: String).
  Source   ::= Sf + Se.
  //..
}
```

$$[\ ]: R_Y \mapsto R_{Y'}$$

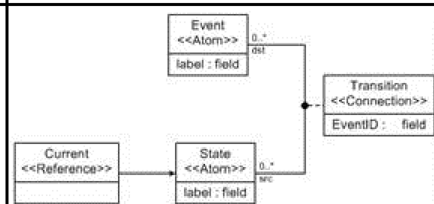
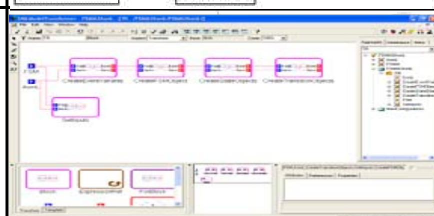
```
transform BG_DenotationalSemantics
  from in1::AcausalBG
  to out1::DAEquations
{
  Eq(e_a,p_x) :- x is Se, Src(a,x).
  Eq(f_a,p_x) :- x is Sf, Src(a,x).
  Eq(e_a, Mul(p_x,f_a)) :- x is R, Dst(a,x).
  DiffEq(e_a, Mul(Inv(p_x),f_a)) :-
    x is C, Dst(a,x).
//…
}
```

$$D(Y',C')=\{r \in R_{Y'} \mid r \models C'\}$$
$$[\ ]: R_{Y'} \mapsto R_{Y''}$$

```
domain DAEquations
{
  primitive Variable ::=
    (name: String, id: String).
  primitive Param ::= (id: String).
  primitive Neg ::= (Term).
  primitive Inv ::= (Term).
  //..
  Term ::= Variable + Param + Neg + Inv + Mul + Sum.
  primitive Eq ::= (Variable, Term).
  primitive DiffEq ::= (Variable, Term).
  primitive SumZero ::= (Sum).
  Equation ::= Eq + DiffEq + SumZero.
}
```

Single math framework

Current Work: Semantic Backplane

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

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

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

Semantic Specification Example: Structural Semantics for BGML

```
domain BondGraph
{
primitive Se ::= (id: String, effort: String).
primitive Sf ::= (id: String, flow: String).
primitive R  ::= (id: String, resistance: String).
primitive C  ::= (id: String, capacitance: String).
primitive I  ::= (id: String, inductance: String).
primitive TF ::= (id: String, modulus: String).
primitive GY ::= (id: String, gyrator_modulus: String).
primitive OneJunction ::= (id: String).
primitive ZeroJunction ::= (id: String).
Node ::= Se + Sf + R + C + I + TF + GY + OneJunction +
ZeroJunction.
// exactly one outgoing connection, and no incoming
badSe := se is Se, count(Connection(_, se, _)) != 1.
badSe := se is Se, count(Connection(_, _, se)) > 0.
```

```
partial model RC1 of ExtendedBondGraphWithCausality
{
JunctionState(oj1, "ON")

R1 is R("R1", "r1", "R1")
   C1 is C("C1", "c1i", "c1", "C1")
oj1 is OneJunction("oj1", "OFF", "", "", "oj1")
Se1 is Se("Se1", "se1", "Se1")
BondE2J("0", Se1, oj1, "0")
BondJ2E("1", oj1, C1, "1")
   BondJ2E("2", oj1, R1, "2")
}
```

```
domain ExtendedBondGraphWithCausality extends ExtendedBondGraph
{
[Closed(junction)][Function(junction -> State)]
primitive JunctionState ::= (junction: Junction, State:
Enum_BondGraphElements_InitialState).

CausalStrokePosition ::= { SRC, DST, OFF }.

[Closed(bond)][Function(bond -> causal_stroke)]
primitive CausalStrokeJ2J ::= (bond: BondJ2J, causal_stroke:
CausalStrokePosition).
[Closed(bond)][Function(bond -> causal_stroke)]
primitive CausalStrokeJ2E ::= (bond: BondJ2E, causal_stroke:
CausalStrokePosition).
[Closed(bond)][Function(bond -> causal_stroke)]
primitive CausalStrokeE2J ::= (bond: BondE2J, causal_stroke:
CausalStrokePosition).

CausalStroke ::= (A: BGNode, B: BGNode, stroke: CausalStrokePosition).
CausalStroke(A, B, stroke) :-
CausalStrokeJ2J(BondJ2J(_, A, B, _), stroke);
CausalStrokeJ2E(BondJ2E(_, A, B, _), stroke);
CausalStrokeE2J(BondE2J(_, A, B, _), stroke).
CausalStroke(A, B, SRC) :- CausalStroke(B, A, DST).
CausalStroke(A, B, DST) :- CausalStroke(B, A, SRC).
CausalStroke(A, B, OFF) :- CausalStroke(B, A, OFF).

// Description with positive rules
BondE ::= (e: BGNode, j: Junction).
BondE(x,y) :- BondJ2E(_,y,x,_); BondE2J(_,x,y,_).
BondJ ::= (j: Junction, je: BGNode).
BondJ(x,y) :- BondJ2E(_,x,y,_); BondE2J(_,y,x,_); BondJ2J(_,x,y,_); BondJ2J(_,y,x,_).
```

Semantic Specification Example: Denotational Semantics

```
domain DiffEquations
{
primitive Effort ::= (id: Natural). // effort
connection
primitive Flow ::= (id: Natural). // flow
connection
primitive Value ::= (id: String). // parameter
value
primitive Derivative ::= (term: Term). // dx
primitive Add ::= (term1: Term, term2: Term). //
+
primitive Mul ::= (term1: Term, term2: Term). // *
primitive Equals ::= (term1: Term, term2: Term).
// =
primitive Neg ::= (term: Term). // unary -

Term ::= Effort + Flow + Value + Derivative +
Add + Mul + Equals + Neg.
}
```

```
transform T from BondGraph as in1 to DiffEquations as out1
{
// Source of effort
out1.Equals(Effort(id), Value(effort)) :- Connection(id, Se(_,effort), _).

// Source of flow
out1.Equals(Flow(id), Value(flow)) :- Connection(id, sf, Sf(_,flow)).

// Flow = Capacitance * derivative(Effort)
out1.Equals(Flow(id), Mul(Value(capacitance), Derivative(Effort(id)))) :-
Connection(id, C(_, capacitance), _);
Connection(id, _, C(_, capacitance)).

// kcl
out1.Equals(Flow(id1), Flow(id2)) :- Succ(OneJunction(_), id1, id2).

// kvl
out1.Equals(Effort(id1), Effort(id2)) :- Succ(ZeroJunction(_), id1, id2).
}
```

BondGraph → Differential Equations

Example: Formal Operational Semantics for Finite Automata

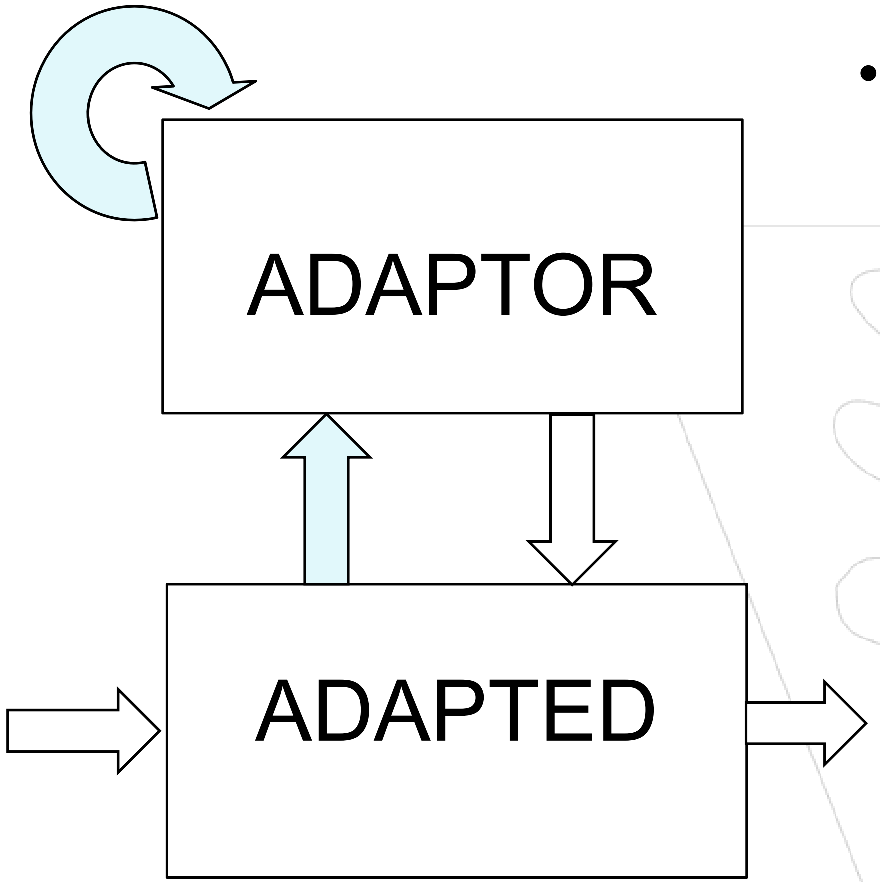
```
domain DFA {
  primitive Event ::= (lbl: Integer).
  primitive State ::= (lbl: Integer).
  primitive Transition ::= (src: State, trg: Event, dst: State).
  primitive Current    ::= (st: State).
  nonDeterTrans := Transition(s, e, sp), Transition(s, e, tp), sp != tp.
  conforms      := !nonDeterTrans.
}
```

```
transform Step<fire: in1.Event>  from in1::DFA  to out1::DFA
 {
  out1.State(x) :- in1.State(x).
  out1.Event(x) :- in1.Event(x).
  out1.Transition(s, e, sp) :- in1.Transition(s, e, sp).
  out1.Current(sp) :- in1.Current(s), in1.Transition(s, fire, sp).
  out1.Current(s) :- in1.Current(s),
  fail in1.Transition(s, fire, _).
 }
```

Overview

- Model-Based Design for CPS in META
  - Design flow and design infrastructure
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Thoughts on Resilience

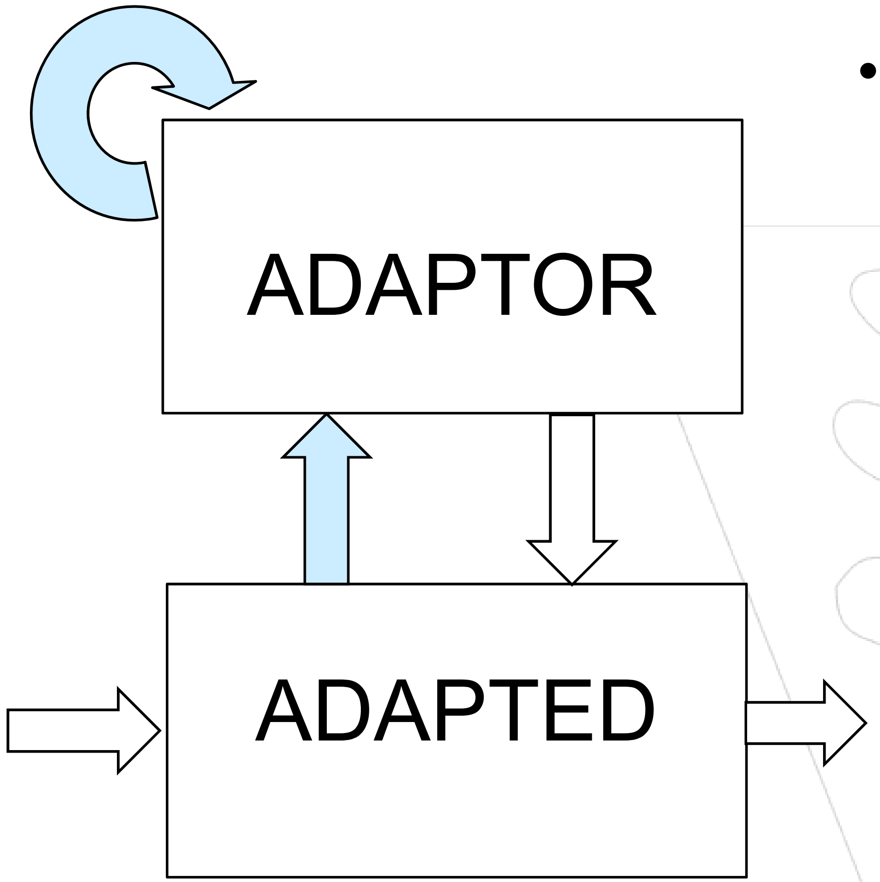
Thoughts on Resilience



(conceptualization)

- **Adaptor:** Reasoning on a different level of abstraction (meta-level)
  - What is the right level of abstraction?
    - structure
    - functionalities
    - dynamics…
  - How to derive them/link them from design models?

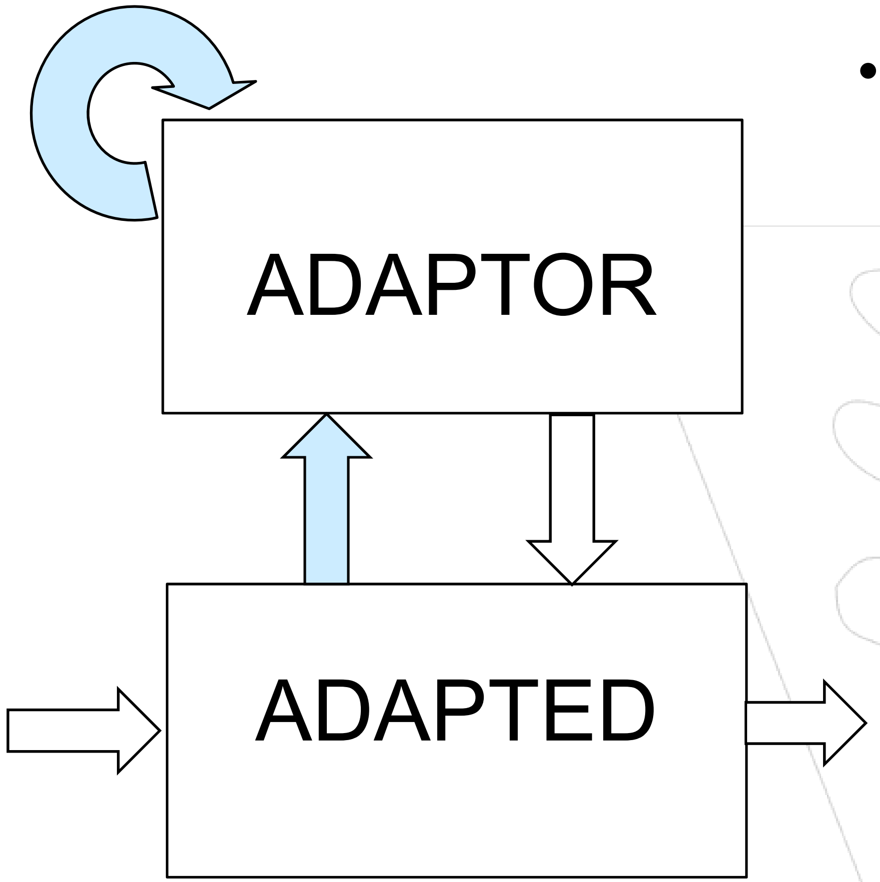
More Thoughts on Resilience



- **Adaptation method:** "embedded" design- space exploration
  - How to construct it?
    - e.g. hierarchically layered alternatives
    - parameterization
  - How to represent it?
    - e.g. OBDD
  - How to characterize it?
    - size
    - tangledness wrt selected properties

Construction of "Embedded Design Space" is the primary way of adjusting flexibility v.s. speed

...More Thoughts on Resilience



- **Engineering the design space**
  - How to achieve decoupling for *selected properties*?
    - stability – time varying delays
      method1: passive dynamics
      method2: masking/isolation layer
  - How to achieve decoupling for *multiple properties*?
    - stability & liveness
    - stability & liveness & safety
    - …….

Ongoing Work

- META and VF beta testing starts in 15 September 2012
- Trial design competition for undergarduates (UC Berkeley, MIT, SUDT, Vanderbilt)
- FANG competition starts in January 2013
- Model Integrated Computing (MIC) tools are available
- Open source META tool suite will be available in January 2013

References

- Janos Sztipanovits, Xenofon Koutsoukos, Gabor Karsai, Nicholas Kottenstette, Panos Antsaklis, Vijay Gupta, Bill Goodwine, John Baras, and Shige Wang, "Towards a Science of Cyber-Physical System Integration", *Proceedings of the IEEE,* Special Issue on Cyber-Physical Systems, 100(1), 29-44, January 2012
- Jackson, E., Sztipanovits, J.: 'Formalizing the Structural Semantics of Domain-Specific Modeling Languages," *Journal of Software and Systems Modeling* pp. 451-478, September 2009
- Jackson, Thibodeaux, Porter, Sztipanovits: "Semantics of Domain-Specific Modeling Languages," in P. Mosterman, G. Nicolescu: Model-Based Design of Heterogeneous Embedded Systems. Pp. 437-486, CRC Press, November 24, 2009
- Ethan K. Jackson, Wolfram Schulte, and Janos Sztipanovits: The Power of Rich Syntax for Model-based Development, MSR Technical Report, 2009
- Kai Chen, Janos Sztipanovits, Sandeep Neema: "Compositional Specification of Behavioral Semantics," in *Design, Automation, and Test in Europe:* The *Most Influential Papers of 10 Years DATE,* Rudy Lauwereins and Jan Madsen (Eds), Springer 2008
- Nicholas Kottenstette, Joe Hall, Xenofon Koutsoukos, Panos Antsaklis, and Janos Sztipanovits, "Digital Control of Multiple Discrete Passive Plants Over Networks", International Journal of Systems, Control and Communications (IJSCC), Special Issue on Progress in Networked Control Systems. 3(2), 194-228, 2011
- Graham Hemingway, Himanshu Neema, Harmon Nine, Janos Sztipanovits, and Gabor Karsai: Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach , SIMULATION 0037549711401950, 2011 doi:10.1177/0037549711401950