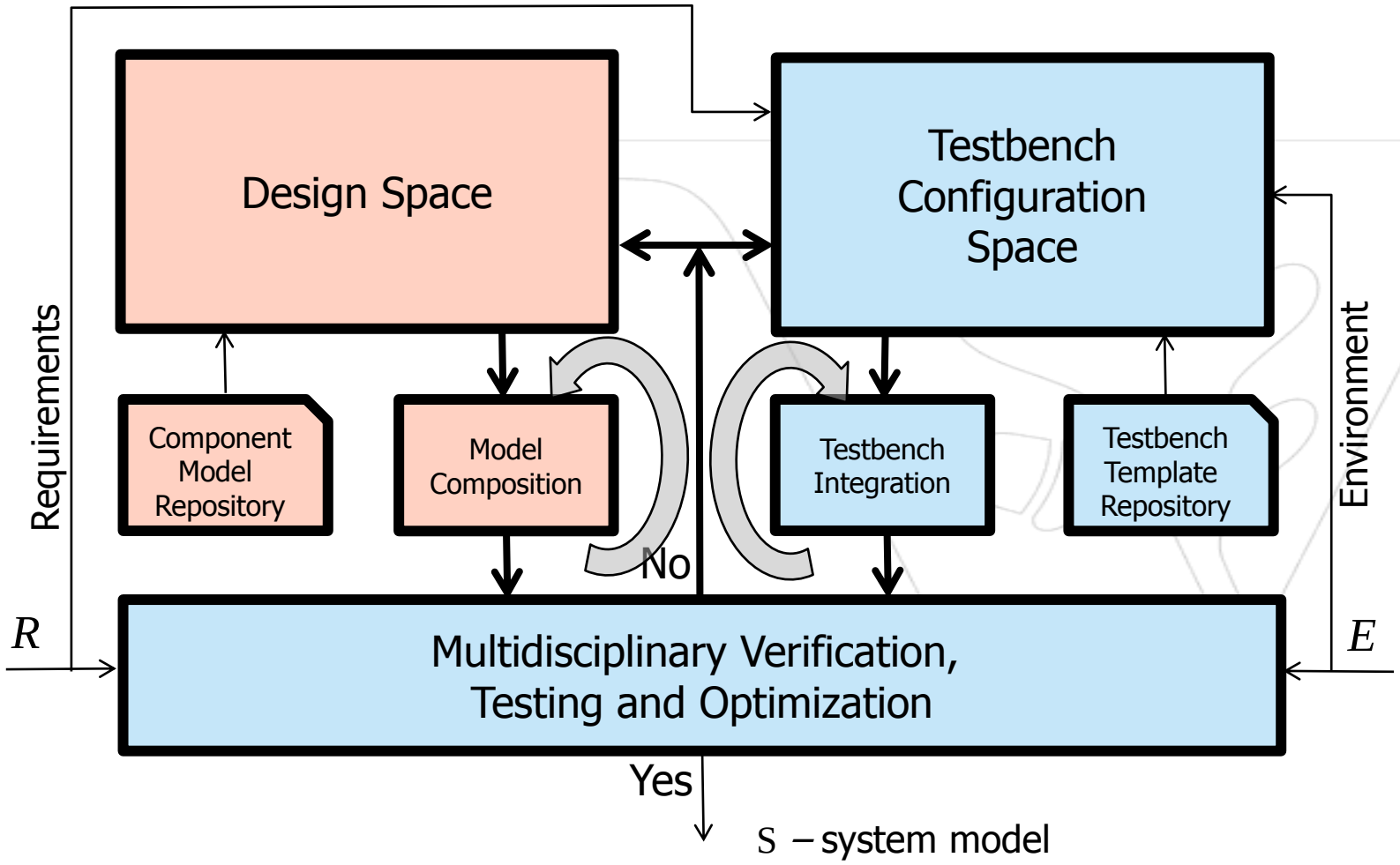
Use Case for Formal Methods: Model-Integration Platform for CPS Design

Janos Sztipanovits
Vanderbilt University

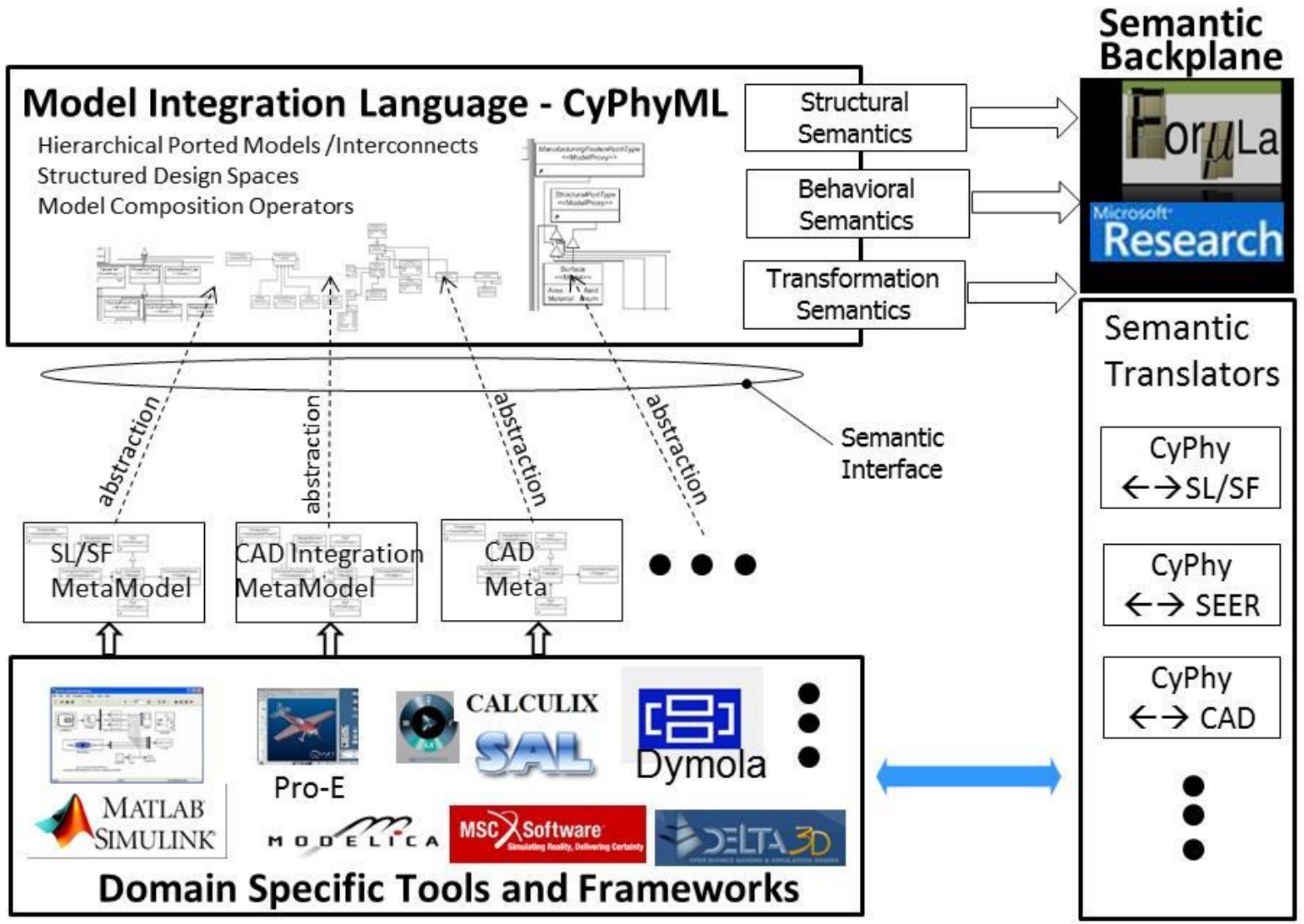
Model- and Component-based Design for CPS



- Drive-train and hull design for FANG vehicle; AVM progr. 29 opensource and 8 commercial tools
- Component models that capture both computational and physical models
- Modeling language for evaluation testbenches
- Design space exploration strategies incorporating multidisciplinary verification, testing and optimization

Janos Sztipanovits, Ted Bapty, Xenofon Koutsoukos, Zsolt Lattmann, Sandeep Neema, and Ethan Jackson. Model and tool integration platforms for cyber-physical system design. *Proceedings of the IEEE,* (99), 2018.

Model-Integration Platform



- Model Integration Languages (MIL) are changing because
  - the component models are built with different modeling tools
  - the composed analytics models depend on the key requirements
- Semantic precision of MILs requires explicit modeling of their semantics
- We used MSR FORMULA-2 as the framework for representing
  - formal semantics of semantic interfaces
  - formal semantics of model integration constructs
  - formal semantics of model transformations
- In FANG-1 challenge 19,696 lines of FORMULA spec.;11, 560 is generated and 8,136 is manually written

“Hidden Formal Methods”

Z3

Microsoft Research
https://github.com/Z3Prover/z3

FORMULA
Modeling Foundations.

```
DesignElementToPortContainment ::= new (src:DesignElement, dst:Port).
// Union types for ports
Port ::= PowerPortType
     + SignalPortType.
MechanicalPowerPortType ::= TranslationalPowerPort
                          + RotationalPowerPort.
PowerPortType ::= MechanicalPowerPortType
                + ThermalPowerPort
                + HydraulicPowerPort
                + ElectricalPowerPort.
SignalPortType ::= InputSignalPort
                 + OutputSignalPort.
// Connections of power and signal ports
PowerFlow ::=
  new (name:String,src:PowerPortType,dst:PowerPortType,...).
InformationFlow ::=
  new (name:String,src:SignalPortType,dst:SignalPortType,...).
```

Microsoft Research
github.com/Microsoft/formula

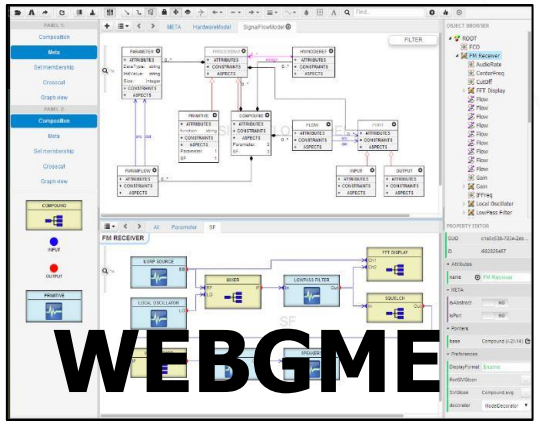
Are the models well formed?
How to synthesize/auto-complete models?

Formula
Z3
WebGME Link
Libraries

Semantic Domain: Algebraic Data Types + Logic

Tight integration is essential

Domain Specific Modeling Languages



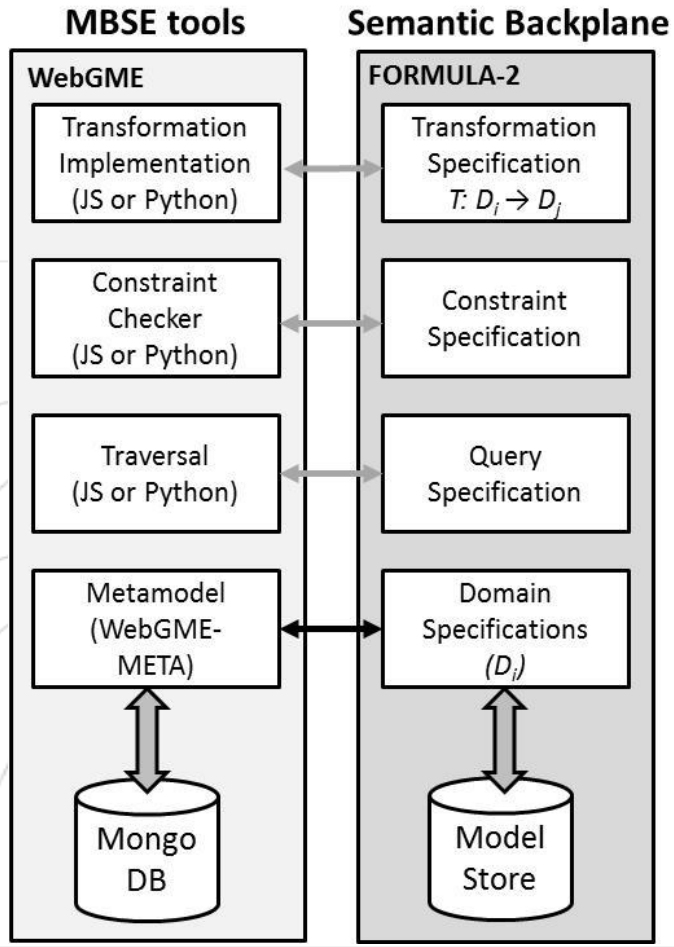
Vanderbilt
https://github.com/webgme

Metaprogrammable Modeling Tool
WebGME

Domain-Specific Modeling Languages
Programmable syntax & semantics
Configurable visualization
Distributed modeling, version mgmt.

Integration Between Model- and Data-driven Tool suites

- WebGME – building models in a collaborative manner with customized visualization
- FORMULA – logic-based modeling language for executable specification of semantics
- Tight integration is in progress



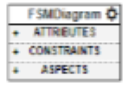
The tool suites have complimentary strengths and with the emergence of combined application domains we need services that cut across the tool suites

Define WebGME Metamodel Semantics Using FORMULA 2

- Principles
  - Models are Labeled Graphs
  - Metamodels are modeled as Typed Graphs
  - ➢ Semantics of WebGME metamodeling language is defined as typed graphs and conformance constraints
- Tool integration concept
  - Translator from WebGME to Formula 2 is implemented as a WebGME plugin
  - WebGME embeds a FORMULA editor
  - The WebGME and Forlula 2 representation of models and metamodels are kept synchronized

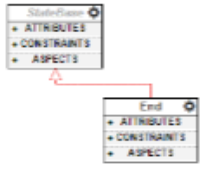
Partial representation of translation rules

Concept description	WebGME (Meta) representation	Formula translation

Component		`FSMDiagram ::= new (id: String, parent: any {NULL},`
		` attributes: any Attr__FSMDiagram,`
		` pointers: any Ptr__FSMDiagram).`

Containment		`StateBase ::= new (...parent: any FSMDiagramTYPE + {NULL},...).`
Attribute		`Transition ::= new (...attributes: any Attr__Transition...).` `Attr__Transition ::= new (guard: String, name: String, operation: String).`

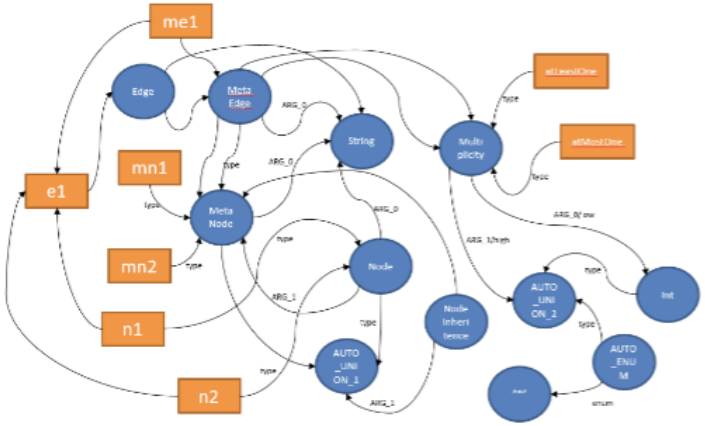
Pointer (one to one association)		`Transition ::= new (...pointers: any Ptr__Transition).`
		`Ptr__Transition ::= new (...dst: any StateBaseTYPE + {NULL},`
		` src: any StateBaseTYPE + {NULL}).`

Inheritance		`StateBase ::= new (...).`
		`End ::= new (...).`
		`StateBaseTYPE ::= StateBase + ... + End.`

Anastasia Mavridou, Tamas Kecskes, Qishen Zhang, and Janos Sztipanovits. A common integrated framework for heterogeneous modeling services. In Proceedings of the 6th International Workshop on the\ Globalization of Modeling Languages, co-located with MODELS 2018 (GEMOC 2018), October 2018.

Graph-based specification of WebGME metamodel semantics

- **Labeled Graph** – A set of vertices and a set of edges, in which edge is a binary relation over two vertices. Each vertex and edge is mapped to label of string type.
- **Typed Graph** – Extend the Labeled Graph above with an additional mapping that maps each node to it type node.



(a) FORMULA Metamodeling in Graph Representation

```
domain MetaGraph
{
  MetaNode ::= new (name: String).
  MetaEdge ::= new (name: String, src: MetaNode,
    dst: MetaNode, ms: Multiplicity, md: Multiplicity).
  Node ::= new (name: String, type: MetaNode).
  Edge ::= new (name: String, type: MetaEdge,
    src: Node, dst: Node).
  Multiplicity ::= new (low: Integer,
    high: Integer + {"*"}).
  NodeInheritance ::= new (base: MetaNode,
    instance: MetaNode + Node).
  NodeInstanceOf ::= (MetaNode, MetaNode + Node).
}
model example of MetaGraph
{
  exactlyOne is Multiplicity(1,1).
  atMostOne is Multiplicity(0,1).
  atLeastOne is Multiplicity(1,"*").
  anyNumber is Multiplicity(0,"*").
  mn1 is MetaNode("mn1").
  mn2 is MetaNode("mn2").
  me1 is MetaEdge("me1", mn1, mn2, atLeastOne, atMostOne).
  n1 is Node("n1", mn1).
  n2 is Node("n2", mn2).
  e1 is Edge("e1", me1, n1, n2).
}
```

(b) Domain in FORMULA

Graph structure is a perfect match to describe metamodel/models and their hierarchical relationship