Cyber-Physical Systems Challenge for Model-based Design

Janos Sztipanovits

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

- Ted Bapty, PI
- Ethan Jackson*
- Larry Howard, PI
- Xenofon Koutsoukos
- Zsolt Lattmann
- Tihamer Levendovszky
- Adam Nagel
- Sandeep Neema, PI
- Joseph Porter
- Gabor Simko
- and many others at ISIS – Vanderbilt

* Microsoft Research
Overview

- A Cyber-Physical Systems Design Project: AVM
  - Goals
    - Basic concepts: Vehicle Forge
    - Basic concepts: OpenMETA
  - Information Architecture Challenge
  - OpenMETA Design Flow Integration Challenge
  - Semantic Integration Challenge
    - Structural Semantics
    - Behavioral Semantics
- Summary
DARPA Adaptive Vehicle Make (AVM) Program

A major DARPA program a decade after MoBIES: End-to-end model- and component-based design and integrated manufacturing of a new generation of amphibious infantry vehicle – a complex, real-life cyber-physical system. From infrastructure to manufactured vehicle prototype in five years (2010-2014).

Engineering/economic goals:

- Decrease development time by 80% in defense systems (brings productivity consistent with other industries)
- Enable the adoption of fabless design and foundry concept in CPS
- “Democratize” design by open source tool chain, crowd-sourced model library and prize-based design challenges
Achieve AVM goals by pushing the limits of “correct-by-construction” design using

- **Model-based Technologies**
  Computational models that predict properties of cyber-physical systems “as designed” and “as built”.

  **Challenge:** Develop domain-specific abstraction layers for complex CPS that are evolvable, heterogeneous, yet semantically sound and supported by tools.

- **Component-based Technologies**
  Reusable units of knowledge (models) and manufactured components.

  **Challenge:** Go beyond interoperability – find opportunities for composition where system-level properties can be computed from the properties of components.
Technical Areas

ISIS

Vehicle Forge

Model Library; Curation

FANG Competition Coordination

OpenMETA Tools

Foundry

FANG Competitors

Use Tools

Collaborate Using VF

MFG Feedback

Design Data

Curated Components

Requirements, Test Benches

Seed Designs, Scores

Analysis Components, Designs, Design Spaces

Produces Design Data

Tools
Overview

- A Cyber-Physical Systems Design Project: AVM
  - Goals
  - Basic concepts: Vehicle Forge
  - Basic concepts: OpenMETA
- Information Architecture Challenge
- OpenMETA Design Flow Integration Challenge
- Semantic Integration Challenge
  - Structural Semantics
  - Behavioral Semantics
- Summary
Interface to OpenMETA: VehicleForge

**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 expertise and résumé
- User dashboard showing feeds of activities from projects, public/private messages from other users, announcements from forge-message channels

**Components**
- OIS Data
  - CAT C7.8
  - CAT C7.1

**Design Projects**
- CAT C7.8
  - Design view/visualization

**Designers**
- User dashboard showing feeds of activities from projects, public/private messages from other users, announcements from forge-message channels
Service Integration Platform

- Coordination and Monitoring Tools
- Design-space Evaluation and Visualizers
- Team-collaboration Tools
- Component Discovery and Subscription
- Service and Resource Allocation

Analysis & Simulation Service Providers

Manufacturers & Foundries

Component Vendors

In-cloud Compute & Test bench Services

• MongoDB
• Git, SVN, Swift
• Apache SOLR
• TurboGears (Web Framework)
• REST Service APIs

Exchange

Design & Manufacturing Components

Ontologies

Licensing

Ordering

Teams’ Design Storage

• Sharing and Collaboration
• Cloud-based Analysis
• Access to Remote Resources

Integrated VF Service Gateway

CyPhy Desktop Tool Environment

Browser-based
- Coordination and Monitoring Tools
- Design-space Evaluation and Visualizers
- Team-collaboration Tools
- Component Discovery and Subscription
- Service and Resource Allocation

http://vehicleforge.org/
Overview

- A Cyber-Physical Systems Design Project: AVM
  - Goals
  - Basic concepts: Vehicle Forge
  - Basic concepts: OpenMETA
- Information Architecture Challenge
- OpenMETA Design Integration Challenge
- Semantic Integration Challenge
  - Structural Semantics
  - Behavioral Semantics
- Summary
AVM Components

Components span:
- Multiple physics
- Multiple domains
- Multiple tools

Component-based:
- Physical
- Cyber
- Cyber-Physical

Model-based:
- Model-Integrated
- Design and
- Manufacturing Process
Caterpillar C9 Diesel Engine : AVM Component

**High-Fidelity Modelica Dynamics Model**
- Rotational Power Port
- Signal Port

**Low-Fidelity Modelica Dynamics Model**
- Rotational Power Port
- Signal Port

**Bond Graph Dynamics Model**
- Rotational Power Port
- Signal Port

**Detailed Geometry Model (CAD)**
- Structural Interface
- Structural Interface

**FEA-Ready CAD Model**
- Structural Interface
- Structural Interface

**Throttle**
- Signal Port

**Power Out**
- Rotational Power Port

**Power Interfaces**
- acausal
- physical phen. (torque/angle..)
- power flow

**Signal Interfaces**
- causal/directional
- logical conn.
- no power transfer

**Structural Interfaces**
- named datums
- surface/axis/point
- mapped to CAD

**Param./Property Interfaces**
- characterize
- configure

**Dynamics**

**Detailed Geometry**

**FEA Geometry**

- Weight: 680 kg
- Height: 1070 mm
- Number of Cylinders: 6
- Maximum RPM: 2300 rpm
- Length: 1245 mm
- Width: 894.08 mm
- Maximum Power: 330 kW
- Minimum RPM: 600 rpm
<table>
<thead>
<tr>
<th>Components</th>
<th>Designs</th>
<th>Design Spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pars</td>
<td>C1(p)</td>
<td>C1(p)</td>
</tr>
<tr>
<td>Stru.</td>
<td>C2(q)</td>
<td>C2(q)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C1,2(p,q)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D1,2(p,q)</td>
</tr>
<tr>
<td>Self-contained building block</td>
<td>Instantiate and connect Components</td>
<td>Sets of parameterized architectures</td>
</tr>
<tr>
<td>Properties and Parameters</td>
<td>Parameters, behaviors, geometry are composed</td>
<td>Extended around seed designs</td>
</tr>
<tr>
<td>Wrapper for detailed domain models</td>
<td>Can be wrapped as a component</td>
<td>Shaped by design and manufacturability constraints</td>
</tr>
<tr>
<td>Aggregates the domain interfaces into a single set of component interfaces</td>
<td>Aggregates the component interfaces into a single set of system interfaces.</td>
<td>Accumulates, evolves design and manufacturing knowledge</td>
</tr>
</tbody>
</table>
# Design Flow

<table>
<thead>
<tr>
<th>Architecture Design</th>
<th>Integrated Multi-physics/Cyber Design</th>
<th>Detailed Design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modeling</strong></td>
<td><strong>Modeling</strong></td>
<td><strong>Modeling</strong></td>
</tr>
<tr>
<td>Exploration</td>
<td>Simulation</td>
<td>Analysis</td>
</tr>
<tr>
<td>Rapid exploration</td>
<td>V&amp;V</td>
<td>Deep analysis</td>
</tr>
</tbody>
</table>

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

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

- **Architecture Modeling**
- **Detailed Domain Modeling**
  - CAD
  - FEA; thermal, fluid…
  - Surrogate gen.
- **Detailed Mfg. modeling**
- **RT SW modeling**
Requirements and Test Benches

- Using each component’s mappings to detailed domain models, system-level analyses is automatically composed.
  - Static properties
  - Multi-physics dynamics
  - Geometry
  - FEA
- **META Test Benches** provide an analysis context, including stimulus, loading, and monitoring.
- **Test Benches** include algorithms to produce **Metrics**, which are used to evaluate the design against **Requirements**.
- **META Design Models** are mapped to these **Test Benches**.
- **Design Spaces** can also be mapped to **Test Benches**, enabling rapid evaluation of a family of point designs.
Overview

- A Cyber-Physical Systems Design Project: AVM
  - Goals
  - Basic concepts: Vehicle Forge
  - Basic concepts: OpenMETA

- Information Architecture Challenge
- OpenMETA Design Flow Integration Challenge
- Semantic Integration Challenge
  - Structural Semantics
  - Behavioral Semantics

- Summary
Design Flow Spans Heterogeneous Modeling Domains

<table>
<thead>
<tr>
<th>Architecture Design</th>
<th>Integrated Multi-physics/Cyber Design</th>
<th>Detailed Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Exploration</td>
<td>Modeling Simulation V&amp;V</td>
<td>Modeling Analysis</td>
</tr>
<tr>
<td>Rapid exploration</td>
<td>Exploration with integrated optimization and V&amp;V</td>
<td>Deep analysis</td>
</tr>
</tbody>
</table>

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

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

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

Domain Specific Modeling Languages
Modeling Domains

Key META challenge is modeling cross-domain interactions
Information Flows Across Program Components

Model Library; Curation
- Curated Components

Vehicle Forge
- Analysis
- Collaborates Using VF
- Components, Designs, Design Spaces

OpenMETA Tools
- Uses Tools
- Produces Design Data
- MFG Feedback

Competition Coordination
- Component Model
- Design, Design Space, Test Bench Models
- Component, Design, Design Space Models
- Test Bench Models
- Use cases/Scenarios
- META/MFG Interface

Foundry

Competitors
Information Architecture
Challenges

- Shared conceptualization
- Semantically sound modeling languages
- Integration of many tools and their modeling languages
Towards and Information Architecture

Shared conceptualization
(Vocabularies, ontologies)

Integrated Modeling Languages

CyPhyML
HBG
SignalFlow
How Should We Choose Vocabularies, Ontologies?

- Could not find standards covering even smaller part of the AVM domain
- Grow and evolve vocabularies/ontologies during model library development
- Adopt vocabularies as defined by integrated tools (such as Modelica)
How Should We Choose Modeling Language(s)?

- Define yet another modeling language?
- Choose one that already exists and broad enough to cover the design domain?
- Create a new standard or update an old one?

- What are the implications on tools?
- How about “my freedom of abstractions”?
- What is the language evolution trajectory?
Case for Model Integration Languages...

**Model Integration Language - CyPhy**
- Hierarchical Ported Models /Interconnects
- Structured Design Spaces
- Model Composition Operators

**Semantic Backplane**
- Structural Semantics
- Behavioral Semantics
- Transformation Semantics

**Domain Specific Tools and Frameworks**
- SL/SF MetaModel
- CAD Integration MetaModel
- CAD Meta
- Pro-E
- SAL
- Dymola
- MATLAB/SIMULINK
- MODELICA
- MSC Software
- DELTA 3D

**Model-Based Design**

**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

---

**Domain Specific Design Automation Environments:**
- Automotive
- Avionics
- Sensors...

**Tools:**
- Modeling
- Analysis
- Verification
- Synthesis

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

---

**Mathematical and physical foundations**
Metaprogrammable Design Tools: “Freedom of Abstractions”

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/)

Domain Specific Design Automation Environments:
- Automotive
- Avionics
- Sensors...

Metaprogrammable Tool Infrastructure:
- Model Building
- Model Transf.
- Model Mgmt.
- Tool Integration

Explicit Semantic Foundation:
- Structural
- Behavioral

Design Requirements → Domain-Specific Environments → Production Facilities

Meta Layer

Design Layer

Semantic Foundation Component Libraries

doTransition (fsm as FSM, s as State, t as Transition) =
require s.active
step exitState (s)
step if t.outputEvent <> null then emitEvent (fsm, t.outputEvent)
step activateState (fsm, t.dst)
# OpenMETA Information Architecture

## Design Data Package (DDP)

<table>
<thead>
<tr>
<th>Models and Modeling Languages</th>
<th>CyPhy Model Integration Language</th>
<th>Embedded System Modeling Language (ESMOL)</th>
<th>Modelica</th>
<th>DESERT</th>
<th>CAD</th>
<th>FEA</th>
<th>Parametric Exploration Tool (PET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Model</td>
<td>Modelica</td>
<td>Bond Graph</td>
<td>Qualitative Abstraction</td>
<td>Relational Abstraction</td>
<td>Probab. Analysis (PCC)</td>
<td>Fault Modeling</td>
<td></td>
</tr>
<tr>
<td>Design Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Space Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result Package</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Standardized Vocabularies and Core Types

<table>
<thead>
<tr>
<th>META Ontologies</th>
<th>VehicleForge Ontology</th>
<th>iFAB Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface &amp; Composition Vocabulary</td>
<td>Testing Vocabulary</td>
<td>Vehicle Component Vocabularies</td>
</tr>
<tr>
<td>Behavior Vocabulary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<<note>>
In progress. Currently includes characterizations of supplier data (unknown source for this vocab)
Summary of OpenMETA – Approach in Information Architecture

- Model-Integration Language: CyPhyML
- Use of Metaprogrammable tools (MIC Tool Suite of ISIS/Vanderbilt)
- Use of Semantic Integration (see later)
Overview

- A Cyber-Physical Systems Design Project: AVM
  - Goals
  - Basic concepts: Vehicle Forge
  - Basic concepts: OpenMETA
- Information Architecture Challenge
- OpenMETA Design Flow Integration Challenge
- Semantic Integration Challenge
  - Structural Semantics
  - Behavioral Semantics
- Summary
# Design Flow

<table>
<thead>
<tr>
<th>Architecture Design</th>
<th>Integrated Multi-physics/Cyber Design</th>
<th>Detailed Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling</td>
<td>Exploration</td>
<td>Modeling</td>
</tr>
<tr>
<td>Rapid exploration</td>
<td>Exploration with integrated optimization and V&amp;V</td>
<td>Deep analysis</td>
</tr>
</tbody>
</table>

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

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

- **Architecture Modeling**
- **Detailed Domain Modeling**
  - **CAD**
  - **FEA; thermal, fluid…**
  - **Surrogate gen.**
- **Detailed Mfg. modeling**
- **RT SW modeling**
Design Flow Integration
Challenges

- How to start the design process?
- How to help its convergence to a “good enough” solution?
- How to link all the tools?
OpenMETA “Composers”

Vehicle Forge (Comp Exch) -> Component Models -> Evolve Design Spaces -> Requirements / Test Benches

Seed Design Spaces -> Evolve Design Spaces -> Compose with Test Benches

META Composers

Create Components

OpenMETA Tools (used by Competitors)
Executable Requirements & Test Bench Concepts

- **Scenario Specification**
- **Parameters**
- **Environment Specification**
- **Test Article**
- **Instrumentation**
- **Metrics & Requirements**
Example for Test Benches to Evaluate FANG Requirements
Architecture Exploration
Using & Interface Abstractions
Design Space Exploration Using Multi-Fidelity ODEs

Uncertainty Propagation & Estimation

Simulation Test Bench for Behavioral Properties

- Design Space Exploration
- Multi-Fidelity Behavior Models
- Multiple Physics Domains
- Multi-Component Modeling

PCC
Design Space Exploration Using Geometry and FEA

- CAD Testbench for Physical Properties
  - KPP
    1) Bounding box
    2) Center of Gravity
    3) Dimensions

- FEA Testbench for Structural Properties
  - KPP
    1) maximum shear stress,
    2) maximum bearing stress,
    3) maximum Von Mises stress
    4) factor-of-safety
OPenMETA SW Tool Chain

- Time-triggered Model of Computation
- TT bus (or emulated TT bus)
- Event-triggered Model of Computation
- CAN bus

Design Architectures
with ideal comp.
dynamics

Design Architectures
with deployed comp.
dynamics
Design Space Visualization
Pairwise Metric Visualization
Geometric Reasoning: CAD Assembly Composition

META Model of Structural Connections

CAD - Independent Assembly

Specific Drivers

iF AB Interface (partial)

BOM, Assembly, GD&T, …
Overview

- A Cyber-Physical Systems Design Project: AVM
  - Goals
  - Basic concepts: Vehicle Forge
  - Basic concepts: OpenMETA
- Information Architecture Challenge
- OpenMETA Design Flow Integration Challenge
- Semantic Integration Challenge
  - Structural Semantics
  - Behavioral Semantics
- Summary
Need for Formal Semantics

META Semantic Integration

- Simulink/Stateflow
- Embedded Software Modeling
- Hybrid Bond Graph
- Modelica
- TrueTime
- Functional Mock-up Unit

Composition
- Continuous Time
- Discrete Time
- Discrete Event
- Energy flows
- Signal flows
- Geometric

Equations
- Modelica-XML
- Modelica-XML

FMU
- S-function
- FMU-ME
- FMU-CS

High Level Architecture Interface (HLA)

Formal Verification
- Qualitative reasoning
- Relational abstraction
- Model checking
- Bounded model checking

Stochastic Co-Simulation
- Open Modelica
- Dymola

Distributed Simulation
- NS3
- OMNET
- Delta-3D
- CPN
Concept of “Semantic Integration”

Physical Component Structure

CyPhy Component
- Bond Graph
- Modelica

CyPhy Integration Model

Tool Integration

Semantics Model
- BG Semantics
- BG2Mod Semantics
- Mod. EQ Semantics
- CyPhy Semantics
- CyPhy Comp Semantics

VER SIM
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

FORMULA: http://research.microsoft.com/formula
Convergence in Formal Framework: FORMULA

- History: Foundations for Embedded Systems ITR; Ethan Jackson at VU 2005-2008
- Microsoft Research (Bellevue & Aachen); 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
**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:** \( D(Y, C) = \{\text{nil}\} \)
- **DSML composing:** \( D_1 \ast D_2 | D_1 + D_2 | D' \text{ includes } D \)
- **Model finding:** \( S = \{ s \in D | s| = P \} \)
- **Transforming:** \( m' = T(m); m' \in X; m \in Y \)
**Behavioral Semantics** defines exhibited behavior of models by
1. Specifying a translation to a domain with well-understood *operational semantics*
2. Specifying a translation to a mathematical domain defining behaviors *denotationally* (e.g. symbolic DAEs)

**Use of Behavioral Semantics Specifications:**
- Validating/understanding behaviors via simulation
- Generating behaviors using “reference semantics” and testing tools w.r.t. reference semantics
- Invariance checking
- Formalization -> first steps toward proofs
- Tracking dependences in tool suites
## Layers of the Semantic Backplane

<table>
<thead>
<tr>
<th>Functions</th>
<th>(Meta)Models</th>
<th>Languages</th>
<th>Tools</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metamodeling</td>
<td></td>
<td>MetaGME</td>
<td>• GME</td>
<td>• DSML spec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• MetaGME-2-Formula</td>
<td>• Constraint Checking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Metaprog.</td>
</tr>
<tr>
<td>Transformation Modeling</td>
<td></td>
<td>UMTL</td>
<td>• GReAT</td>
<td>• Transf. spec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• UDM</td>
<td>• Compiling spec to transformer</td>
</tr>
<tr>
<td>Formal Metamodeling</td>
<td></td>
<td>Formula (MSR)</td>
<td>• Domain Comp.</td>
<td>• Metamod. checking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Trace Gen.</td>
<td>• Example gen.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Semantic units</td>
</tr>
<tr>
<td>Formal Transformation Modeling</td>
<td></td>
<td></td>
<td>• Semantic Anchoring</td>
<td>• Semantics for complex</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DSMLs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Composiiiton</td>
</tr>
</tbody>
</table>
Structure of the Semantic Backplane

Informal models/metamodels

- Instance Model
  - Instantiates
  - Semantic Mapping (Execution of Logic Programs)
  - Instance Models (Formal Structures in Term Algebra)

Formal models/metamodels

- Metamodelll
  - Defines
  - Structural Constraints Metamodelll (Term Algebraic Representation)
  - Semantic Mapping Rules (Logic Programs)
  - Structural Constraints

Testing and verification

Updates, Revisions, extensions

Specification for Operational Semantics (Logic Program)

Grounding Denotational Semantics in Mathematics

Semantic Domain (Term Algebraic Representation)

Denotational Semantics in Mathematics

Structural Constraints

Informal models/metamodels

- Instance Model (Term Algebraic Representation)
  - Specifies
  - Metamodelll (Term Algebraic Representation)

- Instance Model
  - Instantiates
  - Semantic Mapping (Execution of Logic Programs)

- Instance Models (Formal Structures in Term Algebra)
Metamodel and Formal Metamodel - ADTs

```
1  domain AcausalBG_elements
2  {
3    primitive Sf ::= (id: String).
4    primitive Se ::= (id: String).
5    primitive R ::= (id: String).
6    primitive C ::= (id: String).
7    primitive I ::= (id: String).
8    primitive TF ::= (id: String).
9    primitive GY ::= (id: String).
10   primitive ZeroJunction ::= (id: String).
11   primitive OneJunction ::= (id: String).
12   Source ::= Sf + Se.
13   Storage ::= C + I.
14   OnePort ::= Source + R + Storage.
15   TwoPort ::= TF + GY.
16   BGElem ::= OnePort + TwoPort.
17   Junction ::= ZeroJunction + OneJunction.
18   BGNode ::= BGElem + Junction.
19   primitive Bond ::= (id: String).
20  } [Closed]
21  } [Closed]
```

Metamodel of a simplified Acausal Bond Graph DSML

Formal Metamodel of a simplified Bond Graph DSML
Part of Structural Semantics for Acausal Bond Graphs

- Structural semantics is composed of constraints on model structure
- Modeling tools need to check constraints during modeling
- A well-formed model can be mapped into some behavior

```prolog
domain AcausalBG extends AcausalBG_elements {
  invalidBondDef := a is Bond, no Src(a,_).
  invalidBondDef := a is Bond, no Dst(a,_).
  bondConn(a,x) :- Src(a,x); Dst(a,x).
  atLeastOneConnection(x) :- bondConn(_,x).
  atLeastTwoConnections(x) :-
    bondConn(a,x), bondConn(b,x), a != b.
  exactlyOneConnection(x) :-
    atLeastOneConnection(x),
    no atLeastTwoConnections(x).
  invalidBlock := x is OnePort,
    no exactlyOneConnection(x).
  invalidBlock := x is TwoPort,
    no exactlyTwoConnections(x).
  invalidBlock := x is R, Src(_,x);
    x is C, Src(_,x);
    x is I, Src(_,x).
  invalidBlock := x is TwoPort, no Src(_,x);
    x is TwoPort, no Dst(_,x).
  conforms := !invalidBlock &
    !invalidBondDef &
    !invalidSrcDef &
    !invalidDstDef.
}
```
Specifying Behavioral Semantics

\[ D(Y, C) = \{ r \in R_Y \mid r = C \} \]

\[[ \ ]]: R_Y \mapsto R_Y' \]

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

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

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

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.
}
```

1 transform BG_DenotationalSemantics
from in1::AcausalBG
to out1::DAEquations
{
  Eq(e, pa) :- x is Se, Src(a,x).
  Eq(f, pa) :- x is Sf, Src(a,x).
  Eq(e, Mul(pa, f)) :- x is R, Dst(a,x).
  DiffEq(e, Mul(Inv(pa), f)) :-
     x is C, Dst(a,x).
  //..
}
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.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, _).
}
CyPhyML - Modelica Power Connections

The behavioral semantics of Modelica power ports is the same as that of CyPhyML. For example, in electrical domain effort is voltage and flow is current in both CyPhyML and Modelica.

\[ \forall (x, y) \in P \ (v_x = e_x \wedge f_y = f_y) \]

where \( P \) is the set of Modelica - CyPhyML power port mappings.

\[ \text{Equals}(\text{cyphyEffort}, \text{modelicaEffort}), \]
\[ \text{Equals}(\text{cyphyFlow}, \text{modelicaFlow}) :: \]
\[ \text{ModelicaPowerPortMap} = \text{ModelicaPort}, \text{cyphyPort}, \text{PowerVarNaming}(\text{cyphyPort}, \text{cyphyEffort}, \text{cyphyFlow}), \text{PowerVarNaming}(\text{modelicaPort}, \text{modelicaEffort}, \text{modelicaFlow}). \]

CyPhyML - Modelica Signal Connections

The behavioral semantics of Modelica signal ports is the same as that of CyPhyML.

\[ \forall (x, y) \in P \ (v_x = s_y) \]

where \( P \) is the set of Modelica - CyPhyML signal port mappings.

\[ \text{Assign}(\text{cyphySignal}, \text{modelicaSignal}) :: \]
\[ \text{ModelicaSignalPortMap} = \text{ModelicaPort}, \text{cyphyPort}, \text{SignalVarNaming}(\text{cyphyPort}, \text{cyphySignal}), \text{SignalVarNaming}(\text{modelicaPort}, \text{modelicaSignal}). \]

CyPhyML - SignalFlow Signal Connections

While signal ports in signal-flow are discrete-time ports, signal ports in CyPhyML are continuous-time. Thus, signal-flow output signals are integrated into CyPhyML by means of a hold function.

\[ \forall (x, y) \in P \ (v_y := \text{hold}(v_x)) \]

where \( P \) is the set of SignalFlow output - CyPhyML signal port mappings.
Summary

- **Understanding current limits of correct-by-construction design using model-based verification**
  - Significant scalability problems even in relatively simple (but real) systems
  - Scalable verification requires strong restrictions on modeling abstractions (e.g. linear hybrid dynamics, order reduction) and NOT high data fidelity
  - The resulting uncertainty is epistemic and cannot be characterized probabilistically
  - What is the impact of this on Probabilistic Certificate of Correctness? (E.g. the confidence cannot be improved by statistical methods beyond some “epistemic limit”)

---