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

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

<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><strong>Exploration</strong></td>
<td><strong>Simulation</strong></td>
<td><strong>Analysis</strong></td>
</tr>
<tr>
<td><strong>Deep analysis</strong></td>
<td><strong>Behavior exploration using</strong></td>
<td><strong>“Deep”Domain</strong></td>
</tr>
<tr>
<td><strong>Rapid exploration</strong></td>
<td><strong>Exploration with integrated optimization and V&amp;V</strong></td>
<td><strong>Analysis</strong></td>
</tr>
<tr>
<td><strong>Structure/CAD/Mfg</strong></td>
<td><strong>Physics-based</strong></td>
<td><strong>Deep analysis</strong></td>
</tr>
<tr>
<td><strong>Rapid architecture exploration</strong></td>
<td><strong>Behavior exploration using lumped parameter dynamics</strong></td>
<td><strong>“Deep”Domain Analysis</strong></td>
</tr>
<tr>
<td><strong>Engineered design space</strong></td>
<td><strong>Basic structure synthesis (CAD)</strong></td>
<td><strong>CAD</strong></td>
</tr>
<tr>
<td><strong>Static constraints (multiphysics)</strong></td>
<td><strong>Manufacturing Constraint Checking</strong></td>
<td><strong>FEA; thermal, fluid…</strong></td>
</tr>
<tr>
<td><strong>100s of “interesting” configurations remains</strong></td>
<td><strong>Composition with testbenches, PCC</strong></td>
<td><strong>Surrogate gen.</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Parametric design tradeoffs</strong></td>
<td><strong>Detailed Mfg. modeling</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Selecting few target configurations</strong></td>
<td><strong>RT SW modeling, code generation and testing</strong></td>
</tr>
</tbody>
</table>
Interaction Among Major Program Components

Model Library; Model Curation

Curated Components

Vehicle Forge

META-X Tools

Uses Tools

META-X Produces Design TDP

FANG Lead-Coordinator

AVM Component Model

Uses Tools

Collaborates Using VF

META/MFG Interface

META-X Tool Competitors

FANG Competitors
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 Design - Project Set-up
- Project and design space templates
- Built-in design space structure visualization
- META tool downloads

Component Design
- Sharing project design and analysis artifacts
- META tool downloads for project design space visualization
- VF provides tools for project design space visualization

Component Exchange
- Curated component sharing
- Taxonomy-based component discovery
- Drag-and-drop publishing, subscription and forking
- Change notifications

Design Interaction
- Sharing of project design and analysis artifacts through hosted version control repositories
- VF uses DIF to interrelate design and forge artifacts
- VF agents react to repository changes, e.g., by scheduling analyses and pre-processing
- Designers engage with design spaces and individual design artifacts through VF repository browsers

VF provides collaboration tools for project management, discussions and documentation
- Designers meet, chat and design with Electrotank VCDE

Design Workflow Support
- Sharing of project design and analysis artifacts
- META tool downloads for project design space visualization
- VF provides tools for project design space visualization

Ontology Administration
- Version-controlled ontology definitions
- Exportable schema definitions for tools
- Cloud-based analysis job scheduling
- Presentation using built-in and custom web visualizers

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

Ontology Hosting and Administration
- Version-controlled ontology definitions
- Exportable schema definitions for tools
- Cloud-based analysis job scheduling
- Presentation using built-in and custom web visualizers

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

Ontology Hosting and Administration
- Version-controlled ontology definitions
- Exportable schema definitions for tools
- 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
**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
**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/](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
Components of a CPS

- **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

Components span:
- Multiple physics
- Multiple domains
- Multiple tools
## CPS Design Flow Requires Model Integration

### Architecture Design
- **Modeling**
- **Exploration**
  - Rapid exploration

### Integrated Multi-physics/Cyber Design
- **Modeling**
- **Simulation**
- **V&V**
  - Exploration with integrated optimization and V&V

### Detailed Design
- **Modeling**
- **Analysis**
  - Deep analysis

- **SW**
- **Physics-based**
- **Structure/CAD/Mfg**

**Domain Specific Modeling Languages**

- **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**
• **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**.

**Caterpillar C9 Diesel Engine : AVM Component**

- **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
Ontologies and DSMLs

DSMLs

AVM Component Model
CyPhy Design Model
CyPhy Design-Space Model
AVM MFG Model
META Core Types
IFAB Core Types
Other Types

Shared ontologies
## Example: Architecture Modeling

<table>
<thead>
<tr>
<th>Sublanguage / Capability</th>
<th>Formalism, Language Constructs, Examples</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture Modeling</strong></td>
<td>Hierarchical Module Interconnect</td>
<td>Systems Architect</td>
</tr>
<tr>
<td></td>
<td>- Components</td>
<td>- Explore Design Space</td>
</tr>
<tr>
<td></td>
<td>- Interfaces</td>
<td>- Derive Candidate Designs</td>
</tr>
<tr>
<td></td>
<td>- Interconnects</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Properties</td>
<td></td>
</tr>
<tr>
<td><strong>Design Space Modeling</strong></td>
<td>Hierarchically Layered Parametric Alternatives</td>
<td>Systems Architect</td>
</tr>
<tr>
<td></td>
<td>- Alternatives/Options</td>
<td>- Define Design Space</td>
</tr>
<tr>
<td></td>
<td>- Parameters</td>
<td>- Define Constraint</td>
</tr>
<tr>
<td></td>
<td>- Constraints</td>
<td></td>
</tr>
</tbody>
</table>
## Example: Dynamics Modeling

### Physical Dynamics Modeling
- **Hybrid Bond Graphs**
  - Efforts, Flows,
  - Sources, Capacitance, Inductance,
  - Resistance,
  - Transformers Gyrators,

### Computational Dynamics Modeling
- **Dataflow + Stateflow + TT Schedule**
  - Interaction with Physical Components
  - Cyber Components
  - Processing Components

### Component Engineer
- model dynamics with Hybrid Bond Graphs

### System Engineers
- Compose system dynamics

### Domain Engineers
- design controller

### System Engineers
- Processor allocate
- Platform Effects
### Solid Modeling (CAD / Geometry)

**Component Engineer**
- Defines Structural Interface
- Defines Architecture

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

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

### Manufacturing Modeling

**Standard Structural Interfaces (ex: SAE #1)**

**MechanicalFasten**

**AdhesiveFasten**

**ElectronicsFasten**

**WeldedFasten**

**Fastener Types**
- Nut/Bolt/Washer (Hand)
- Number/Type/Fasteners: 12
- Fastener Diameter: 0.4375
- Fastener Pitch: 14
- Fastener Edge Distance: 0

**Example: Physical Structure and Manufacturing Modeling**

- Power Out (SAE #1)
- Power In (SAE #1)

- C9_Diesel
- VU_ISG_V1

- Standard Structural Interfaces

- Component Engineer
  - Defines Part Cost
  - Defines Structural Interface, Fastener
### Example: Requirement Modeling

<table>
<thead>
<tr>
<th>Requirements</th>
<th>DOORS SySML</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable Requirements (Constraints)</td>
<td>DESSERT Constraint Language</td>
<td>Systems Engineer - Requirement decomposition</td>
</tr>
<tr>
<td>Executable Requirements (Test Bench)</td>
<td>Domain-Specific Instantiations - Context - System Configuration - KPP Evaluation</td>
<td>Systems Engineer Domain Engineer - Requirement decomposition</td>
</tr>
<tr>
<td>Executable Requirements (Verification Test Bench)</td>
<td>- Context - Statistical Parameters - Temporal Monitors</td>
<td>Verification Engineer - Requirement decomposition</td>
</tr>
</tbody>
</table>

**Example Constraint:**

```plaintext
constraint ForceOnGround_TracksArchitecture_withISG() {
    ((25000 + (self.parent().Weight() * 2.204623)) <= (17 * children("TrackPack").ContactPatch()))
}
```

**Bayesian Statistical Model Checker (Clarke et al):**

\[ \sim F[0,5.5s](f1 \leq 0 \land f2 \leq 180) \]
Physical components are involved in multiple physical interactions (multi-physics)
Challenge: How to compose multi-models for heterogeneous physical components
Cyber-physical components are modeled using multiple abstraction layers. Challenge: How to compose abstraction layers in heterogeneous CPS components?

- **Dynamics:** \( B(t) = \kappa_p(B_1(t),...,B_j(t)) \)
  - Properties: stability, safety, performance
  - Abstractions: continuous time, functions, signals, flows,…

- **Software:** \( B(i) = \kappa_c(B_1(i),...,B_k(i)) \)
  - Properties: deadlock, invariants, security,…
  - Abstractions: logical-time, concurrency, atomicity, ideal communication,…

- **Systems:** \( B(t_j) = \kappa_p(B_1(t_i),...,B_k(t_i)) \)
  - Properties: timing, power, security, fault tolerance
  - Abstractions: discrete-time, delays, resources, scheduling,…
Dimensions of Composition in CPS

Model abstraction

Physical domains

Hierarchical decomposition

Physical

Electrical

Hydraulic

Mechanical

Electromagnetic

Thermal

Cyber

Behavior Abstraction Layers

Hierarchical decomposition

Continuous/discrete time

Logical time

Discrete event

Refinements
Case for Model Integration Languages...

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

### Example for Model Integration

#### Architectural Design
- **Modeling**
- **Exploration**
  - Rapid exploration

#### Integrated Multi-Physics/Cyber Design
- **Modeling**
- **Simulation**
- **V&V**
  - Exploration with integrated optimization and V&V

#### Detailed Design
- **Modeling**
- **Analysis**
  - Deep analysis

**Domain Specific Modeling Languages**

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

- **Design Space + Constraint Modeling**
- **Architecture Modeling**
- **Dynamics Modeling**
- **Computational Behavior Modeling**
- **CAD/Thermal Modeling**
- **Manufacturing Modeling**

**Diagram Elements**
- SW
- Physics-based
- Structure/CAD/Mfg
Example: Model Integration for Lumped Parameter Dynamics

- Verification Tools:
  - QR
  - RA
  - Analysis

- Multimodel Simulation Tools:
  - NS3
  - OMNET
  - Delta-3D
  - CPN

- Equations
  - Modelica-XML

- FMU-ME
  - S-function
  - FMU CS

- Distributed Multimodel Simulation

- Composition
  - CT
  - Power Port
  - Signal
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

Speciation 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**

<table>
<thead>
<tr>
<th>Languages</th>
<th>Models</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>GReAT-Model Transf. Language MetaGME++</td>
<td>FORMULA Consistency Proofs</td>
<td>MetaGME++-2 FORMULA</td>
</tr>
<tr>
<td>FORMULA ASML</td>
<td>FORMULA ASML</td>
<td>GReAT (FORMULA)</td>
</tr>
</tbody>
</table>

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

OCL Constraints:
\[
\text{self.transTo->forAll(s | s <> self)}
\]

Basic metamodeling notation: UML Class Diagram + OCL

MetaGME metamodel of simple statecharts

Model-editor generated from metamodel
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.

\[
L = \langle Y, R_Y, C, \left\{ [ ] \right\}_{i \in I} \rangle \\
D(Y, C) = \{ r \in R_Y | r \models C \} \\
[ ]: R_Y \mapsto R_Y
\]
Behavioral Semantics

- Given a DSML

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

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

\[ [ ]: R_Y \mapsto R_Y \]

- 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

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

\[
\left[ \right] : R_y \mapsto R_{y'}
\]

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

\[
\left[ \right] : R_{y'} \mapsto R_{y''}
\]

Heterogeneous math domain; Operational semantics

Reasonable tool support; Easy to understand

C++ Interpreter/Generator

Executable Model (Simulators)

Executable Code

Executable Specification

Graph rewriting rules

Graph transformation semantics

Representation as AST
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

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

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

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

Single math framework

```
1  domain AcausalBG_elements
2  {
3    primitive SF ::= (id: String).
4    primitive Se ::= (id: String).
5    primitive R ::= (id: String).
6    //...
7    primitive TF ::= (id: String).
8    primitive GY ::= (id: String).
9    primitive ZeroJunction ::= (id: String).
10   primitive OneJunction ::= (id: String).
11   Source ::= SF + Se.
12   //..
13  }
14
1  transform B6_DenotationalSemantics
2  from in1::AcausalBG
3  to out1::DAEquations
4  {
5    Eq(ea, px) :- x is Se, Src(a, x).
6    Eq(fa, px) :- x is Sf, Src(a, x).
7    Eq(ea, Mul(px, fa)) :- x is R, Dst(a, x).
8    DiffEq(ea, Mul(Inv(px), fa)) :-
9       x is C, Dst(a, x).
10   //...
11  }
12
1  domain DAEquations
2  {
3    primitive Variable ::= (name: String, id: String).
4    primitive Param ::= (id: String).
5    primitive Neg ::= (Term).
6    primitive Inv ::= (Term).
7    primitive DiffEq ::= (Variable, Term).
8    primitive SumZero ::= (Term).
9    Term ::= Variable + Param + Neg + Inv + Mul + Sum.
10   primitive Eq ::= (Variable, Term).
11   primitive DiffEq ::= (Variable, Term).
12   primitive SumZero ::= (Term).
13   Equation ::= Eq + DiffEq + SumZero.
14  }
```
<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>MetaGME</td>
<td>MetaGME-2-</td>
<td>GME</td>
<td>DSML spec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Formula</td>
<td>Constraint Checking</td>
<td>Constraint Checking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Metaprog.</td>
<td>Metaprog.</td>
</tr>
<tr>
<td>Transformation Modeling</td>
<td>UMTL</td>
<td>GReAT</td>
<td>Transf. spec.</td>
<td>Transf. spec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UDM</td>
<td>Compiling spec to</td>
<td>Compiling spec to transformer</td>
</tr>
<tr>
<td>Formal Metamodeling</td>
<td>Formula (MSR)</td>
<td>Domain Comp.</td>
<td>Domain Comp.</td>
<td>Domain Checking</td>
</tr>
<tr>
<td>Modeling</td>
<td></td>
<td>Semantic</td>
<td>Semantic units</td>
<td>Semantic units</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Anchoring</td>
<td></td>
<td>Semantics for complex DSMLs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Compositions</td>
</tr>
</tbody>
</table>
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).
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.
}

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)]
CausalStroke(A, B, DST) :- CausalStroke(B, A, SRC).
CausalStroke(A, B, OFF) :- CausalStroke(B, A, OFF).
}

partial model RC1 of ExtendedBondGraphWithCausality
{
JunctionState(oj1, "ON")
R1 is R("R1", "r1", "R1")
  C1 is C("C1", "c1", "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")
}
Semantic Specification
Example: Denotational Semantics

transform $T$ from BondGraph as $in_1$ to DiffEquations as $out_1$
{
  // Source of effort
  $out_1$ Equals($Effort(id)$, $Value(\text{effort})$) :- $Connection(id, Se(_, \text{effort}), _)$. \\

  // Source of flow
  $out_1$ Equals($Flow(id)$, $Value(\text{flow})$) :- $Connection(id, sf, Sf(_, \text{flow}))$. \\

  // Flow = Capacitance * derivative(Effort)
  $out_1$ Equals($Flow(id)$, $Mul$value(capacitance), $Derivative(Effort(id))$) :- $Connection(id, C(_, \text{capacitance}), _)$$Connection(id, _, C(_, \text{capacitance}))$. \\

  // kcl
  $out_1$ Equals($Flow(id_1)$, $Flow(id_2)$) :- $Succ(OneJunction(_), id_1, id_2)$. \\

  // kvl
  $out_1$ Equals($Effort(id_1)$, $Effort(id_2)$) :- $Succ(ZeroJunction(_), id_1, id_2)$. 
}

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.
}
Example: Formal Operational Semantics for Finite Automata

```plaintext
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(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

- **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?

(conceptualization)
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
    - .......

ADAPTOR

ADAPTED
Ongoing Work

- META and VF beta testing starts in 15 September 2012
- Trial design competition for undergraduates (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


- 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