Domain Specific Modeling Languages For Cyber Physical Systems: A Semantics Perspective

Janos Sztipanovits

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

CPS Summer School
Georgia Tech
27 June 2011
CPS is a rapidly emerging, cross-disciplinary field with well-understood and urgent need for **formal methods** driven by challenges in

- model-based design
- system verification and
- manufacturing
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
CPS is About Engineered Systems

<table>
<thead>
<tr>
<th>Sectors</th>
<th>Opportunities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health and Biomedical</td>
<td>In-home healthcare delivery. More capable biomedical devices for measuring health. New prosthetics for use within and outside the body. Networked biomedical systems that increase automation and extend the biomedical device beyond the body.</td>
</tr>
<tr>
<td>Smart Grid</td>
<td>Highway systems that allow traffic to become denser while also operating more safely. A national power grid that is more reliable and efficient.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sectors</th>
<th>Goals</th>
</tr>
</thead>
</table>
| Aerospace | • Aircraft that fly faster and further on less energy.  
            • Air traffic control systems that make more efficient use of airspace. |
| Automotive | • Automobiles that are more capable and safer but use less energy.  
               • Highways that are safe, higher throughput and energy efficient. |
| Defense  | • Fleets of autonomous, robotic vehicles  
               • More capable defense systems  
               • Integrated, maneuverable, coordinated, energy efficient  
               • Non-military applications |

Energy Internet: When IT Meets ET
Known Drivers of CPS

- Networking and Information Technology (NIT) have been increasingly used as *universal system integrator* in human – scale and societal – scale systems.
- Functionality and salient system characteristics emerge through the interaction of *networked physical and computational objects*.
- Engineered products turn into **Cyber-Physical Systems (CPS)**: networked interaction of physical and computational processes.
The Good News...

Networking and computing delivers precision and flexibility in **interaction** and **coordination**

<table>
<thead>
<tr>
<th>Computing/Communication</th>
<th>Integrated CPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rich time models</td>
<td>Elaborate coordination of physical processes</td>
</tr>
<tr>
<td>Precise interactions across highly extended spatial/temporal dimension</td>
<td>Hugely increased system size with controllable, stable behavior</td>
</tr>
<tr>
<td>Flexible, dynamic communication mechanisms</td>
<td>Dynamic, adaptive architectures</td>
</tr>
<tr>
<td>Precise time-variant, nonlinear behavior</td>
<td>Adaptive, autonomic systems</td>
</tr>
<tr>
<td>Introspection, learning, reasoning</td>
<td>Self monitoring, self-healing system architectures and better safety/security guarantees.</td>
</tr>
</tbody>
</table>
Fusing networking and computing with physical processes brings new unsolved problems.

### Computing/Communication
- Cyber vulnerability
- New type of interactions across highly extended spatial/temporal dimension
- Flexible, dynamic communication mechanisms
- Precise time-variant, nonlinear behavior
- Introspection, learning, reasoning

### Integrated CPS
- Physical behavior of systems can be manipulated
- Lack of composition theories for heterogeneous systems: much unsolved problems
- Vastly increased complexity and emergent behaviors
- Lack of theoretical foundations for CPS dynamics
- Verification, certification, predictability has fundamentally new challenges.
Modeling Layer

- **Systems Engineering**: Operation research, Reliability, Requirement spec., ...
- **Control Engineering**: Foundation of system theory: Linear, Nonlinear, ...
- **Software Engineering**: Formal methods, Model-based SE, RT software, ..
- **Communication Engineering**: Information theory, Layered protocols, ...

(Re)-convergence of Systems, Control, Software, Communication Engineering
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
Components of a CPS

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

Functional:
- Implement some function in the design
- Interconnect: acts as the facilitators for physical interactions

Computation and communication:
- Requires a physical platform to run/to communicate

Physical with deeply embedded computing and communication

Physical

Cyber

Cyber-Physical
CPS Design Flow Requires Model Integration

<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**
- **Low-Res Component Modeling**

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

- **Architecture Modeling**
- **Dynamics, RT Software, CAD, Thermal, …**
- **Detailed Domain Modeling**

Domain Specific Modeling Languages
## Example: Architecture Modeling

<table>
<thead>
<tr>
<th>Sublanguage / Capability</th>
<th>Formalism, Language Constructs, Examples</th>
<th>Usage</th>
</tr>
</thead>
</table>
| **Architecture Modeling** | [Diagram showing hierarchical module interconnects with components, interfaces, interconnects, parameters, and properties.] | Systems Architect  
  - Explore Design Space  
  - Derive Candidate Designs |
| **Design Space Modeling** | [Diagram showing hierarchically layered parametric alternatives with alternatives/options, parameters, and constraints.] | Systems Architect  
  - Define Design Space  
  - Define Constraints |
## 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 Engineers
- Model dynamics with Hybrid Bond Graphs
- Compose system dynamics

### Domain Engineers
- Design controller
- Processor allocate
- Platform Effects

### Component Engineers
- Processor Topology
- Sensor Actuator
- Software Assembly
- Allocation
### Example: Physical Structure and Manufacturing Modeling

#### Solid Modeling (CAD / Geometry)

<table>
<thead>
<tr>
<th>Structural Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined with Peer Roles:</td>
</tr>
<tr>
<td>- Axis</td>
</tr>
<tr>
<td>- Point</td>
</tr>
<tr>
<td>- Surface</td>
</tr>
<tr>
<td>- CAD Links</td>
</tr>
</tbody>
</table>

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

- Power Out (SAE #1)
- Power In (SAE #1)
- G9_Diesel
- VU_IIG_V1

#### Component Engineer
- Defines Structural Interface System Engineer
- Defines Architecture

#### Manufacturing Modeling

<table>
<thead>
<tr>
<th>Component Manuf. Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
</tr>
<tr>
<td>- Material</td>
</tr>
<tr>
<td>- Fab Proc</td>
</tr>
<tr>
<td>- Complexity</td>
</tr>
<tr>
<td>- Shape/Wt</td>
</tr>
<tr>
<td>OTS: Cost/unit Structural Interfaces</td>
</tr>
<tr>
<td>Fastener Types, Num#</td>
</tr>
</tbody>
</table>

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

- Fastener Type: Nuts/Bolts/Washers (Hand)
- Num/OfFasteners: 12
- FastenerDiameter: 0.4375
- FastenerPitch: 14
- FastenerEdgeDistance: 0

- MechanicalFasten
- AdhesiveFasten
- ElectronicsFasten
- WeldedFasten
Model Integration Challenge: Physics

Heterogeneity of 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

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

- **Plant Dynamics Models**
  - Abstractions: continuous time, functions, signals, flows, ...
  - Properties: stability, safety, performance

- **Controller Models**
  - \( B(t) = \kappa_p(B_1(t), ..., B_j(t)) \)

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

- **Software Component Code**
  - \( B(i) = \kappa_c(B_1(i), ..., B_k(i)) \)

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

- **Resource Management Models**
  - \( B(t_j) = \kappa_p(B_1(t_i), ..., B_k(t_i)) \)

Heterogeneity of Abstractions
A Pragmatic Approach: Model Integration Language

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

Structural Semantics
Generative Rules
MetaGME

Semantic Translators
CyPhy \(\leftrightarrow\) SL/SF
CyPhy \(\leftrightarrow\) SEER
CyPhy

Tools and Frameworks \(\rightarrow\) Assets / IP / Designer Expertise

Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
What Do We Expect From Formal Semantics?

- Specify
- Unambiguate
- Compute
Models represent:

- **Structure** (logical, physical, etc.)
- **Behavior** (continuous, discrete, etc.)

Mathematical Domains:
- graphs
- term algebra + logic

DSML Semantics:

- **Structural** (set of well-formed model structures)
- **Behavioral** (set of feasible behaviors)

Behavioral Semantics:
- denotational
- operational
Example 1/2

Physical Structure (components and terminals)

Transformation:

\[ m_{bg} = T(m_{ph}) \]
Example 2/2

operational: simulated trajectories

Simulink model of the system

m_{sl} = T(m_{bg})

denotational: mathematical equations

m_{de} = T(m_{bg})

\begin{align}
OJ1: & e_1 - e_2 - e_3 - e_4 = 0 \\
OJ1: & f_1 = f_2 = f_3 = f_4 \\
Se: & e_1 = E(t) \\
R_{arm}: & e_2 = R_{arm} \cdot f_2 \\
L_{arm}: & e_3 = L_{arm} \cdot \dot{f}_3 \\
GY: & e_5 = f_1 \cdot K_T \\
GY: & e_4 = f_2 \cdot K_{EMF} \\
OJ2: & e_5 - e_6 - e_7 = 0 \\
OJ2: & f_6 = f_7 = f_8 \\
R_{fric}: & e_6 = R_{fric} \cdot \dot{f}_6 \\
m: & e_7 = m \cdot \dot{f}_2
\end{align}
Modeling Language Semantics
Has Extensive Research History

- Broy, Rumpe ‘1997
- Harel ‘1998
- Harel and Rumpe ‘2000
- Tony Clark, Stuart Kent, Bernhard Rumpe, Kevin Lano, Jean-Michel Bruel and Ana Moreira - Precise UML Group
- Edward Lee, Alberto Sangiovanni-Vincentelli ‘2004
- Joseph Sifakis ‘2005
- ...
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
**Specification of Domain-Specific Modeling Languages**

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

Abstract syntax of DSML-s are defined by metamodels.

A metamodelling language is one of the DSML-s.

Semantics of metamodelling languages: structural semantics.

MetaGME metamodel of simple statecharts  Model-editor generated from metamodel
- Gives semantics to metamodels
- A domain D is given by
  - An alphabet Σ
  - A finite set of n-ary function symbols Y that describes the relations between members of the alphabet
  - A set of model realizations \( R_Y \) – a term algebra over Y generated by Σ
  - A set of constraints C such that \( r \in R_Y, r \not\in C \Rightarrow r \in D \)
- We denote D = (Σ, Y, R_Y, C)
Complex constraints cannot be captured by simple type systems. Common fix is to use a constraint language (e.g. OCL).

We use Logic Programming because:

- LP extends term algebra semantics while supporting declarative rules
- The fragment of LP supported is equivalent to full first-order logic over term algebras
- Unlike purely algebraic specs, there is a clear execution semantics for logic programs making it possible to specify model transformations in the same framework
- Many analysis techniques is available for LP.
Model realization that satisfies the domain constraints is simply called a model of a domain.

The decision procedure for domain constraints satisfaction is as follows:
- Represent the model realization as a logic formula $\Psi(r)$.
- Compute deductive closure of a sum of the formula $\Psi(r)$ and $C$.
- Examine the deductive closure to find if $r$ satisfies the domain rules.

Constraints are given as proofs:
- Positive domain: $r$ satisfies constraints if any wellform (. ) term can be derived.
- Negative domain: $r$ satisfies constraints if it is impossible to derive any malform (. ) term.
**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 term algebra extended with Logic Programming. This mathematical structure is the semantic domain of metamodeling languages.

**Use of structural semantics:**
- Conformance testing: \( x \in D \)
- Non-emptiness checking: \( D(Y, C) \neq \{\text{nil}\} \)
- DSML composing: \( D_1 \cdot D_2, D_1 + D_2 \) \( D' \) includes \( D \)...
- Model finding: \( S = \{s \in D \mid s \models P\} \)
- 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.

---

**Formalization of Structural Semantics**

\[
L = (Y, R_Y, C, \{ \lip_{i \in J} \})
\]

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

\[
\lip: R_Y \rightarrow R_Y
\]
GME-FORMULA Tool Interfaces

Generic Modeling Environment (ISIS)

Modeling Lngs
Constraint Defs
Relations among Modeling Lngs and Models
Models

Metamodel Translator
Formula Domain
Validation Tool
Analyzer Tool

Model Translator
Formula Model
FORMULA (Microsoft Research)
Constraint definition in Formula
Generated Domain (SLP)
Generated Model (SLP)
Checking constraints
FORMULA (Schulte, Jackson et al, MSR) - A tool suite for building models and analyzing their properties. Co-developed with the European Microsoft Innovation Center (EMIC), Aachen, Germany

GME-FORMULA translator – Extension of the MIC tool suite (VU-ISIS in cooperation with MSR)

Analysis tools – Domain and Model Equivalence, Domain Composition, Model Completion (VU-ISIS in cooperation with MSR)
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
**Behavioral Semantics**

- Given a DSML

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

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

\[
[ ]: R_Y \mapsto R_Y
\]

- Behavioral semantics will be defined by specifying the transformation between the DSML and a modeling language with behavioral semantics.
Implicit Methods for Specifying Behavioral Semantics

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

\[
\begin{bmatrix}
\end{bmatrix} : R_Y \mapsto R_{Y'}
\]

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

\[
\begin{bmatrix}
\end{bmatrix} : R_{Y'} \mapsto R_{Y''}
\]

Representation as AST

C++ Interpreter/Generator

Graph rewriting rules

Executable Model (Simulators)

Executable Code

Executable Specification
Explicit Methods for Specifying Behavioral Semantics

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

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

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

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

Representation as AST

Explicit

C++ Interpreter/Generator

Graph rewriting rules

Executable Model (Simulators)

Executable Code

Executable Specification
Specifying Behavioral Semantics With Semantic Anchoring

**Abstract State Machine Formalism**

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

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

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

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

**Representation as AST**

**MIC-UDM**

**MIC-GME**

**MIC-GReAT** (Karsai, VU-ISIS)

**Graph rewriting rules**

**Abstract Data Model**

**Model Interpreter**
Example Specification: FSM

Abstract Data Model

```csharp
structure Event
    eventType as String

class State
    initial as Boolean
    var active as Boolean = false

class Transition

abstract class FSM
    abstract property states as Set of State
    get
    abstract property transitions as Set of Transition
    get
    abstract property outTransitions as Map of <State, Set of Transition>
    get
    abstract property dstState as Map of <Transition, State>
    get
    abstract property triggerEventType as Map of <Transition, String>
    get
    abstract property outputEventType as Map of <Transition, String>
    get
```

Interpreter

```csharp
abstract class FSM
    Run (e as Event) as Event?
    step
        let CS as State = GetCurrentState ()
        step
        let enabledTs as Set of Transition = {t | t in outTransitions (CS) where e.eventType = triggerEventType(t)}
        step
        if Size (enabledTs) >= 1 then
            choose t in enabledTs
            step
            CS.active := false
            step
            dstState(t).active := true
            step
            if t in me.outputEventType then
                return Event(outputEventType(t))
            else
                return null
        else
            return null
```

Underlying abstract machine - ASM Language: AsmL
Yuri Gurevich, MSR
Semantic anchoring of DSMLs using “semantic units”
Compositional specification of semantics for heterogeneous modeling languages
Investigating alternative frameworks (e.g. based on FORMULA)
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
Capturing Physical Semantics

Modeling Language Semantics:

Structural
(set of well-formed model structures)

Physical
(set of feasible behaviors)

Behavioral
(set of feasible behaviors)

Rational:
• Get the physics right
• The rest is mathematics (Kalman, 2005)

• denotational

• operational
Energy is conserved at couplings between domains.
Heat energy generated on dissipative elements: creates additional energy coupling.
Physical Semantics: Behavioral Implications

One Junction Rule

\[ \sum_{i} e_i = 0 \]
\[ f_i = f_k; i, k \in N \]

Rate of power transfer between components is balanced

Denotational behavioral semantics

Denotational behavioral semantics

- \( OJ1: e_1 - e_2 - e_3 - e_4 = 0 \)
- \( OJ1: f_1 = f_2 = f_3 = f_4 \)

\( Se: e_1 = E(t) \)
\( R_{arm}: e_2 = R_{arm} \times f_2 \)
\( L_{arm}: e_3 = L_{arm} \times f_3 \)
\( GY_1: e_4 = f_3 \times K_T \)
\( GY_1: e_4 = f_2 \times K_{GMF} \)

Denotational behavioral semantics

- \( OJ2: e_5 - e_6 - e_7 = 0 \)
- \( OJ2: f_5 = f_6 = f_7 \)

\( R_{fric}: e_8 = R_{fric} \times f_8 \)
\( m: e_7 = m \times f_2 \)
Physical Semantics: Ongoing Work

- Extend metamodeling language and metaprogrammable modeling tool (GME) with generative constructs
- Make specification of generative modeling constructs integrated with metamodeling
- Extend structural semantics and tools with dynamic constructs
- Develop rule libraries for relevant cross-physical domains
Overview

- Cyber-Physical Systems (CPS)
  - CPS and Domain Specific Modeling Languages
  - Model Integration Challenge
- Formal Semantics of DSMLs
  - Structural Semantics
  - Behavioral Semantics
- Practical Use of Formal Semantics
  - Addressing Horizontal Heterogeneity
  - Addressing Vertical Heterogeneity
- Summary
Integration Inside Abstraction Layers: Composition

**Plant Dynamics Models** ↔ **Controller Models**

**Properties**: stability, safety, performance

**Abstractions**: continuous time, functions, signals, flows,…

**Dynamics**: \( B(t) = \kappa_p (B_1(t), ..., B_j(t)) \)

**Software Design**

**Software Architecture Models** ↔ **Software Component Code**

**Properties**: deadlock, invariants, security,…

**Abstractions**: logical-time, concurrency, atomicity, ideal communication,…

**Software**: \( B(i) = \kappa_c (B_1(i), ..., B_k(i)) \)

**System/Platform Design**

**System Architecture Models** ↔ **Resource Management Models**

**Properties**: timing, power, security, fault tolerance

**Abstractions**: discrete-time, delays, resources, scheduling,…

**Systems**: \( B(t_j) = \kappa_p (B_1(t_i), ..., B_k(t_i)) \)
**Integration Across Abstraction Layers: Much Unsolved Problems**

Controller dynamics is developed without considering implementation uncertainties (e.g. word length, clock accuracy) optimizing performance.

*Assumption:* Effects of digital implementation can be neglected

Software architecture models are developed without explicitly considering systems platform characteristics, even though key behavioral properties depend on it.

*Assumption:* Effects of platform properties can be neglected

System-level architecture defines implementation platform configuration. Scheduling, network uncertainties, etc. are introduce time variant delays that may require re-verification of key properties on all levels.
Dealing With Leaky Abstractions

- Leaky abstractions are caused by lack of composability across system layers.

Consequences:
- intractable interactions
- unpredictable system level behavior
- full-system verification does not scale

Solution: simplification strategies
- Decoupling: Use design concepts that decouple systems layers for selected properties
- Cross-layer Abstractions: Develop methods that can handle effects of cross-layer interactions
Example for Decoupling: Passive Dynamics

Goals:

- Effect of “leaky abstraction”: loss of stability due to implementation-induced time delays (networks, schedulers)
- *Passivity* of dynamics decouples stability from time varying delays
- *Compositional* verification of essential dynamic properties
  - stability
  - safety
- Hugely decreased verification complexity
- Hugely increased flexibility
Passivity-based Design and Modeling Languages 1/4

Modeling Language Semantics:

- **Structural** (set of well-formed model structures)
- **Behavioral** (set of feasible behaviors)

Heterogeneous Abstractions (stability)

Physical (struct. and behav. constraints)

Fix for stability:
- Passivity-based design

Structural constraints are more involved (next page)

\[ \dot{x} = f(x,u) \bigg|_{t_1}^{t_2} \int_{t_1}^{t_2} u^T(t)y(t)dt + V(x(t_1)) \geq V(x(t_2)) \]

for all \( t_2 \geq t_1 \) and the input \( u(t) \in U \)  [Antsaklis '2008]
Constrain modeling language with constructs below:

\[ u_{pk}(i) = \frac{1}{\sqrt{2b}}(b\theta_{pk}(i) + \tau_{dck}(i)) \]

\[ v_{c1}(j) = \frac{1}{\sqrt{2b}}(b\theta_{dp1}(j) - \tau_{c1}(j)) \]

**Bilinear transform:**
- Power and wave vars.

- **Bilinear transform** (b)
- Power and Wave variables
- Passive down- and up-sampler (PUS, PDS)

- Delays
- Power junction
- Passive dynamical system  

[Kottenstette '2011]
Passivity-based Design and Modeling Languages 3/4

Constrain modeling language with composition constraints below:

- Negative feedback interconnection of two passive systems is passive.
- Parallel interconnection of two passive systems is still passive.

Extensive research in the VU/ND/UMD NSF project toward correct-by-construction design environments (where correct-by-construction means what the term suggest).
For LTI passive systems, we can always assume quadratic storage function

\[ V(x) = \frac{1}{2} x^T P x \text{ where } P = P^T > 0. \]

For continuous-time system this leads to the following LMI

\[
\begin{bmatrix}
A^T P + PA & PB - C^T \\
B^T P - C & -D - D^T
\end{bmatrix} \leq 0
\]

In discrete-time the LMI becomes the following

\[
\begin{bmatrix}
A^T PA - P & A^T PB - C^T \\
B^T PA - C & B^T PB - D - D^T
\end{bmatrix} \leq 0
\]

[Antsaklis ‘2008]
Penetration of networking and computing in engineered systems forces a grand convergence across engineering disciplines.

Signs of this convergence presents new opportunities and challenges for formal methods research:

- New foundation for model integration – emergence of metaprogrammable tool suites and multi-modeling
- Embedding physical semantics in modeling languages

Model-based design facilitates a necessary convergence among software, system, control and network engineering.
- 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. (Accepted for publication)
- Xenofon Koutsoukos, Nicholas Kottenstette, Joe Hall, Emeka Eyisi, Heath Leblanc, Joseph Porter and Janos Sztipanovits, “A Passivity Approach for Model-Based Compositional Design of Networked Control Systems”, ACM Transactions on Computational Logic. (Accepted for publication)