Domain Specific Modeling Languages for Cyber Physical Systems: Where Are Semantics Coming From?

Janos Sztipanovits
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN 37221
Email: janos.sztipanovits@vanderbilt.edu

http://www.artist-embedded.org/
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>

**Sectors**

<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  
- Resistant to cyber attacks |
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**

#### Computing/Communication
- Rich time models
- Precise interactions across highly extended spatial/temporal dimension
- Flexible, dynamic communication mechanisms
- Precise time-variant, nonlinear behavior
- Introspection, learning, reasoning

#### Integrated CPS
- Elaborate coordination of physical processes
- Hugely increased system size with controllable, stable behavior
- Dynamic, adaptive architectures
- Adaptive, autonomic systems
- Self monitoring, self-healing system architectures and better safety/security guarantees.
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.
Control Systems package org.apache.tomcat.session;
import org.apache.tomcat.core.*;
import org.apache.tomcat.util.StringManager;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Core implementation of a server session
 *
 * @author James Duncan Davidson [duncan@eng.sun.com]
 * @author James Todd [gonzo@eng.sun.com]
 */
public class ServerSession {
    private StringManager sm = StringManager.getManager("org.apache.tomcat.session");
    private Hashtable values = new Hashtable();
    private Hashtable appSessions = new Hashtable();
    private String id;
    private long creationTime = System.currentTimeMillis();
    private long thisAccessTime = creationTime;
    private long lastAccessed = creationTime;
    private int inactiveInterval = -1;

    public ServerSession(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public long getCreationTime() {
        return creationTime;
    }

    public long getLastAccessedTime() {
        return lastAccessed;
    }

    public ApplicationSession getApplicationSession(Context context, boolean create) {
        ApplicationSession appSession = (ApplicationSession)appSessions.get(context);
        if (appSession == null && create) {
            // XXX
            // sync to ensure valid?
            appSession = new ApplicationSession(id, this, context);
            appSessions.put(context, appSession);
        }
        // XXX
        // make sure that we haven't gone over the end of our
        // inactive interval -- if so, invalidate and create
        // a new appSession
        return appSession;
    }

    void removeApplicationSession(Context context) {
        appSessions.remove(context);
    }

    /**
     * Called by context when request comes in so that accesses and
     * inactivities can be dealt with accordingly.
     */
    void accessed() {
        // set last accessed to thisAccessTime as it will be left over
        // from the previous access
        lastAccessed = thisAccessTime;
        thisAccessTime = System.currentTimeMillis();
    }

    void validate()

    (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 span:
- Multiple physics
- Multiple domains
- Multiple tools

**Physical**
- Functional: implements some function in the design
- Interconnect: acts as the facilitators for physical interactions

**Cyber**
- Computation and communication that implements some function
- Requires a physical platform to run/to communicate

**Cyber-Physical**
- Physical with deeply embedded computing and communication
CPS Design Flow Requires Model Integration

<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**      | Hierarchical Module Interconnect - Components - Interfaces - Interconnects - Parameters - Properties       | Systems Architect
|                                |                                                                                                          |   - Explore Design Space                   |
|                                |                                                                                                          |   - Derive Candidate Designs               |
| **Design Space Modeling**      | Hierarchically Layered Parametric Alternatives - Alternatives/Options - Parameters - Constraints          | Systems Architect
|                                |                                                                                                          |   - Define Design Space                    |
|                                |                                                                                                          |   - Define Constraint                      |
Example: Dynamics Modeling

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

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)

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

### Manufacturing Modeling

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

- **Structural Interfaces**
  - Fastener Types, Num# …

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

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

### Example: Physical Structure and Manufacturing Modeling

<table>
<thead>
<tr>
<th>Component Engineer</th>
<th>Component Engineer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines Structural Interface System</td>
<td>Defines Part Cost</td>
</tr>
<tr>
<td>Defines Architecture</td>
<td>Defines Structural Interface, Fastener</td>
</tr>
</tbody>
</table>

- **Fastener Type:** Nuts/Bolts/Washers (Hand)
- **Number of Fasteners:** 12
- **Fastener Diameter:** 0.4375
- **Fastener Pitch:** 14
- **Fastener Edge Distance:** 0

---

**Example**

- Physical Structure and Manufacturing Modeling
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:** \( B(t_j) = \kappa_p(B_1(t_j),...,B_k(t_j)) \)
- *Properties*: timing, power, security, fault tolerance
- *Abstractions*: discrete-time, delays, resources, scheduling,
A Pragmatic Approach: Model Integration Language

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, ...)

**Behavior**
(cont. discrete, ...)

**DSML Semantics**

Models represent:

**Structure**
(set of well-formed model structures)

**Behavioral**
(set of feasible behaviors)

**Behavioral**
(set of feasible behaviors)

**Mathematical Domains:**
- graphs
- term algebra + logic

**Modeling Language Semantics:**

**Structural**

- denotational

- operational
Example 1/2

Physical Structure (components and terminals)

Transformation:

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

Bond Graph model (energy flows)
Example 2/2

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

**Operational**: simulated trajectories

**Simulink model of the system**

**Denotational**: mathematical equations

### Equations:

1. Objective (OF1)
   - O1: \( e_1 - e_2 - e_3 - e_4 = 0 \)
   - O2: \( f_1 = f_2 = f_3 = f_4 \)

2. Subjective (S)
   - Se: \( e_4 = B(t) \)

3. Constraints (C)
   - R: \( e_5 = f_2 * K_T \)
   - G: \( e_4 = f_2 * K_{EMF} \)

4. Objective (OF2)
   - O2: \( e_5 - e_6 - e_7 = 0 \)
   - O2: \( f_5 = f_6 = f_7 \)

5. Friction (R_fric)
   - \( f_6 = R_{fric} * f_6 \)

6. Mass (m)
   - \( e_7 = m * f_2 \)
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
**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 metamodeling language is one of the DSML-s.

Semantics of metamodeling languages: structural semantics.

OAuth Constraints:
\[
\text{self.transTo->forall}(s \mid s \neq \text{self})
\]

Basic metamodeling notation: UML Class Diagram + OCL

MetaGME metamodel of simple statecharts

Model-editor generated from metamodel
Gives semantics to metamodels

A domain $D$ is given by

- An alphabet $\Sigma$
- A finite set of $n$-ary function symbols $\Upsilon$ that describes the relations between members of the alphabet
- A set of model realizations $R_\Upsilon$ – a term algebra over $\Upsilon$ generated by $\Sigma$
- A set of constraints $C$ such that $r \in R_\Upsilon, r \models C, \rightarrow r \in D$

We denote $D = (\Sigma, \Upsilon, R_\Upsilon, C)$
- 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: \( D_1 \ast D_2 \mid D_1 + D_2 \mid D' \text{ includes } D \mid ... \)
- 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 = \langle Y, R_Y, C, \{ \[ \] \}_{\mathcal{J}} \rangle \\
D(Y,C) = \{ r \in R_Y \mid r \models C \} \\
[ \[ \]]: \mathcal{R}_Y \mapsto \mathcal{R}_Y.
\]

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

Jackson & Sz. ’2007
Jackson, Schulte, Sz. ’2008
Jackson & Sz. ‘2009
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)
Example

```plaintext
domain DFA {
  primitive Event ::= (lbl: Integer).
  primitive State ::= (lbl: Integer).
  [Closed(src, trg, dst)]

  primitive Transition ::= (src: State, trg: Event, dst: State).
  [Closed(st)]
  primitive Current ::= (st: State).
  nonDeterTrans ::= Transition(s, e, sp), Transition (s, e, tp), sp != tp.
  conforms ::= !nonDeterTrans.
}
```

```
model A1 of DFA {
  e1 is Event(1) e2 is Event(2)
  s1 is State(1) s2 is State(2)
  Transition(s1, e1, s2)
  Transition(s2, e2, e1)
  Current(s1)
}
Ongoing Work

- 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
Given a DSML

\[ L = \langle Y, R_Y, C, \left[ \_ \right] \rangle_{i \in J} \]

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

\[ \left[ \_ \right] : R_Y \mapsto R_Y \]

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

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

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

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

\[
\begin{bmatrix} \text{[ ]} \end{bmatrix} : R_{Y'} \mapsto R_{Y''}
\]
Explicit Methods for Specifying Behavioral Semantics

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

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

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

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

Representation as AST

Explicit

C++ Interpreter/Generator

Graph rewriting rules

Executable Model (Simulators)

Executable Code

Executable Specification
Specifying Behavioral Semantics
With Semantic Anchoring

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

\[ \square: R_Y \rightarrow R_{Y'} \]

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

\[ \square: R_{Y'} \rightarrow R_{Y''} \]

Representation as AST

MIC-UDM
MIC-GME

Graph rewriting rules

MIC-GReAT (Karsai, VU-ISIS)

Abstract State Machine Formalism

Abstract Data Model
Model Interpreter
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

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
Ongoing Work

- Semantic anchoring of DSMLs using “semantic units”
- Compositional specification of semantics for heterogeneous modeling languages
- Investigating alternative frameworks (e.g. based on FORMULA)
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)
- **Behavioral**: (set of feasible behaviors)
  - denotational
  - operational

Rational:
- Get the physics right
- The rest is mathematics (Kalman, 2005)
Energy is conserved at couplings between domains
Physical Semantics: Structural Implications 2/2

Collateral energy flow
…other rules…

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

One Junction Rule

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

- \( S \Theta e_0 = E(t) \)
- \( R_{arm} e_2 = R_{arm} f_2 \)
- \( l_{arm} e_3 = l_{arm} f_3 \)
- \( G_Y: e_5 = f_1 \times K_T \)
- \( G_Y: e_6 = f_2 \times K_{EMF} \)

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

- \( R_{fric} e_6 = R_{fric} f_6 \)
- \( m: e_7 = m \times f_2 \)
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**

**Physical design**

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

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

**Software design**

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

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

**System/Platform Design**

**Systems:** \( B(t_j) = \kappa_p(B_1(t_i), \ldots, B_k(t_i)) \)
- **Properties:** timing, power, security, fault tolerance
- **Abstractions:** discrete-time, delays, resources, scheduling,
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.
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)
  - Structural constraints are more involved (next page)

- **Behavioral**
  - (set of feasible behaviors)

- **Denotational**

Fix for stability:
- Passivity-based design

\[
\dot{x} = f(x,u) \\
y = h(x,u).
\]

\[
\int_{t_1}^{t_2} u^T(t)y(t)dt + V(x(t_1)) \geq V(x(t_2))
\]

for all \( t_2 \geq t_1 \) and the input \( u(t) \in U \)

[Antsaklis ‘2008]
Passivity-based Design and Modeling Languages 2/4

Constrain modeling language with constructs below:

\[
\begin{align*}
\mathbf{u}_{pk}(i) &= \frac{1}{\sqrt{2b}}(b\theta_{pk}(i) + \tau_{dck}(i)), \\
\mathbf{v}_{c1}(j) &= \frac{1}{\sqrt{2b}}(b\theta_{dp1}(j) - \tau_{c1}(j))
\end{align*}
\]

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]
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).
Constrain modeling language behavior with these constraints (for LTI)

- For LTI passive systems, we can always assume quadratic storage function

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

- 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
References

- Jackson, E., Sztipanovits, J.: ‘Formalizing the Structural Semantics of Domain-Specific Modeling Languages,”
  *Journal of Software and Systems Modeling* pp. 451-478, September 2009
- Jackson, Thibodeaux, Porter, Sztipanovits: “Semantics of Domain-Specific Modeling Languages,”
- 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 Eiysi, 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)