MODEL-INTEGRATED DESIGN IN SOFTWARE, SYSTEMS AND CONTROL ENGINEERING

Janos Sztipanovits
ISIS, Vanderbilt University

ISIS

- Established by the School of Engineering of Vanderbilt University in 1998
- Academic/professional research organization
- Personnel:
  - 38 Research Scientists & Staff Engineers
  - 7 Faculty (EECS)
  - 6 Admin Staff
  - 50+ Graduate students

Overview

- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

Overview

- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

Trends in Systems Industry

Sectors	Opportunities	
Transportation	Aircraft that fly faster and further on less energy. Air traffic control systems that make more efficient use of airspace. Automobiles that are more capable and safer but use less energy.	
Defense	More capable defense systems; defense systems that make better use of networked fleets of autonomous vehicles.	
Energy and Industrial Automation	New and renewable energy sources. Homes, office, buildings and vehicles that are more energy efficient and cheaper to operate.	

What Are the Drivers of These Trends?

- 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

Why Is CPS Significant?

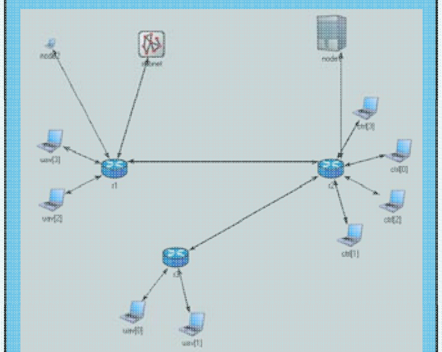
- The share of value of embedded computing components in different industries:

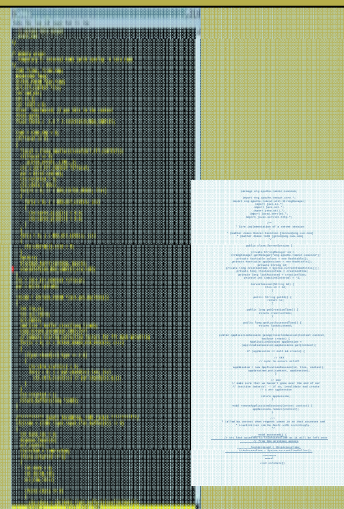
	2003	2009
Automotive and airspace systems	52%	56%
Aerospace	52%	54%
Health/Medical equipment	50%	52%
Industrial automation	43%	48%
Telecommunications	56%	58%
Consumer electronics and Intelligent Homes	60%	62%

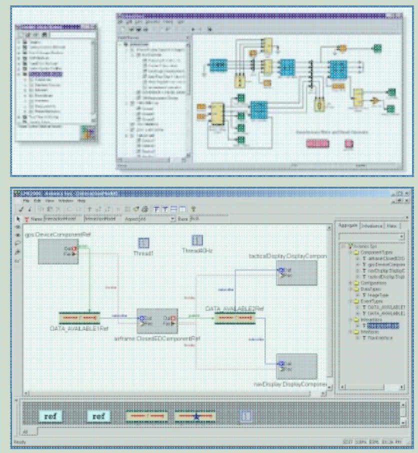
Source: "Study of Worldwide Trends and R&D Programm*es* in Embedded Systems in View of Maximising the Impact of a Technology Platform in the Area" EU Commission, 2005

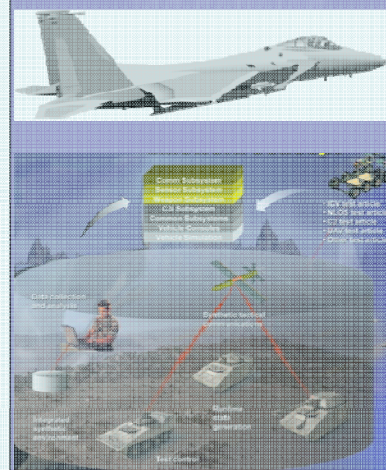
Why is CPS Hard?

Network	Software	Control	Systems





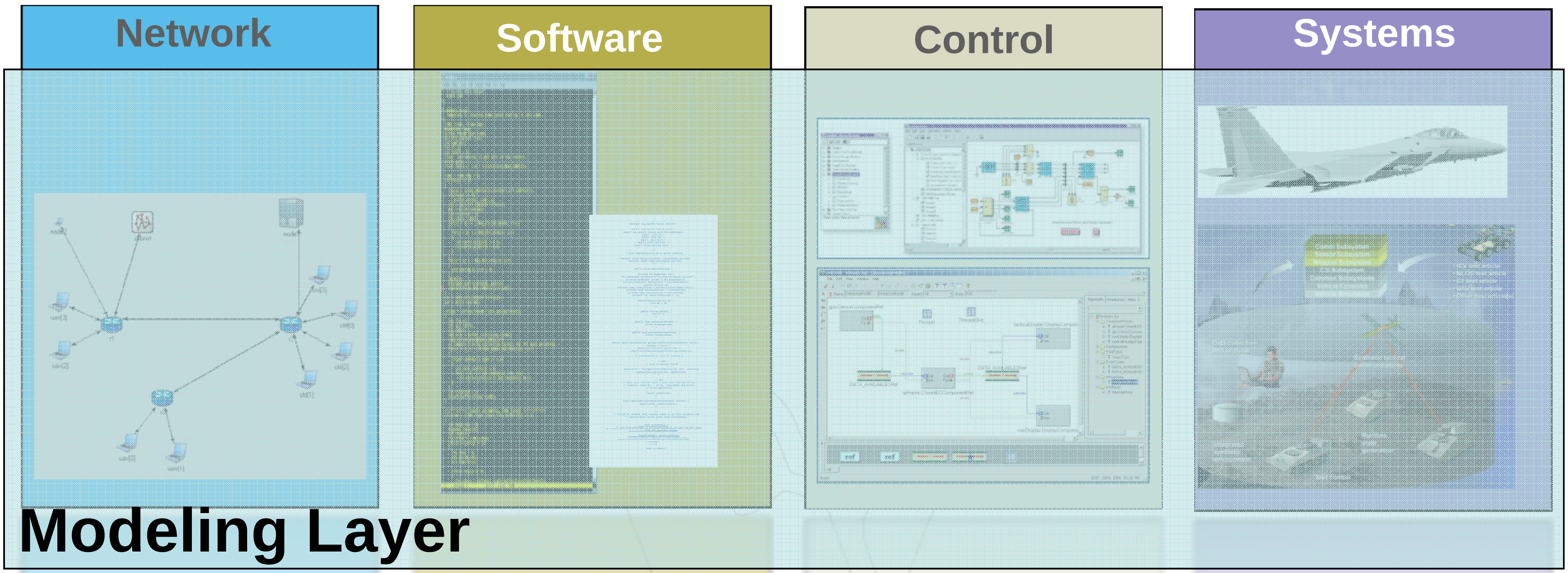




Crosses Interdisciplinary Boundaries

- Disciplinary boundaries need to be realigned
- New fundamentals need to be created
- New technologies and tools need to be developed
- Education need to be restructured

Foundation for Convergence: Model-Based Design



- **Systems Engineering**: Model-based design has been the state of practice
- **Control Engineering**: Wide acceptance (MathWorks Simulink/StateFlow)
- **Software Engineering**: Increasing acceptance due to OMG's MDA push and wider availability of tool suites
- **Network Engineering:** modeling networks in abstraction layers (TCP/IP), research linking structural and behavioral properties

Overview

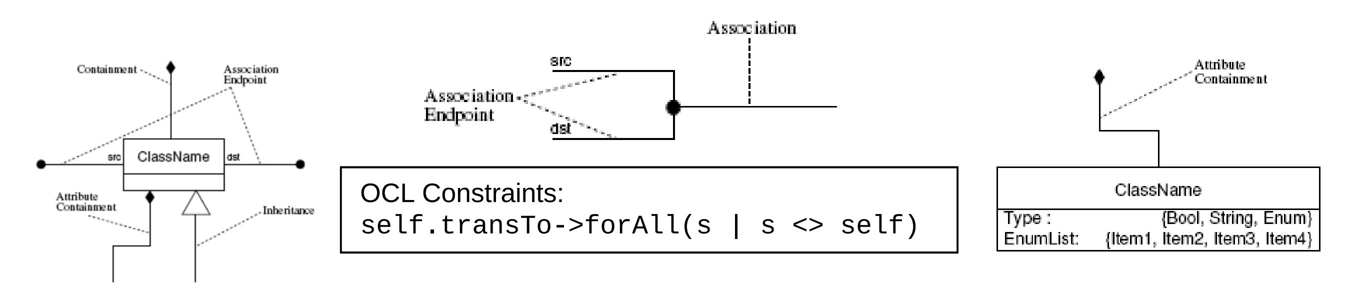
- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

Define Domain-Specific Modeling Languages

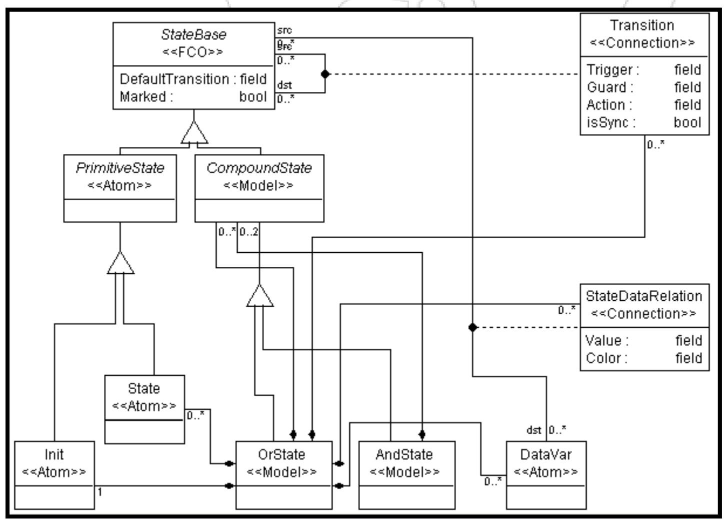
Key Concept: Modeling languages define a set of well- formed models and their interpretations. The interpretations are mappings from one domain to another domain.

Abstract syntax of DSML-s are defined by metamodels.

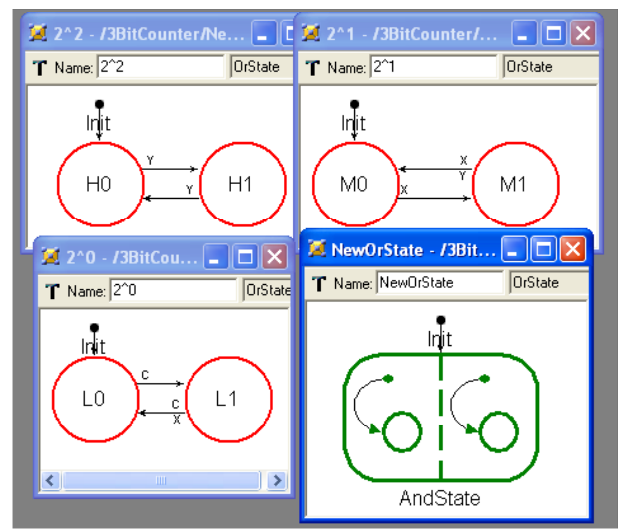
A metamodeling language is one of the DSML-s: the same tool can be used for modeling and metamodeling.



Basic metamodeling notation: UML Class Diagram + OCL



MetaGME metamodel of simple statecharts



Model-editor generated from metamodel

Use Precise Structural Semantics…

$$L = \langle Y, R_Y, C, ([\ ]_i)_{i \in J} \rangle$$
$$D(Y,C) = \{ r \in R_Y \mid r \models C \}$$
$$[\ ] : R_Y \mapsto R_{Y'}$$

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 & Sztipanovits
- EMSOFT 2006
- MODELS 2007
- SOSYM 2009

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 a mathematical structure. This mathematical structure is the semantic domain of metamodeling languages.

Arguments for investigating structural semantics:

- Conformance testing: $x \in D$
- Non-emptiness checking: $D(Y,C) \neq \{nil\}$
- DSML composing: $D_1 * D_2 \,|\, D_1 + D_2 \,|\, D' \text{ includes } D \,|\ldots$
- Model finding: $S = \{ s \in D \,|\, s \models P \}$
- Transforming: $m' = T(m);\ m' \in X;\ m \in Y$

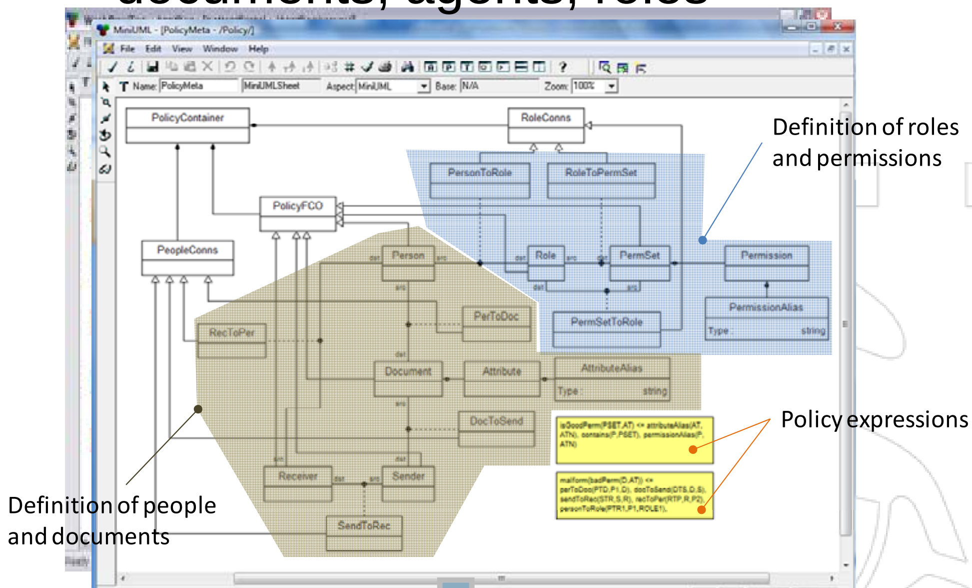
Notes on the selected formalism:

- Term algebra semantics extended with Logic Programming (LP)
- Fragment of LP is equivalent to full first-order logic
- Provide semantic domain for model transformations.

Example Application: Policy Aware Health Information Systems

Models of information flows, documents, agents, roles

Models of privacy policies (HIPAA)



Nurses should tag health questions

G $\forall$p, q, s, m. inrole(p, nurse) $\wedge$ send(p, q, m)
$\wedge$ contains(m, s, health-question)
$\Rightarrow$ tagged(m, s, health-question)

Doctors should answer health ques.

G $\forall$p, q, s, m. inrole(p, doctor) $\wedge$ send(q, p, m)
$\wedge$ contains(m, s, health-question)
$\Rightarrow$ **F** $\exists$m'. send(p, s, m')
$\wedge$ contains(m', s, health-answer)

Werner, Mathe Sztipanovits, 2009

Mitchell et al, 2006

Common Semantic Domain

Semantic domain for policies and information models are matched:

- structural constraints on models -> structural semantics
  (these policies can be expressed in the context of models using OCL
- policy models temporal constraints on system behavior -> behavioral semantics + LTL
- the generated system controls information flows and monitors policy violations

Overview

- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

Use Precise Behavioral Semantics

- Given a DSML

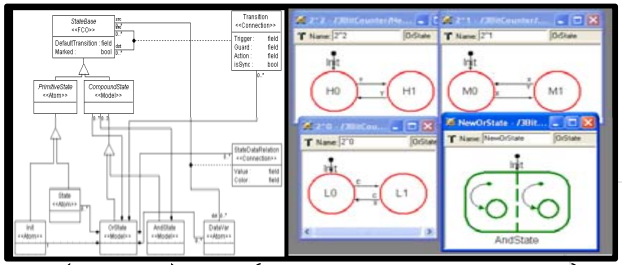
$$L = \langle Y, R_Y, C, ([\ ]_i)_{i \in J} \rangle$$

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

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

- 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

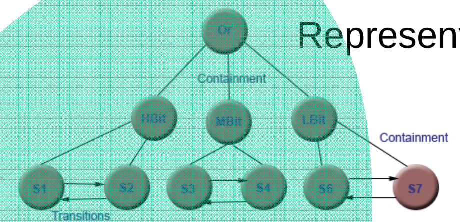


$$D(Y,C) = \{ r \in R_Y \mid r \models C \}$$

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

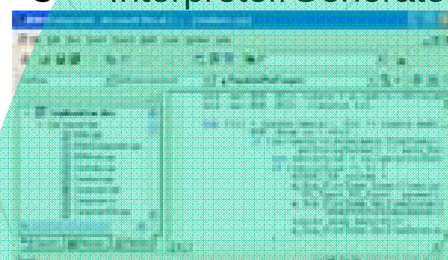
$$D(Y',C') = \{ r \in R_{Y'} \mid r \models C' \}$$
$$[\ ] : R_{Y'} \mapsto R_{Y''}$$

Representation as AST

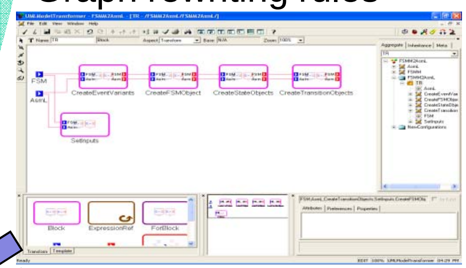


Implicit

C++ Interpreter/Generator



Graph rewriting rules

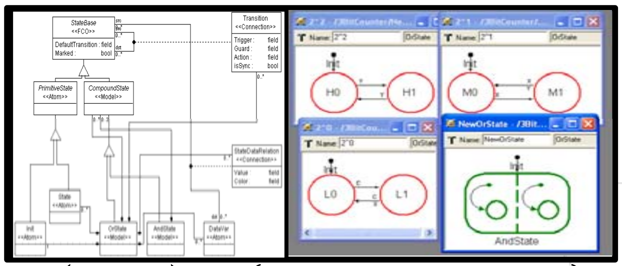


Executable Model (Simulators)

Executable Code

Executable Specification

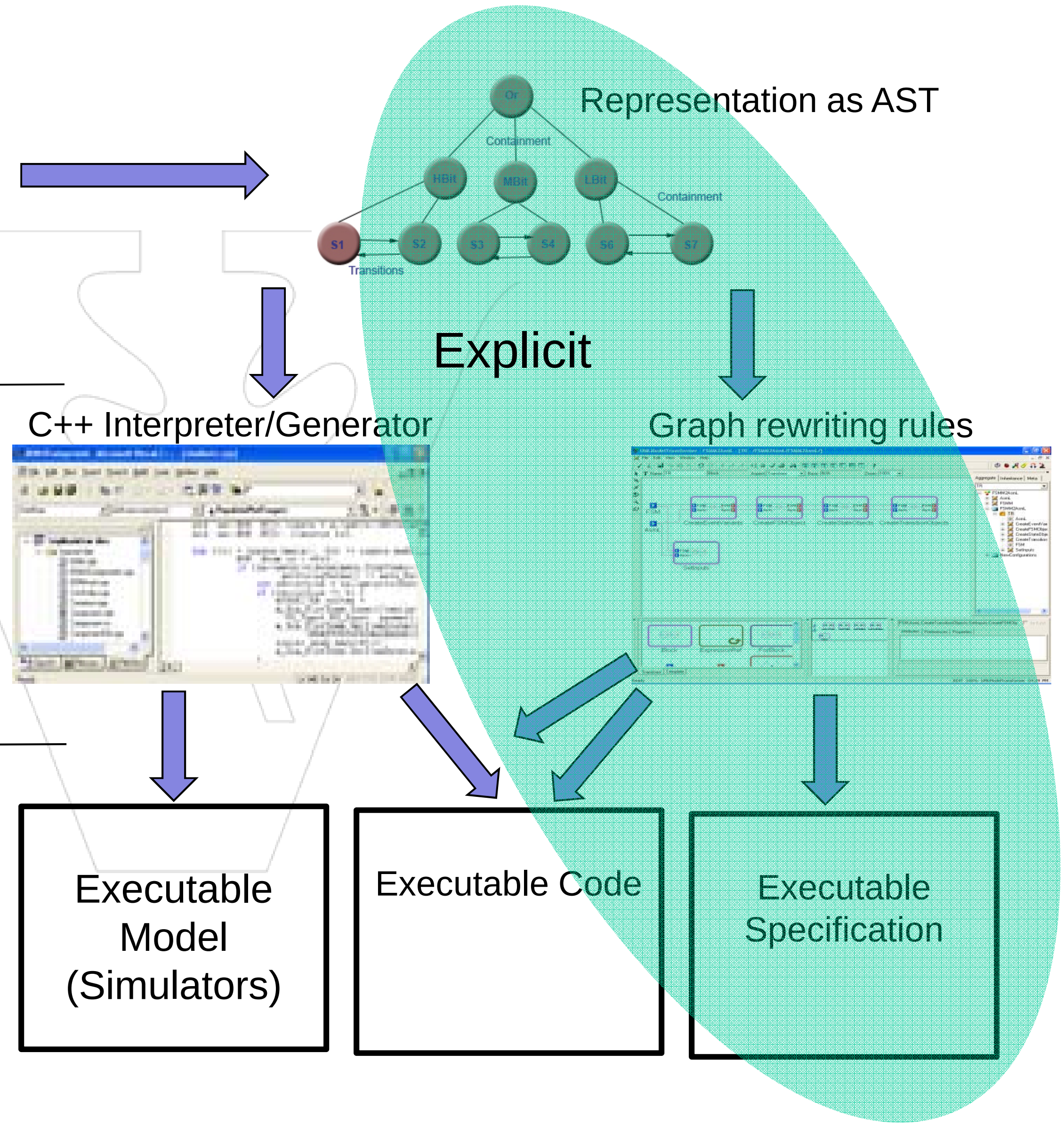
Explicit Methods for Specifying Behavioral Semantics



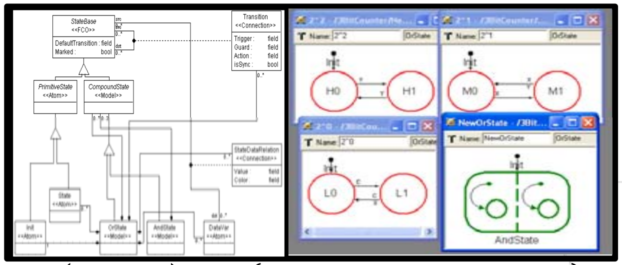
$$D(Y,C)=\{r\in R_Y \mid r\models C\}$$

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

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

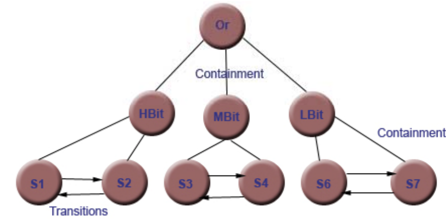


Specifying Behavioral Semantics With Semantic Anchoring



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

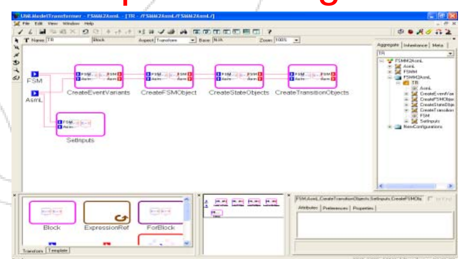
Representation as AST



MIC-UDM
MIC-GME

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

Graph rewriting rules



MIC-GReAT

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

Abstract State Machine Formalism

```
structure Event
  eventType as String

class State
  id         as String
  initial    as Boolean
  var active as Boolean = false

class Transition
  id as String

abstract class FSM
  id as String

  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>
```

Abstract Data Model

```
React (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
          // WriteLine ("Execute transition: " + t.id)
           CS.active := false
         step
           dstState(t).active := true
         step
           if t in me.outputEventType then
             return Event(outputEventType(t))
           else
             return null
     else
       if Size(enabledTs) > 1 then
         error ("NON-DETERMINISM ERROR!")
       else
         return null
```

Model Interpreter

ASML

Chen, Sztipanovits, Neema, DATE 2007

Status Report

- Much work needs to be done
  - Compositionality and scaling
  - Better link between denotational and operational approaches
  - Approachable formal framework (such as ASM, SLP, other?)
  - Probabilistic models
  - Design automation tools for composing DSMLs
  - Transitioning...

Overview

- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

Model-Based Tool Chains

Domain Specific Design Automation Environments:

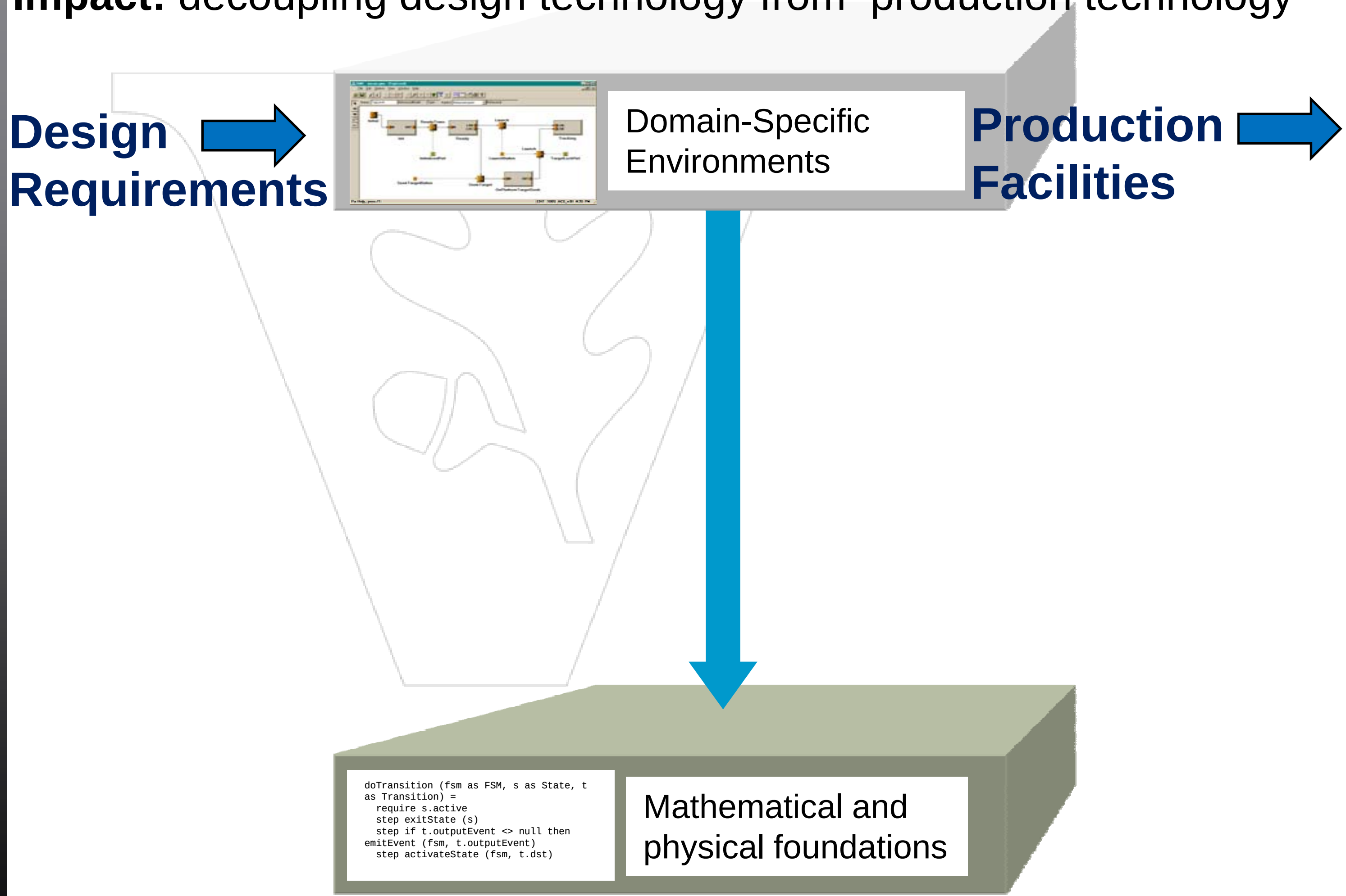
- *Automotive*
- *Avionics*
- *Sensors…*

Tools:

- *Behavioral Sim.*
- *Analysis*
- *Verification*
- *Synthesis*

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: decoupling design technology from production technology



Tool Chain Composition

Domain Specific Design Automation Environments:

- *Automotive*
- *Avionics*
- *Sensors…*

Metaprogrammable Tool Infrastructure

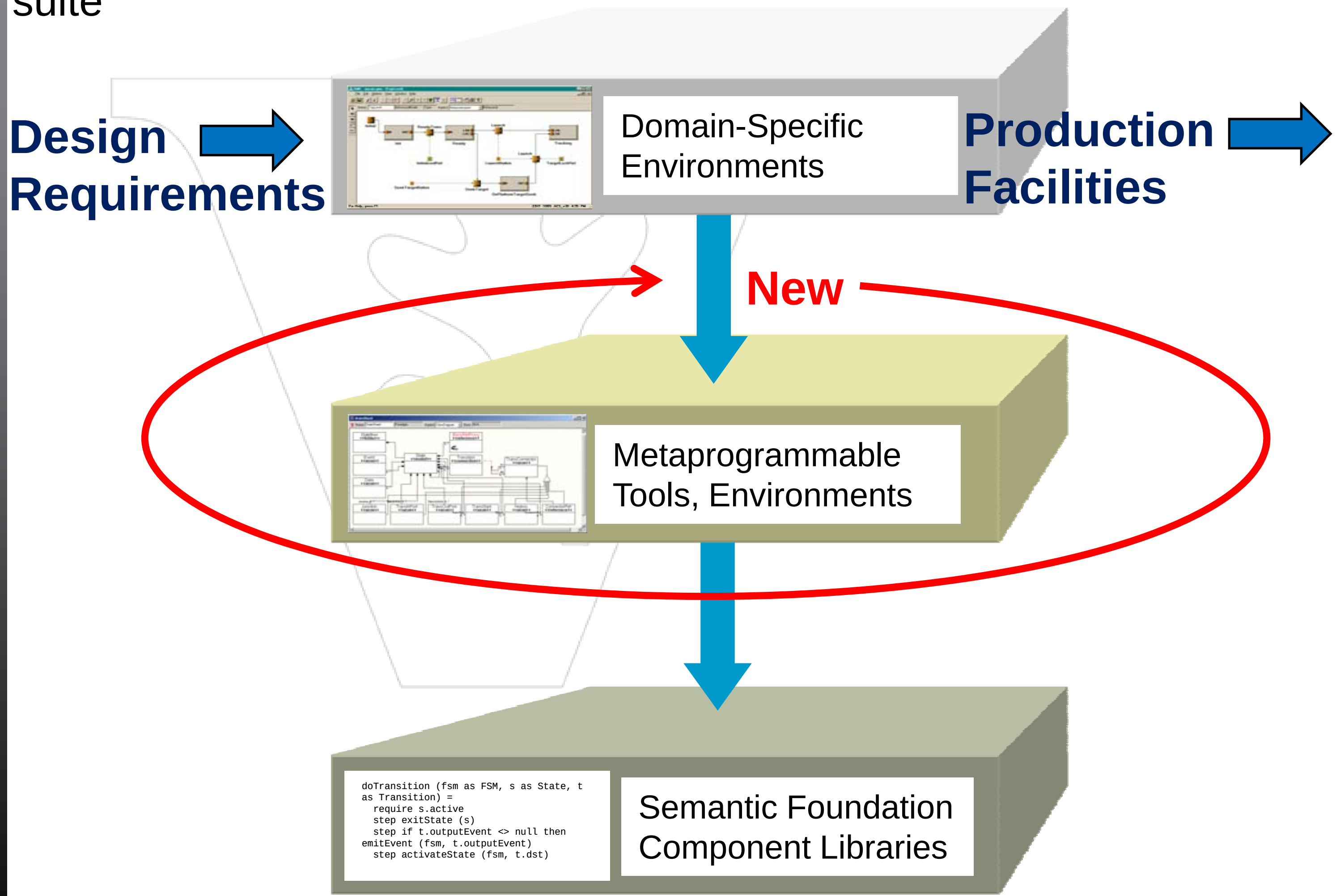
- *Model Building*
- *Model Transform.*
- *Model Mngmt*
- *Tool Integration*

Semantic Foundation

- *Structural*
- *Behavioral*

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



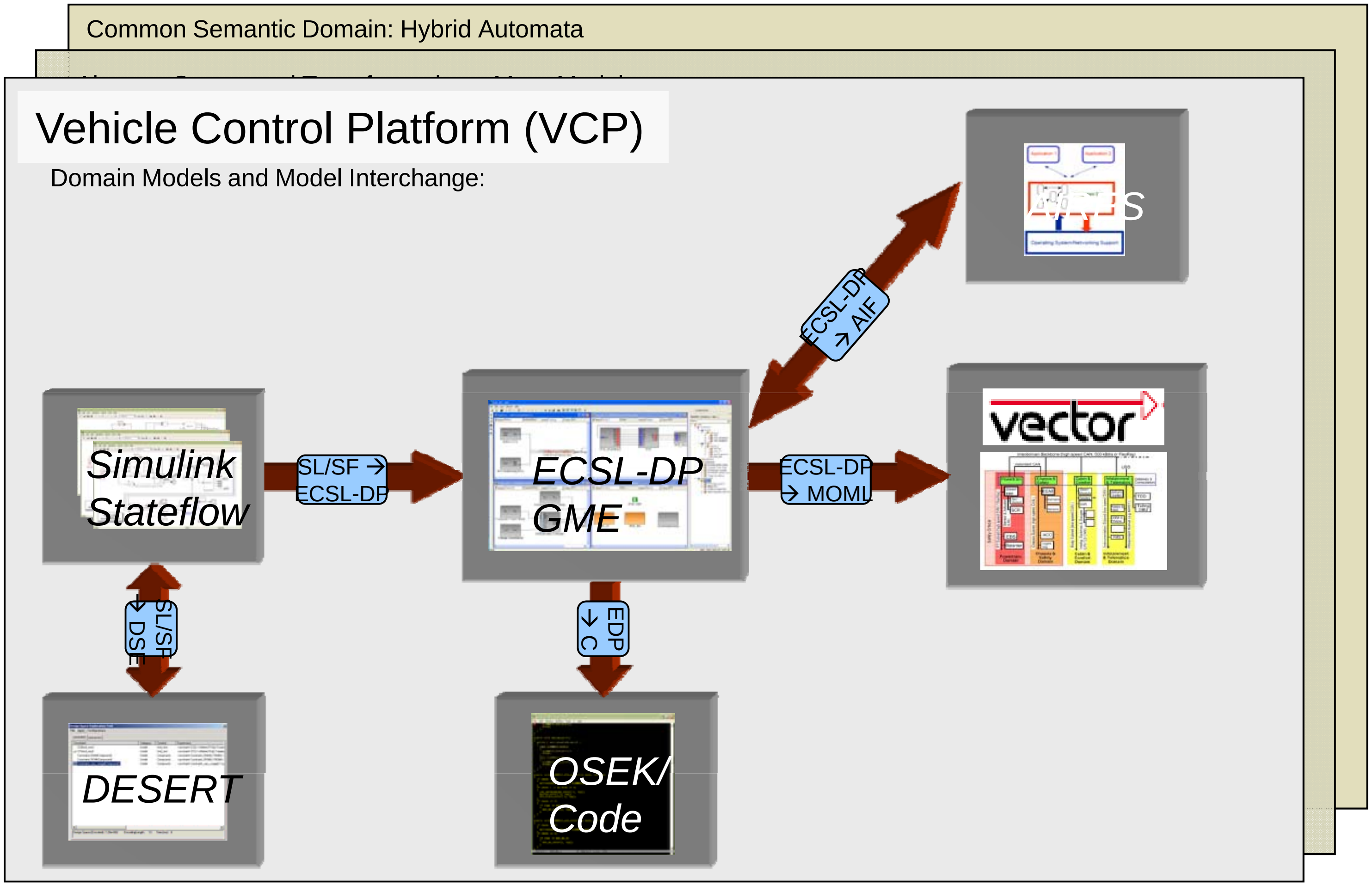
Tool Chain Example: VCP

Abstraction layers are defined by DSML-s of simulation, analysis And synthesis tools.

Design models are refined, transformed and analyzed in the design flow.

Analysis tools are integrated in the design flow by model transformation components

Key Idea: Use best-of-breed tools and multiple modeling languages in design flows.



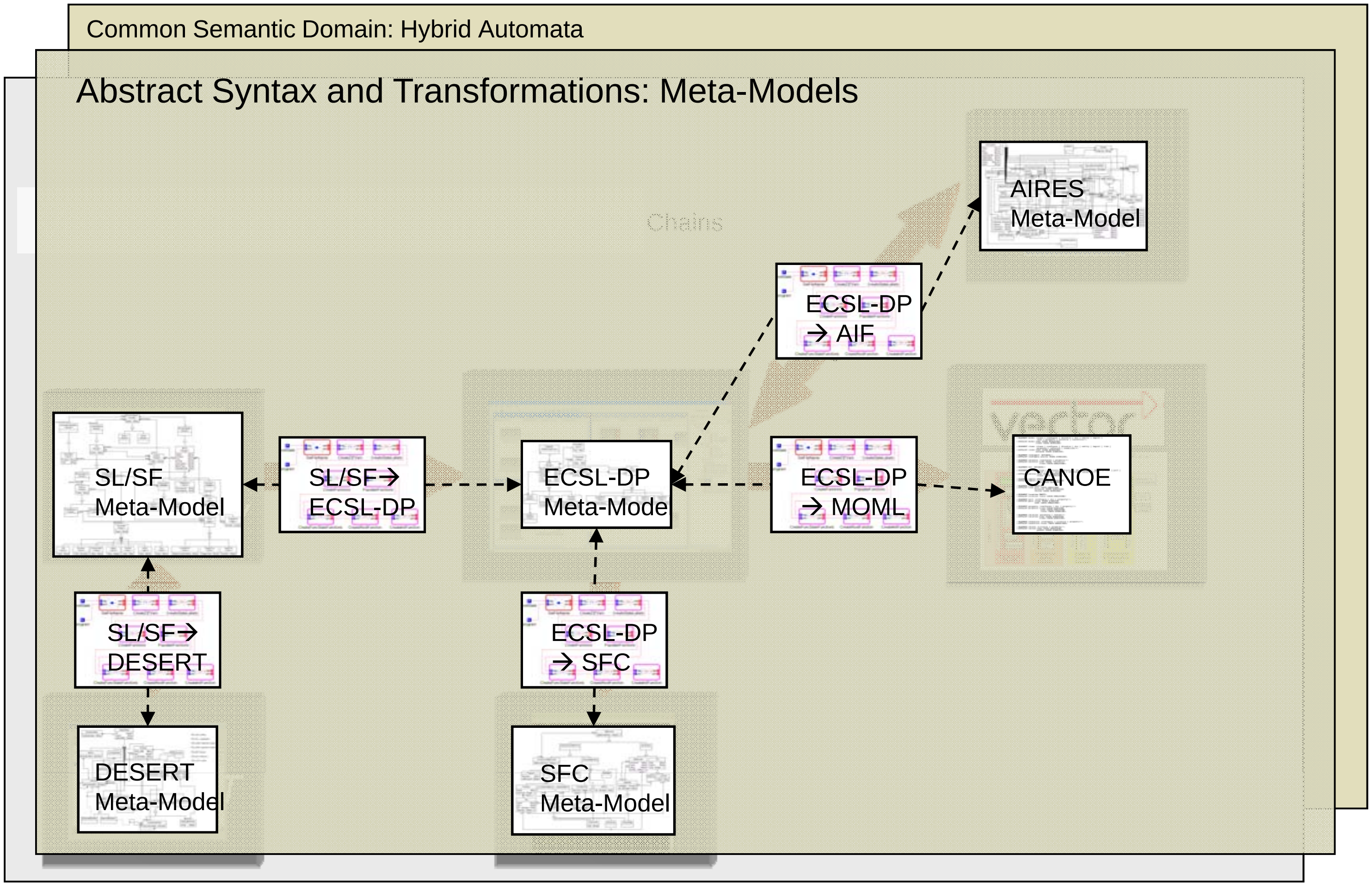
Integration of VCP

Abstraction layers are defined by formally specified DSML-s.

Metamodels are used for expressing relationship among models used in the the design flow.

Models of model transformations specify the “glue” that connect analysis tools to the design flow.

Key Idea: Integrate domain and tool specific models through metamodeling and model transformations.



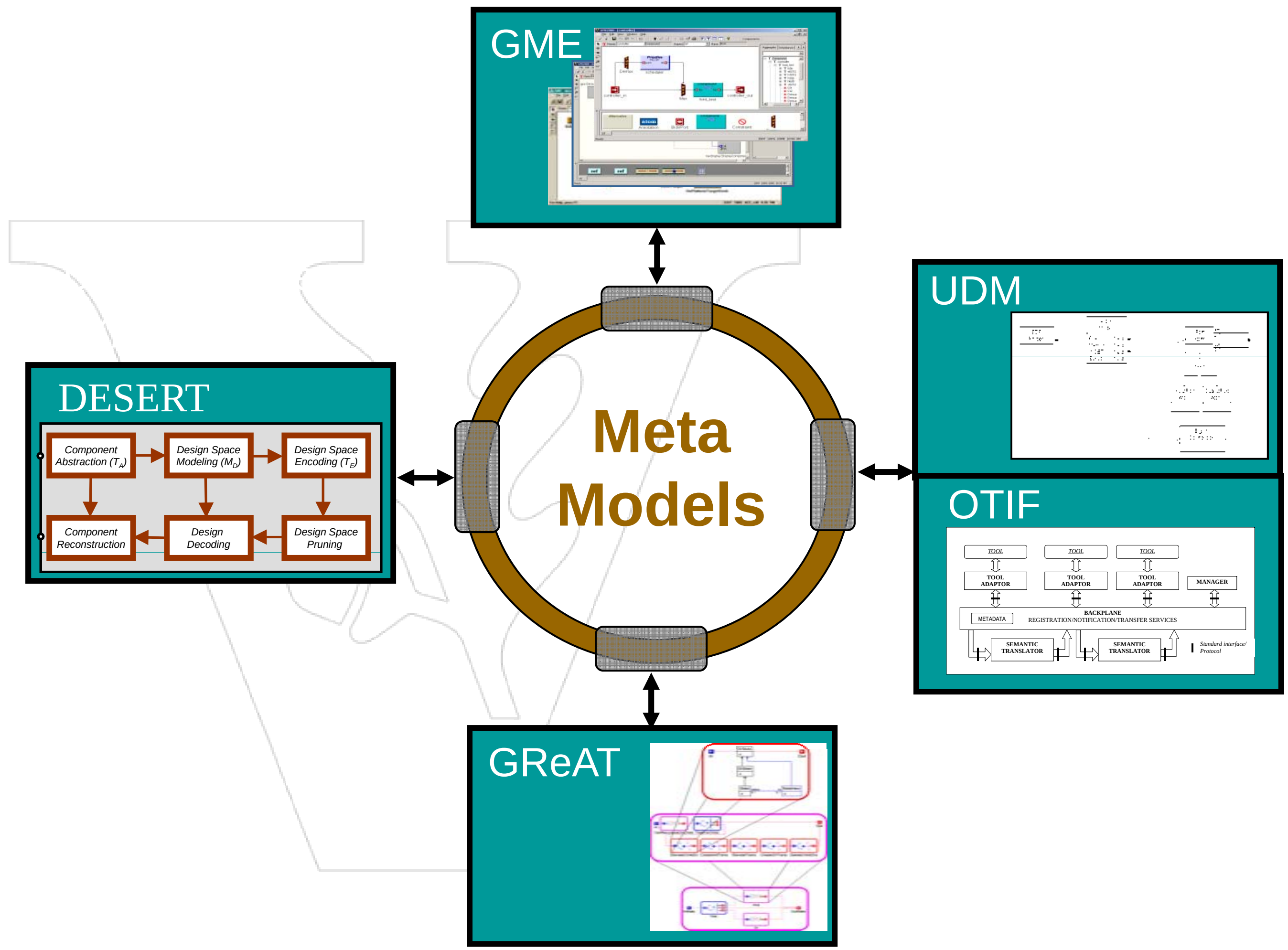
Integrated MIC Tool Suite

Domain independent metaprogrammable tool base for domain specific design flows

Application diversity of the MIC tool suite is huge:

- *Aerospace*
- *Automotive*
- *Health Information Systems*
- *Networked system integration*
- *System security*
- *. . . .*

The MIC tool suite has been evolving over 20 years



ESCHER Quality Controlled Repository:
http://escher.isis.vanderbilt.edu

Status Report

- Significant and sustained research effort
  - **U.S.**: Berkeley (Ptolemy, Metropolis); CMU (Checkmate); Eclipse tools (IBM, many contributors); MIT (Alloy); UPenn (Charon); Vanderbilt (MIC)
  - **EU:** Verimag (BIP); BUTE (VIATRA); TU Vienna, OFFIS; TU Munich, ...
- Lack of major transitioning success in new domains (DARPA's Meta 2 program is a hope for breakthrough)
- Need for broadening application domains (medical, SoS,...)
- Need for significant scaling up model management tools

Overview

- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

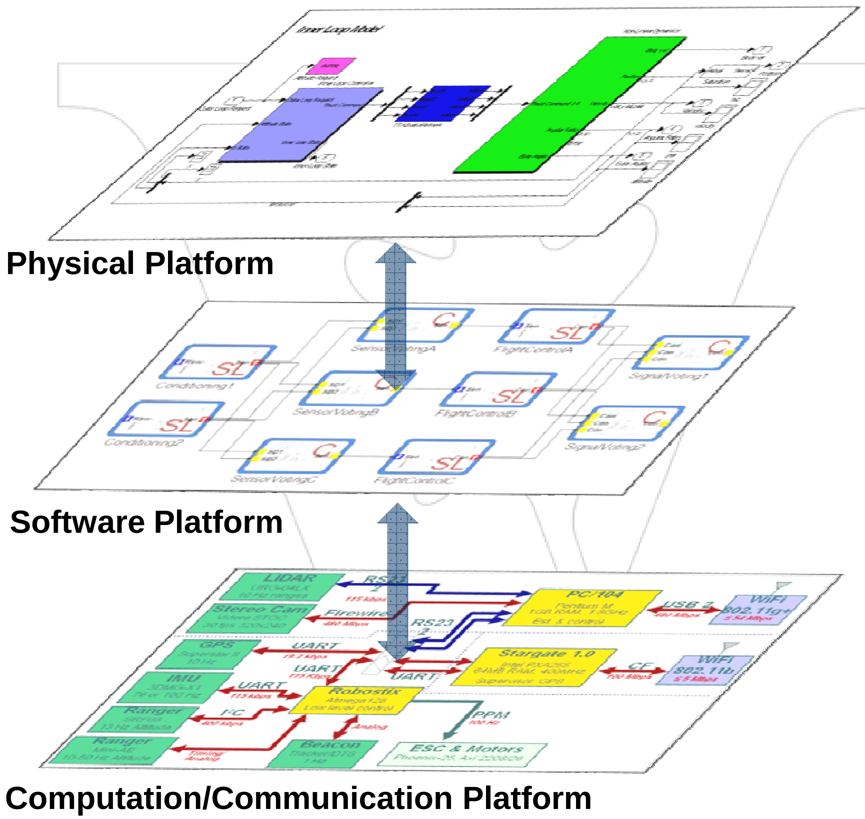
Model-Based Design

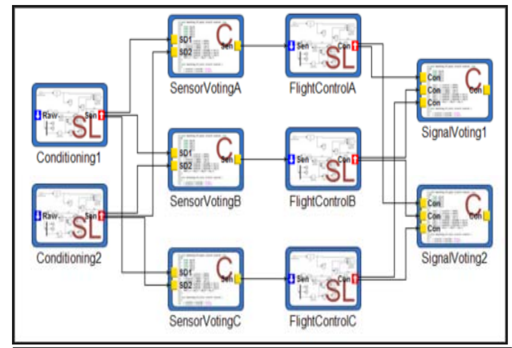
Key Idea: Manage design complexity by creating abstraction layers in the design flow.

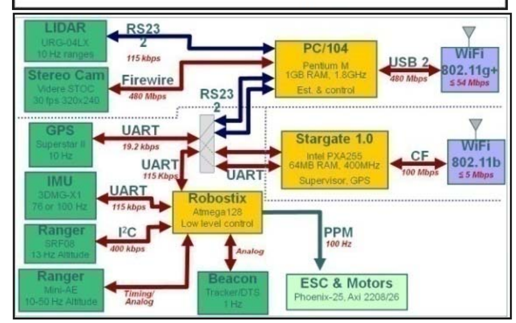
Abstraction layers define platforms.

Abstractions are linked through mapping.

Abstraction layers allow the verification of different properties .

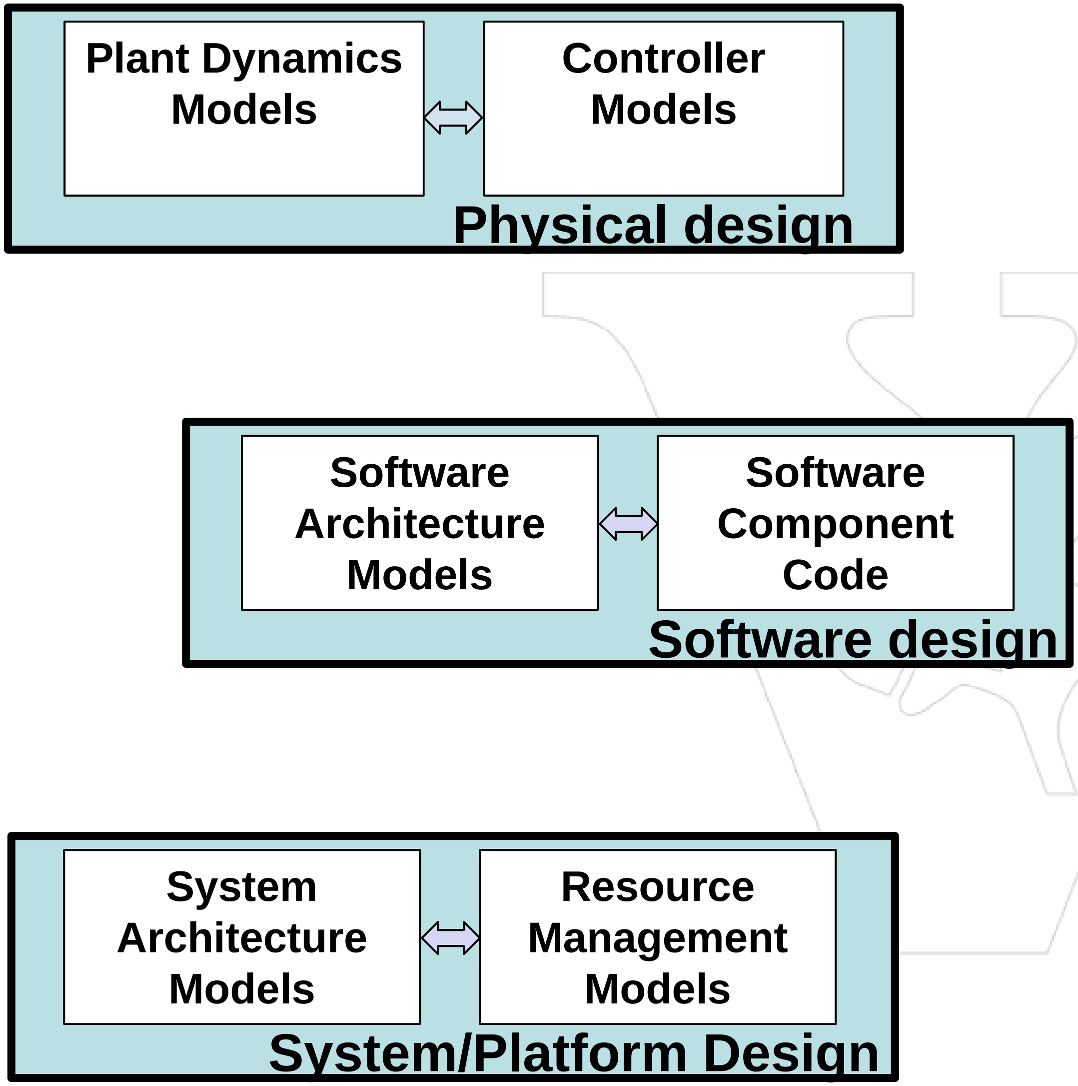






Frameworks and Tools for High-Confidence Design of Adaptive, Distributed Embedded Control Systems
MURI Project; Vanderbilt – UC Berkeley, CMU and Stanford

Integration Inside Abstraction Layers: Composition



Dynamics: $B(t) = \kappa_p(B_1(t), \ldots, B_j(t))$

- *Properties*: stability, safety, performance
- *Abstractions*: continuous time, functions, signals, flows,…

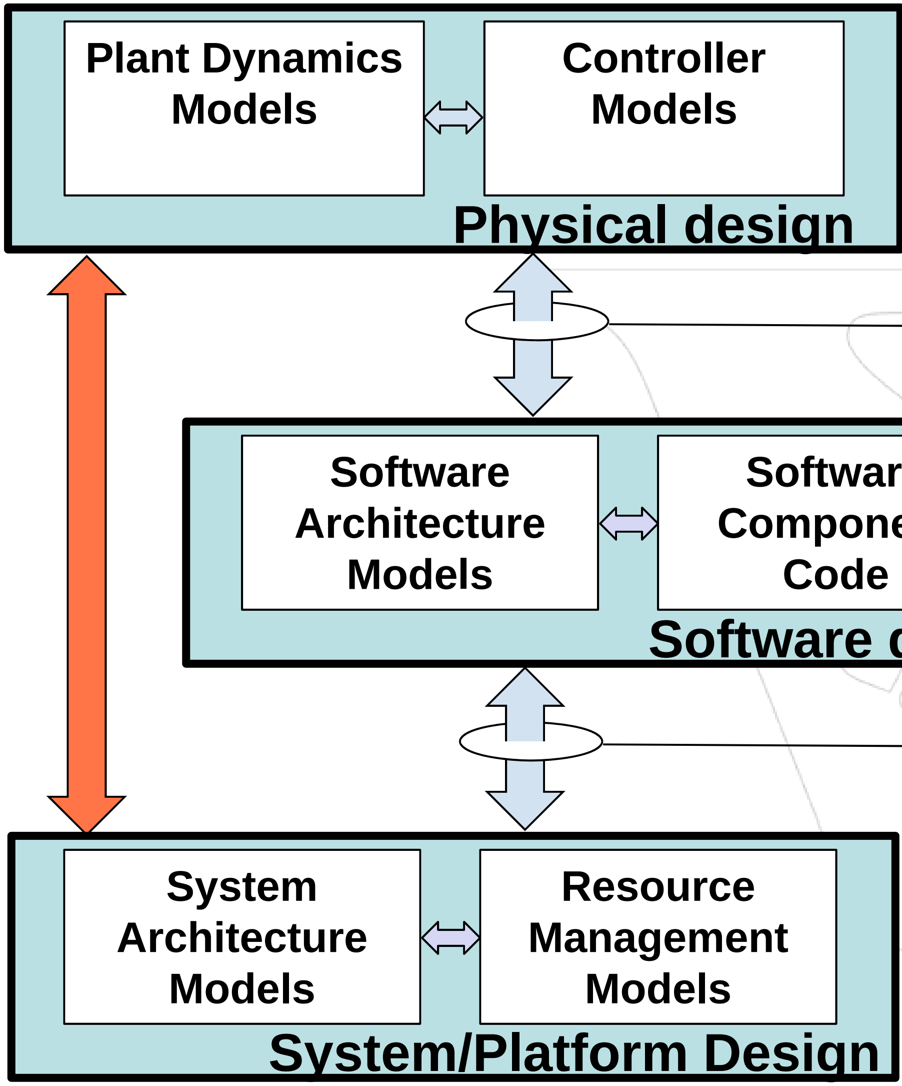
Software : $B(i) = \kappa_c(B_1(i), \ldots, B_k(i))$

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

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,

Integration Across Abstraction Layers: Much Unsolved Problems



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

<u>Assumption:</u> 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.

<u>Assumption:</u> 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.

Challenge to Compositionality: Heterogeneity

- Consequence of the lack of composability across system layers
  - intractable interactions
  - unpredictable system level behavior
  - full-system verification does not scale
- Active research: 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**

Physical layer: Passivity-based design

<u>Key idea</u>: Passivity-based design of networked control systems provides robustness to time-varying delays

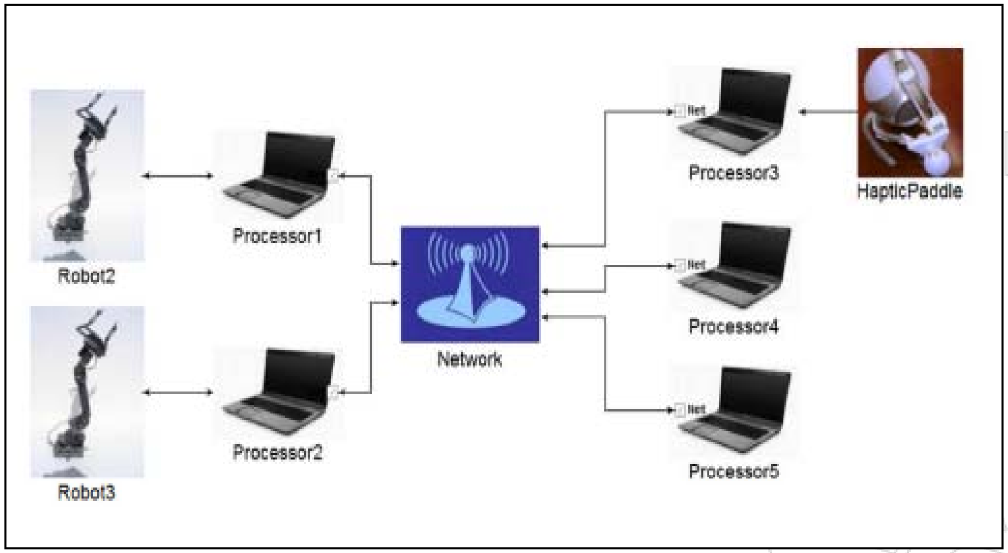
- Various mathematical definitions
  - A passive system only stores and dissipates energy but cannot generate energy of its own
- Passive systems interact in a stable manner
  - When connected in either a parallel or negative feedback manner the overall system remains passive
- Passive control theory applies to
  - Linear and nonlinear systems
  - Continuous and discrete-time systems
- Easier and safer to control
  - Independent joint PD controller for robotic manipulator
  - Asymptotic stability for set-point tracking

Background on Passivity

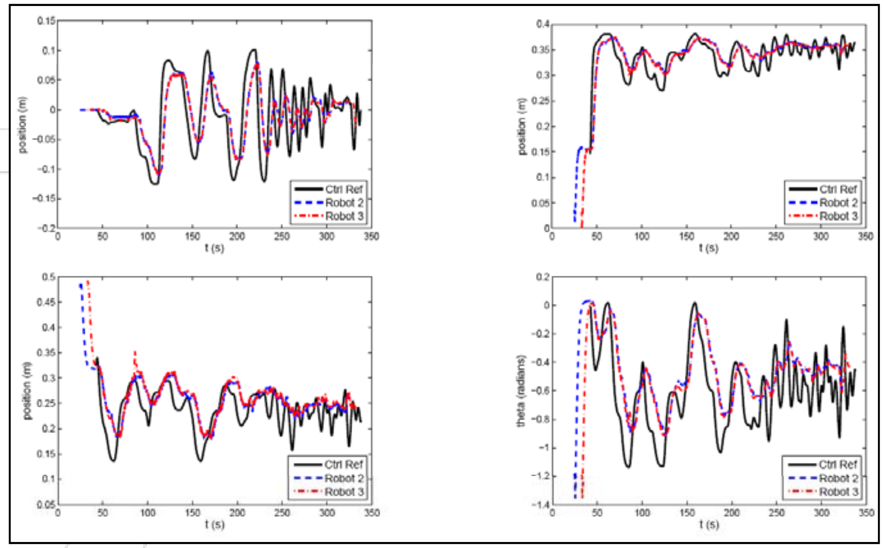
- Milestones:
  - Wave digital filters (Fettweis, 70's)
  - Dissipative dynamical systems (Willems, 70's)
  - Resonator-bank implementation structures (Peceli, 80's)
  - Teleoperation over the Internet (Niemmeyer, 04)
  - Power junctions (Kottenstette, Antsaklis, 08)
- Work at ISIS:
  - Design tool suite for high confidence systems (Eyisi, Hall, Hemingway, Porter, Karsai, Kottenstette, Koutsoukos, Sztipanovits)

Illustration of Passive Dynamics

Experimental Setup

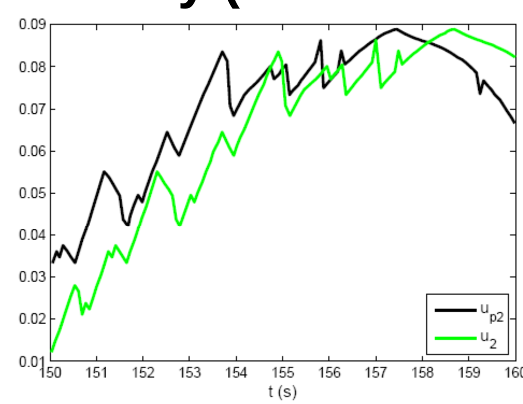


Joint Angle and Reference



- Two CrustCrawler robotic arms
  - 4 DOF with AX-12 smart servos at each joint
- Novint haptic paddle
- Five networked Windows PCs with Matlab/Simulink

Time delay (Robot 2 and PJ)



Status Report

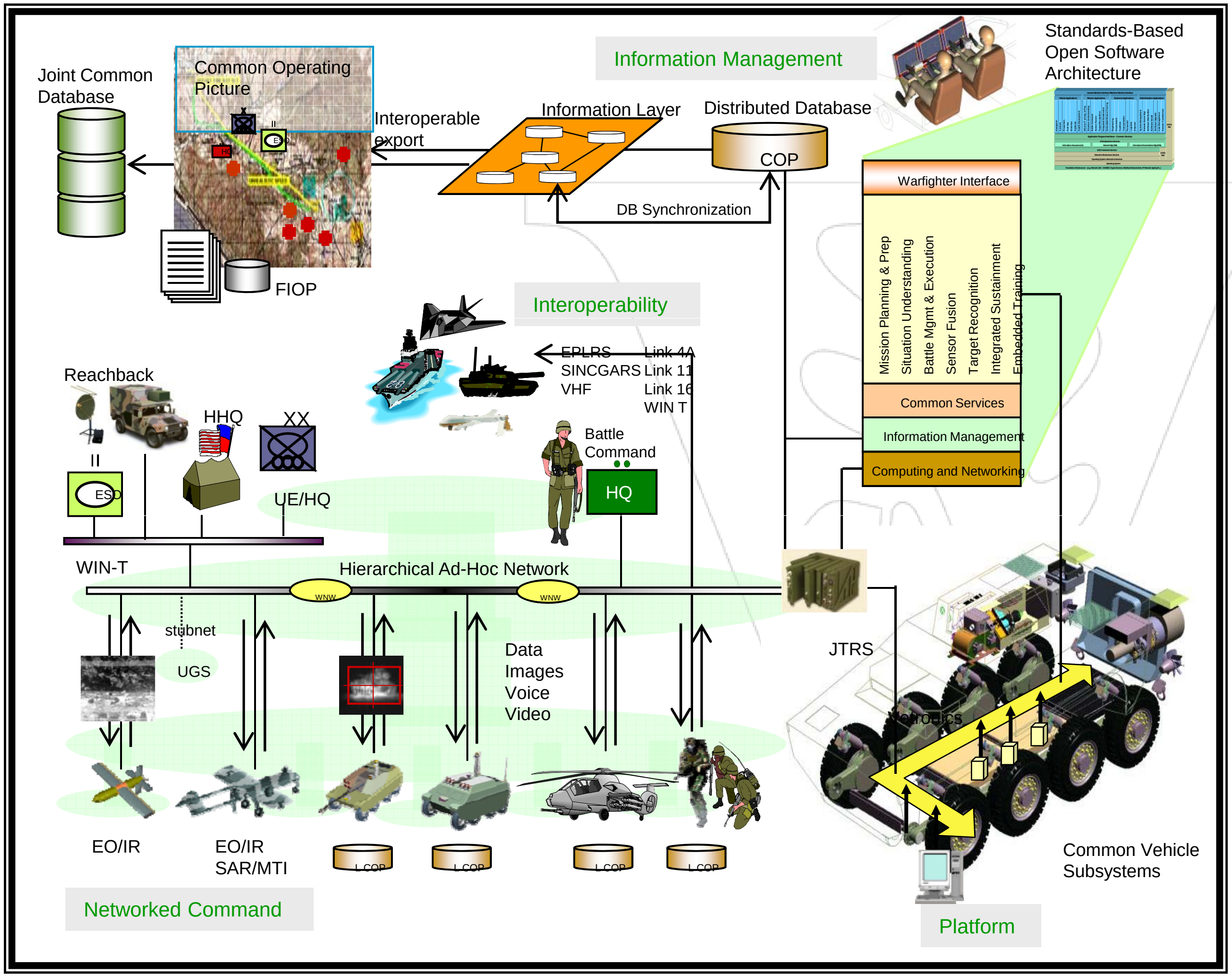
- Need to address more fundamentals:
  - extending theories for decoupling
  - developing theory of compositionality among system layers (vertical composition)
  - extending compositionality for multiple properties, e.g. stability, safety and invariants
- Early signs of increased attention
  - CPS research programs in US (NSF Center at Vanderbilt/Notre Dame/U. Maryland on Science of System Integration)
  - New conference sequence

Overview

- Cyber-Physical Systems (CPS)
- Model-Based Design
  - Structural Semantics
  - Behavioral Semantics
- Convergence
  - Towards Agile Design Automation
  - Towards Composition in Heterogeneous Systems
  - Examples
- Summary

Example 1: System of System Integration

Future Combat System



- Heterogeneous
- Open Dynamic Architecture
  - heterogeneous networking
  - heterogeneous components
- Very high level concurrency with complex interactions
- *Challenges*:
  - understanding and
  - predicting behavior

How to achieve predictability with limited/partial compositionality?

Real-Life SoS Development

- All integration categories are present (component, layer, SoS)
- Systems are evolving along “spiral-outs”
- New technical challenges are emerging and potential solutions need to be rapidly explored
- All layers of the system are subject to modifications, there are no well defined synchronization points in the development process
- **Integration is inherently incremental; deployed systems need to be integrated with components on different level of maturity: prototypical and with simulated systems/components.**

How Is It Solved Today?

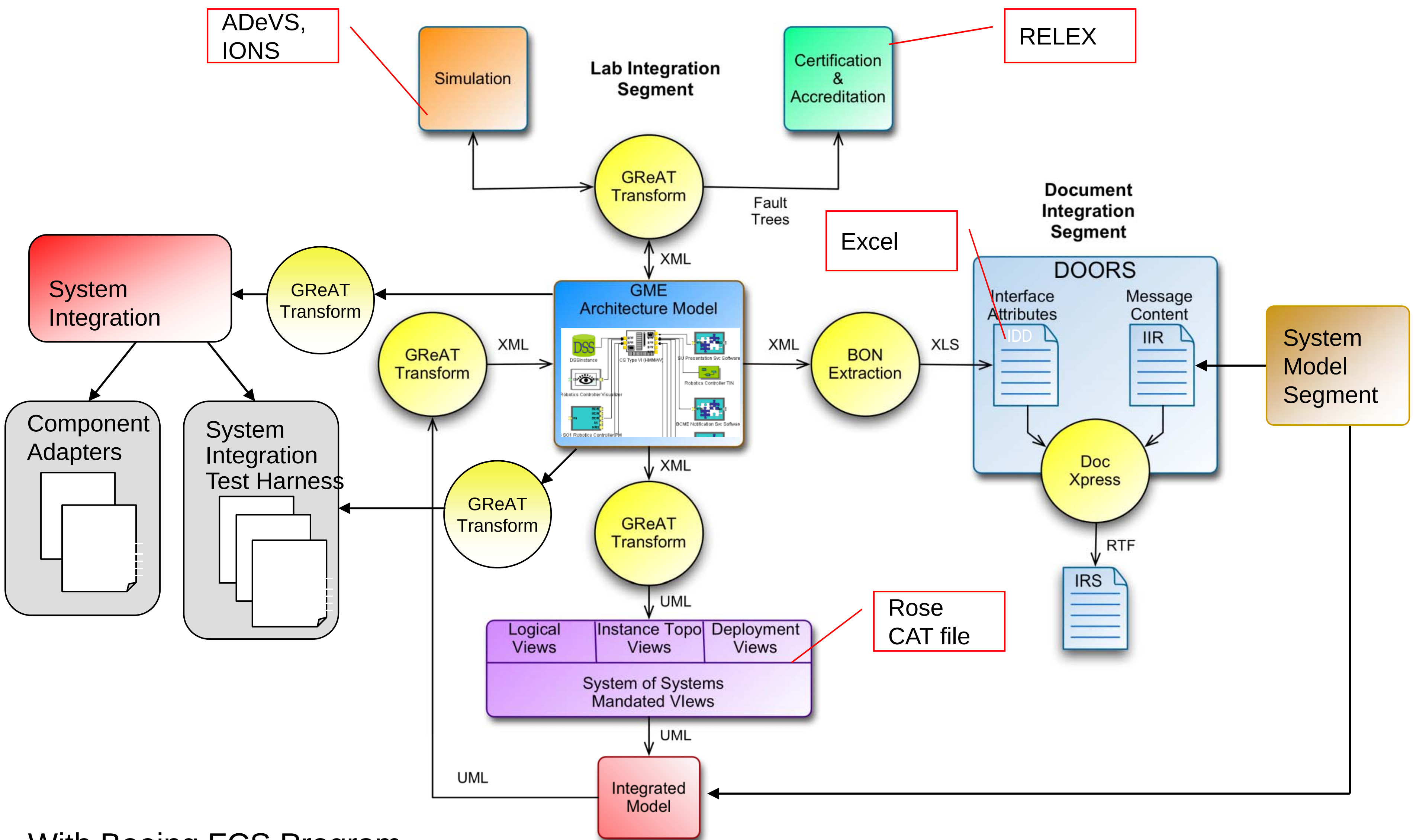
- Systems are integrated when all components are delivered
  - Acquisition pushes in this direction
- Integration means: “Make it working somehow”
- System Integration Labs do not offer support for spiral development
- There is no approach to deal with incomplete specifications and components

System Integration is the highest risk, most expensive, least predictable step in SoS development

Emerging Solution: Model-Based Integration

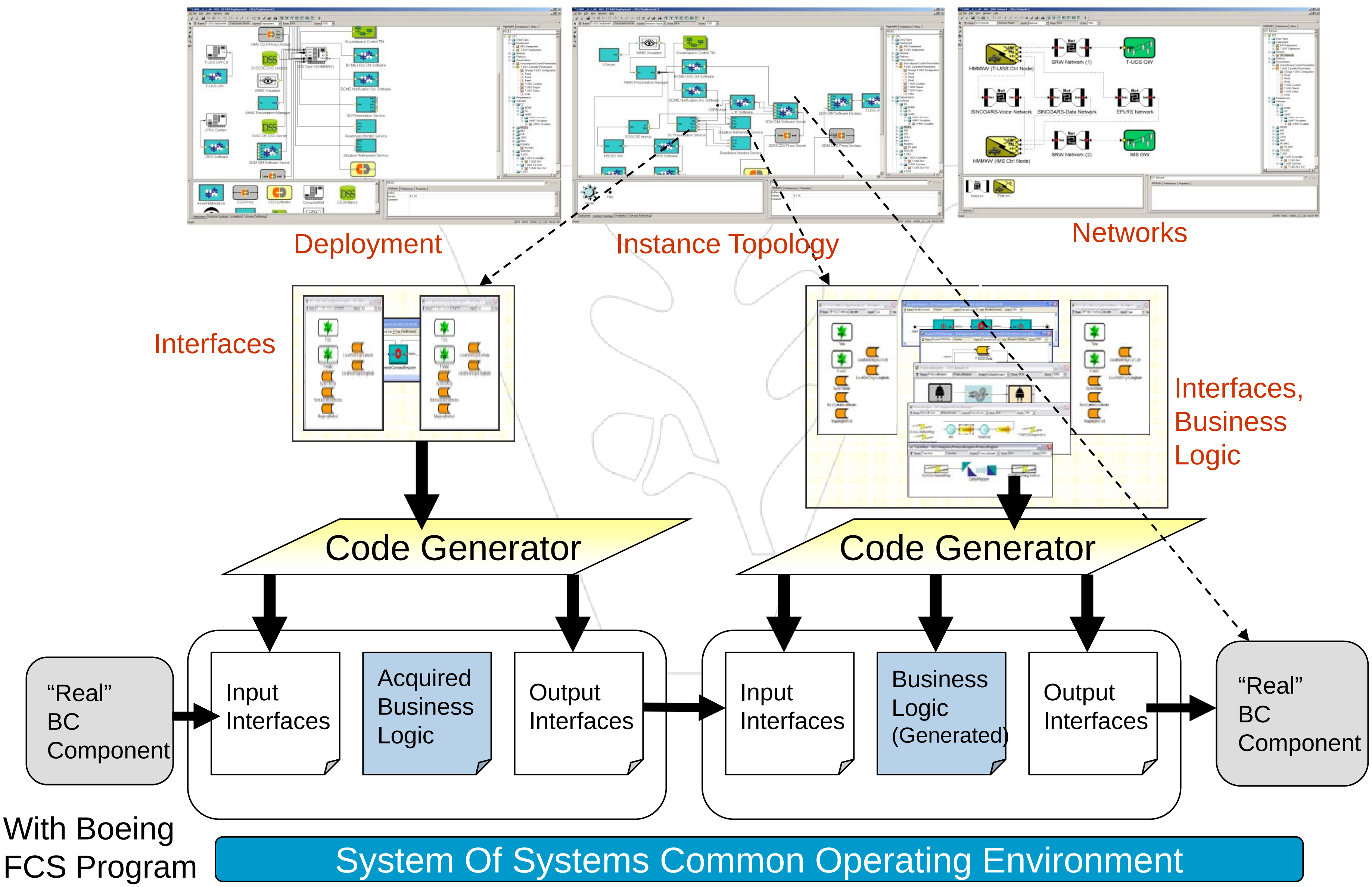
- Apply Models Early
- Apply Models Often
- Use Every Opportunity
  - Requirements/Architecture Integration
  - Architecture/Design Integration
  - Design Assessment/Verification
  - Prototyping/Scaling
  - Implementation
  - Scaling
  - Testing

Tool Chain for Architecture Exploration in FCS

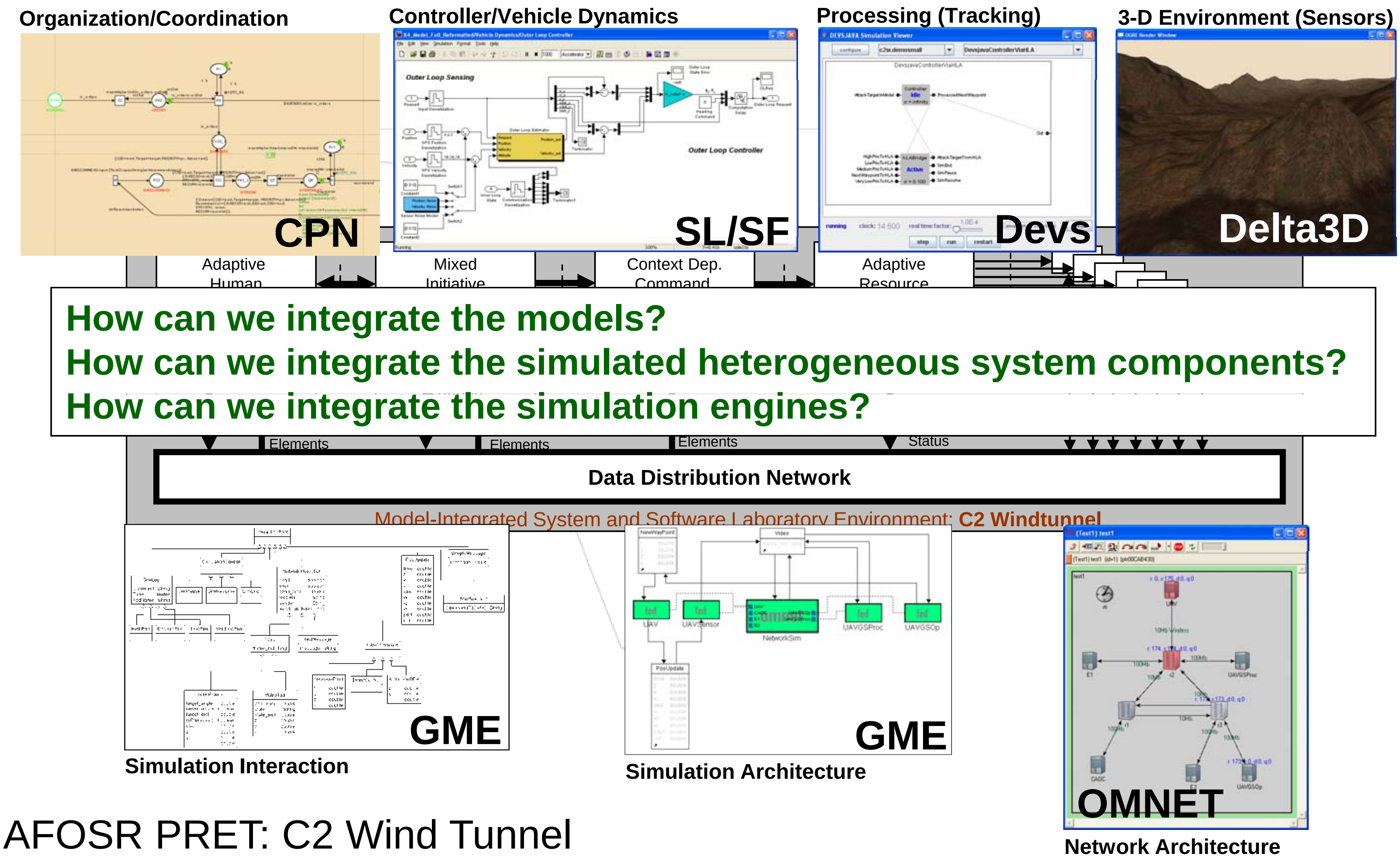


With Boeing FCS Program

Risk Mitigation: Surrogate Modeling and Synthesis



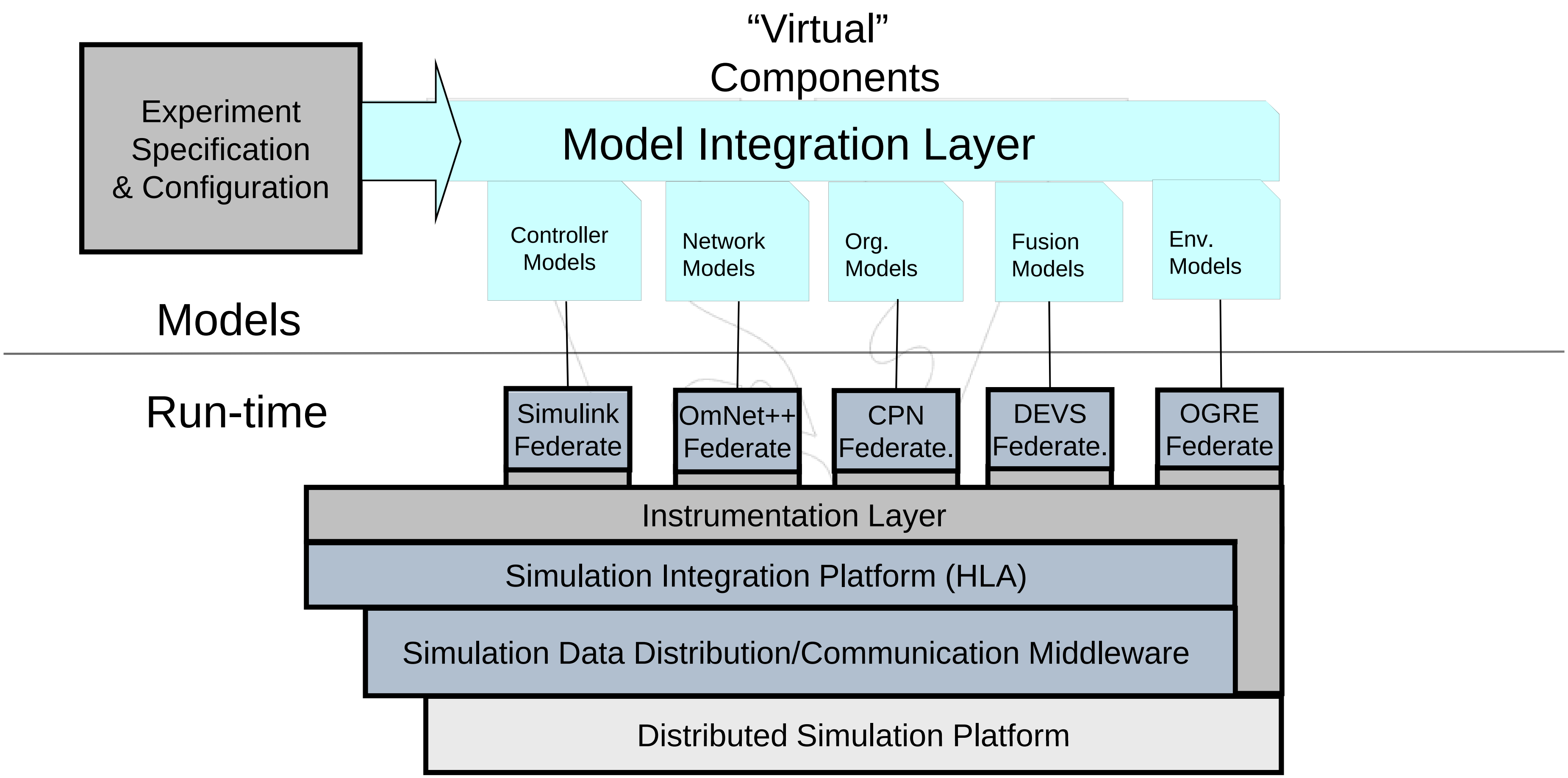
Example 2: Heterogeneous Simulation Integration



How can we integrate the models?
How can we integrate the simulated heterogeneous system components?
How can we integrate the simulation engines?

AFOSR PRET: C2 Wind Tunnel

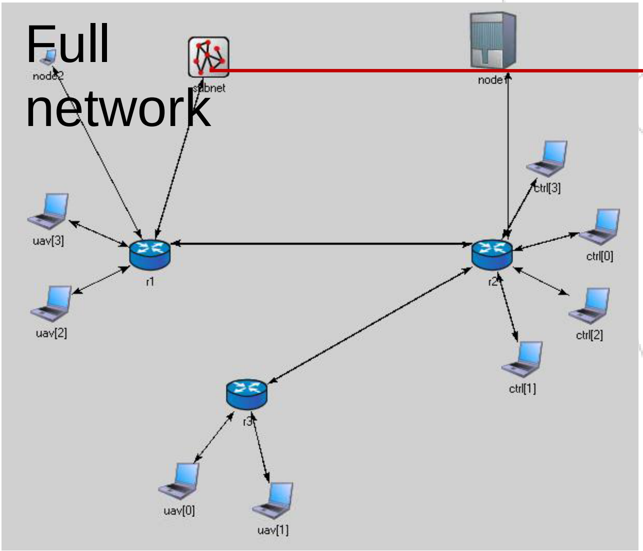
Model-based Integration Architecture

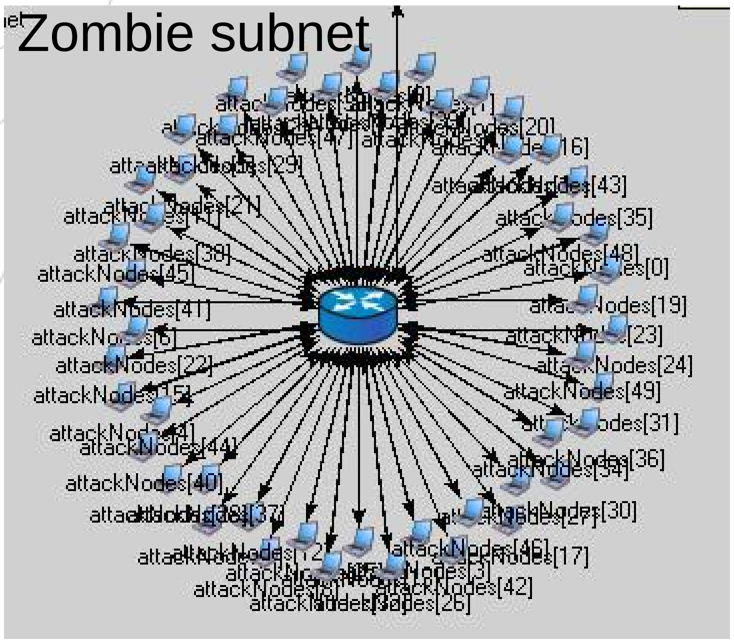


https://wiki.isis.vanderbilt.edu/OpenC2WT

Experiments: Impact of Cyber Attacks

- **Network attack:**
  - A sub-network with hundreds of zombie nodes attacks a critical router on the main network.
  - Flood attack on udp, tcp or ping





Summary

- Penetration of networking and computing in all engineered systems forces a broad based convergence across engineering disciplines.
- Signs of this convergence is present in many areas from which we discussed two:
  - Design Automation – emergence of metaprogrammable tool suites and multimodeling
  - System Integration – re-integration of computer and systems science
- Model-based design facilitates a necessary convergence among software, system, control and network engineering