Composition Challenges in Heterogeneous Systems

Janos Sztipanovits
ISIS, Vanderbilt University

24 October, 2010

Outline

1. Motivation for multi-modeling
2. Towards composition of modeling languages
   - Structural semantics
   - Behavioral semantics
3. Summary

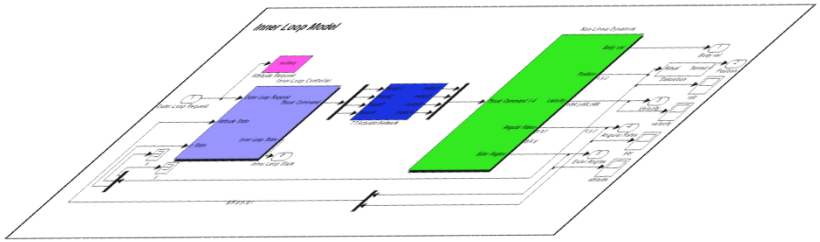
Model-Based Design & Platforms

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

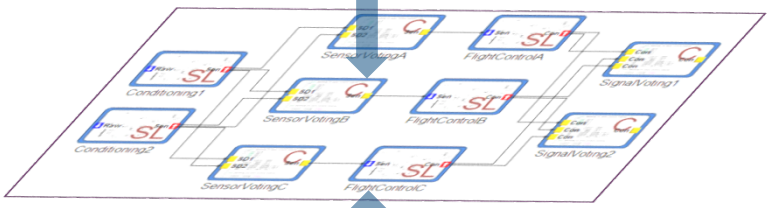
Abstraction layers define platforms.

Abstractions are linked through mapping.

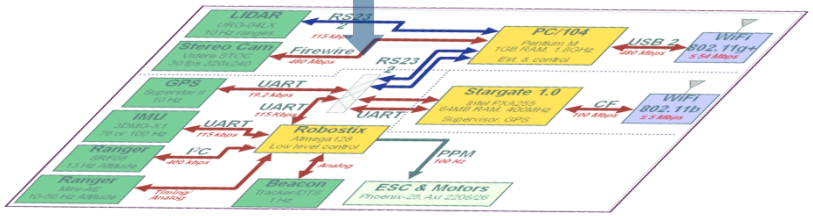
Abstraction layers allow the verification of seelcted properties .



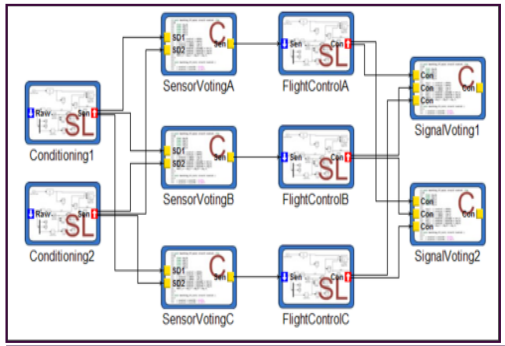
Physical Platform

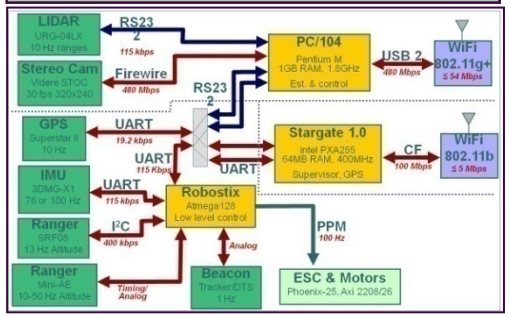


Software Platform



Computation/Communication Platform





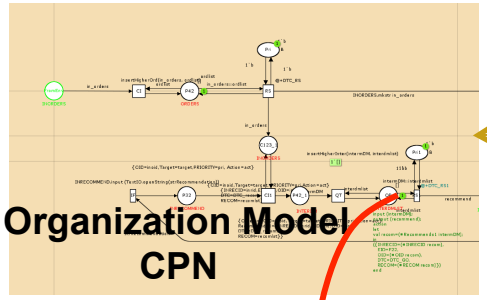
Claire Tomlin, UC Berkeley

Why Multi-modeling and DSMLs?

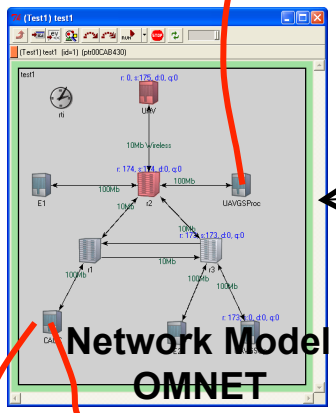
- Systems are heterogeneous (physical, software, computation/networking platforms, human,..)
- Extensive model libraries (MATLAB, MODELICA, OmNET, OPNET, Dymola, LabVIEW,..)
- Established V&V techniques (UPPAAL, SPIN, PRISM,…)
- Composition frameworks (BIP, Ptolemy, DDS, BPEL,…)
- Simulation environments (SimuLink, DEVS,…)
- Belief in standards (SysML, AADL, UML,…)
- User communities and accepted practices
- …

Composition Problems in Multi-modeling

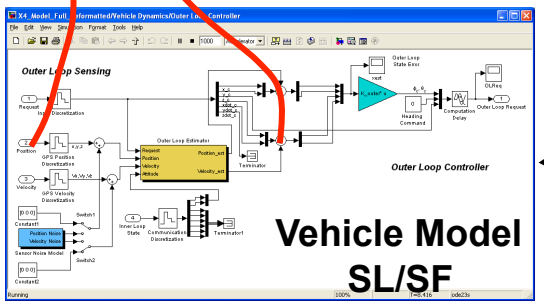
Integrating Models



Organization Model
CPN



Network Model
OMNET



Vehicle Model
SL/SF

Composing Modeling Languages

Domain of *well-formed*
CPN Models

?

Domain of *well-formed*
OMNET Models

?

Domain of *well-formed*
SL/SF Models

Path Towards Multi-modeling

- Understanding the composition of domains: **Structural Semantics**
- Understanding the behavioral interaction among domains: **Behavioral Semantics**

Outline

1. Motivation for multi-modeling
2. Towards composition of modeling languages
   - **Structural semantics**
   - Behavioral semantics
3. Summary

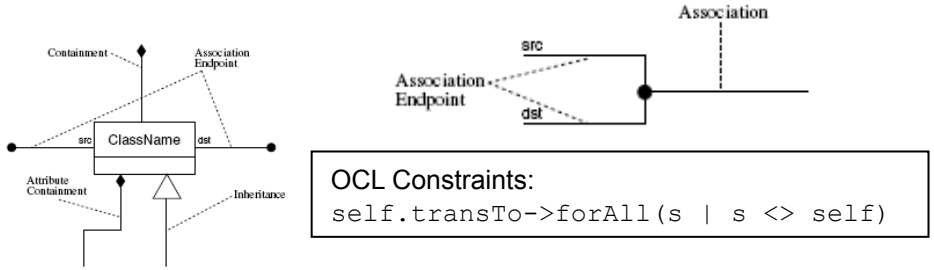
Specification of Domain-Specific Modeling Languages

Abstract syntax of DSML-s are defined by metamodels.

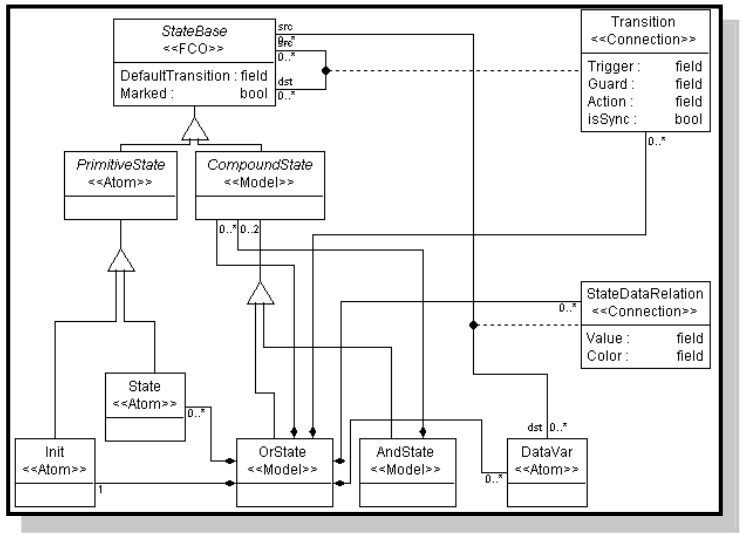
A metamodeling language is one of the DSML-s.

Semantics of metamodeling languages: structural semantics.

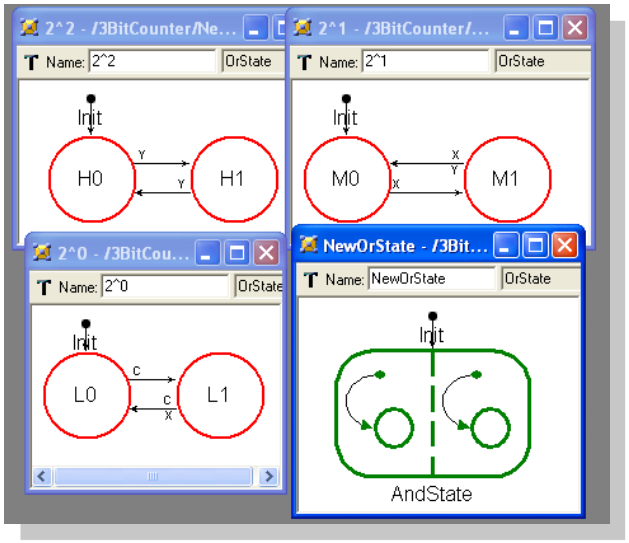
Key Concept: Modeling languages define a set *of well- formed models* and their *interpretations*. The interpretations are mappings from one domain to another domain.



Basic metamodeling notation: UML Class Diagram + OCL



MetaGME metamodel of simple statecharts



Model-editor generated from metamodel

Formalization of Structural Semantics

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

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: $m' = T(m); m' \in X; m \in Y$

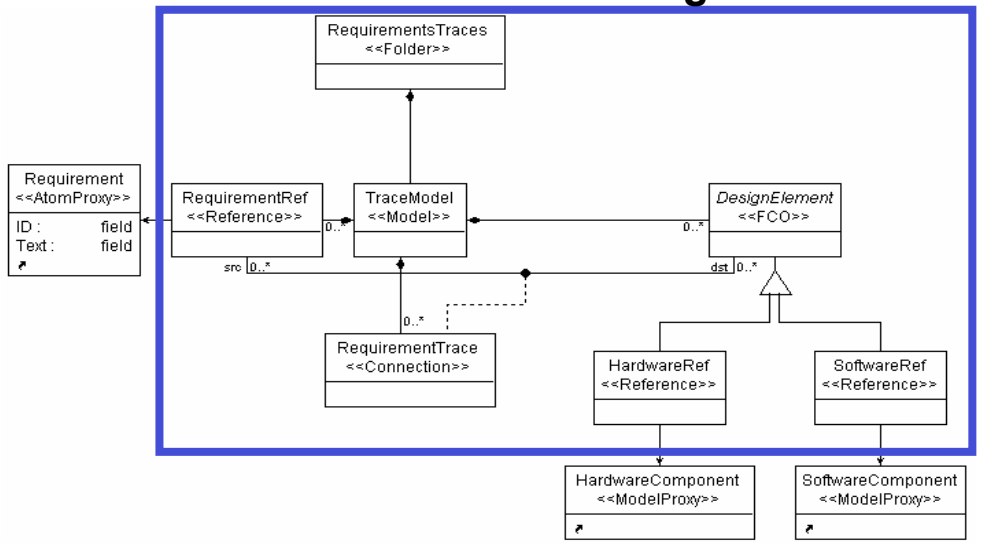
Notes on the selected formalism:
- Term algebra semantics extended with Logic Programming (LP)
- Fragment of LP is equivalent to full first-order logic
- Provide semantic domain for model transformations.

DSML Composition Using Structural Semantics

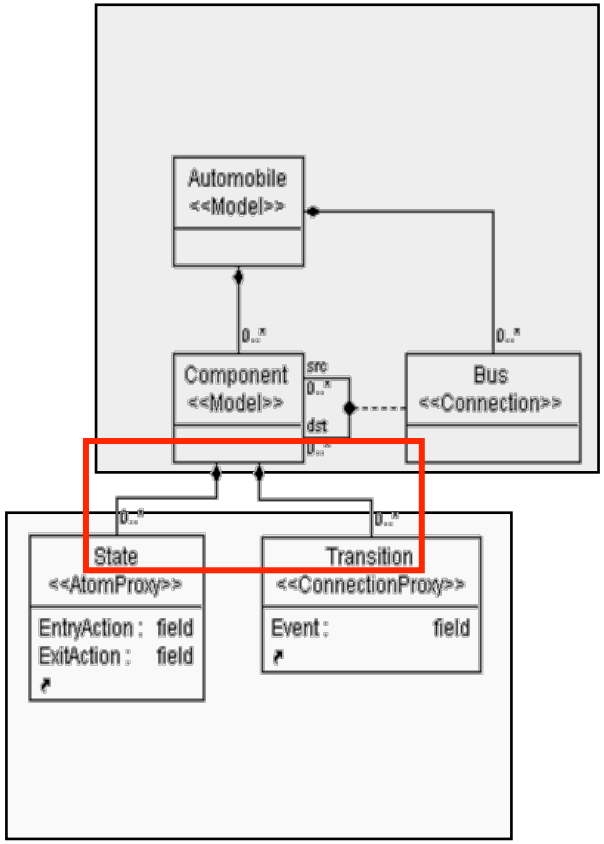
- Domains D_1 and D_2 are composed using some appropriate structural composition operators.
- Sufficient if behavioral semantics of D_1 and D_2 are
  - the same, or
  - not interesting, or
  - D_1 (with structural semantics only) is used for the structural extension of D_2 (that may have behavioral semantics)

Example Operators for Structural Composition

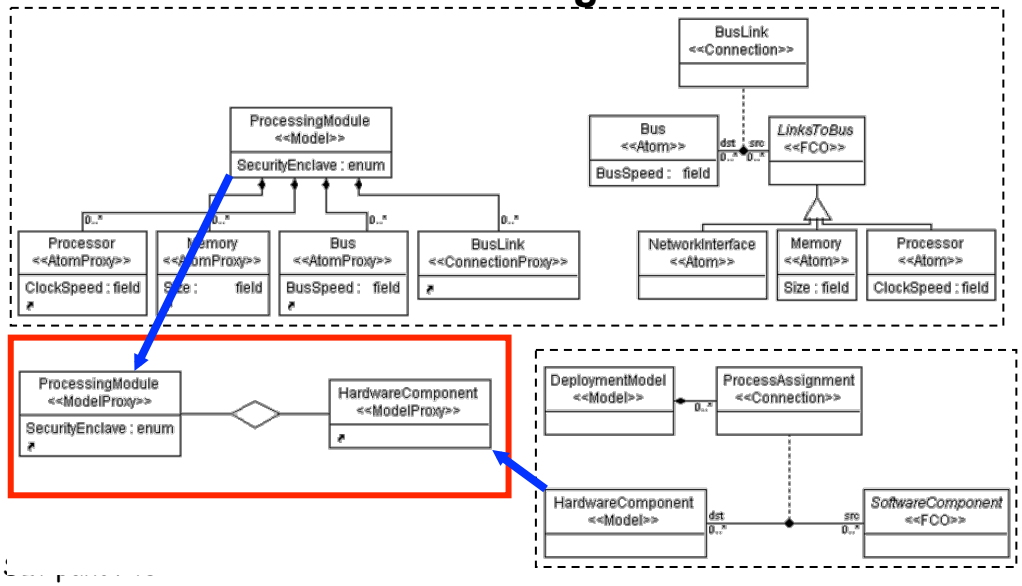
Metamodel Interfacing



Class Refinement

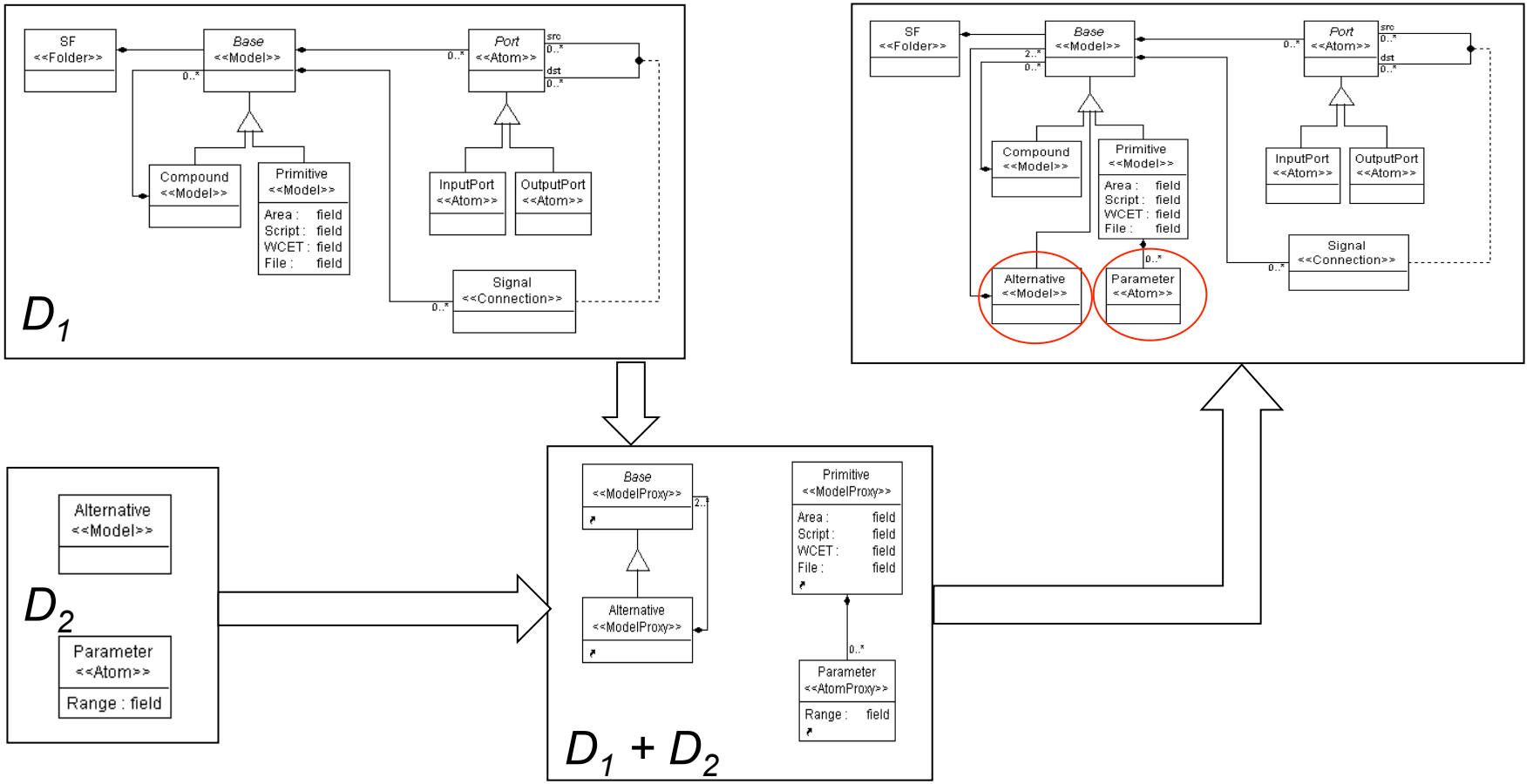


Class Merge



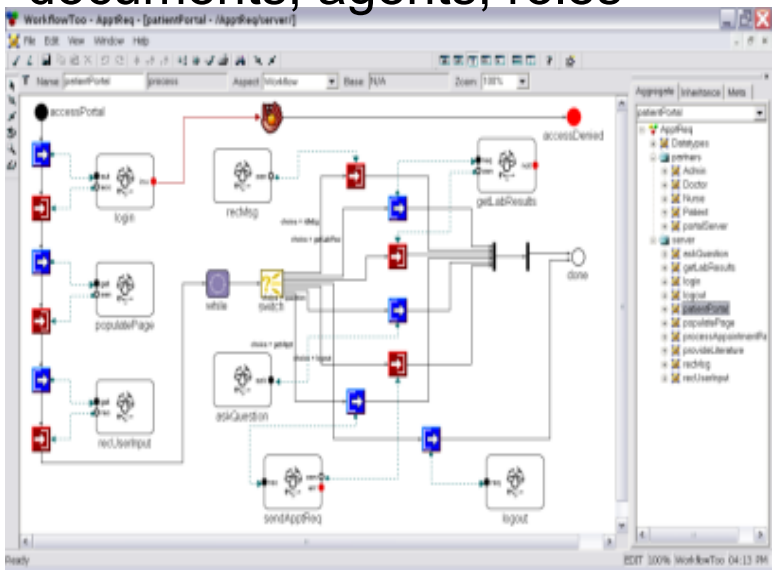
Example for Structural Composition: Design Space

D_1 is a component modeling language with behavioral semantics, D_2 is an abstract design space construction language.



Example for Structural Composition: Adding Privacy to HIS

D_1 Models of information flows, documents, agents, roles



D_2 Models of privacy policies (HIPAA)

Nurses should tag health questions

G ∀p, q, s, m. inrole(p, nurse) ∧ send(p, q, m)
∧ contains(m, s, health-question)
⇒ tagged(m, s, health-question)

Doctors should answer health ques.

G ∀p, q, s, m. inrole(p, doctor) ∧ send(q, p, m)
∧ contains(m, s, health-question)
⇒ **F** ∃m'. send(p, s, m')
∧ contains(m', s, health-answer)

Werner, Mathe Sztipanovits, 2009

Common Semantic Domain

Mitchell et al, 2006

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 structure -> structural semantics + LTL
- the generated system controls information flows and monitors policy violations

Structural Composition: Ongoing Work

- Further examples for complex multi-model environments (DARPA Meta -2)
- Integration with FORMULA-2
- Extension to dynamic structural semantics – LTL, MLTL

Outline

1. Motivation for multi-modeling
2. Towards composition of modeling languages
   - Structural semantics
   - **Behavioral semantics**
3. Summary

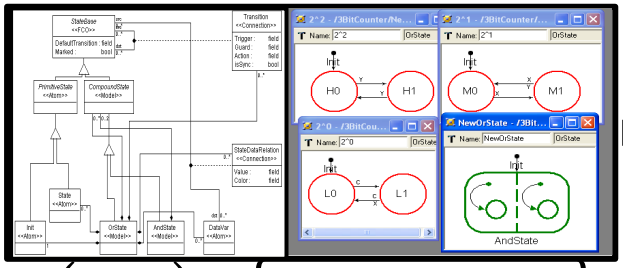
Behavioral Semantics

- Given a DSML

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

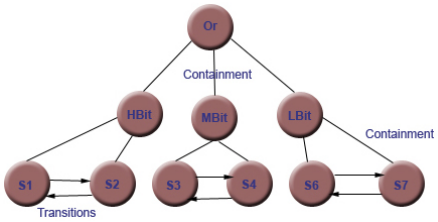
- Behavioral semantics will be defined by specifying the transformation between the DSML and a modeling language with behavioral semantics.

Formalization of Behavioral Semantics



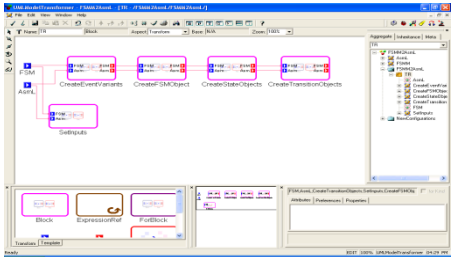
$$D(Y,C) = \{r \in R_Y \mid r \models C\}$$

Representation as AST



MIC-UDM
MIC-GME

Graph rewriting rules



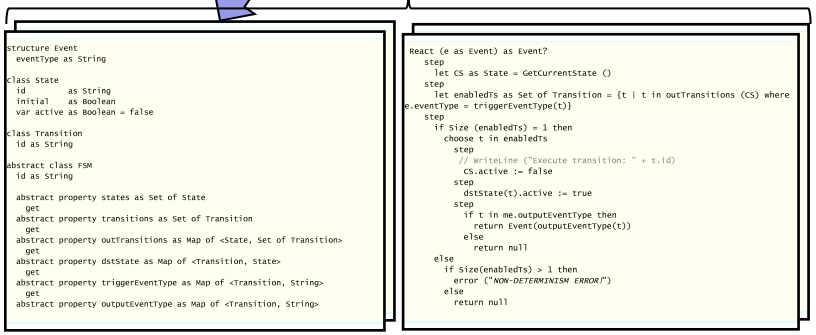
MIC-GReAT

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

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

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

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>
```

```
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
```

Abstract Data Model

Model Interpreter

- **ASML**
- BIP
- Formula

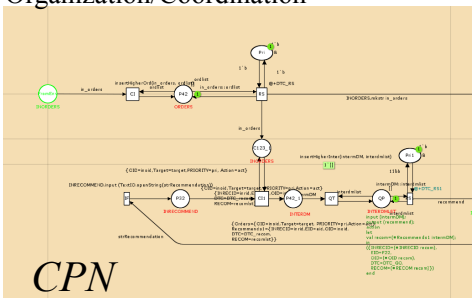
Integration of DSMLs with Heterogeneous Behavioral Semantics

- Model integration languages
  - common interaction semantics, component behaviors are defined using different languages
  - multiple interaction semantics, heterogeneous component behaviors defined by different languages

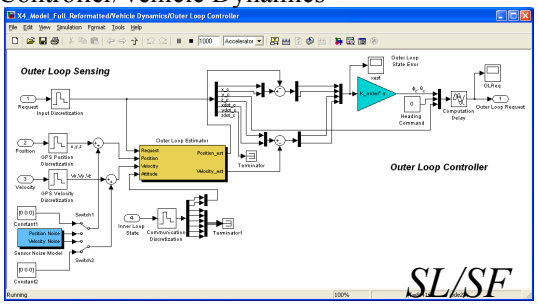
- Enforcing interactions across DSMLs through model transformations
- Translation of component languages into common semantic domain

Example: Multi-Model Simulation Integration

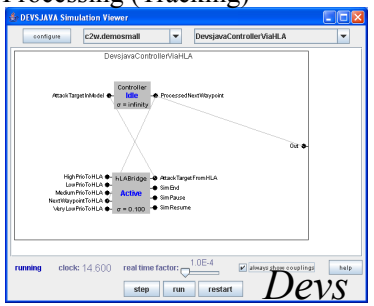
Organization/Coordination



Controller/Vehicle Dynamics

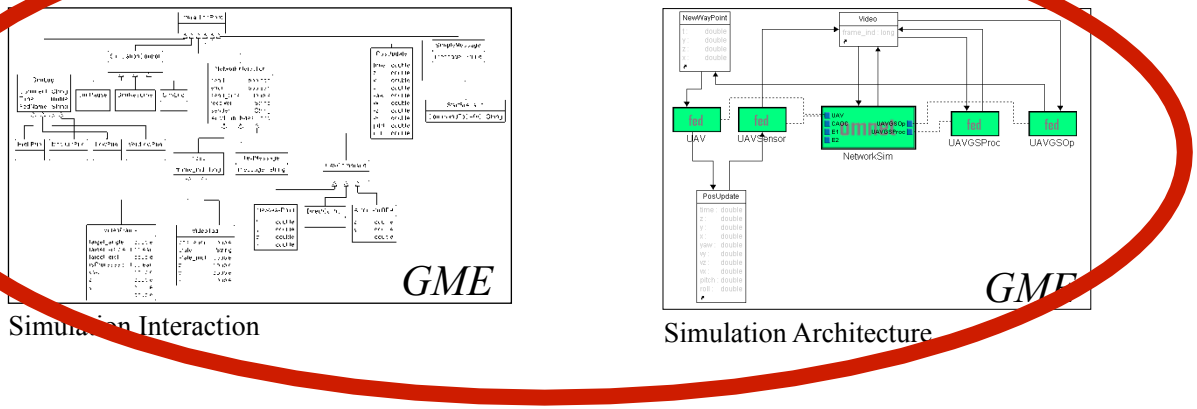


Processing (Tracking)



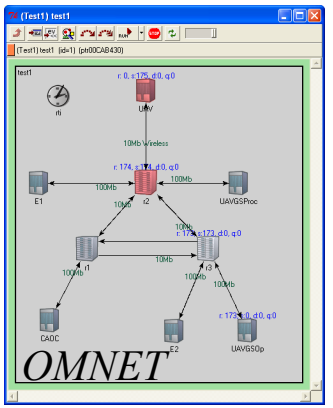
3-D Environment (Sensors)

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



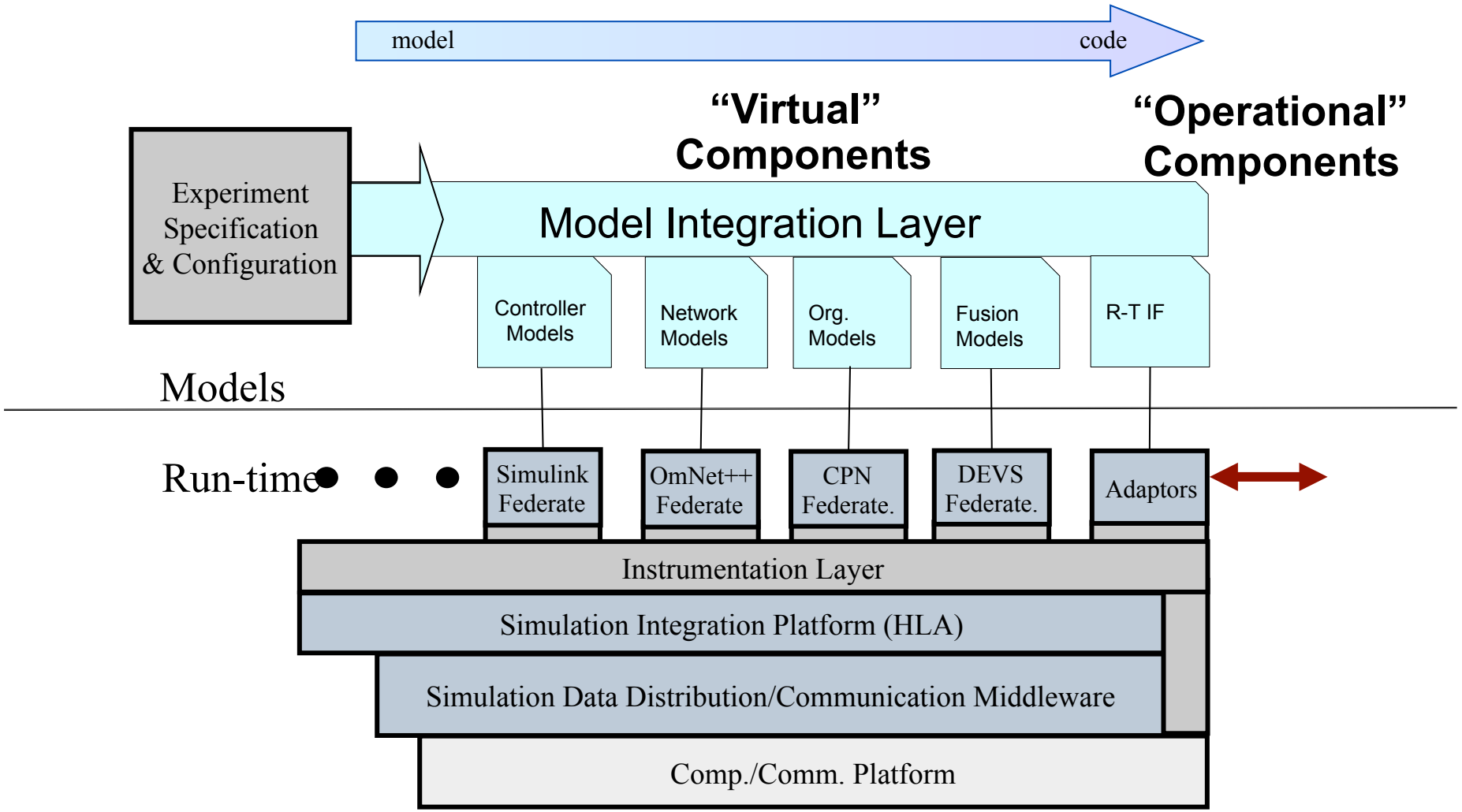
Simulation Interaction

Simulation Architecture



Network Architecture

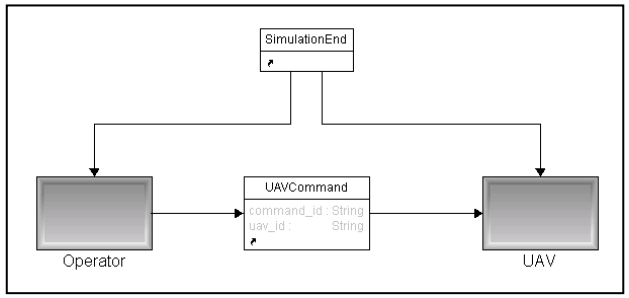
Integration Architecture



Integration Models: Dataflow and Data

Integration Models

Interaction Models
(data flow, timing, parameters)



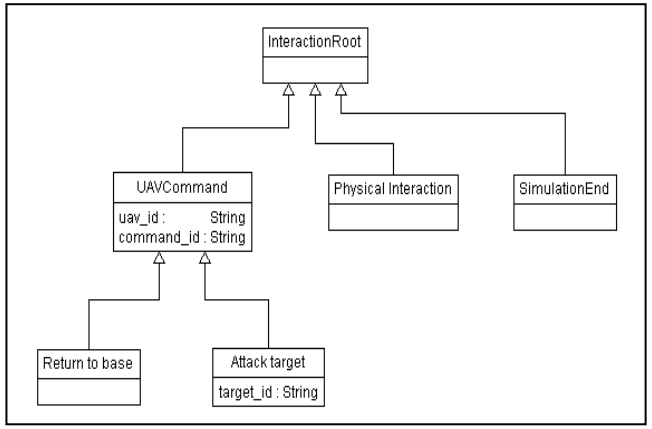
configuration

Based on integration model configuration files are generated for the various simulation components.
- *configure how the component is connected to the simulation (input-output binding)*

Domain specific simulation components

OMNET component	*CPN component*
Simulink component	*Delta3D…*

Data models
(interaction and object models)



transformation

Federates have to have a common data model to be able to share data.
- *data model can be imported from domain specific models*
- *domain specific models can be generated from data models*

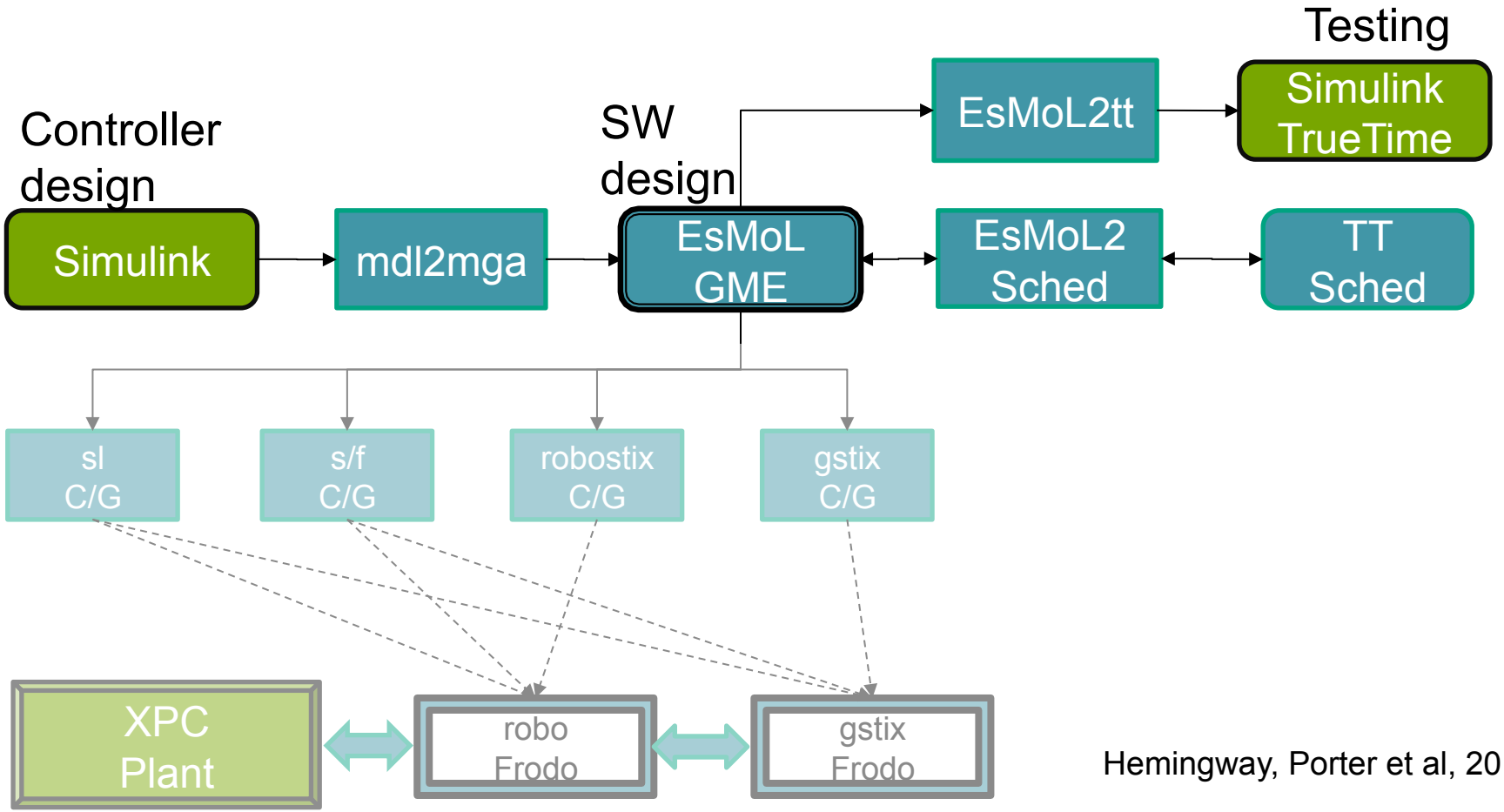
Domain specific simulation models

Omnet models	*CPN models*
Simulink models	*Delta3D…*

Hemingway et al, 2010

Example: Enforced Interaction Between DSMLs

Model-based testing of dynamics requires models that reflect effects of SW implementation



Hemingway, Porter et al, 2010

Summary

- Composing/integrating modeling languages (and models) is integral part of model-based design
- Many techniques are emerging – aligned with underlying semantics of component languages
- Need for model integration patterns and tools