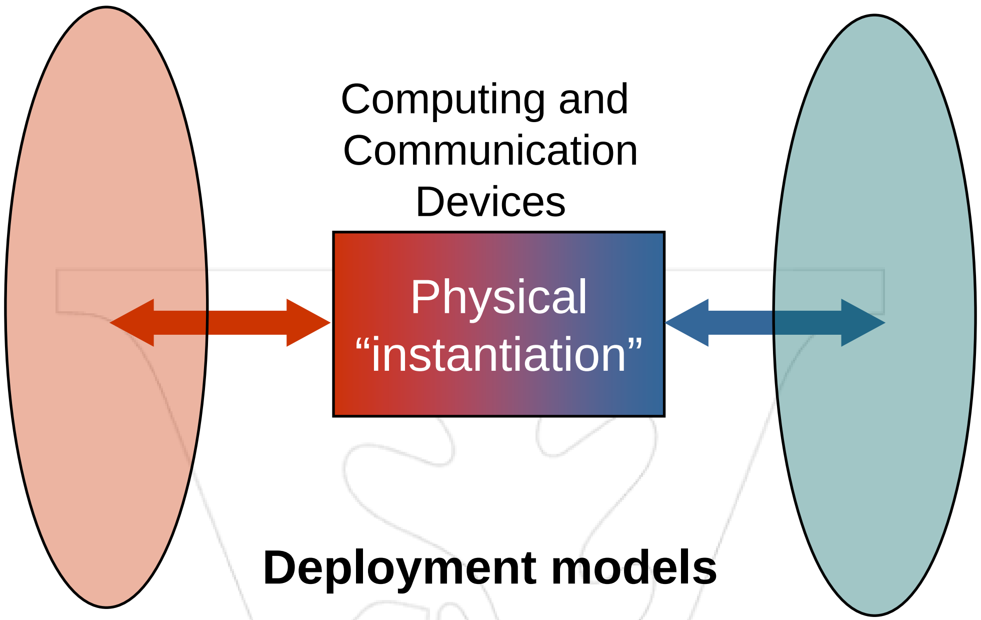
Towards the Compositional Specification of Semantics for Heterogeneous DSML-s

Janos Sztipanovits
ISIS, Vanderbilt University

October 26, 2006

Composition Domains

Computation system composition domains:
- SW functional components
- MoC abstractions
- Comm. abstractions



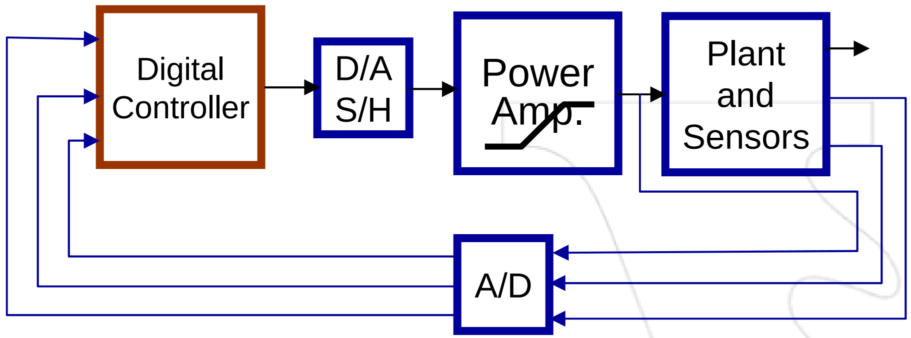
Physical system composition domains:
- Dynamics
- Power
- Comp. Arch.
- Comm. Arch.

- Detailed physical characteristics of the devices (phys. architecture, speed, bus structure, bandwidth, …)
- Interactions between code and physical behavior (speed, power dissipation)
- Lower layers of code interact with application code (scheduler, memory manager, middleware services, …)
- Interference across modules due to shared physical resources

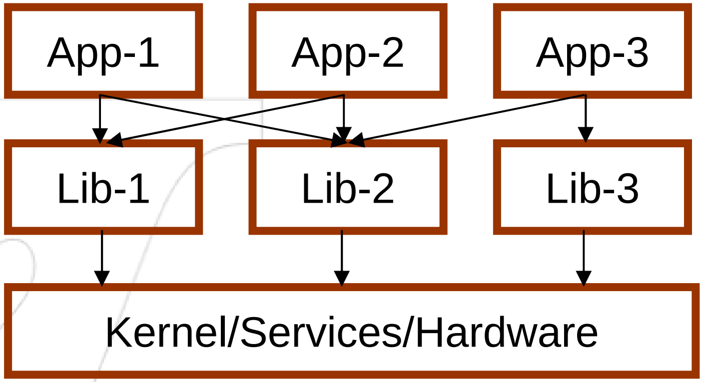
Component-based design with cross-cutting constraints is a very hard problem

Design Aspects Are Not Orthogonal

Controller Dynamics



Embedded Software



- control law
- tolerated error, stability
- sampling rate

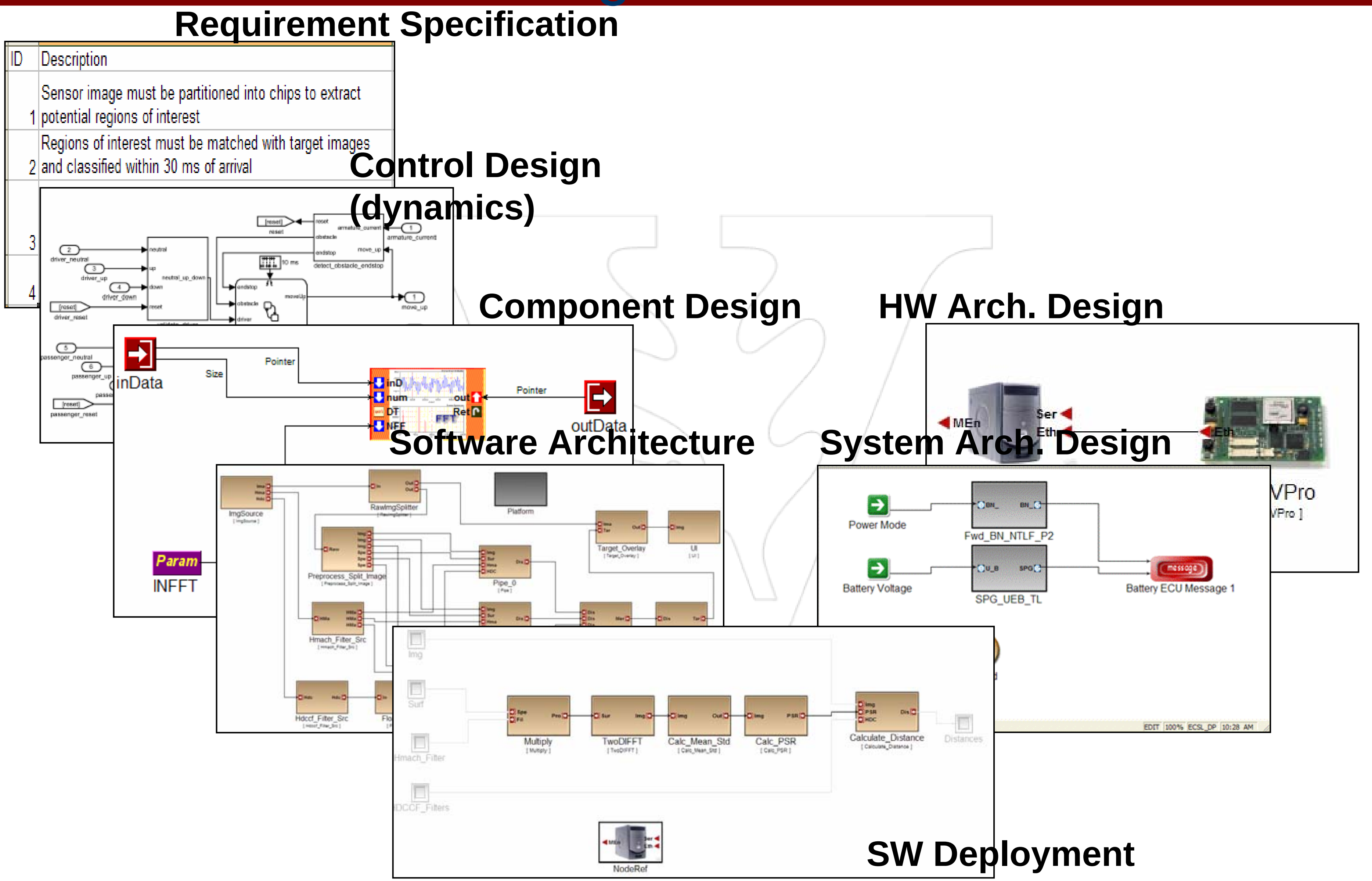
- limit-cycle oscillation
- loop delay
- noise

- HW/SW architecture
- Data types selection
- Scheduling policy,...

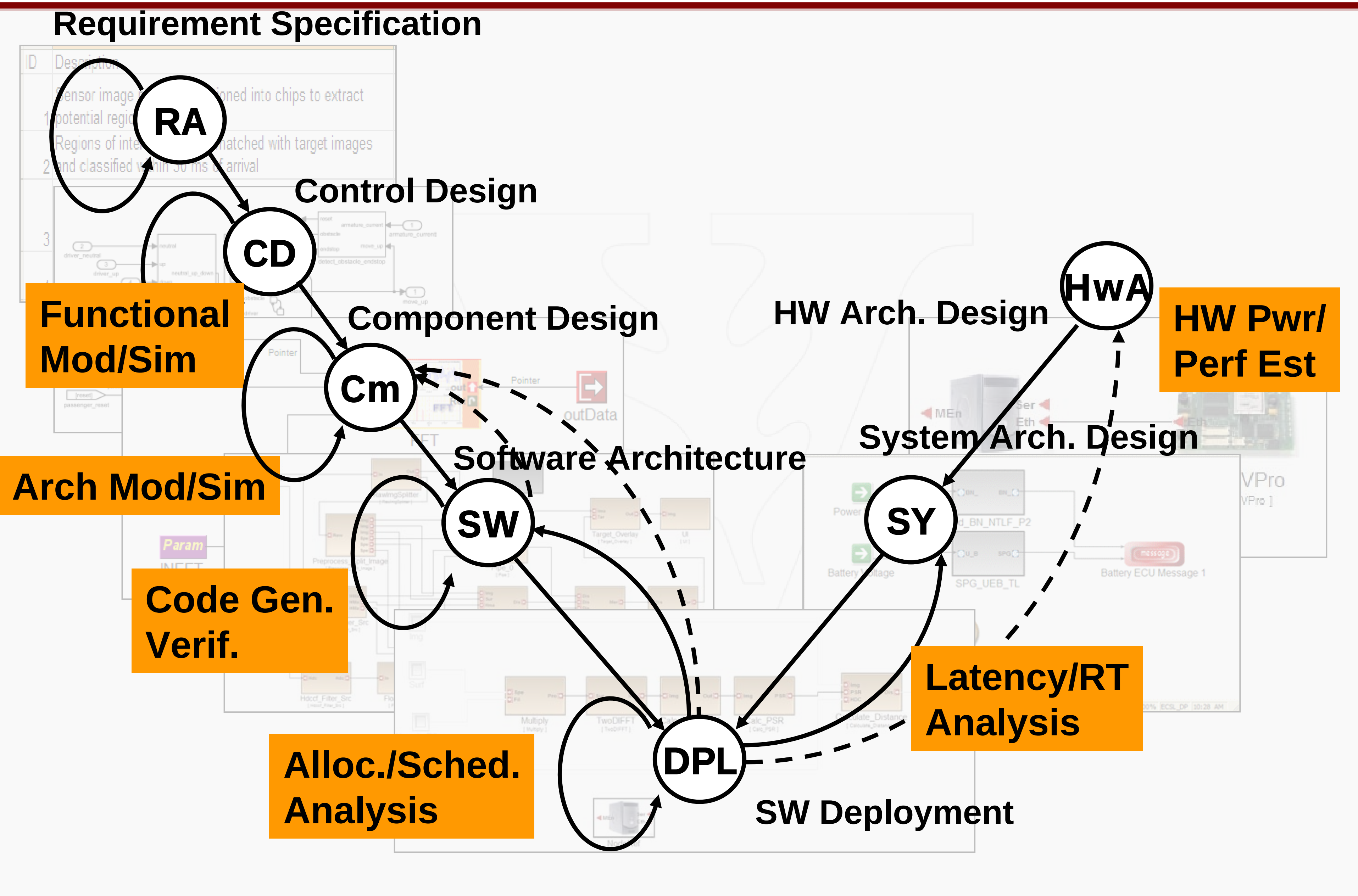
- Numeric accuracy*
- Latency
- Jitter

*Quantization, saturation, truncation,..

Design Aspects in a Simplified ES Design Flow



Design Flow: Tools and Analysis



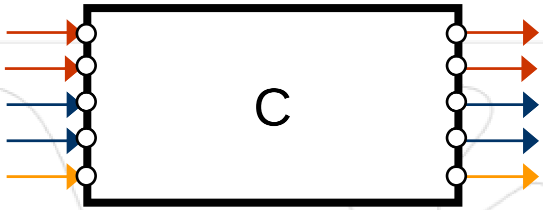
First Attempt to Answer Questions

- Q1: What are the basic concepts for describing components?
  Several, structured in different design aspects and defined by metamodels.
- Q2: What types of component interaction are supported?
  Several, structured in different design aspects and defined by metamodels and semantics.
- Q3: What kind of resources can be modeled and are they first class citizens of the formalism?
  There are modeling aspects focusing on resources and there are component attributes in other aspects the establish the links.
- Q4: How do you think the following models, styles and design principles are interrelated and can be combined:
  - synchrony v.s. asynchrony: essential for heterogeneous, networked systems
  - event-triggered/data-triggered/time triggered: all needed
  - separation of concerns: **this is the crucial point**

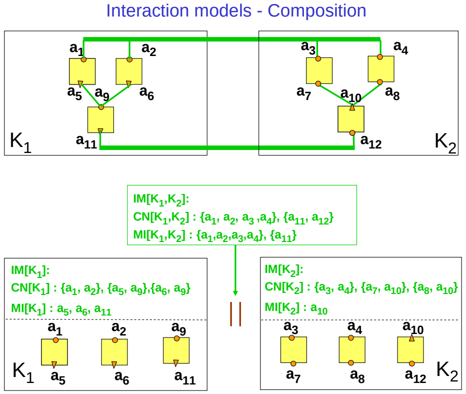
Approach-1: Components Are Single Artifacts

Component: locus of activities interacting with other components via well defined interfaces

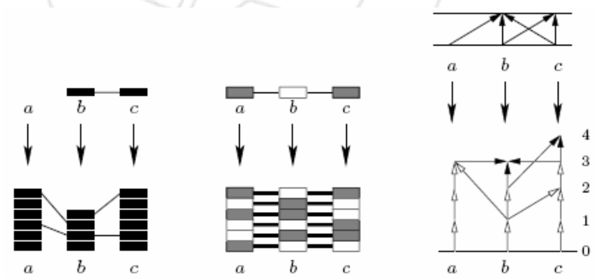
- Enriched interfaces
- Precise interaction models
- Heterogeneity



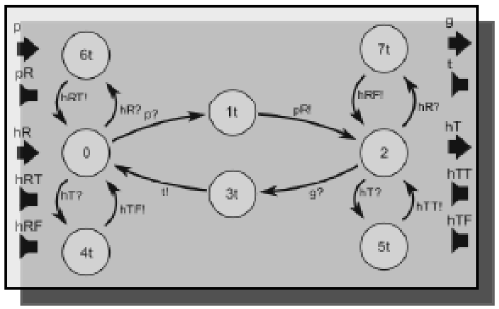
The component concept is extended with more elaborate interfaces to enable system composition along multiple design aspects.



Graphics taken from SIFAKIS, J. Modeling Real-Time Systems seminar presentation, Vanderbilt, June 5, 2005



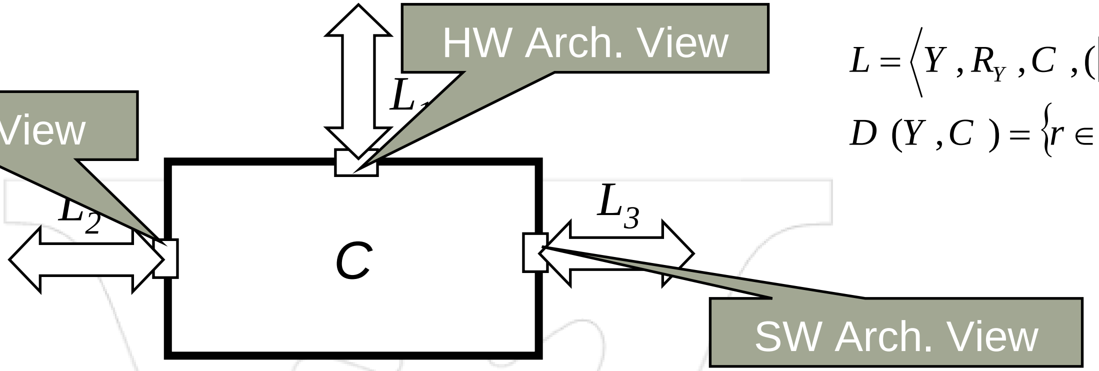
Graphics taken from BENVENISTE, A., CAILLAUD, B., CARLONI, L.P., and SANGIOVANNI-VINCENTELLI, A.L. Tag Machines *Proceedings of the Fifth International Conference on Embedded Software (EMSOFT), 2005*



LEE, E. A., and Xiong, Y., "A Behavioral Type System and Its Application in Ptolemy II," *Aspects of Computing Journal,* special issue on "Semantic Foundations of Engineering Design Languages."

Approach-2: Components Are Design Spaces

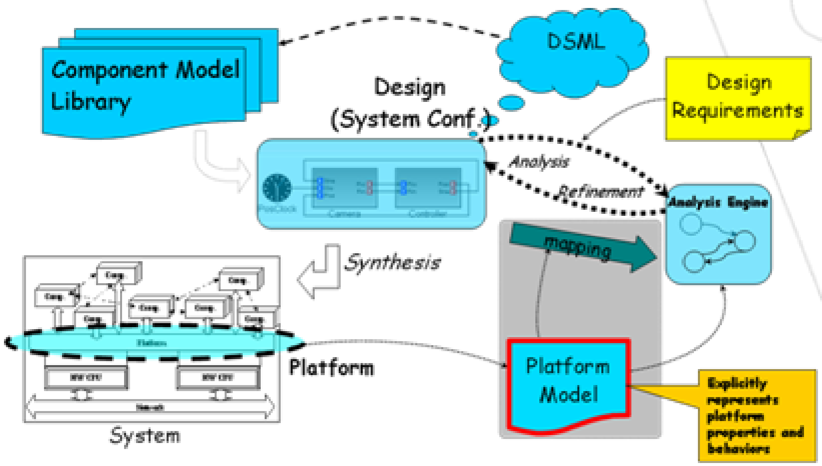
Component: is a design space defined by a set of interacting modeling aspects



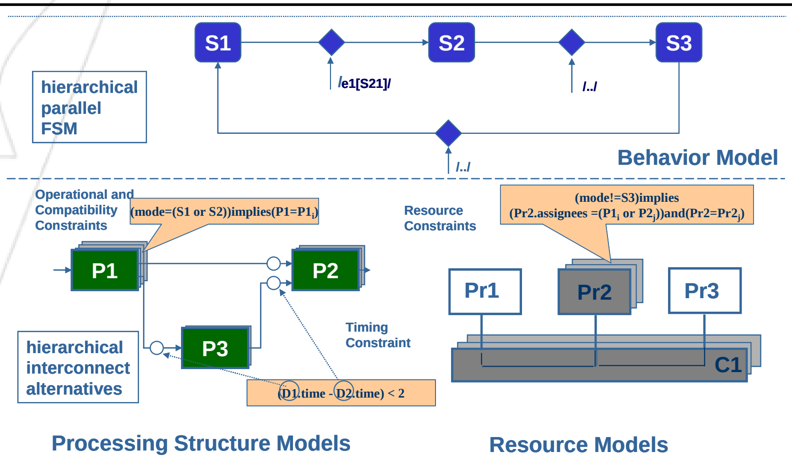
$$L = \langle Y, R_Y, C, (\llbracket\ \rrbracket_i)_{i \in J} \rangle$$

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

Interaction among modeling aspects is defined by some transformation (e.g. $T = (R_{Y_1} \times R_{Y_2}) \mapsto R_{Y_3}$) or by constraints over the design space (e.g. $D'(Y', C) = \{ r \in (R_{Y_1} \times R_{Y_2} \times R_{Y_3}) \mid r \models C \}$)

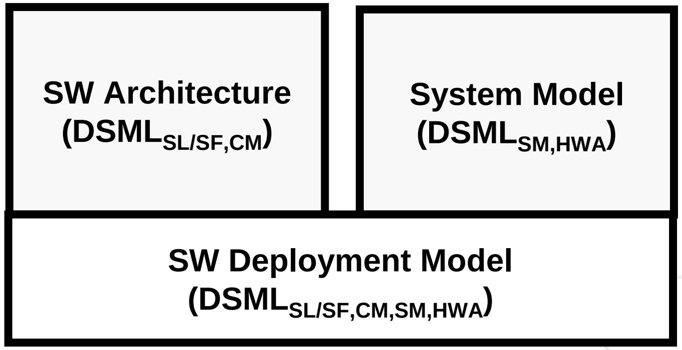


T. Szemethy, G. Karsai, "Platform Modeling and Model Transformations for Analysis," *Journal of Universal Computer Science*, vol. 10, no. 10, pp 1383-1406, 2004.



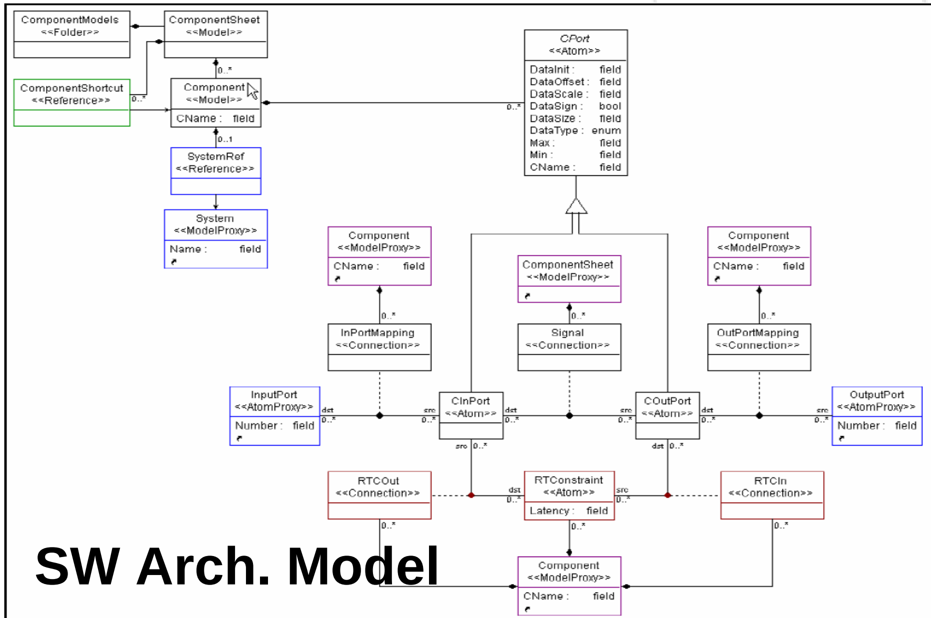
Neema S., Sztipanovits J., Karsai G., .Ken Butts: **Constraint-Based Design-Space Exploration and Model Synthesis** , EMSOFT 2003, LNCS 2855, Philadelphia, PA, October 2, 2003.

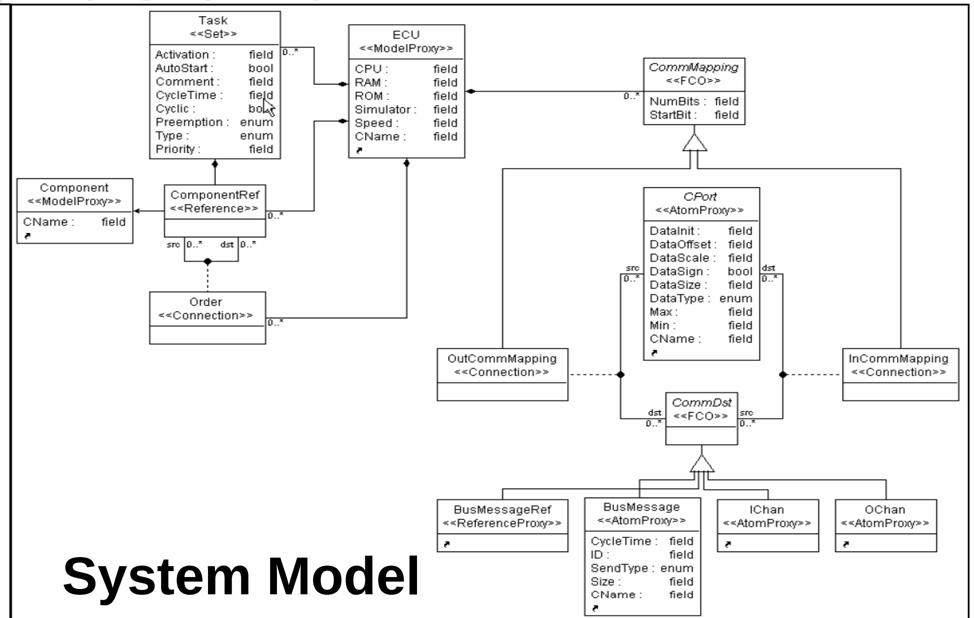
Design Space Specification and Composition Requires DSML Composition



Deployment Example: SW Architecture Model needs to be composed with System Architecture Model by allocating SW components to OSEK Tasks and Communication Channels.

Tools: GME, AIRES (schedulability), CANoe (Bus emulator)





SW Deployment: SW Components – System Mapping

Structural Semantics Is Important

$$L = \langle Y, R_Y, C, (\llbracket\ \rrbracket_i)_{i \in J} \rangle$$

$$\llbracket\ \rrbracket : R_Y \mapsto R_{Y'}$$

Structural Interpretation :

$$(\llbracket r \rrbracket = \{true\}) \Leftrightarrow (r \models C)$$

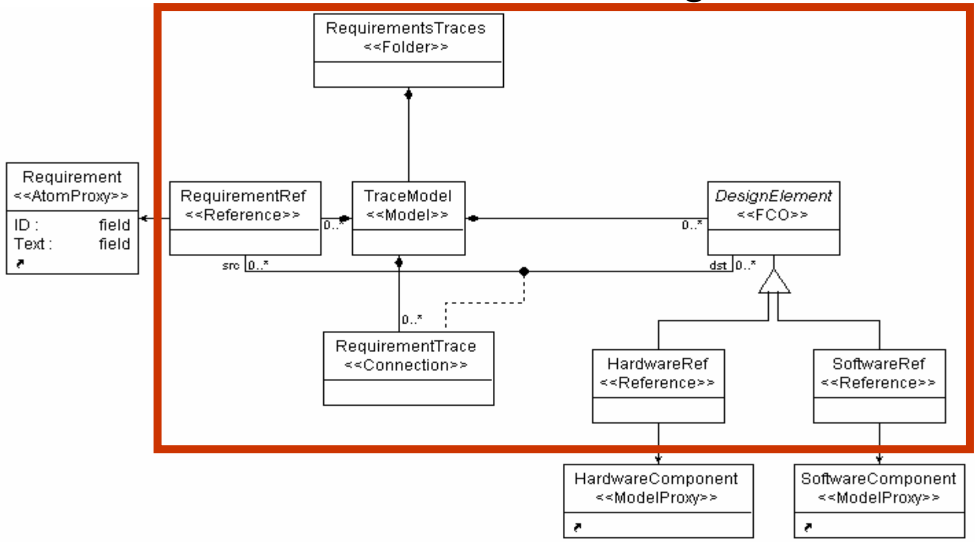
$$(r \not\models C) \Leftrightarrow (\llbracket r \rrbracket = \{false\}).$$

Jackson, Sztipanovits
EMSOFT’06

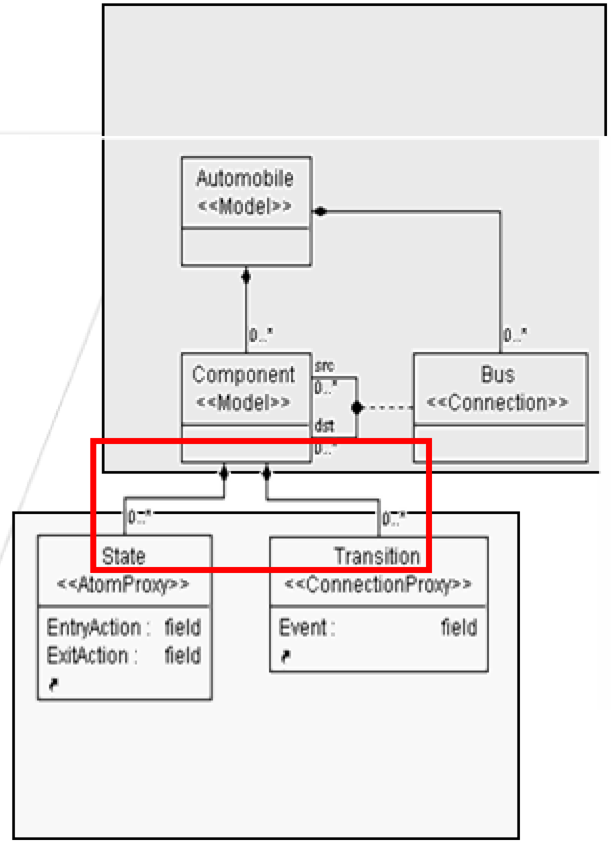
- DSML Composition (metamodel composition) methods in the Generic Modeling Environment (GME):
  - Class Merge
  - Metamodel Interfacing
  - Class Refinement
  - Template Instantiation
  - Metamodel Transformations
- Analysis Tools:
  - OCL constraint checker
  - FORMULA (Jackson)

Structural Composition is Supported by Metamodeling Abstractions

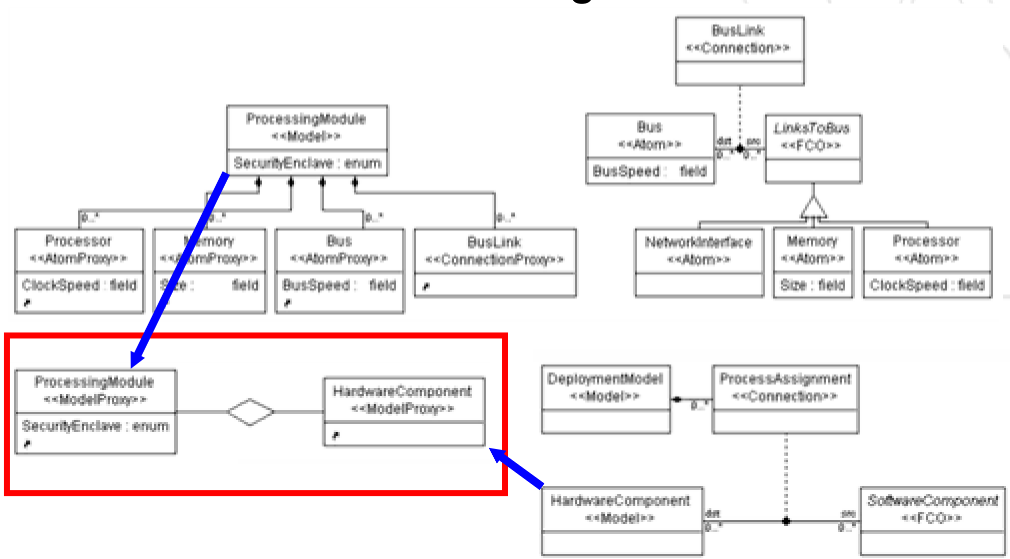
Metamodel Interfacing



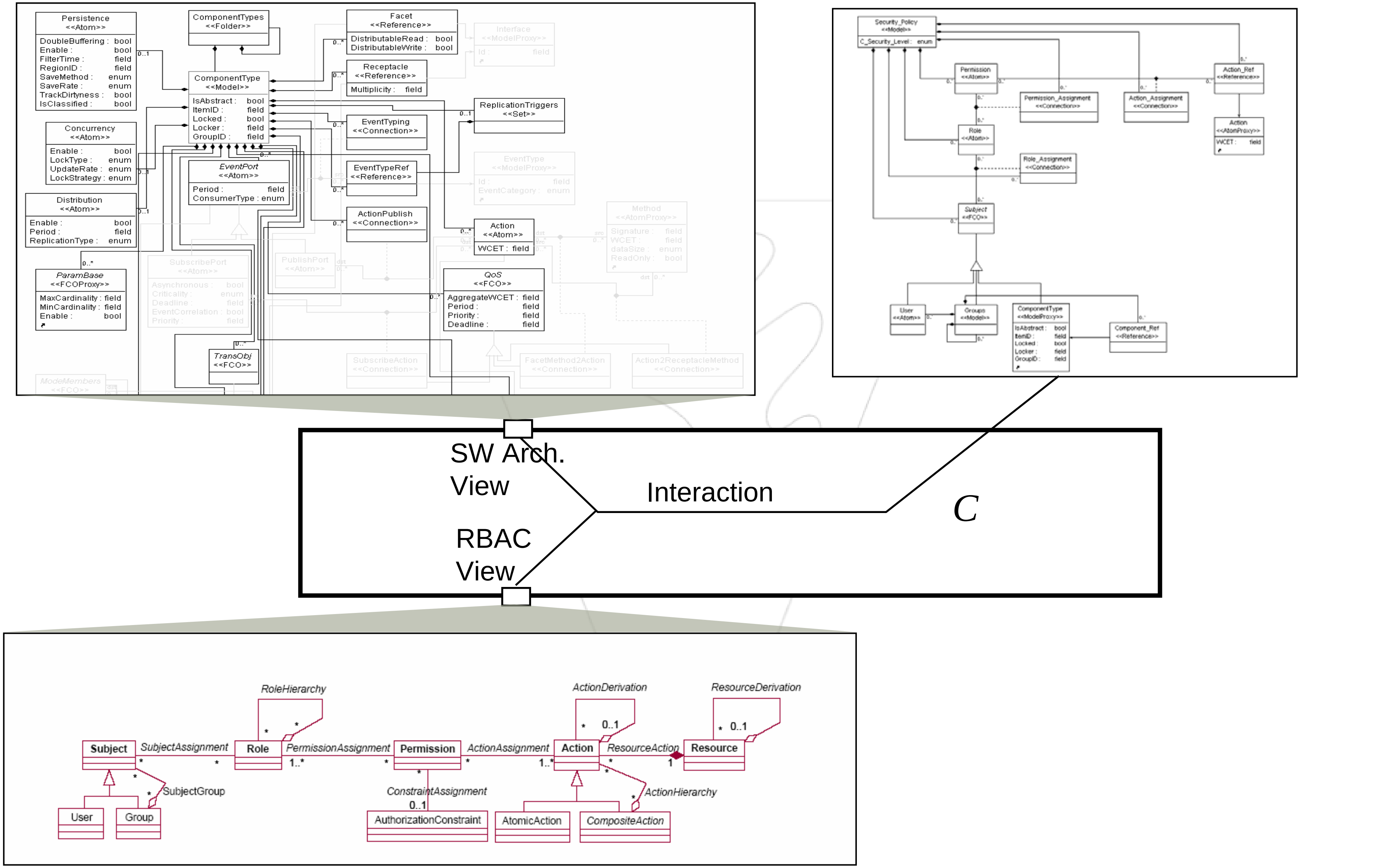
Class Refinement



Class Merge



There Are Many Interesting Applications



Composition of Behavioral Semantics

- Given a DSML

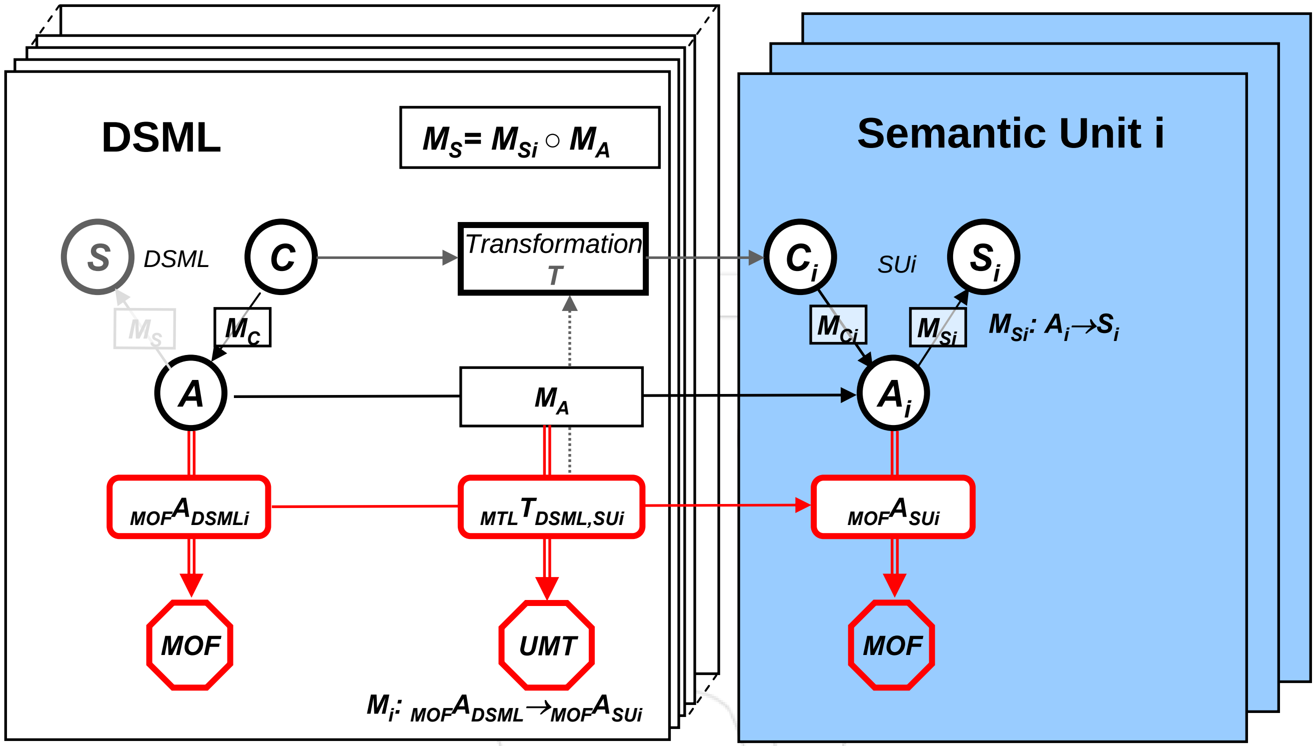
$$L = \left\langle Y, R_Y, C, (\llbracket\ \rrbracket_i)_{i \in J} \right\rangle$$

the transformational interpretation $\llbracket\ \rrbracket^T$ is a mapping:

$$\llbracket\ \rrbracket^T : R_Y \mapsto R_{Y'}$$

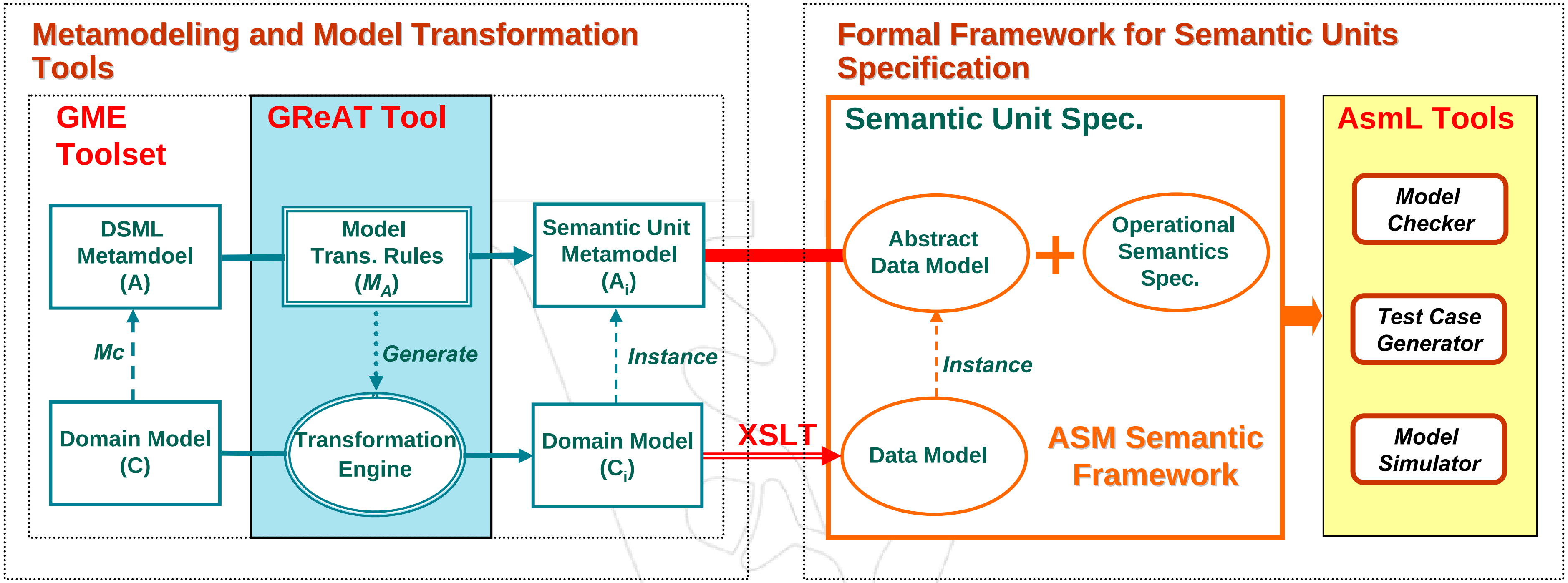
- Behavioral semantics will be defined by specifying the transformation of the DSML models to models with operational semantics.
- Goal: Semantically robust design environment for composing DSML-s

DSML Design Through Semantic Anchoring



- Step 1
  - Specify the DSML $<A, C, M_c>$ by using MOF-based metamodels.
- Step 2
  - Select appropriate semantic units $L = <A_i, C_i, \mathrm{M}_{\mathrm{Ci}}, \mathrm{S}_{\mathrm{i}}, \mathrm{M}_{\mathrm{Si}}>$ for the behavioral aspects of the DSML.
- Step 3
  - Specify the semantic anchoring $\mathrm{M}_\mathrm{A} = \mathrm{A} \text{ -> } \mathrm{A}_\mathrm{i}$ by using UMT.

Experimental Tool Suite for Semantic Anchoring



- Metamodeling and Model Transformation Tools
  - GME: Provide a MOF-based metamodeling and modeling environment.
  - GReAT: Build on GME for metamodel to metamodel transformation.
- Tools for Semantic Unit Specification
  - ASM: A particular kind of mathematical machine, like the Turing machine. (Yuri Gurevich)
  - AsmL: A formal specification language based on ASM. (Microsoft Research)

Example: Synchronous Data Flow

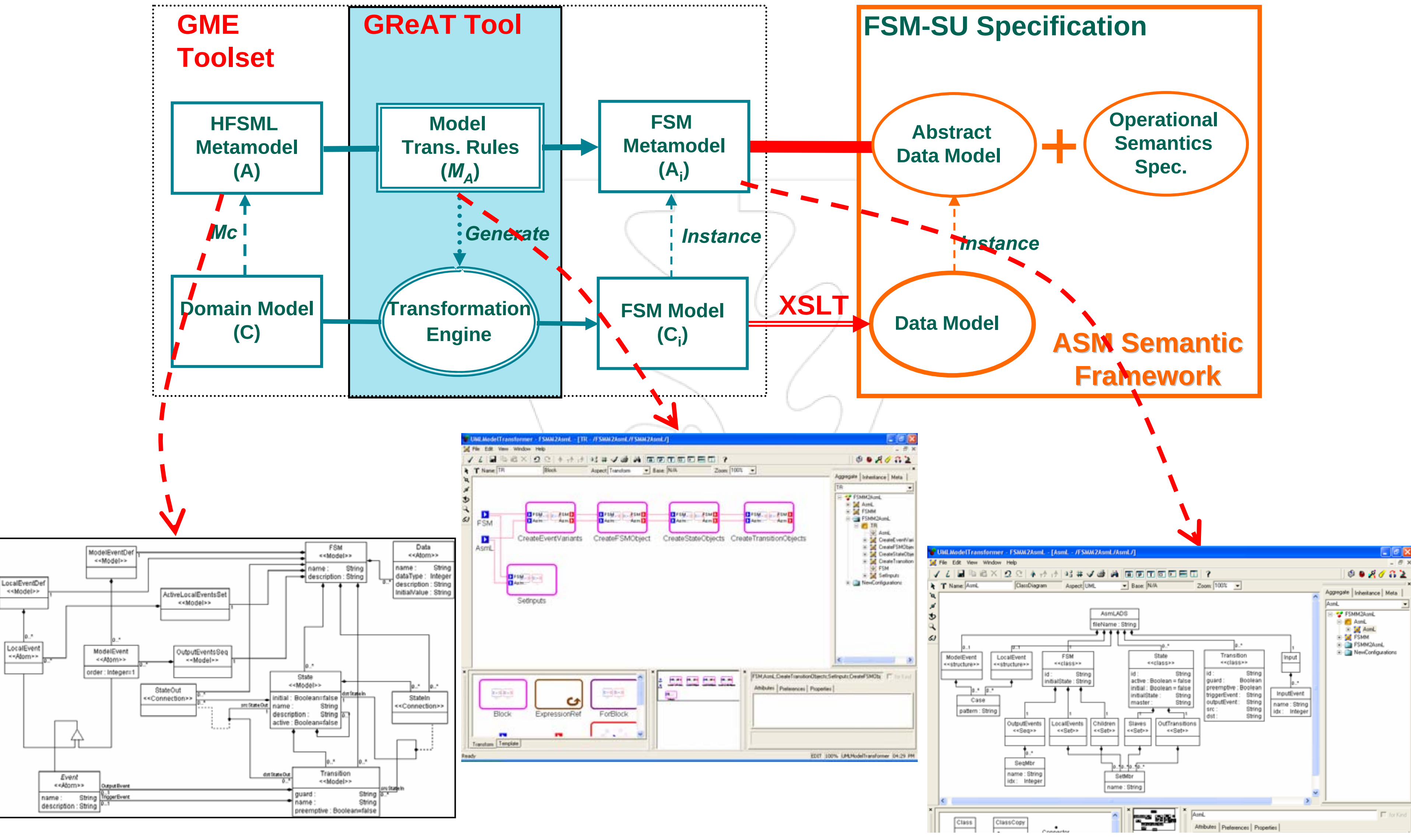
Abstract Data Model

```
structure Value
  case IntValue
    v as Integer
  case DoubleValue
    v as Double
  case BoolValue
    v as Boolean

//Data Token, it may contain a value or a null data
structure Token
  value as Value?

//Data Port, when exist is true, the port has an effective data token
class Port
  id        as String
  var token as Token   = Token (null)
  var exist as Boolean = false

//Data Channel connecting two data ports
class Channel
  id as String
  srcPort as Port
  dstPort as Port

//A Node is a basic unit is the Data Flow. It may be an action or a Guard
abstract class Node
  id as String

  abstract property inputPorts  as Seq of Port
    get
  abstract property outputPorts as Seq of Port
    get
  //The Run method takes tokens from its input ports, do actions and set output
  //tokens in the output ports
  abstract Fire ()


//Dynamic Data Flow Semantic Unit
abstract class SDF
  id as String

  abstract property nodes as Set of Node
    get
  abstract property channels as Set of Channel
    get
  abstract property inputPorts as Seq of Port
    get
  abstract property outputPorts as Seq of Port
    get
```

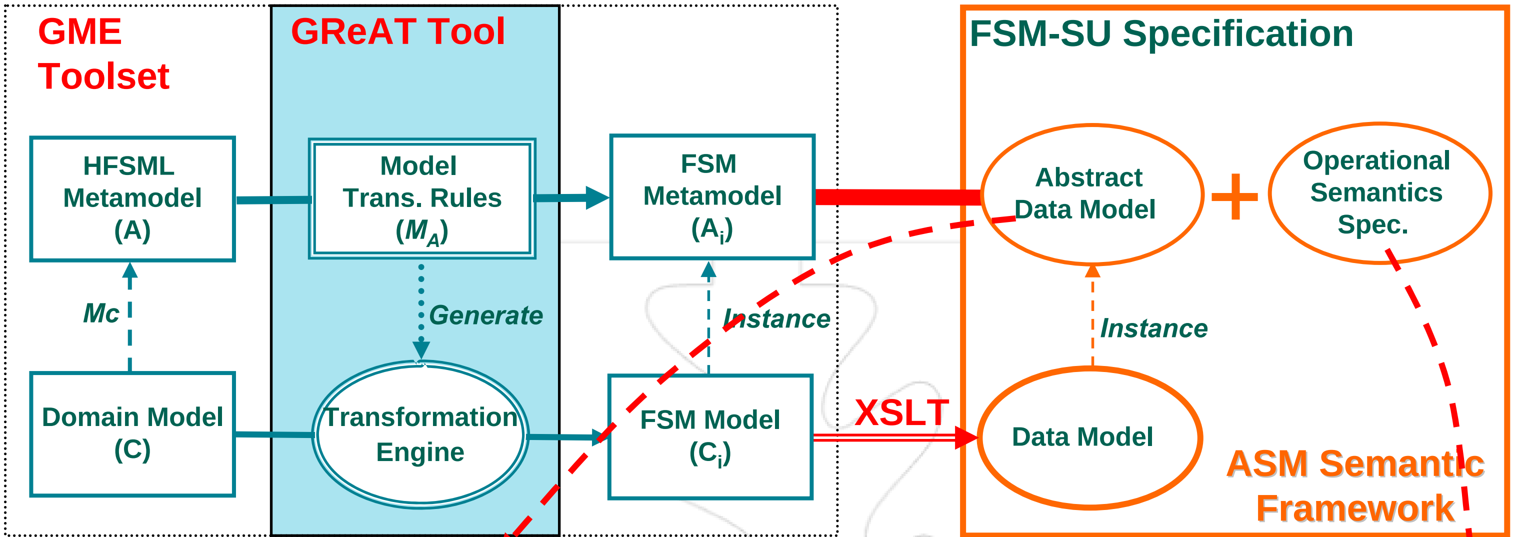
Model Interpreter

```
Run (n as Node)
    require n in me.EnabledNodes ()
    step
      n.Fire ()
    step
      if exists p in n.inputPorts where p.exist then
        error ("After the firing of a node, all input tokens should be consumed
by the node.")
    step
      if exists p in n.outputPorts where not p.exist then
        error ("After the firing of a node, each of its output port should have
one output token.")
    step
      forall c in me.channels where c.srcPort.exist
        if c.dstPort.exist then
          error ("A input port receives more than one token.")
        else
          WriteLine ("Channel " + c.id + " is sending data tokens.")
          c.dstPort.token := c.srcPort.token
          c.dstPort.exist := true
        c.srcPort.exist := false

  //Return all nodes in the SDF that have all its required data tokens to fire.
  EnabledNodes () as Set of Node
    return {n | n in me.nodes where forall p in n.inputPorts where p.exist}

  Initialize ()
    forall p in me.inputPorts where p.exist
      forall c in me.channels where p.id = c.srcPort.id
        c.dstPort.token := c.srcPort.token
        c.srcPort.exist := false
        c.dstPort.exist := true

  ClearPorts ()
    forall c in me.channels
      if c.srcPort.exist then
        c.srcPort.exist := false
      if c.dstPort.exist then
        c.dstPort.exist := false
```

Example: HFSML => FSM-SU



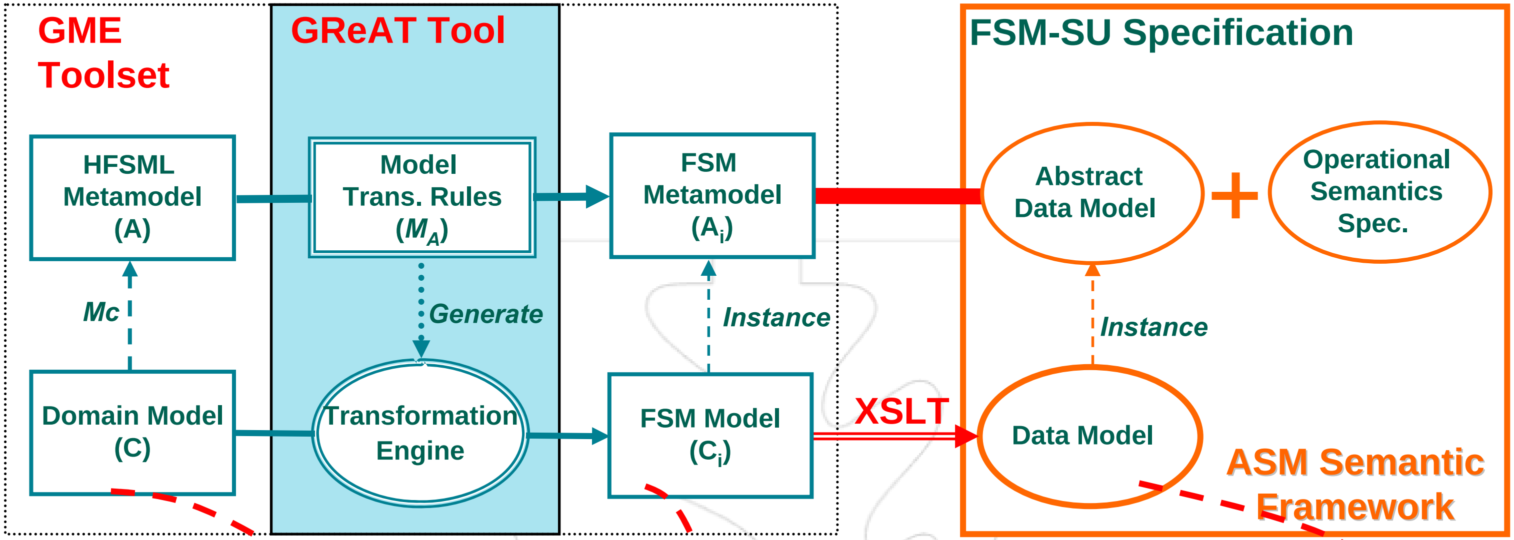
Example: HFSML => FSM-SU

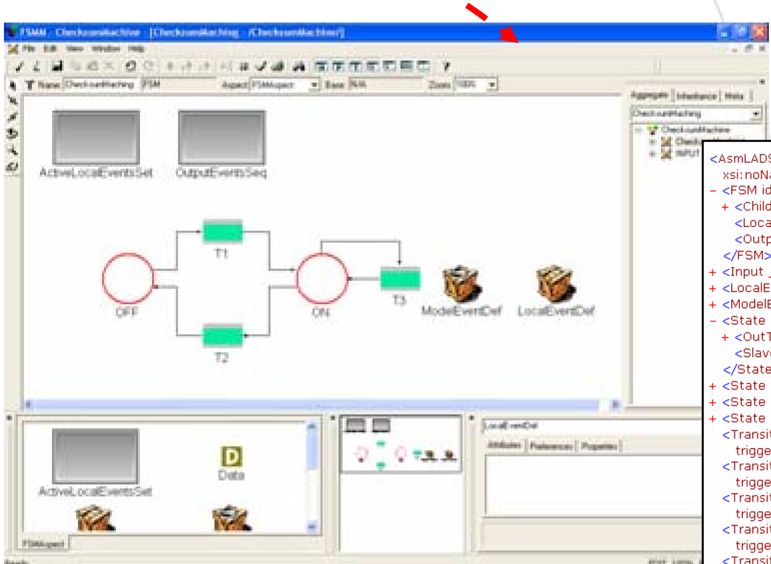



```
structure Event
  eventType as String

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
```

Example: HFSML => FSM-SU





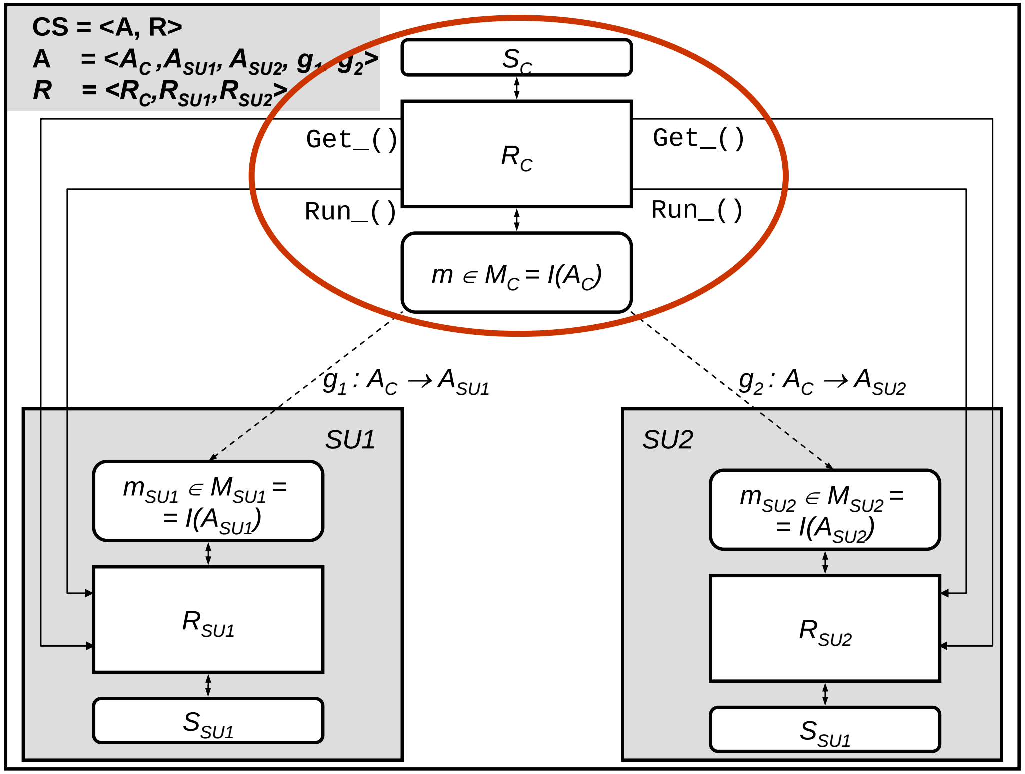

```
<AsmLADS _id="id988" fileName="" xmlns:xsi="http://www.w3.org/
  xsi:noNamespaceSchemaLocation="UDM\AsmL.xsd">
- <FSM id="ChecksumMaching" _id="id9d5" initialState="OFF">
  + <Children _id="id9f4">
    <LocalEvents _id="id9e5" />
    <OutputEvents _id="id9e0" />
  </FSM>
+ <Input _id="id98b">
+ <LocalEvent _id="id9bf">
+ <ModelEvent _id="id9ad">
- <State id="OFF" _id="ida17" active="false" master="" initial="true"
  + <OutTransitions _id="ida5b">
    <Slaves _id="ida3d" />
  </State>
+ <State id="ON" _id="ida18" active="false" master="" initial="false"
+ <State id="ZERO" _id="ida74" active="false" master="ON" initial="false" initialState="">
+ <State id="ONE" _id="ida75" active="false" master="ON" initial="false" initialState="">
  <Transition id="T11" _id="idade" dst="ONE" src="ZERO" guard="true" preemptive="false" outputEvent=""
    triggerEvent="LocalEvent.one" />
  <Transition id="T12" _id="idadf" dst="ZERO" src="ONE" guard="true" preemptive="false" outputEvent=""
    triggerEvent="LocalEvent.one" />
  <Transition id="T13" _id="idae0" dst="ZERO" src="ZERO" guard="true" preemptive="false" outputEvent=""
    triggerEvent="LocalEvent.zero" />
  <Transition id="T14" _id="idae1" dst="ONE" src="ONE" guard="true" preemptive="false" outputEvent=""
    triggerEvent="LocalEvent.zero" />
  <Transition id="T1" _id="idb0e" dst="ON" src="OFF" guard="true" preemptive="false" outputEvent="ModelEvent.start"
    triggerEvent="" />
  <Transition id="T2" _id="idb0f" dst="OFF" src="ON" guard="true" preemptive="false" outputEvent="" triggerEvent="ModelEvent.stop" />
  <Transition id="T3" _id="idb10" dst="ON" src="ON" guard="true" preemptive="false" outputEvent=""
    triggerEvent="ModelEvent.reset" />
</AsmLADS>
```


```
initStateAutomaton() as StateAutomaton
  let S1 = State("S1", true)
  let S2 = State("S2", false)
  let T1 = Transition("T1")
  let e1 = Event("e1")
  let S = {S1, S2}
  let T = {T1}
  let E = {e1}
  let Connections = {T1 -> (S1, S2)}
  let TriggerEvent = {T1 -> e1}
  let OutputEvent = {T1 -> e1}
  let InitialState = S1
  return new StateAutomaton(S,T,E,Connections,TriggerEvent,OutputEvent,
InitialState)
```

Heterogeneous DSMLs

- The semantics of a heterogeneous DSML is probably not captured by a single predefined semantic unit.
- Heterogeneity of systems
  - Complex systems are composed of heterogeneous components using heterogeneous interactions. Modeling and design of heterogeneous systems is a significant challenge.
- Heterogeneity of tool chains
  - Tool chains supporting domain-specific design flows integrate modeling, analysis and synthesis tools using DSMLs with overlapping semantics.

Compositional Specification of Semantics



Remark: The behavioral composition specifies a controller, which restricts the executions of actions. Since the behavior of the component semantic units can be described as partial orders on the sets of actions (POMSET) they can perform, the behavioral composition is modeled mathematically as a composition of POMSETs (Pratt).

- Structural Composition yields the composed Abstract Data Model, $A = < A_C, A_{SU1}, A_{SU2}, g_1, g_2 >$ where *g1*, *g2* are the partial maps between concepts in A_C, A_{SU1}, and A_{SU2} .
- Behavioral composition is completed by the R_C set of rules that together with R_{SU1} and R_{SU2} form the R rule set for the composed semantics.

Road Ahead

- Continue in deepening the theory and expanding the scope of the compositional specification of semantics.
- Extend the semantic anchoring tools toward becoming a DSML Design Tool Suite.
- Further research on design space composition.