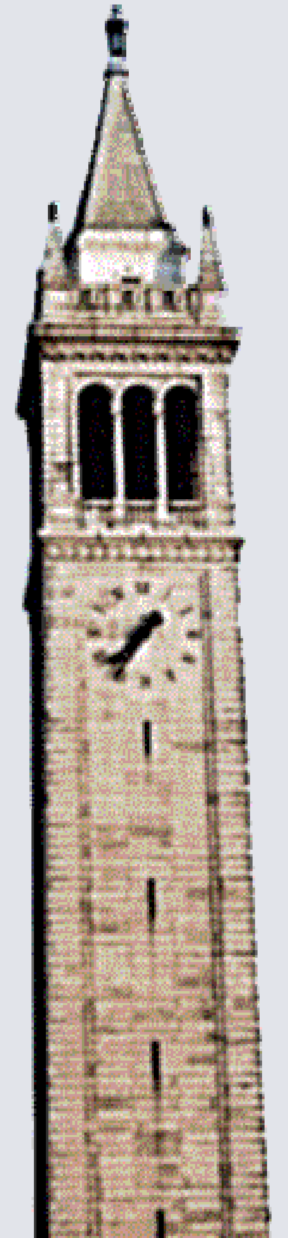
Model-Based Design

Edited and presented by
Janos Sztipanovits
Vanderbilt University

Chess Review
October 4, 2006
Alexandria, VA



Model-Based Design

Model-based design focuses on the *formal representation, composition, and manipulation of models* during the design process.

System Composition Approaches

Component Behavior	**Modeled on different levels of abstraction:** •Generalized transition systems (FSM, Time Automata, Cont. Dynamics, Hybrid), fundamental role of time models • Precise relationship among abstraction levels • Research: dynamic/adaptive behavior
Interaction	**Expressed as a system topology :** • Module Interconnection (Nodes, Ports, Connections) • Hierarchy • Research: dynamic topology **Describes interaction patterns among components:** • Set of well-defined Models of Computations (MoC) (SR, SDF, DE,…) • Heterogeneous, precisely defined interactions • Research: interface theory (time, resources,..)
Scheduling/ Resource Mapping	**Mapping/deploying components on platforms:** • Dynamic Priority • Behavior guarantees • Research: composition of schedulers

Tool Composition Approaches

Domain-Specific Tools; Design Environments	**Domain-Specific Design Flows and Tool Chains:** • ECSL - Automotive • ESML - Avionics • SPML - Signal Processing • CAPE/eLMS
Metaprogrammable Tools, Integration Frameworks	**MIC Metaprogrammable Tool Suite: (mature or in maturation program)** • Metamodeling languages • Modeling Tools • Model Transformations • Model Management • Design Space Construction and Exploration • Tool Integration Framework
Semantic Foundation	**Semantic Foundations (work in progress):** • Semantic Anchoring Environment (SAE) • Verification • Semantic Integration

Intersection of System and Tool Composition Dimensions

	Semantic Foundation;	Metaprogrammable Tools, Environments	Domain-Specific Tools, Tool Chains
Component Behavior	Semantic Units and Semantic Anchoring	Metamodels, Metamodel Composition & Metaprogrammable Tool Chain Composition	Model Composition in Domain-Specific Design Flows
Interaction	Compositional Semantics		
Resource Modeling (Schedule)			

Domain Specific Design Flows and Tool Chains

- Integration of tools into tool chains
  - ECSL – Control
  - ESML - Avionics
  - SPP - Signal Processing
  - FCS – Networked Embedded Systems
  - SCA – Software Defined Radio
- Integration among tool frameworks: Metropolis, Ptolemy II, MIC, Simulink/Stateflow, ARIES, CheckMate,…
- **www.escherinstitute.org**

Intersection of System and Tool Composition Dimensions

Component Behavior

Interaction

Resource Modeling (Schedule)

Semantic Units and Semantic Anchoring

Compositional Semantics

Metamodels, Metamodel Composition & Metaprogrammable Tool Chain Composition

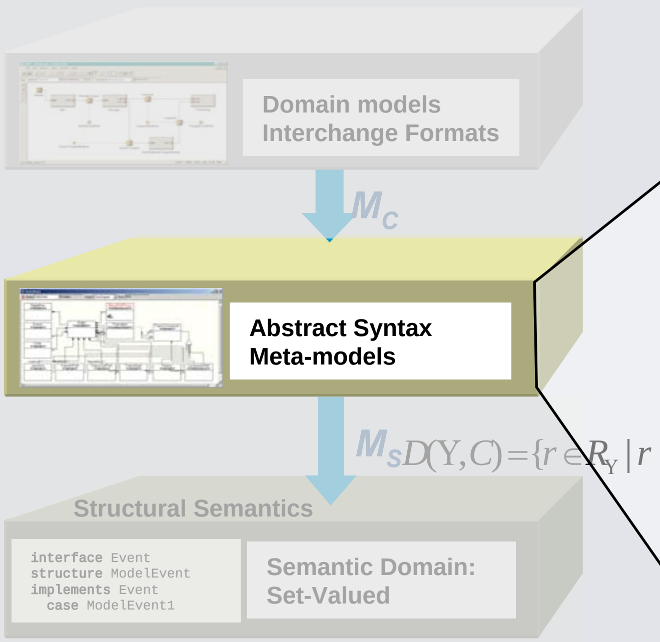
Model Composition in Domain-Specific Design Flows

Semantic Foundation;

Metaprogrammable Tools, Environments

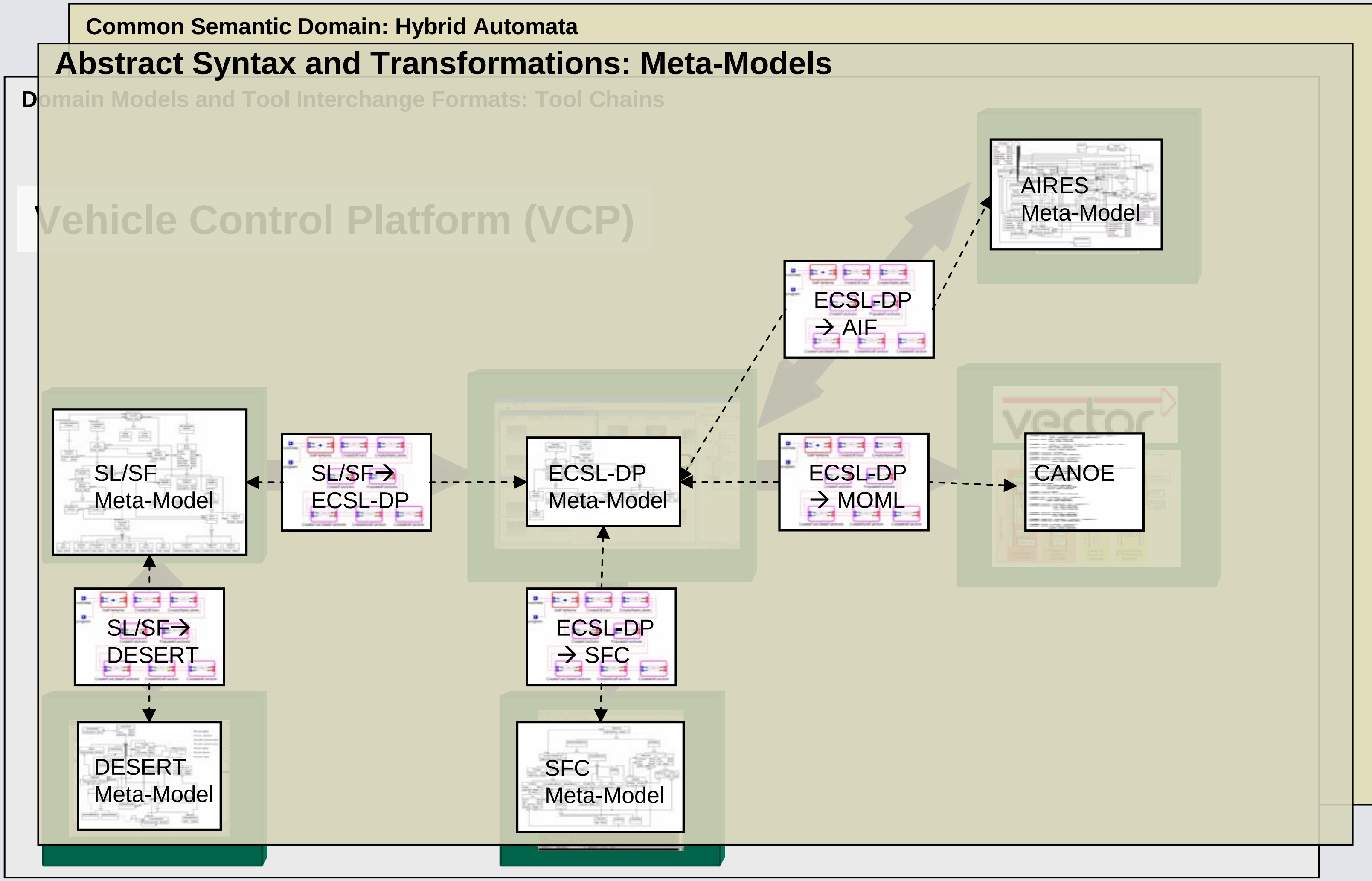
Domain-Specific Tools, Tool Chains

Syntactic Layer

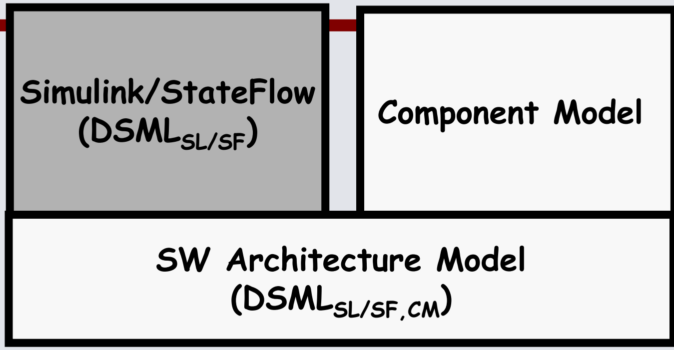


- Modeling & Metamodeling
- Model Data Management
- Model Transformation
- Tool Integration
- Design-Space Exploration

Metamodeling View of a Tool Chain



Need for Metamodel Composition:

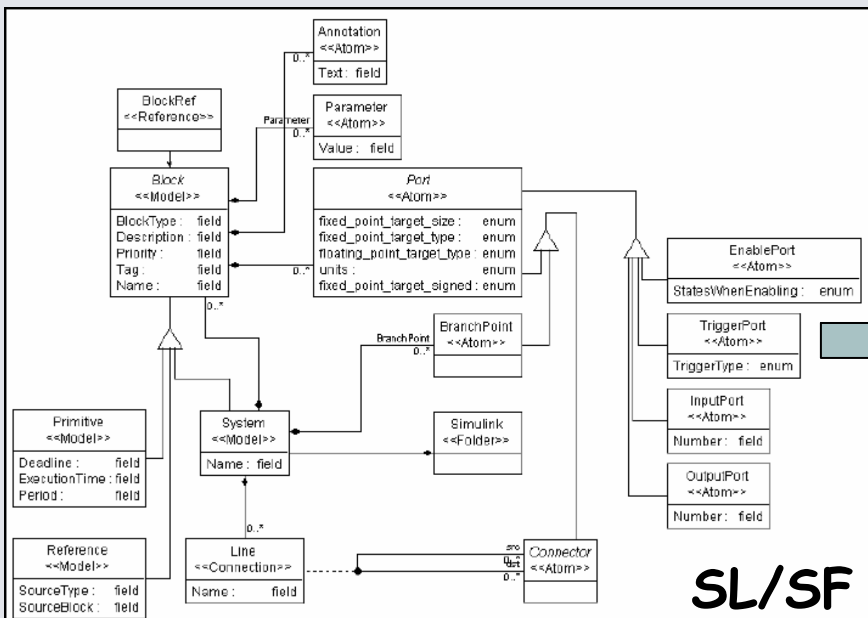


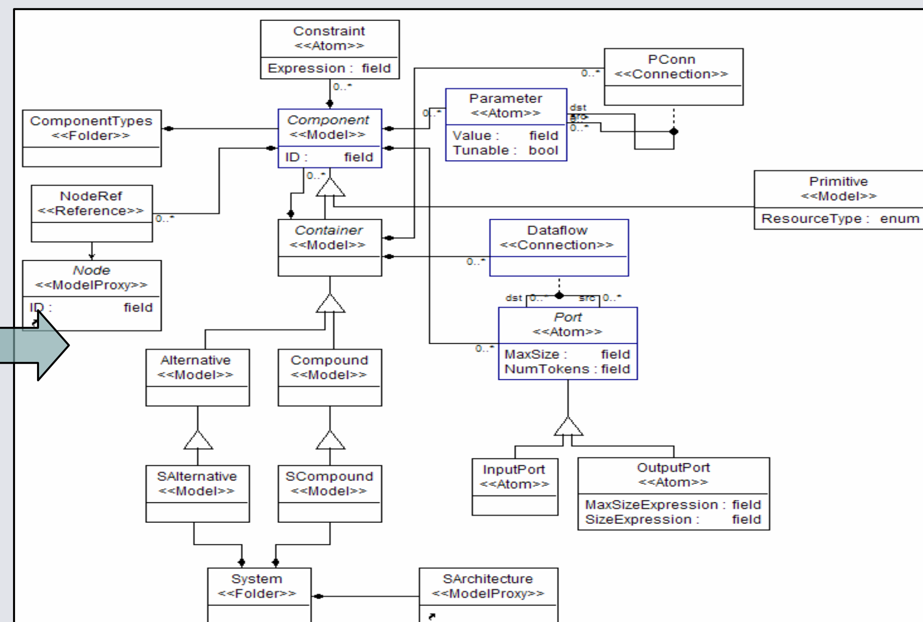
<u>Objective</u>: Optimize the SW architecture by selecting a component model and by allocating functions to components.

<u>Platform</u>: Heterogeneous Dataflow Component Model

<u>Tools:</u>

GME, GReAT, C Compiler, WCET Analyzer





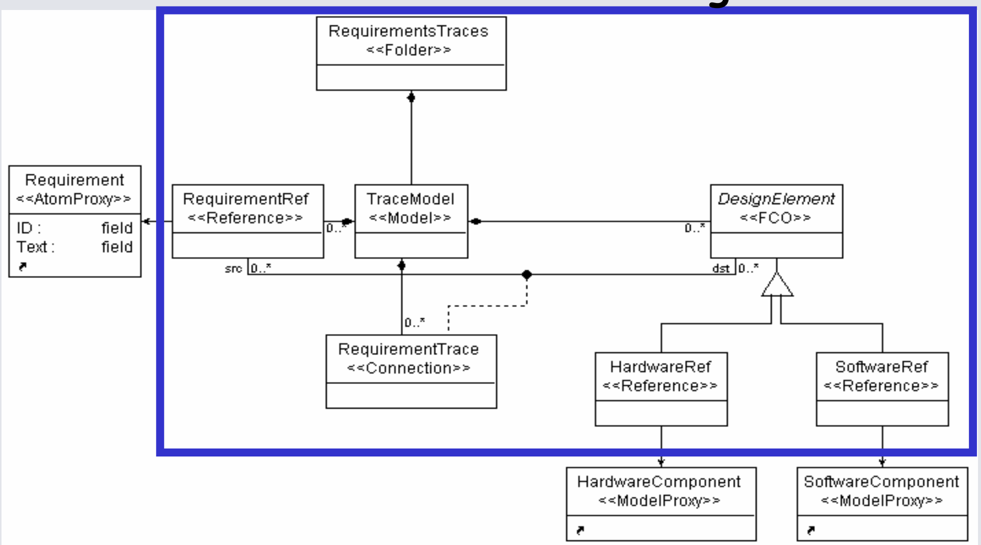
Functional blocks – SW Component Mapping

Solutions for Compositional Metamodeling

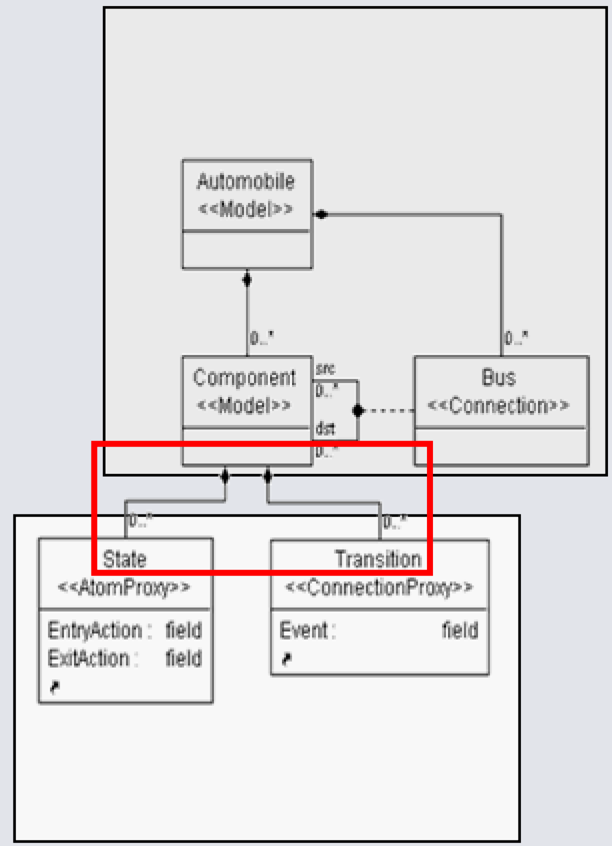
- Goal: Composing modeling languages (not models)
- Metamodel composition methods in the Generic Modeling Environment (GME):
  - Class Merge
  - Metamodel Interfacing
  - Class Refinement
  - Template Instantiation
  - Metamodel Transformations

Metamodel Composition Methods

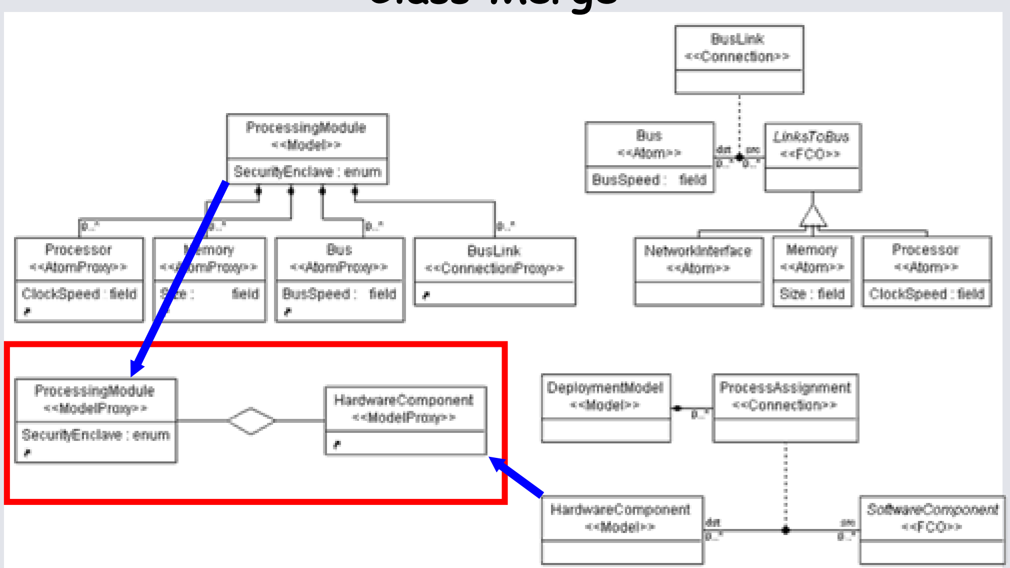
Metamodel Interfacing



Class Refinement



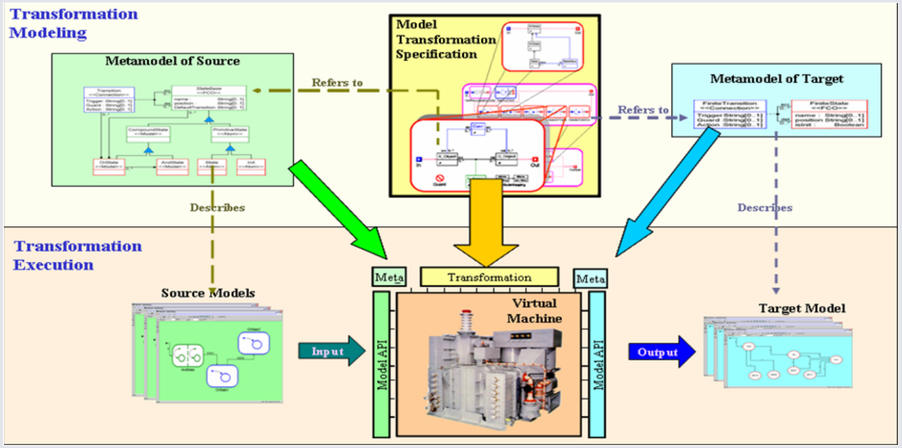
Class Merge



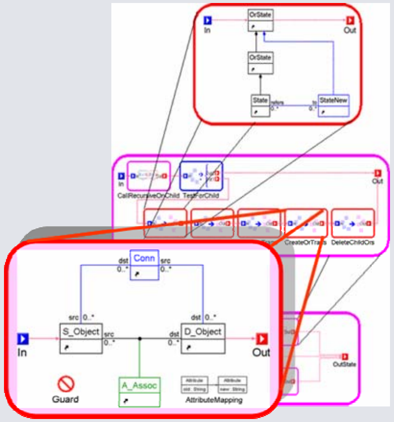
Summary of Progress in Model Transformations

- Complex model transformations can be formally specified in the form of executable graph transformation rules
- G/T semantics is very powerful but the implementation needs to be tailored for efficiency
- GReAT is an open source, metamodel-based model transformation language supported by tools: modeling tool, rewriting engine, code generator and debugger. It is based on attributed/typed graph matching, multi-domain rewriting rules, and explicitly sequenced rewriting operators.
- Highlights of GReAT extensions: shared spaces, sorting of match results, cross-products of matches, higher-order operators (groups)
- Applications of GReAT:
  - Simulink/Stateflow verifying code generator
  - Several model transformation tools in embedded system toolchains
  - Semantic anchoring of domain-specific modeling languages

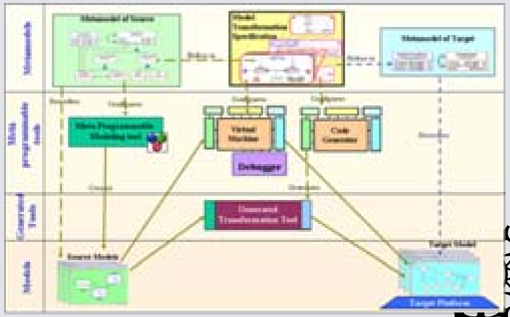
Concept: Metamodel-based Transformations



Language: Graph Transforms



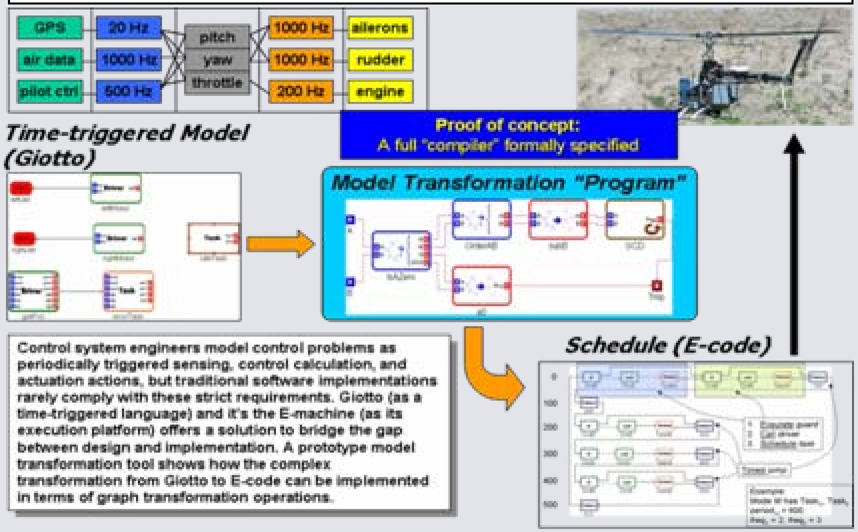
Toolsuite: GReAT



(Karsai et al, 2005-2006)

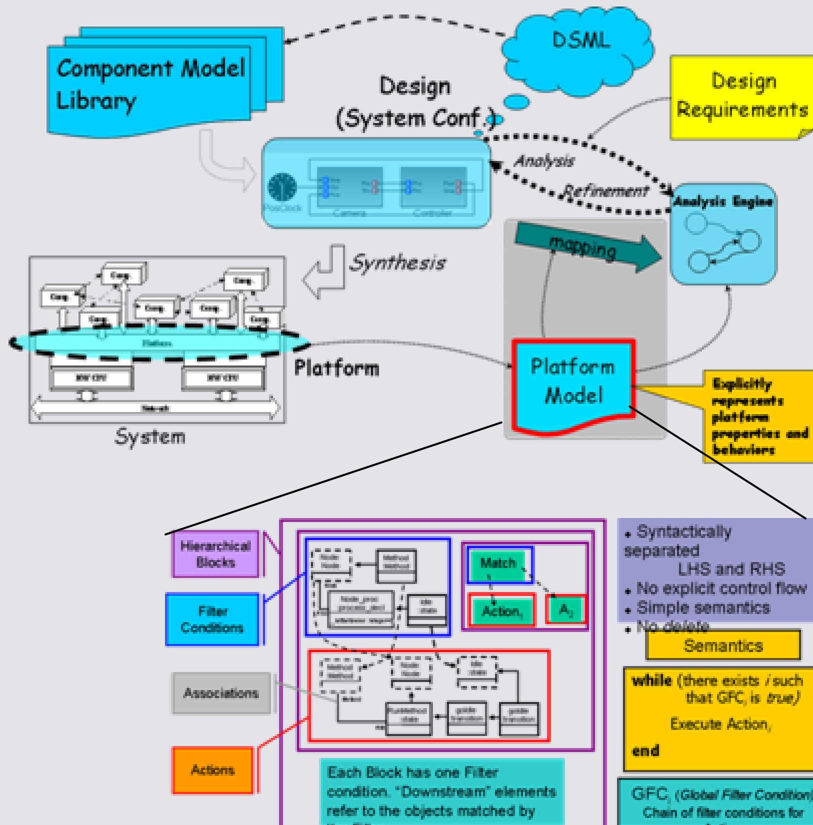
Major Applications of Model Transformations

Model Transformations for Schedule Generation:

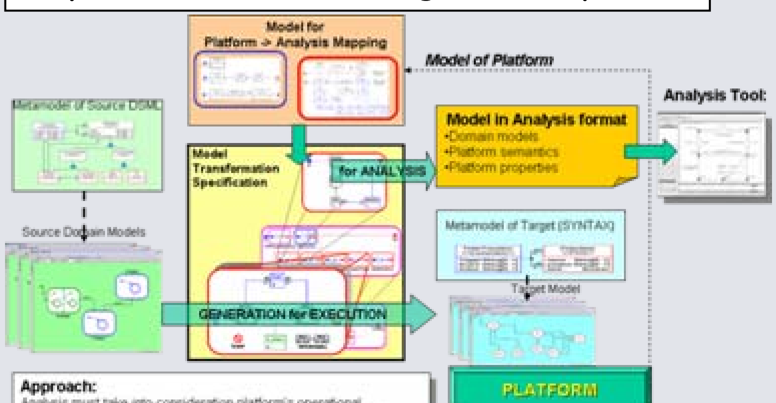


Control system engineers model control problems as periodically triggered sensing, control calculation, and actuation actions, but traditional software implementations rarely comply with these strict requirements. Giotto (as a time-triggered language) and it's the E-machine (as its execution platform) offers a solution to bridge the gap between design and implementation. A prototype model transformation tool shows how the complex transformation from Giotto to E-code can be implemented in terms of graph transformation operations.

Explicit Platform Modeling Language:



Implicit Platform Modeling for Analysis:

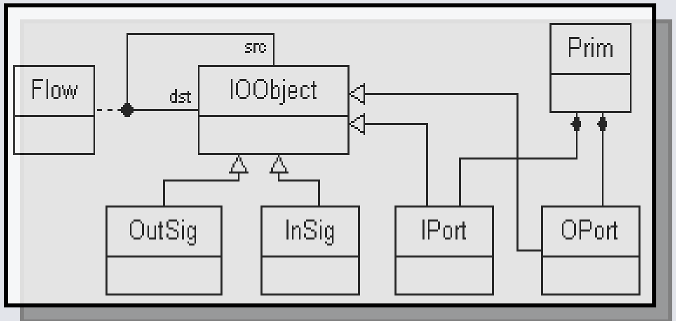


Approach:
Analysis must take into consideration platform's operational semantics, i.e. model of computation. To make this possible, an explicit platform model is required that proscribes how the platform's concepts are mapped into the concepts of the analysis language. This mapping defines the model transformation needed.

(Karsai et al, 2005-2006)

Structural Semantics of Models and Metamodels

We followed a **formal logic approach** to structural semantics. A metamodel is mapped to a set of n-ary function symbols and constraints over an associated Herbrand Universe.



Primitives of DSP Domain

$$\Upsilon = \begin{cases} insig(X) : \texttt{X is an input signal} \\ outsig(X) : \texttt{X is an output signal} \\ prim(X) : \texttt{X is a basic DSP operation} \\ iport(X,Y) : \texttt{X has an input port Y} \\ oport(X,Y) : \texttt{X has an output port Y} \\ inst(X,Y) : \texttt{X is the DSP operation Y} \\ flow(X_1,Y_1,X_2,Y_2) : \texttt{Data goes from oport } Y_1 \\ \texttt{on } X_1 \texttt{to iport } Y_2 \texttt{ on } X_2 \end{cases}$$

Some Constraints with NAF

1. Instances must use primitives that are defined: $inst(x, prim(y)) \wedge !prim(y) \Rightarrow malform(x)$,
2. Ports are placed on defined primitives: $iport(prim(x), y) \wedge !prim(x) \Rightarrow malform(y)$,
3. Dataflow connections must start on defined ports: $flow(x, oport(prim(y), z), u, w) \wedge !oport(prim(y), z) \Rightarrow malform(z)$.

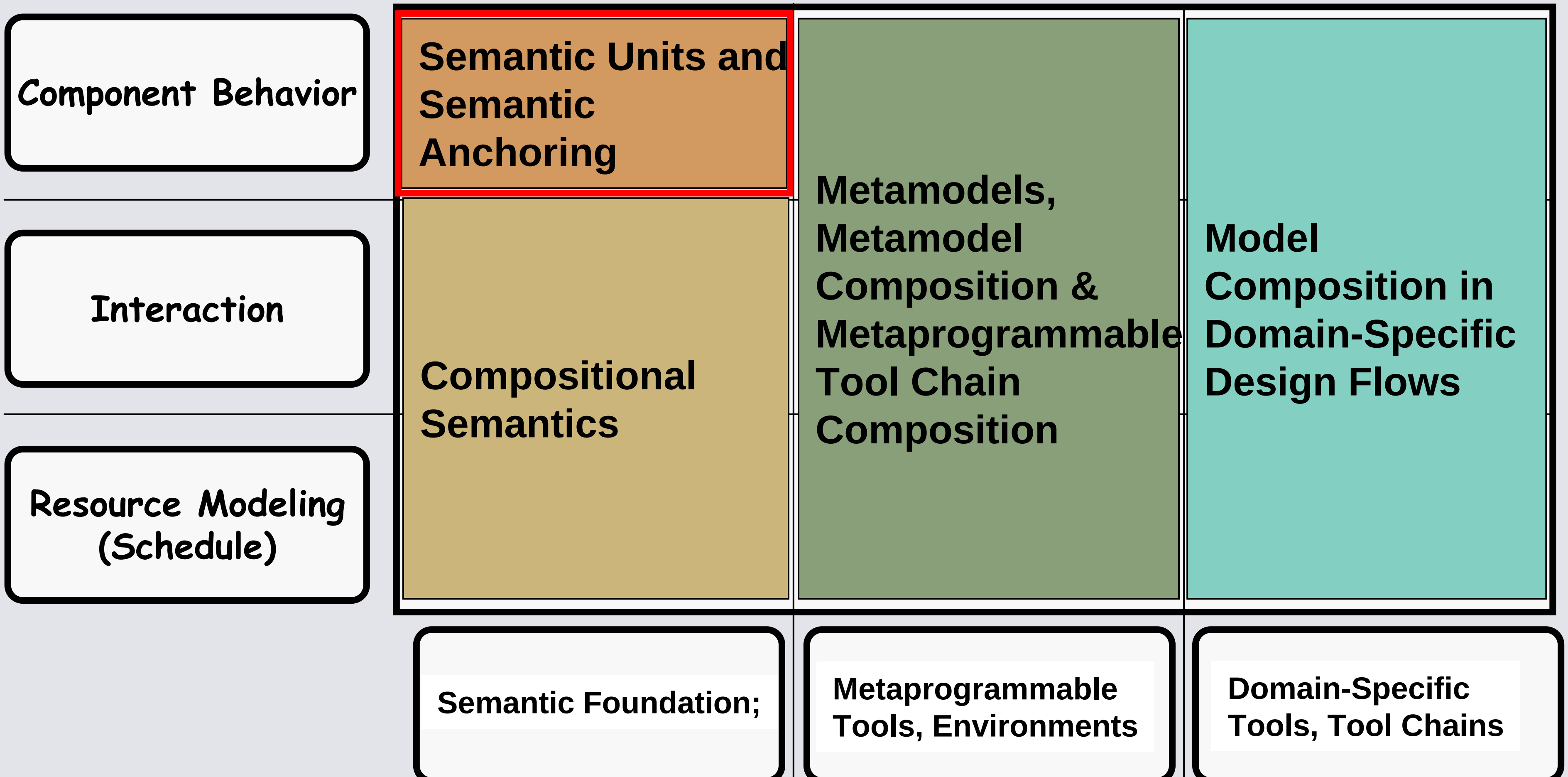
These are the function symbols and some constraints for the example metamodel

We use an **inference procedure** to prove well-formedness or mal-formedness. This inference mechanism is well-defined an tool independent.

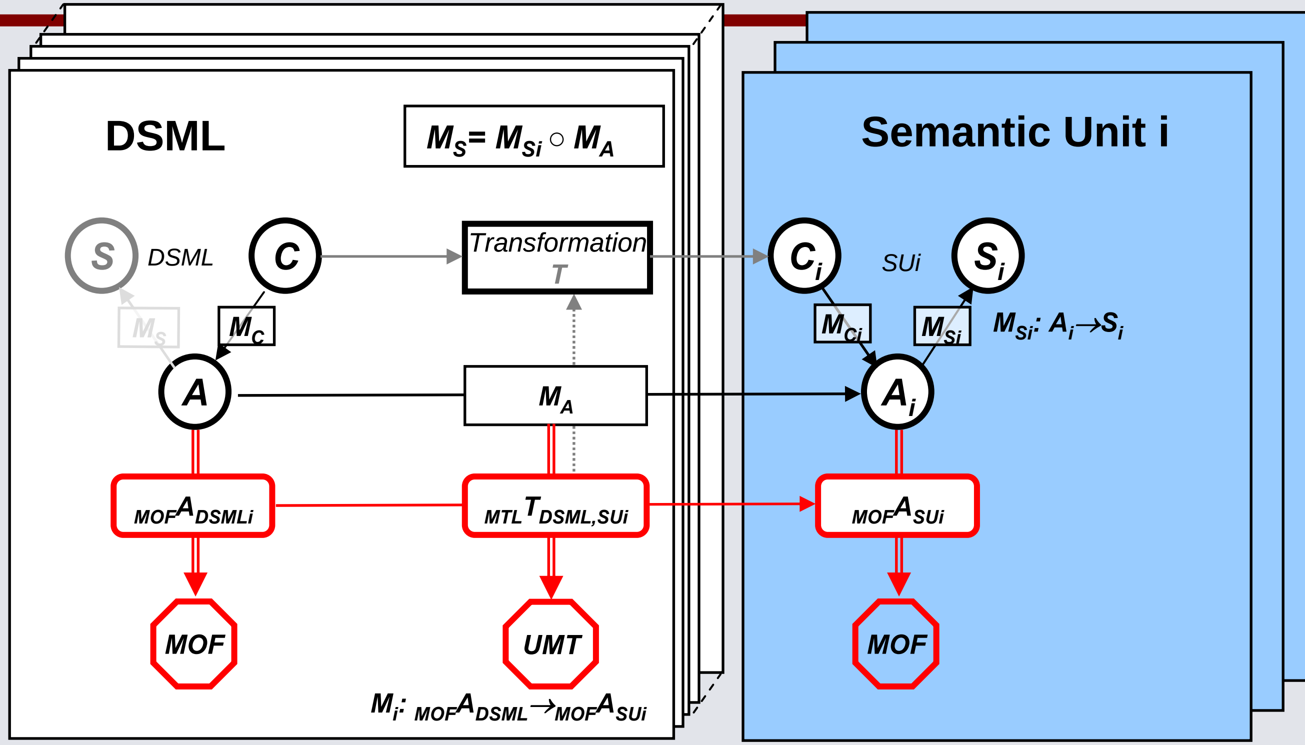
We have constructed an **automatic theorem prover** that answers questions about structural semantics (see poster).

(Jackson, Sztipanovits 2006)

Intersection of System and Tool Composition Dimensions



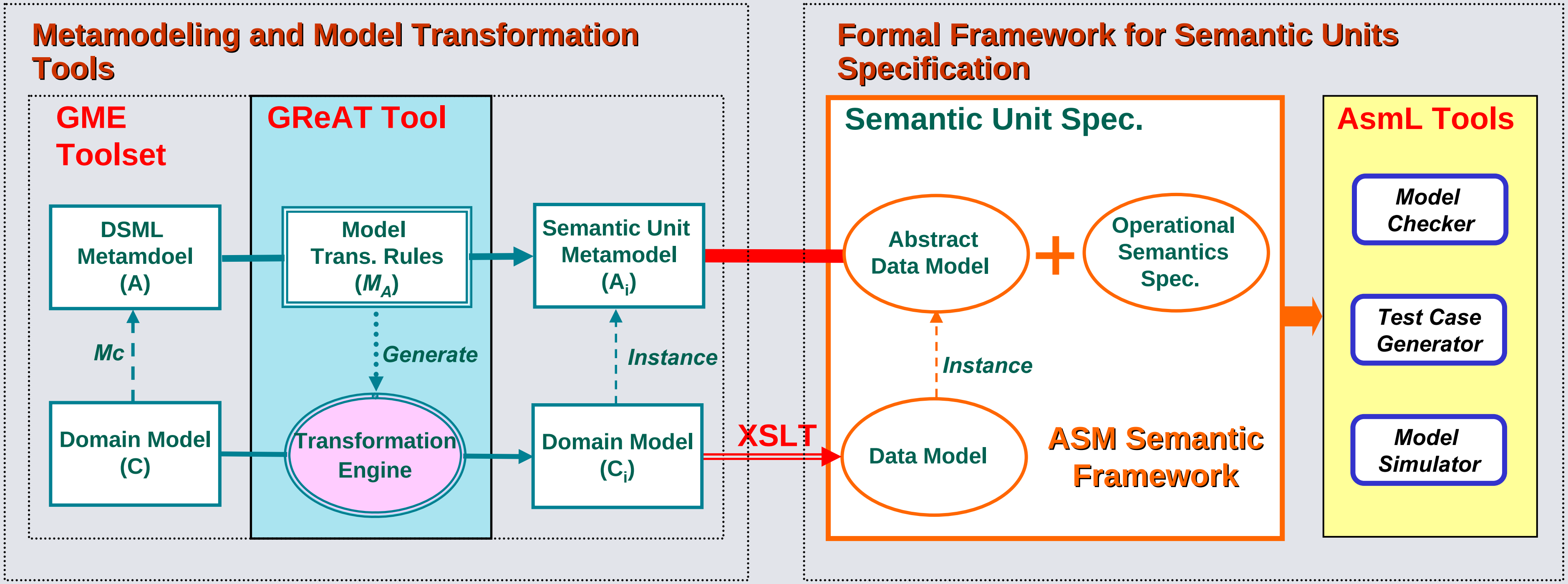
Semantic Anchoring of DSML-s



- Step 1
  - Specify the DSML <*A, C, M$_C$*> by using MOF-based metamodels.
- Step 2
  - Select appropriate semantic units *L* = < *A$_i$, C$_i$*, M_{Ci}, S_i, M_{Si}> for the behavioral aspects of the DSML.
- Step 3
  - Specify the semantic anchoring M_A = *A* -> *A*$_i$ by using UMT.

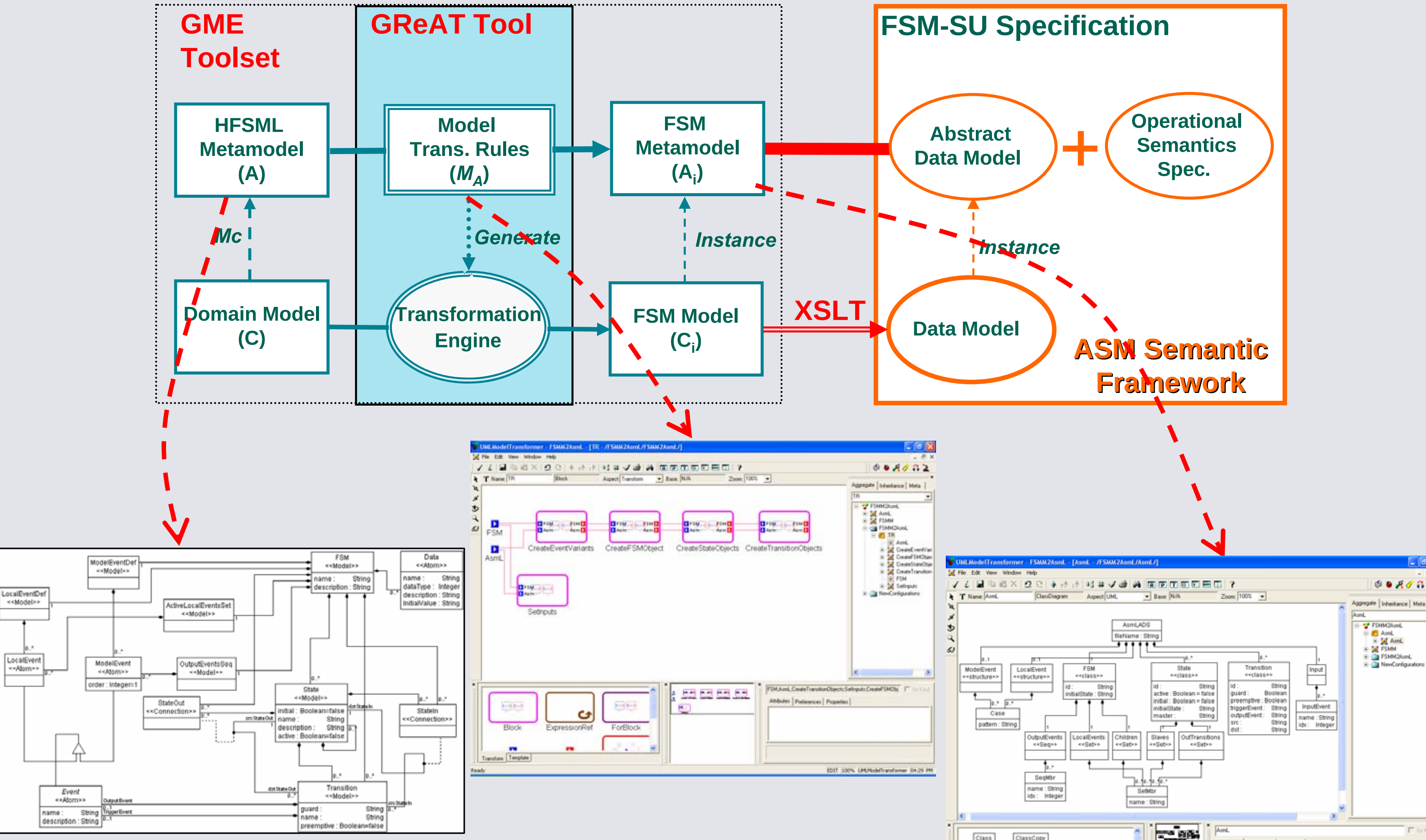
(Chen and Sztipanovits, 2005-2006)

Experimental Tool Suite for Semantic Anchoring

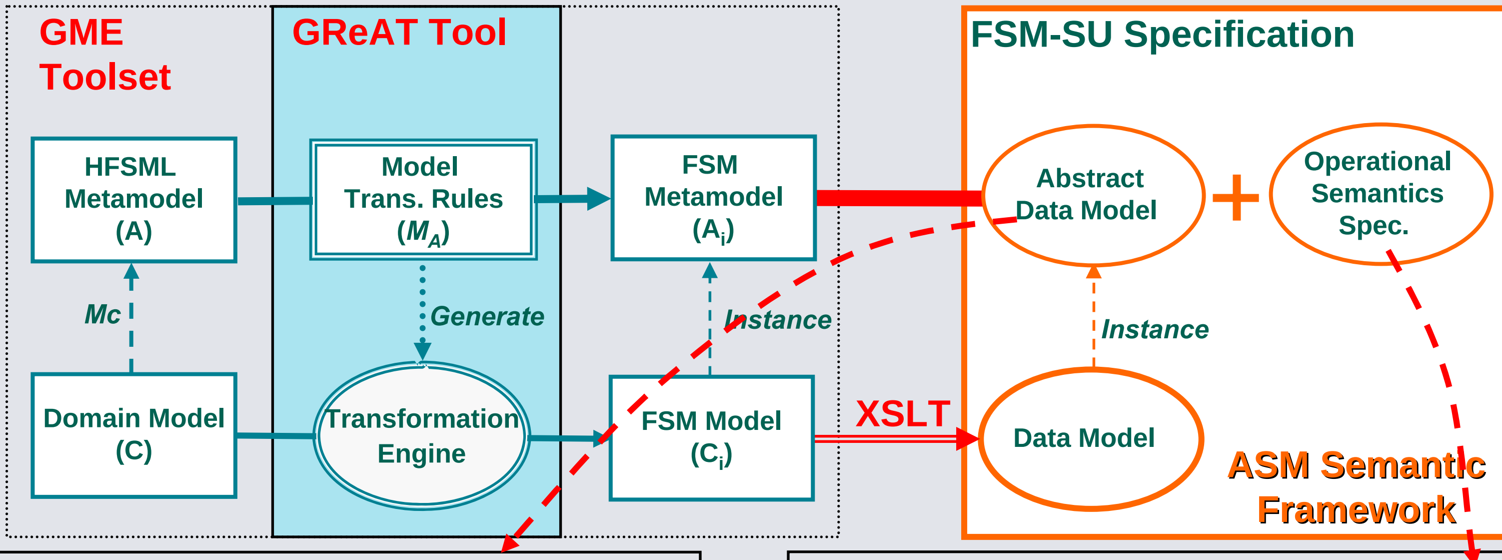


- Metamodeling and Model Transformation Tools
  - GME: Provide a MOF-based metamodeling and modeling environment.
  - GReAT: Build on GME for metamodel to metamodel transformation.
- Tools for Semantic Unit Specification
  - ASM: A particular kind of mathematical machine, like the Turing machine. (Yuri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

Example: HFSML -> FSM-SU; 1/3



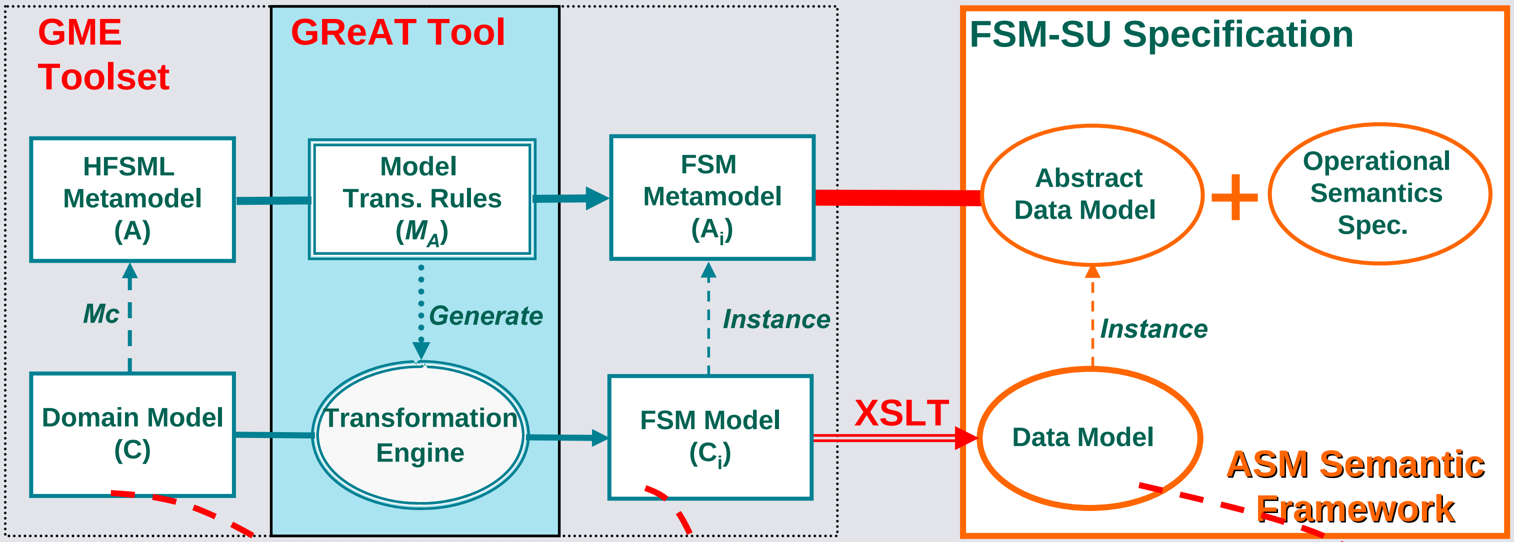
Example: HFSML -> FSM-SU; 2/3

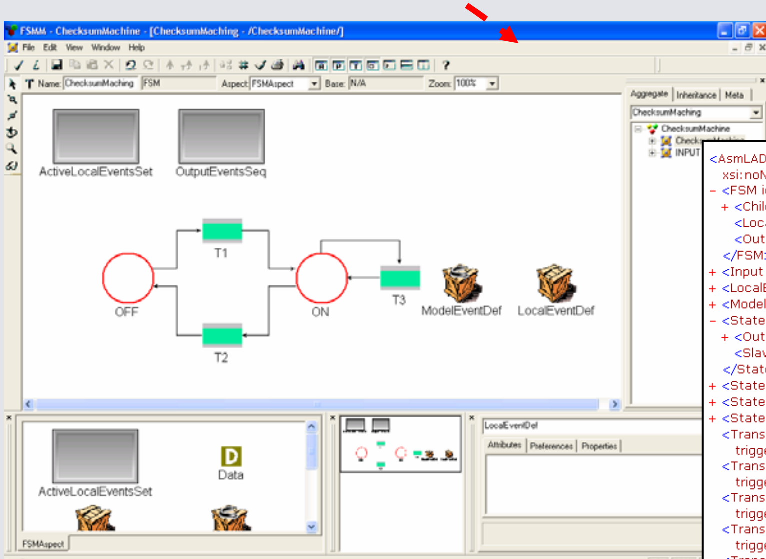



```
structure Event
  eventType as String

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; 3/3






```
<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)
```

Intersection of System and Tool Composition Dimensions

Component Behavior

Interaction

Resource Modeling (Schedule)

Semantic Units and Semantic Anchoring

Compositional Semantics

Metamodels, Metamodel Composition & Metaprogrammable Tool Chain Composition

Model Composition in Domain-Specific Design Flows

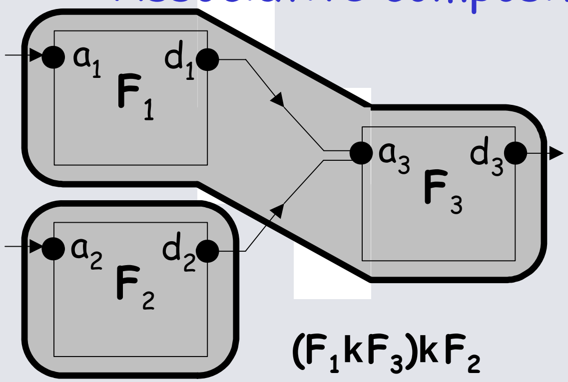
Semantic Foundation;

Metaprogrammable Tools, Environments

Domain-Specific Tools, Tool Chains

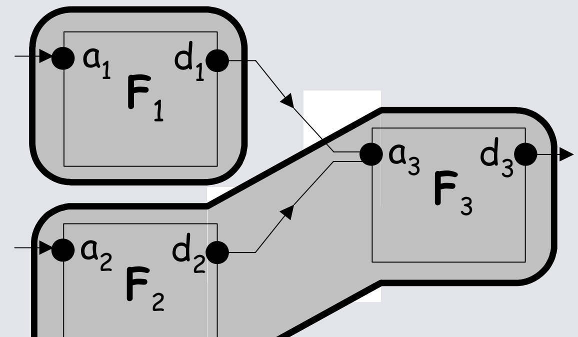
Component-based Analysis

- Incremental design
  - Associative composition



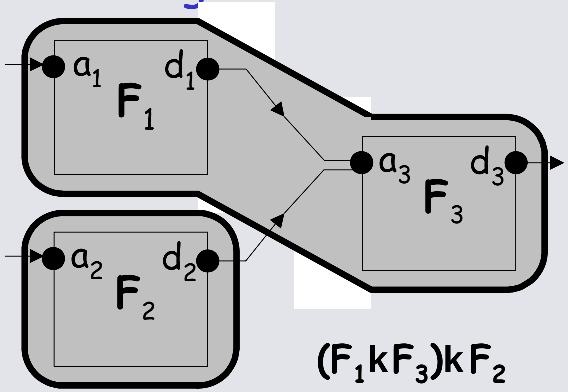
$(F_1kF_3)kF_2$

$\overset{?}{=}$



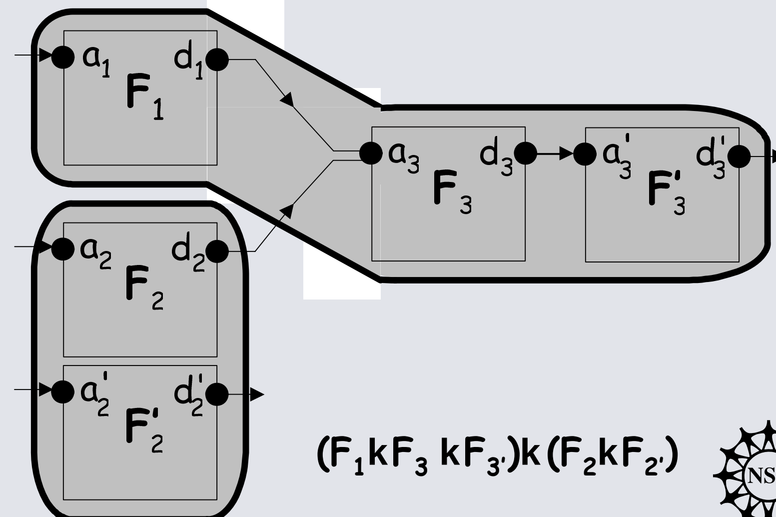
$F_1k(F_2kF_3)$

- Independent implementability
  - No global checks



$(F_1kF_3)kF_2$

o



$(F_1kF_3\ kF_{3'})k(F_2kF_{2'})$

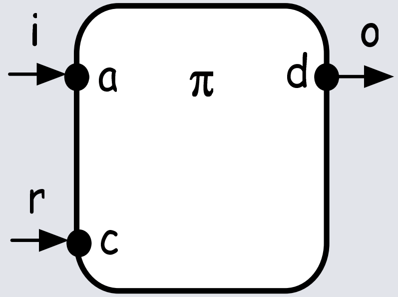
(Matic and Henzinger, 2006)

Real-time Interface

Assumption

requests bounded by a

capacity larger than c

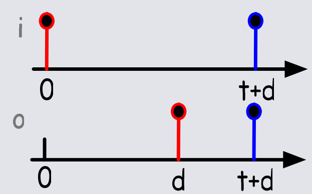


Guarantee

output latency bounded by d

Output rate function i^d

$i^d(t) = i(t+d)$



Interface predicate

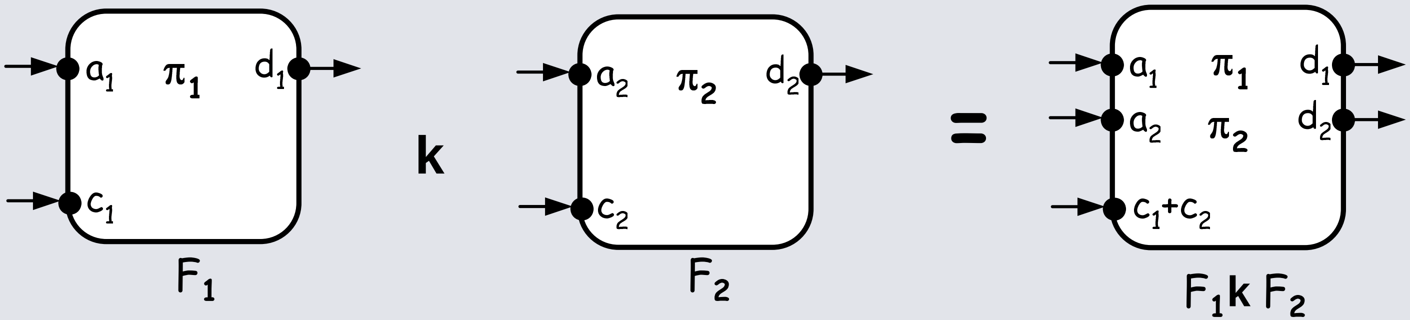
input $\phi^I = (r \geq c \ \wedge \ i \leq a)$

output $\phi^O = o \leq i^d$

(Matic and Henzinger, 2006)

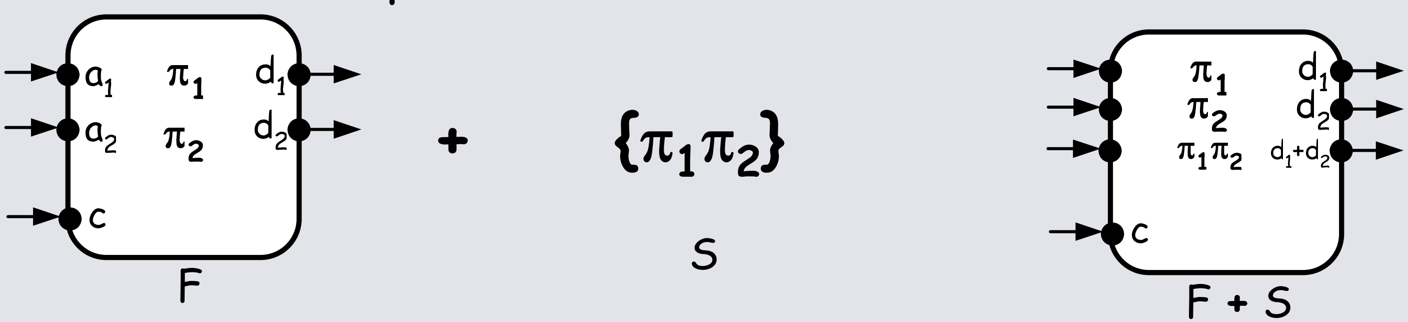
Interface Algebra

- Composition operation ||



$c_1+c_2 \cdot 1$

- Connection operation +



$i_1+i_{12} \cdot a_1$

$i_2+i^{d1}_{12} \cdot a_2$

- Refinement relation ·

  F′ refines F if

  - S_F μ $S_{F'}$
  - for each port valuation of F there exists a valuation of F′ :

$$\phi^I_F \Rightarrow \phi^I_{F'} \qquad \phi^O_{F'} \Rightarrow \phi^O_F$$

(Matic and Henzinger, 2006)

Algebra Properties

- Incremental design
  - (FkG)kH is defined

) Fk(GkH) is def. Æ (FkG)kH = Fk(GkH)

  - (FkG)©S is defined

) (F©S)kG is def. Æ (FkG)©S = (F©S)kG

- Independent refinement
  - FkG is defined Æ F’ ¹ F

) F’k G is def. Æ F’k G ¹ FkG

  - F©S is defined Æ F’ ¹ F

) F’© S is def. Æ F’© S ¹ F©S

for all j=1,…,n: F'_j ¹ F_j

$E(F'_1,\ldots,F'_n)$ ¹ $E(F_1,\ldots,F_n)$

(Matic and Henzinger, 2006)