2006 Bellairs CAMPaM Workshop
25 April 2006

Old and New Bits and Pieces of a CAMPaM Framework

Hans Vangheluwe

McGill

Modelling, Simulation and Design Lab (MSDL)
School of Computer Science, McGill University, Montréal, Canada

Overview

1. Some examples of (Multi-Formalism) Modelling
2. Domain-Specific (Visual) Modelling – DS(V)M
   - What/Why of DS(V)M (and DS(V)Ls) ?
3. Building DS(V)M Tools Effectively
   (a) Specifying **syntax** of DS(V)Ls:
      - **abstract** (**meta-modelling**)
      - **concrete** (visual)

   (b) Modelling Reactive Visual Modelling Environments
      - multi-formalism
      - nesting/scoping of behaviour
      - glue reactive behaviour, syntax check and layout

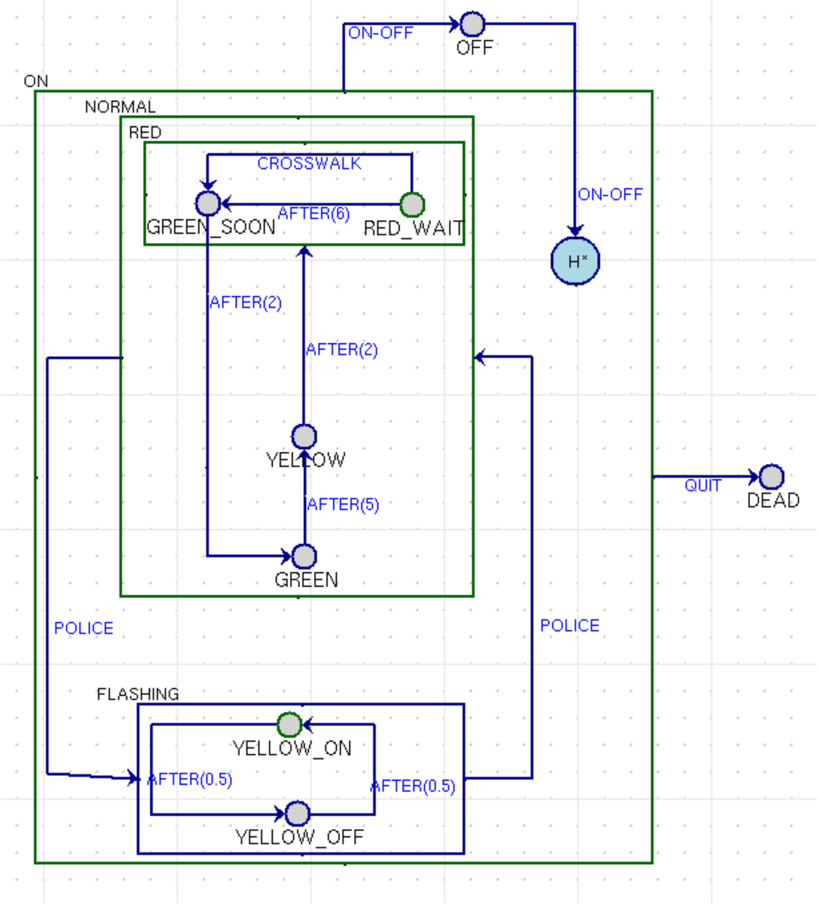
   (c) Specifying DS(V)L **semantics**: **transformations**

(d) Modelling (and executing) **transformations**: **graph rewriting**

4. DSVM examples
   - step-by-step, in a tool
   - Formalism Transformation uses
5. Semper Variabilis: dealing with evolution
6. Conclusions

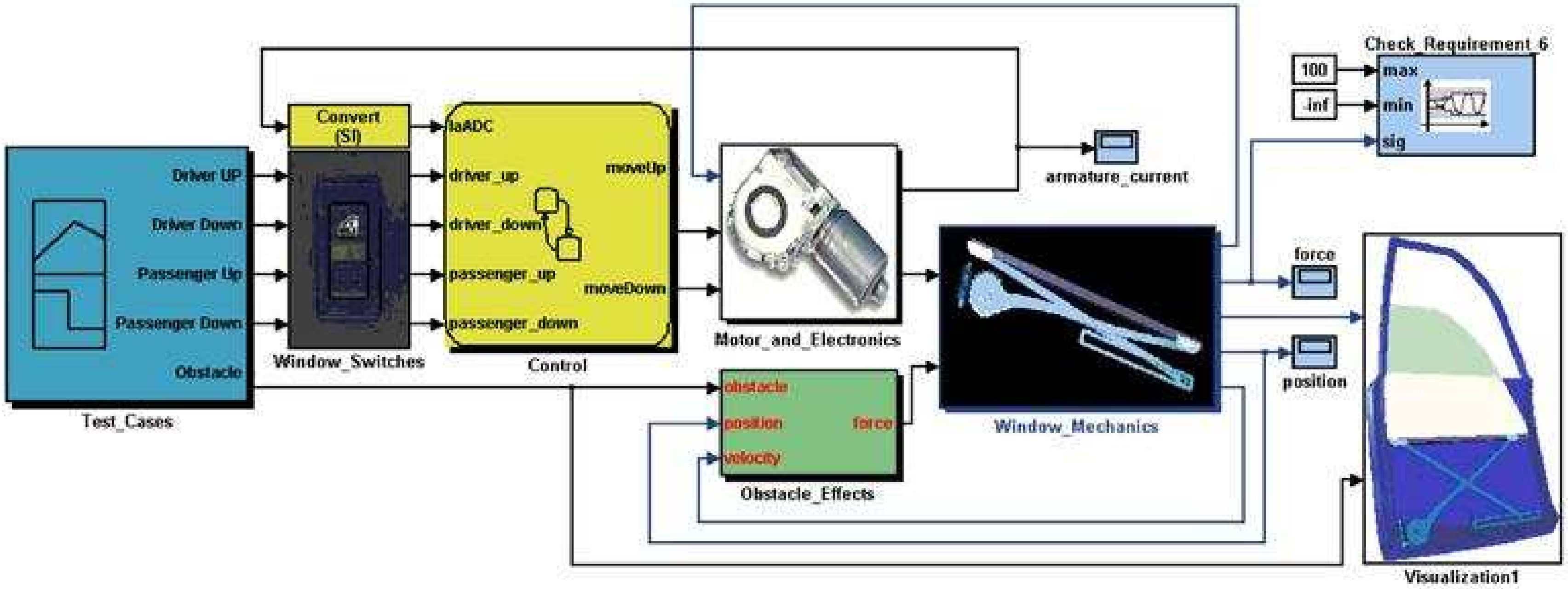
Available Information,

Questions to be Answered, . . .
⇒ Abstraction Level/Formalism



Need Multiple Formalisms: Power Window

The Model

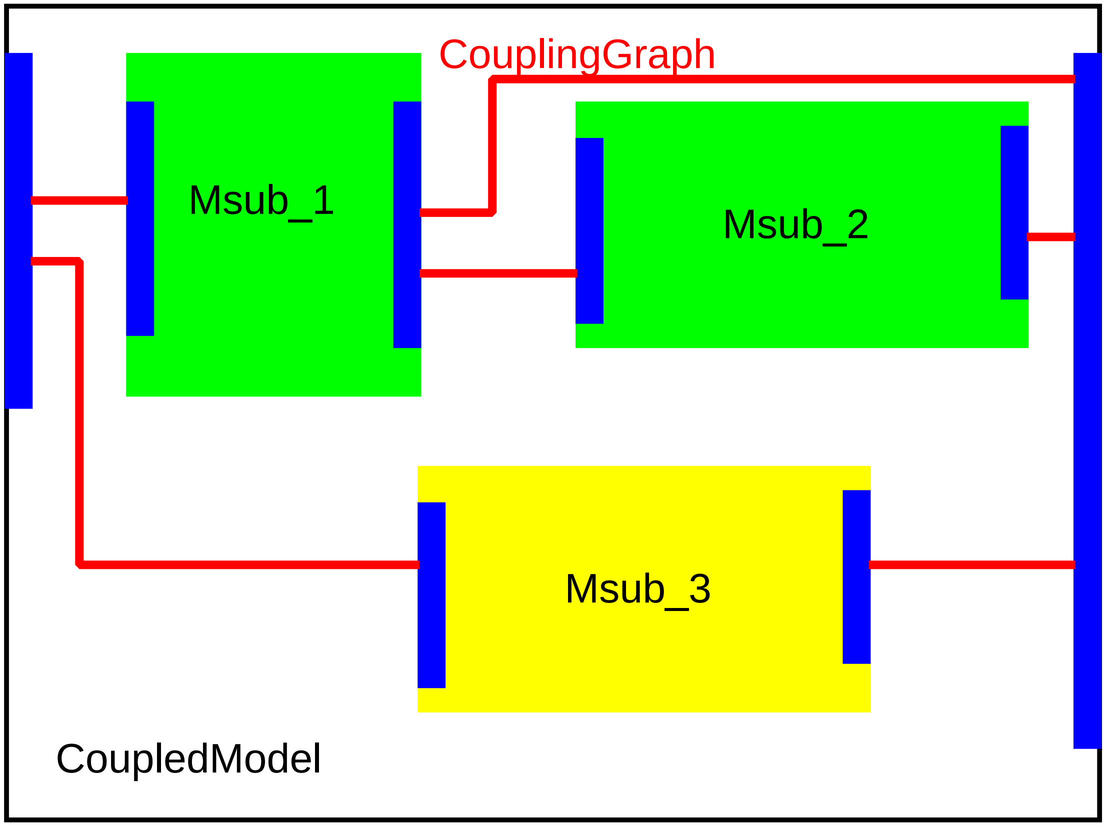


www.mathworks.com/products/demos/simulink/PowerWindow/html/PowerWindow1.html

Semantics of Coupled Models

- Super-formalism subsumes all formalisms
- Co-simulation (coupling resolved at trajectory level)
- Transform to common formalism

Multi-formalism coupled model: co-simulation



Co-simulation of multi-formalism coupled models

- Sub-models simulated with formalism-specific simulators.
- Interaction due to coupling is resolved at trajectory level.

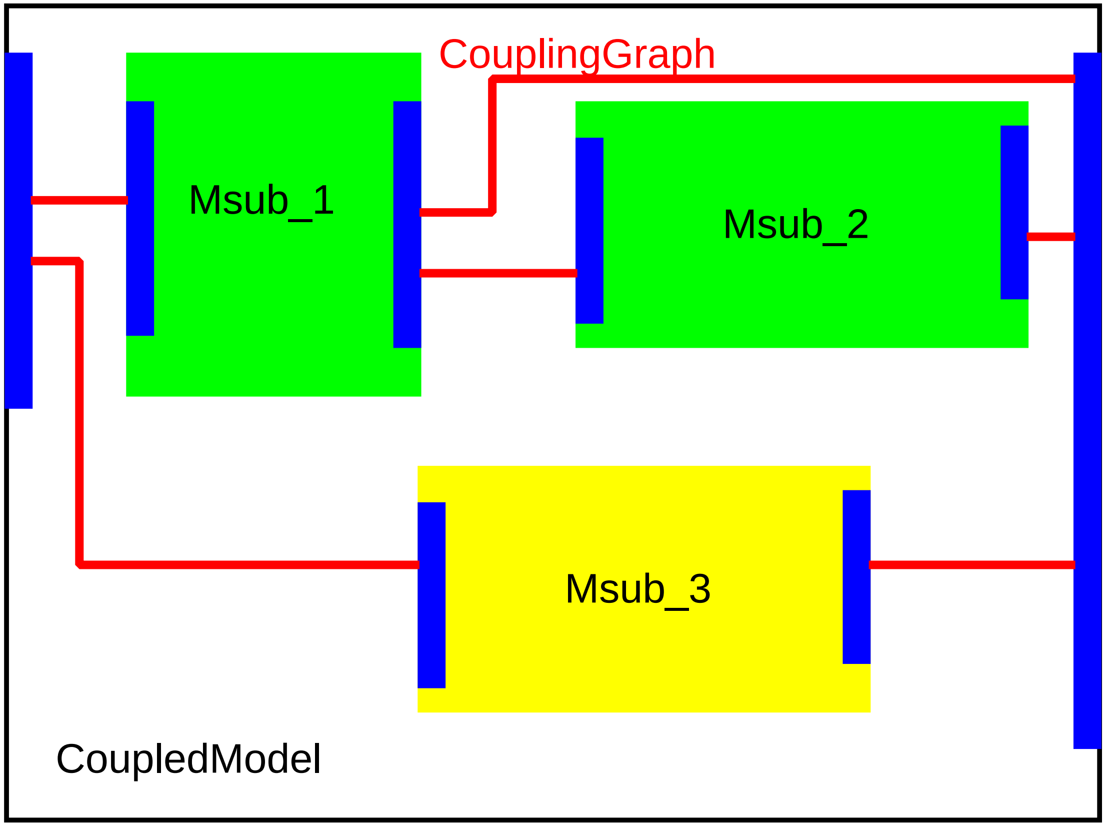
→ Loss of information.

→ Questions can *only* be answered at trajectory level.

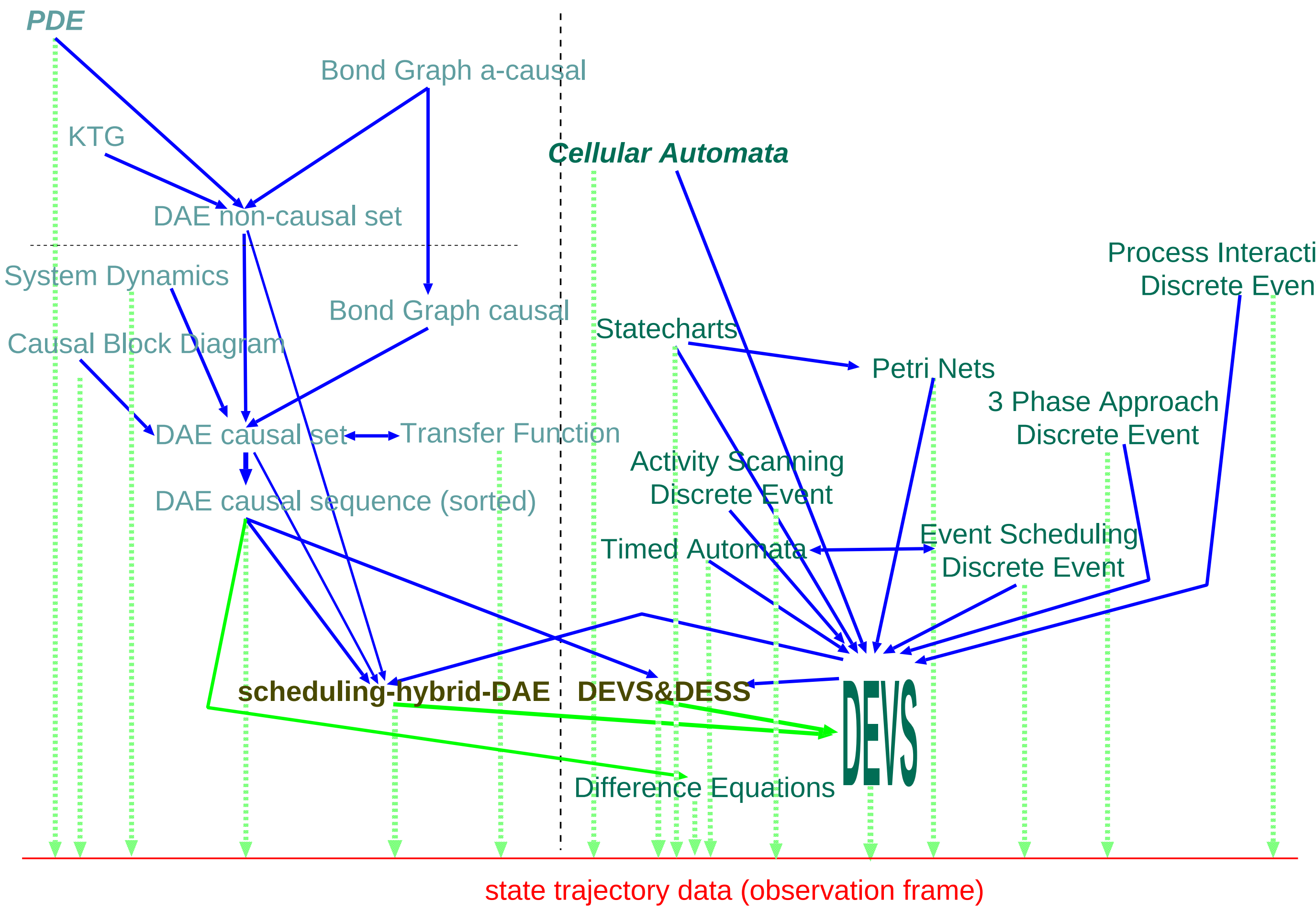
→ Speed and numerical accuracy problems
for continuous formalisms.

→ Meaningful for discrete-event formalisms (beware of legitimacy !).
Basis of the DoD High Level Architecture (HLA)
for simulator interoperability.

Multi-formalism coupled model: multi-formalism modelling



Formalism Transformation Graph



Multi-formalism modelling $\neq$ co-simulation

1. Start from a coupled multi-formalism model. Check consistency of this model (*e.g.*, whether causalites and types of connected ports match).

2. Cluster all formalisms described in the same formalism.

3. For each cluster, implement closure under coupling.

4. Look for the best common formalism in the Formalism Transformation Graph all the remaining different formalisms can be transformed to. Worst case: trajectory level (fallback to co-simulation).

5. Transform all the sub-models to the common formalism.

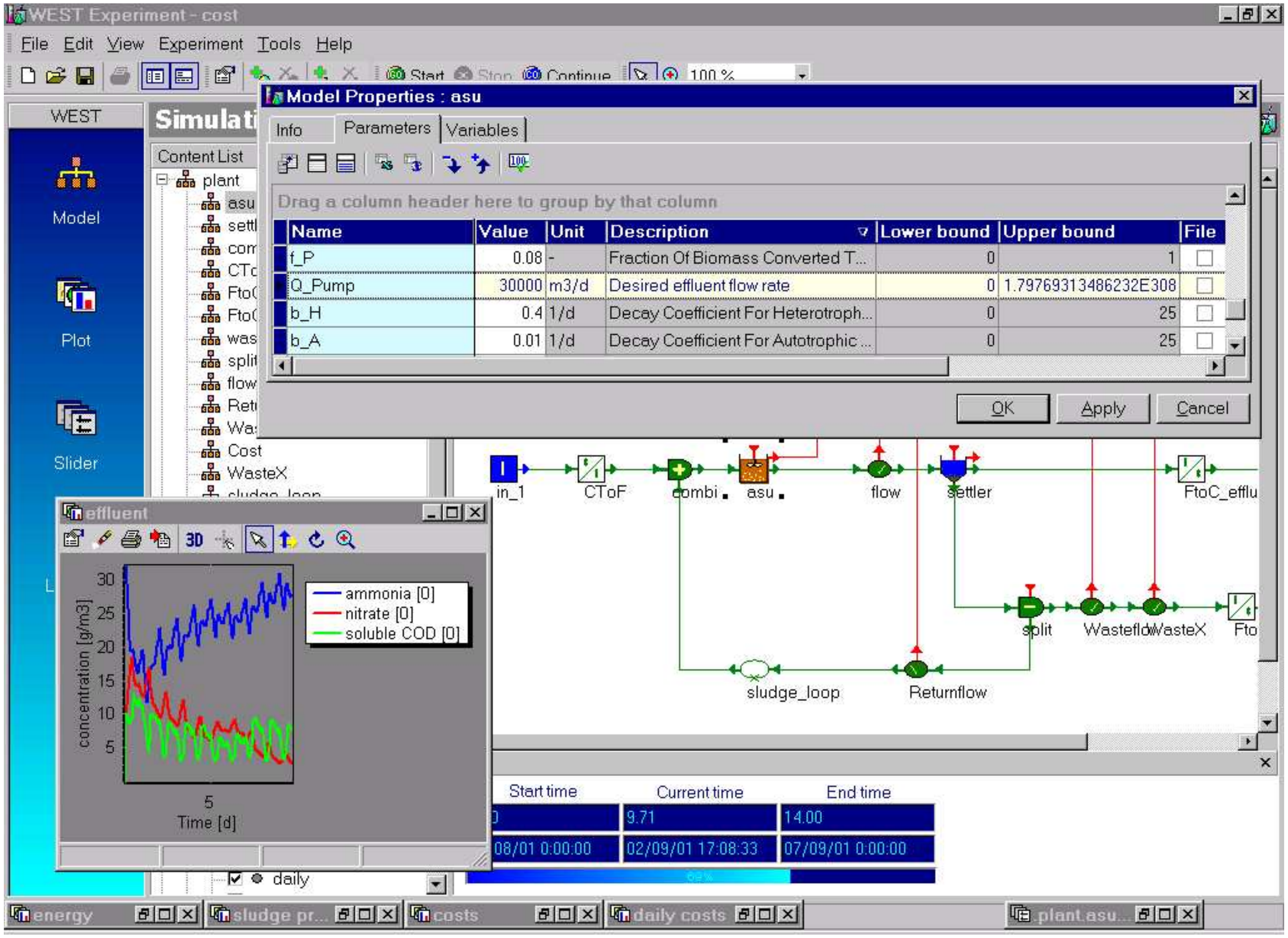
6. Implement closure under coupling of the common formalism.

Domain-Specific Modelling Example

NATO's Sarajevo WWTP

`www.nato.int/sfor/cimic/env-pro/waterpla.htm`

DS(V)M Environment



Model Properties : asu

Info | Parameters | Variables

Drag a column header here to group by that column

Name	Value	Unit	Description	Lower bound	Upper bound	File
f_P	0.08	-	Fraction Of Biomass Converted T...	0	1	□
Q_Pump	30000	m3/d	Desired effluent flow rate	0	1.79769313486232E308	□
b_H	0.4	1/d	Decay Coefficient For Heterotroph...	0	25	□
b_A	0.01	1/d	Decay Coefficient For Autotrophic ...	0	25	□

OK Apply Cancel

Start time	Current time	End time
0	9.71	14.00
08/01 0:00:00	02/09/01 17:08:33	07/09/01 0:00:00

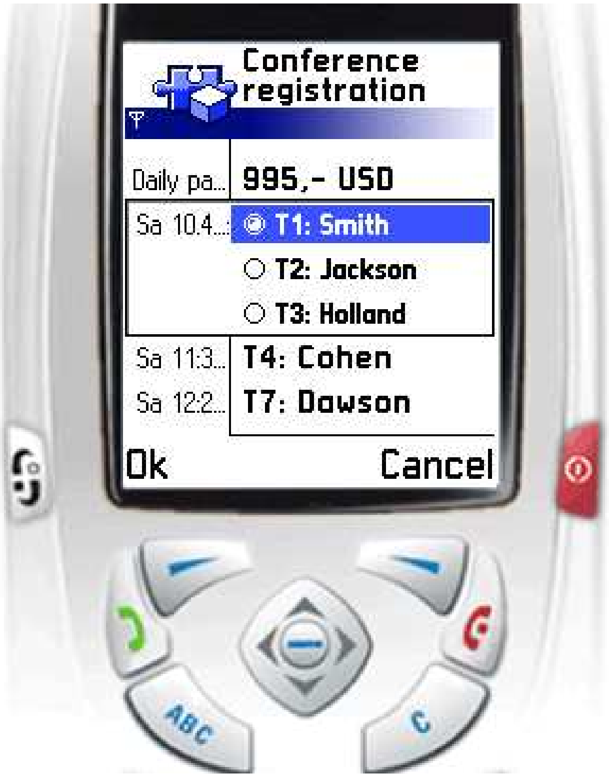
www.hemmis.com/products/west/

Why DS(V)M ?
(as opposed to General Purpose modelling)

- **match the user's mental model** of the problem domain
- **maximally constrain** the user (to the problem at hand)
  ⇒ easier to learn
  ⇒ avoid errors
- **separate** domain-expert's work
  from analysis/transformation expert's work

Anecdotal evidence of 5 to 10 times speedup

DS(V)M Example in Software Domain
smart phones, the application



Conference registration

Daily pa... 995,- USD

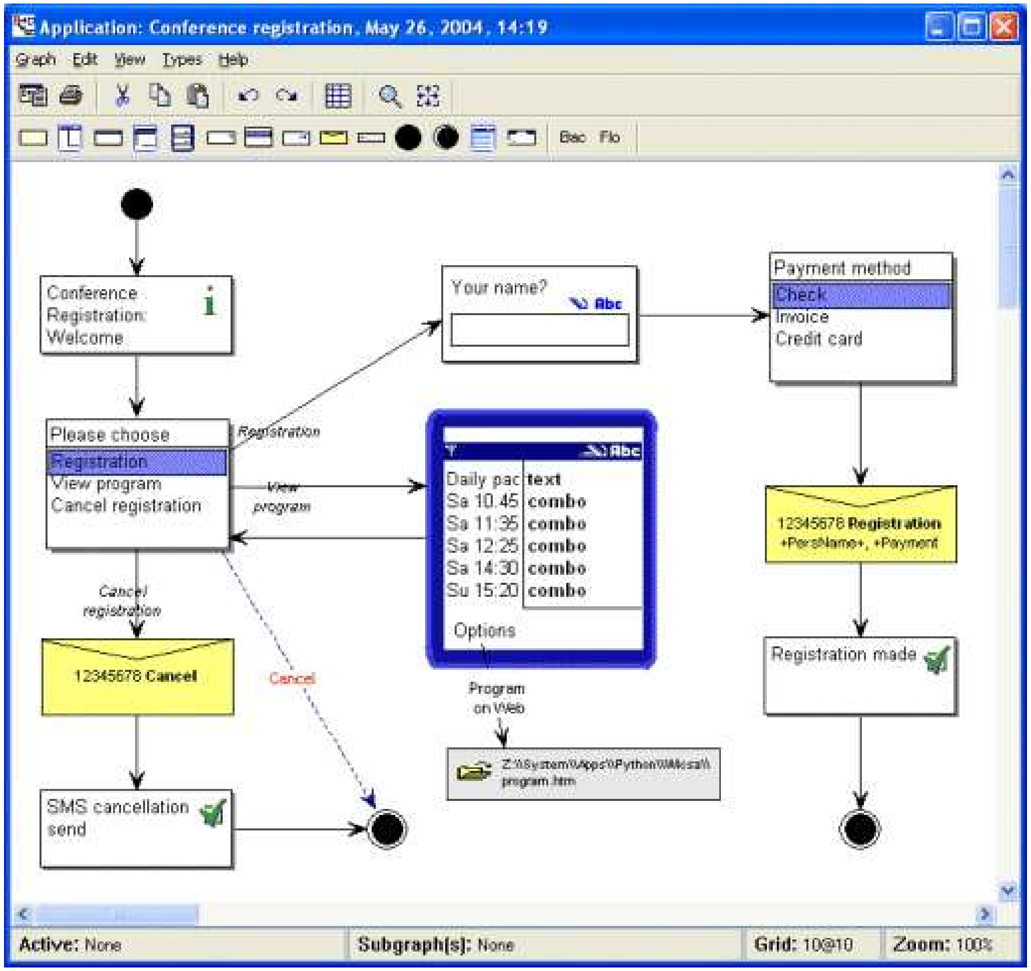
Sa 10.4... ◉ T1: Smith
○ T2: Jackson
○ T3: Holland

Sa 11:3... T4: Cohen

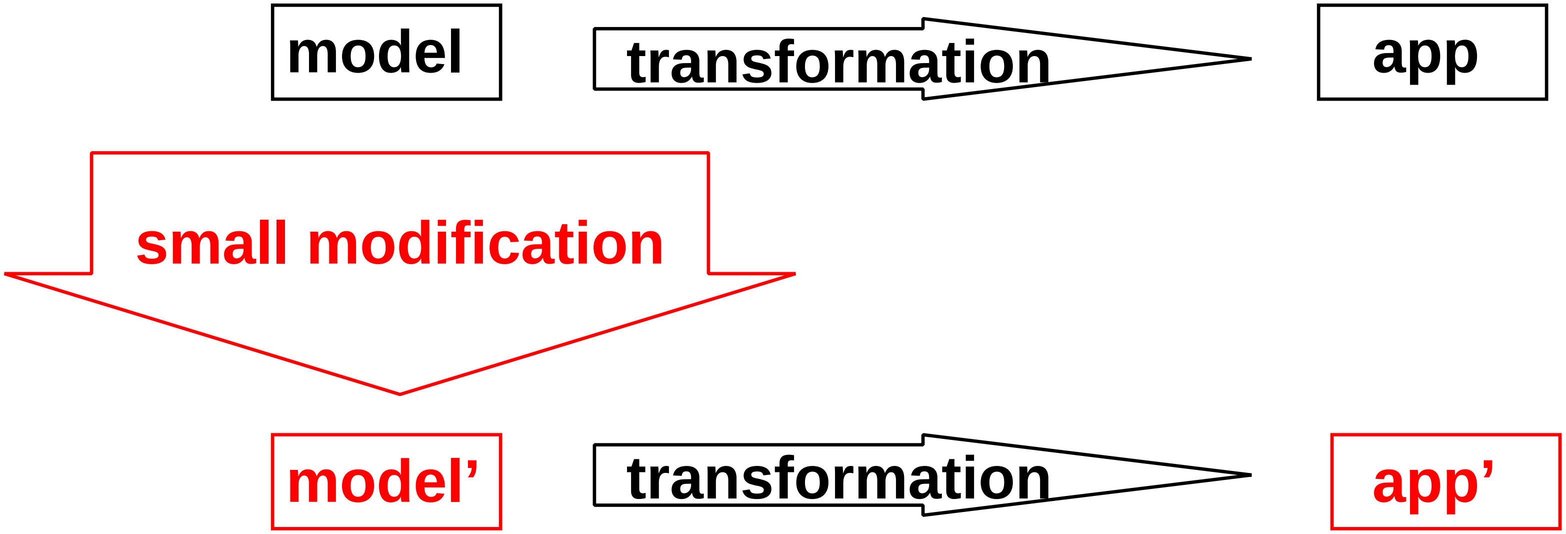
Sa 12:2... T7: Dawson

MetaEdit+ (`www.metacase.com`)

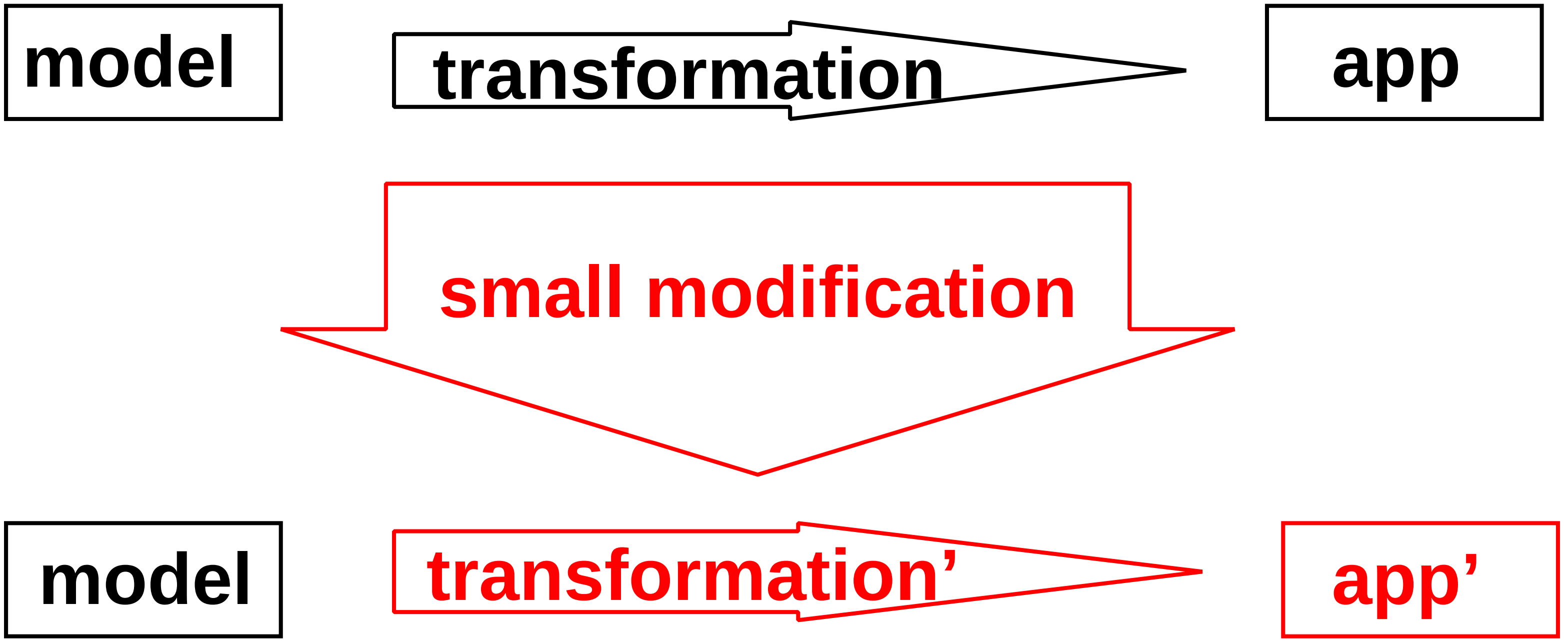
DS(V)M Example: smart phones, the Domain-Specific model



Model-Based Development: Modify the Model



Model-Based Development: Modify the Transformation



Divide and Conquer: Transformation may be multi-step

Usual advantages . . .

Building DS(V)M Tools Effectively . . .

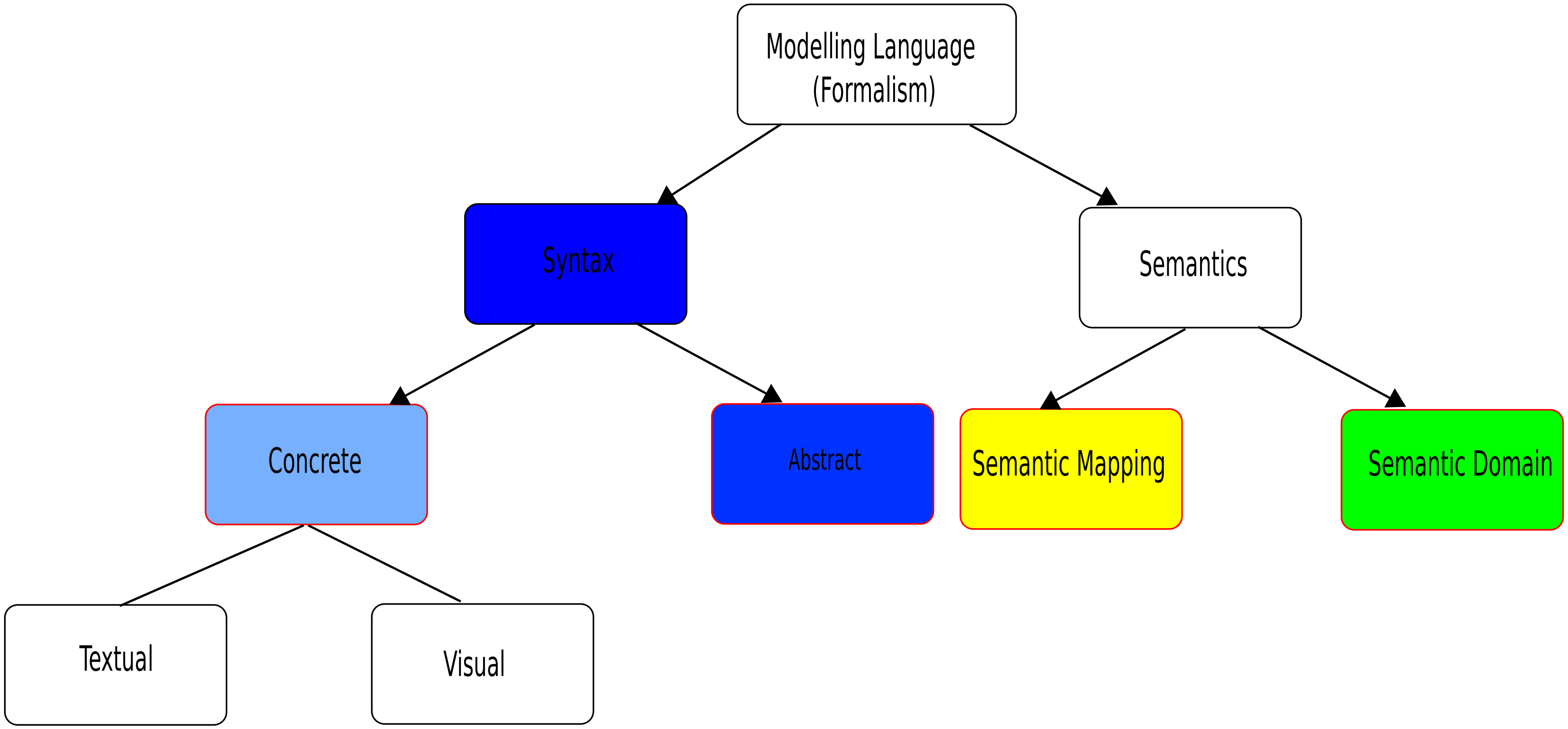
- **development cost** of DS(V)M Tools may be prohibitive !
- we want to effectively (rapidly, correctly, re-usably, . . .)

  1. Specify DS(V)L **syntax**:
     - **abstract** ⇒ **meta-modelling**
     - **concrete** (textual/visual)
  2. Modelling Reactive Visual Modelling Environments
     - multi-formalism
     - nesting/scoping of behaviour
     - glue reactive behaviour, syntax check and layout
  3. Specify DS(V)L **semantics**:
     transformation
  4. Model (and analyze and execute) model **transformations**:
     ⇒ **graph rewriting**

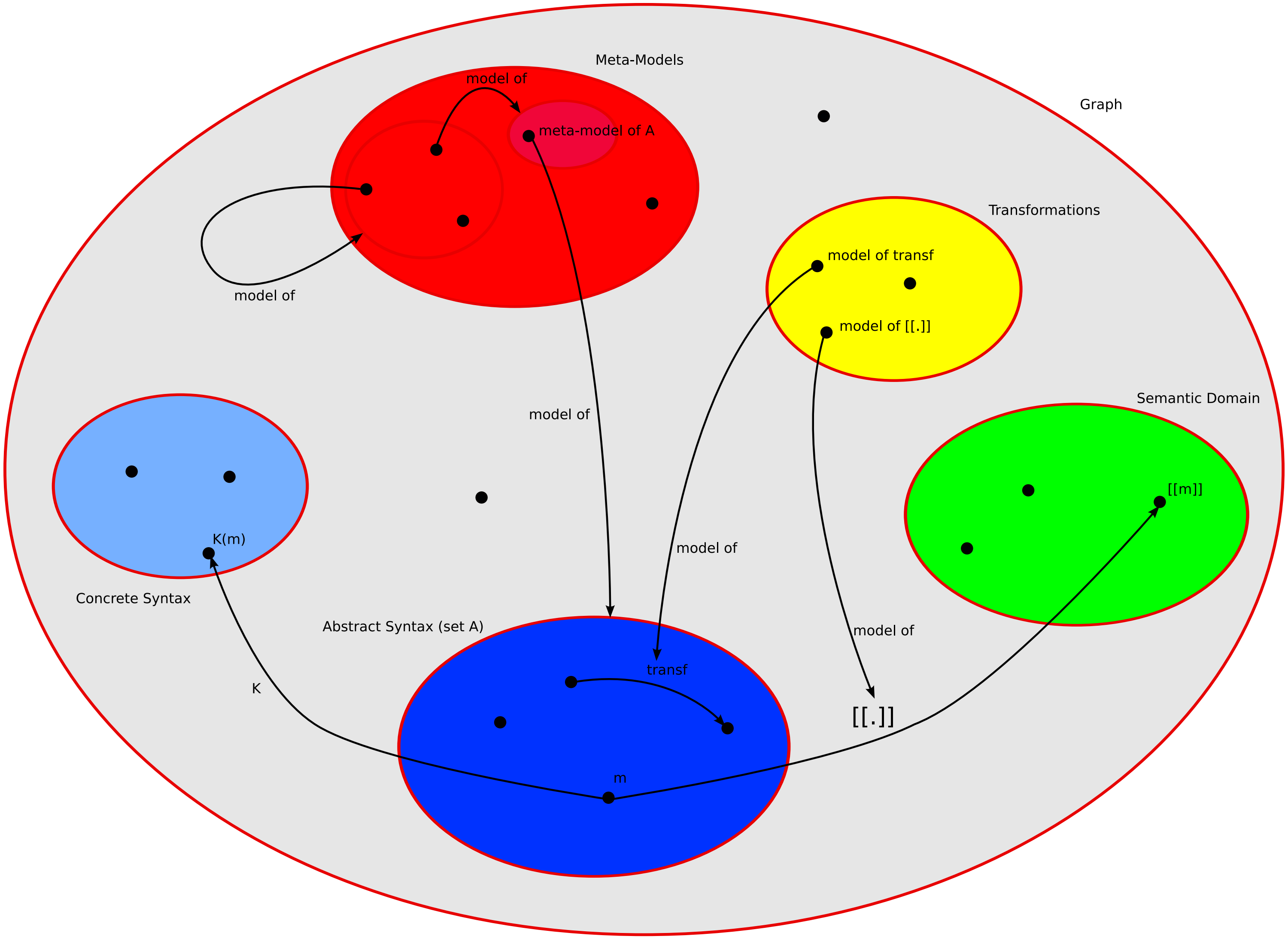
$\Rightarrow$ **model** **everything**

(in the most appropriate formalism,
at the appropriate level of abstraction)

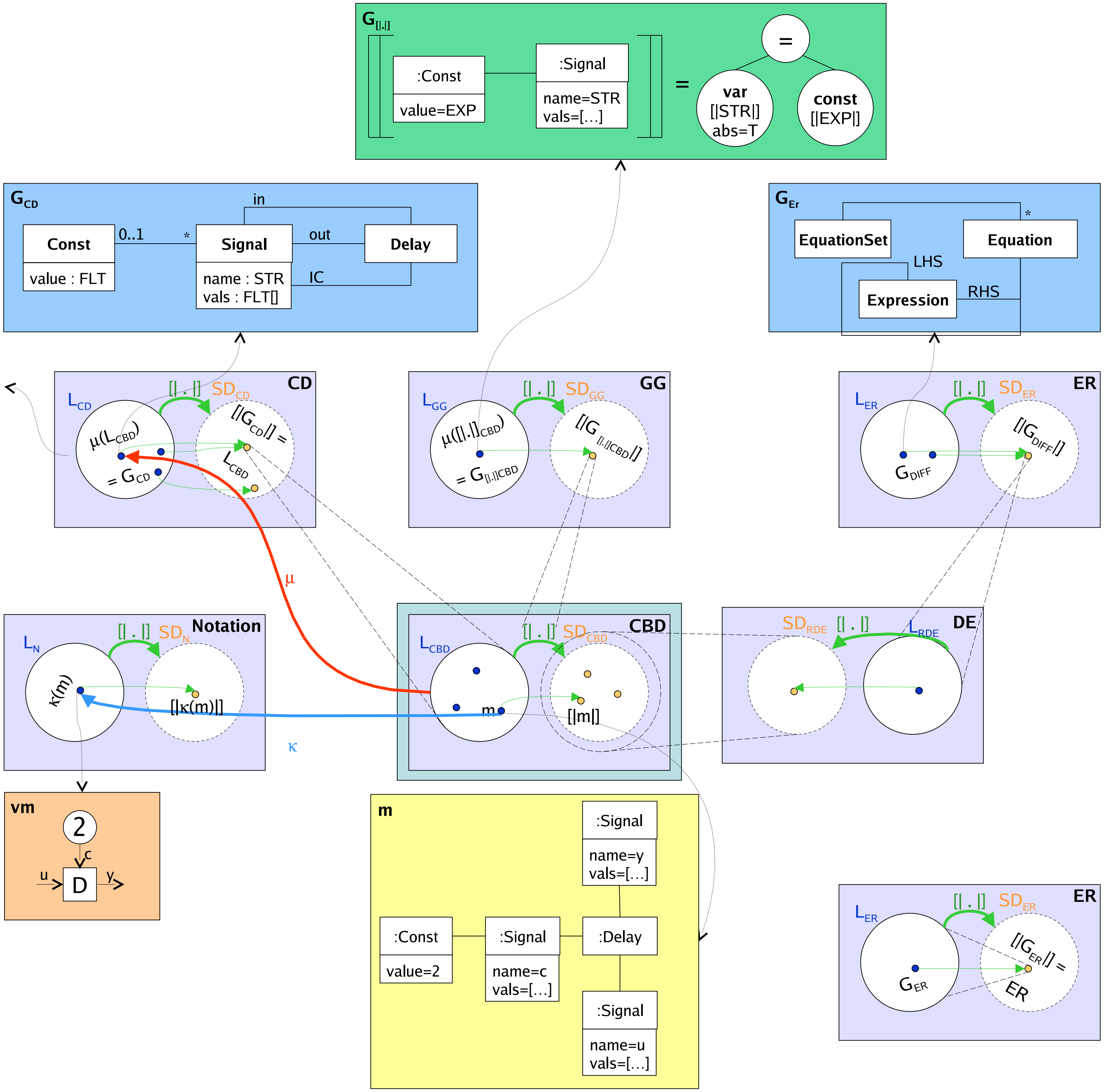
Dissecting a Modelling Language



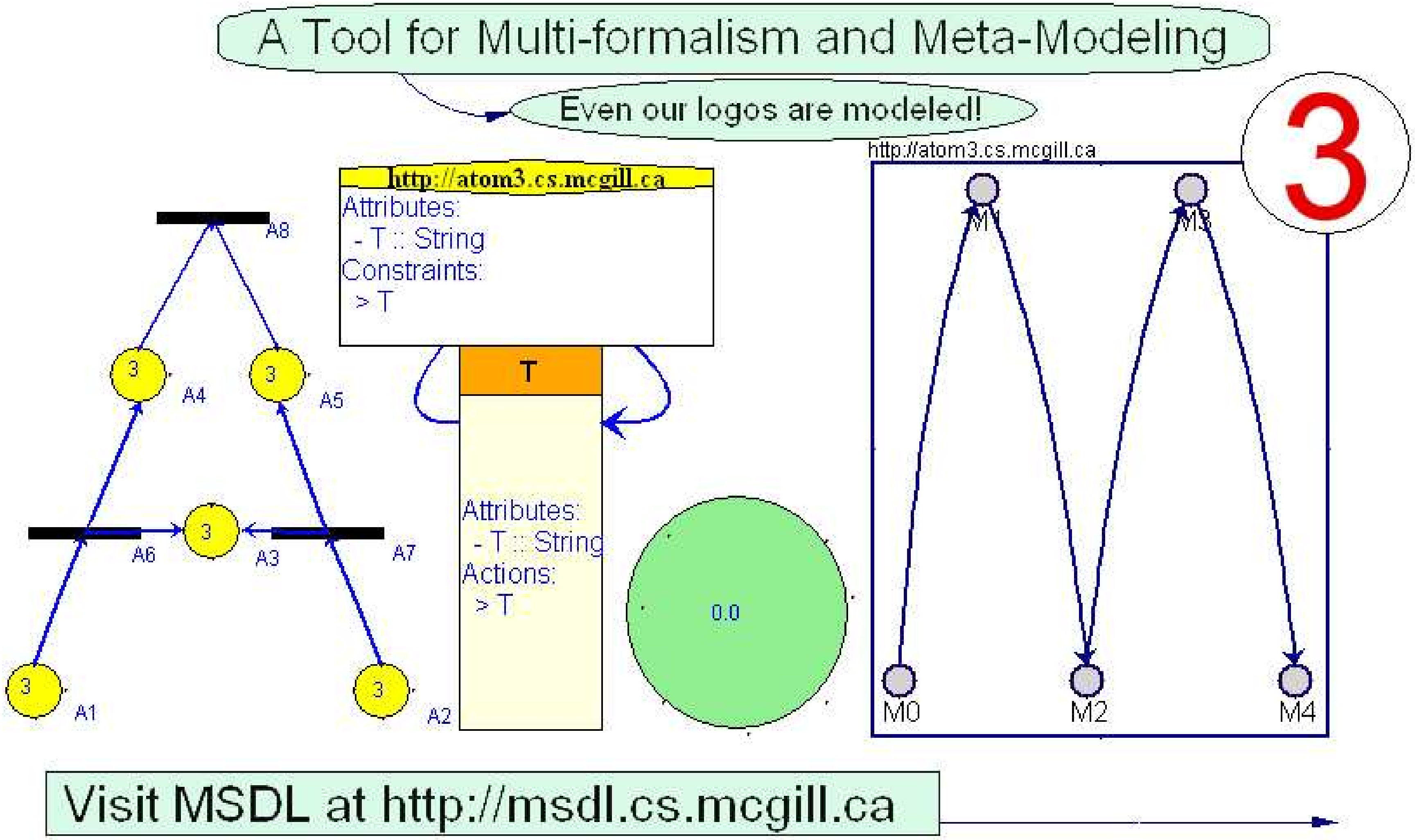
Modelling Languages as Sets



Last year’s (more complete) version


G[|.|]
:Const
value=EXP
:Signal
name=STR
vals=[...]
=
=
var
[|STR|]
abs=T
const
[|EXP|]
GCD
in
Const
value : FLT
0..1
*
Signal
name : STR
vals : FLT[]
out
IC
Delay
GEr
*
EquationSet
Equation
LHS
Expression
RHS
CD
LCD
[| . |]
SDCD
μ(LCBD)
= GCD
[|GCD|] =
LCBD
GG
LGG
[| . |]
SDGG
μ([|.|]CBD)
= G[|.|]CBD
[|G[|.|]CBD|]
ER
LER
[| . |]
SDER
GDIFF
[|GDIFF|]
μ
Notation
LN
[| . |]
SDN
κ(m)
[|κ(m)|]
κ
CBD
LCBD
[| . |]
SDCBD
m
[|m|]
DE
SDRDE
[| . |]
LRDE
vm
2
c
u
D
y
m
:Signal
name=y
vals=[...]
:Const
value=2
:Signal
name=c
vals=[...]
:Delay
:Signal
name=u
vals=[...]
ER
LER
[| . |]
SDER
GER
[|GER|] =
ER

From now on: tool view using AToM^3

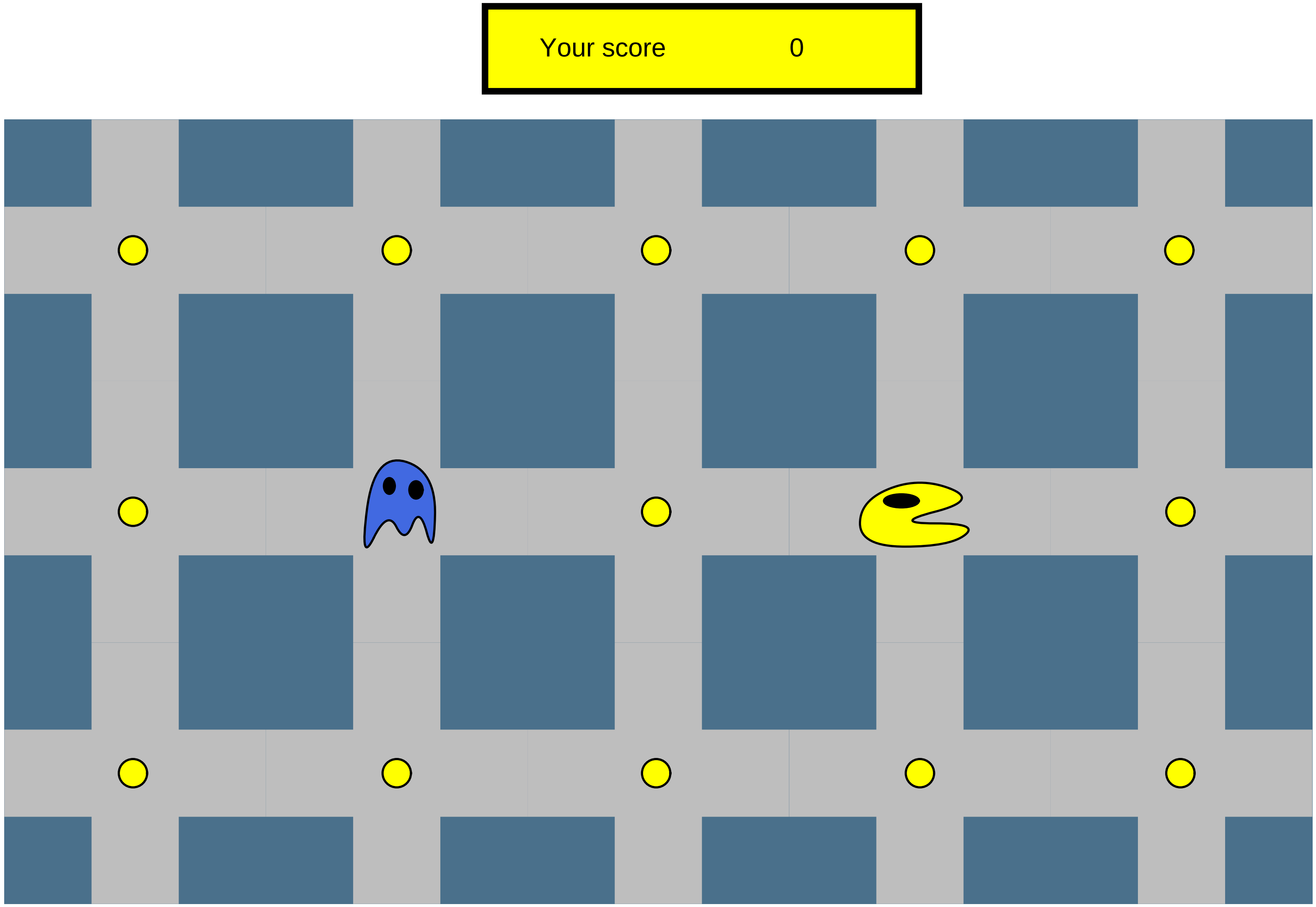


A Tool for Multi-formalism and Meta-Modeling

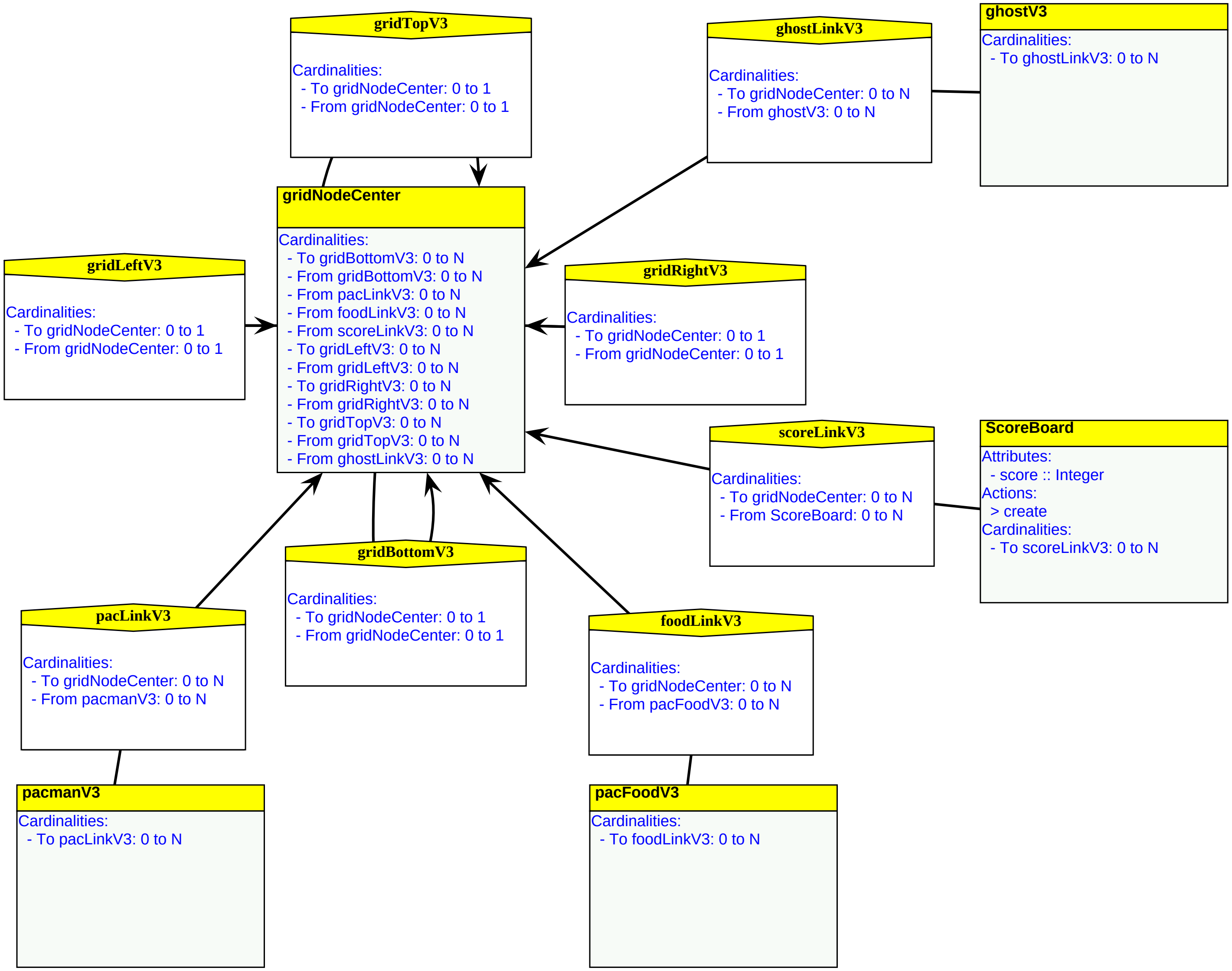
Even our logos are modeled!

Visit MSDL at http://msdl.cs.mcgill.ca

A model in the PacMan Formalism



Modelling Abstract Syntax (meta-model)



gridTopV3

Cardinalities:
- To gridNodeCenter: 0 to 1
- From gridNodeCenter: 0 to 1

ghostLinkV3

Cardinalities:
- To gridNodeCenter: 0 to N
- From ghostV3: 0 to N

ghostV3

Cardinalities:
- To ghostLinkV3: 0 to N

gridNodeCenter

Cardinalities:
- To gridBottomV3: 0 to N
- From gridBottomV3: 0 to N
- From pacLinkV3: 0 to N
- From foodLinkV3: 0 to N
- From scoreLinkV3: 0 to N
- To gridLeftV3: 0 to N
- From gridLeftV3: 0 to N
- To gridRightV3: 0 to N
- From gridRightV3: 0 to N
- To gridTopV3: 0 to N
- From gridTopV3: 0 to N
- From ghostLinkV3: 0 to N

gridLeftV3

Cardinalities:
- To gridNodeCenter: 0 to 1
- From gridNodeCenter: 0 to 1

gridRightV3

Cardinalities:
- To gridNodeCenter: 0 to 1
- From gridNodeCenter: 0 to 1

scoreLinkV3

Cardinalities:
- To gridNodeCenter: 0 to N
- From ScoreBoard: 0 to N

ScoreBoard

Attributes:
- score :: Integer

Actions:
- > create

Cardinalities:
- To scoreLinkV3: 0 to N

gridBottomV3

Cardinalities:
- To gridNodeCenter: 0 to 1
- From gridNodeCenter: 0 to 1

pacLinkV3

Cardinalities:
- To gridNodeCenter: 0 to N
- From pacmanV3: 0 to N

foodLinkV3

Cardinalities:
- To gridNodeCenter: 0 to N
- From pacFoodV3: 0 to N

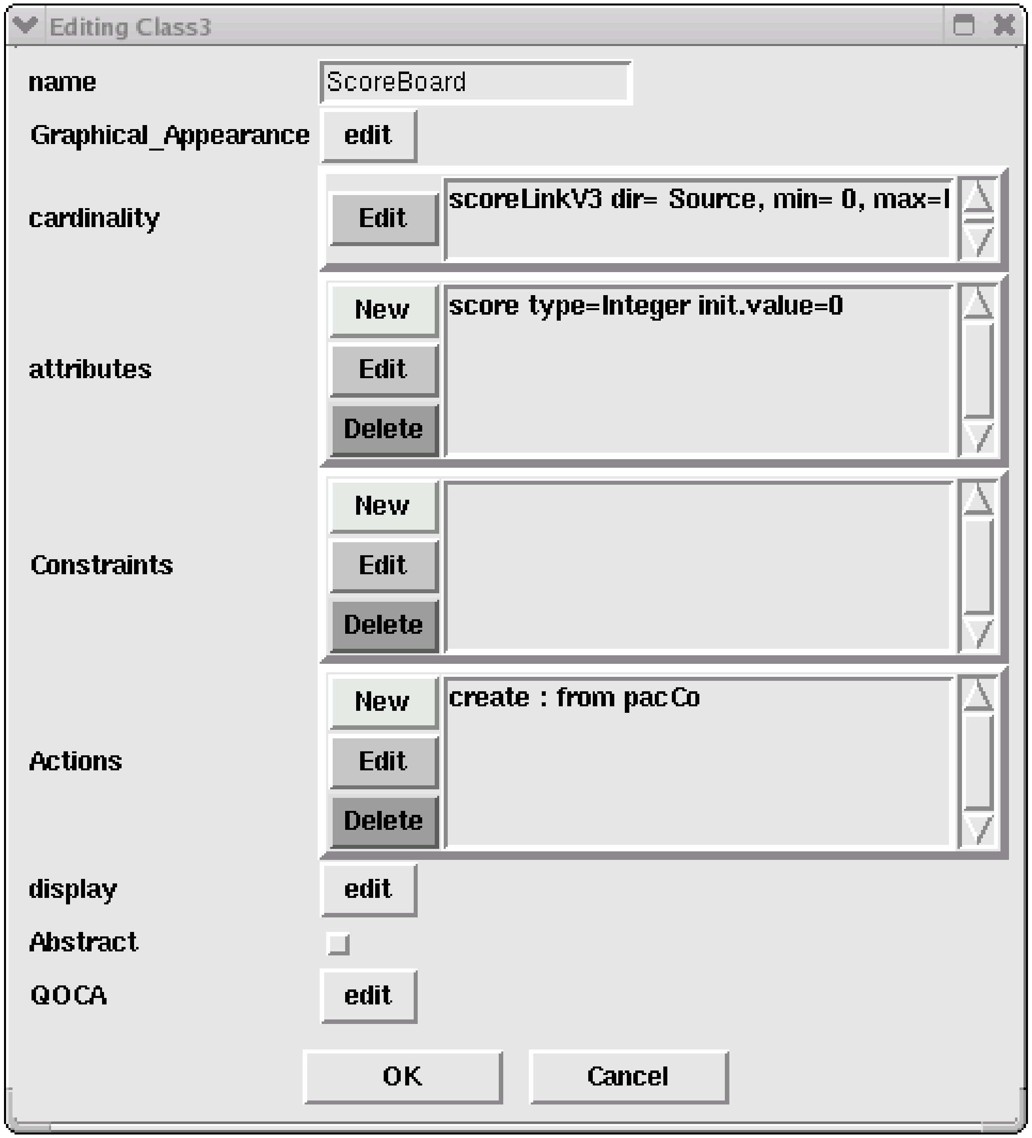
pacmanV3

Cardinalities:
- To pacLinkV3: 0 to N

pacFoodV3

Cardinalities:
- To foodLinkV3: 0 to N

Modelling the Scoreboard Entity



name ScoreBoard

Graphical_Appearance

cardinality scoreLinkV3 dir= Source, min= 0, max=I

attributes score type=Integer init.value=0

Constraints

Actions create : from pacCo

display

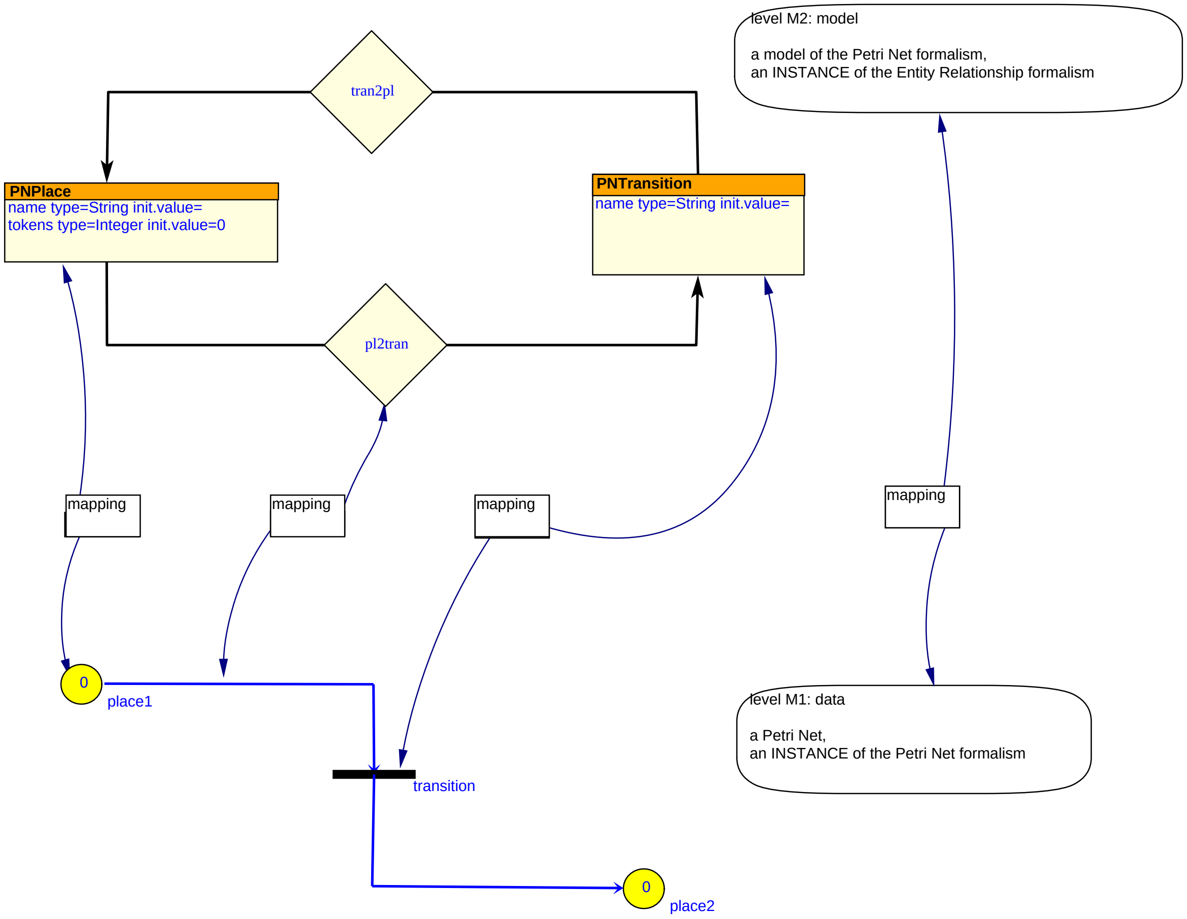
Abstract □

QOCA

Synthesis of Code from this Design model

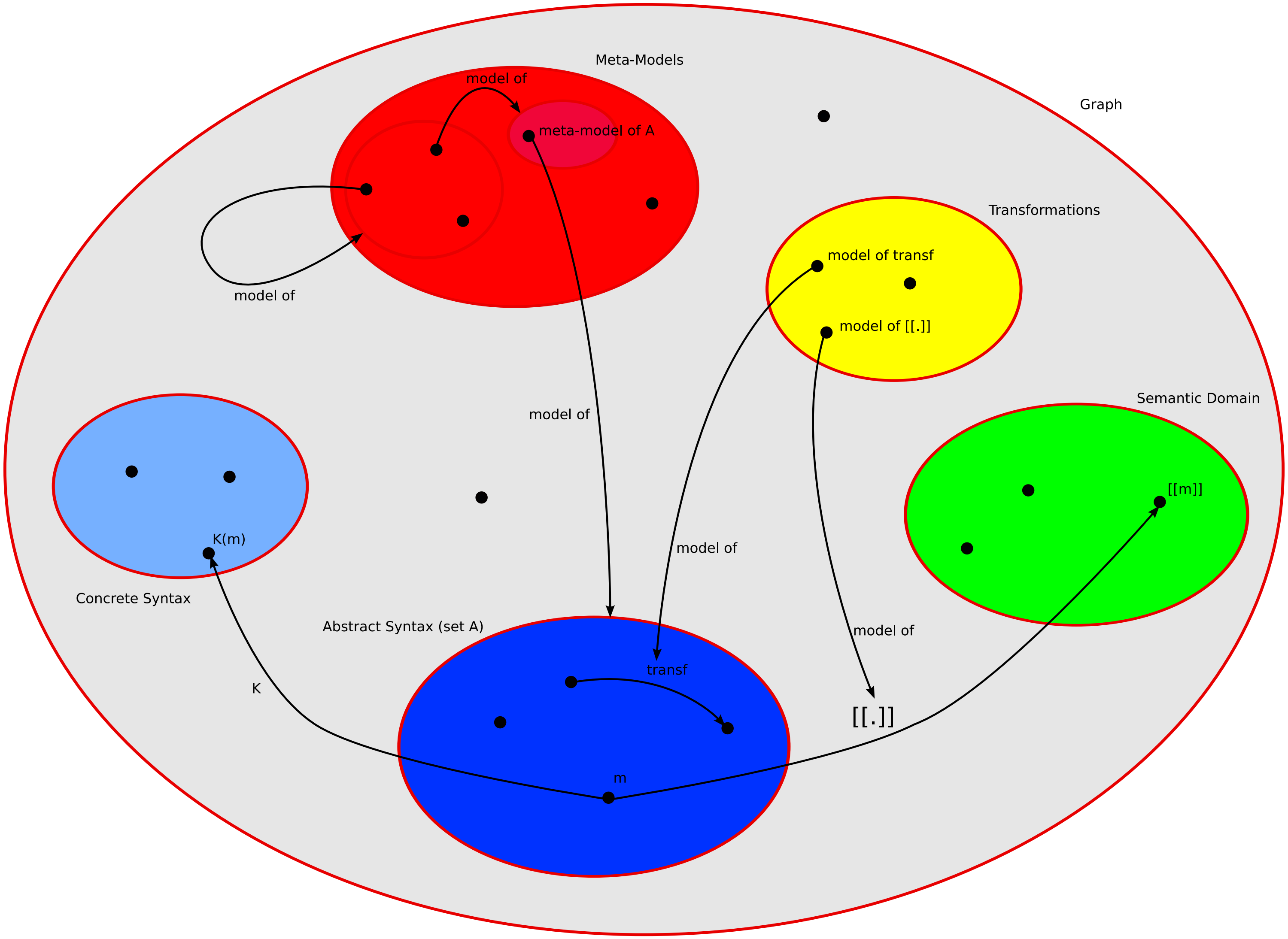
```
class ScoreBoard(ASGNode, ATOM3Type):

  def __init__(self, parent = None):
     ASGNode.__init__(self)
     ATOM3Type.__init__(self)
     self.graphClass_ = graph_ScoreBoard
     self.isGraphObjectVisual = True
     self.parent = parent
     self.score=ATOM3Integer(0)
     self.generatedAttributes = {’score’: (’ATOM3Integer’ ) }
     self.directEditing = [1]

  def clone(self):
     cloneObject = ScoreBoard( self.parent )
     for atr in self.realOrder:
        cloneObject.setAttrValue(atr, self.getAttrValue(atr).clone() )
     ASGNode.cloneActions(self, cloneObject)
     return cloneObject
```

Meta-modelling: model-instance morphism or . . .

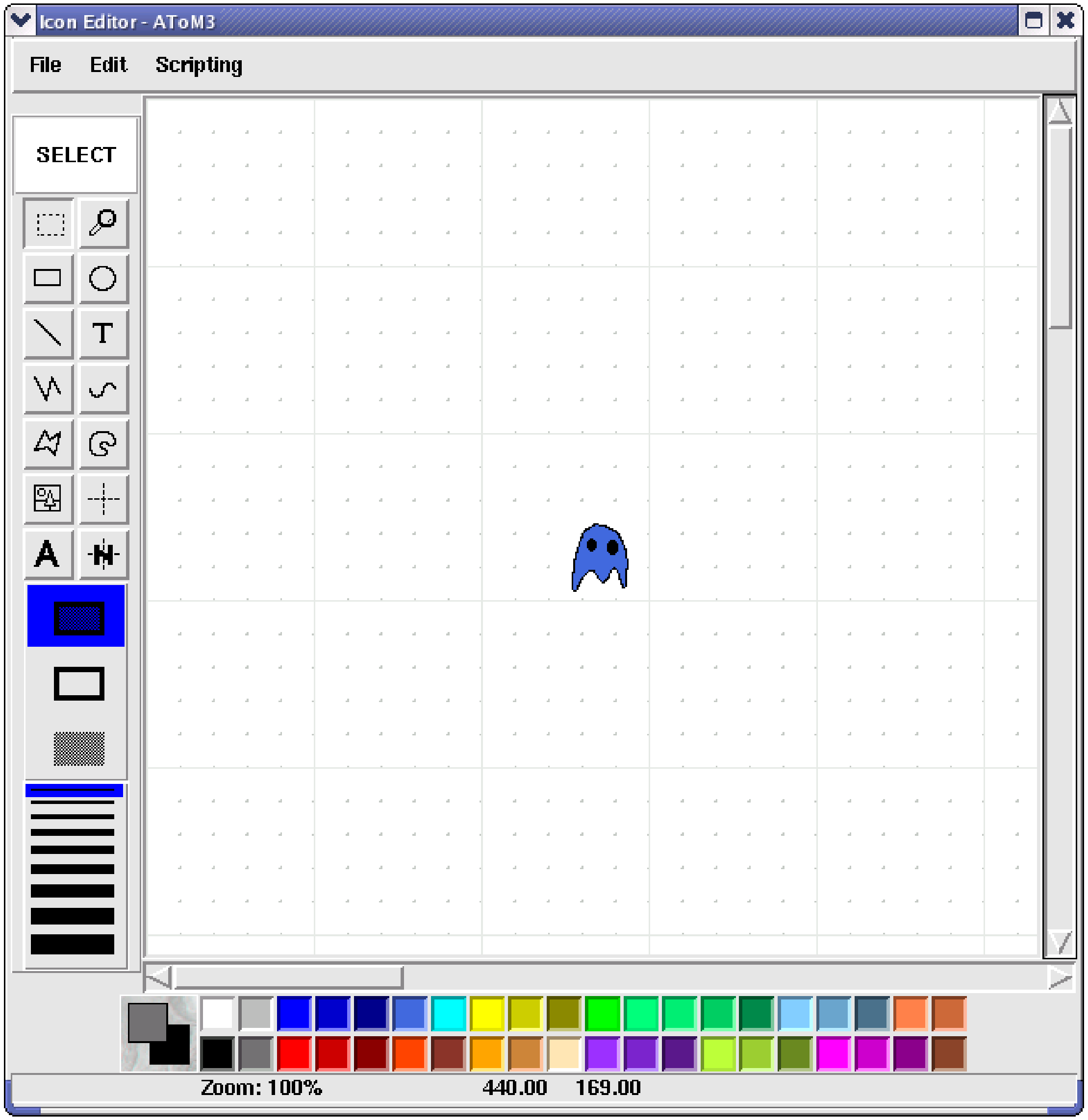


level M2: model
a model of the Petri Net formalism,
an INSTANCE of the Entity Relationship formalism
tran2pl
PNPlace
name type=String init.value=
tokens type=Integer init.value=0
PNTransition
name type=String init.value=
pl2tran
mapping
mapping
mapping
mapping
0
place1
transition
0
place2
level M1: data
a Petri Net,
an INSTANCE of the Petri Net formalism

Meta-meta-. . . : Meta-circularity



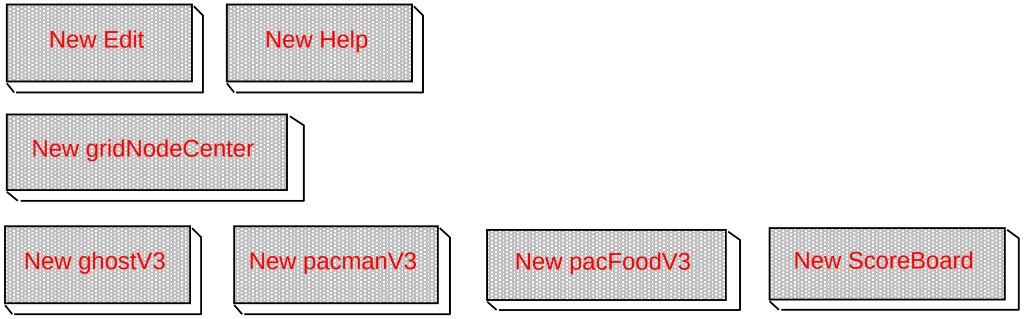
Modelling Ghost Concrete Visual Syntax



PacFoodLink Concrete Visual Syntax

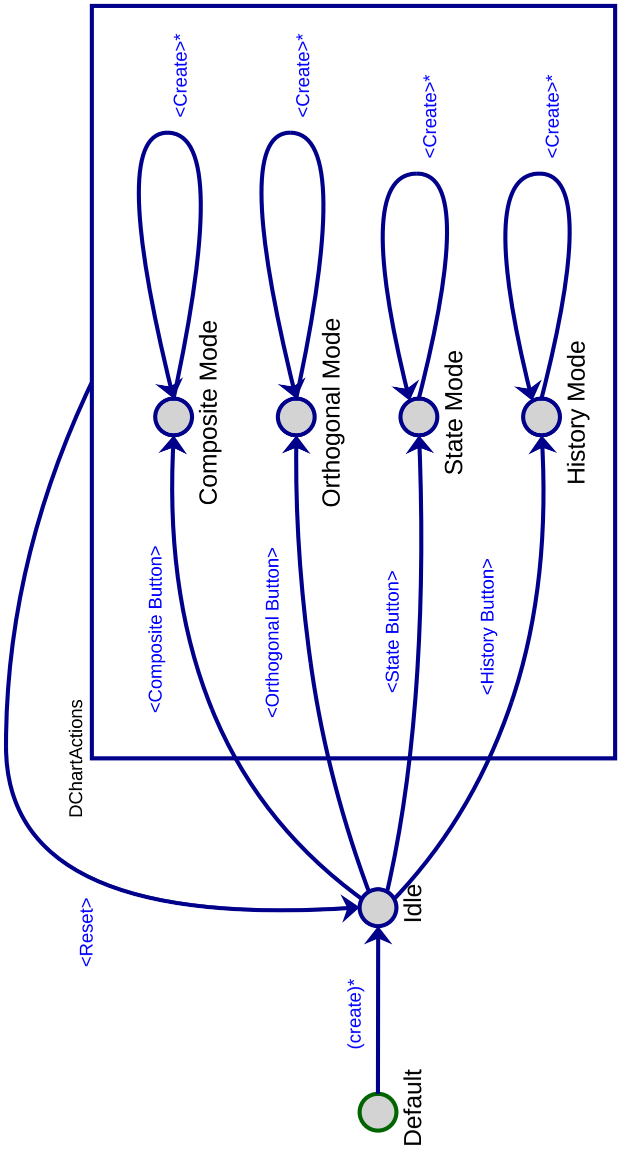
```
# Get n1, n2 end-points of the link
n1 = self.in_connections_[0]
n2 = self.out_connections_[0]

# g1 and g2 are the graphEntity visual objects
g0 = self.graphObject_   # the link
g1 = n1.graphObject_     # first end-point
g2 = n2.graphObject_     # second end-poing

# Get the high level constraint helper and solver
from Qoca.atom3constraints.OffsetConstraints
     import OffsetConstraints
oc = OffsetConstraints(self.parent.qocaSolver)

# The constraints
oc.CenterX((g1, g2, g0))
oc.CenterY((g1, g2, g0))
oc.resolve()
```

Synthesize + Customize Buttons model

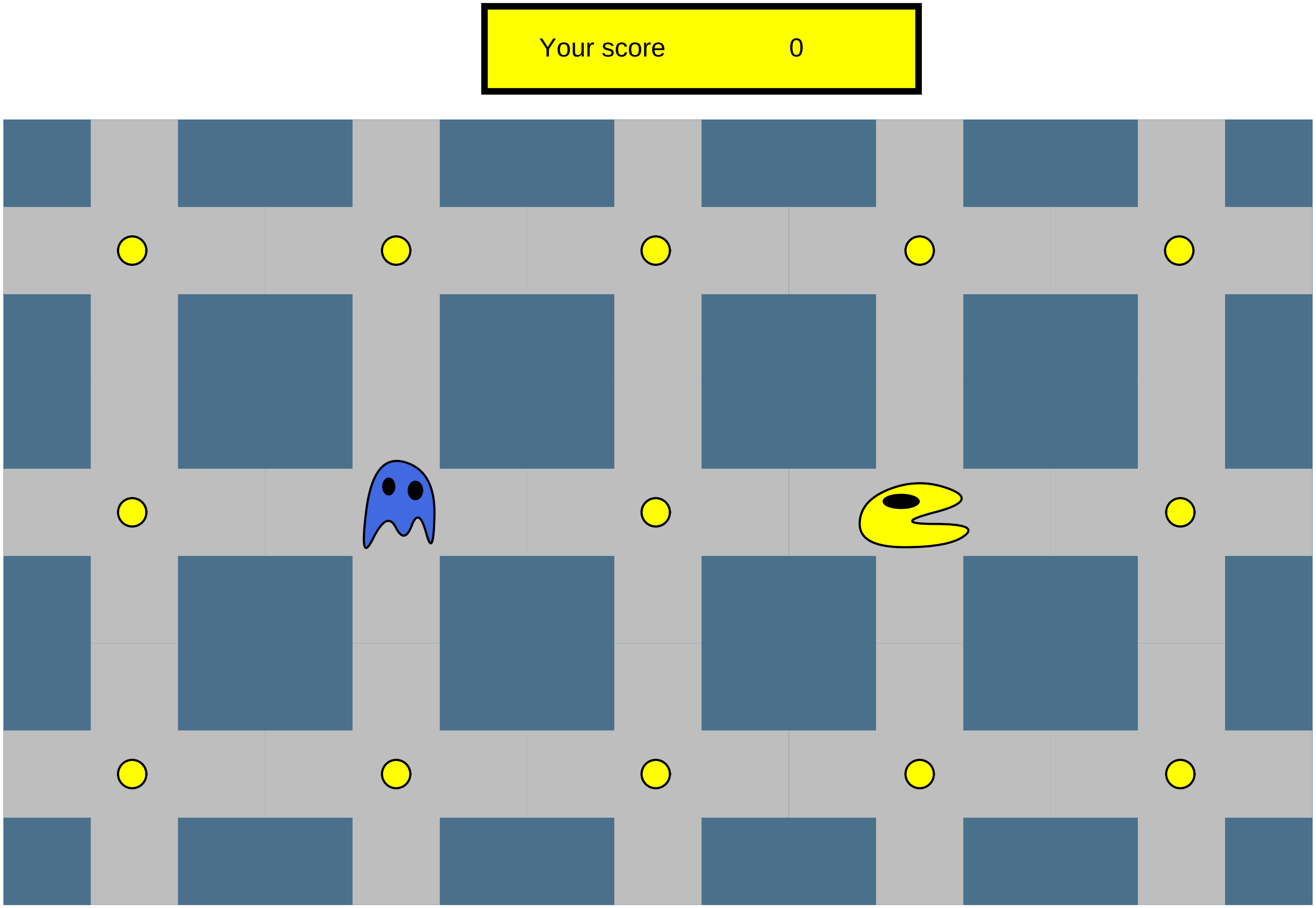


Note: create *vs.* execute

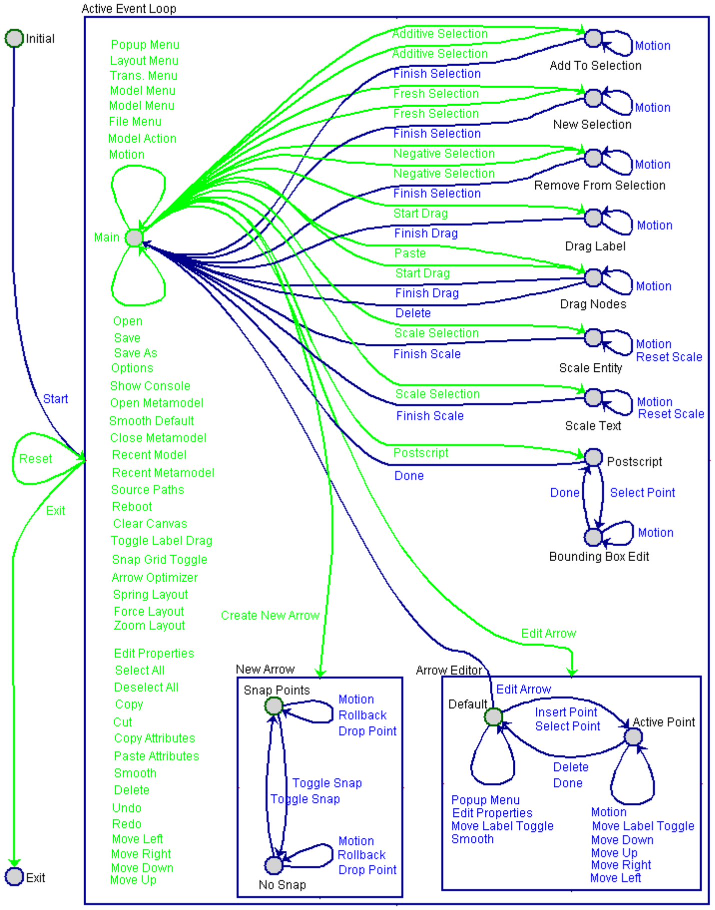
Buttons behaviour model

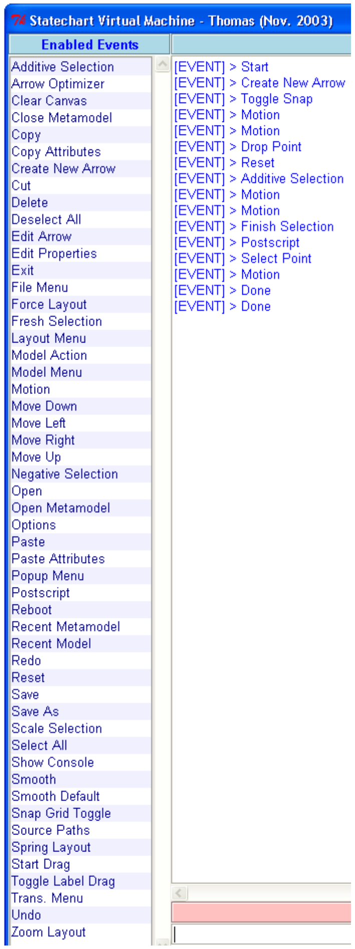


Can now build valid PacMan models ?



Model the GUI's Complete Reactive Behaviour ! in the Statechart formalism



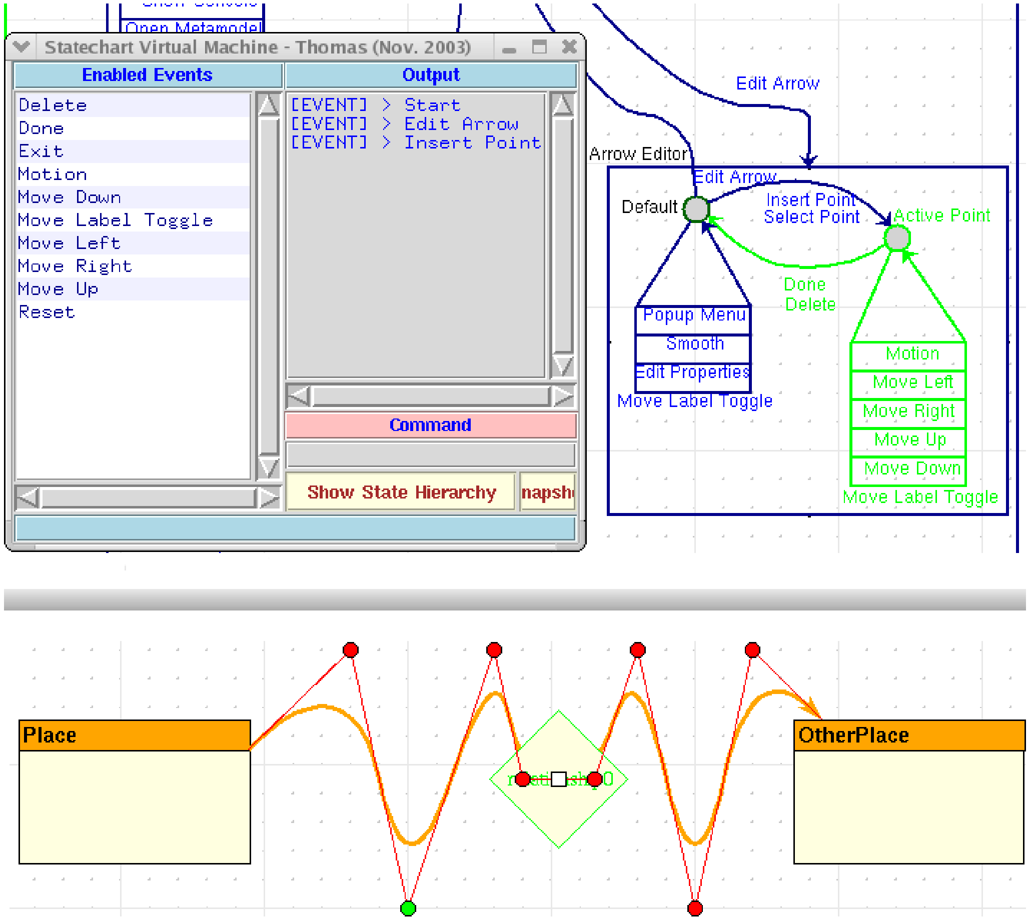

Statechart Virtual Machine - Thomas (Nov. 2003)

Enabled Events

Additive Selection
Arrow Optimizer
Clear Canvas
Close Metamodel
Copy
Copy Attributes
Create New Arrow
Cut
Delete
Deselect All
Edit Arrow
Edit Properties
Exit
File Menu
Force Layout
Fresh Selection
Layout Menu
Model Action
Model Menu
Motion
Move Down
Move Left
Move Right
Move Up
Negative Selection
Open
Open Metamodel
Options
Paste
Paste Attributes
Popup Menu
Postscript
Reboot
Recent Metamodel
Recent Model
Redo
Reset
Save
Save As
Scale Selection
Select All
Show Console
Smooth
Smooth Default
Snap Grid Toggle
Source Paths
Spring Layout
Start Drag
Toggle Label Drag
Trans. Menu
Undo
Zoom Layout

```
[EVENT] > Start
[EVENT] > Create New Arrow
[EVENT] > Toggle Snap
[EVENT] > Motion
[EVENT] > Motion
[EVENT] > Drop Point
[EVENT] > Reset
[EVENT] > Additive Selection
[EVENT] > Motion
[EVENT] > Motion
[EVENT] > Finish Selection
[EVENT] > Postscript
[EVENT] > Select Point
[EVENT] > Motion
[EVENT] > Done
[EVENT] > Done
```

The GUI’s reactive behaviour in action



Statechart Virtual Machine - Thomas (Nov. 2003)

Enabled Events

Delete
Done
Exit
Motion
Move Down
Move Label Toggle
Move Left
Move Right
Move Up
Reset

Output

```
[EVENT] > Start
[EVENT] > Edit Arrow
[EVENT] > Insert Point
```

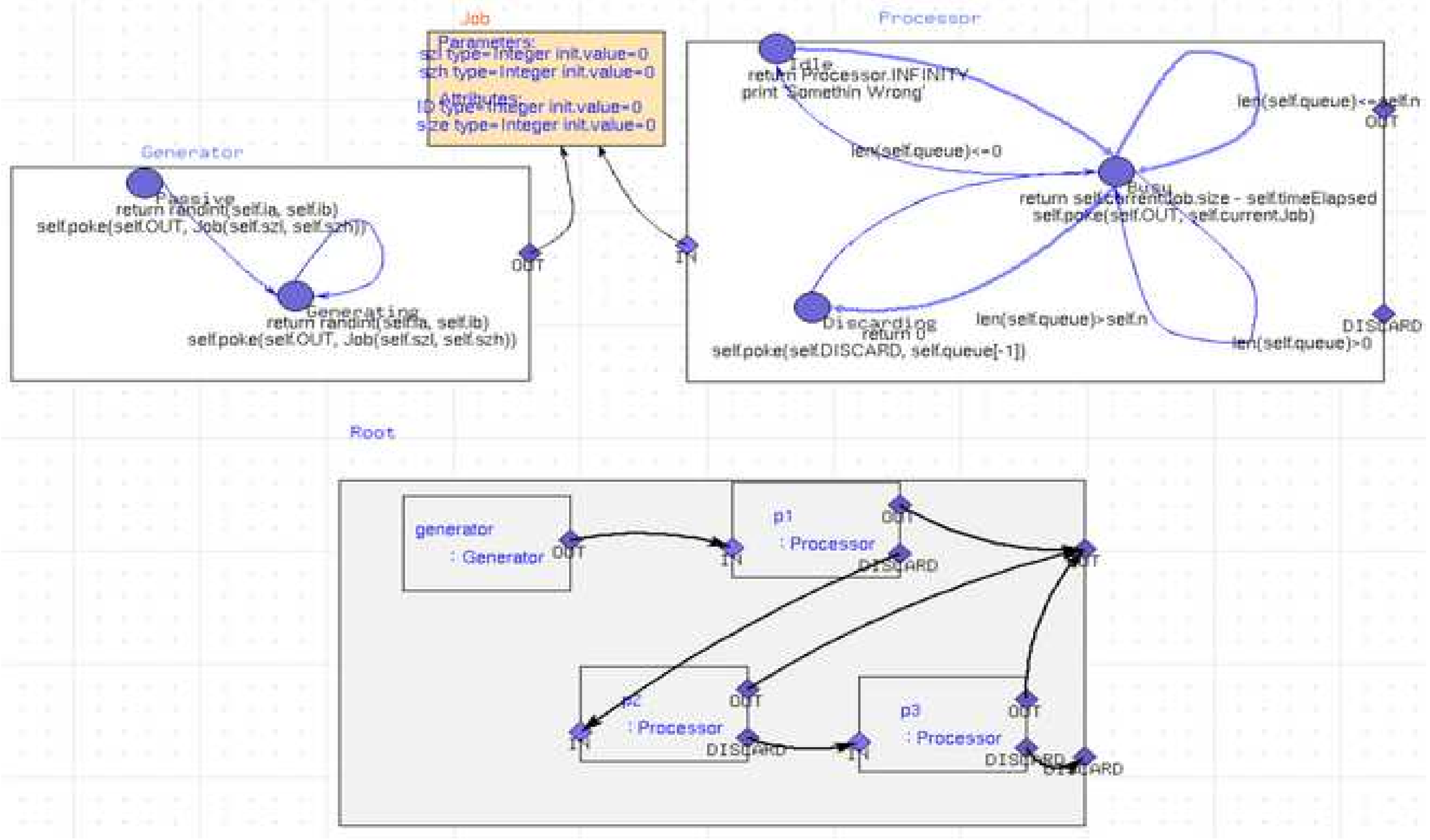
Command

Show State Hierarchy

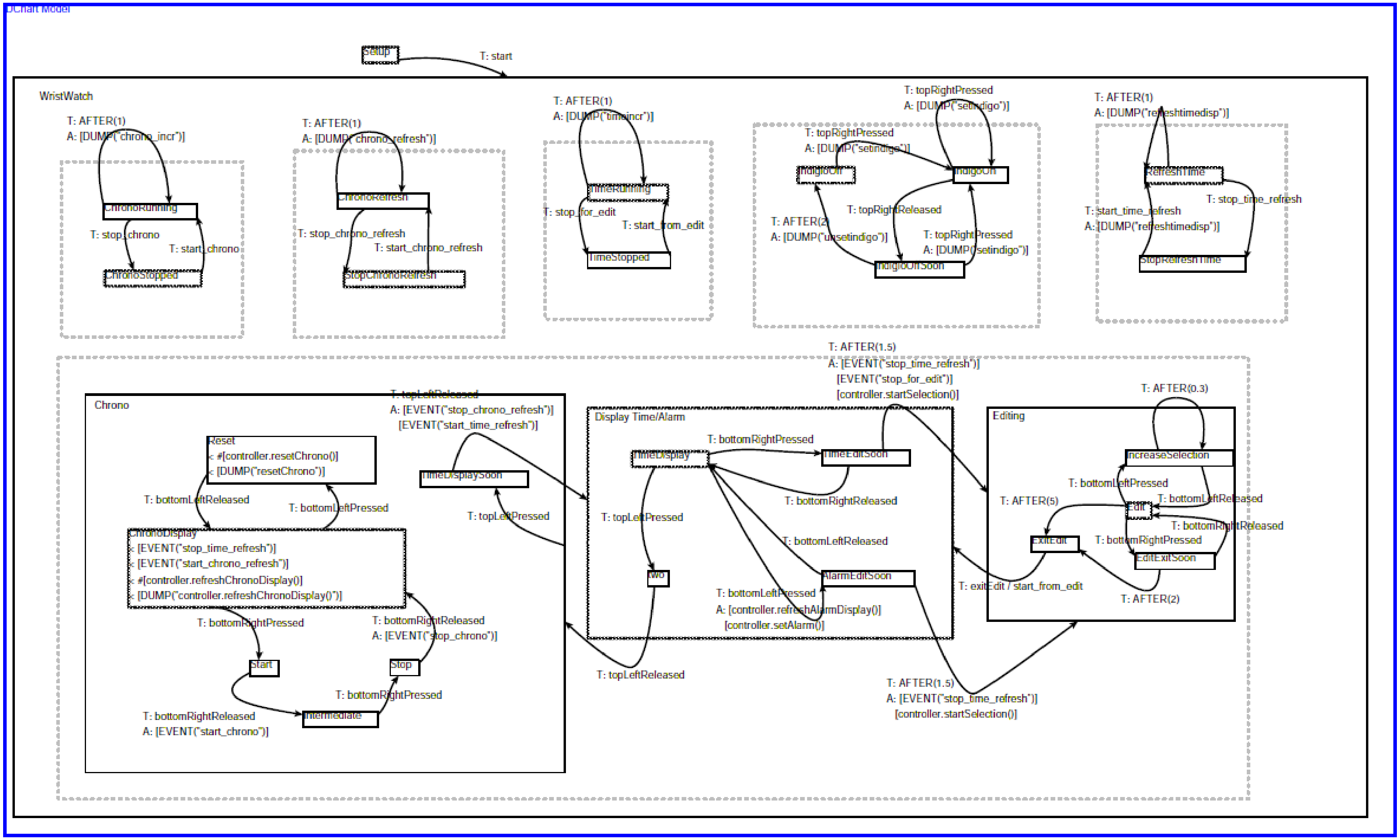
challenge: what is the *optimal* formalism to specify GUI reactive behaviour ?

Optimal formalism: need more modularity !

Example with nesting: DEVS


Job
Parameters:
szl type=Integer init.value=0
szh type=Integer init.value=0
Attributes:
ID type=Integer init.value=0
size type=Integer init.value=0
Processor
Idle
return Processor.INFINITY
print 'Somethin Wrong'
len(self.queue)<=self.n
OUT
len(self.queue)<=0
Busy
return self.currentJob.size - self.timeElapsed
self.poke(self.OUT, self.currentJob)
IN
Discarding
return 0
self.poke(self.DISCARD, self.queue[-1])
len(self.queue)>self.n
len(self.queue)>0
DISCARD
Generator
Passive
return randint(self.ia, self.ib)
self.poke(self.OUT, Job(self.szl, self.szh))
Generating
return randint(self.ia, self.ib)
self.poke(self.OUT, Job(self.szl, self.szh))
OUT
Root
generator
: Generator
OUT
IN
p1
: Processor
OUT
DISCARD
OUT
p2
: Processor
IN
OUT
DISCARD
p3
: Processor
IN
OUT
DISCARD
DISCARD

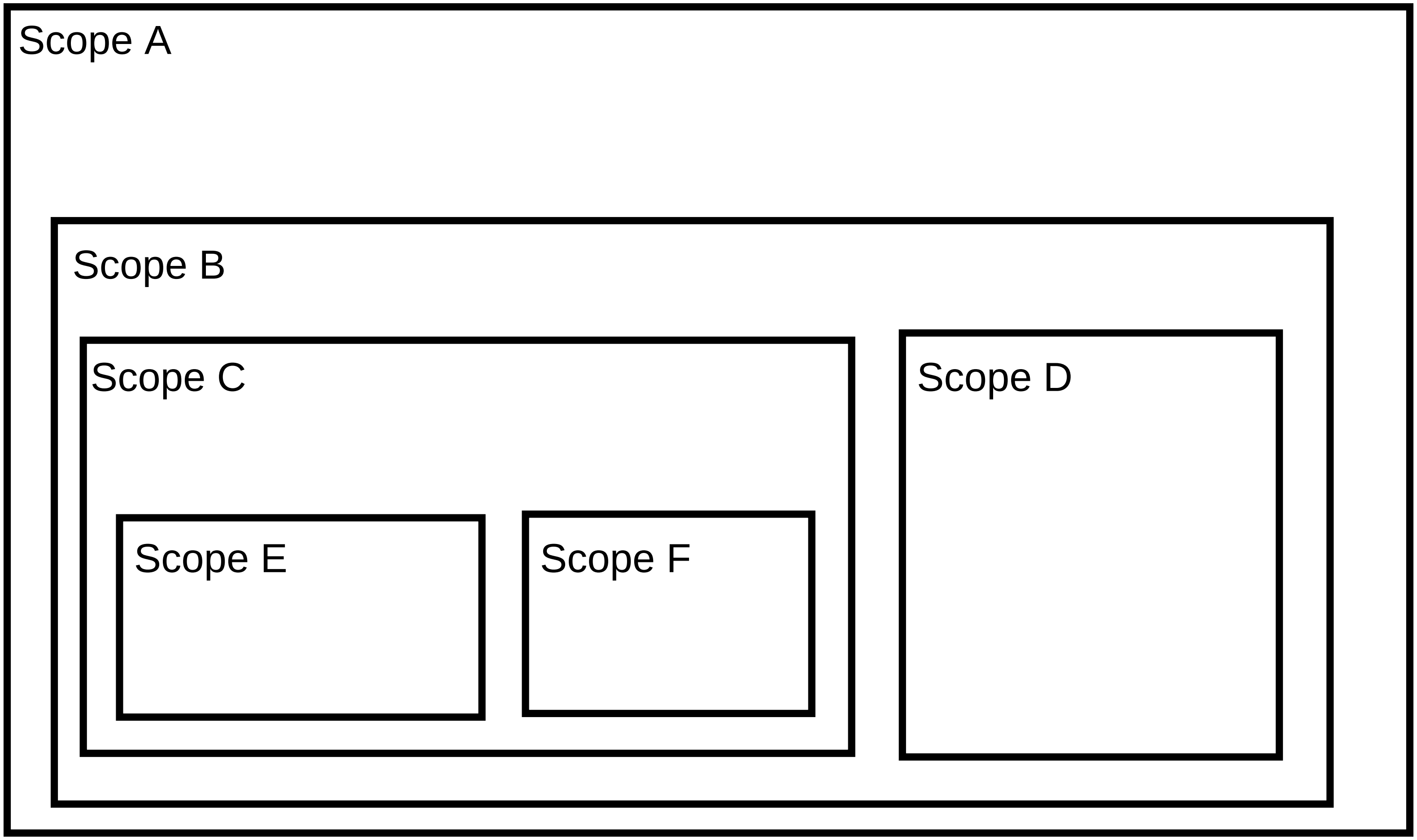
Example with nesting: DCharts



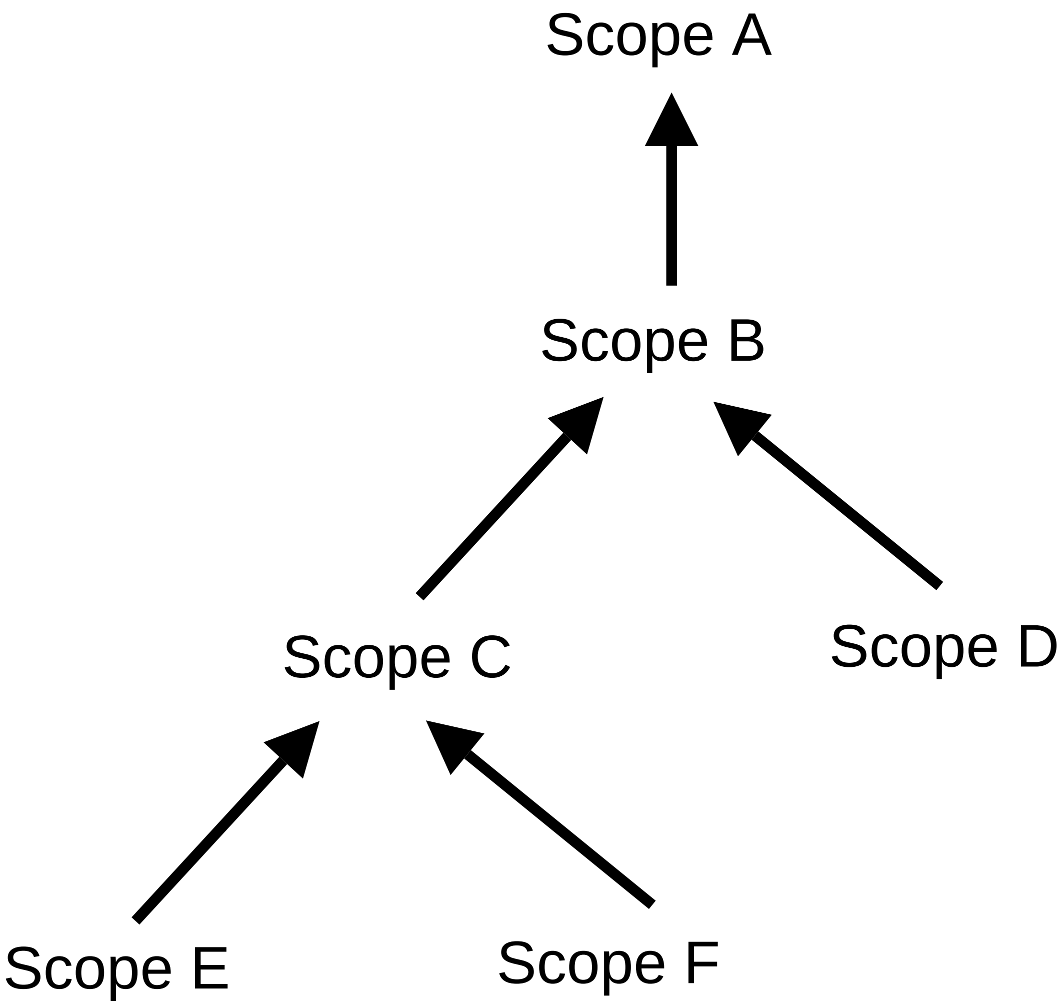
Modelling Reactive Visual Modelling Environments

- multi-formalism, encapsulated
- nesting/scoping
- glue reactive behaviour, syntax check, and layout

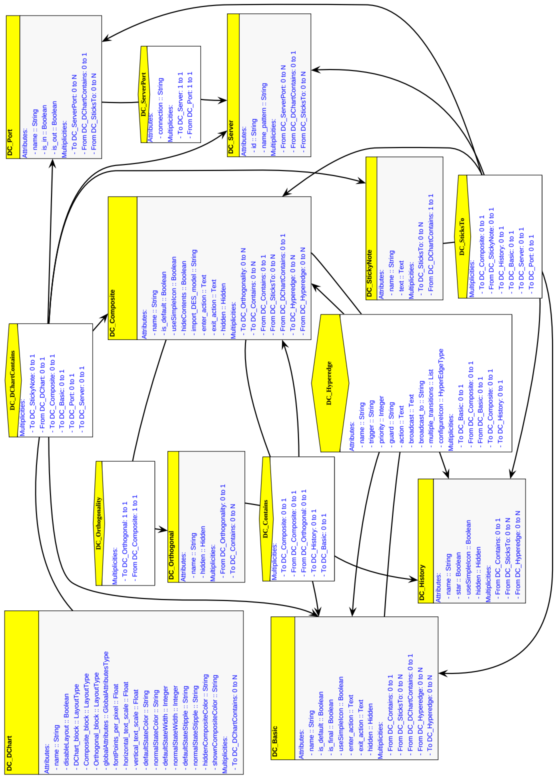
Scope Hierarchy



Nested Event Propagation



DCharts meta-model in Class Diagrams



DC_DChart
Attributes:
- name :: String
- disableLayout :: Boolean
- DChart_block :: LayoutType
- Composite_block :: LayoutType
- Orthogonal_block :: LayoutType
- globalAttributes :: GlobalAttributesType
- fontPoints_per_pixel :: Float
- horizontal_text_scale :: Float
- vertical_text_scale :: Float
- defaultStateColor :: String
- normalStateColor :: String
- defaultStateWidth :: Integer
- normalStateWidth :: Integer
- defaultStateStipple :: String
- normalStateStipple :: String
- hiddenCompositeColor :: String
- shownCompositeColor :: String
Multiplicities:
- To DC_DChartContains: 0 to N

DC_DChartContains
Multiplicities:
- To DC_StickyNote: 0 to 1
- From DC_DChart: 0 to 1
- To DC_Composite: 0 to 1
- To DC_Basic: 0 to 1
- To DC_Port: 0 to 1
- To DC_Server: 0 to 1

DC_Port
Attributes:
- name :: String
- is_in :: Boolean
- is_out :: Boolean
Multiplicities:
- To DC_ServerPort: 0 to N
- From DC_DChartContains: 0 to 1
- From DC_SticksTo: 0 to N

DC_Orthogonality
Multiplicities:
- To DC_Orthogonal: 1 to 1
- From DC_Composite: 1 to 1

DC_Composite
Attributes:
- name :: String
- is_default :: Boolean
- useSimpleIcon :: Boolean
- hideContents :: Boolean
- import_DES_model :: String
- enter_action :: Text
- exit_action :: Text
- hidden :: Hidden
Multiplicities:
- To DC_Orthogonality: 0 to N
- To DC_Contains: 0 to N
- From DC_Contains: 0 to 1
- From DC_SticksTo: 0 to N
- From DC_DChartContains: 0 to 1
- To DC_Hyperedge: 0 to N
- From DC_Hyperedge: 0 to N

DC_ServerPort
Attributes:
- connection :: String
Multiplicities:
- To DC_Server: 1 to 1
- From DC_Port: 1 to 1

DC_Orthogonal
Attributes:
- name :: String
- hidden :: Hidden
Multiplicities:
- From DC_Orthogonality: 0 to 1
- To DC_Contains: 0 to N

DC_Server
Attributes:
- id :: String
- name_pattern :: String
Multiplicities:
- From DC_ServerPort: 0 to N
- From DC_DChartContains: 0 to 1
- From DC_SticksTo: 0 to N

DC_Contains
Multiplicities:
- To DC_Composite: 0 to 1
- From DC_Composite: 0 to 1
- From DC_Orthogonal: 0 to 1
- To DC_History: 0 to 1
- To DC_Basic: 0 to 1

DC_Basic
Attributes:
- name :: String
- is_default :: Boolean
- is_final :: Boolean
- useSimpleIcon :: Boolean
- enter_action :: Text
- exit_action :: Text
- hidden :: Hidden
Multiplicities:
- From DC_Contains: 0 to 1
- From DC_SticksTo: 0 to N
- From DC_DChartContains: 0 to 1
- From DC_Hyperedge: 0 to N
- To DC_Hyperedge: 0 to N

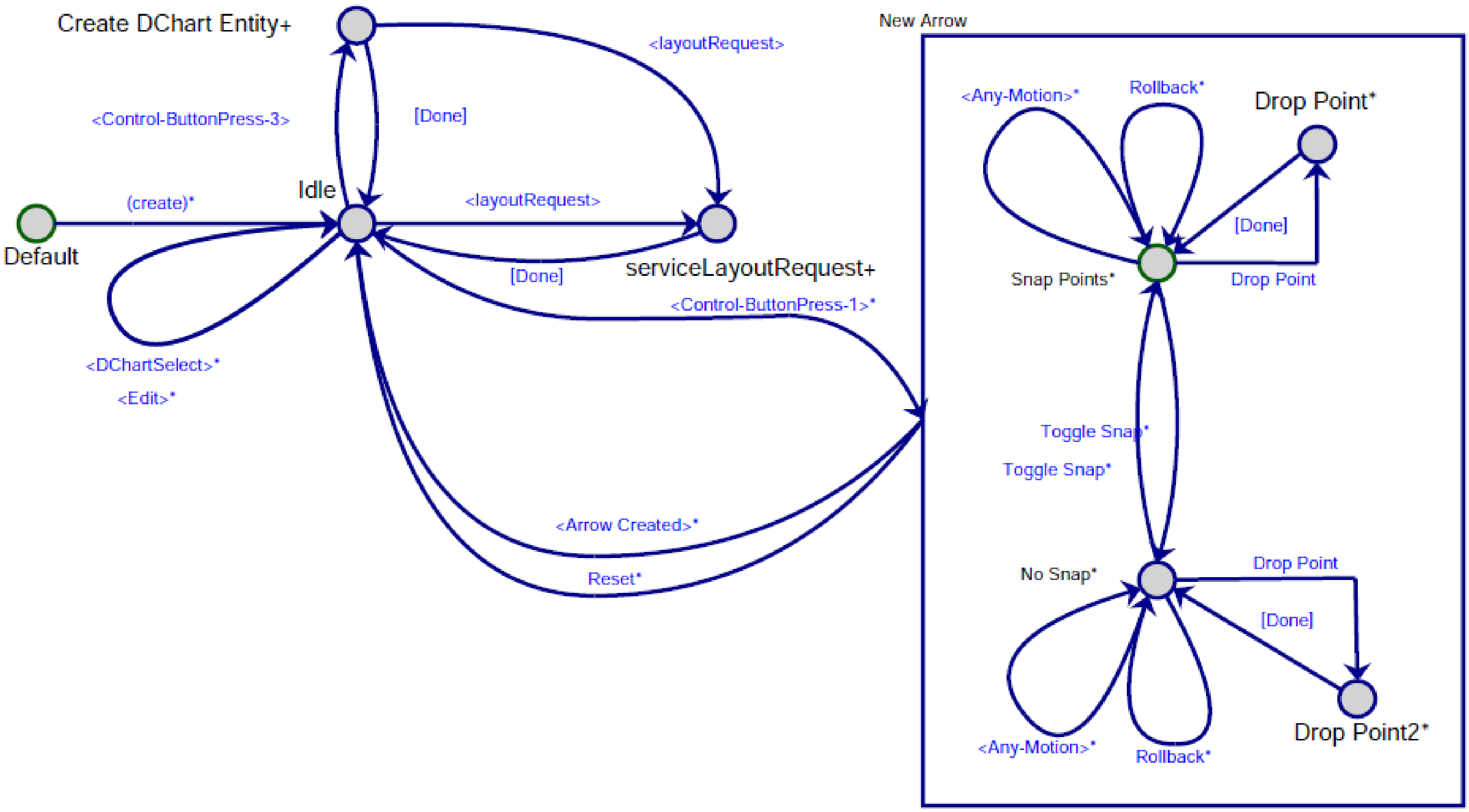
DC_Hyperedge
Attributes:
- name :: String
- trigger :: String
- priority :: Integer
- guard :: String
- action :: Text
- broadcast :: Text
- broadcast_to :: String
- multiple_transitions :: List
- configureIcon :: HyperEdgeType
Multiplicities:
- To DC_Basic: 0 to 1
- From DC_Composite: 0 to 1
- From DC_Basic: 0 to 1
- To DC_Composite: 0 to 1
- To DC_History: 0 to 1

DC_StickyNote
Attributes:
- name :: String
- text :: Text
Multiplicities:
- To DC_SticksTo: 0 to N
- From DC_DChartContains: 1 to 1

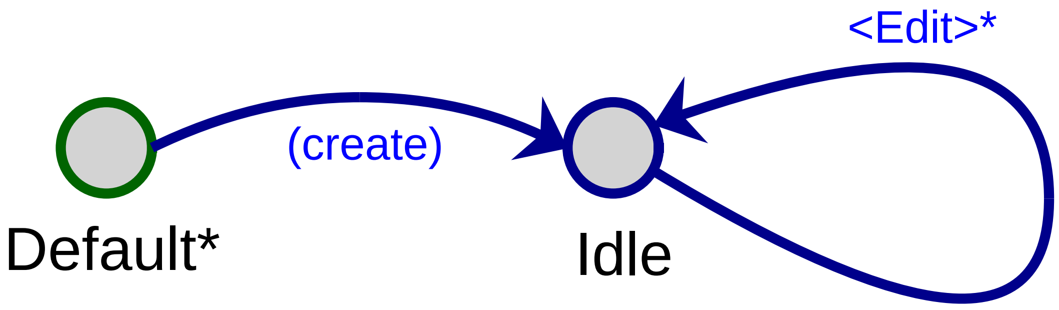
DC_History
Attributes:
- name :: String
- star :: Boolean
- useSimpleIcon :: Boolean
- hidden :: Hidden
Multiplicities:
- From DC_Contains: 0 to 1
- From DC_SticksTo: 0 to N
- From DC_Hyperedge: 0 to N

DC_SticksTo
Multiplicities:
- To DC_Composite: 0 to 1
- From DC_StickyNote: 0 to 1
- To DC_History: 0 to 1
- To DC_Basic: 0 to 1
- To DC_Server: 0 to 1
- To DC_Port: 0 to 1

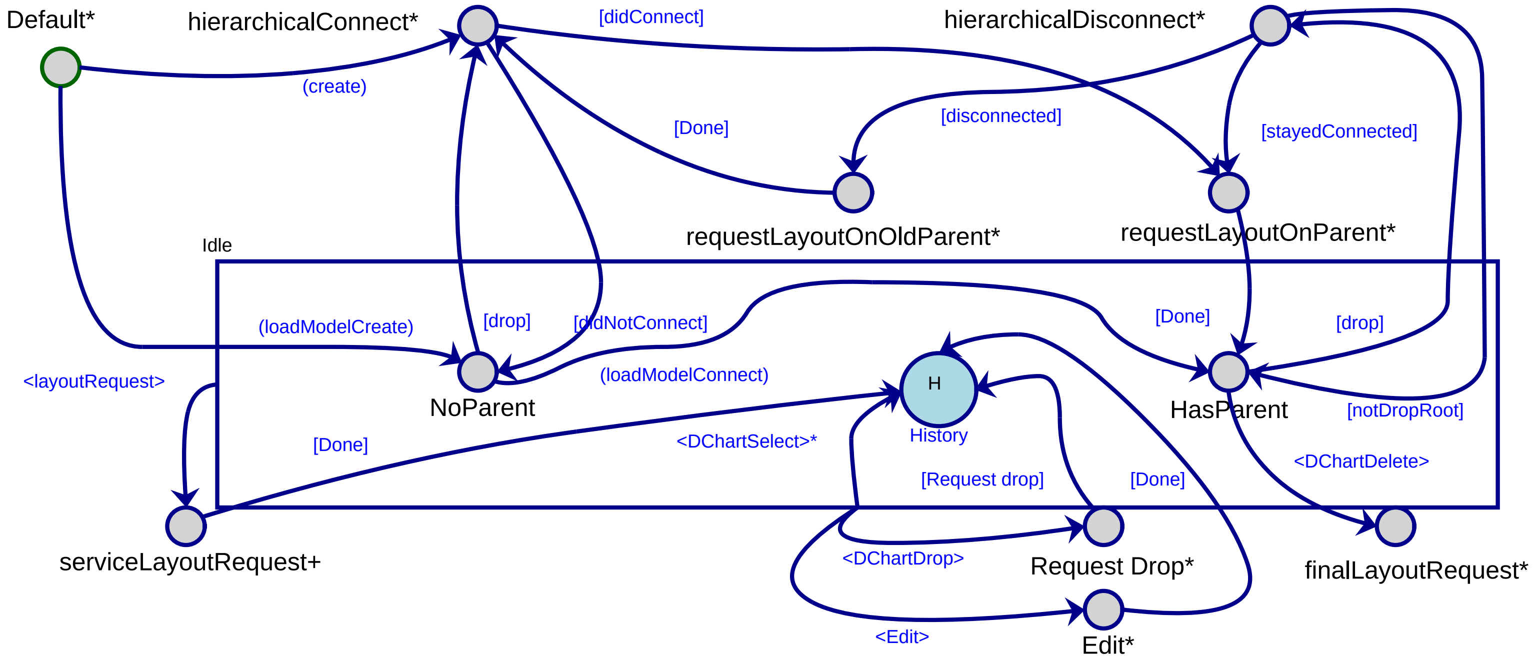
Overall DChart Modelling Environment Behaviour



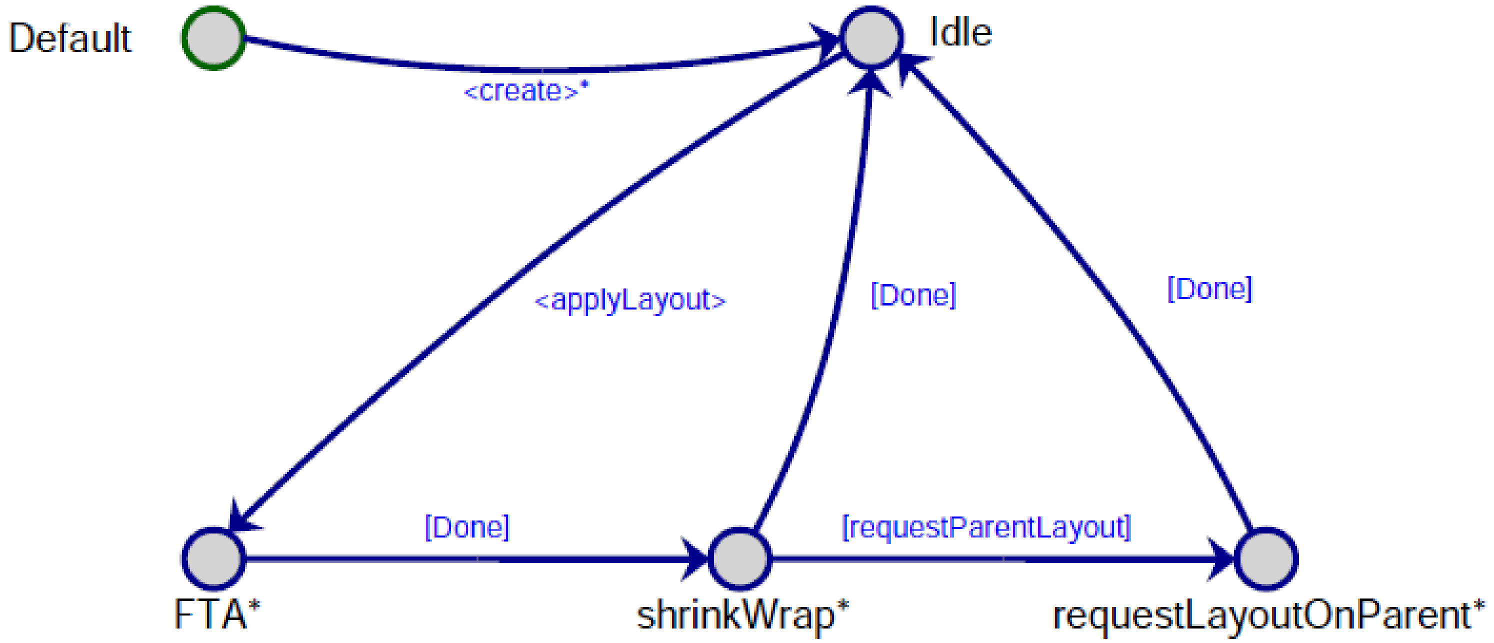
“DChart Transition” Behaviour



“DChart Composite” Behaviour



“DCharts (Force Transfer) Layout” Behaviour



Graph Grammars to Specify Model Transformations

Rationale:

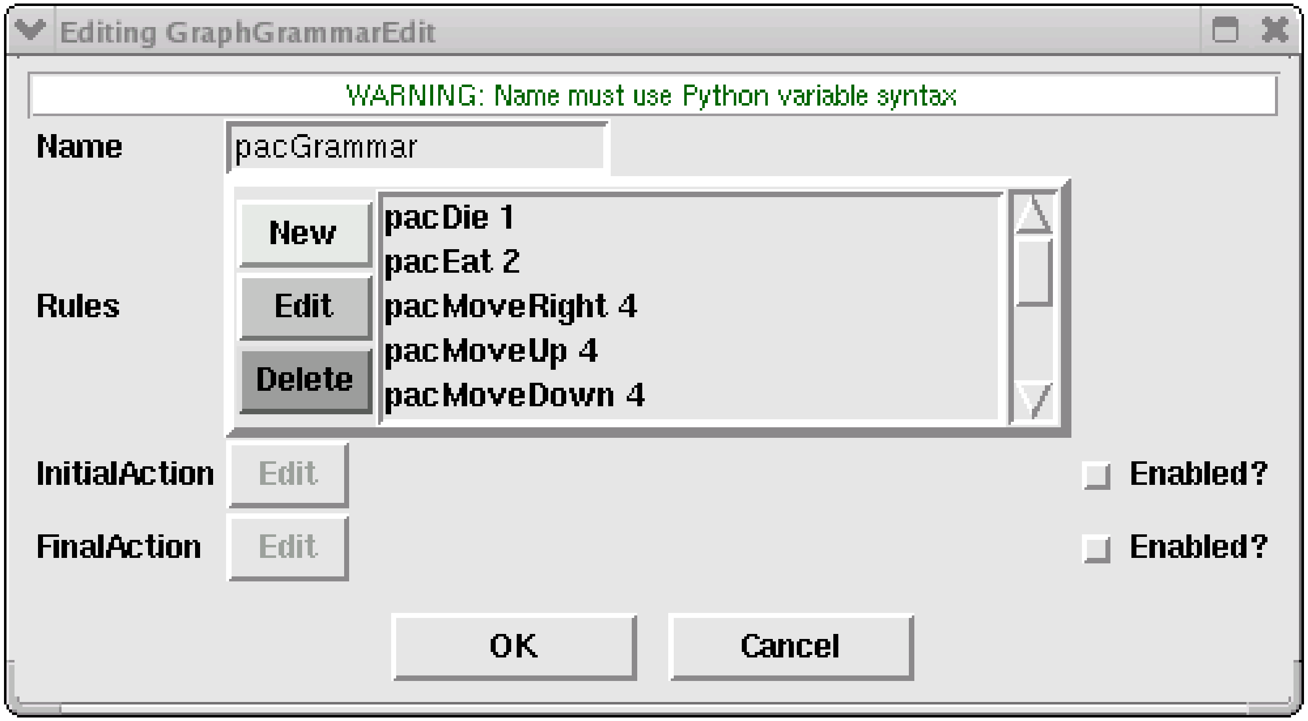
Models are often graph-like ⇒ natural to express model transformation by means of graph transformation models.

Ehrig, H., G. Engels, H.-J. Kreowski, and G. Rozenberg.
Handbook of graph grammars and computing by graph transformation.
1999. World Scientific.

Tools:

GME/GReAT, PROGRES, AGG, AToM^3, Fujaba, GROOVE, . . .
First two used (and Fujaba) in large industrial applications.

Model Operational Semantics using GG



WARNING: Name must use Python variable syntax

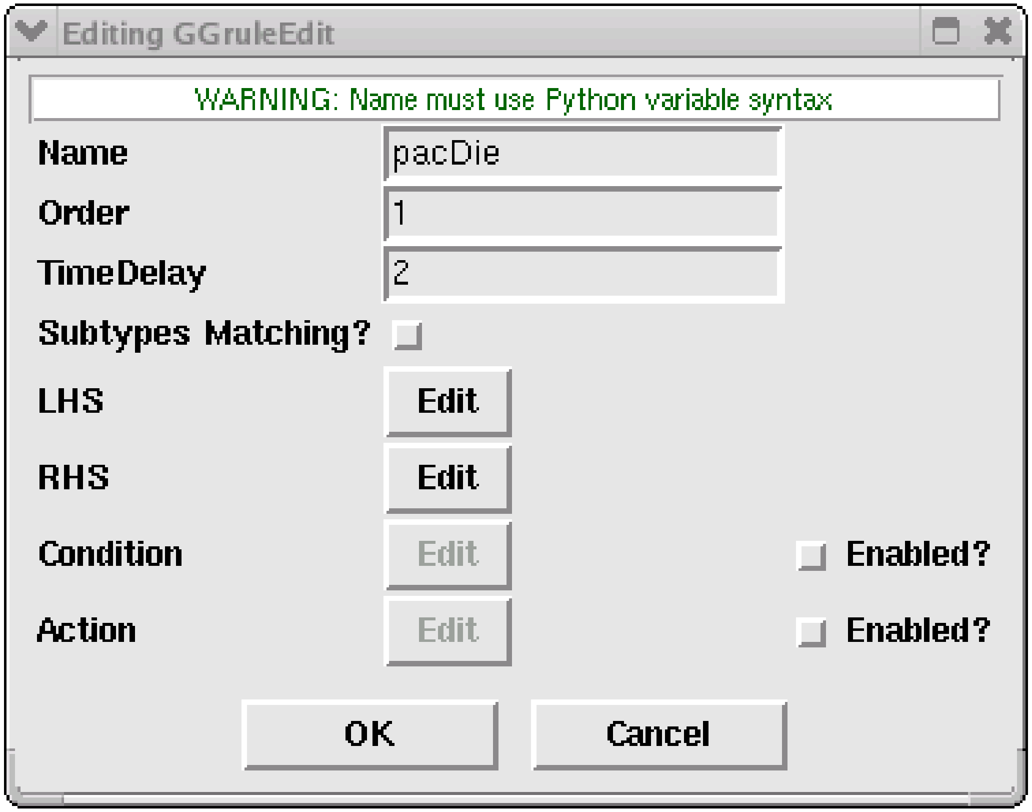
Name: pacGrammar

Rules:
pacDie 1
pacEat 2
pacMoveRight 4
pacMoveUp 4
pacMoveDown 4

InitialAction □ Enabled?

FinalAction □ Enabled?

PacMan Die rule



WARNING: Name must use Python variable syntax

Name: pacDie

Order: 1

TimeDelay: 2

Subtypes Matching? □

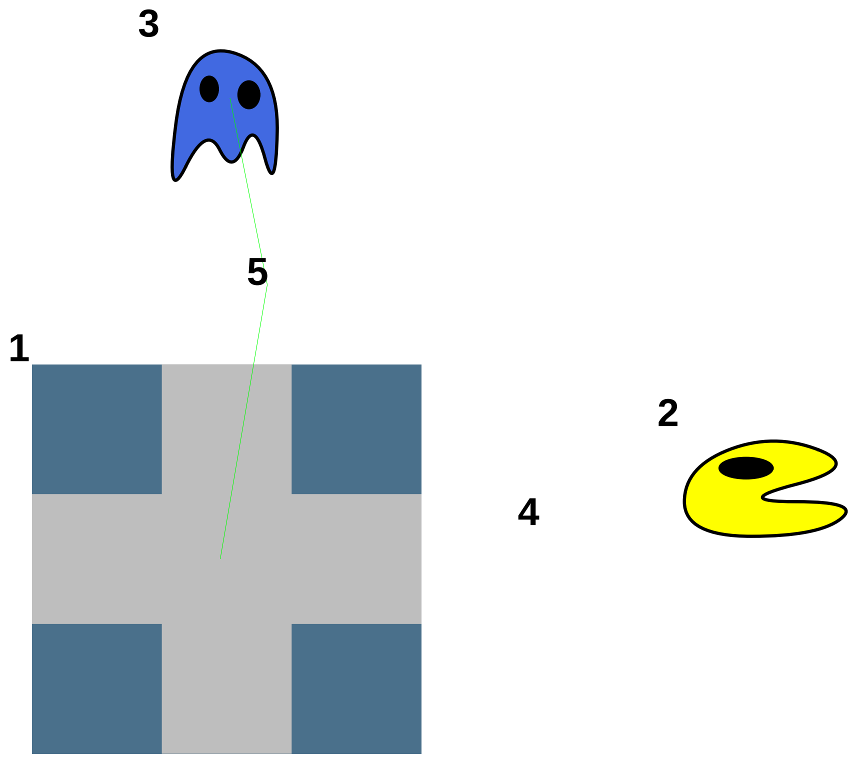
LHS

RHS

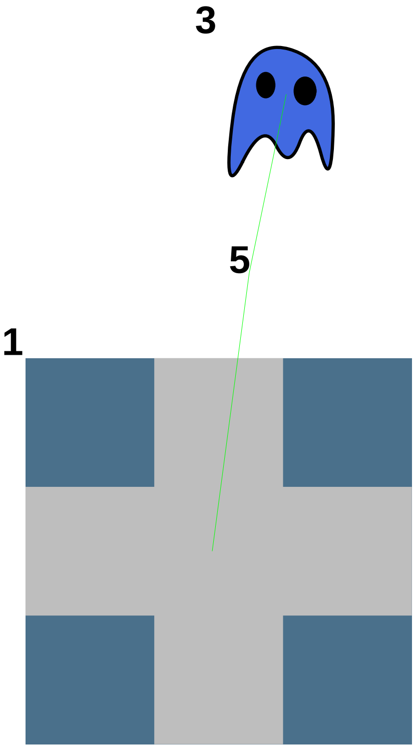
Condition □ Enabled?

Action □ Enabled?

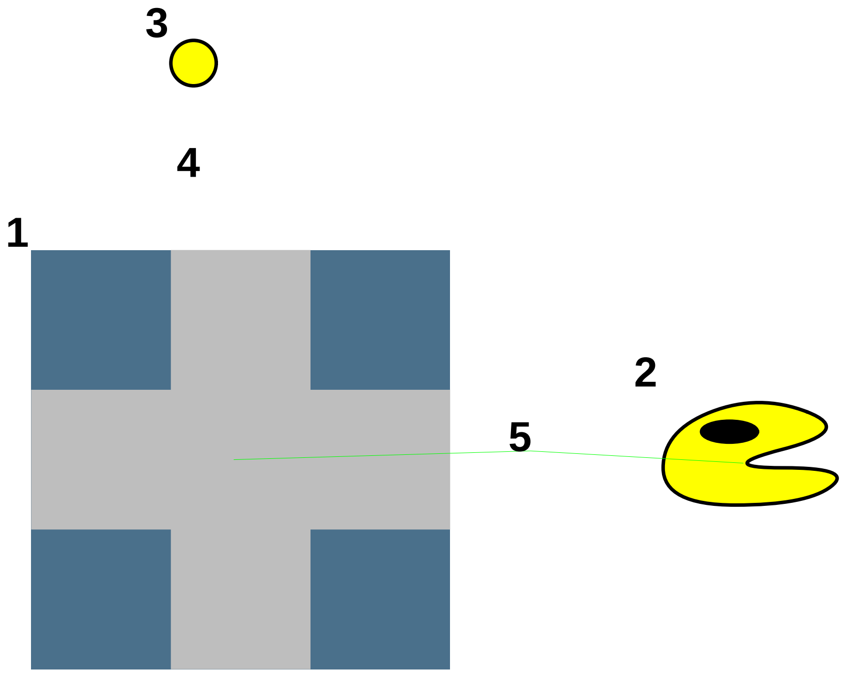
PacMan Die rule LHS



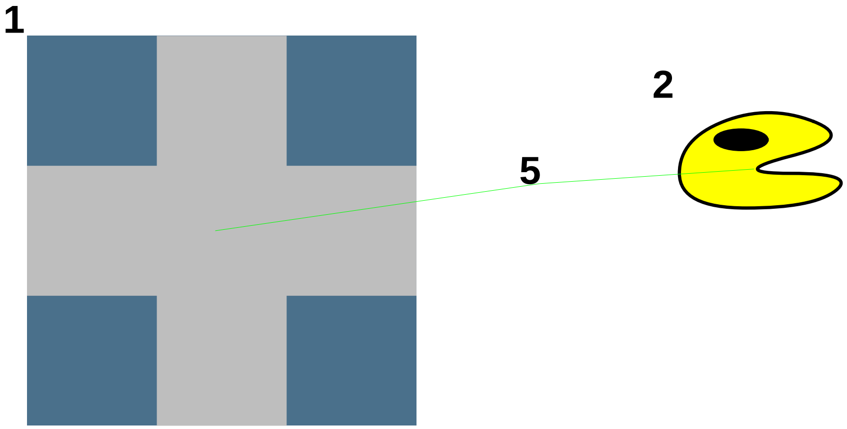
PacMan Die rule RHS



PacMan Eat rule LHS

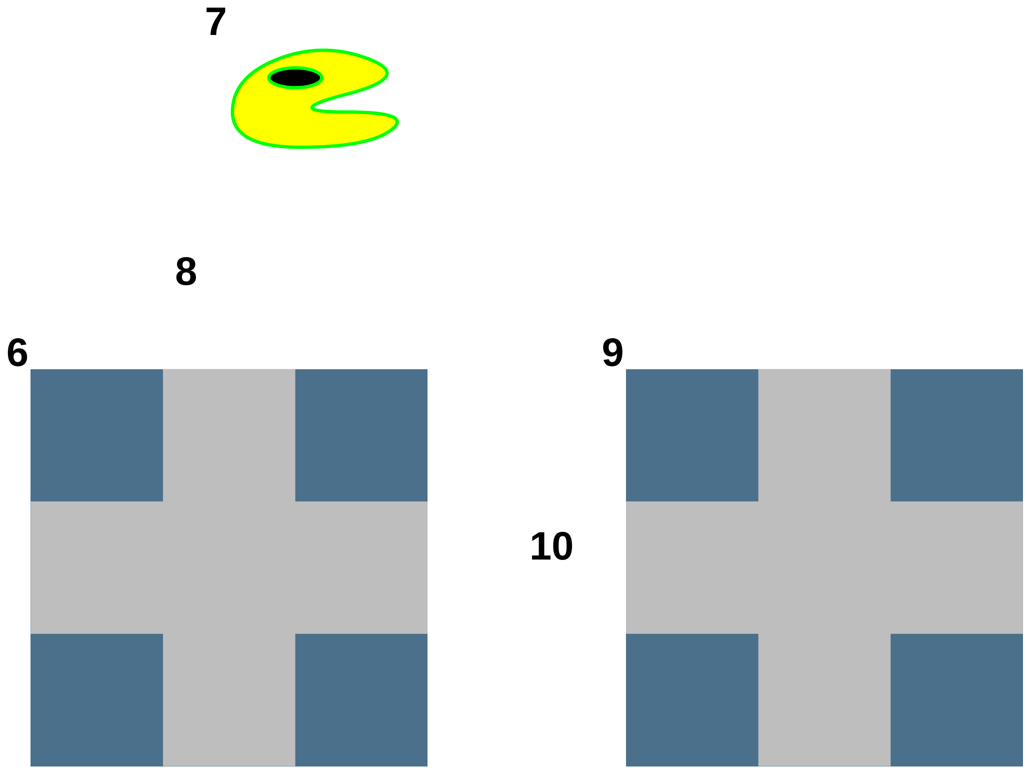


PacMan Eat rule RHS

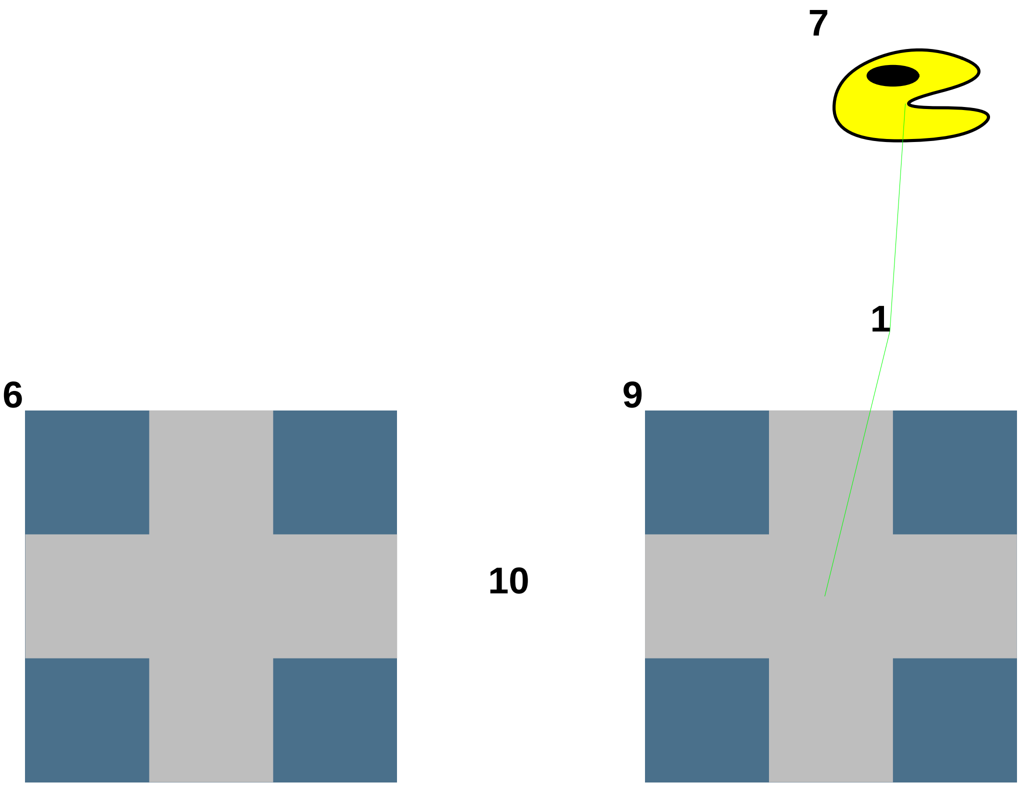


```
scoreBoard = None
scoreBoards = atom3i.ASGroot.listNodes['ScoreBoard']
if (not scoreBoards):
  return
else:
  scoreBoard = scoreBoards[0]
  scoreVal = scoreBoard.score.getValue()
  scoreBoard.score.setValue(scoreVal+1)
  scoreBoard.graphObject_.ModifyAttribute('score',scoreVal+1)
```

PacMan Move rule LHS



PacMan Move rule RHS



Specifying/Executing Transformations using Graph Grammars

(+) Models are often Graph-like

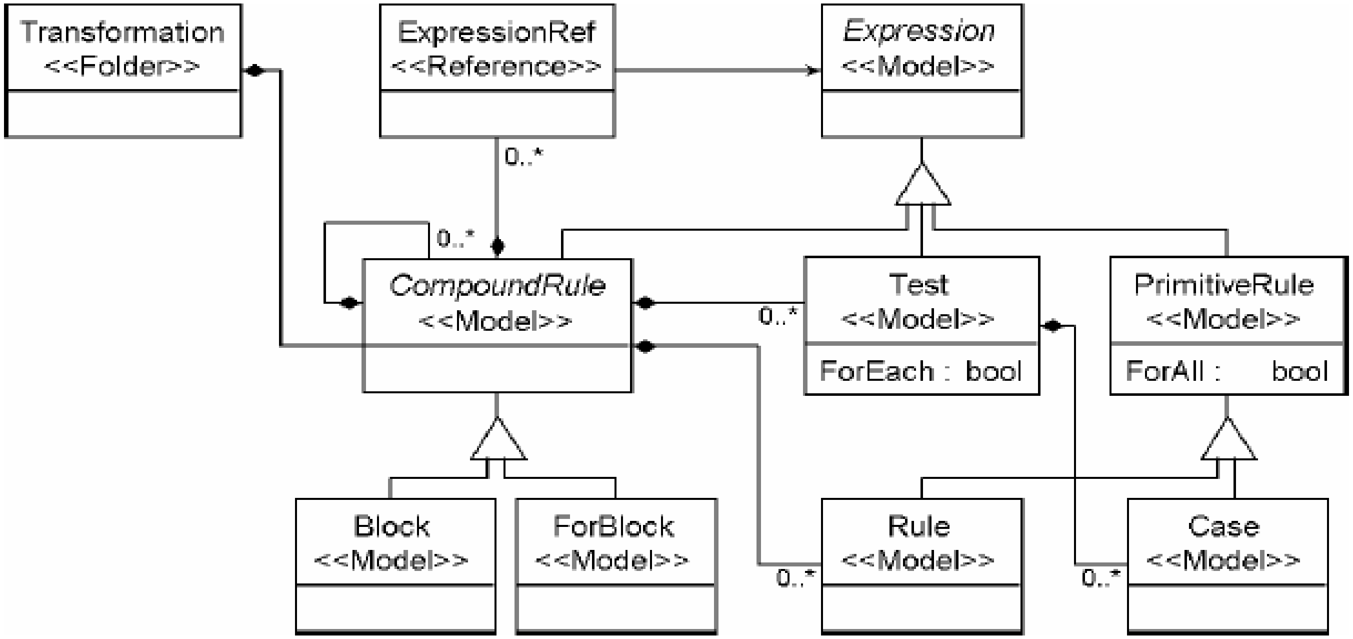
(+) Visual specification (documentation)
For insight/debugging: execution + visual display
For performance: execution on data structures in memory

(+) Little or no programming knowledge required (allows understanding/modification by domain-experts)

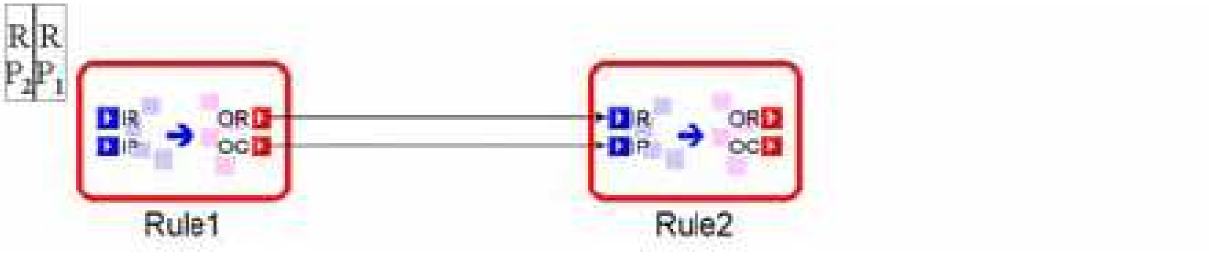
(-) Does it scale up ?
Yes, need to use modular GGs (*e.g.,* GReAT, PROGRES)

(-) Performance is bad ! (due to sub-graph matching)
But sometimes no alternative
– model transformation for graph-like models
– don't want to code matching yourself
But give (domain-specific) hints to kernel (Marc Provost's thesis)
But use as specification for manual implementation
– executable specification = reference implementation
– automatic generation of unit tests

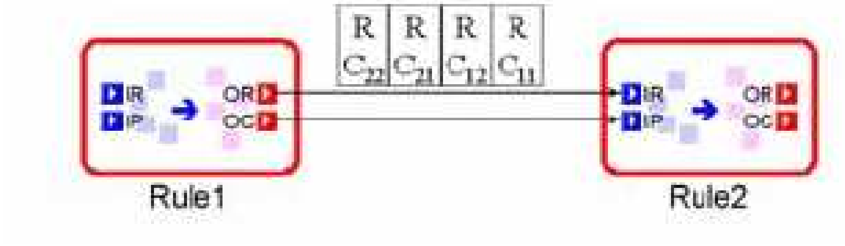
Modular Graph Rewriting: Control Structures



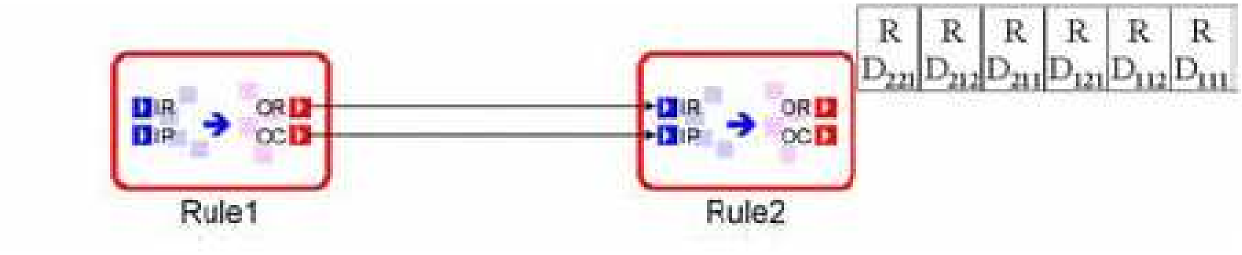
GReAT Control Structures: Sequence



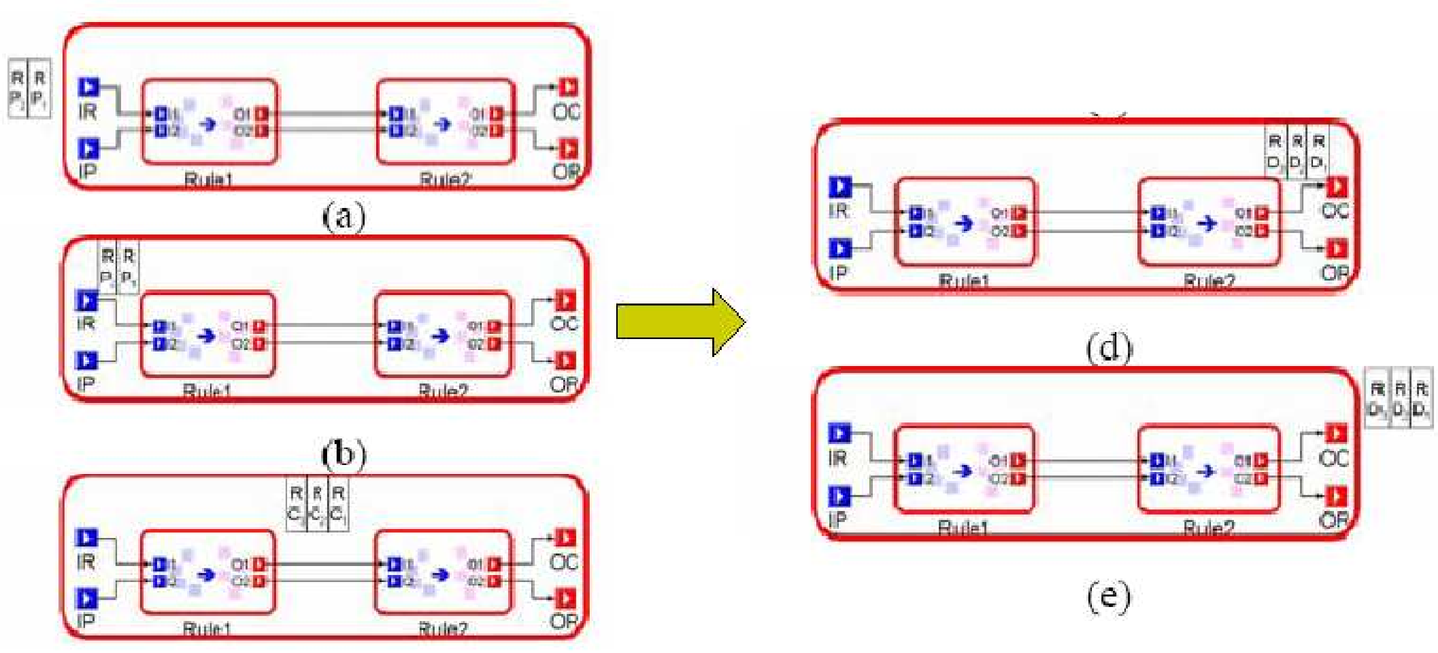
(a)



(b)



GReAT Control Structures: Nesting



(a)

(b)

(c)

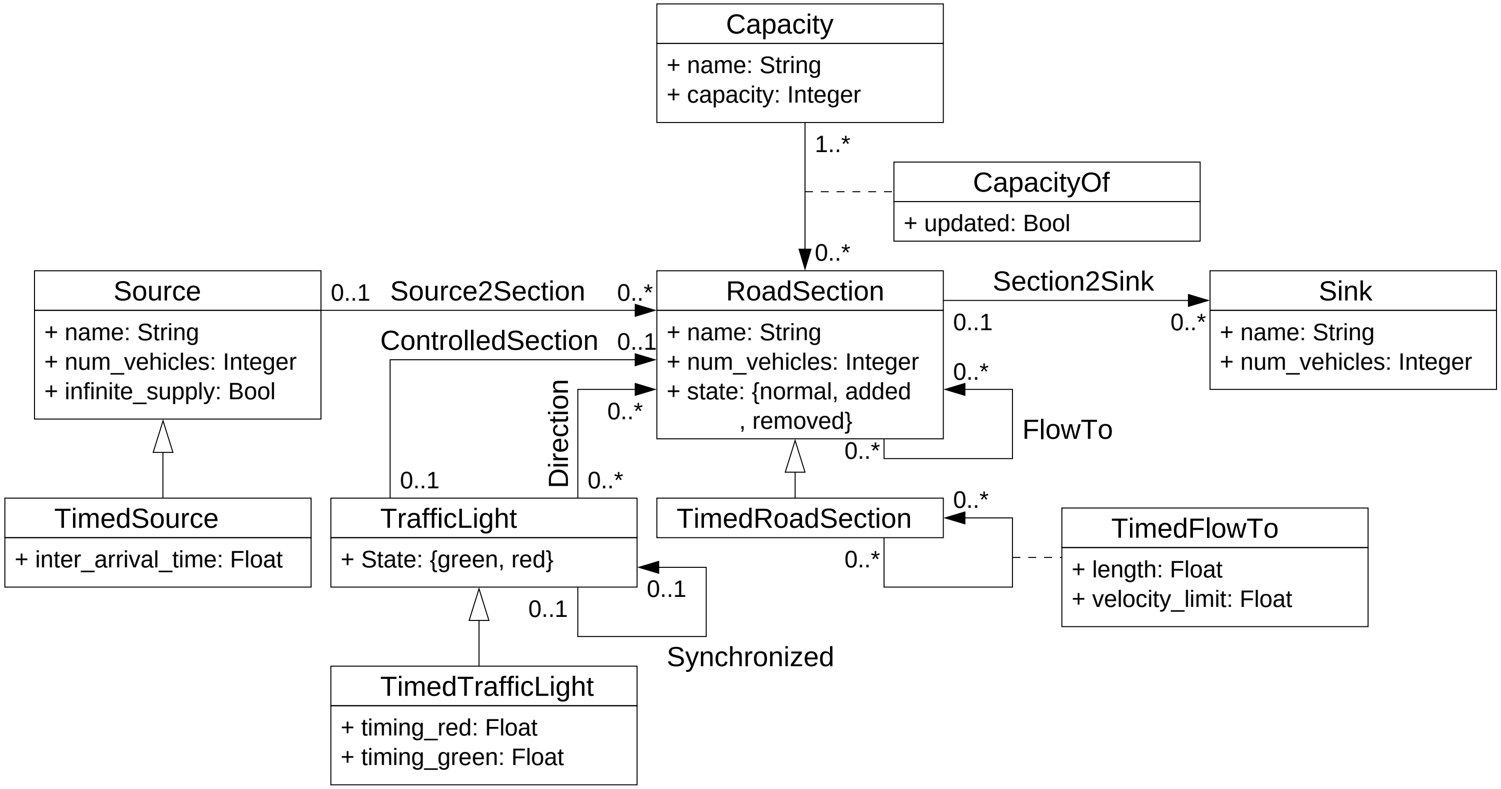
(d)

(e)

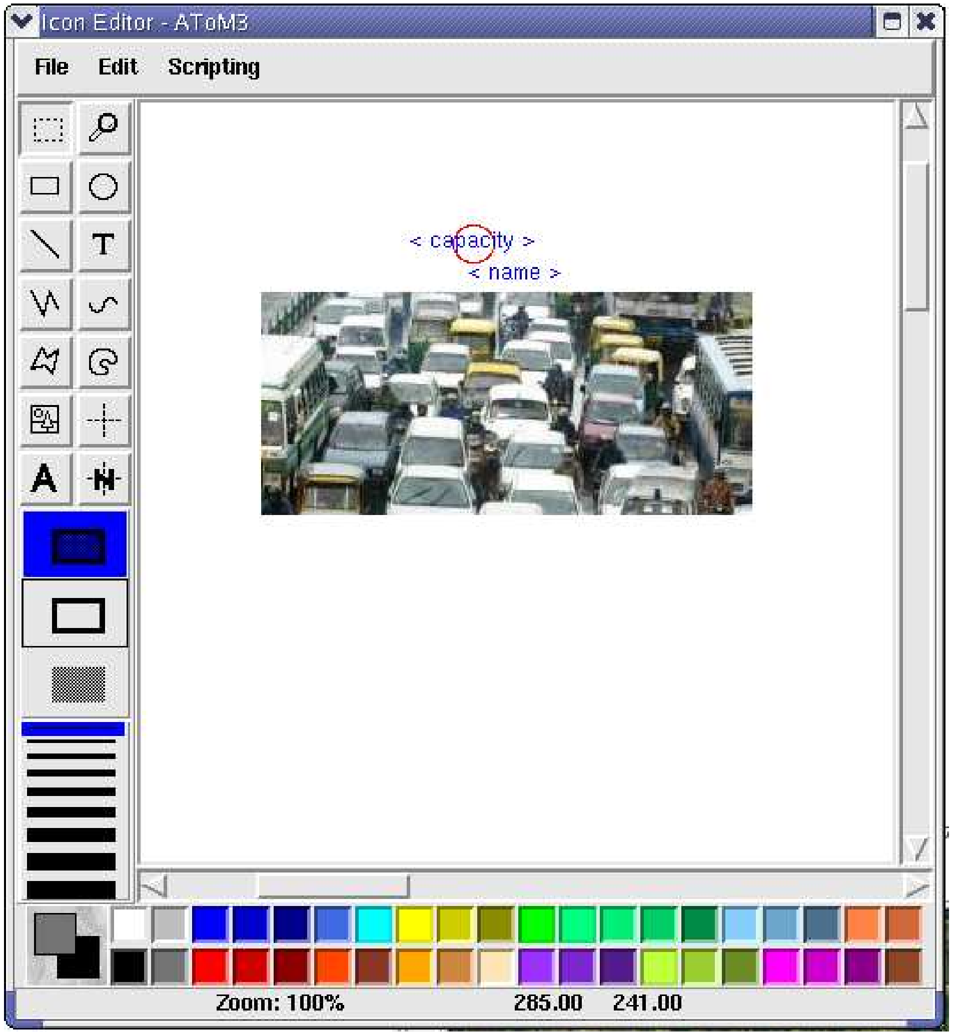
Current work:
use DEVS as control framework

Formalism Transformation Example: Model/Analyze/Simulate Traffic Networks

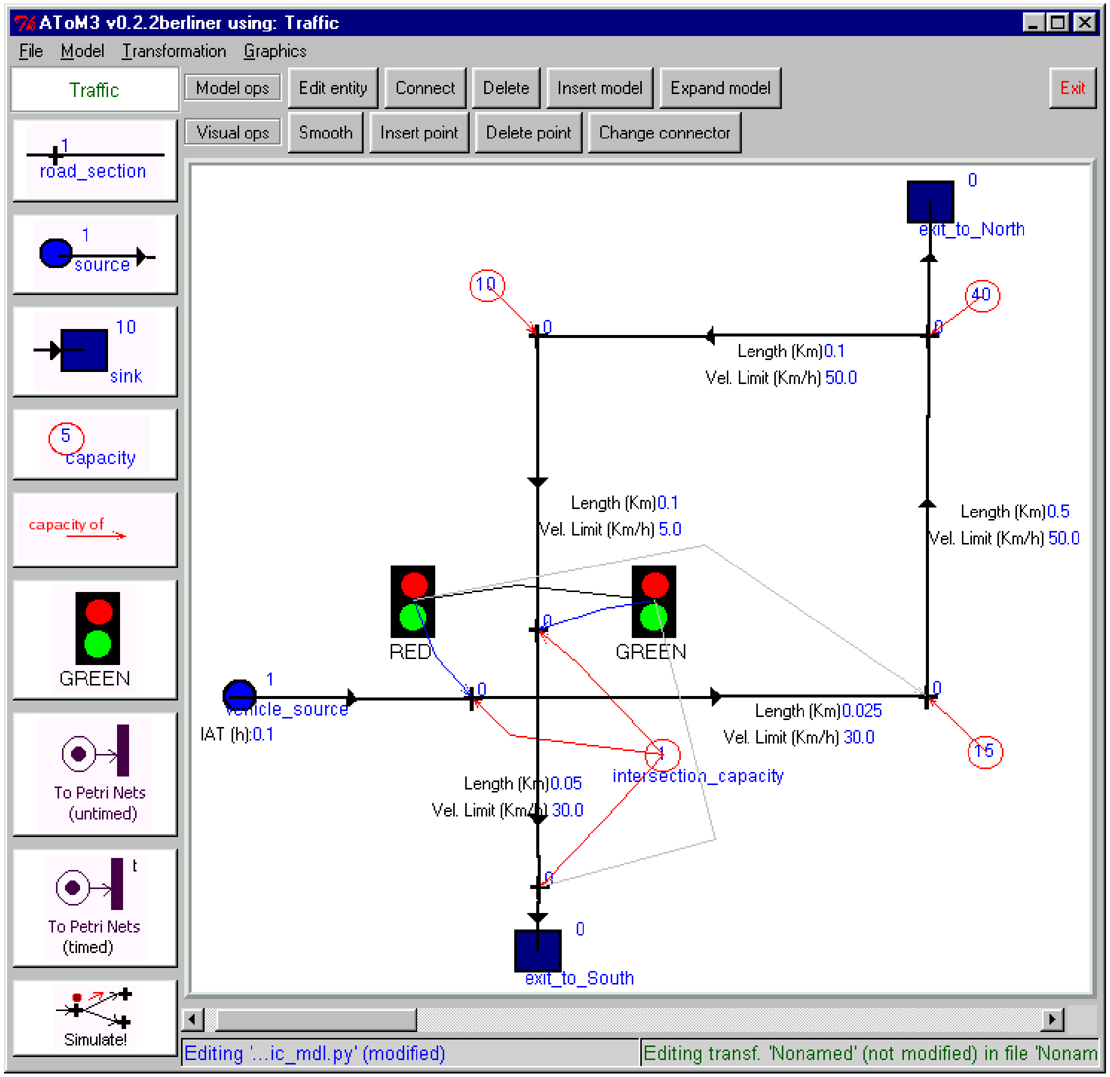
Un-timed and timed Traffic meta-model (a UML Class Diagram)



Traffic Concrete Syntax (the Capacity Entity)



Synthesized Traffic
Visual Modelling Environment



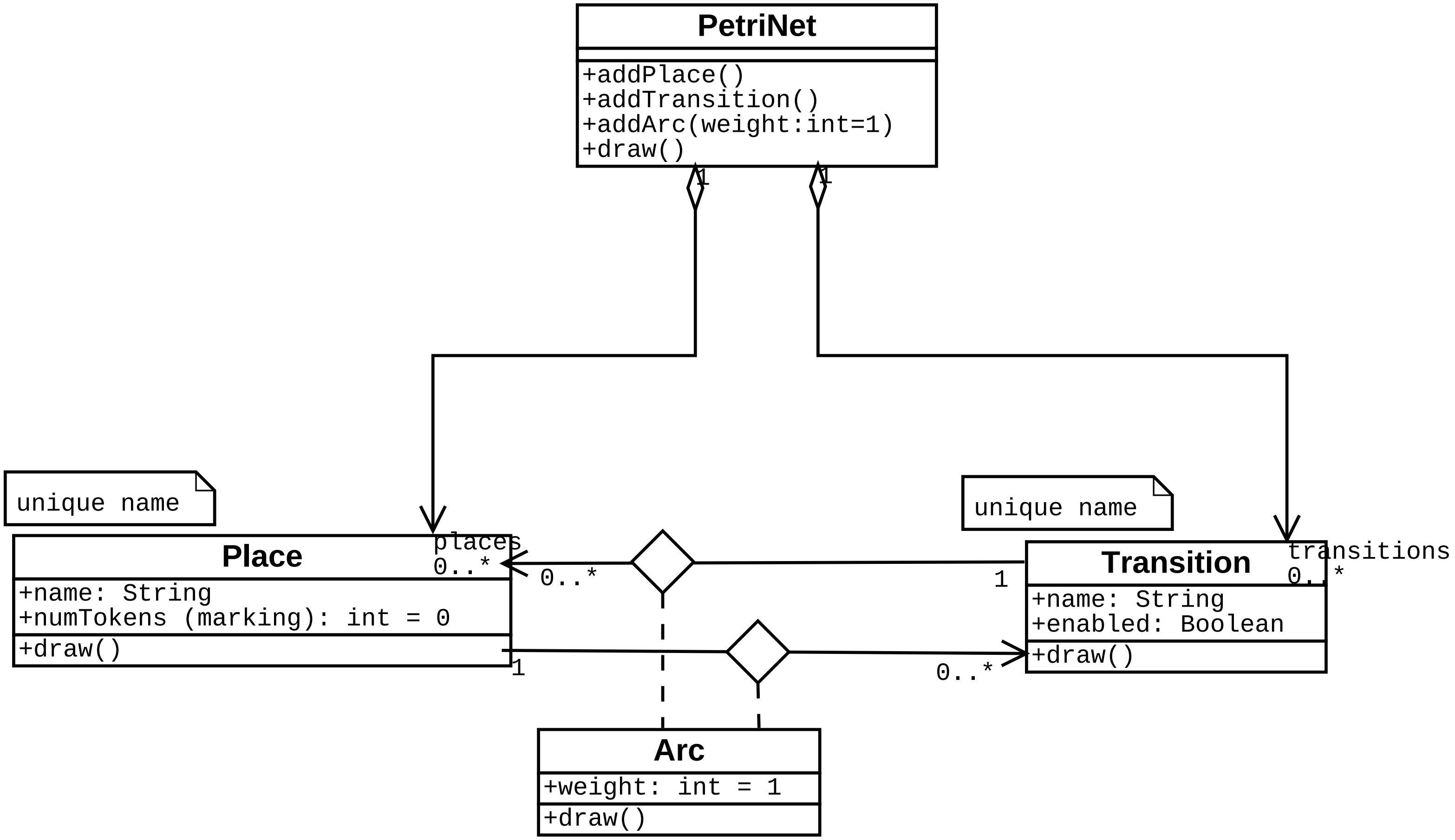
Editing '...ic_mdl.py' (modified)

Editing transf. 'Nonamed' (not modified) in file 'Nonam

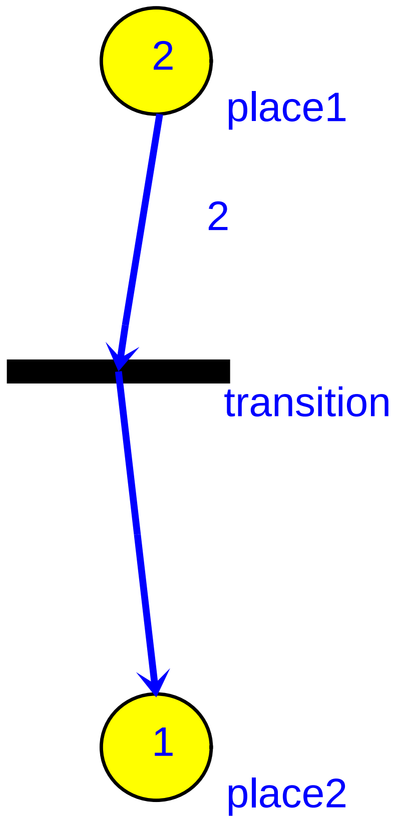
Modelling Traffic's Semantics

- choices: timed, un-timed, . . . (level of abstraction)
- **denotational**: map onto known formalism (TTPN, PN)
  . . . good for analysis purposes
- **operational**: procedure to execute/simulate model
  . . . may act as a reference implementation
- note: need to **prove** consistency between denotational and operational semantics if both are given !

Place-Transition Petri Net Abstract Syntax (UML Class Diagram formalism)



Place-Transition Petri Net Concrete Syntax



Petri Net Behaviour

State Transition Function f of marked Petri net (P, T, A, w, x_0)

$$f : \mathbb{N}^n \times T \rightarrow \mathbb{N}^n$$

is defined for transition $t_j \in T$ if and only if

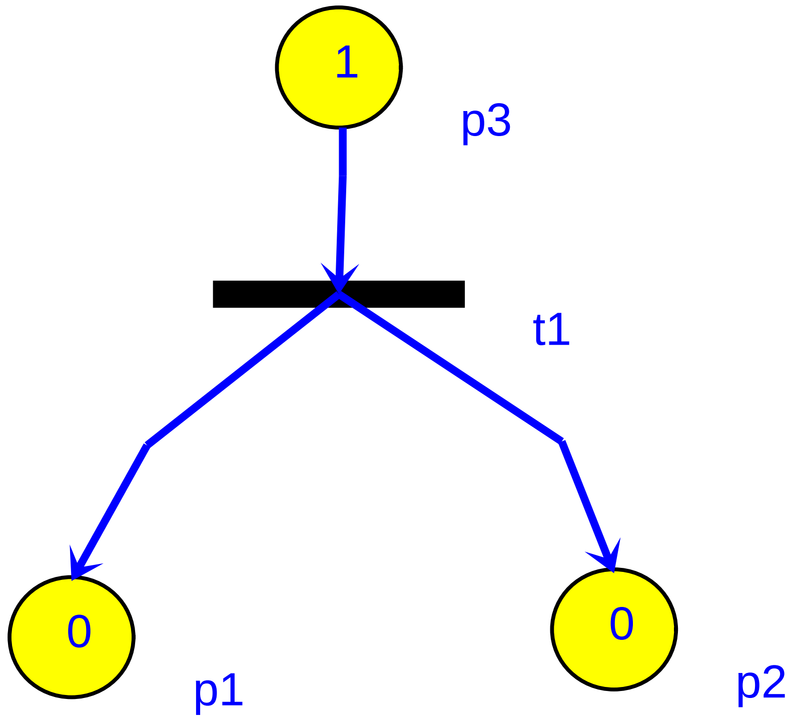
$$x(p_i) \geq w(p_i, t_j), \forall p_i \in I(t_j)$$

If $f(\mathbf{x}, t_j)$ is defined, set $\mathbf{x}' = f(\mathbf{x}, t_j)$ where

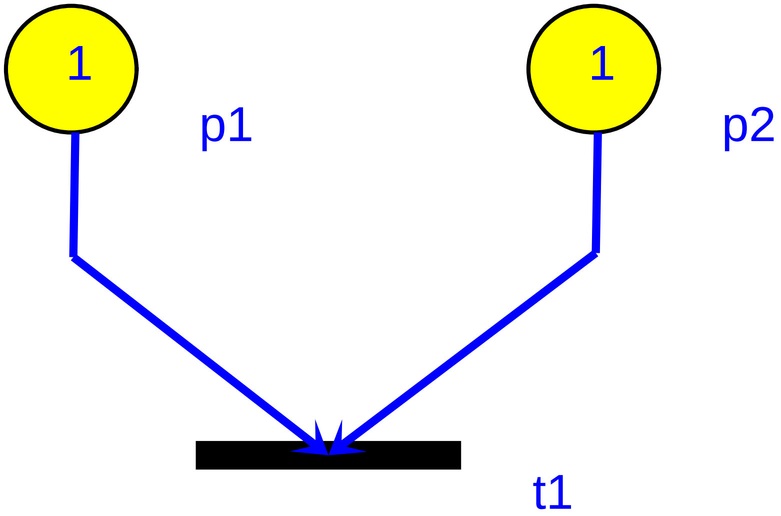
$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i)$$

- State transition function f based on *structure* of Petri net
- Number of tokens *need not be conserved* (but can)

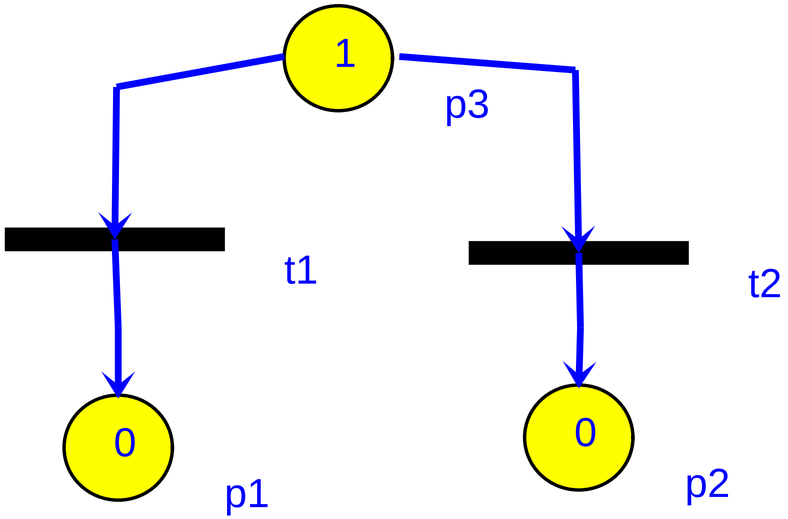
Behaviour: Fork



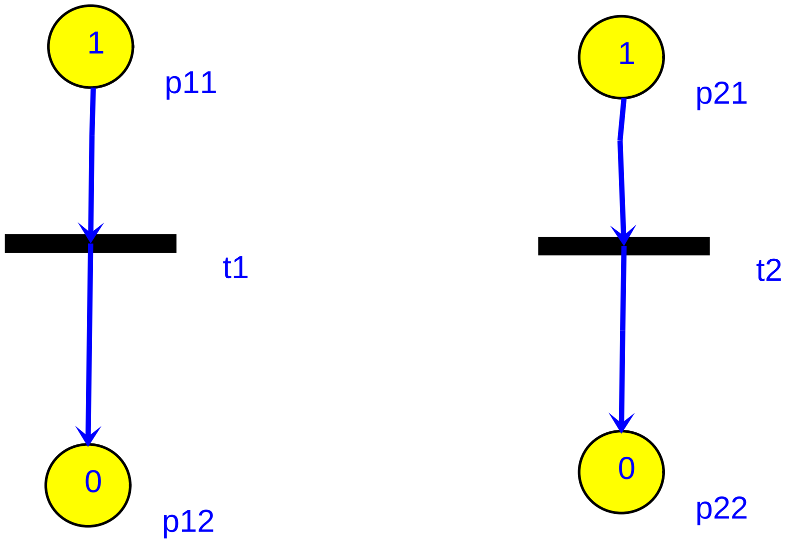
Behaviour: Join



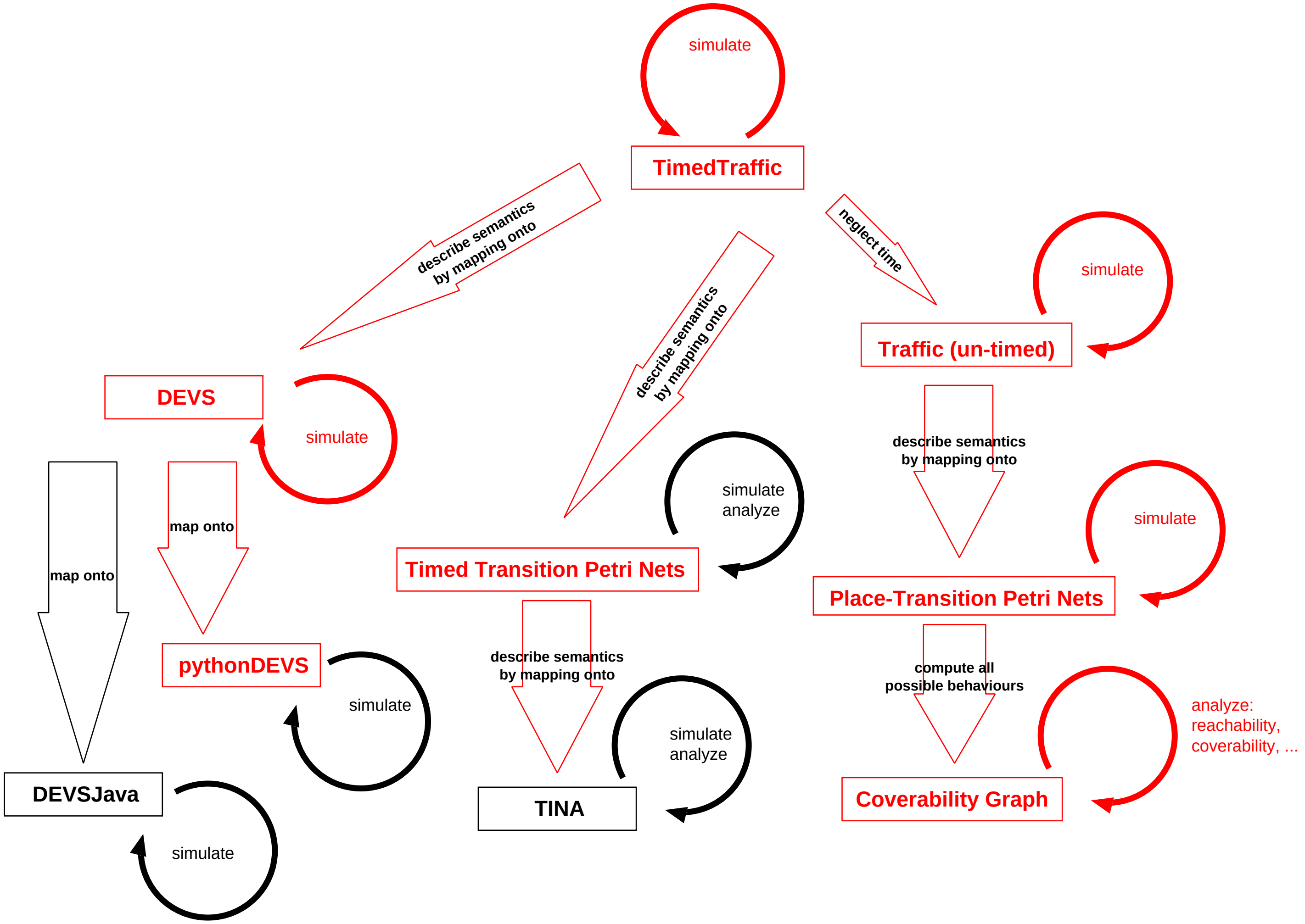
Behaviour: Conflict, choice, decision



Behaviour: Concurrency



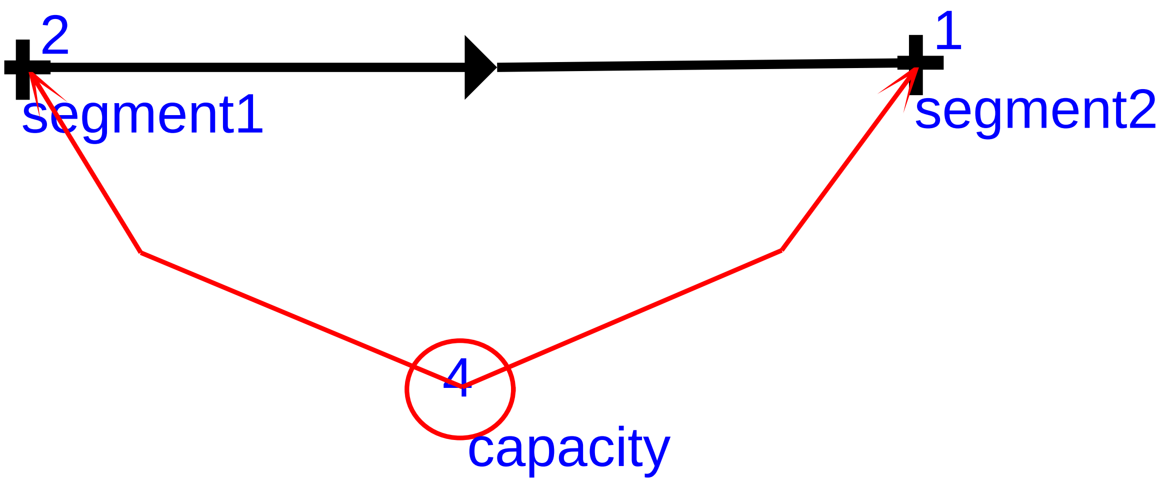
The Big Picture: Transformations



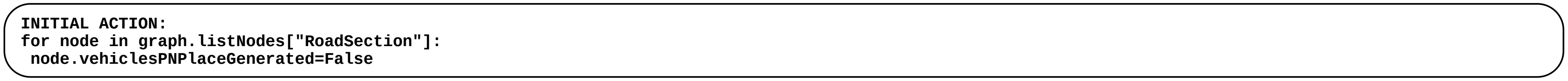
Traffic's (un-timed) semantics in terms of Petri Nets

- need a meta-model of **Traffic** (shown before)
- need a meta-model of **Petri Net**s (shown before)
- need a meta-model of **Generic Graph** (glue)
- need a model of the mapping: **Traffic** ⇒ **Petri Net**

A very simple Traffic model

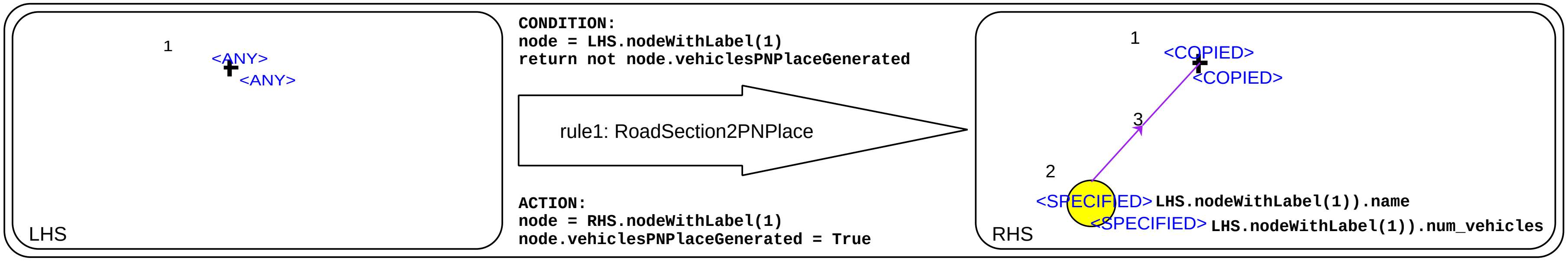


Traffic to Petri Net Graph Grammar rules



```
INITIAL ACTION:
for node in graph.listNodes["RoadSection"]:
 node.vehiclesPNPlaceGenerated=False
```

Traffic to Petri Net Graph Grammar rules

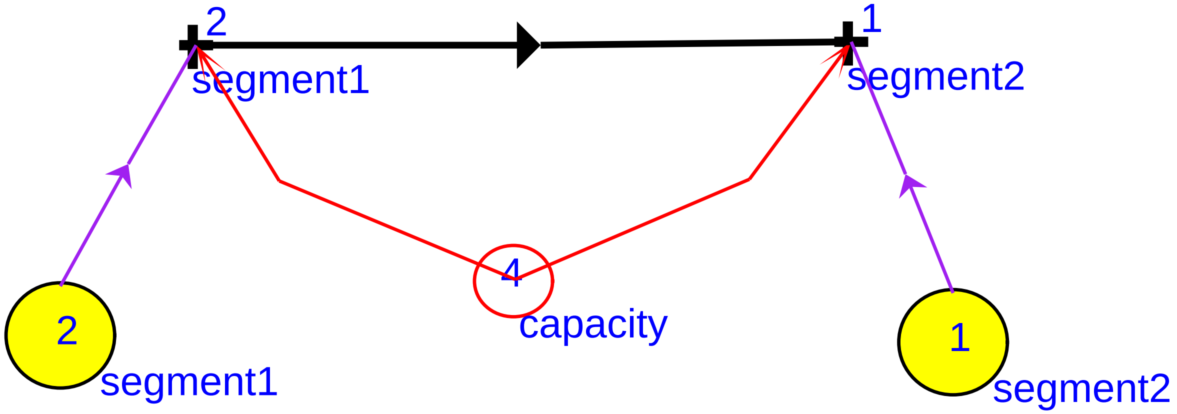


```
CONDITION:
node = LHS.nodeWithLabel(1)
return not node.vehiclesPNPlaceGenerated
```

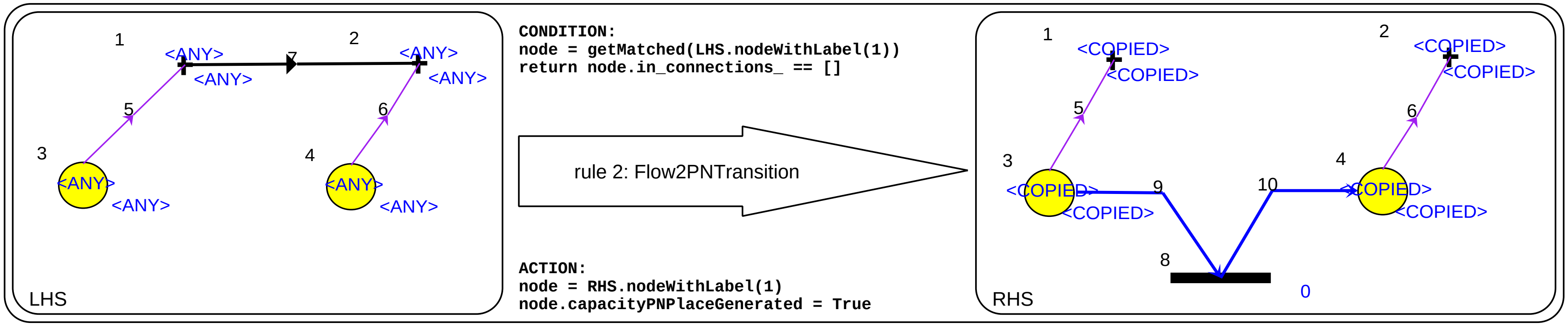
```
ACTION:
node = RHS.nodeWithLabel(1)
node.vehiclesPNPlaceGenerated = True
```

LHS
1
<ANY>
<ANY>
rule1: RoadSection2PNPlace
RHS
1
<COPIED>
<COPIED>
3
2
<SPECIFIED> LHS.nodeWithLabel(1)).name
<SPECIFIED> LHS.nodeWithLabel(1)).num_vehicles

Road Sections converted to Petri Net Places



Traffic to Petri Net Graph Grammar rules

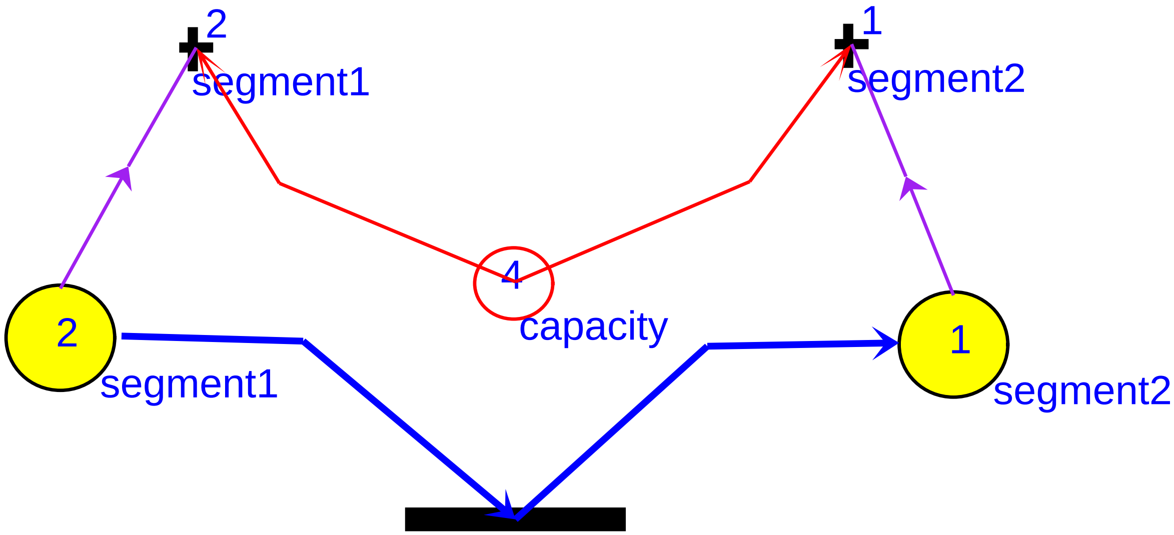


```
CONDITION:
node = getMatched(LHS.nodeWithLabel(1))
return node.in_connections_ == []
```

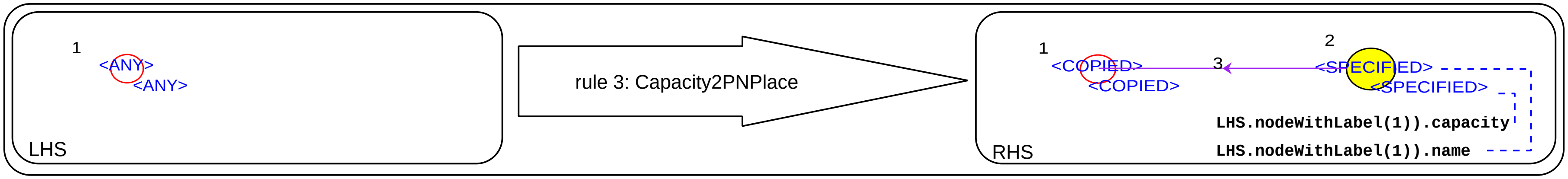
rule 2: Flow2PNTransition

```
ACTION:
node = RHS.nodeWithLabel(1)
node.capacityPNPlaceGenerated = True
```

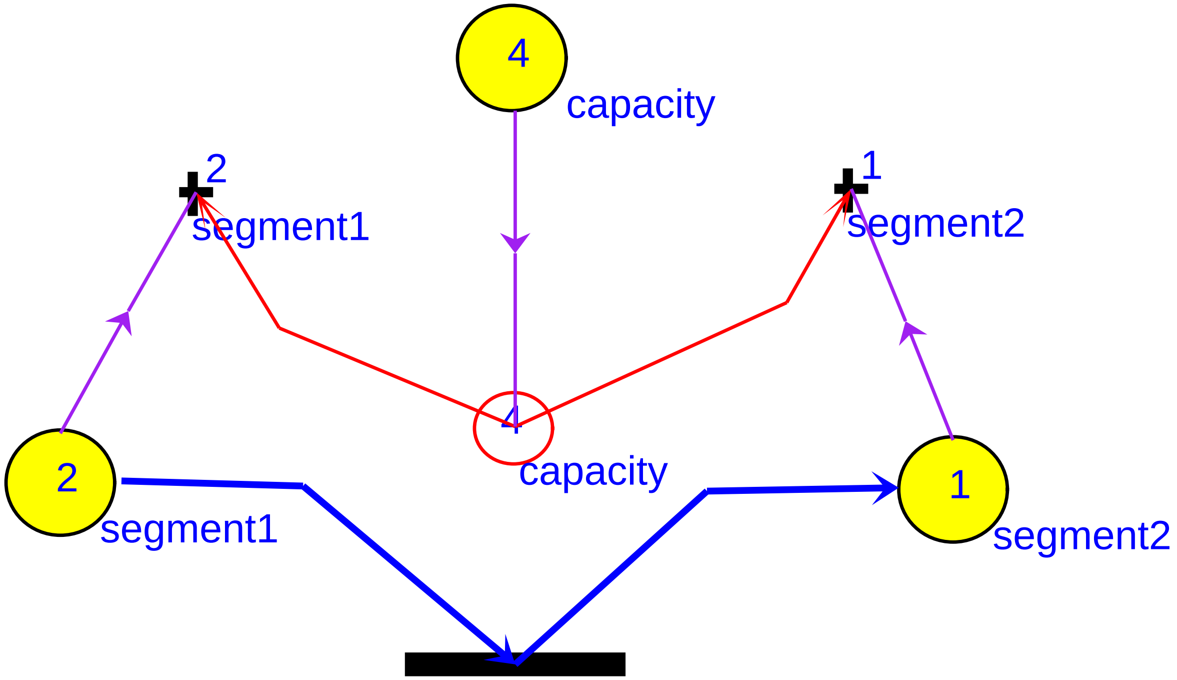
Traffic Flow to Petri Net Transitions



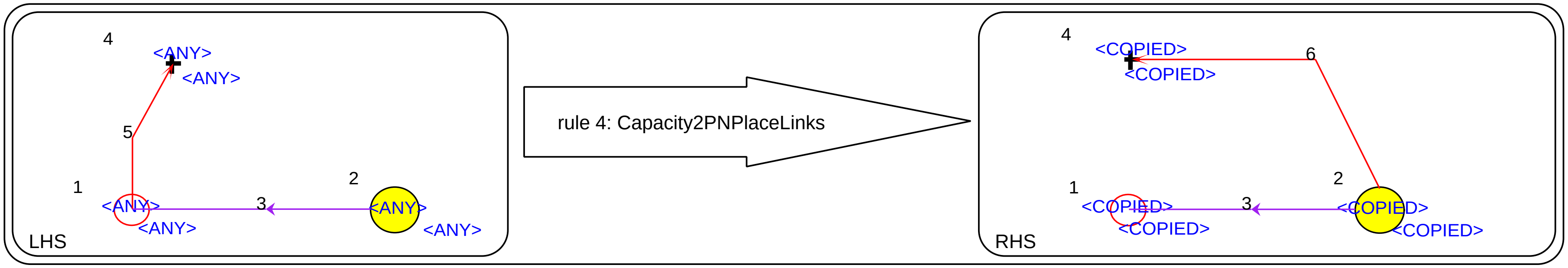
Traffic to Petri Net Graph Grammar rules



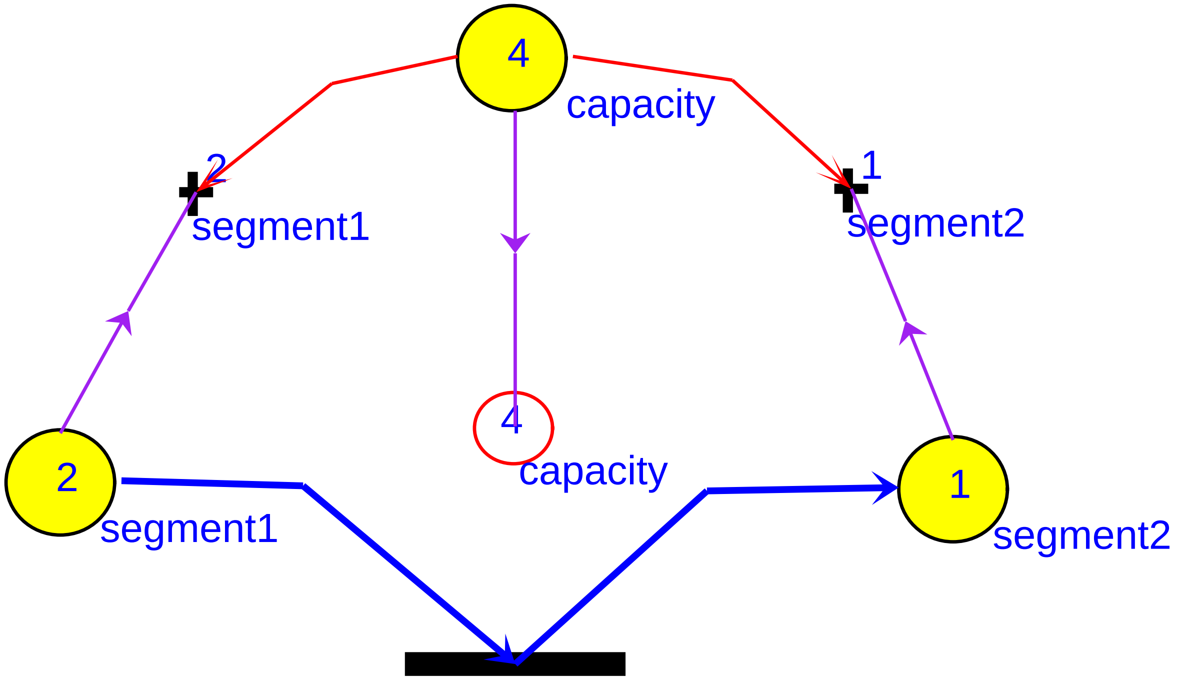
Traffic Capacity to Petri Net Place



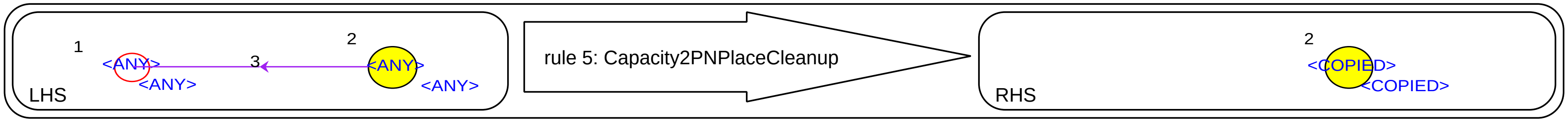
Traffic to Petri Net Graph Grammar rules



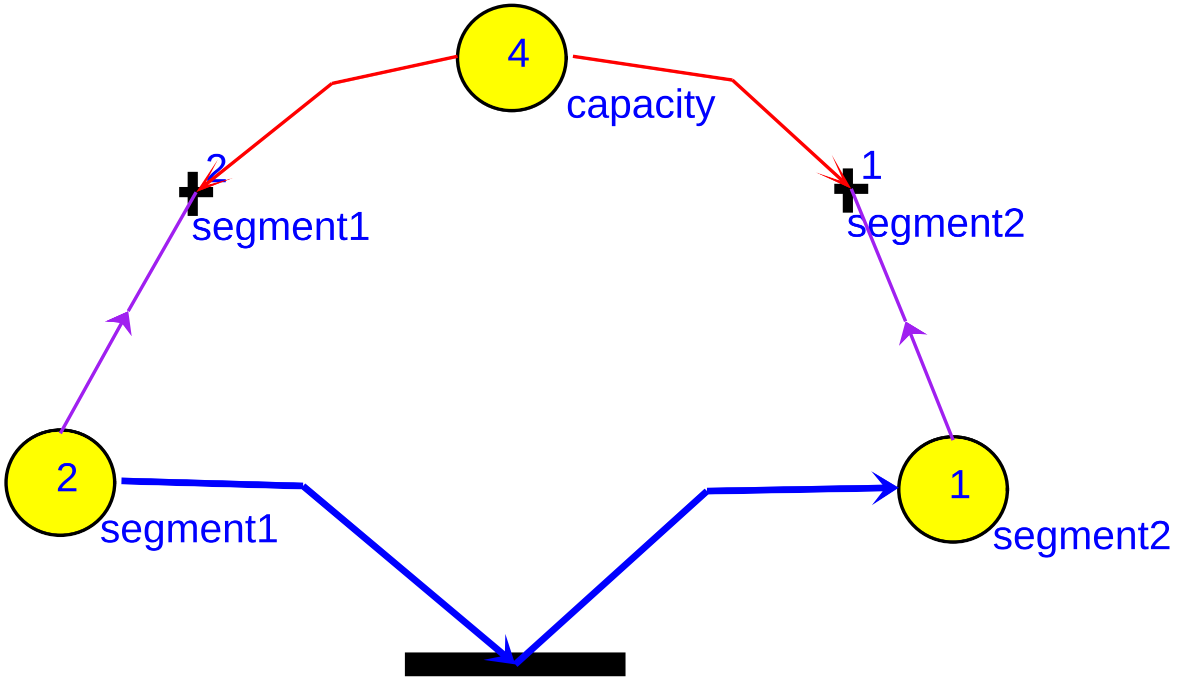
Traffic Capacity to Petri Net Place (links)



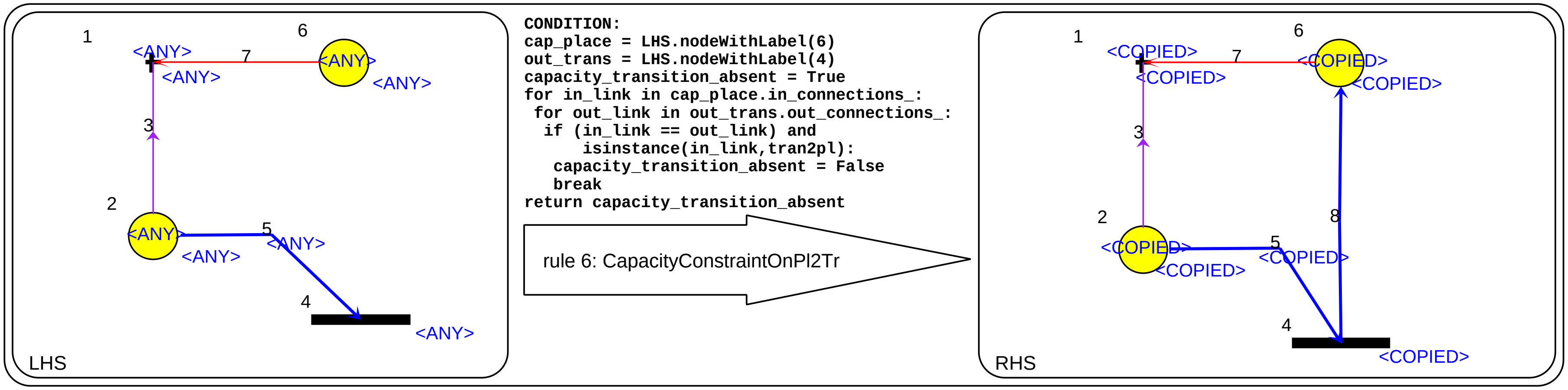
Traffic to Petri Net Graph Grammar rules



Traffic Capacity to Petri Net Place cleanup

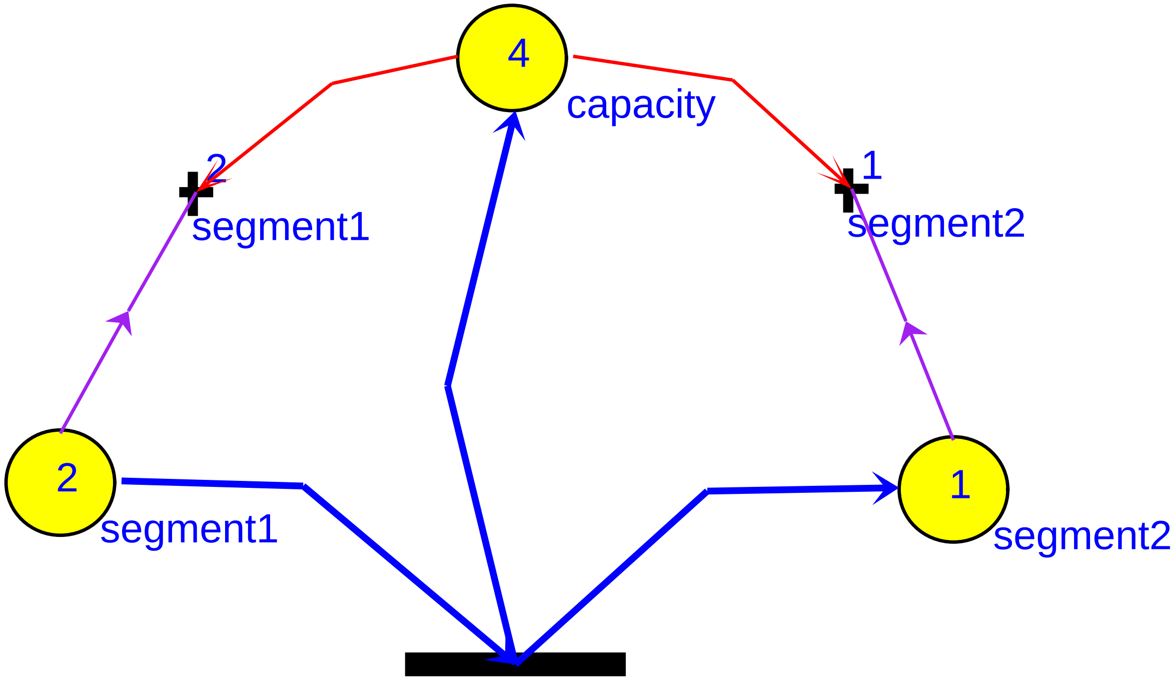


Traffic to Petri Net Graph Grammar rules

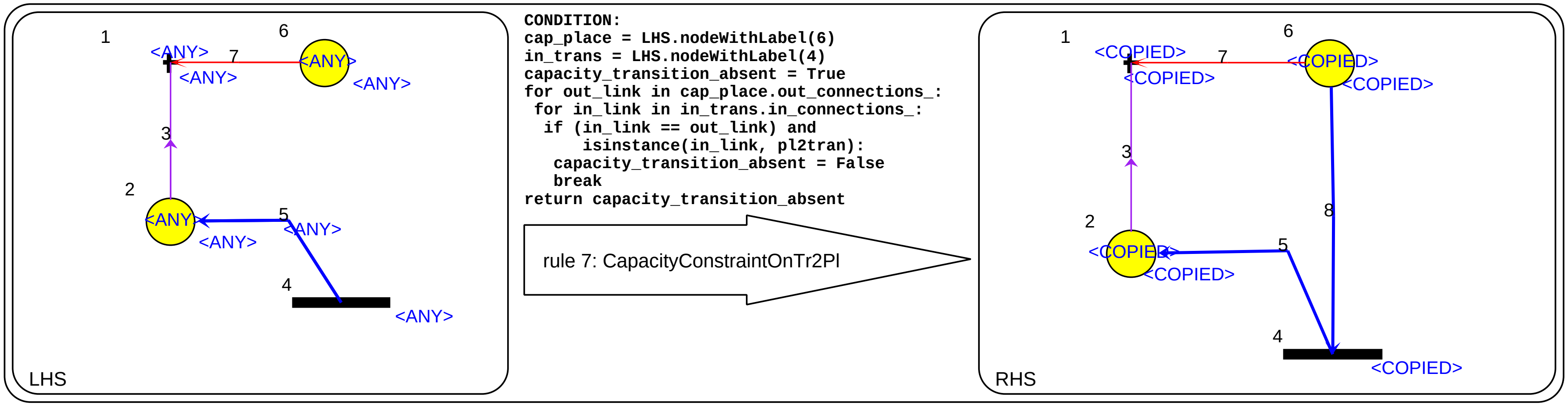


```
CONDITION:
cap_place = LHS.nodeWithLabel(6)
out_trans = LHS.nodeWithLabel(4)
capacity_transition_absent = True
for in_link in cap_place.in_connections_:
 for out_link in out_trans.out_connections_:
  if (in_link == out_link) and
      isinstance(in_link,tran2pl):
   capacity_transition_absent = False
   break
return capacity_transition_absent
```

Capacity Constraint on Place to Transition

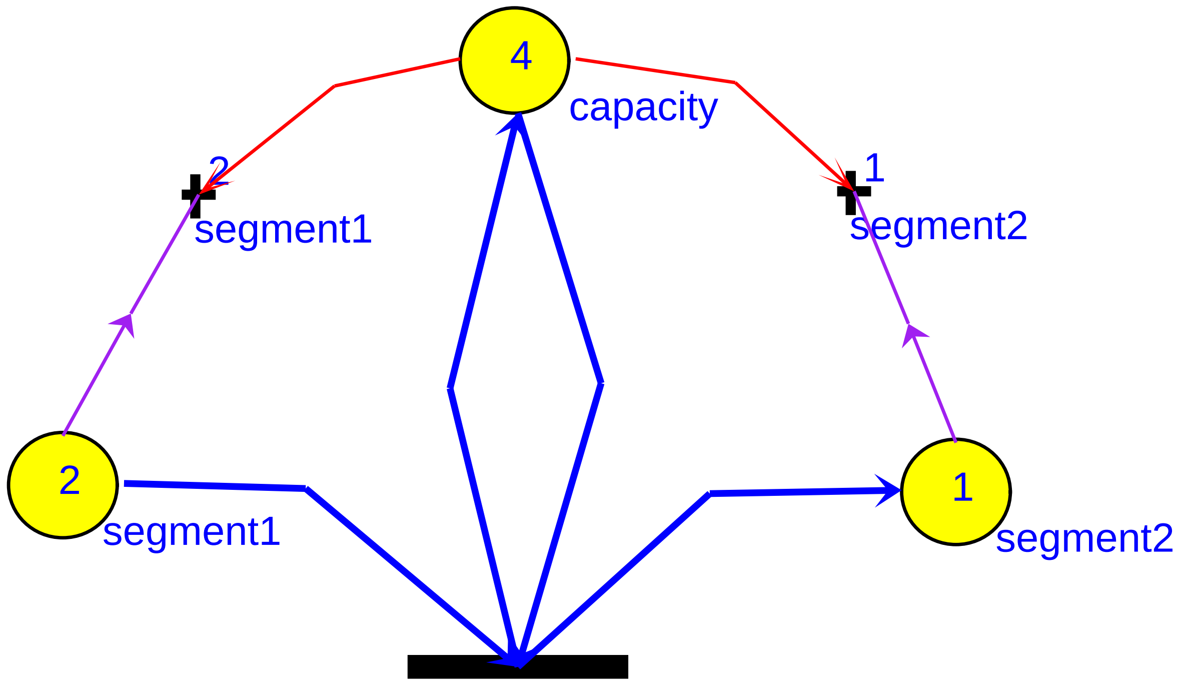


Traffic to Petri Net Graph Grammar rules

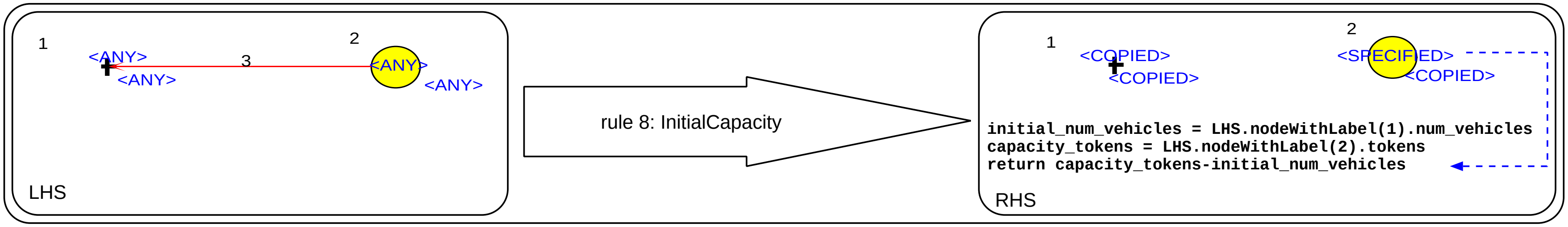


```
CONDITION:
cap_place = LHS.nodeWithLabel(6)
in_trans = LHS.nodeWithLabel(4)
capacity_transition_absent = True
for out_link in cap_place.out_connections_:
 for in_link in in_trans.in_connections_:
  if (in_link == out_link) and
      isinstance(in_link, pl2tran):
   capacity_transition_absent = False
   break
return capacity_transition_absent
```

Capacity Constraint on Transition to Place

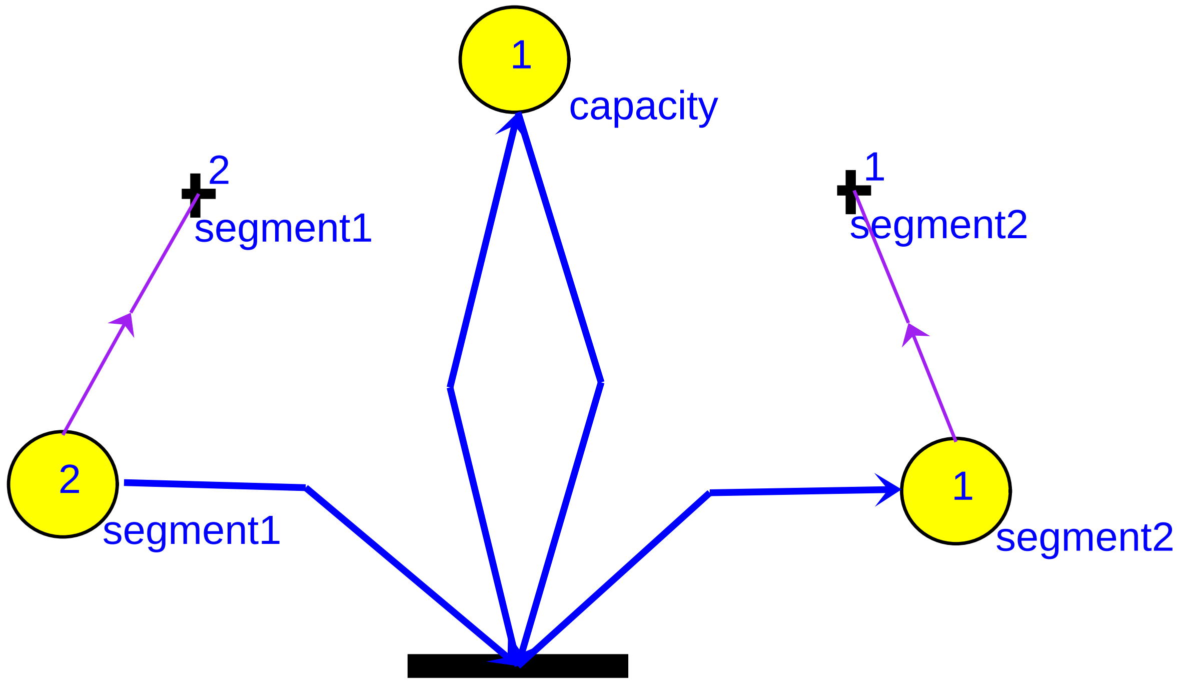


Traffic to Petri Net Graph Grammar rules

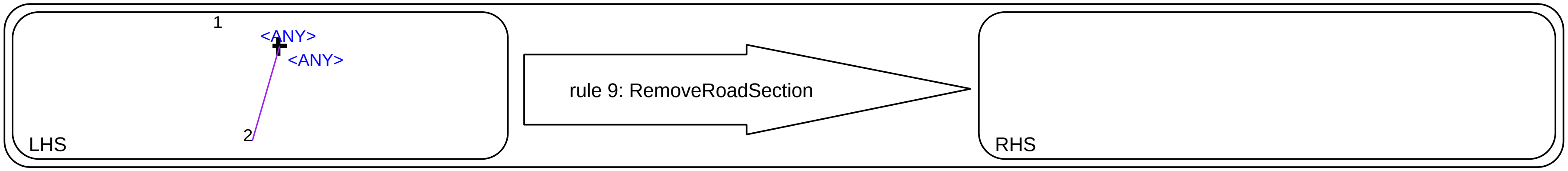


```
initial_num_vehicles = LHS.nodeWithLabel(1).num_vehicles
capacity_tokens = LHS.nodeWithLabel(2).tokens
return capacity_tokens-initial_num_vehicles
```

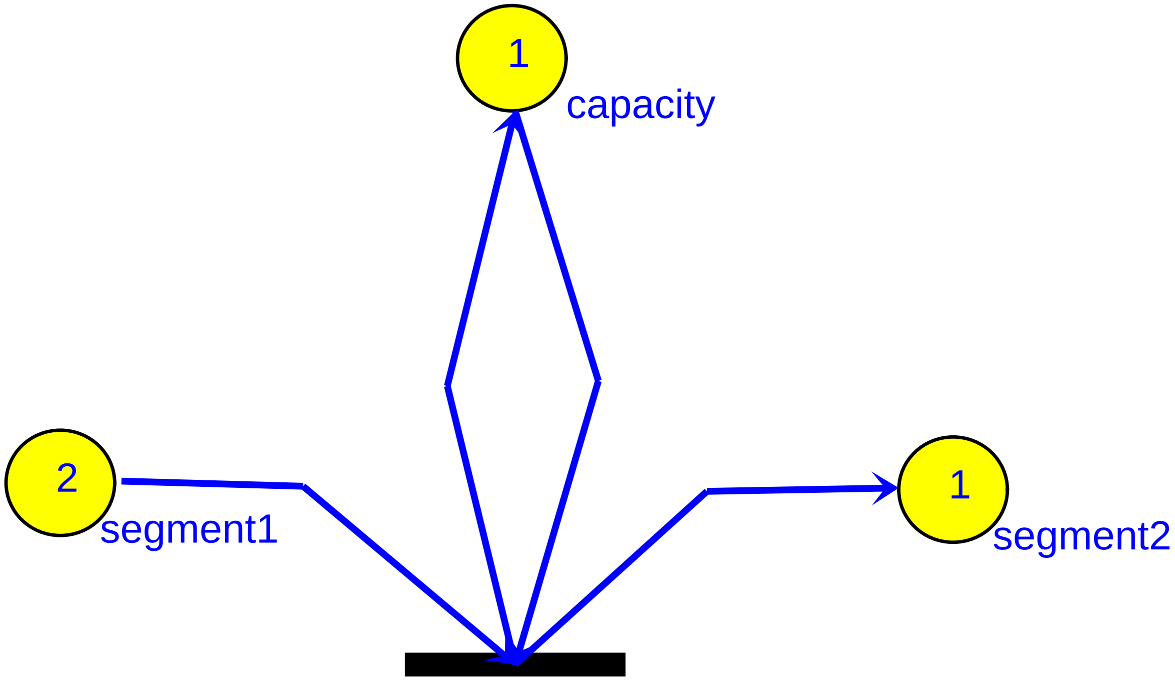
Model Initial Capacity (applied rule twice)



Traffic to Petri Net Graph Grammar rules



Removed Traffic Road Section, now only Petri Net



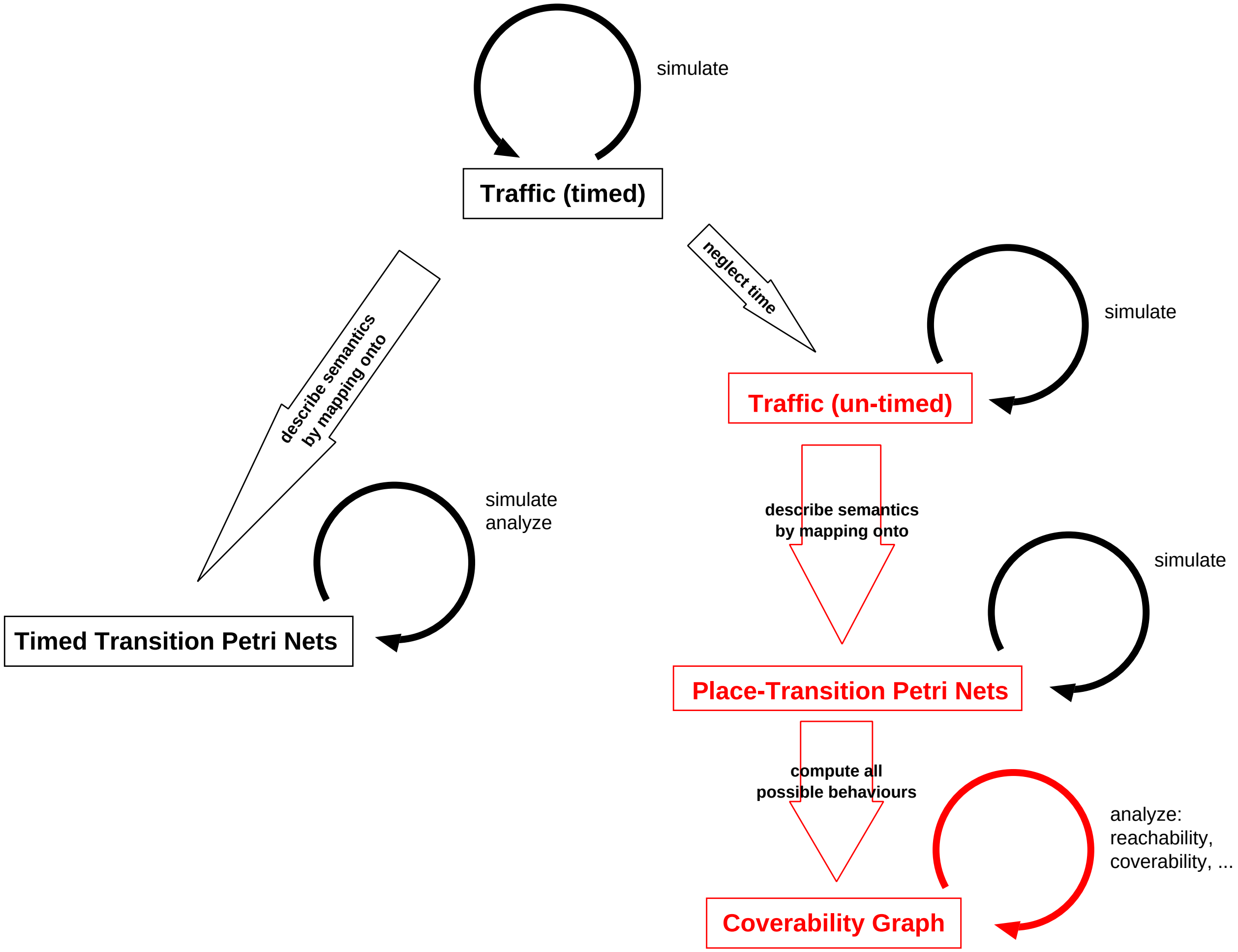
Static Analysis of the Transformation Model

The transformation specified by the Graph Grammar model must satisfy the following requirements:

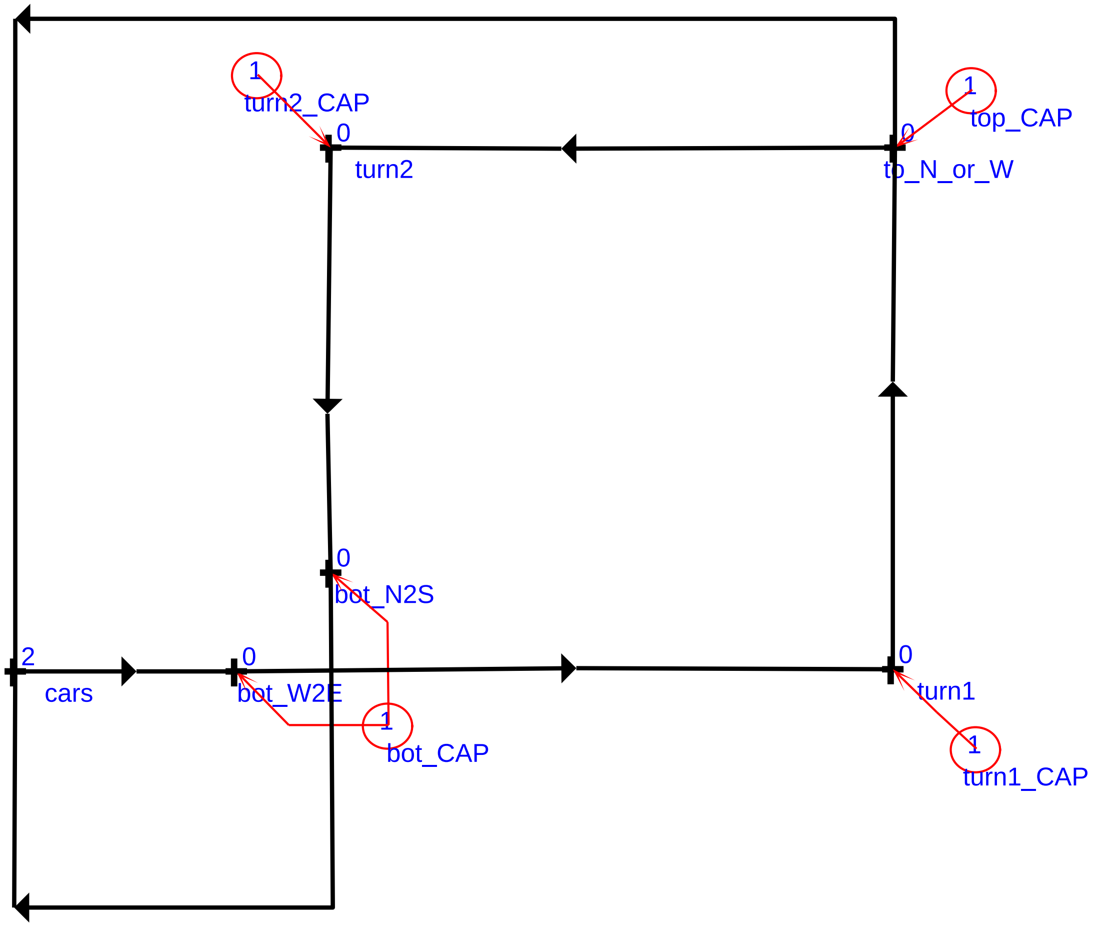
- **Termination:**
  the transformation process is *finite*
- **Convergence/Uniqueness:**
  the transformation results in a *single* target model
- **Syntactic Consistency:**
  the target model must be *exclusively* in the target formalism

These properties can often (but not always)
be **statically** checked/proved.

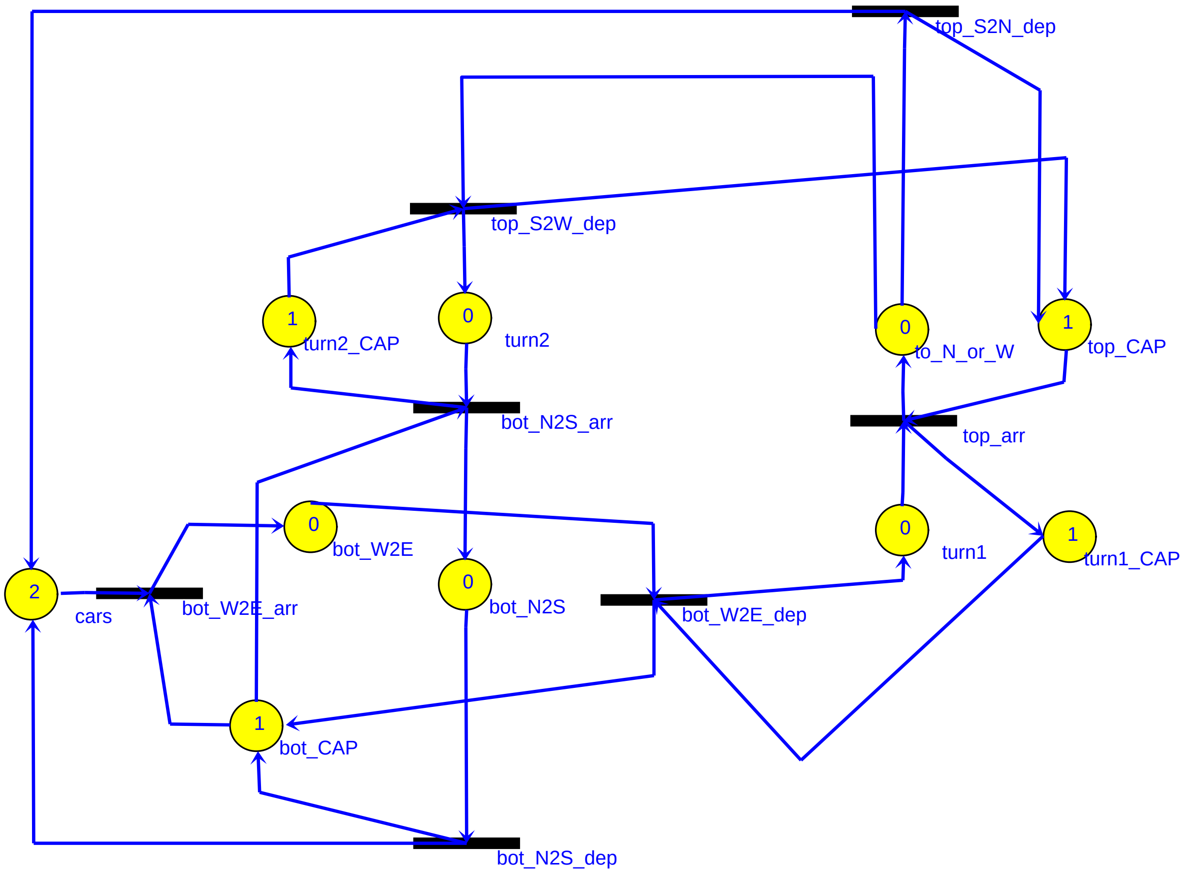
Un-timed Analysis



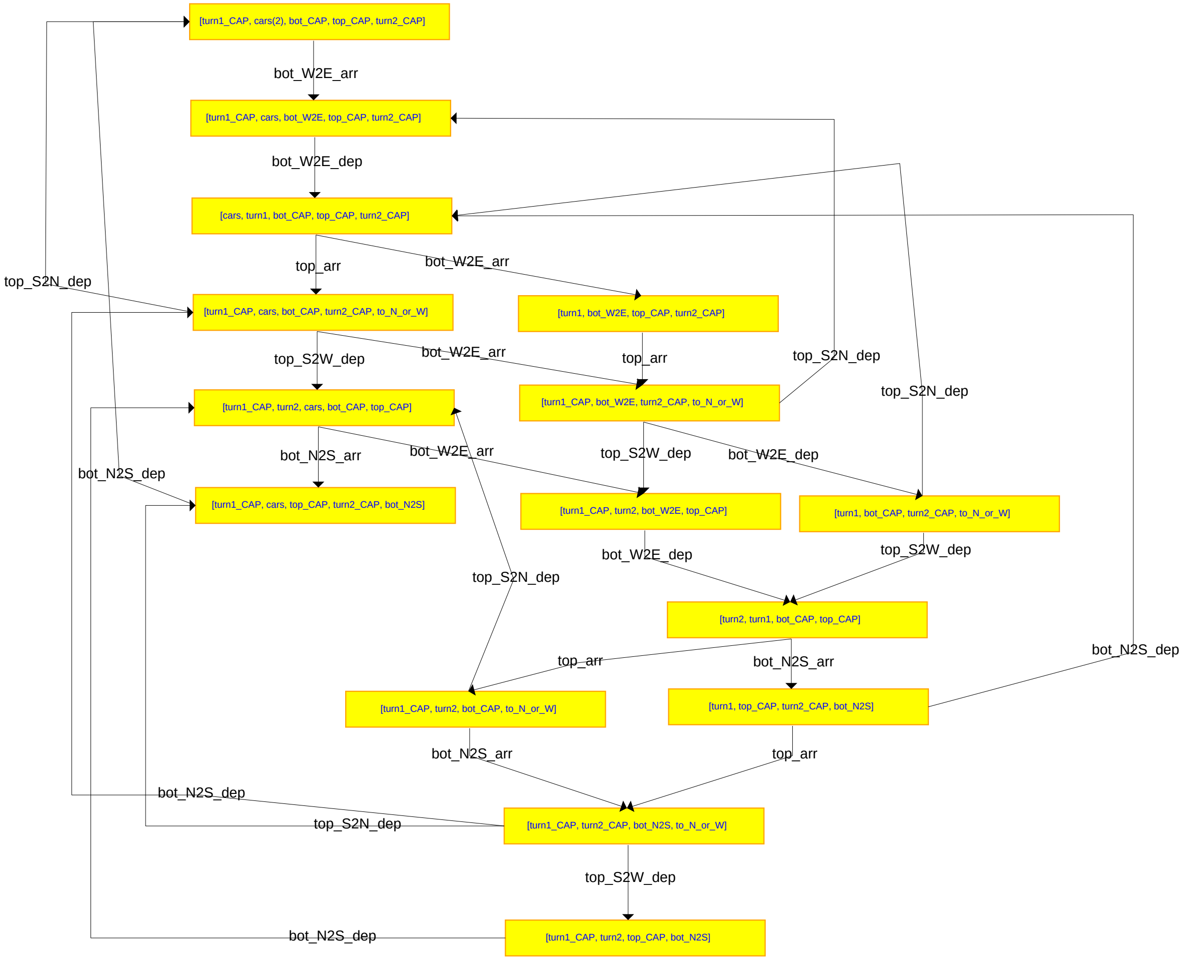
An un-timed Traffic model



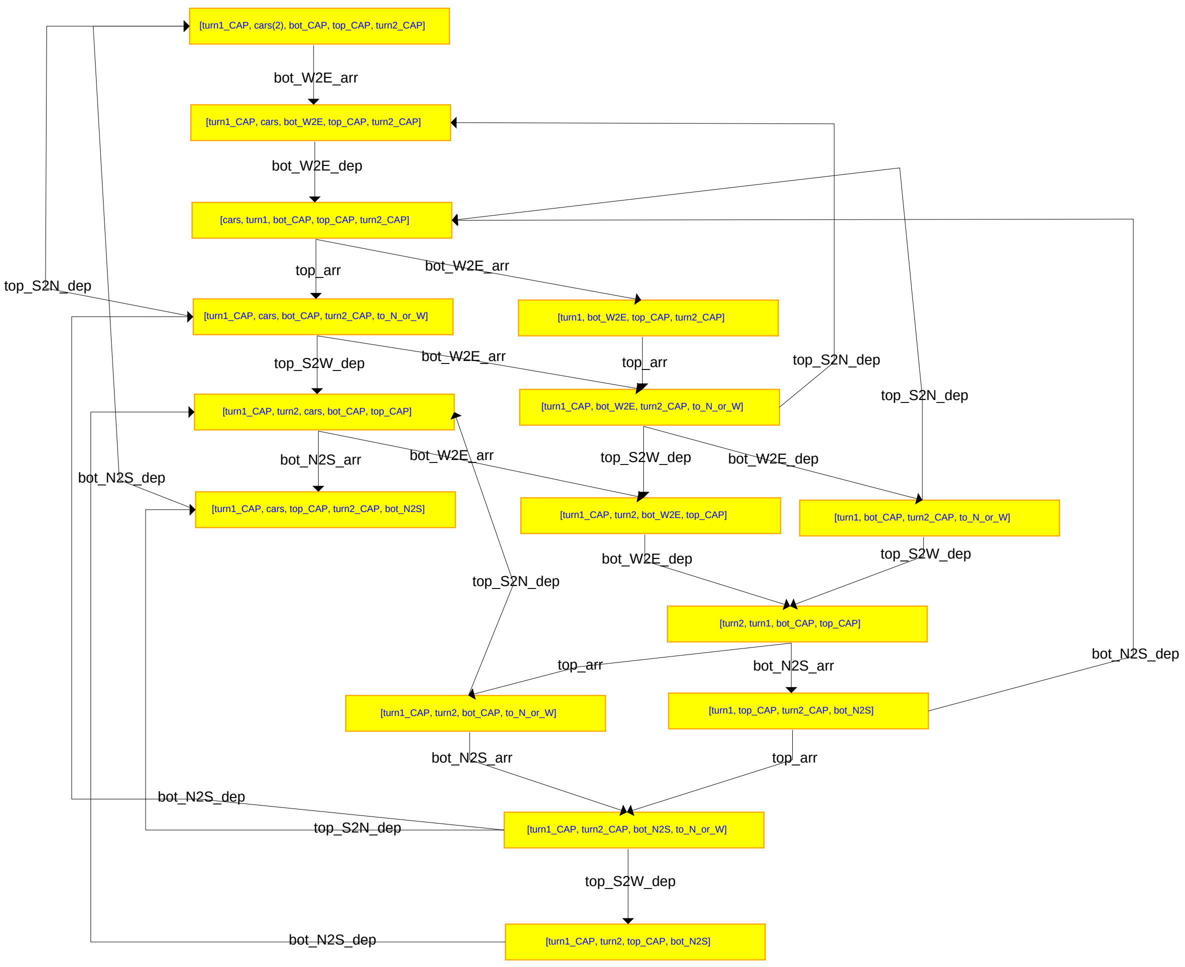
the Petri Net describing its behaviour obtained by Graph Rewriting



Analysis: Coverability Graph of the Petri Net


[turn1_CAP, cars(2), bot_CAP, top_CAP, turn2_CAP]
bot_W2E_arr
[turn1_CAP, cars, bot_W2E, top_CAP, turn2_CAP]
bot_W2E_dep
[cars, turn1, bot_CAP, top_CAP, turn2_CAP]
top_arr
bot_W2E_arr
top_S2N_dep
[turn1_CAP, cars, bot_CAP, turn2_CAP, to_N_or_W]
[turn1, bot_W2E, top_CAP, turn2_CAP]
top_S2W_dep
bot_W2E_arr
top_arr
top_S2N_dep
top_S2N_dep
[turn1_CAP, turn2, cars, bot_CAP, top_CAP]
[turn1_CAP, bot_W2E, turn2_CAP, to_N_or_W]
bot_N2S_arr
bot_W2E_arr
top_S2W_dep
bot_W2E_dep
bot_N2S_dep
[turn1_CAP, cars, top_CAP, turn2_CAP, bot_N2S]
[turn1_CAP, turn2, bot_W2E, top_CAP]
[turn1, bot_CAP, turn2_CAP, to_N_or_W]
bot_W2E_dep
top_S2W_dep
top_S2N_dep
[turn2, turn1, bot_CAP, top_CAP]
top_arr
bot_N2S_arr
bot_N2S_dep
[turn1_CAP, turn2, bot_CAP, to_N_or_W]
[turn1, top_CAP, turn2_CAP, bot_N2S]
bot_N2S_arr
top_arr
bot_N2S_dep
top_S2N_dep
[turn1_CAP, turn2_CAP, bot_N2S, to_N_or_W]
top_S2W_dep
bot_N2S_dep
[turn1_CAP, turn2, top_CAP, bot_N2S]

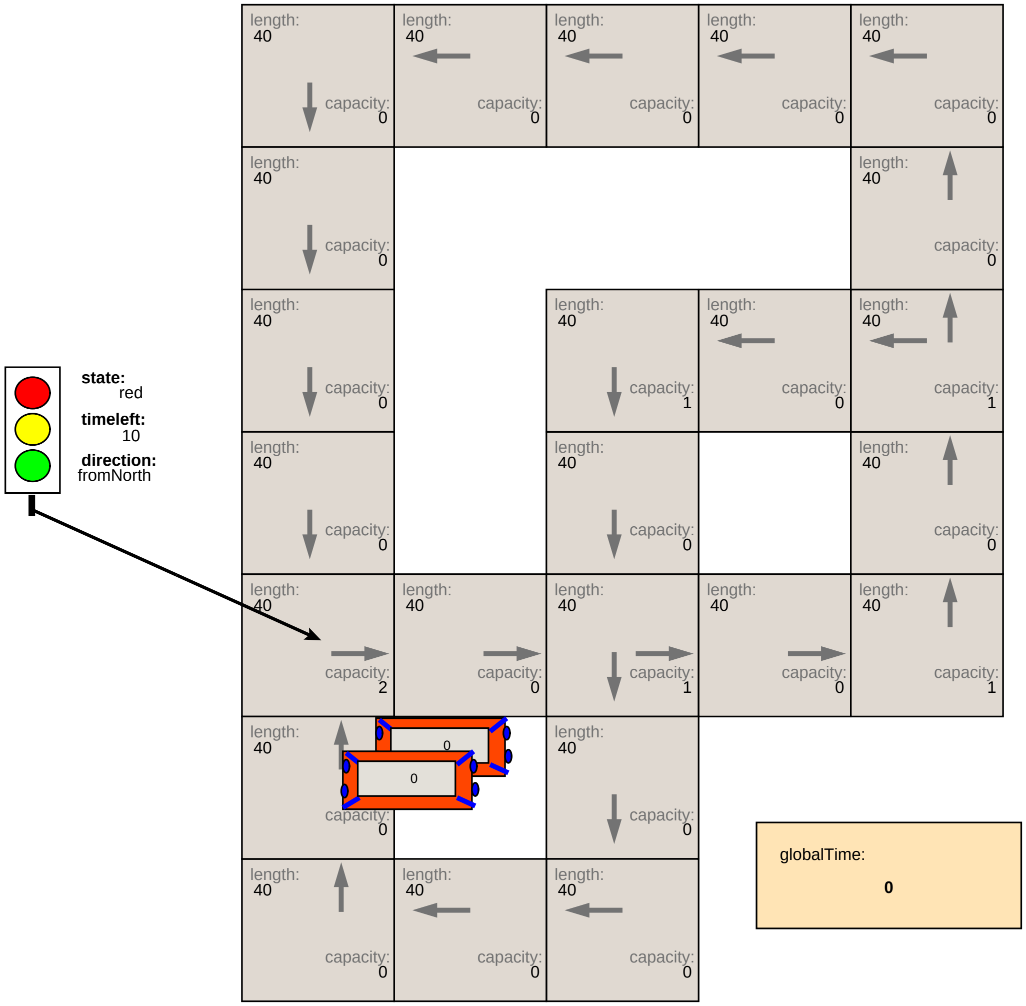
Liveness Analysis


[turn1_CAP, cars(2), bot_CAP, top_CAP, turn2_CAP]
bot_W2E_arr
[turn1_CAP, cars, bot_W2E, top_CAP, turn2_CAP]
bot_W2E_dep
[cars, turn1, bot_CAP, top_CAP, turn2_CAP]
top_arr
bot_W2E_arr
top_S2N_dep
[turn1_CAP, cars, bot_CAP, turn2_CAP, to_N_or_W]
[turn1, bot_W2E, top_CAP, turn2_CAP]
top_S2W_dep
bot_W2E_arr
top_arr
top_S2N_dep
top_S2N_dep
[turn1_CAP, turn2, cars, bot_CAP, top_CAP]
[turn1_CAP, bot_W2E, turn2_CAP, to_N_or_W]
bot_N2S_arr
bot_W2E_arr
top_S2W_dep
bot_W2E_dep
bot_N2S_dep
[turn1_CAP, cars, top_CAP, turn2_CAP, bot_N2S]
[turn1_CAP, turn2, bot_W2E, top_CAP]
[turn1, bot_CAP, turn2_CAP, to_N_or_W]
bot_W2E_dep
top_S2W_dep
top_S2N_dep
[turn2, turn1, bot_CAP, top_CAP]
bot_N2S_dep
top_arr
bot_N2S_arr
[turn1_CAP, turn2, bot_CAP, to_N_or_W]
[turn1, top_CAP, turn2_CAP, bot_N2S]
bot_N2S_arr
top_arr
bot_N2S_dep
top_S2N_dep
[turn1_CAP, turn2_CAP, bot_N2S, to_N_or_W]
top_S2W_dep
bot_N2S_dep
[turn1_CAP, turn2, top_CAP, bot_N2S]

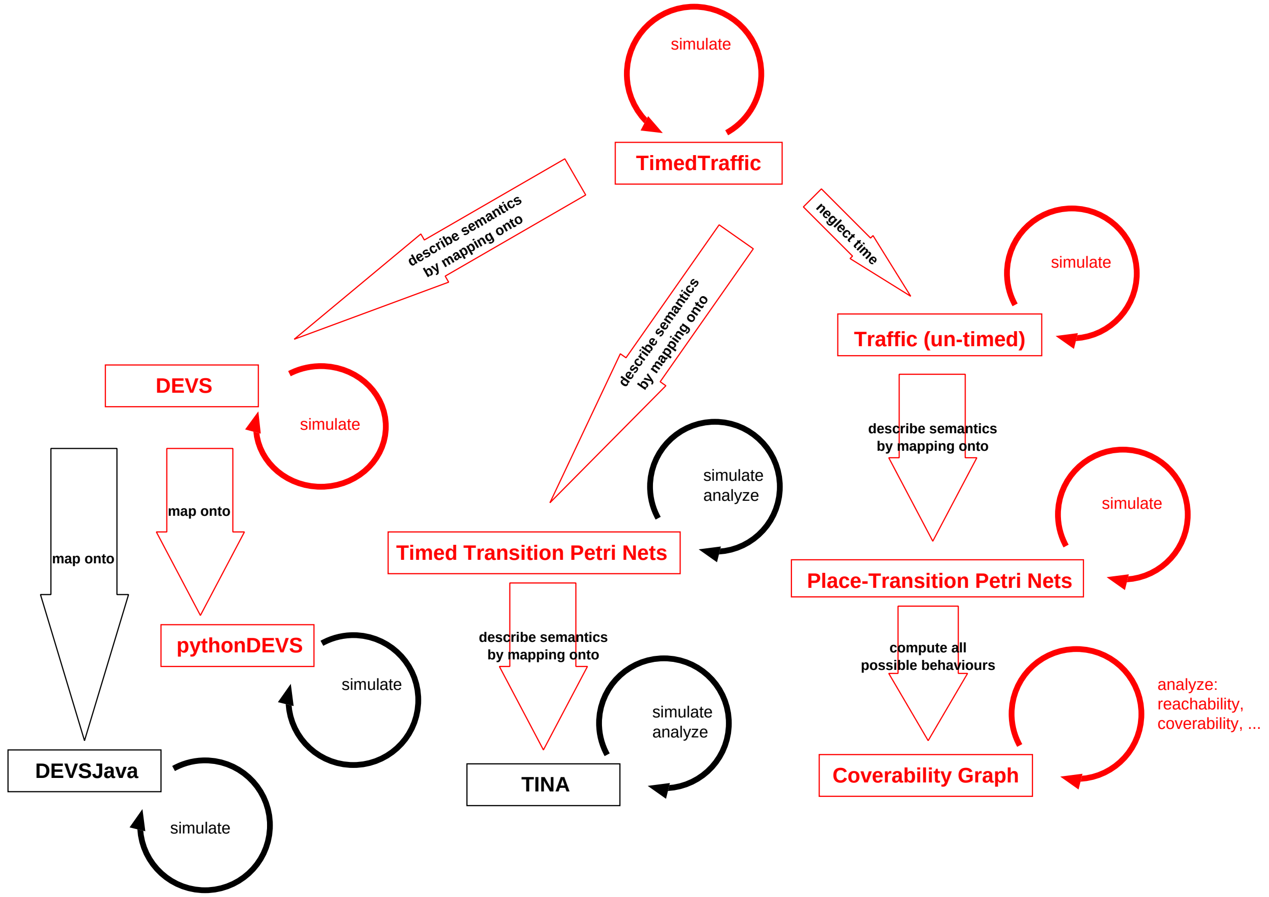
Conservation Analysis

```
1.0 x[turn1_CAP] + 1.0 x[turn1] = 1.0

1.0 x[cars] + 1.0 x[bot_W2E] + 1.0 x[turn1] +
1.0 x[to_N_or_W] + 1.0 x[turn2] + 1.0 x[bot_N2S] = 2.0

1.0 x[top_CAP] + 1.0 x[to_N_or_W] = 1.0

1.0 x[turn2_CAP] + 1.0 x[turn2] = 1.0

1.0 x[bot_CAP] + 1.0 x[bot_W2E] + 1.0 x[bot_N2S] = 1.0
```

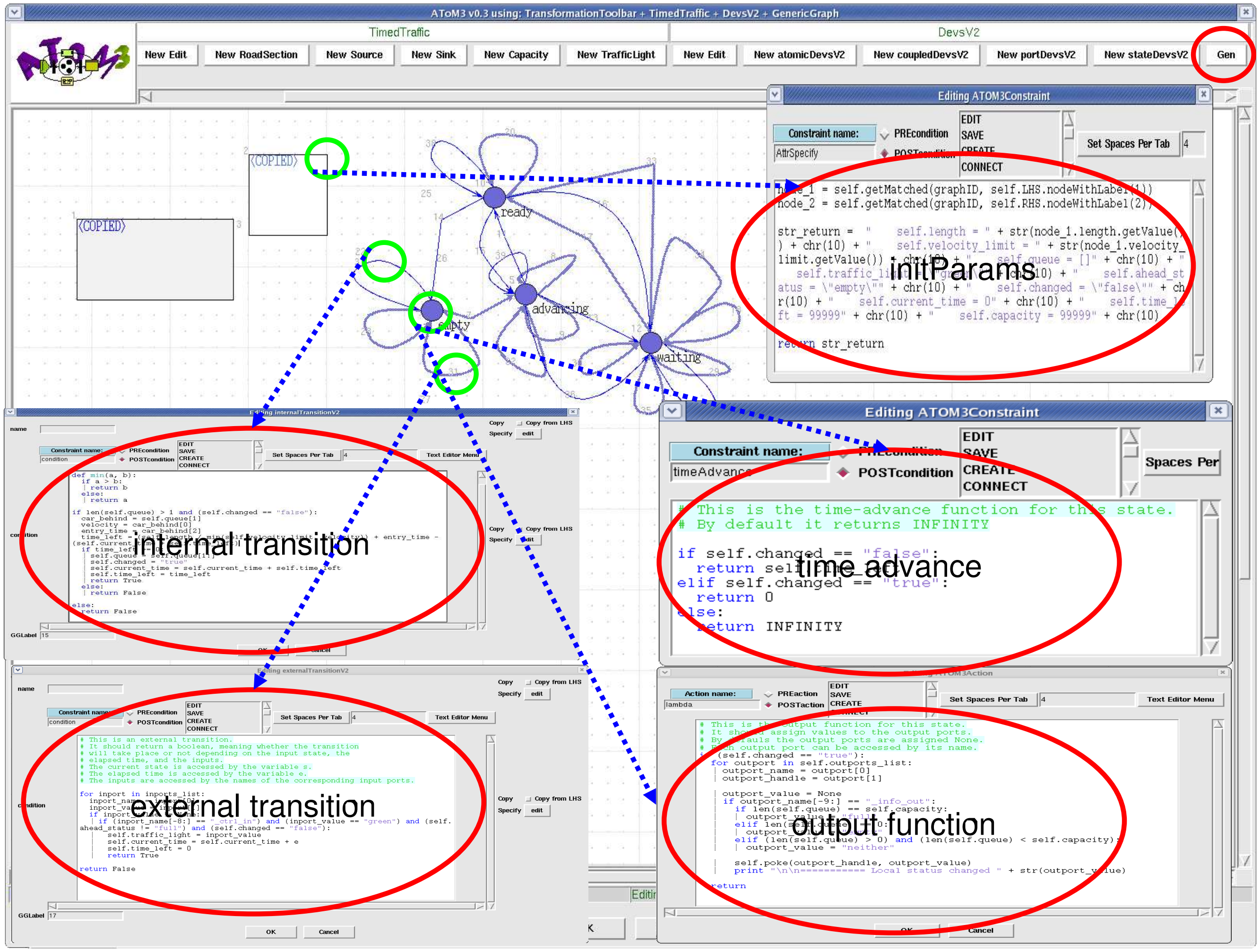
Timed Traffic Network



Mapping onto DEVS for Simulation (performance Analysis)


simulate
TimedTraffic
describe semantics by mapping onto
describe semantics by mapping onto
neglect time
simulate
Traffic (un-timed)
DEVS
simulate
describe semantics by mapping onto
simulate analyze
simulate
map onto
map onto
Timed Transition Petri Nets
Place-Transition Petri Nets
pythonDEVS
simulate
describe semantics by mapping onto
compute all possible behaviours
simulate analyze
analyze: reachability, coverability, ...
DEVSJava
TINA
Coverability Graph
simulate

Timed Traffic mapped onto a DEVS model



initParams

Constraint name: AttrSpecify

```
node_1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node_2 = self.getMatched(graphID, self.RHS.nodeWithLabel(2))

str_return =  "    self.length = " + str(node_1.length.getValue()
) + chr(10) + "    self.velocity_limit = " + str(node_1.velocity_
limit.getValue()) + chr(10) + "    self.queue = []" + chr(10) + "
   self.traffic_light = \"green\"" + chr(10) + "    self.ahead_st
atus = \"empty\"" + chr(10) + "    self.changed = \"false\"" + ch
r(10) + "    self.current_time = 0" + chr(10) + "    self.time_le
ft = 99999" + chr(10) + "    self.capacity = 99999" + chr(10)

return str_return
```

internal transition

Constraint name: condition

```
def min(a, b):
  if a > b:
    return b
  else:
    return a

if len(self.queue) > 1 and (self.changed == "false"):
  car_behind = self.queue[1]
  velocity = car_behind[0]
  entry_time = car_behind[2]
  time_left = (self.length / min(self.velocity_limit, velocity)) + entry_time -
(self.current_time + self.time_left)
  if time_left <= 0:
    self.queue = self.queue[1:]
    self.changed = "true"
    self.current_time = self.current_time + self.time_left
    self.time_left = time_left
    return True
  else:
    return False

else:
  return False
```

GGLabel 15

time advance

Constraint name: timeAdvance

```
# This is the time-advance function for this state.
# By default it returns INFINITY

if self.changed == "false":
  return self.time_left
elif self.changed == "true":
  return 0
else:
  return INFINITY
```

external transition

Constraint name: condition

```
# This is an external transition.
# It should return a boolean, meaning whether the transition
# will take place or not depending on the input state, the
# elapsed time, and the inputs.
# The current state is accessed by the variable s.
# The elapsed time is accessed by the variable e.
# The inputs are accessed by the names of the corresponding input ports.

for inport in inports_list:
  inport_name = inport[0]
  inport_value = inport[1]
  if inport_value != None:
    if (inport_name[-8:] == "_ctrl_in") and (inport_value == "green") and (self.
ahead_status != "full") and (self.changed == "false"):
      self.traffic_light = inport_value
      self.current_time = self.current_time + e
      self.time_left = 0
      return True

return False
```

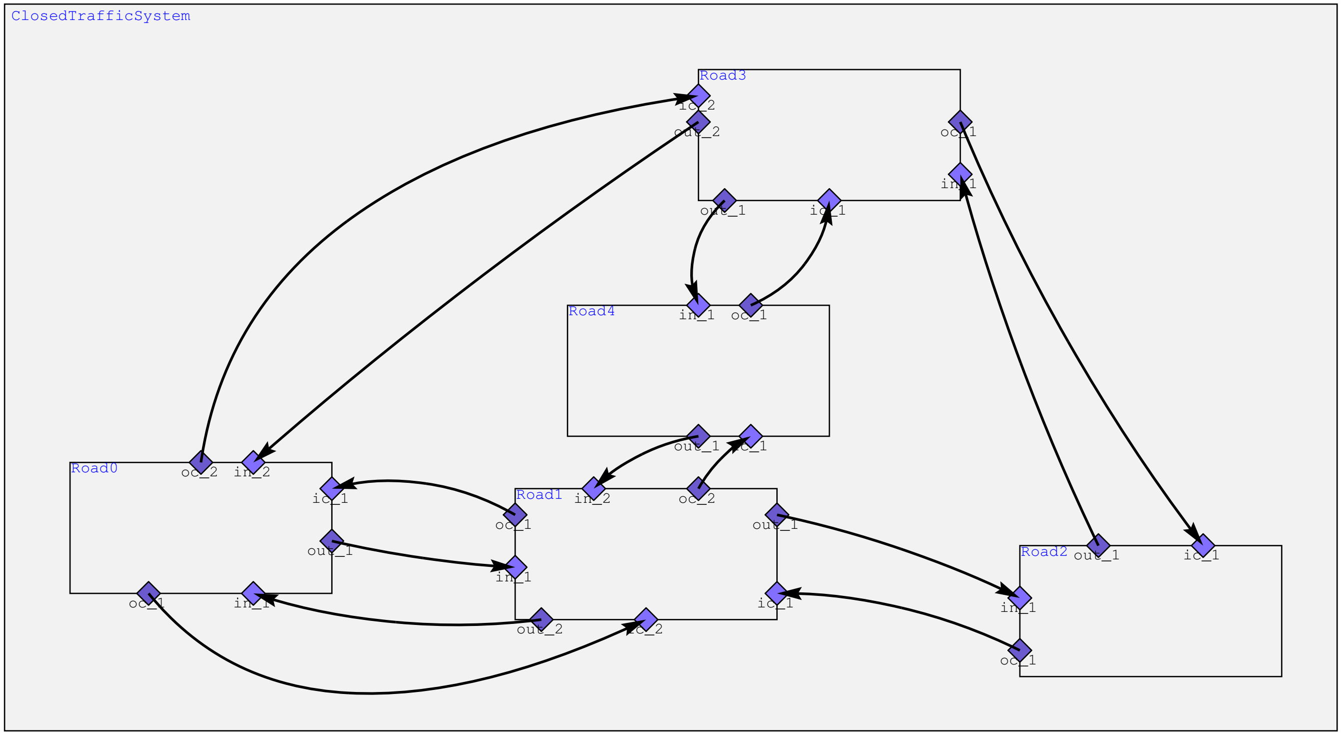
GGLabel 17

output function

Action name: lambda

```
# This is the output function for this state.
# It should assign values to the output ports.
# By defauls the output ports are assigned None.
# Each output port can be accessed by its name.
if (self.changed == "true"):
  for outport in self.outports_list:
    outport_name = outport[0]
    outport_handle = outport[1]

    outport_value = None
    if outport_name[-9:] == "_info_out":
      if len(self.queue) == self.capacity:
        outport_value = "full"
      elif len(self.queue) == 0:
        outport_value = "empty"
      elif (len(self.queue) > 0) and (len(self.queue) < self.capacity):
        outport_value = "neither"

      self.poke(outport_handle, outport_value)
      print "\n\n========== Local status changed " + str(outport_value)

return
```

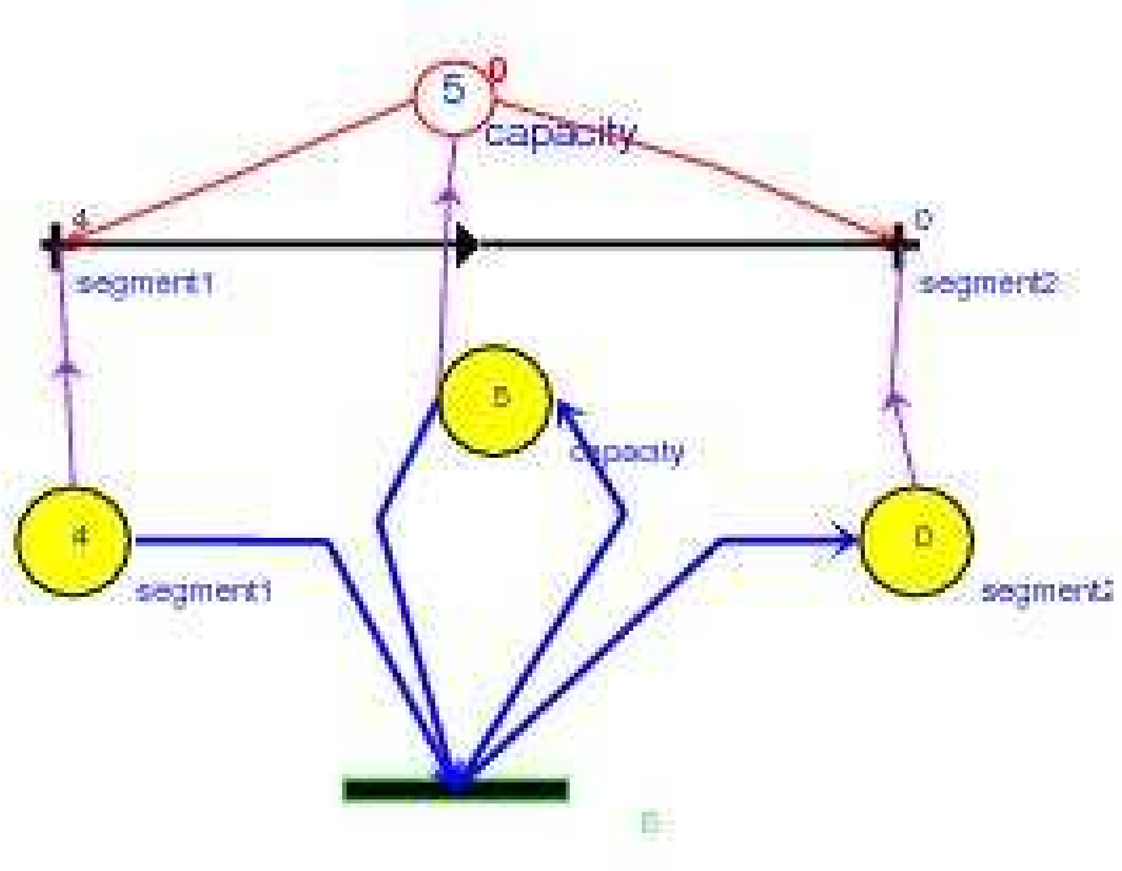
Timed Traffic mapped onto a DEVS model



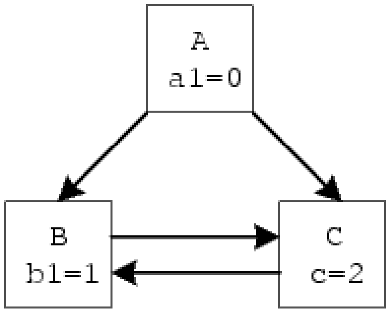
Semper Variabilis: Model Evolution

- model evolution
- meta-model evolution
- semantics evolution

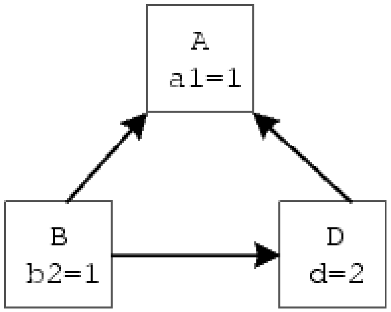
Model Evolution poor man's approach: Backward Links



Model Evolution (Version Control): need Model Comparison



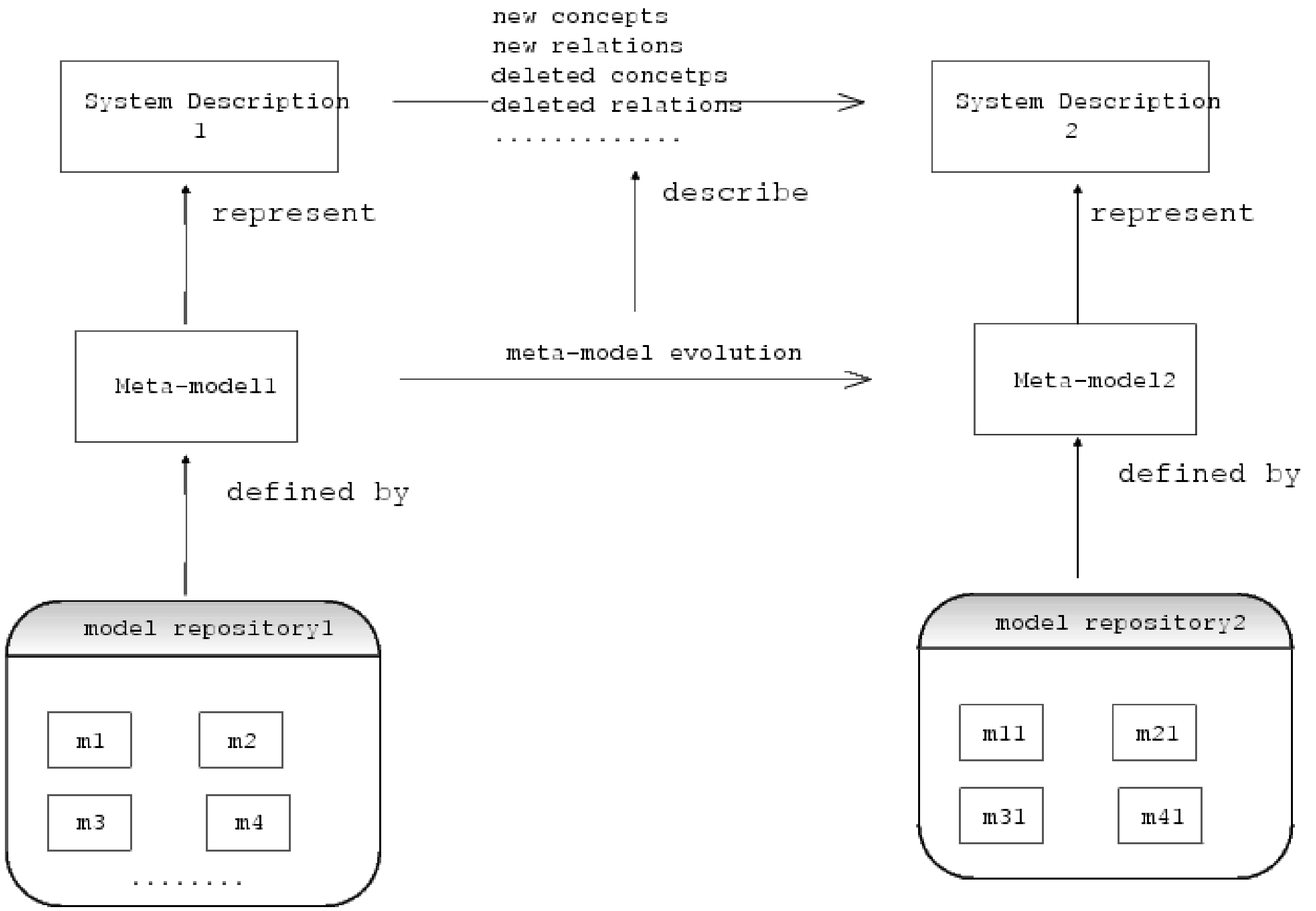
M-old



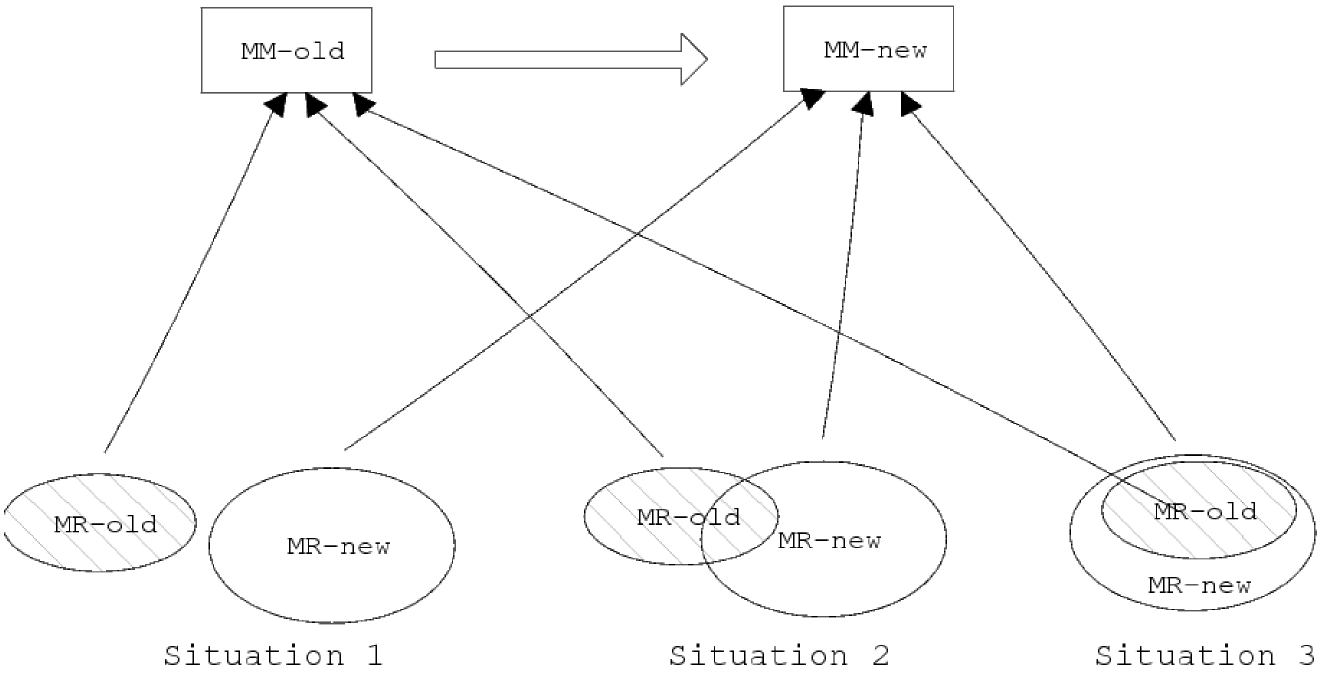
M-new

express difference as sequence of creation, removal, attribute change

Meta-model evolution

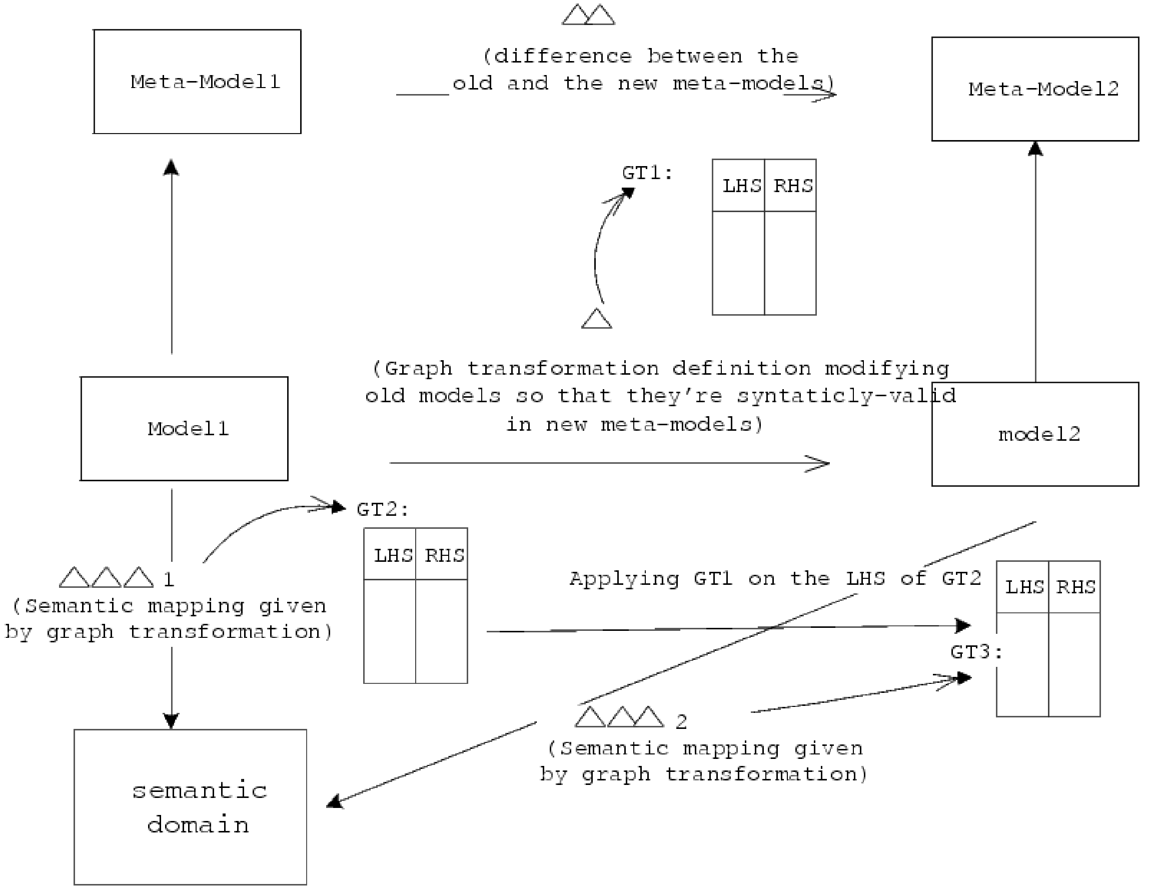


Cases



MM-old
MM-new
MR-old
MR-new
MR-old
MR-new
MR-old
MR-new
Situation 1
Situation 2
Situation 3

Semantics evolution



Conclusions

model **everything** !