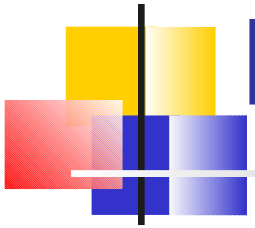
DEVS modeling of Traffic in AToM3

Presented by

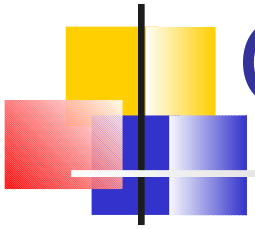
Ximeng Sun

April 11, 2005



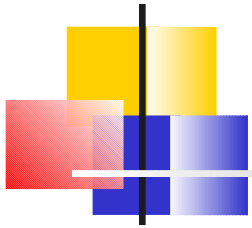
References

- [1] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim.
  Theory of Modeling and Simulation.
  Academic Press, 2000.
- [2] Hans Vangheluwe, Juan de Lara.
  Computer Automated Multi-Paradigm Modelling for Analysis and Design of Traffic Networks.
  Proceedings of the 2004 Winter Simulation Conference.
- [3] Ernesto Posse and Jean-Sebastien Bolduc.
  Generation of DEVS Modeling and Simulation Environments.
  Proceedings of the 2003 Summer Simulation MultiConference, 2003
- [4] Modelling, Simulation and Design Lab.
  AToM3 V0.3: A Tool for Multi-formalism and Meta-Modelling
  http://msdl.cs.mcgill.ca/MSDL/research/
- [5] Bernard P. Zeigler, Hessam S. Sarjoughian.
  Introduction to DEVS Modeling and Simulation with JAVA.
  http://www.acims.arizona.edu/SOFTWARE/software.shtml#DEVSJAVA



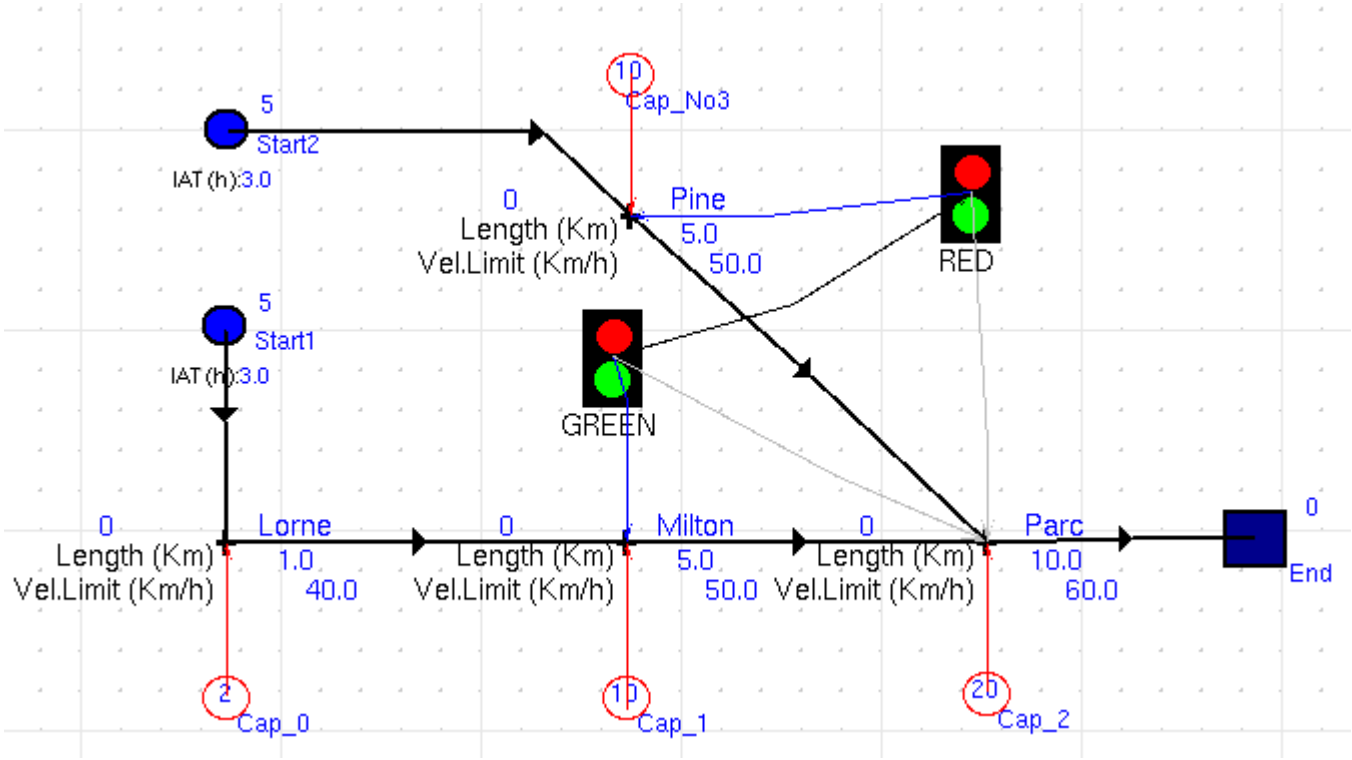
Outline

- Introduction
  - TimedTraffic Formalism
  - DEVS Formalism
- Map TimedTraffic to DEVS
  - Meta-Modeling
    - TimedTraffic Meta-Model
    - DEVS Meta-model
  - Model Transformation
  - Code Generation (Python and Java)
- Demo
- Conclusion



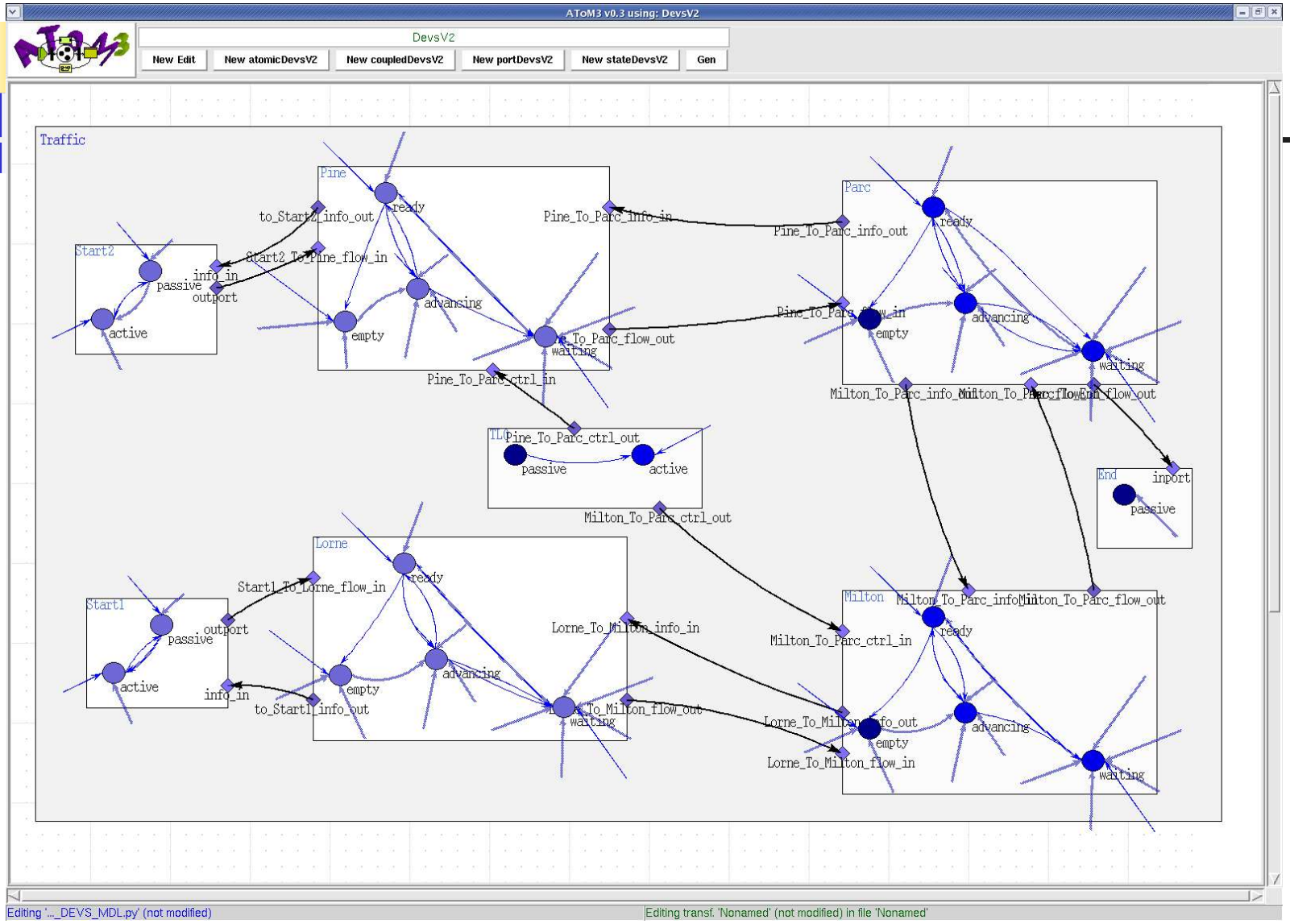
- Introduction
  - TimedTraffic Formalism
  - DEVS Formalism
- Map TimedTraffic to DEVS
  - Meta-Modeling
    - TimedTraffic Meta-Model
    - DEVS Meta-model
  - Model Transformation
  - Code Generation
- Demo
- Conclusion

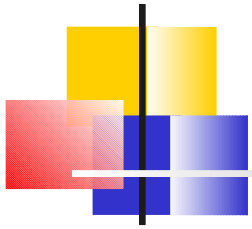
TimedTraffic Formalism



Based on models described in [2].

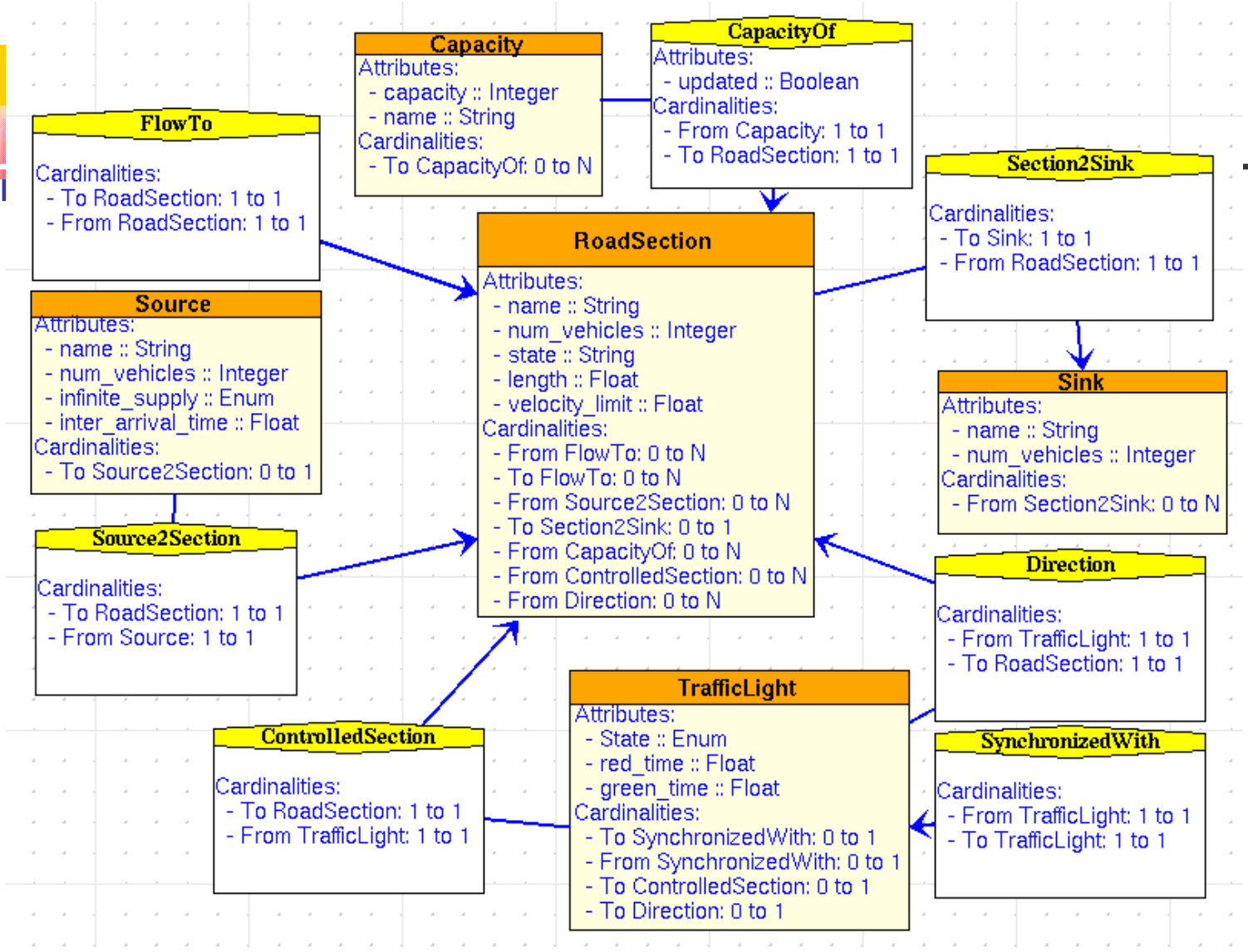
DEVS Formalism





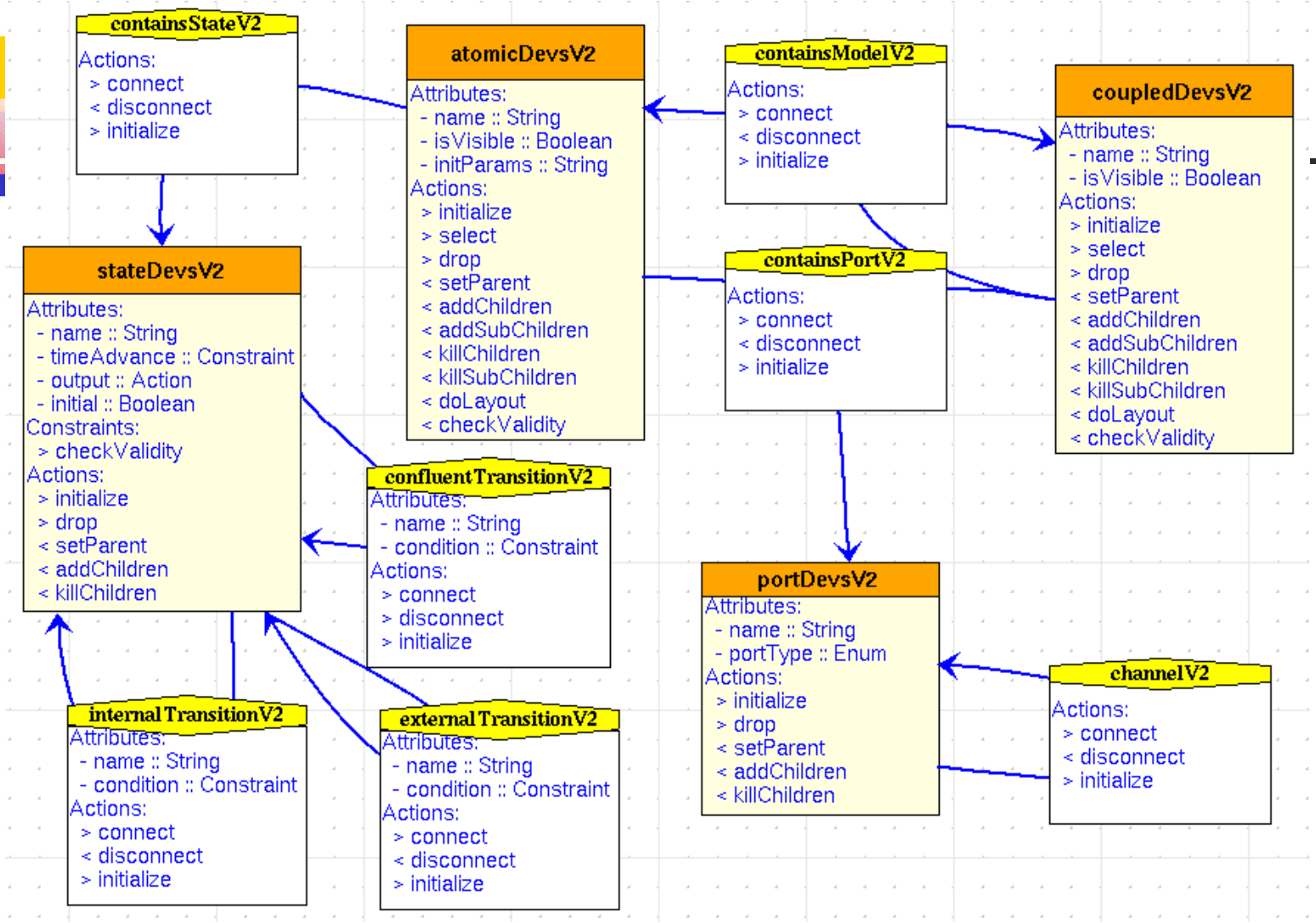
- Introduction
  - TimedTraffic Formalism
  - DEVS Formalism
- Map TimedTraffic to DEVS
  - Meta-Modeling
    - TimedTraffic Meta-Model
    - DEVS Meta-model
  - Model Transformation
  - Code Generation
- Demo
- Conclusion

TimedTraffic Meta-Model



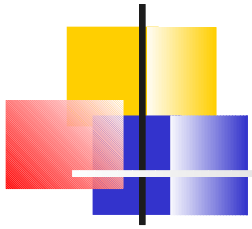
Based on the meta-model described in [2].

DEVS Meta-Model



Based on the meta-model described in [3] and the work done by Denis Dube (http://moncs.cs.mcgill.ca/people/hv/teaching/MSBDesign/presentations/050324.DenisDube.pdf)

TimedTraffic to DEVS Transformation Rules



- AToM3 can automatically generate nice documents for all transformation rules.

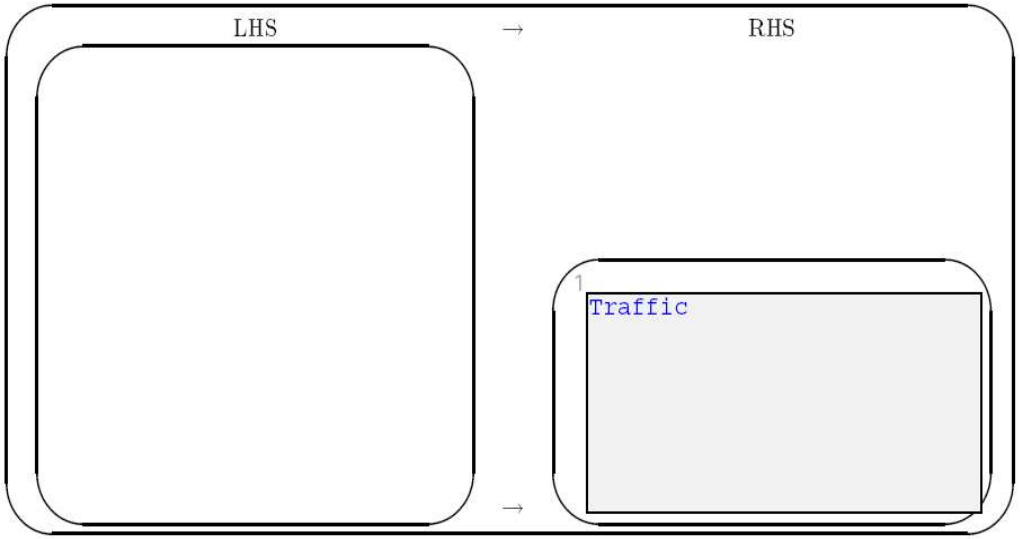
Rule 0: Initialization

Global initial action:

```
self.rootTrafficGenerated = False

for node in graph.listNodes["Source"]:
   node.sourceGeneratorGenerated = False

for node in graph.listNodes["Sink"]:
   node.sinkCollectorGenerated = False

for node in graph.listNodes["RoadSection"]:
   node.roadAtomicGenerated = False

for node in graph.listNodes["Capacity"]:
   node.capacityInfoGenerated = False

for node in graph.listNodes["TrafficLight"]:
   node.trafficLightWithRoadGenerated = False
   node.trafficLightSyncGenerated = False

self.TL_count = 0
```

Rule 1: Generate Top-level "Traffic" Coupled DEVS

Rule 1 (Order 1): RootTraffic



Precondition:

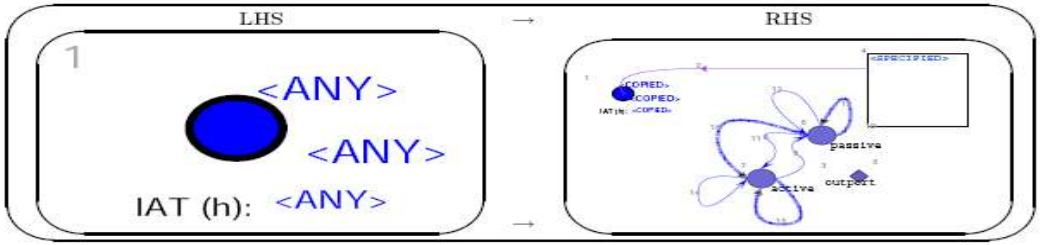
```
return not self.graphGrammar.rootTrafficGenerated
```

Post action:

```
self.graphGrammar.rootTrafficGenerated = True
```

Rule 2: Transform “Source” to “Generator” Atomic DEVS

Rule 2 (Order 2): Source2Generator



Precondition:

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not node.sourceGeneratorGenerated
```

Post action:

```
node_1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node_4 = self.getMatched(graphID, self.RHS.nodeWithLabel(4))

node_1.sourceGeneratorGenerated = True

node_4.connectedWithCoupled = False
node_4.model_type = "generator"
```

Specify: *atomicDevsV2 #4*

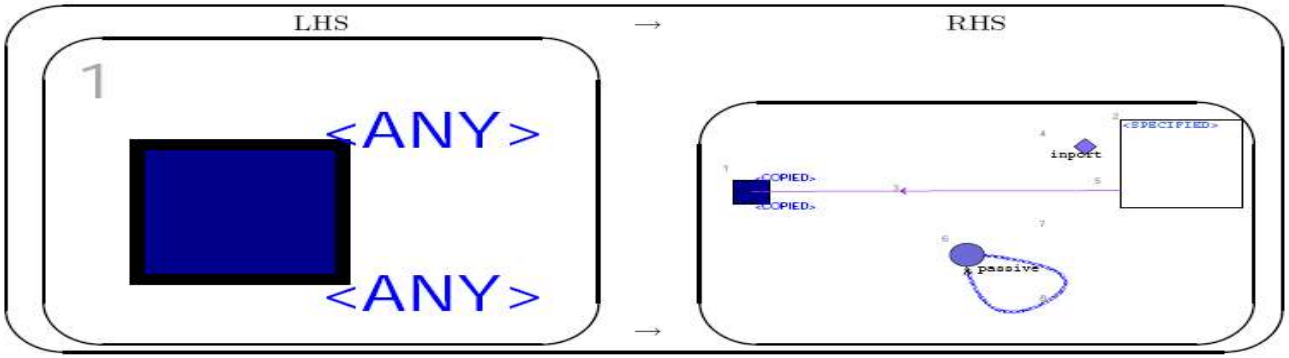
```
node_1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))

str_retrun = "    self.num_vehicles = " + str(node_1.num_vehicles.getValue()) + chr(10)
+ "    self.inter_arrival_time = " + str(node_1.inter_arrival_time.getValue()) + chr(10)
+ "    self.ahead_status = \"empty\"" + chr(10)
+ "    self.time_left = " + str(node_1.inter_arrival_time.getValue()) + chr(10)
+ "    self.remain_vehichles = "

if node_1.infinite_supply.getValue() == True:
  str_retrun = str_retrun + str(99999)
else:
  str_retrun = str_retrun + str(node_1.num_vehicles.getValue())

return str_retrun
```

Specify: *atomicDevsV2 #4*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue()
```

Rule 3: Transform “Sink” to “Collector” Atomic DEVS

Rule 3 (Order 3): Sink2Collector



Precondition:

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not node.sinkCollectorGenerated
```

Post action:

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

node_1.sinkCollectorGenerated = True

node_2.connectedWithCoupled = False
node_2.model_type = "collector"
```

Specify: *atomicDevsV2 #2*

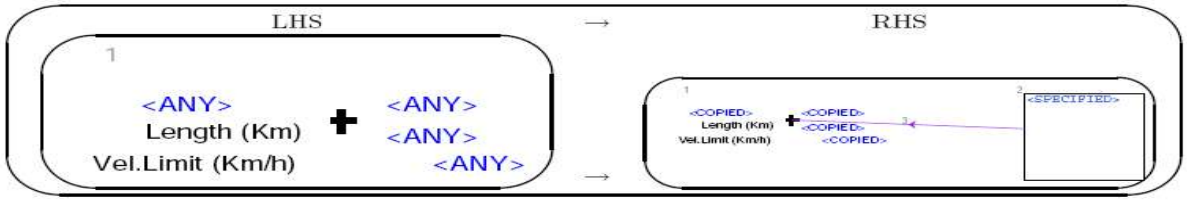
```
return "    self.arrived = 0" + chr(10) + "    self.current_time = 0" + chr(10) + "    self.average = 0" + chr(10)
```

Specify: *atomicDevsV2 #2*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue()
```

Rule 4: Transform “RoadSection” to “Road” Atomic DEVS

Rule 4 (Order 4): Road2Road



Precondition:

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not node.roadAtomicGenerated
```

Post action:

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

node_1.roadAtomicGenerated = True

node_2.connectedWithCoupled = False
node_2.model_type = "road"
```

Specify: *atomicDevsV2 #2*

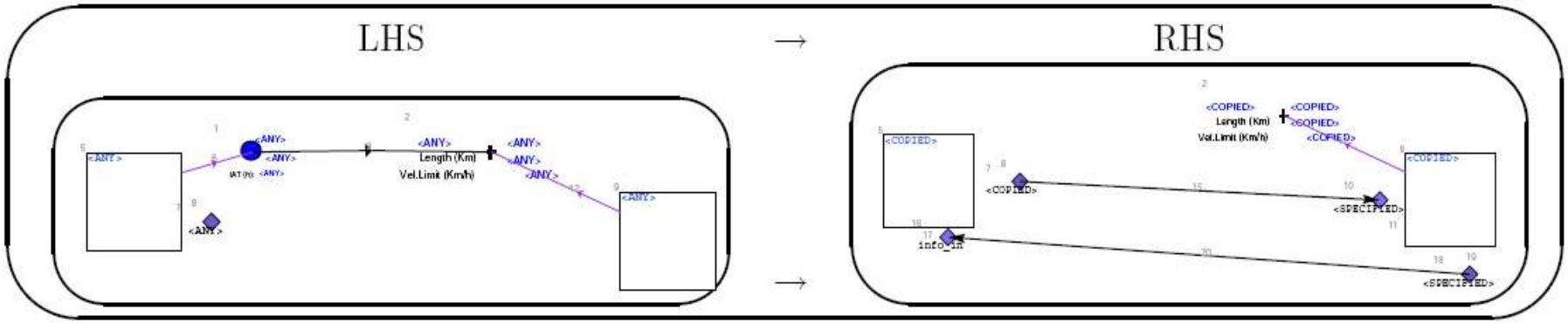
```
node_1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node_2 = self.getMatched(graphID, self.RHS.nodeWithLabel(2))

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_status = \"empty\"" + chr(10) + "    self.changed = \"false\"" + chr(10)
+ "    self.current_time = 0" + chr(10) + "    self.time_left = 99999" + chr(10)
+ "    self.capacity = 99999" + chr(10)

return str_return
```

Specify: *atomicDevsV2 #2*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue()
```

Rule 5: Connect “Generator” with “Road”

Rule 5 (Order 5): GeneratorWithRoad



Specify: *portDevsV2 #10*

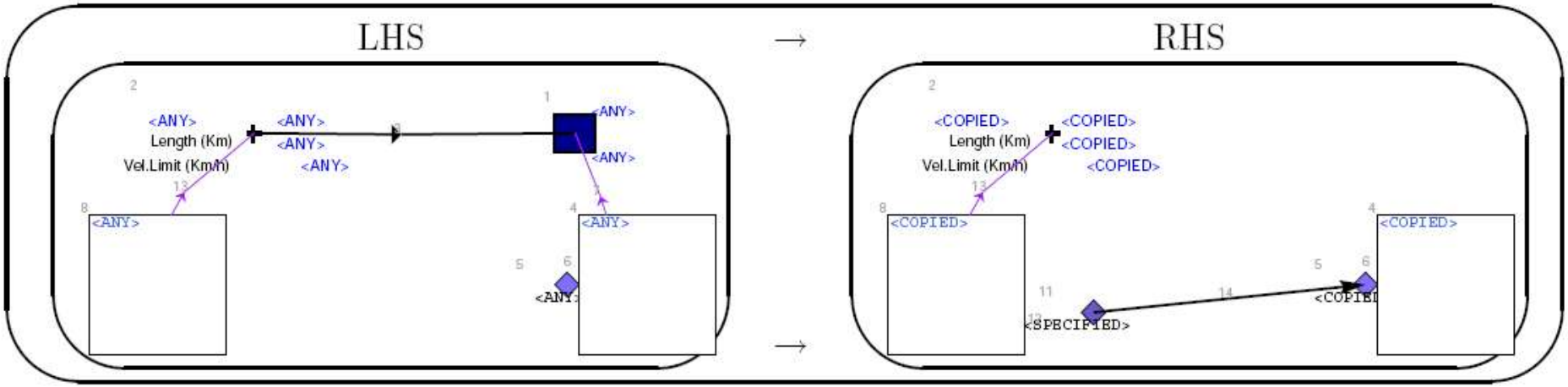
```
return self.getMatched(graphID, self.LHS.nodeWithLabel(5)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_flow_in'
```

Specify: *portDevsV2 #18*

```
return 'to_' + self.getMatched(graphID, self.LHS.nodeWithLabel(5)).name.getValue() + '_info_out'
```

Rule 6: Connect “Road” to “Collector”

Rule 6 (Order 6): RoadWithSink

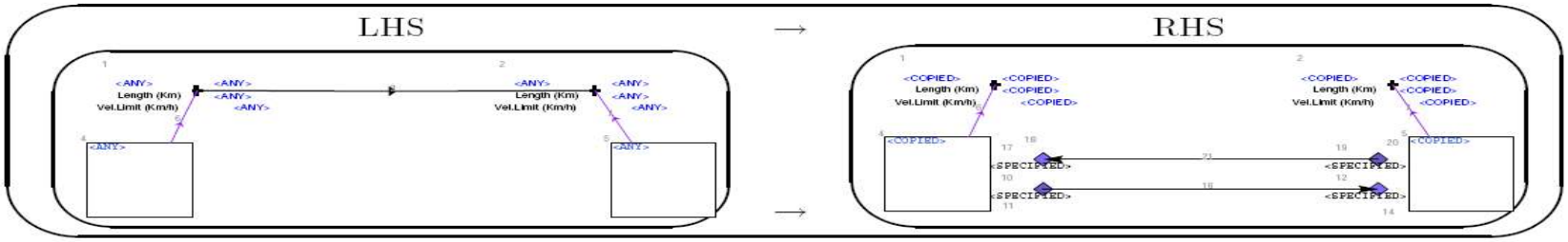


Specify: *portDevsV2 #11*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_flow_out'
```

Rule 7: Transform “FlowTo” to “Channel”

Rule 7 (Order 7): Flow2Channel



Specify: *portDevsV2 #10*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_flow_out'
```

Specify: *portDevsV2 #12*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_flow_in'
```

Specify: *portDevsV2 #17*

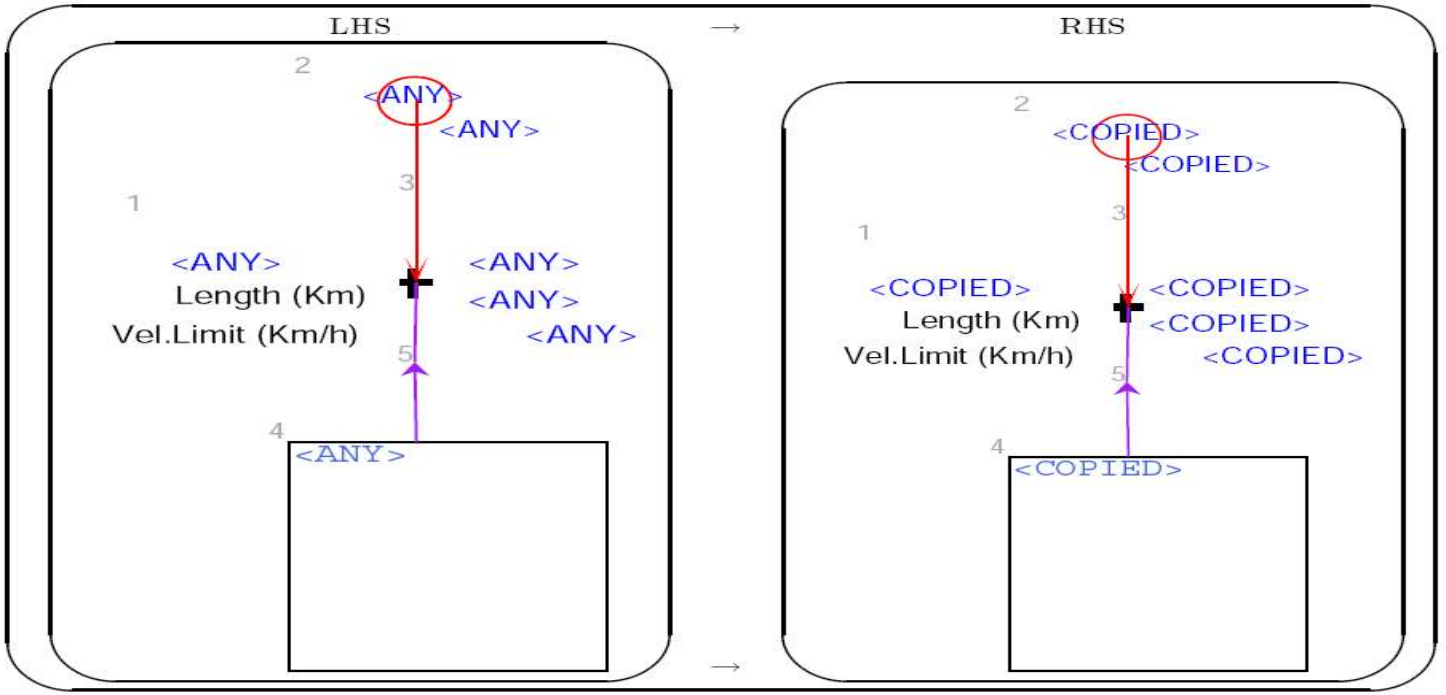
```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_info_in'
```

Specify: *portDevsV2 #19*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_info_out'
```

Rule 8: Copy “Capacity” information into “Road”

Rule 8 (Order 8): CapacityWithRoad



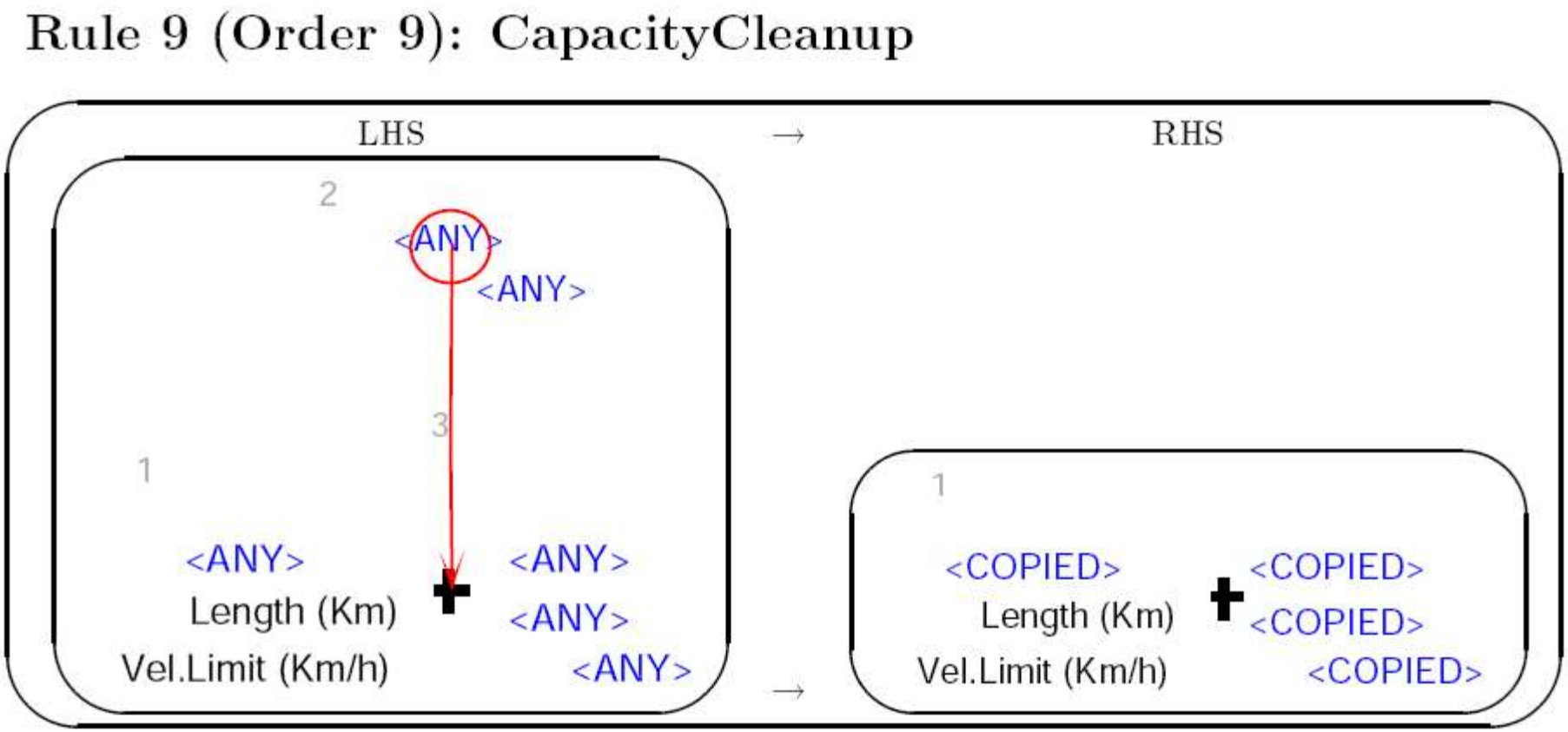
Precondition:

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return not node.capacityInfoGenerated
```

Post action:

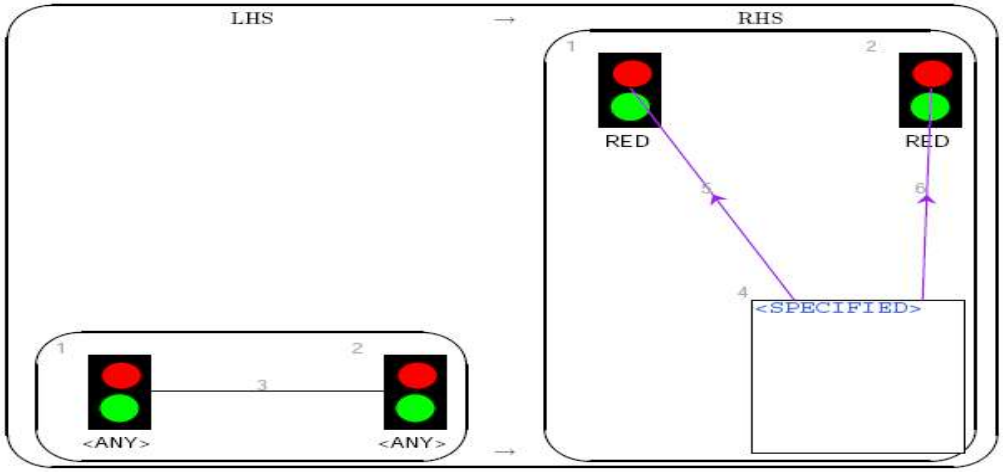
```
self.getMatched(graphID, self.LHS.nodeWithLabel(2)).capacityInfoGenerated = True
```

Rule 9: Remove “Capacity”



Rule 10: Transform “TrafficLight” to “TrafficLight” Atomic DEVS (double)

Rule 10 (Order 10): TrafficLightSync1



Precondition:

```
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
node2 = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return (not node1.trafficLightSyncGenerated) and (not node2.trafficLightSyncGenerated)
```

Post action:

```
self.getMatched(graphID, self.LHS.nodeWithLabel(1)).trafficLightSyncGenerated = True
self.getMatched(graphID, self.LHS.nodeWithLabel(2)).trafficLightSyncGenerated = True

node_4 = self.getMatched(graphID, self.RHS.nodeWithLabel(4))
node_4.connectedWithCoupled = False
node_4.model_type = "trafficLight"

self.graphGrammar.TL_count = self.graphGrammar.TL_count + 1
```

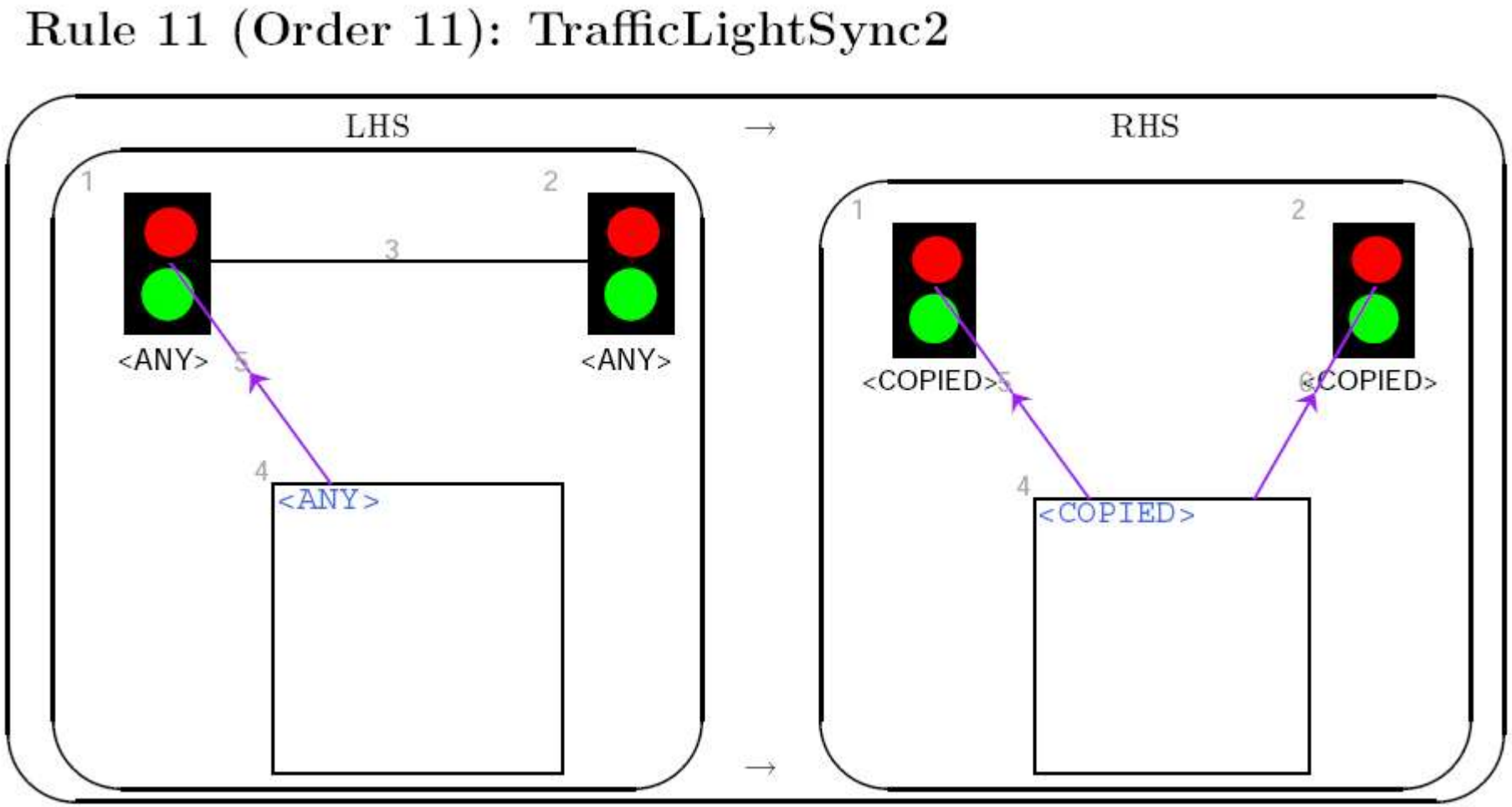
Specify: *atomicDevsV2 #4*

```
return ""
```

Specify: *atomicDevsV2 #4*

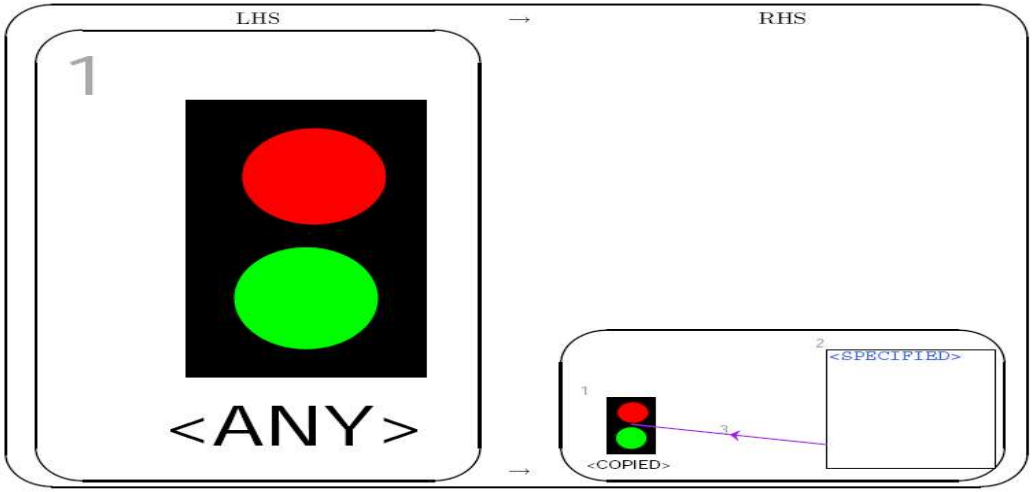
```
return "TL" + str(self.graphGrammar.TL_count)
```

Rule 11: Transform “SyncWith” relation



Rule 12: Transform “TrafficLight” to “TrafficLight” Atomic DEVS (single)

Rule 12 (Order 12): TrafficLightSync3



Precondition:

```
node1 = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not node1.trafficLightSyncGenerated
```

Post action:

```
self.getMatched(graphID, self.LHS.nodeWithLabel(1)).trafficLightSyncGenerated =True

node_2 = self.getMatched(graphID, self.RHS.nodeWithLabel(2))
node_2.connectedWithCoupled = False
node_2.model_type = "trafficLight"

self.graphGrammar.TL_count = self.graphGrammar.TL_count + 1
```

Specify: *atomicDevsV2 #2*

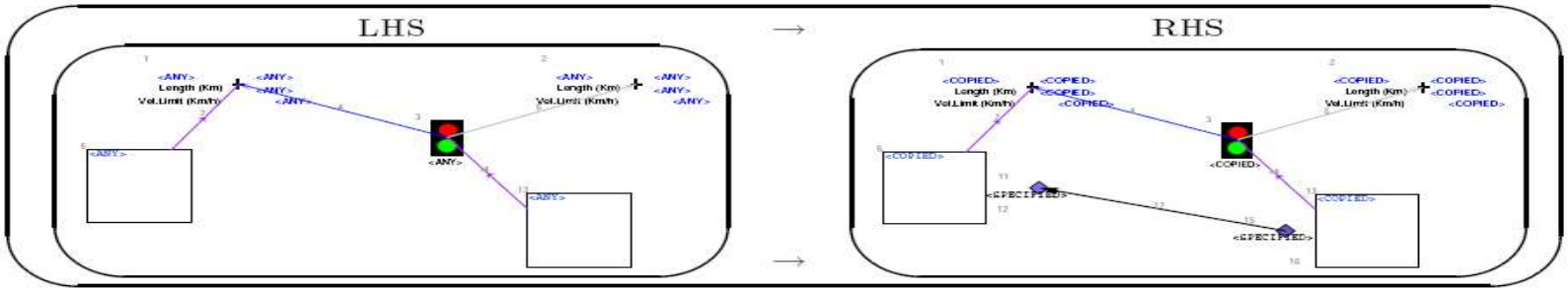
```
return ""
```

Specify: *atomicDevsV2 #2*

```
return "TL" + str(self.graphGrammar.TL_count)
```

Rule 13: Connect “TrafficLight” with “Road”

Rule 13 (Order 13): TrafficLightWithRoad



Precondition:

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
return not node.trafficLightWithRoadGenerated
```

Post action:

```
self.getMatched(graphID, self.LHS.nodeWithLabel(3)).trafficLightWithRoadGenerated = True
```

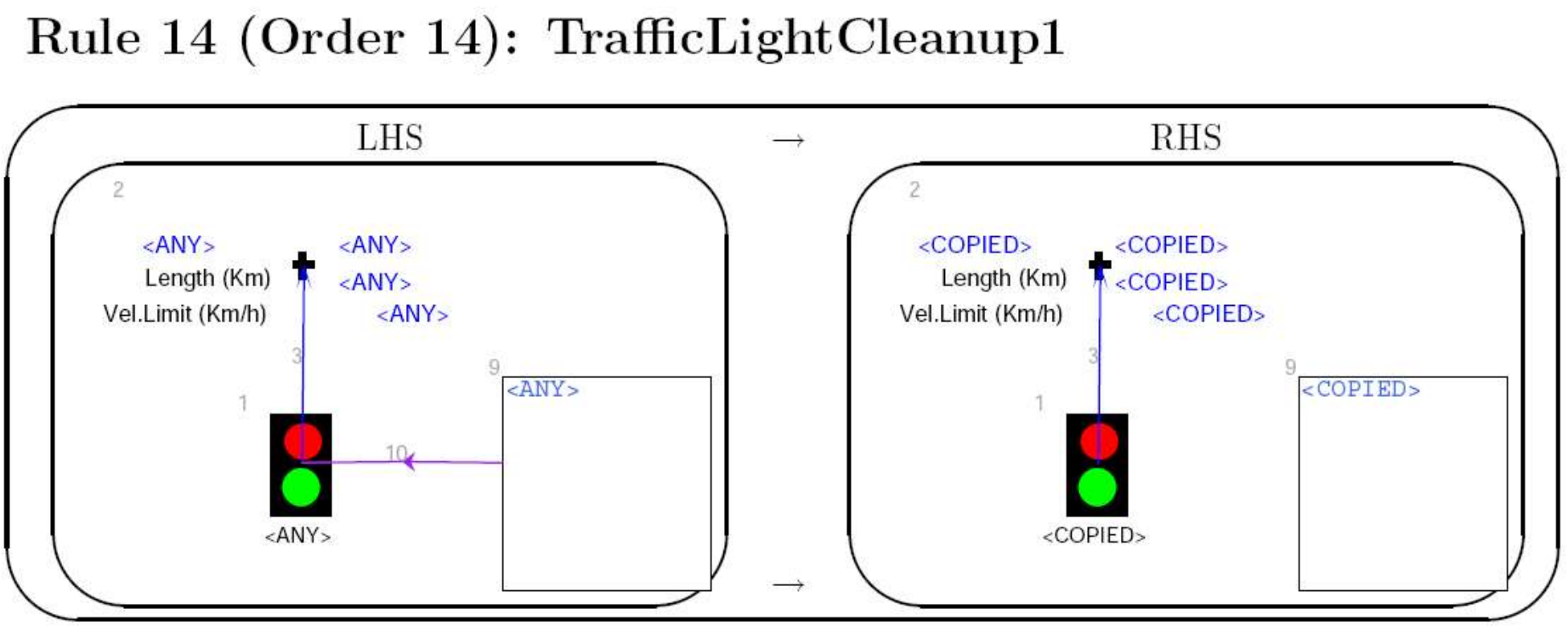
Specify: *portDevsV2 #11*

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_ctrl_in'
```

Specify: *portDevsV2 #15*

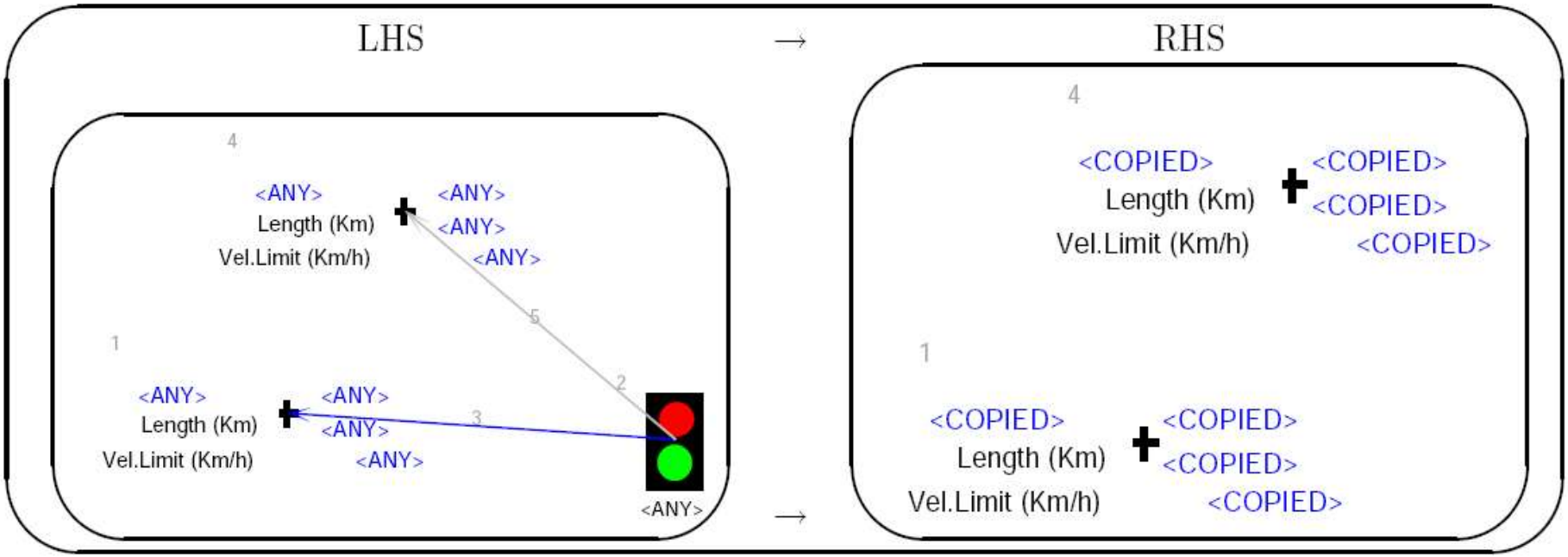
```
return self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.getValue() + '_To_'
+ self.getMatched(graphID, self.LHS.nodeWithLabel(2)).name.getValue() + '_ctrl_out'
```

Rule 14: Remove “TrafficLight” (step 1)

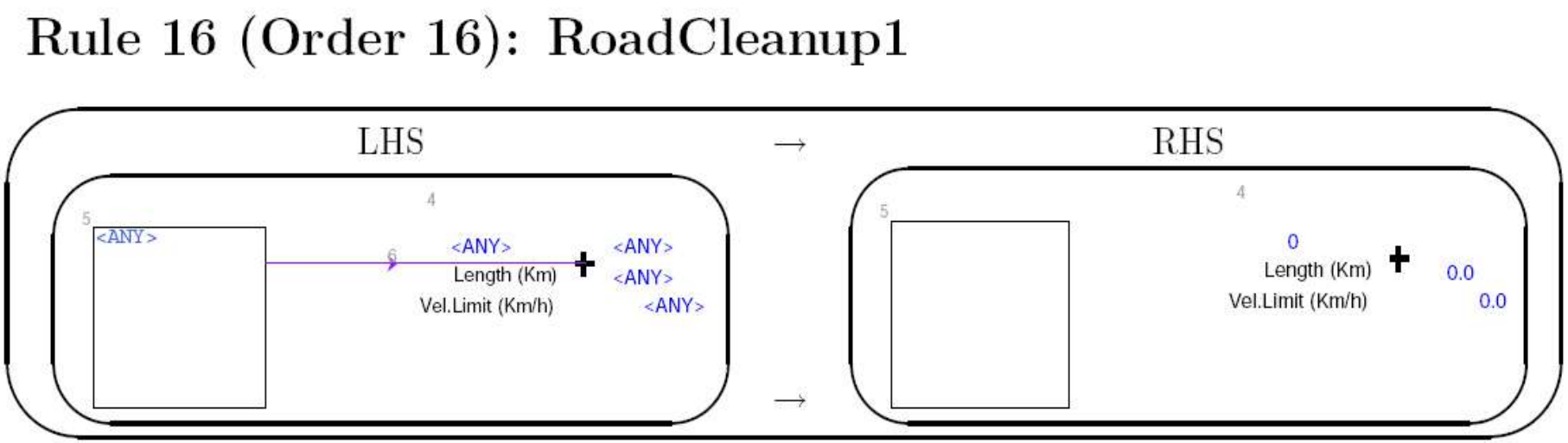


Rule 15: Remove “TrafficLight” (step 2)

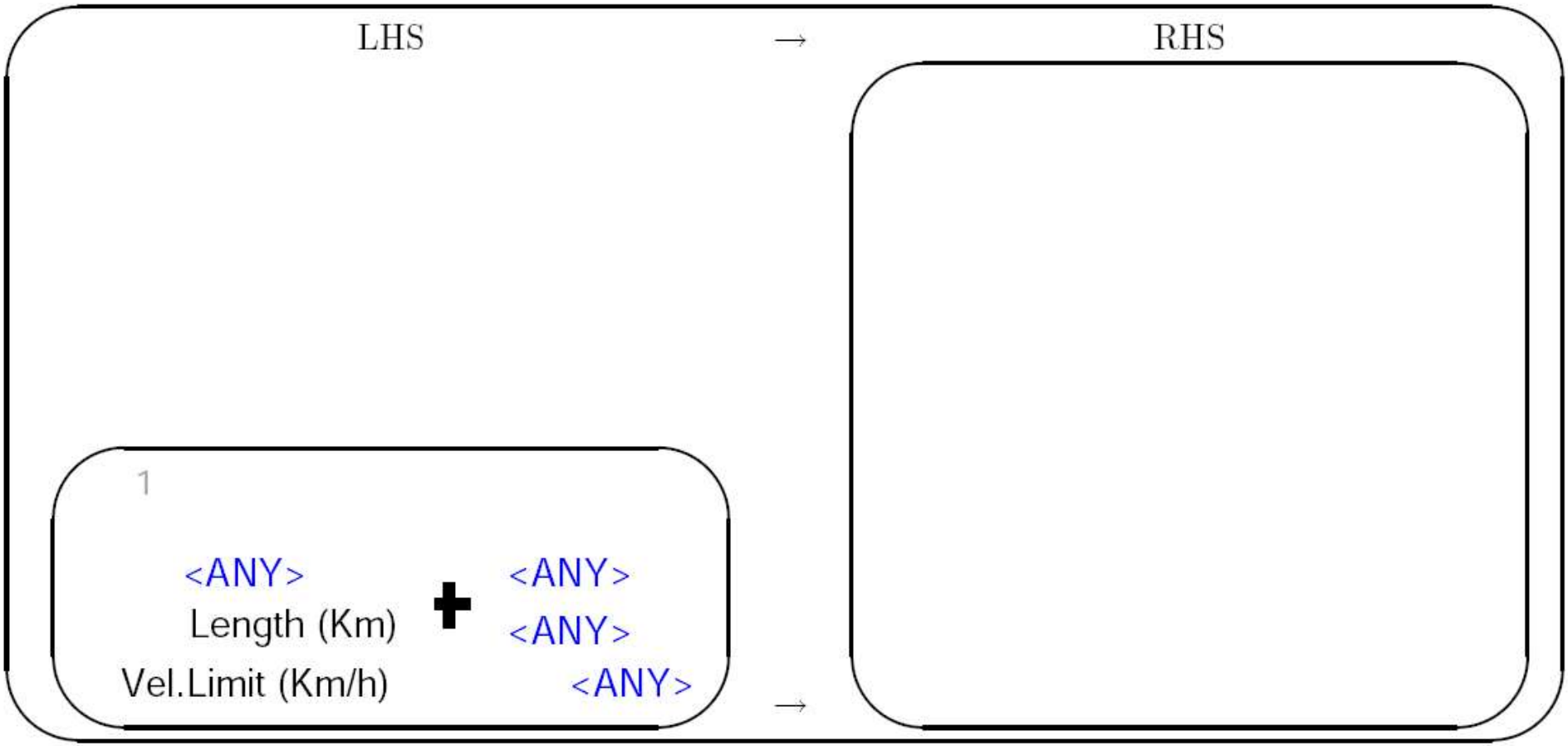
Rule 15 (Order 15): TrafficLightCleanup2



Rule 16: Remove "RoadSection" (step 1)

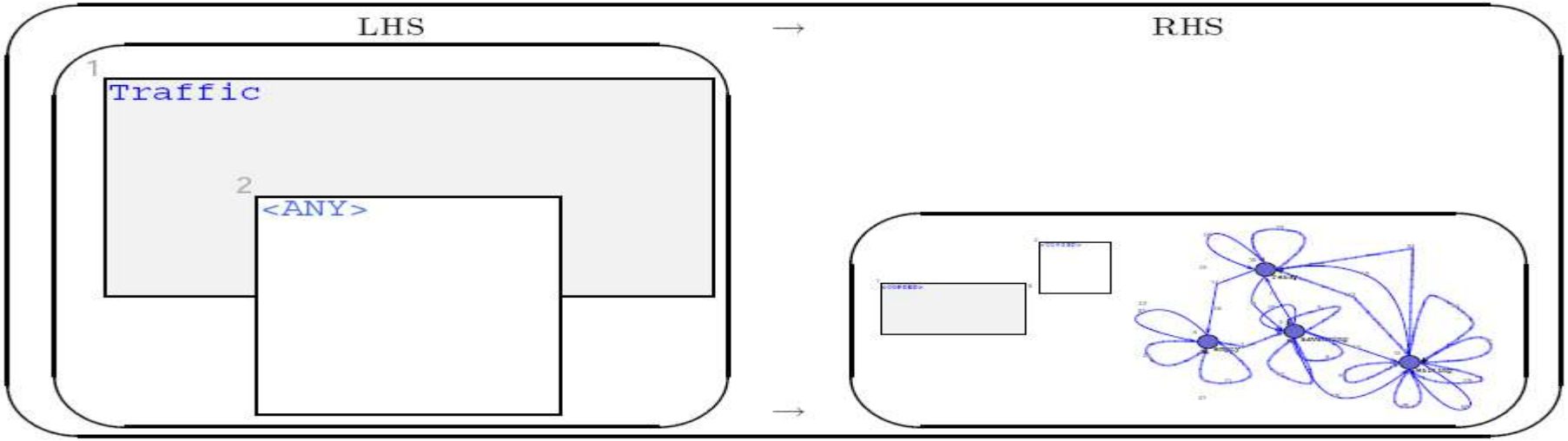


Rule 17: Remove "RoadSection" (step 2)



Rule 18: Connect “Road” Atomic DEVS with “Traffic” Coupled DEVS

Rule 18 (Order 18): ConnectAtomic2Coupled1



Precondition:

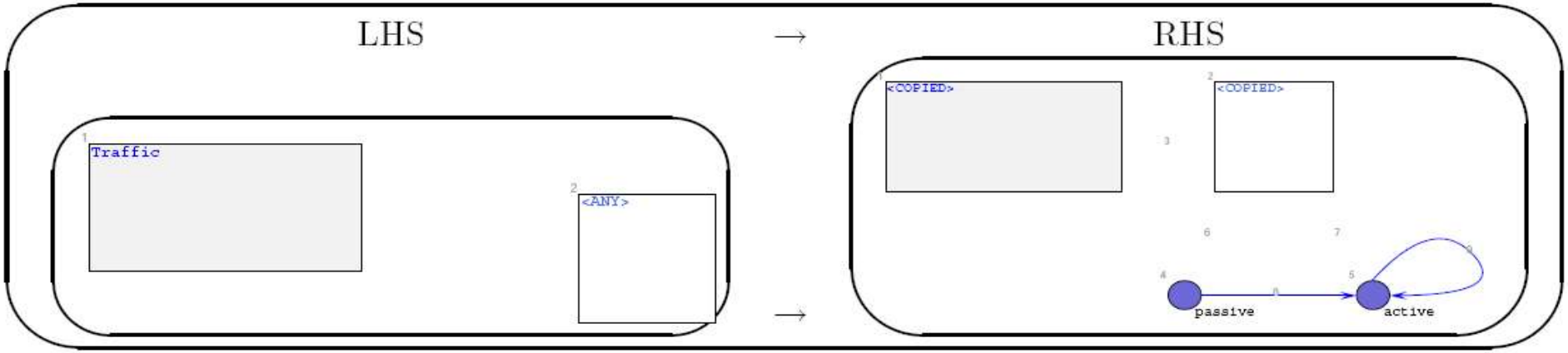
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return (not node.connectedWithCoupled) and (node.model_type == "road")
```

Post action:

```
self.getMatched(graphID, self.LHS.nodeWithLabel(2)).connectedWithCoupled = True
```

Rule 19: Connect "TrafficLight" Atomic DEVS with "Traffic" Coupled DEVS

Rule 19 (Order 19): ConnectAtomic2Coupled2



Precondition:

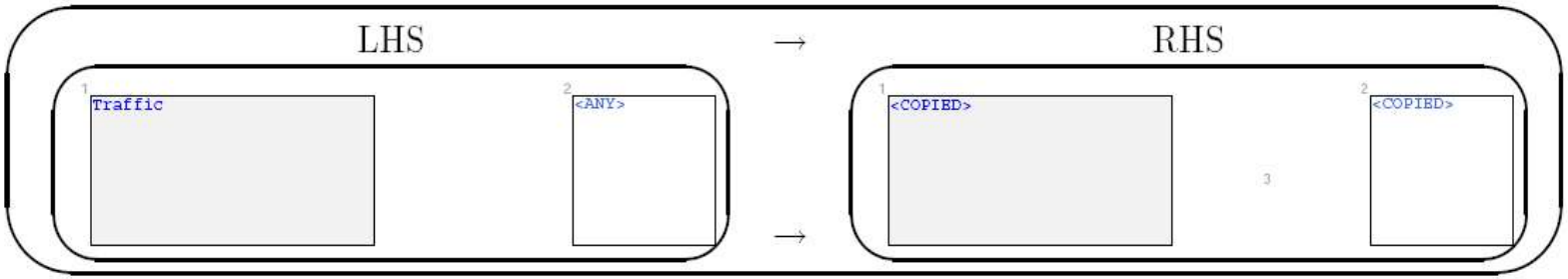
```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return (not node.connectedWithCoupled) and (node.model_type == "trafficLight")
```

Post action:

```
self.getMatched(graphID, self.LHS.nodeWithLabel(2)).connectedWithCoupled = True
```

Rule 20: Connect “Generator” and “Collector” Atomic DEVS with “Traffic” Coupled DEVS

Rule 20 (Order 20): ConnectAtomic2Coupled3

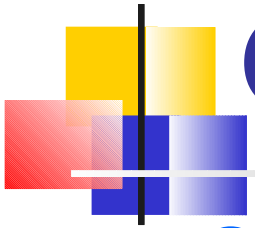


Precondition:

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
return (not node.connectedWithCoupled) and (node.model_type == "generator" or node.model_type == "collector")
```

Post action:

```
self.getMatched(graphID, self.LHS.nodeWithLabel(2)).connectedWithCoupled = True
```



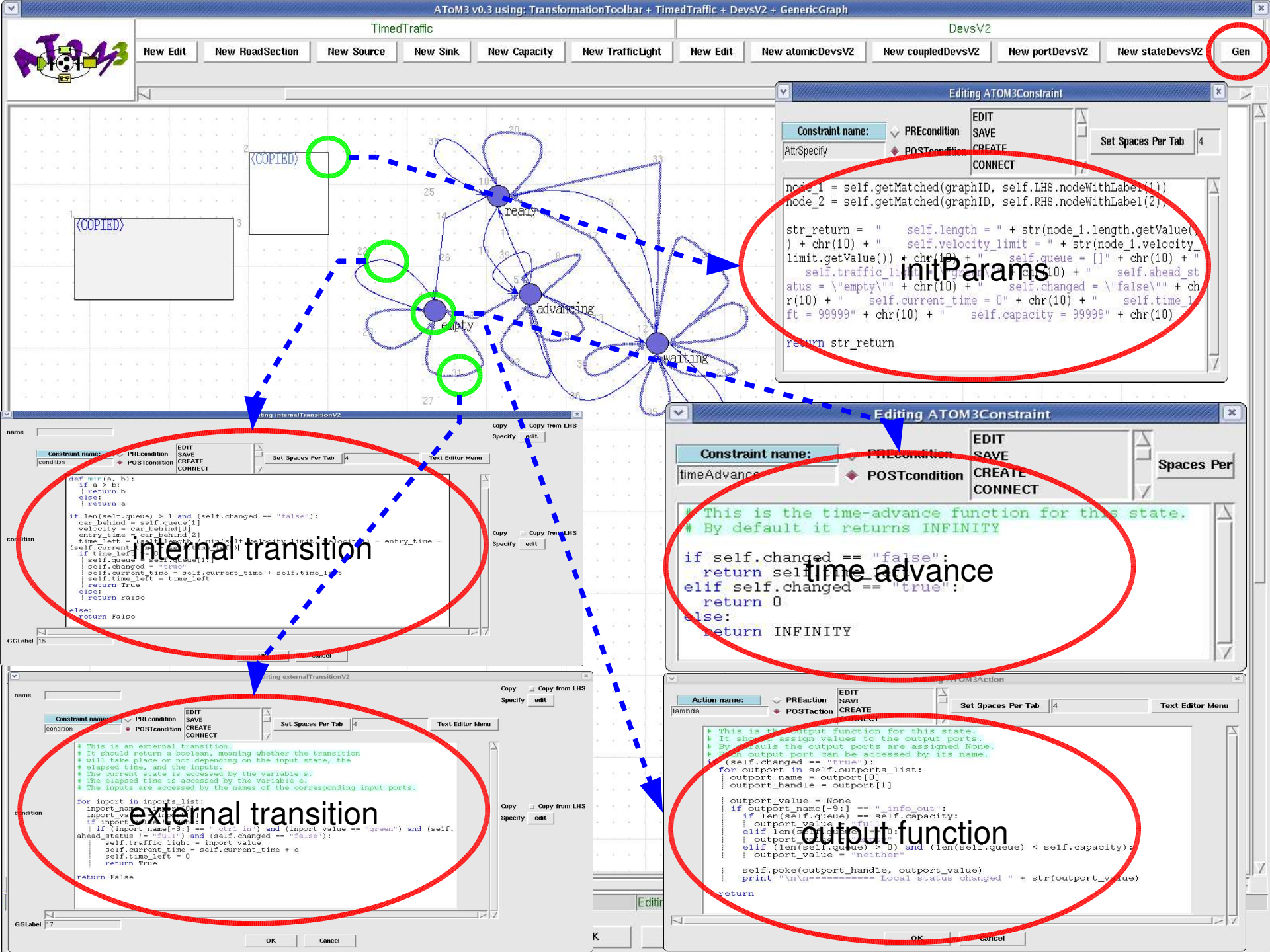
Code Generation

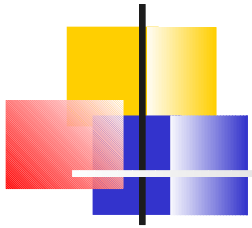
- Simulator in Python (PythonDEVS)

  (Based on the work described in [3].)

  - Write codes to specify initial parameters, time advance function, output function, and conditions for transitions
  - Click “Gen” button in DEVS meta-model toolbar to generate simulators in Python
    - There will be three files at top level: *DEVS.py, Simulator.py, Traffic.py*
    - For each atomic model there will one sub-directory which contains one file, such as *University_Street.py*
  - Run simulation
    - e.g., *$ python Traffic.py 10000*

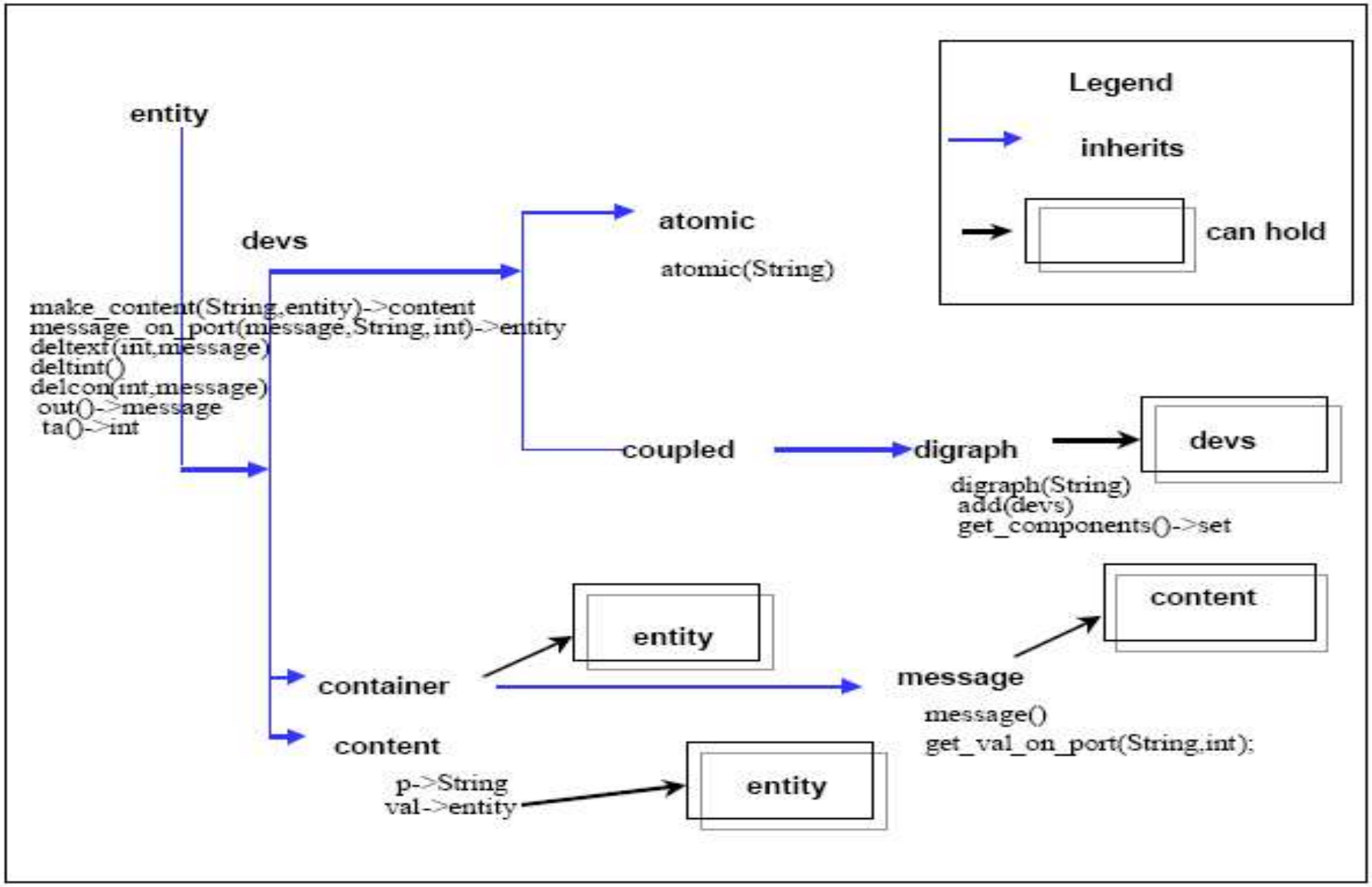
- Simulator in Java (DEVSJAVA)

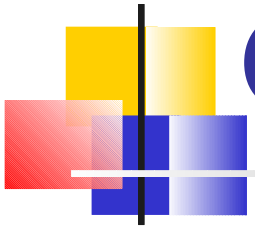

AToM3 v0.3 using: TransformationToolbar + TimedTraffic + DevsV2 + GenericGraph
TimedTraffic
New Edit
New RoadSection
New Source
New Sink
New Capacity
New TrafficLight
DevsV2
New Edit
New atomicDevsV2
New coupledDevsV2
New portDevsV2
New stateDevsV2
Gen
Editing ATOM3Constraint
Constraint name:
PREcondition
POSTcondition
AttrSpecify
EDIT
SAVE
CREATE
CONNECT
Set Spaces Per Tab
initParams
internal transition
time advance
external transition
output function
ready
advancing
empty
waiting
OK
Cancel



- Introduction
  - TimedTraffic Formalism
  - DEVS Formalism
- Map TimedTraffic to DEVS
  - Meta-Modeling
    - TimedTraffic Meta-Model
    - DEVS Meta-model
  - Model Transformation
  - Code Generation
- Demo
- Conclusion

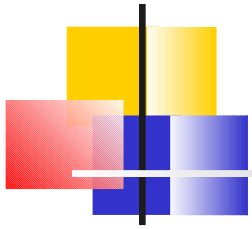
DEVSJAVA Class hierarchy of DEVS classes





Conclusion

- AToM3 is an amazing tool!
- Pure graphical translation from DEVS or other formalism to a real programming language such as Python or Java?
- Using neutral language to describe specific cases in transformation rules?



Thank you!