MoTif:

The Modular Timed Graph Transformation Language

Hans Vangheluwe
Eugene Syriani

Modelling, Simulation and Design Lab
McGill University

McGill

OVERVIEW

- **Introduction**
  - Context
  - Existing Programmed Graph Rewriting Systems
  - Graph transformation in AToM³'s by example
- **The Modular Timed Graph Transformation (MoTif) system**
  - Overview of the DEVS formalism
  - Meta-model, Code generation, Rule compilation & Usage
  - Mimic AToM³ & beyond
- **Conclusion and Future Work**

McGill

CONTEXT

- **Meta-Model + Model Transformation + …**
- **Model Transformation ⇒ Graph Transformation**
- **Types of rule-based Graph Transformations [1]**
  - **Unordered Graph Rewriting: non-deterministic, run till no more rules apply**
  - **Ordered Graph Rewriting: explicit (partial) ordering of rules**
  - **Event-driven Graph Rewriting: external ordering of rules**
- **Ordered Graph Rewriting can be generalized to Programmed Graph Rewriting**

[1] Blostein D., Fahmy H., Grbavec A., Practical Use of Graph Rewriting, Technical Report No. 95-373, 1995.

McGill

IN THE CONTEXT

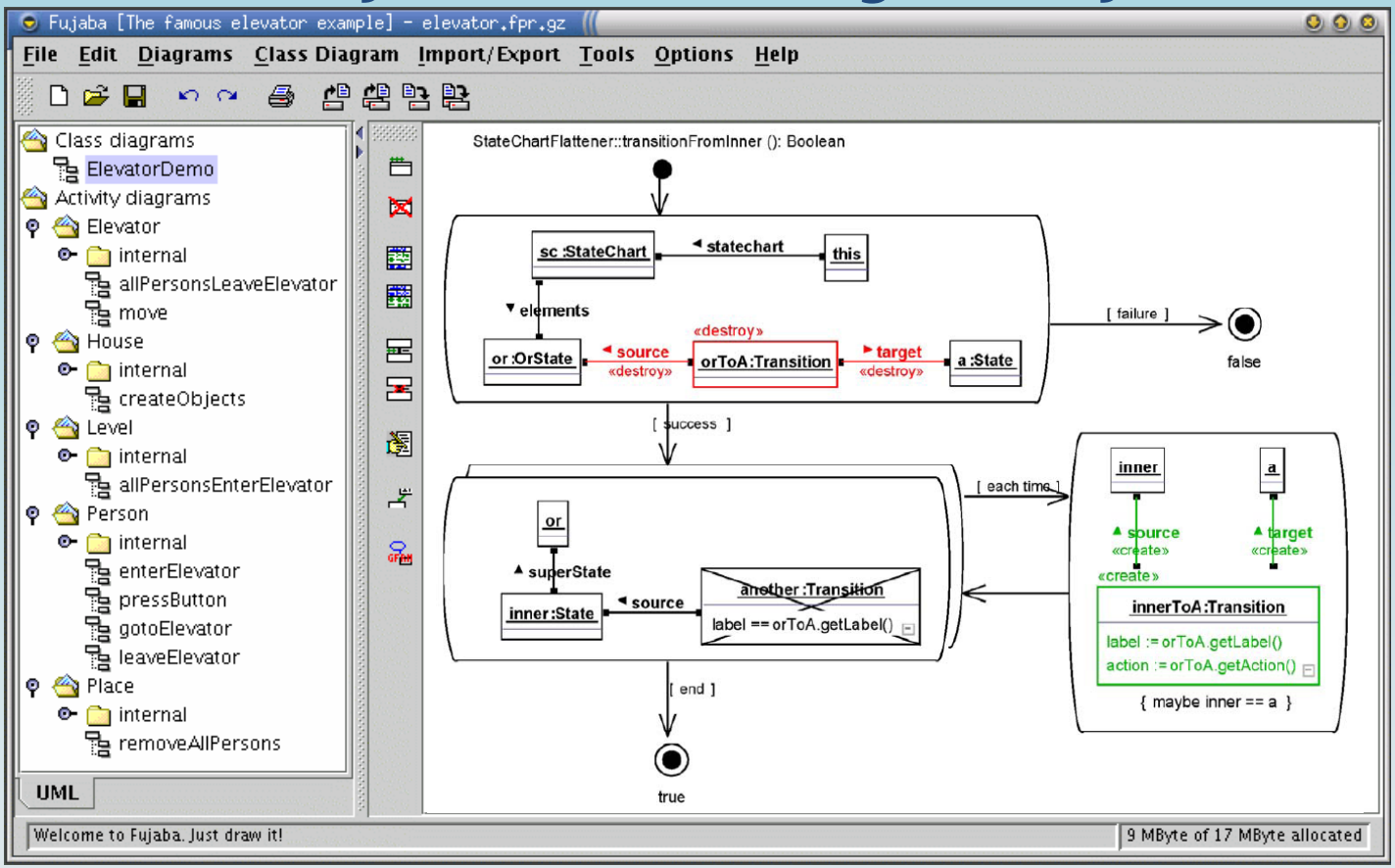
Programmed Graph Rewriting wish list

- Cleanly separate
  1. Transformation entities
  2. Control flow, structure, hierarchy
- Graph transformation control flow primitives [1]
  - Flow: Sequencing, Branching, Looping
  - Structure: Parallelism, Hierarchy
  - Essence: Time, Backtracking

1. Eugene Syriani and Hans Vangheluwe: Programmed Graph Rewriting with DEVS. In: Manfred Nagl and Andy Schürr (eds.), *International Conference on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007)*. Lecture Notes in Computer Science. Springer-Verlag, Kassel (2007).

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

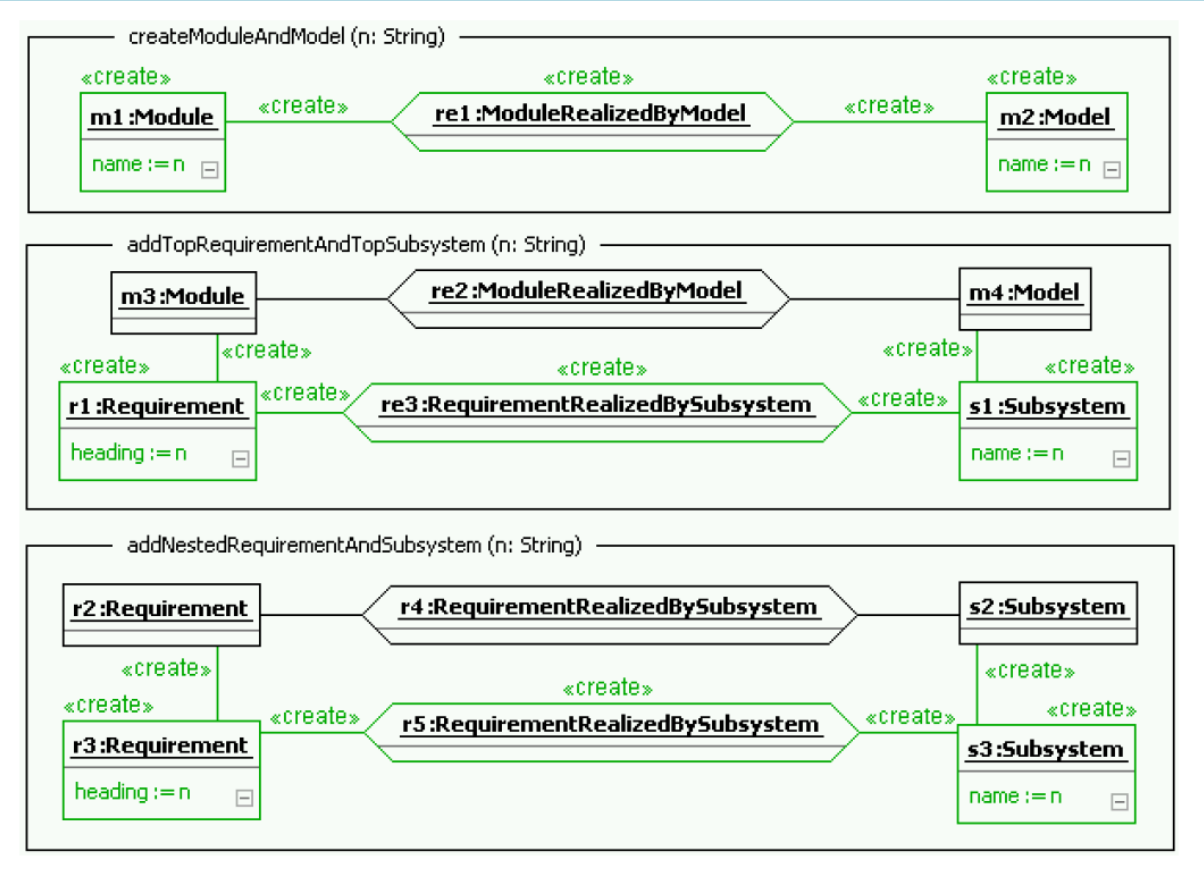
From UML to JAVA And Back Again (FUJABA) [4,5]



[4] Nickel U., Niere J., Zündorf A. Tool demonstration: The FUJABA environment, *Proceedings of ICSE*, ACM Press, pp. 742-745, 2000.
[5] Fischer T. et al. Story Diagrams: an new graph grammar language based on UML and JAVA, *Proceedings of ESEC*, LNCS 1764, pp. 1-21, 2000.

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

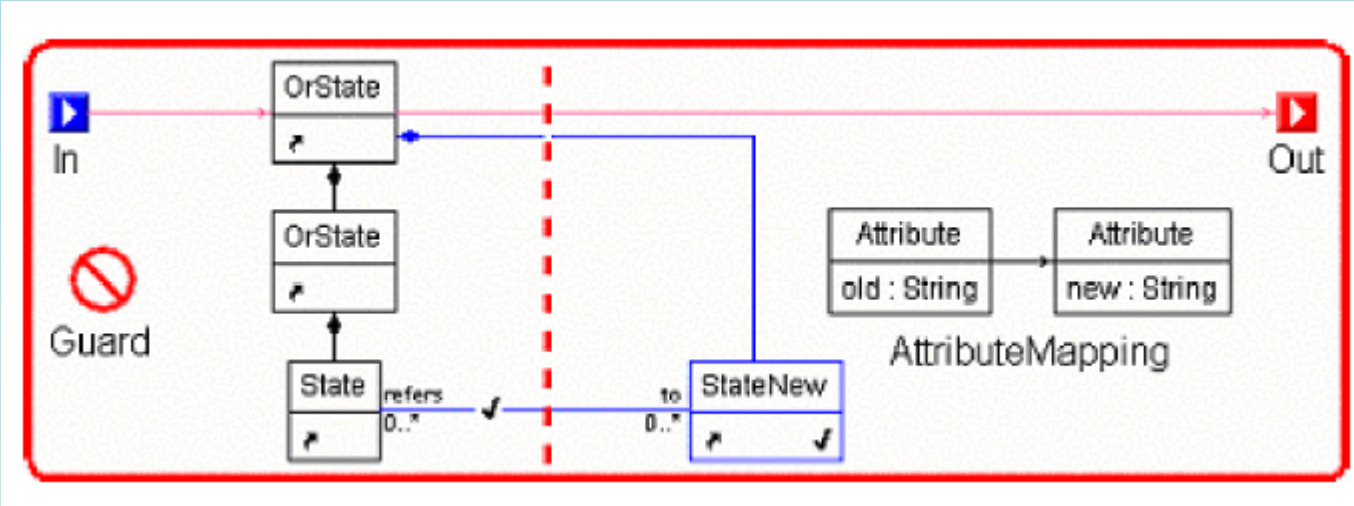
MOFLON [6]

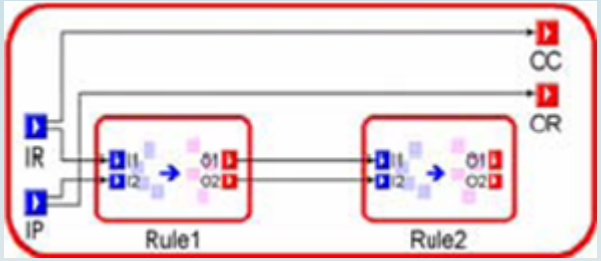


[6] Amelunxen C. et al. Moflon: A standard compliant metamodeling framework with graph transformations, *Proceedings of MDA – Foundations and Applications*, LNCS 4066, pp. 361-375, 2006.

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

Graph Rewriting and Transformation (GReAT) [7,8,9]

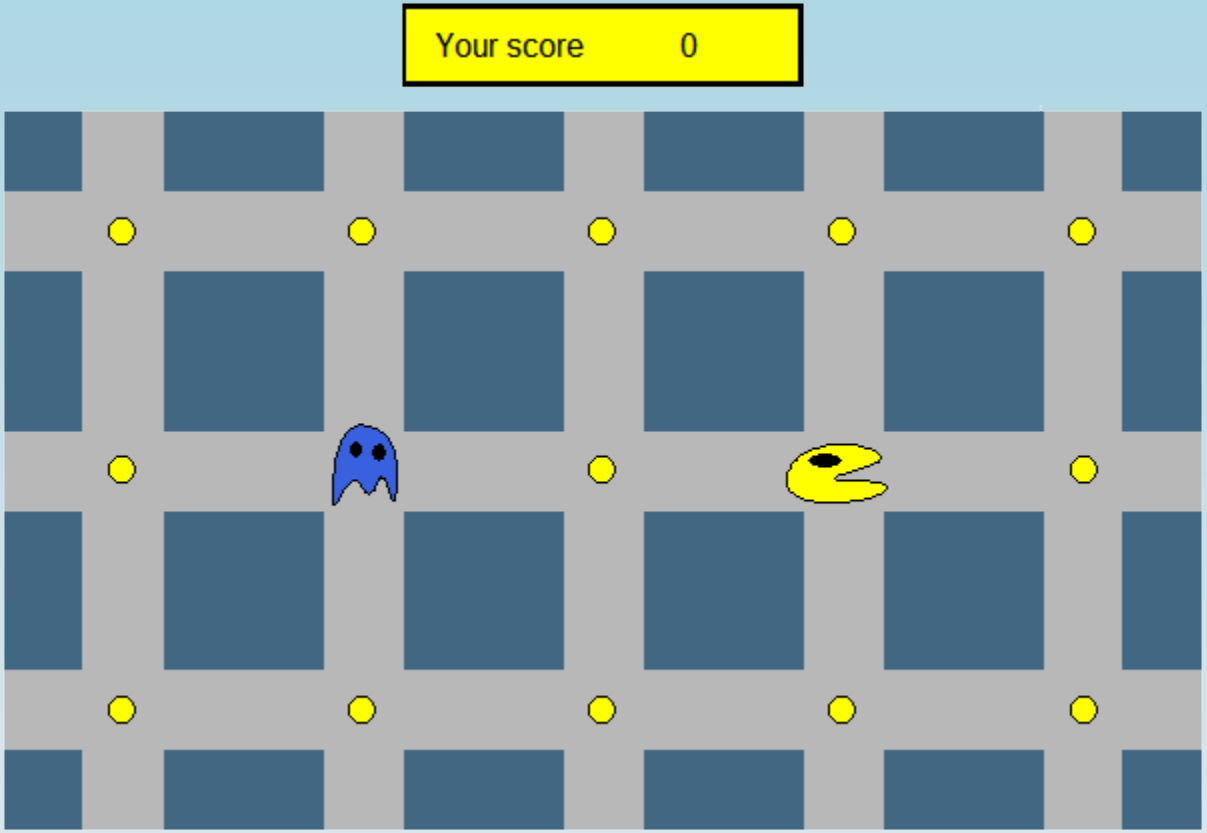




[7] Vizhanyo A. et al., Towards generation of high-performance transformations, *Proceedings of GPCE*, LNCS 3286 pp. 298-396, 2004.
[8] Agrawal A., Metamodel based model transformation language, *Proceedings of OOPSLA*, ACM Press pp. 386-397, 2003.
[9] Agrawal A., The design of a language for model transformations, *SoSym 5*, pp. 261-288, 2005.

RUNNING EXAMPLE WITH AToM³ [12]

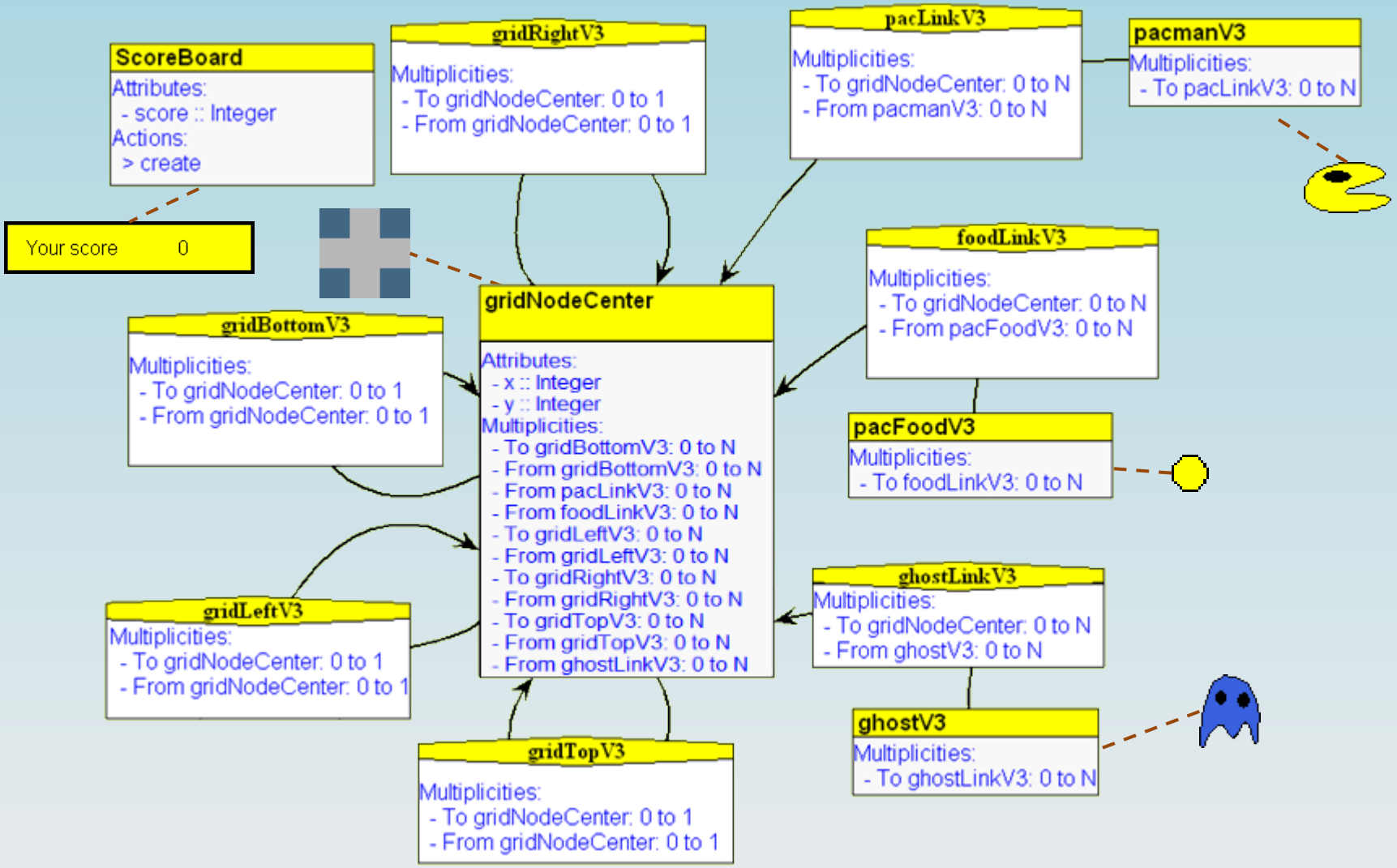
Simplified PacMan formalism [13]



[12] de Lara J., Vangheluwe H., AToM³: A tool for multi-formalism and meta-modelling, *LNCS* (2002), 174-188
[13] Heckel R., Graph Transformation in a nutshell, *ENTCS* (2006), 187-198

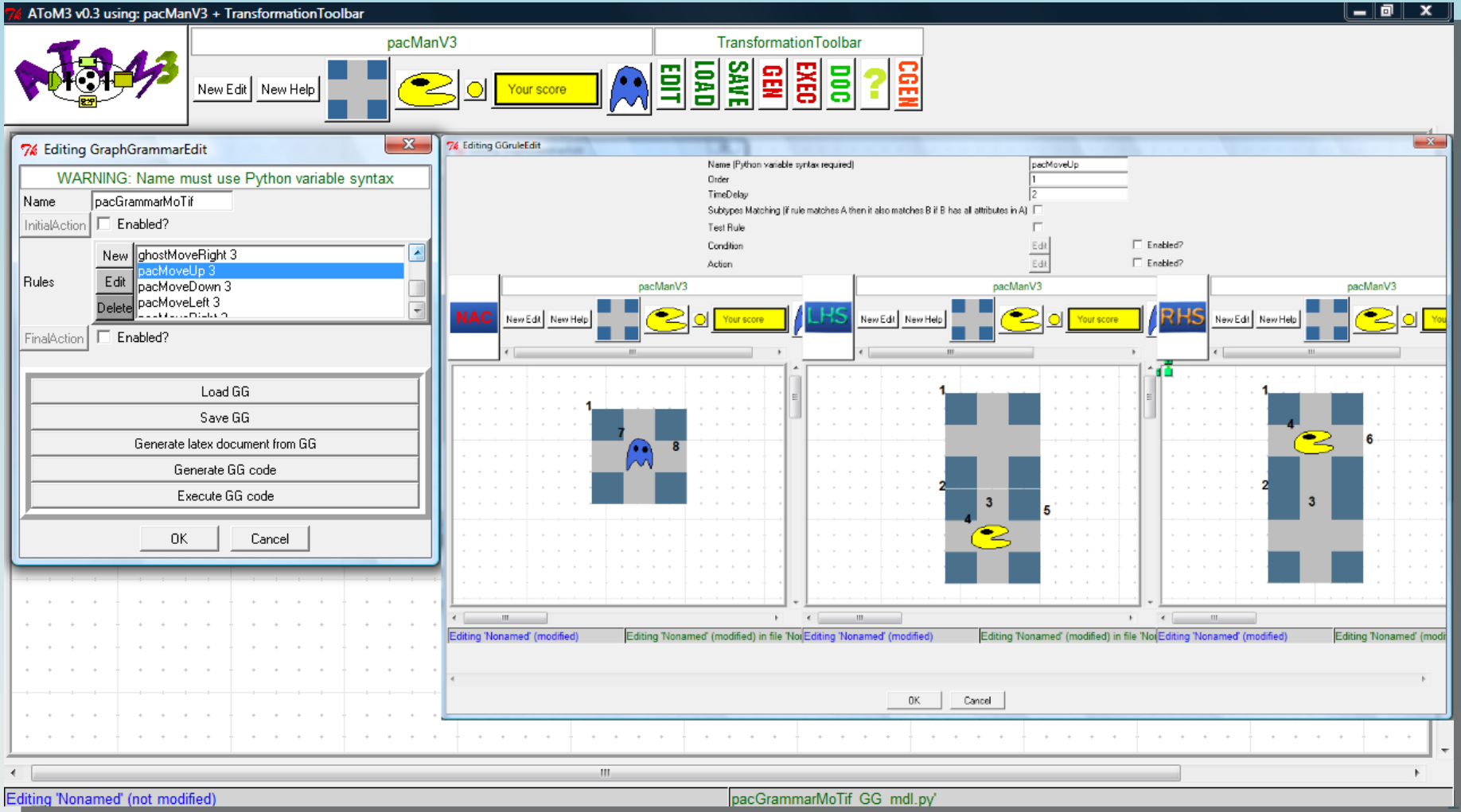
RUNNING EXAMPLE WITH AToM³

Build the Meta-Model of the PacMan formalism



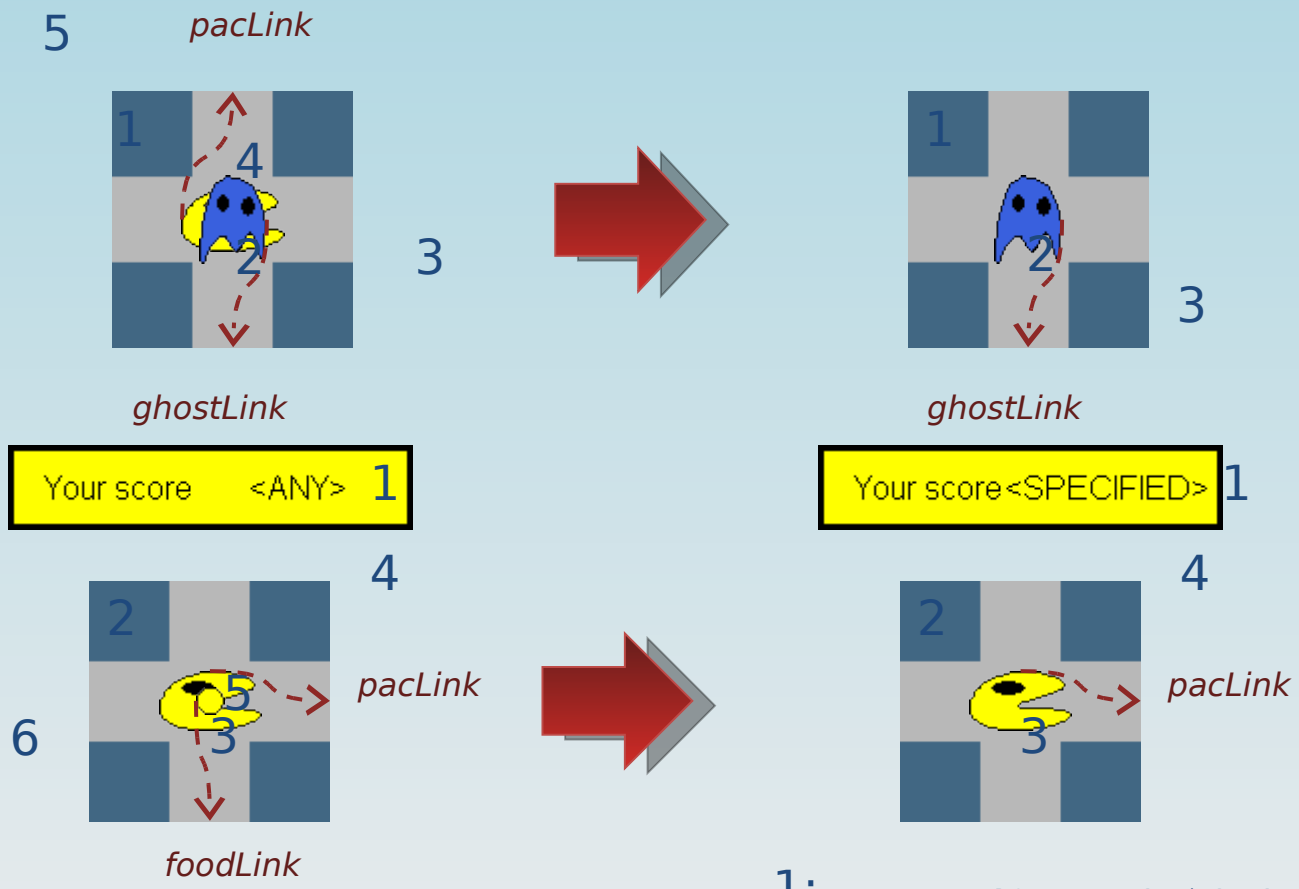
RUNNING EXAMPLE WITH AToM³

Model the Graph Transformation



RUNNING EXAMPLE WITH ATOM³

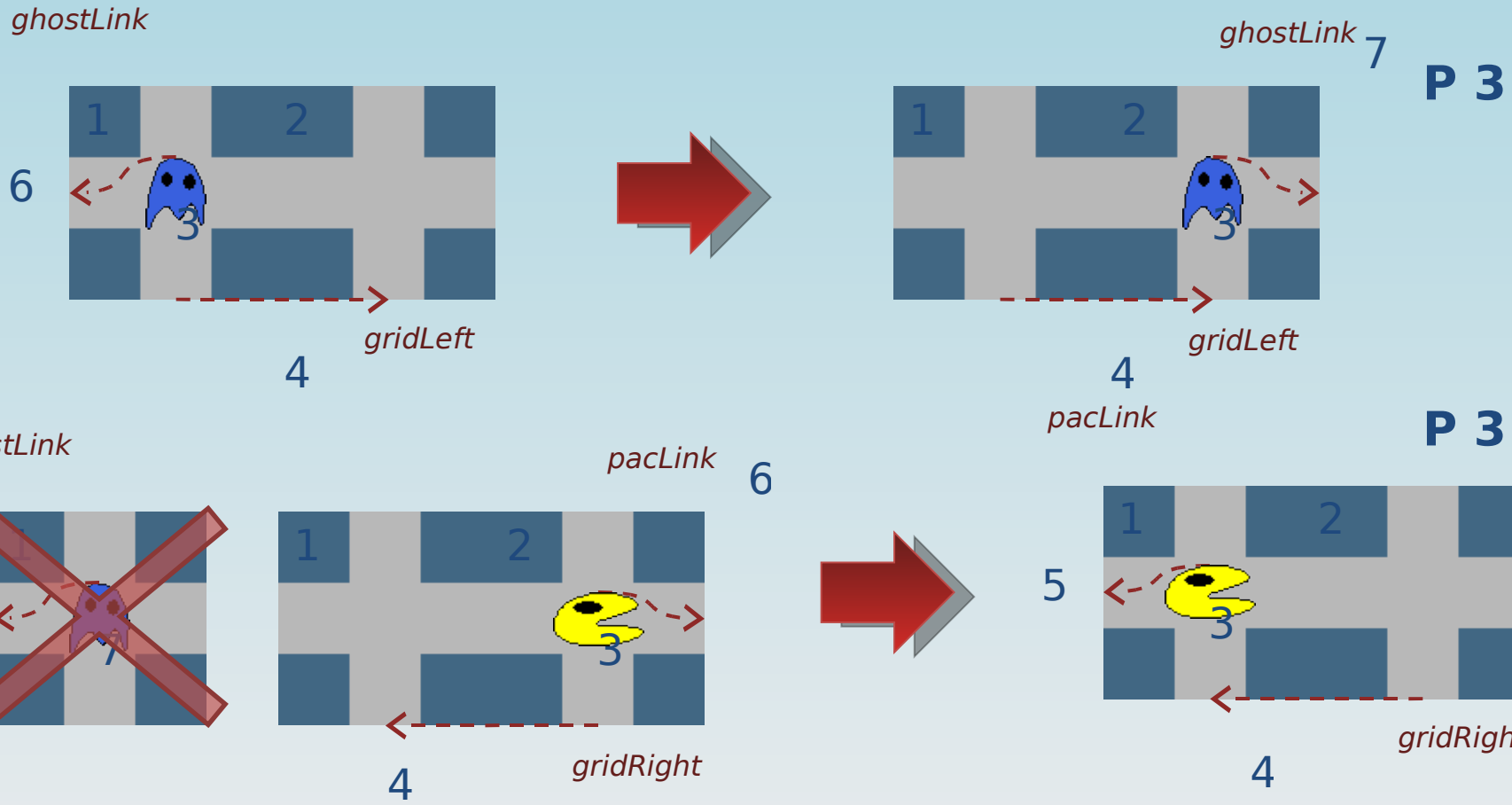
Graph Transformation Rules



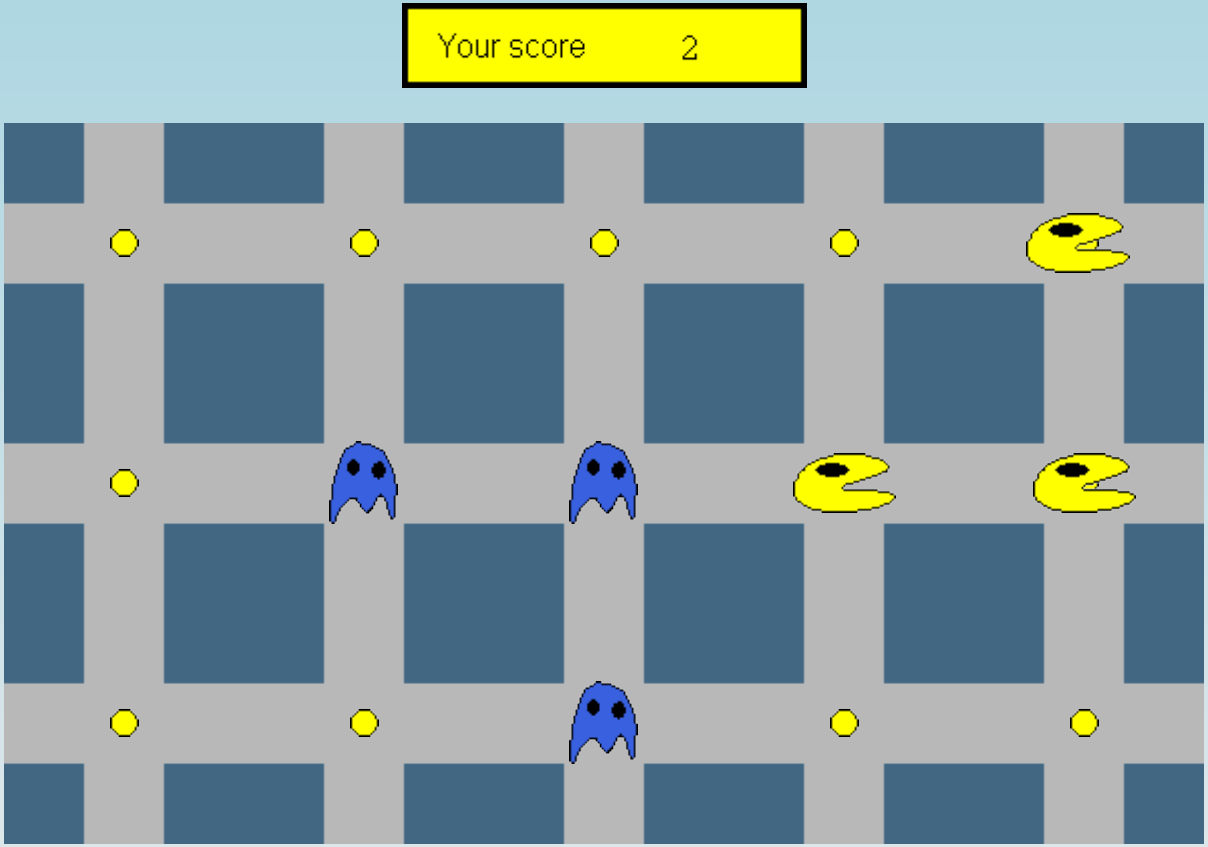
1: `return self.LHS.nodeWithLabel(1).score + 1`

RUNNING EXAMPLE WITH ATOM³

Graph Transformation Rules



RUNNING EXAMPLE WITH AToM³



McGill

RUNNING EXAMPLE WITH ATOM³

- **Capture execution trace:**
  - **Keep log of used rules**

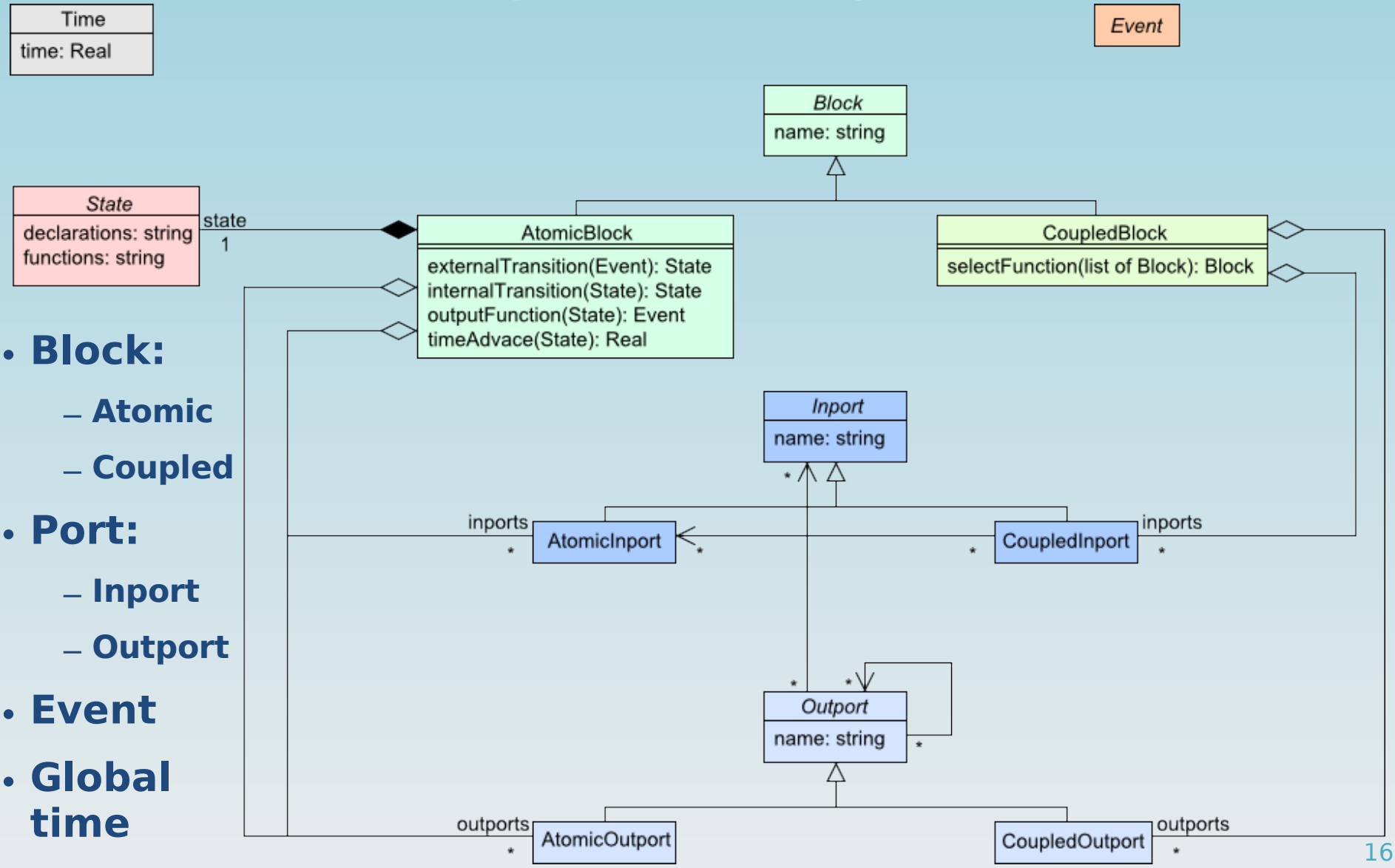
```
Rules Used
>1: PacManUp
>2: Eat
>3: PacManLeft
>4: Eat
>5: PacManDown
>6: Eat
>7: PacManDown
>8: Eat
>9: GhostRight
>10: GhostDown
>11: Kill
>12: GhostUp
>13: GhostLeft
>14: GhostUp
>15: GhostLeft
>16: GhostRight
```

OVERVIEW OF THE DEVS FORMALISM

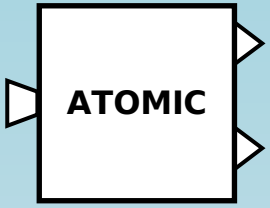
- Bernard Zeigler, late ‘70s
- The Discrete Event System Specification [1] (DEVS) formalism
- DEVS is compositional
- Foundation for compositional modelling and simulation of discrete event systems
- DEVS:
  - Blocks
  - Ports
  - Events
- Semantics: Parallel composition of blocks/models

OVERVIEW OF THE DEVS FORMALISM

- **Block:**
  - **Atomic**
  - **Coupled**
- **Port:**
  - **Inport**
  - **Outport**
- **Event**
- **Global time**

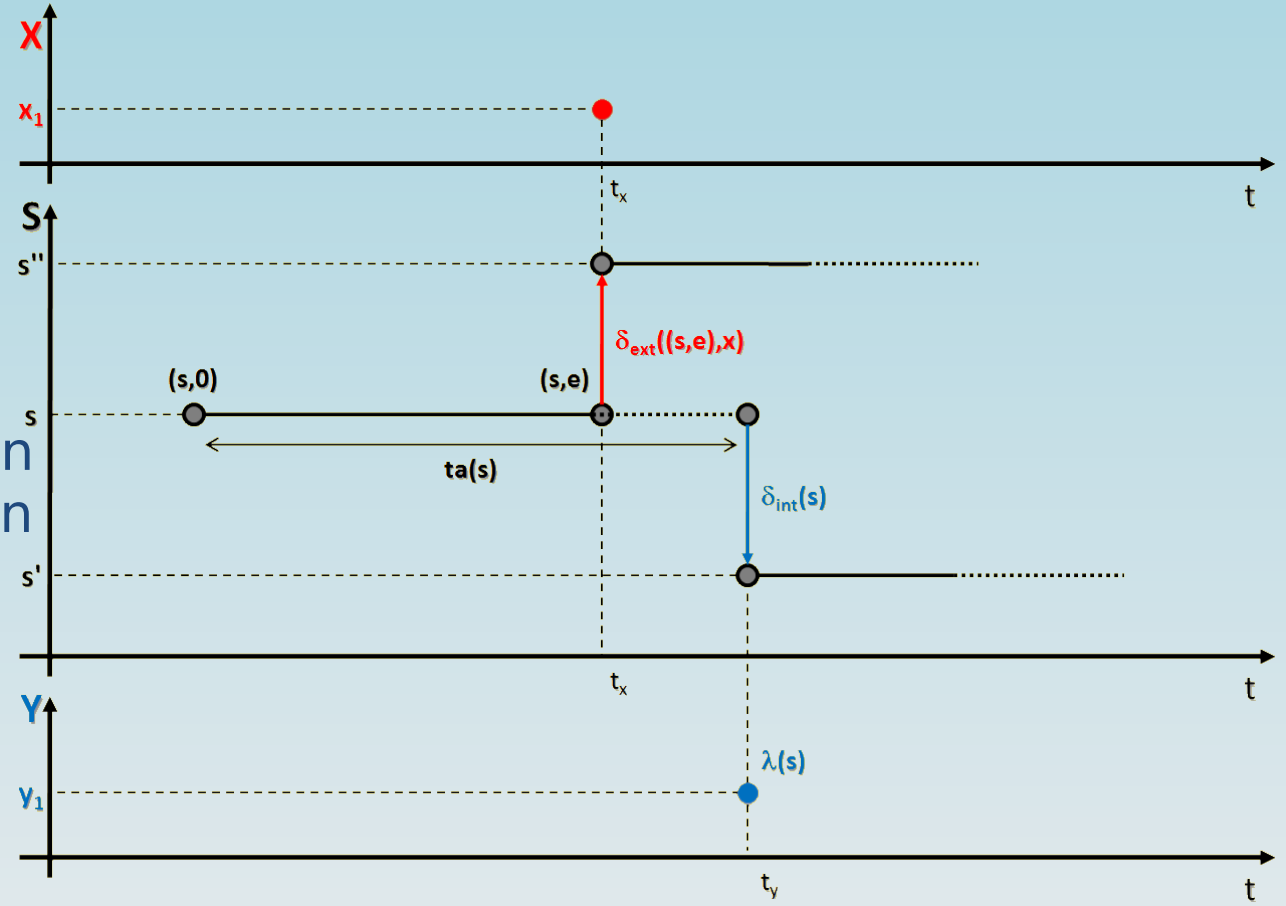


OVERVIEW OF THE DEVS FORMALISM



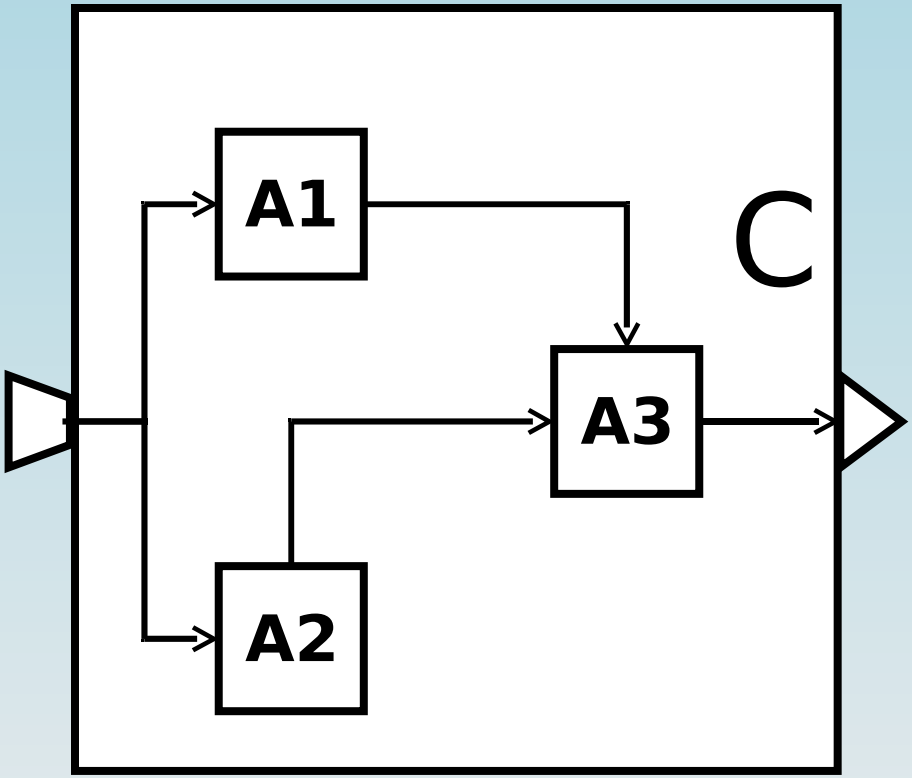
Atomic DEVS:

- Time Advance
- Output Function
- Internal Transition
- External Transition



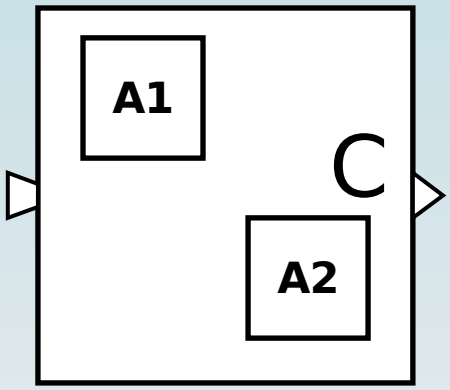
OVERVIEW OF THE DEVS FORMALISM

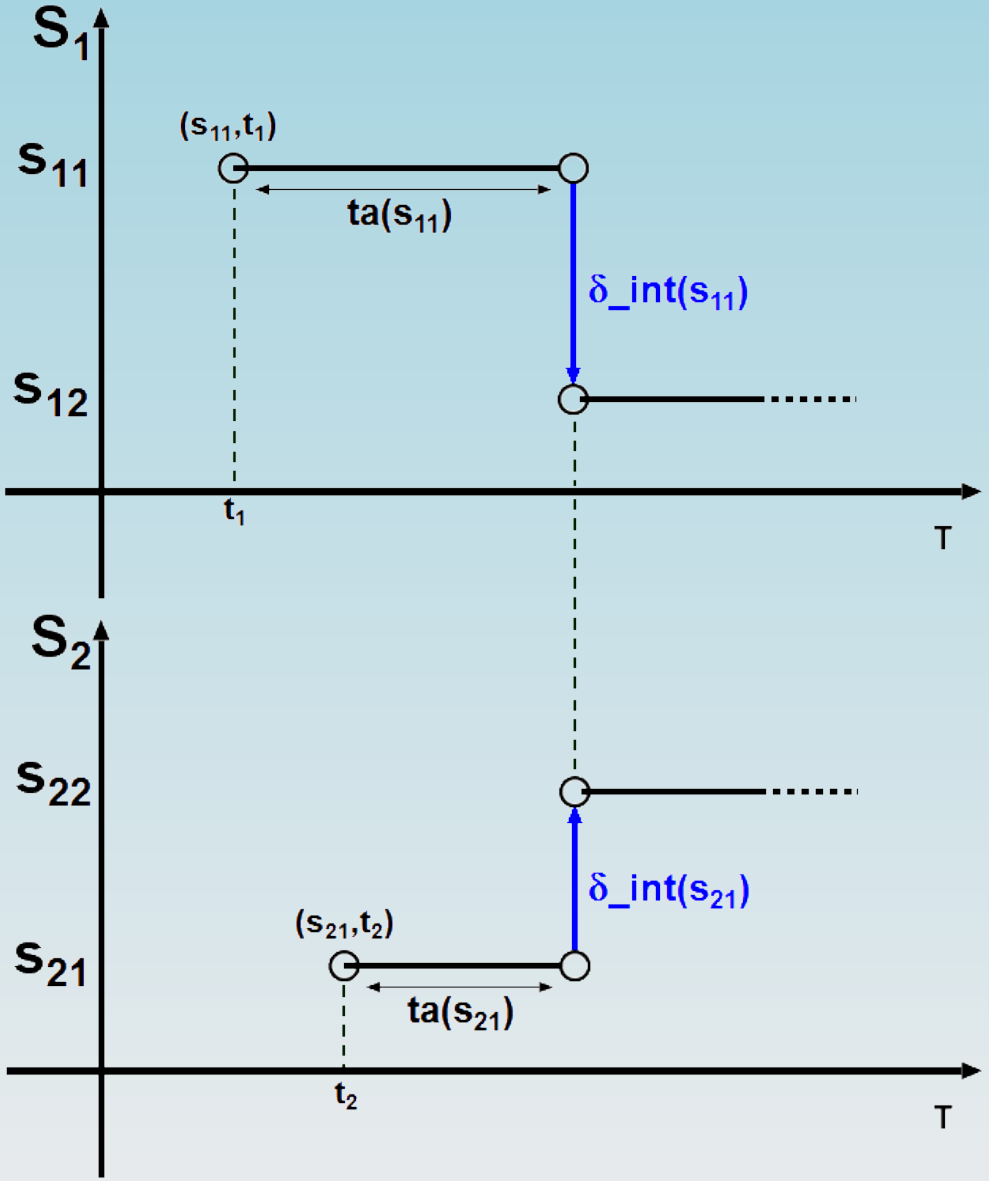
Coupled DEVS



OVERVIEW OF THE DEVS FORMALISM

Coupled DEVS:
– Select Function





OVERVIEW OF THE DEVS FORMALISM

Our implementation: pythonDEVS

```
class AExample(AtomicDEVS):
  def __init__(self):
    self.state = ExampleState()
    self.in = self.addInPort()
    self.out = self.addOutPort()

  def extTransition(self):
    X = self.peak(self.in)
    ...
    return self.state

  def intTransition(self):
    ...
    return self.state

  def outputFnc(self):
    ...
    self.poke(self.out, Y)

  def timeAdvance(self):
    return 1
```

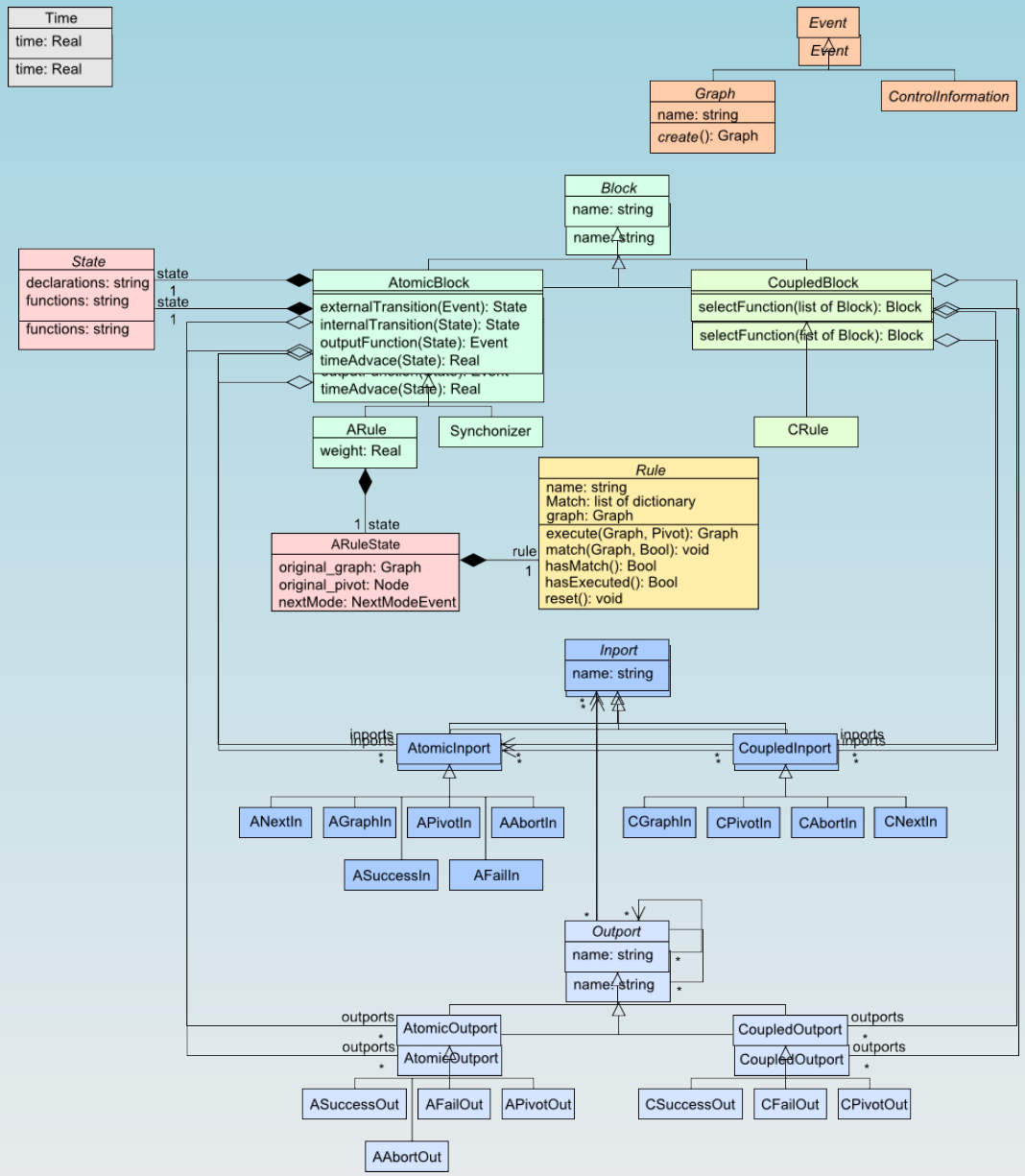
```
class CExample(CoupledDEVS):
  def __init__(self):
    self.M1 = self.addSubModel(Example())
    self.M2 = self.addSubModel(Example())
    self.connectPorts(self.M1.out, self.M2.in)

  def select(self, immList):
    return immList[0]
```

McGill

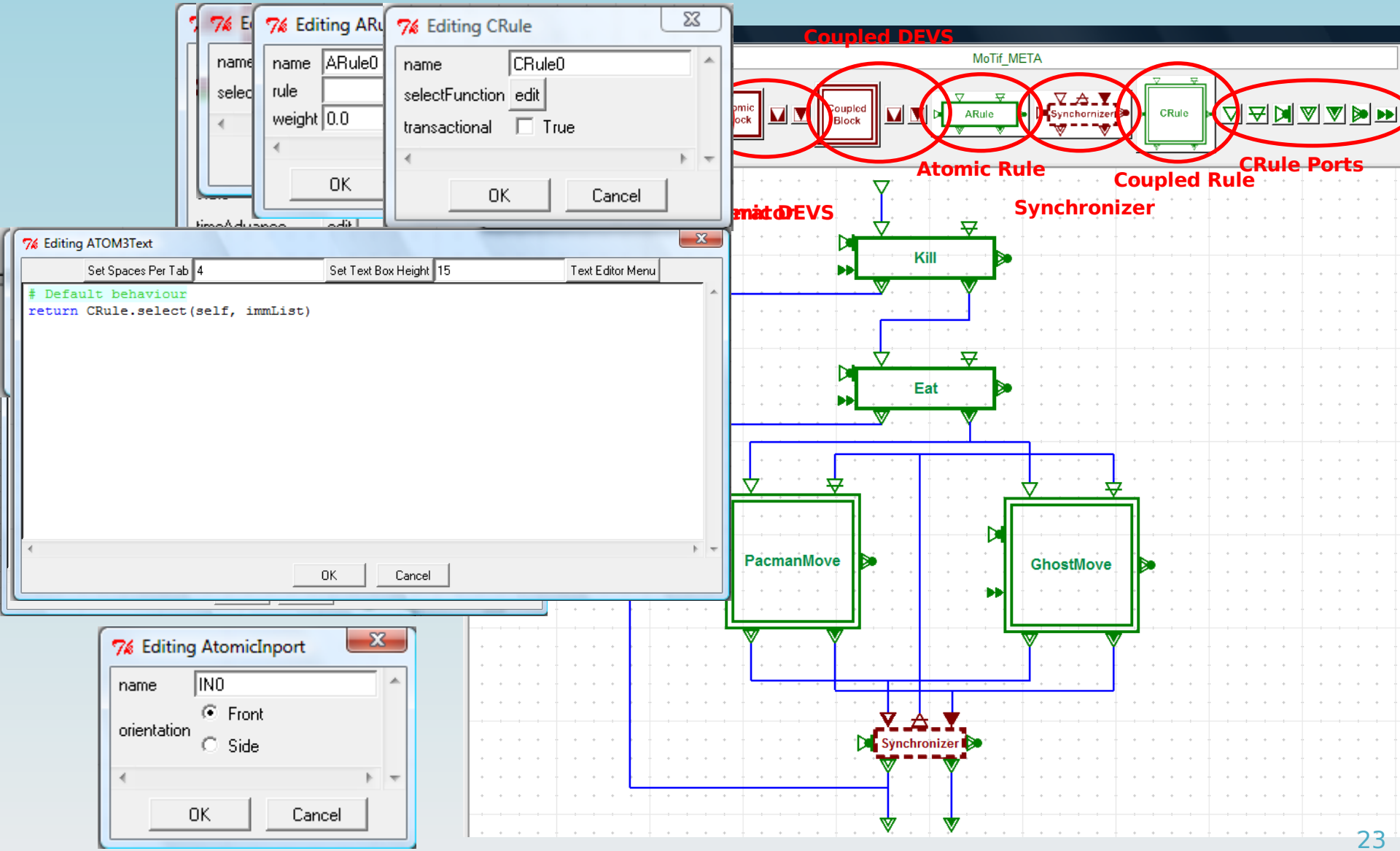
MoTif

- **DEVS blocks**
  - **Atomic block: encapsulate the graph rewriting rule**
  - **Coupled block: encapsulate a structured collection of rules (graph transformation)**
- **Events**
  - **Inport: receive the host graph**
  - **Outport(s): send the transformed graph**

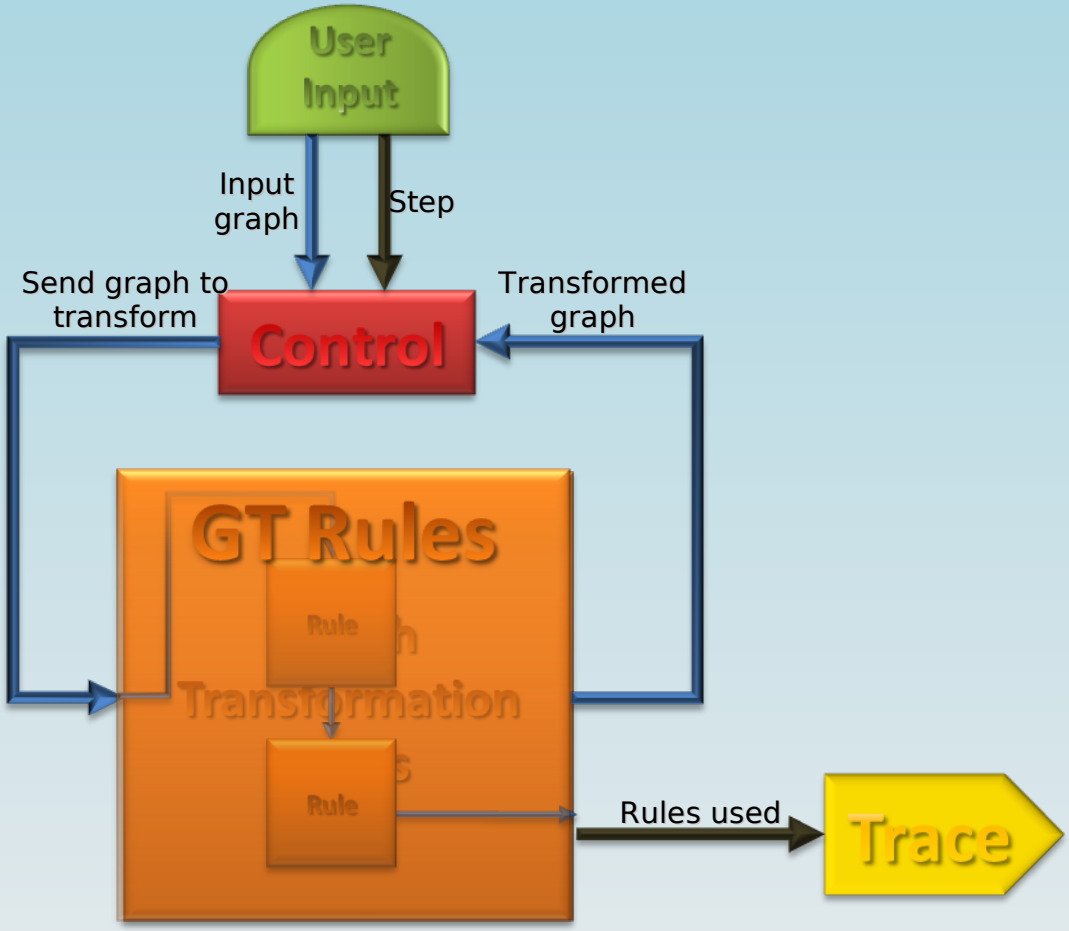
MOTIF META-MODEL



MOTIF MODELLING ENVIRONMENT

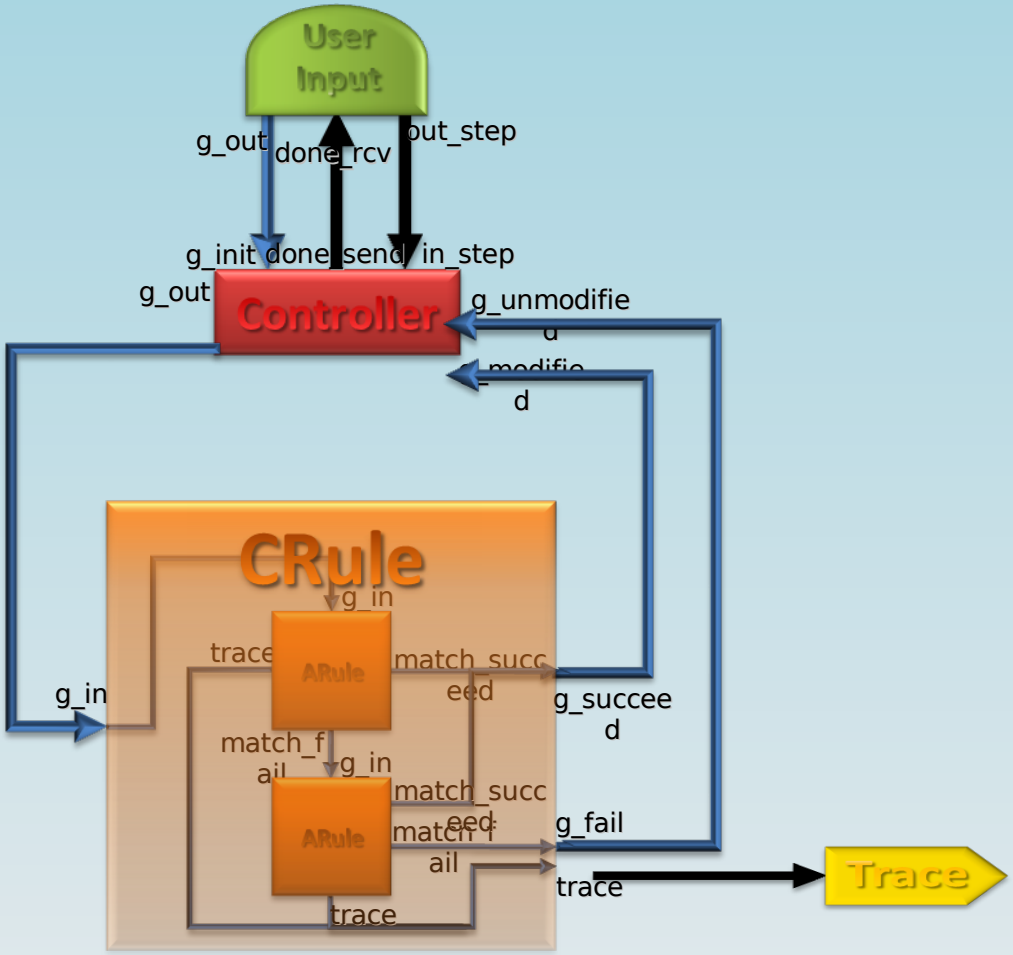


MODEL AToM³'S GRAPH TRANSFORMATION ENGINE



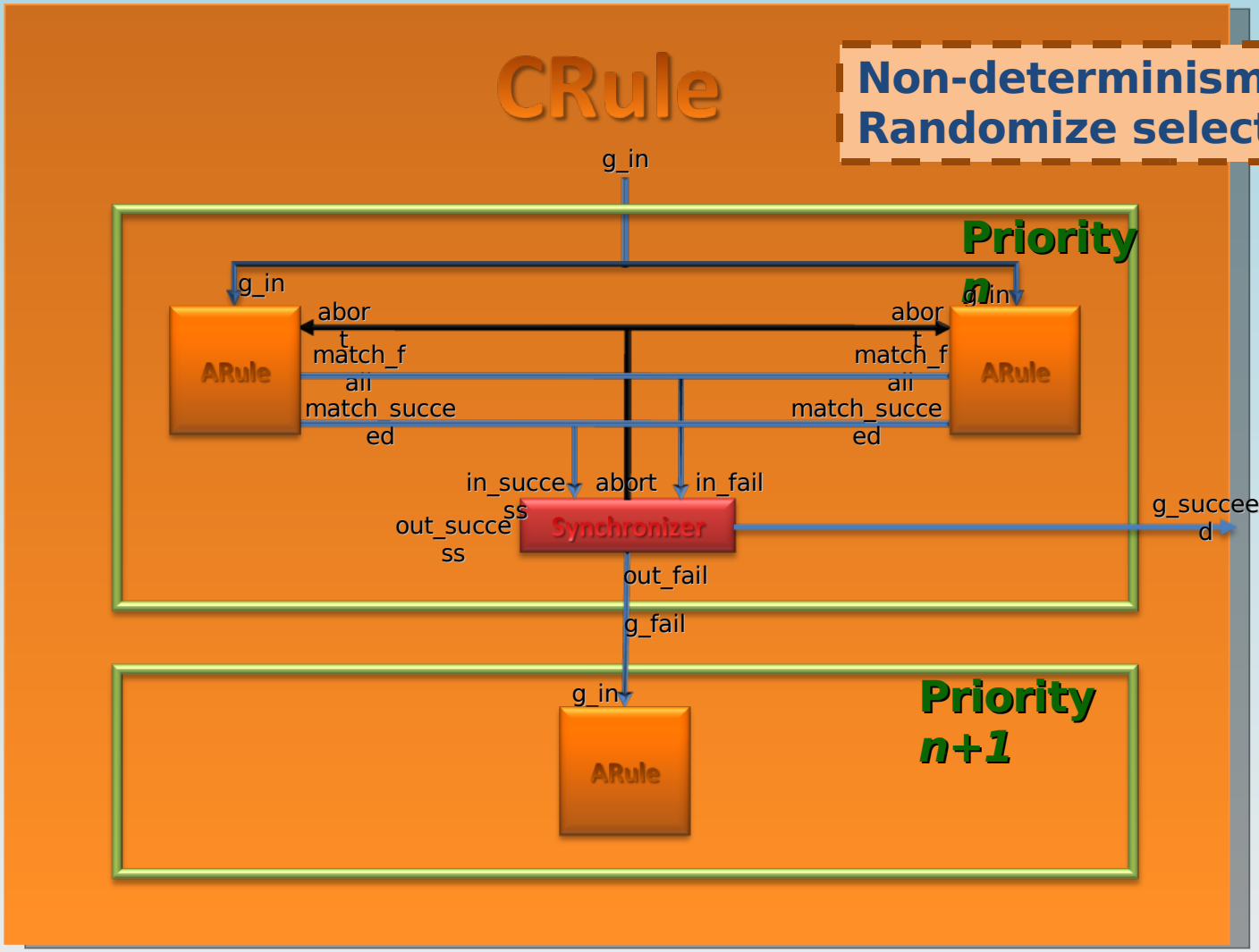
McGill

GRAPH TRANSFORMATION ENGINE



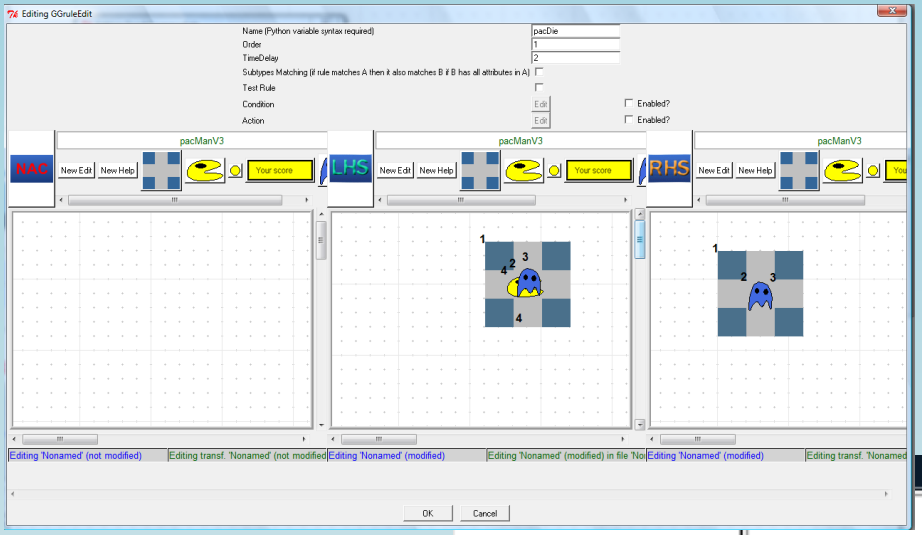
McGill

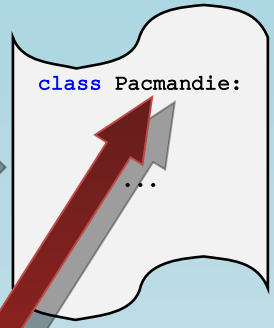
MANAGING PRIORITIES

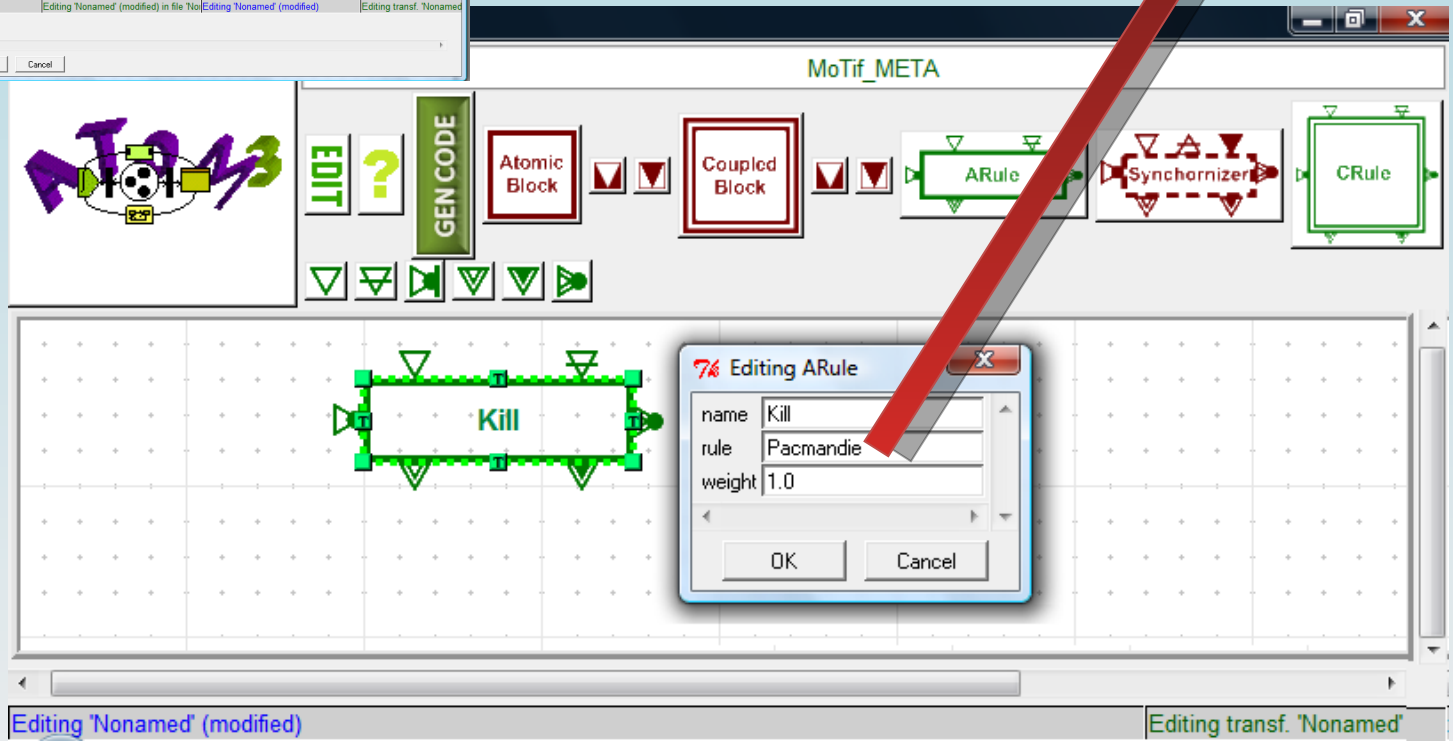


McGill

MoTif & AToM³

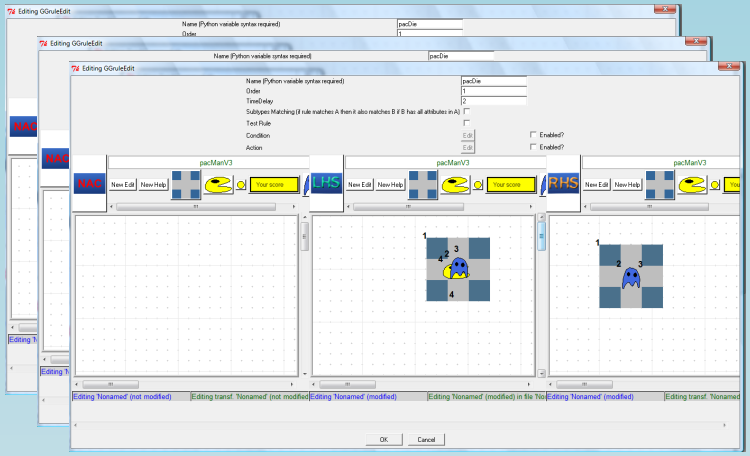






McGill

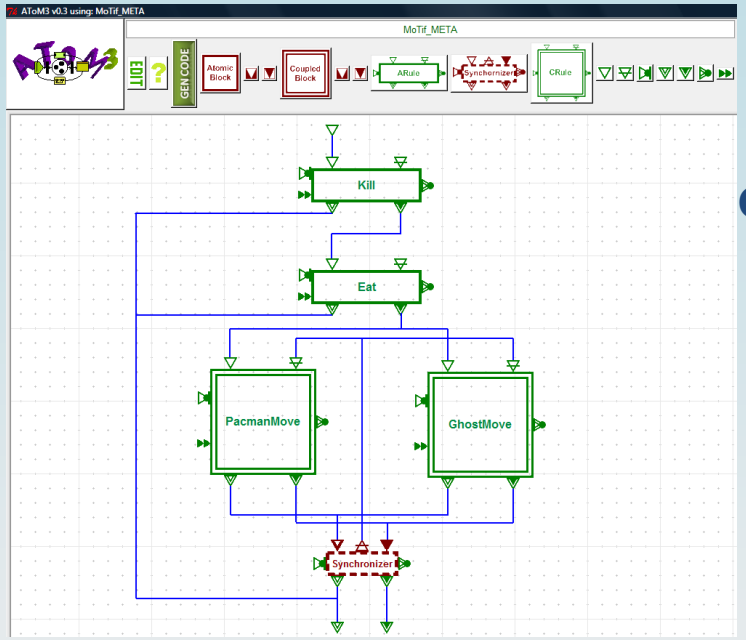
MoTif EXECUTION



COMPILE

```
class
Pacmanmo
    def

class
Pacmandie:

def match():
    ...
...
```

IMPORT



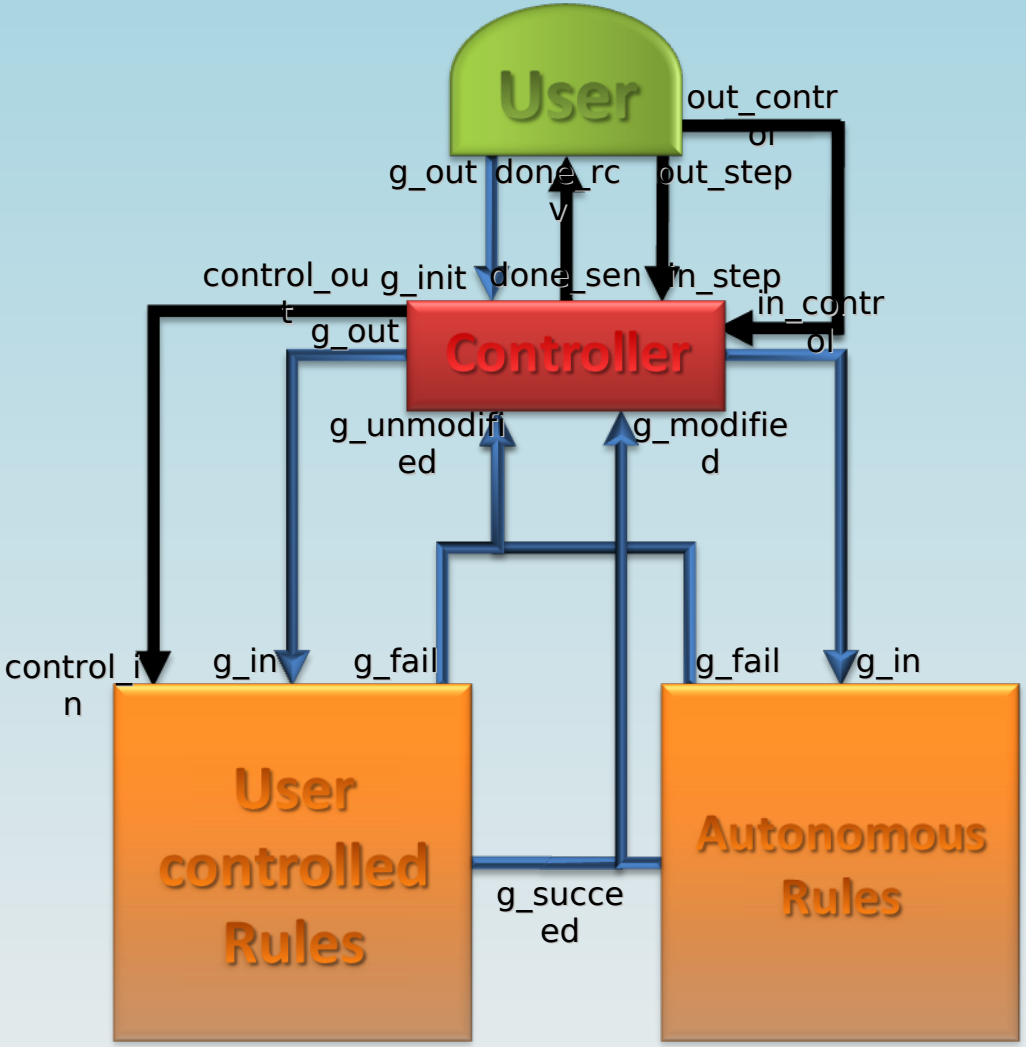
GENERATE

```
class Kill(ARule):
    def __init__(self):
        ARule.__init__(self,
name='Kill')
        self.state =
ARuleState(Pacdie())

    def
weightFunction(self):
        return 1.0
```

SIMULATE

```
Rules Used
>1:  PacManUp
>2:  Eat
>3:  PacManLeft
>4:  Eat
>5:  PacManDown
>6:  Eat
>7:  PacManDown
>8:  Eat
>9:  GhostRight
>10:  GhostDown
>11:  Kill
>12:  GhostUp
>13:  GhostLeft
>14:  GhostUp
>15:  GhostLeft
>16:  GhostRight
```

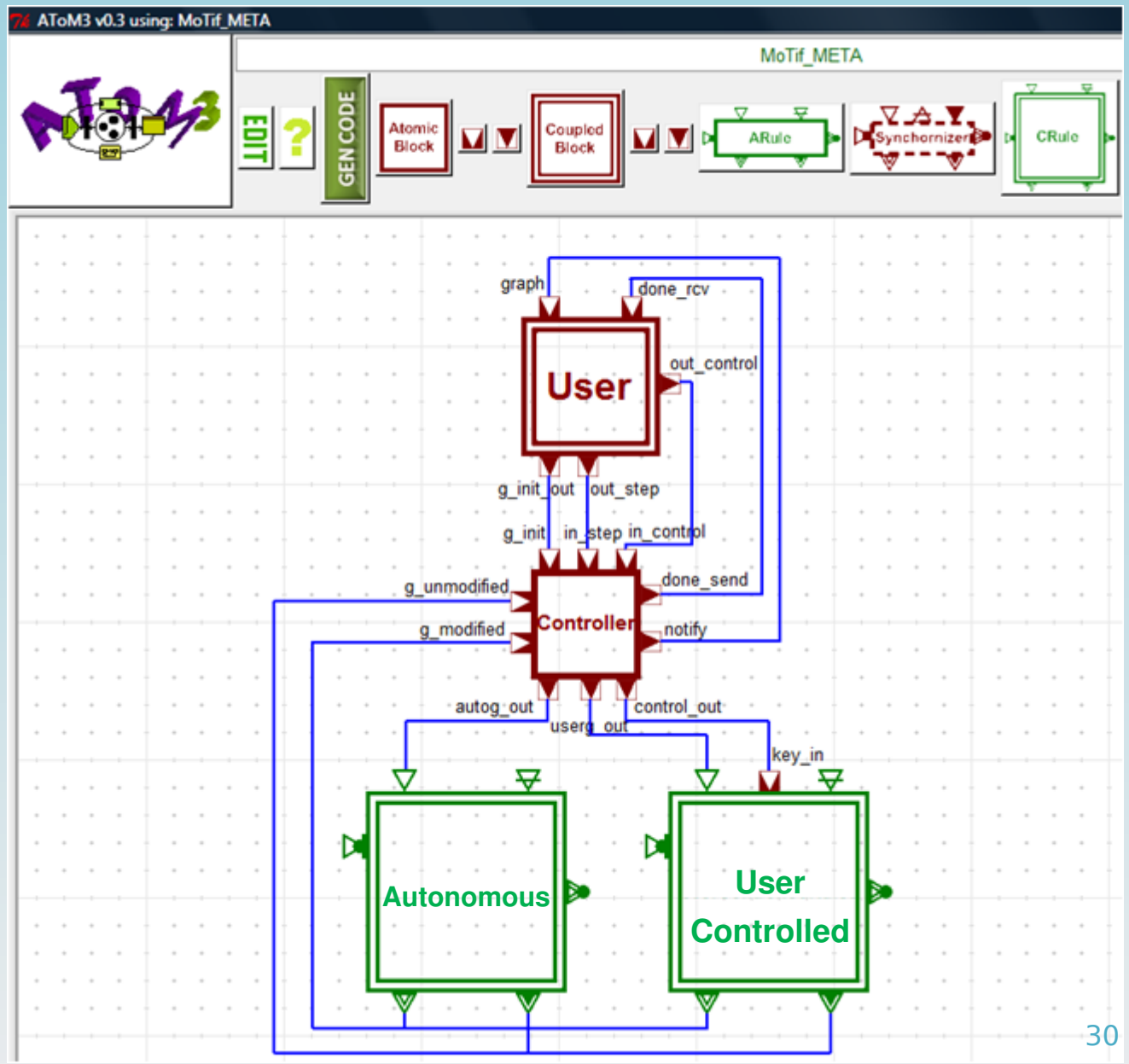
EXTENSION OF ATOM³'S GRAPH TRANSFORMATION ENGINE



McGill

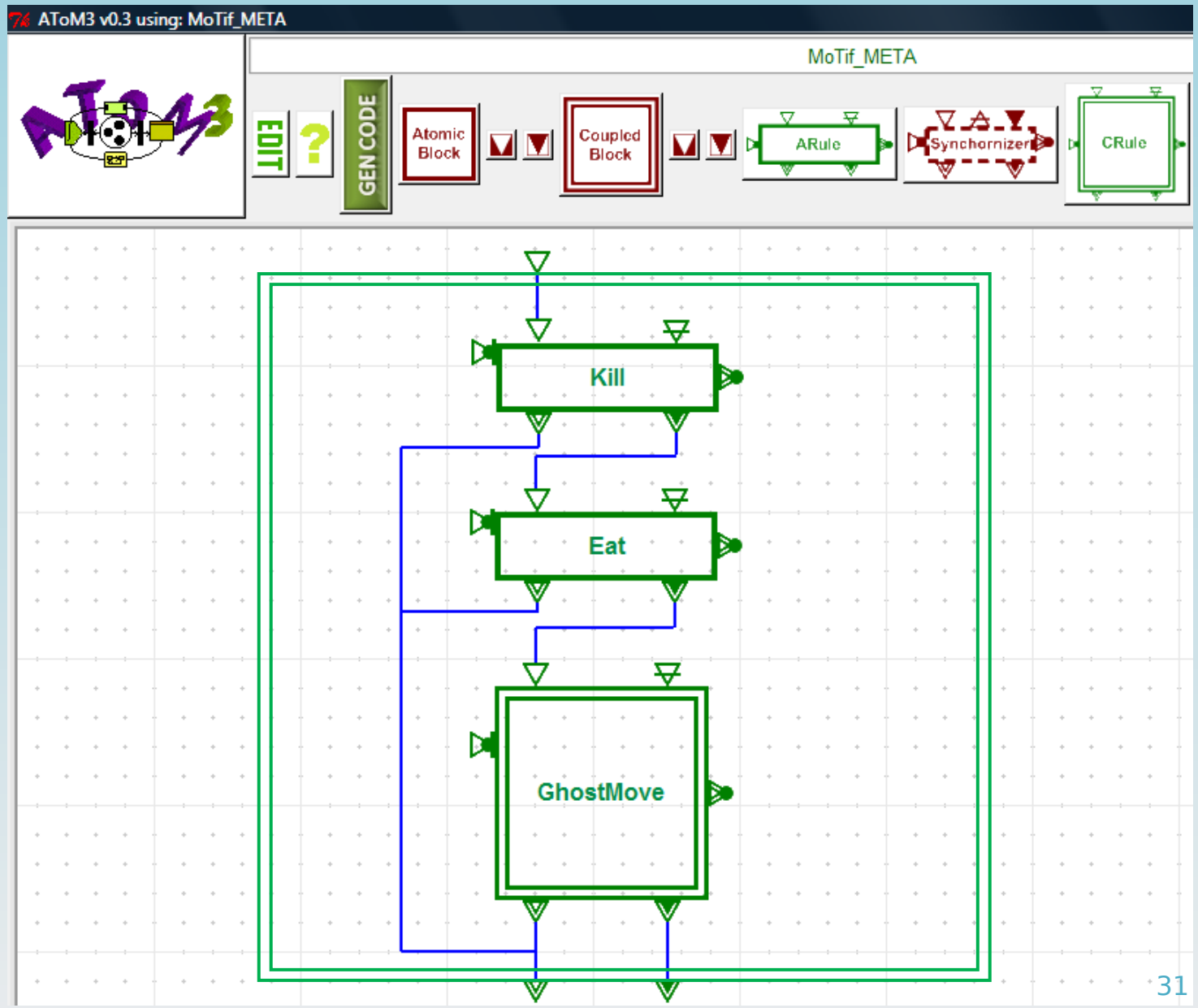
MODELLING OF THE TRANSFORMATION: SYSTEM

- User – Controller – Autonomous loop
- Feed-back to User
- On User interrupt: User Controlled
- Feed-back to User



MODELLING OF THE TRANSFORMATION: AUTONOMOUS *CRule*

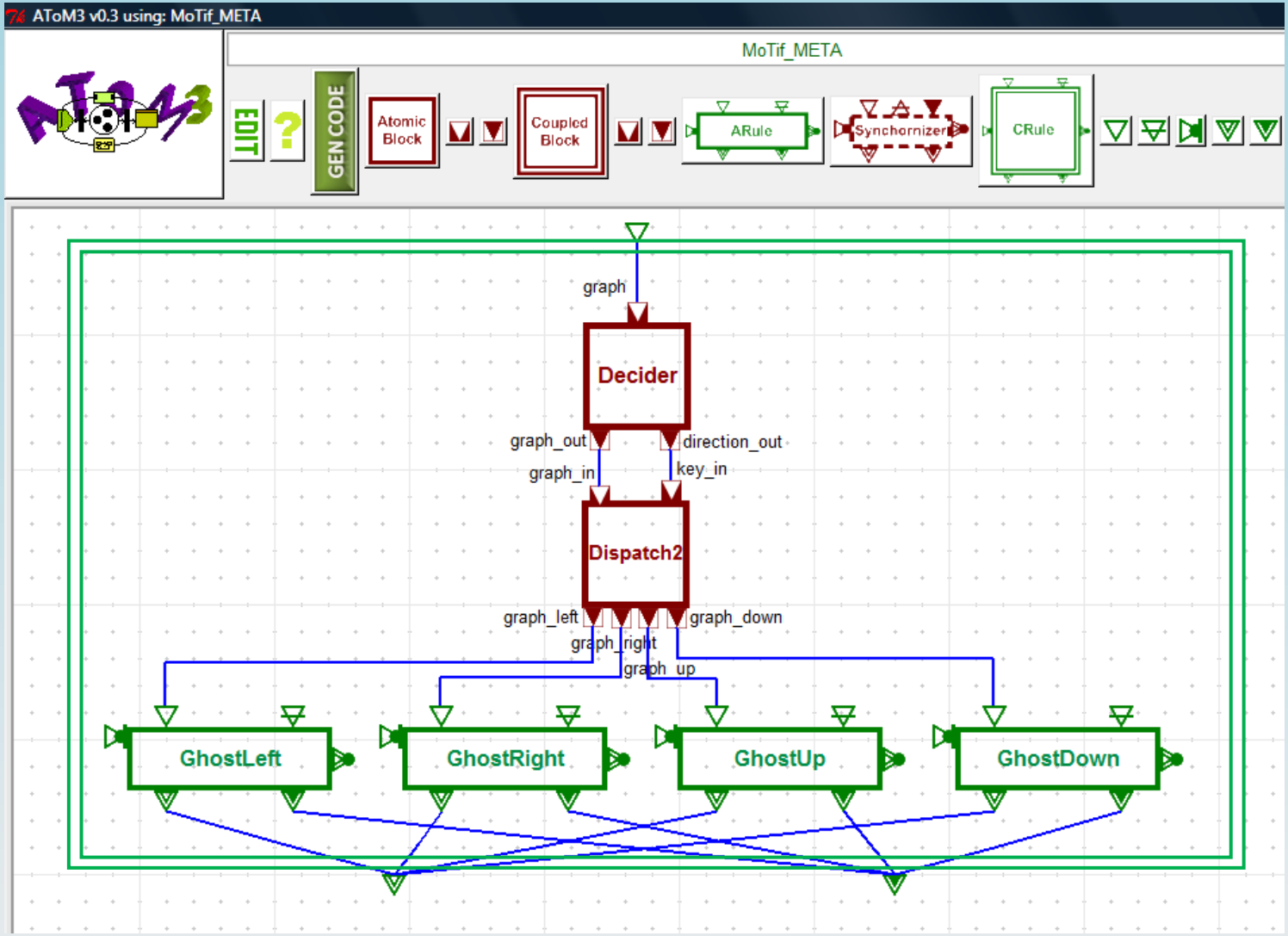
Priorities



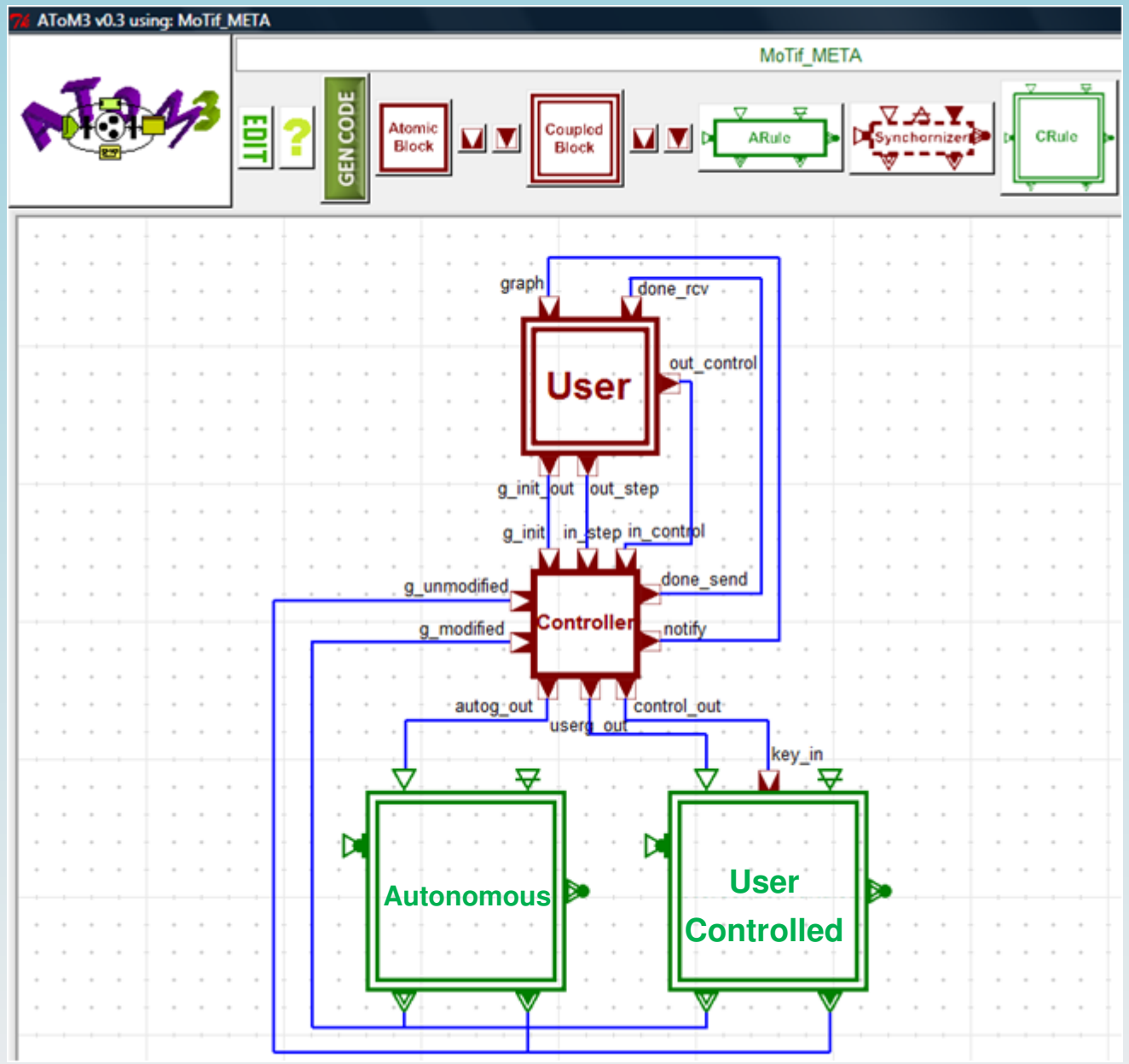
MODELLING OF THE TRANSFORMATION: GHOSTMOVE *CRule*

Decider finds the next move for the ghost

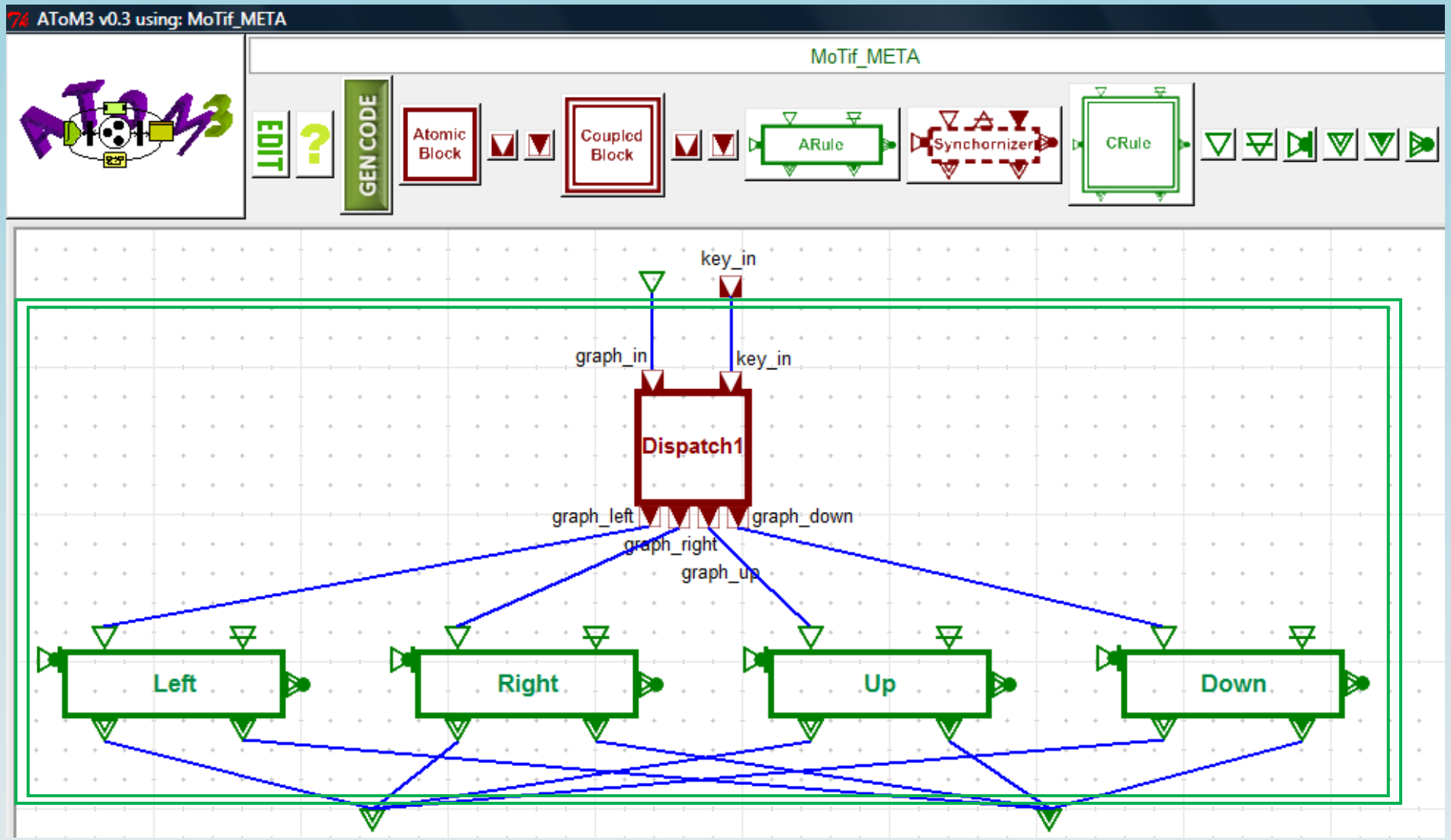
Decider consumes time



MODELLING OF TRANSFORMATION: SYSTEM

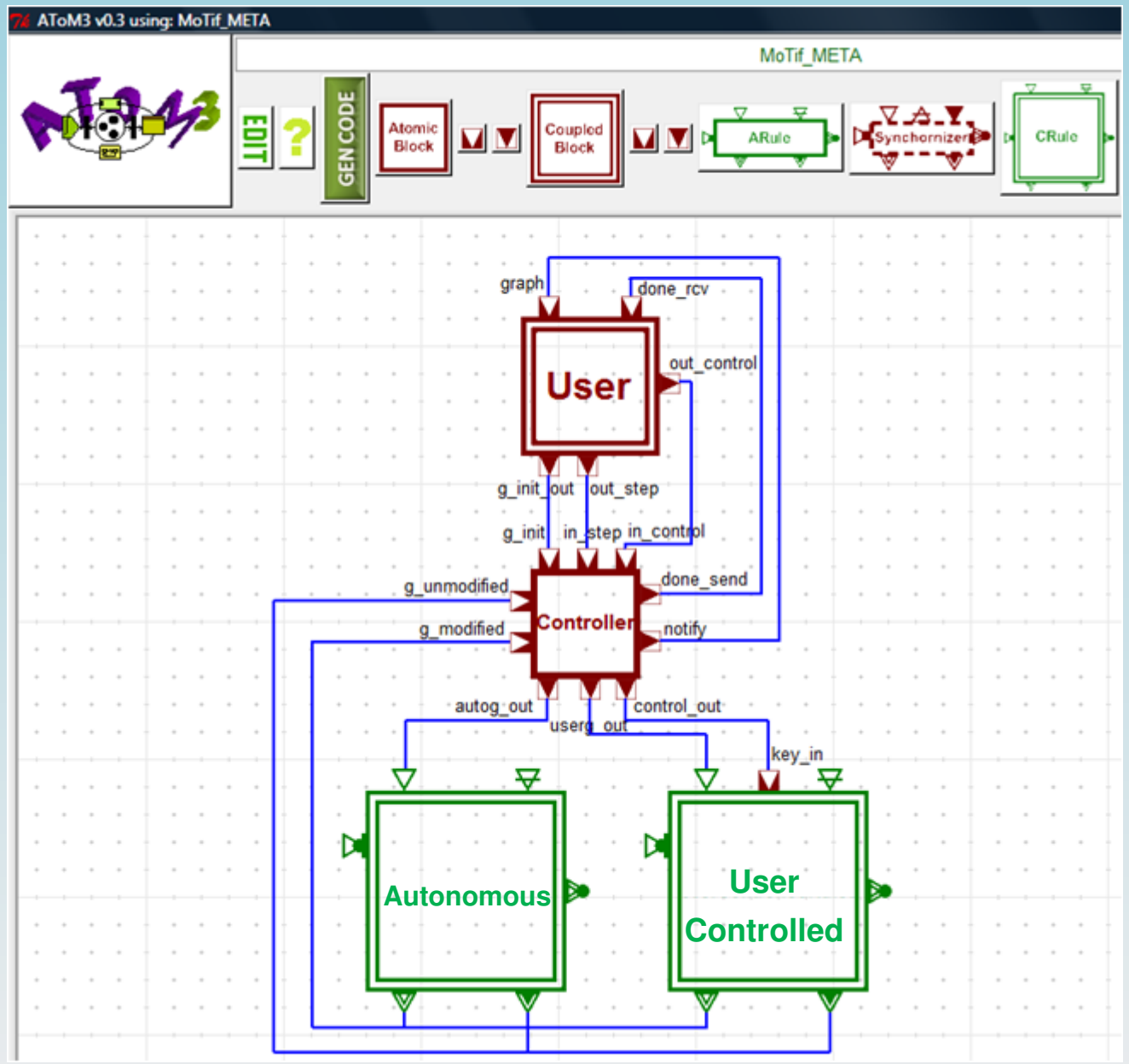


MODELLING OF THE TRANSFORMATION: USERCONTROLLED *CRule*

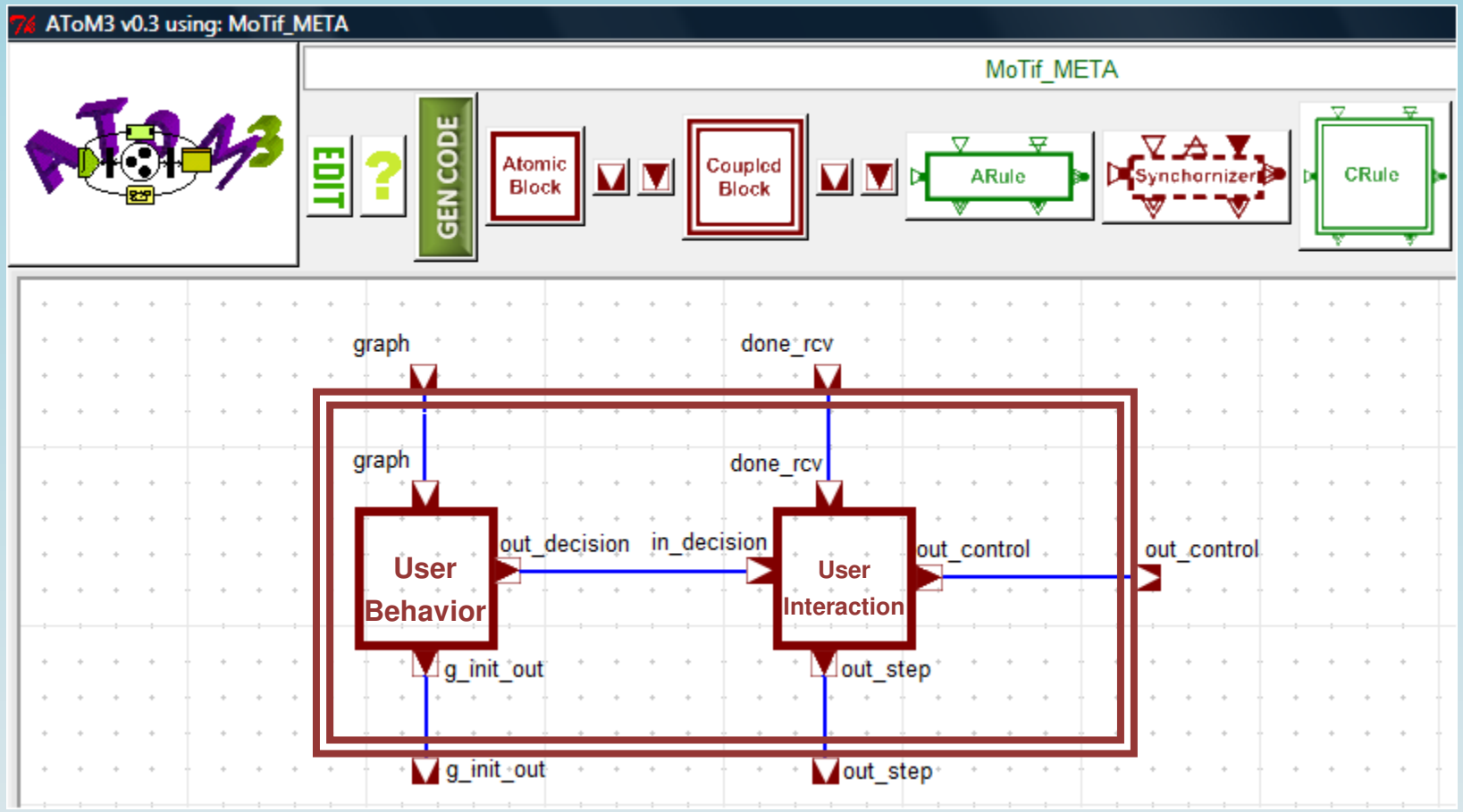


Conditional rule execution

MODELLING OF THE TRANSFORMATION: SYSTEM



MODELLING OF ENVIRONMENT: USER CoupledBlock

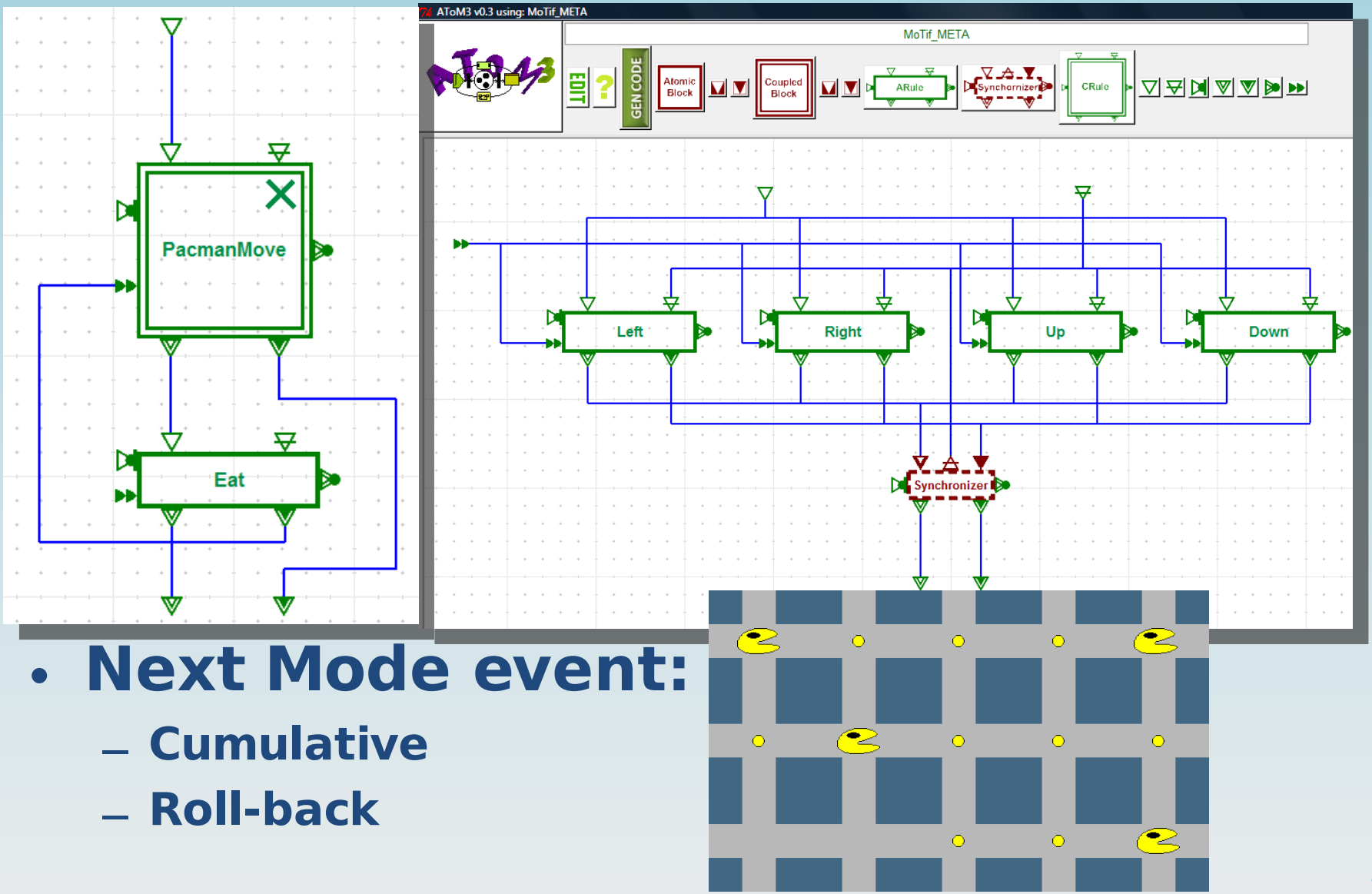


- **Customization**
- **Modularity**

BACKTRACKING

- **What?**
  Algorithmically, used for exploring a search space
- **Why?**
  In graph transformation, needed when a non-deterministic choice is made
  - **Matching level**
  - **Rule level**
- **How?**
  - **Make copies of the graph**
  - **Use checkpoints and transactions**
  - **Have undo/inverse transformations**

BACKTRACKING



- **Next Mode event:**
  - **Cumulative**
  - **Roll-back**

McGill

SUMMARY

Control flow structure properties satisfied

- ✓ Sequence
- ✓ Branching
- ✓ Looping
- ✓ Hierarchy + Modularity
- ✓ (pseudo-)Parallelism
- ✓ Backtracking
- ✓ Time

ONE STEP FURTHER

User - Events

- **Event-driven Graph Rewriting**
- **Modelling of the user**

➢ **Web-based pacman game**

- **AJAX**
- **SVG**
- **Real-time**

McGill

ONE STEP FURTHER

Time

- **Metric, Statistics**
- **Timed graph transformation**
- **Real-Time DEVS**

McGill

FUTURE WORK

Some Extensions

- Optimization (rule compilation)
- Replace transformation blocks by Statecharts, code, ...
- Integrate MoTif in AToM³: 2-way communication
- Scaling for larger models:
  - Database
  - Distributed

FUTURE WORK

Parallelism

- **DEVS is a sequential formalism**
  - **Parallel-DEVS**
  - **Kiltera (CSP-like languages)**
- **Distributed rule application**

Variable Structure Formalisms