McGill

Programmed Graph Rewriting: MoTif

Eugene Syriani

Ph.D. Student in the Modelling, Simulation and Design Lab

School of Computer Science

McGill University

PROGRAMMED GRAPH REWRITING WITH DEVS

Eugene Syriani and Hans Vangheluwe.

Programmed Graph Rewriting with DEVS.

Proceedings of the 4th International Conference Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007),

Lecture Notes in Computer Science (LNCS). Springer-Verlag,

October 2007.

Kassel, Germany.

OVERVIEW

- In the context
- Existing Programmed Graph Rewriting Systems
- AToM^3's graph rewriting engine by example
- Modelled and Modular Timed Graph Transformation (MoTif): Mimic AToM^3 and beyond
- Conclusion and Future Work

IN THE CONTEXT

- MDA = Meta-Model + Model Transformation (+ …)
- Model Transformation ► Graph Transformation
- Types of Graph Transformations [1]
  - Unordered Graph Rewrting: non-deterministic, run till no more
  - Ordered Graph Rewriting: explicit (partial) ordering
  - Event-driven Graph Rewriting: external ordering
- 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.

Programmed Graph Rewriting: MoTif

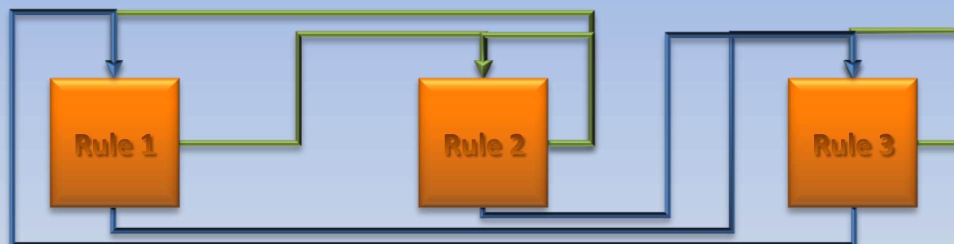
IN THE CONTEXT

Programmed Graph Rewriting CONCRETE SYNTAX

- **Tabular:** success & failure lists for each production

	Rule 1	Rule 2	Rule 3
Rule 1	0	S	F
Rule 2	S	S	F
Rule 3	F	0	S

- **Diagrammatic:** graphical representation of the flow, using arrows



- **Textual:** imperative programming language constructs

```
while(match)
   if Rule1
       match = Rule2;
   elif Rule2
       match = Rule1 || Rule2;
   elif Rule3
       match = Rule3;
   else
       match = Rule1 || Rule2 || Rule3;
```

IN THE CONTEXT

Programmed Graph Rewriting WISH LIST

- Cleanly tear apart
  1. Transformation entities
  2. Control flow, structure, hierarchy
- Graph transformation control flow primitives
  - Sequencing, Branching, Looping
  - Parallelism
  - Hierarchy
  - Time

OVERVIEW

- In the context
- Existing Programmed Graph Rewriting Systems
- AToM^3's graph rewriting engine by example
- Modelled and Modular Timed Graph Transformation (MoTif): Mimic AToM^3 and beyond
- Conclusion and Future Work

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

Programmed Graph Rewriting System (ProGReS) [2,3]

Manfred Nagl

Andy Schürr

Albert Zündorf

University of Aächen, Germany

1988-1999

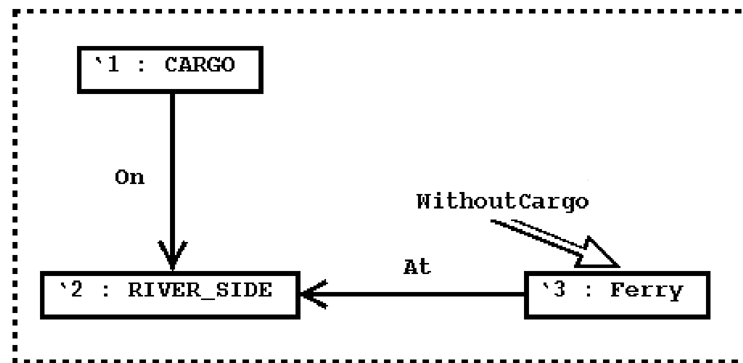
[2] Blostein D., Schürr A., Computing with graphs and graph rewriting, *Proceedings in Informatics*, pp. 1-21, 1999.
[3] Schürr A., Winter A.J., Zündorf A., Graph grammar engineering with progres, *Proceedings of ESEC*, pp. 219-234, 1995.

Programmed Graph Rewriting: MoTif

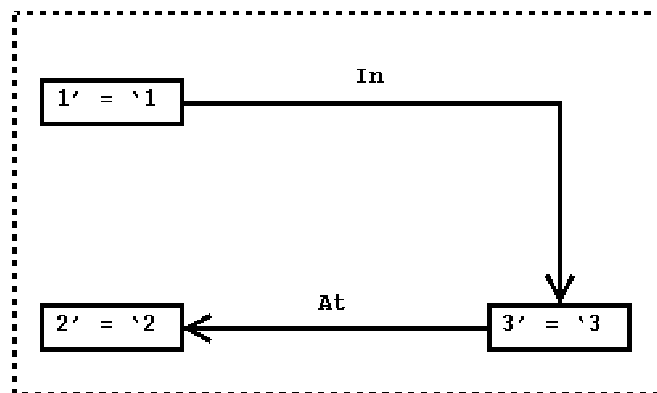
PROGRES

FerryMan: PROGRES-View-1 (V 9.4)

```
production LoadFerry (* Puts one cargo into the ferry    *)
                     (* if the ferry is on the same side *)
                     (* and empty.                       *)
=
```




```
::=
```




```
condition `1.Win > Payment;
transfer 1'.Expenditures := `1.Expenditures + Payment;
         3'.Income := `3.Income + Payment;
end;
```

```
transaction ProcessInput =
    ReadScannerOutput     (* Figure 14: a → b                   *)
  & BuildPhase            (* Figure 14: b → c                   *)
  & ConstrainPhase        (* Figure 14: c → d                   *)
  & Parse                 (* Figure 14: d → f                   *)
                          (* The Parse transaction is shown below. *)
end;

transaction Parse =
    ReduceSymbols         (* letters, digits,... → terms *)
  & loop
      ReduceFactors       (* terms → factors *)
    & ReduceExpressions   (* factors → exprs. The ReduceExpressions *)
                          (* transaction is shown below.            *)
    & ReduceFractionLines (* Figure 14: e → f *)
    end
end;

transaction ReduceExpressions =
    ReduceSingleFactor
  & loop
      ReduceMultiply      (* Implied multiplication: A B *)
    end
  & ...
  & loop
      ReducePlusMinus     (* The ReducePlusMinus rule is shown in *)
                          (* Figure 15.  It reduces expressions   *)
                          (* involving addition and subtraction.  *)
    end
end;
```

PROGRES

- First fully implemented environment for programming through graph transformation

---

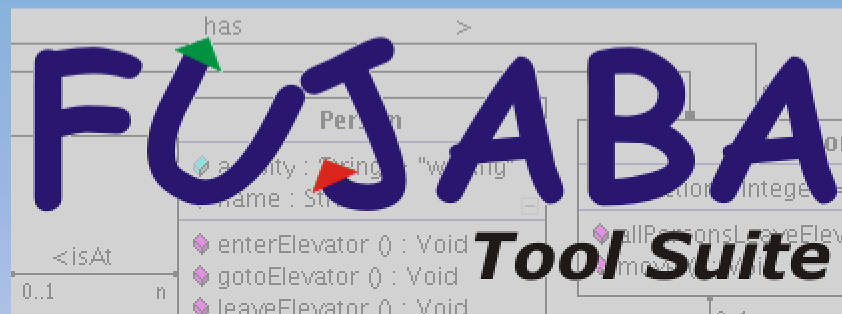
Transformation Control Structure: *Programming constructs*

✓ Sequencing (`sequence`,`&&`,`||`)
✓ Branching (`choose`)
✓ Looping (`loop`)
✓ Hierarchy (encapsulation of rules)
– Parallelism
– Time
+ Transactions
+ Backtracking

McGill

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

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



University of Kassel, Germany

Since 1998

[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.

Programmed Graph Rewriting: MoTif

FUJABA

Fujaba [The famous elevator example] - elevator.fpr.gz

File Edit Diagrams Class Diagram Import/Export Tools Options Help

- Class diagrams
  - ElevatorDemo
- Activity diagrams
  - Elevator
    - internal
    - allPersonsLeaveElevator
    - move
  - House
    - internal
    - createObjects
  - Level
    - internal
    - allPersonsEnterElevator
  - Person
    - internal
    - enterElevator
    - pressButton
    - gotoElevator
    - leaveElevator
  - Place
    - internal
    - removeAllPersons

UML

StateChartFlattener::transitionFromInner (): Boolean

sc :StateChart
statechart
this
elements
«destroy»
or :OrState
source
«destroy»
orToA:Transition
target
«destroy»
a :State
[failure]
false
[success]
[each time]
or
superState
inner:State
source
another :Transition
label == orToA.getLabel()
inner
a
source
«create»
target
«create»
«create»
innerToA:Transition
label := orToA.getLabel()
action := orToA.getAction()
{ maybe inner == a }
[end]
true

Welcome to Fujaba. Just draw it!

9 MByte of 17 MByte allocated

FUJABA

- Implementation (JAVA) oriented: content of class methods is described by Story Charts diagrams

---

Transformation Control Structure: *Story Charts*

✓ Sequencing (`success-fail` transition)
✓ Branching (`if-else` guard on transition)
✓ Looping (`for-all` pattern)
– Hierarchy
– Parallelism
– Time

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

MOFLON [6]

University of Darmstädt, Germany

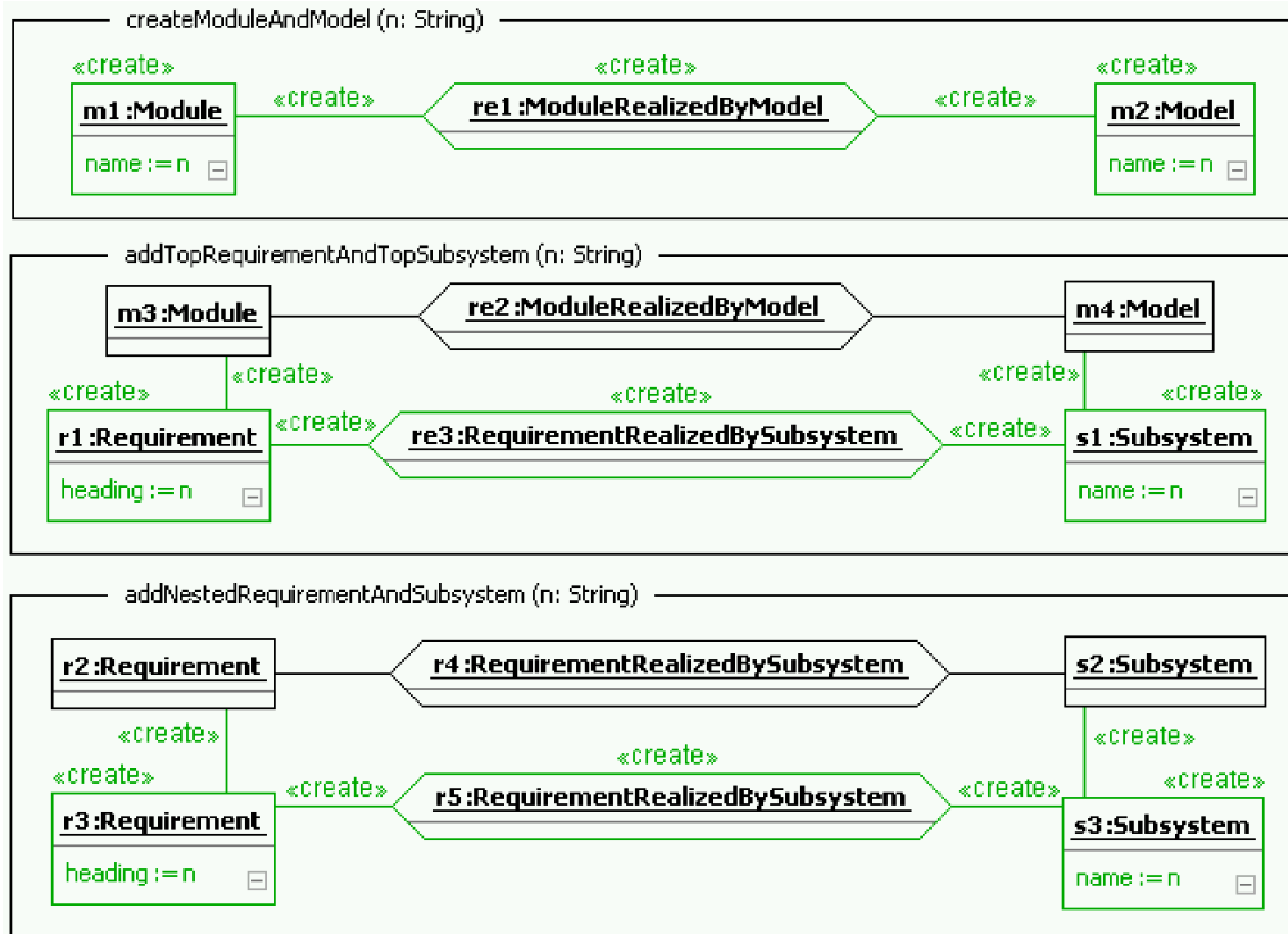
Since 2004

[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.

Programmed Graph Rewriting: MoTif

McGill

MOFLON



MOFLON

- Uses the FUJABA graph transformation engine
- Environment for declarative specification of transformations in terms of a Triple Graph Grammar (TGG)
- TGGs are compiled to Story Diagrams

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

Visual Modelling and Transformation System (VMTS) [10,11]

Tihamér Levendovszky **László Lengyel**

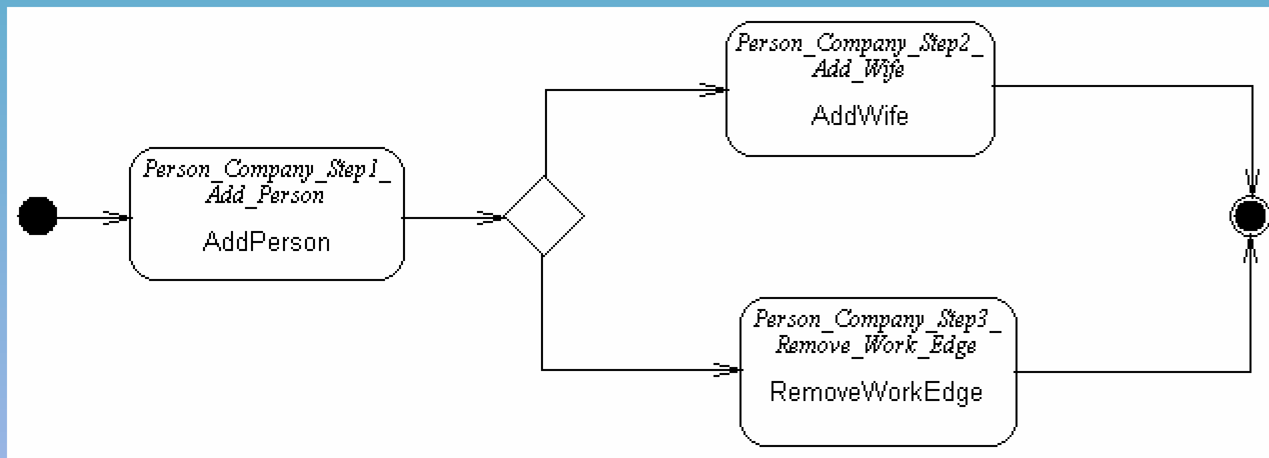
Budapest University of Technology and Economics, Hungary

Since 2004

[10] Lengyel L. et al., Control flow support in meta-model based model trasnformation frameworks, *Proceedings of EUROCON*, pp. 595-598, 2005.
[11] Lengyel L. et al., Model transformation with a visual control flow language, IJCS 1, pp. 45-53, 2006.

Programmed Graph Rewriting: MoTif

VMTS



Imperative OCL Editor

```
lhs_Company.MatchedNode->forone(lhsDD:lhs_Company.MetaType)
{
var p:Person:=new rhs_PersonA.MetaType;
var c_p:work:=new rhs_work.MetaType;
p.name := 'PersonA';
p.age := 30;
c_p.Left := lhsDD;
c_p.Right:=p;
rhs_PersonA.CreatedNode+=p;
rhs_work.CreatedEdge+=c_p
}
```

OK Cancel

Programmed Graph Rewriting: MoTif

VMTS

- Stereotyped-Activity diagram
- LHS and RHS can be linked by XSLT scripts and actions specified in imperative OCL
- QVT realizations

Transformation Control Structure: *VMTS Control Flow Language*

✓ Sequencing (link steps)
✓ Branching (OCL condition)
✓ Looping (link steps and OCL condition)
✓ Hierarchy (composition of primitive steps)
– Parallelism
– Time
+ Pivot Information

McGill

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

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

Gabor Karsai

Aditya Agrawal

Vanderbilt University, Nashville (TN), USA

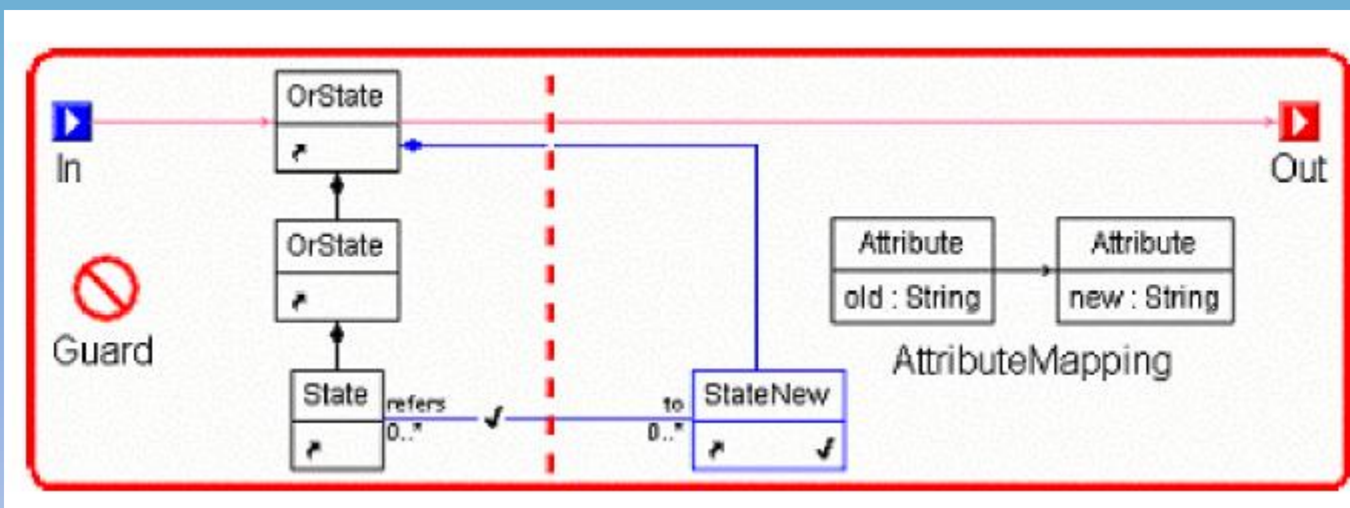
Since 2001

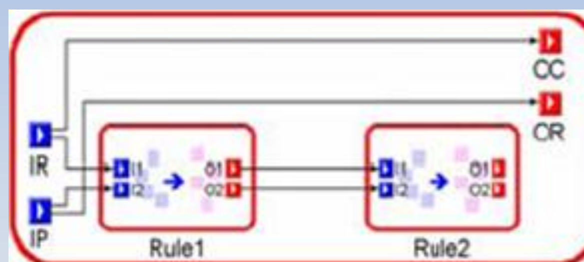
[7] Vizhanyo A. et al., Towards generation of high-performance transformations, *Proceedings of GPCE*, LNCS 3286 pp. 298-396, 2004.
[8] Agrawal A., Metamodel based model transformation language, *Proceedings of OOPSLA*, ACM Press pp. 386-397, 2003.
[9] Agrawal A., The design of a language for model transformations,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

McGill

Programmed Graph Rewriting: MoTif

GREAT





Programmed Graph Rewriting: MoTif

GREAT

- Rules represented in blocks
- Packets (graphs) are sent via ports

Transformation Control Structure: *Custom Control Flow Language*

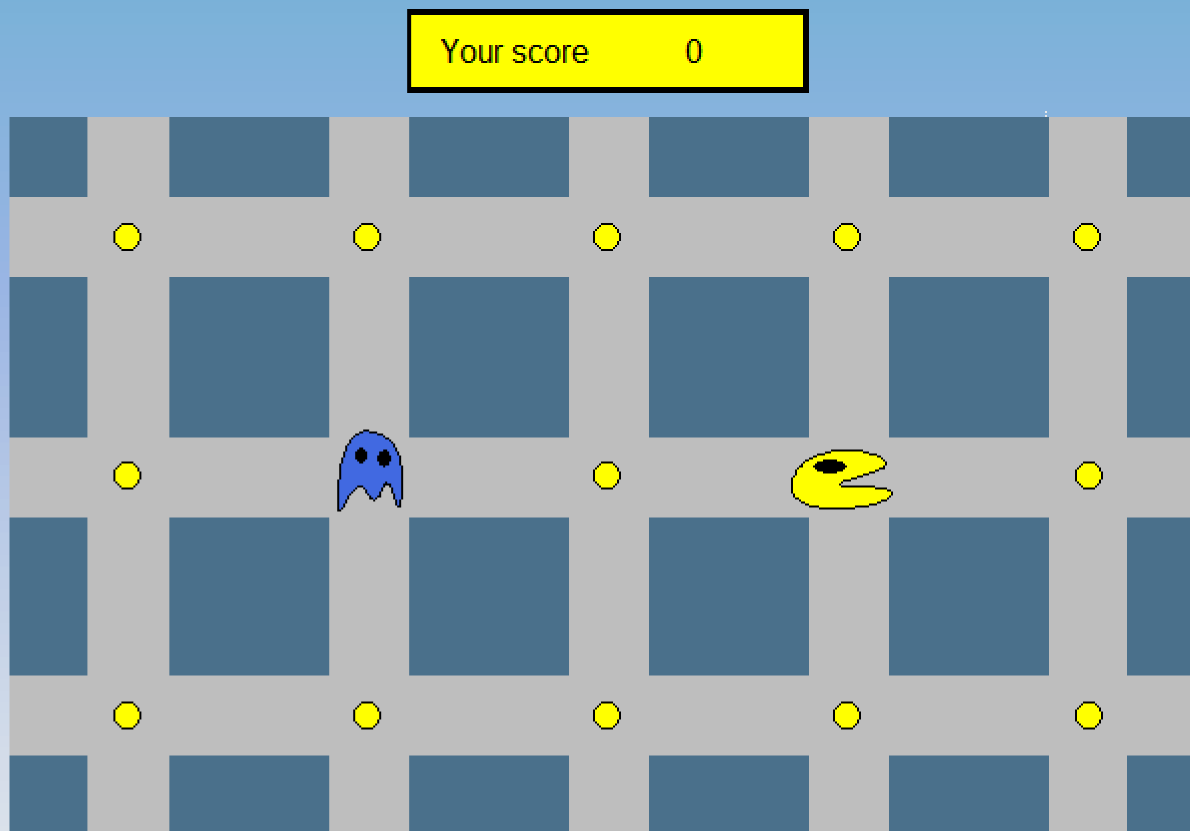
✓ Sequencing (connection of `Inports` and `Outports`)
✓ Branching (`Test-Case` rules)
✓ Looping (recursion)
✓ Hierarchy (`forblock`)
✓ Parallelism (one-to-many connection)
– Time
+ Pivot Information

OVERVIEW

- In the context
- Existing Programmed Graph Rewriting Systems
- $AToM^3$'s graph rewriting engine by example
- Modelled and Modular Timed Graph Transformation (MoTif): Mimic $AToM^3$ and beyond
- Conclusion and Future Work

BUILDING EXAMPLE WITH ATOM^3 [12]

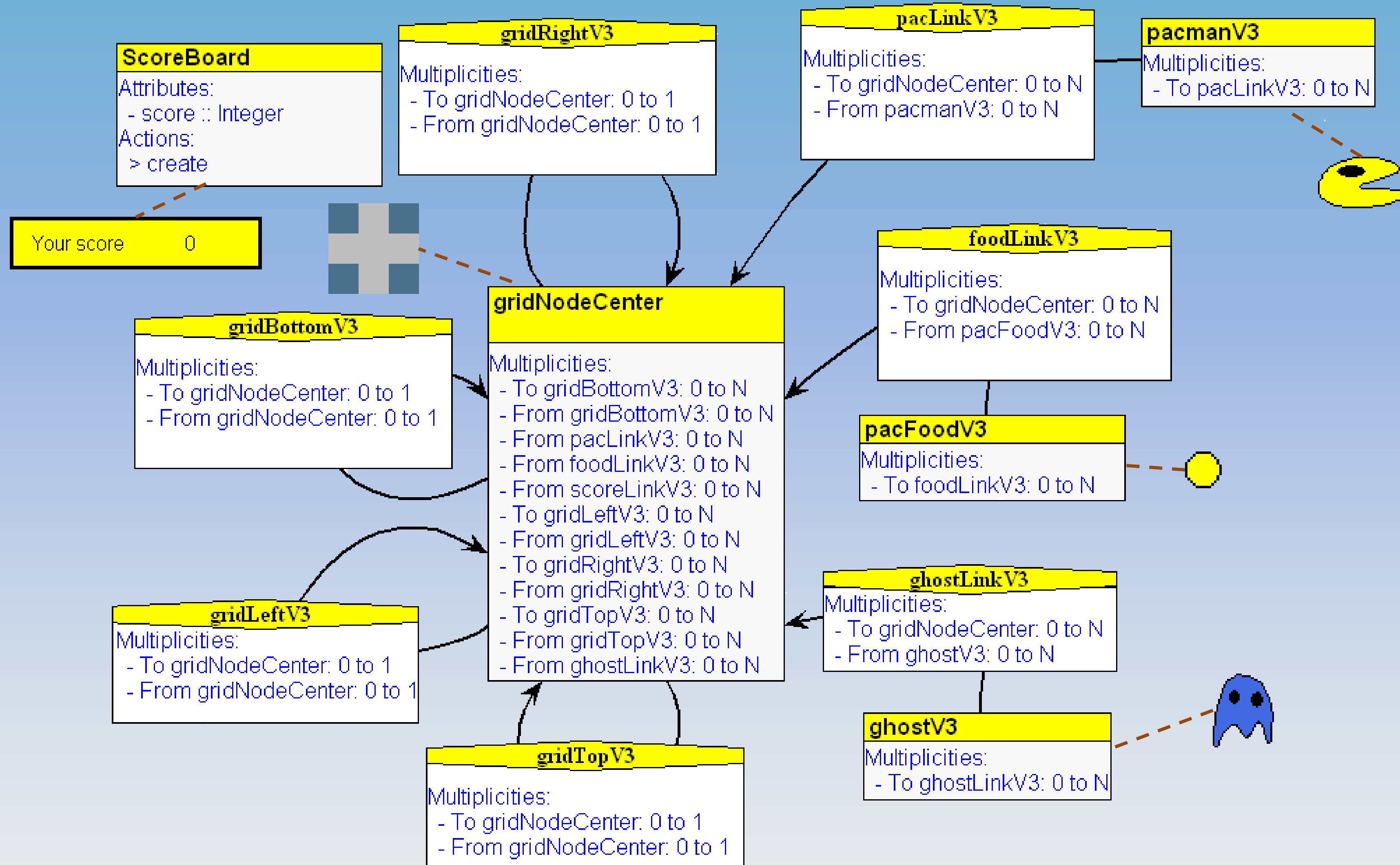
Simplified PacMan formalism [13]



[12] de Lara J., Vangheluwe H., AToM^3: A tool for multi-formalism and meta-modelling, *LNCS* (2002), 174-188
[13] Heckel R., Graph Transformation in a nutshell, *ENTCS* (2006), 187-198

BUILDING EXAMPLE WITH ATOM^3

Build the Meta-Model of the PacMan formalism



BUILDING EXAMPLE WITH ATOM³

Build the Graph Grammar

AToM3 v0.3 using: pacManV3 + TransformationToolbar

pacManV3: New Edit | New Help | New gridNodeCenter | New pacmanV3 | New pacFoodV3 | New ScoreBoard | New ghostV3

TransformationToolbar: EDIT | LOAD | SAVE | GEN | EXEC | DOC | ? | CGEN

Editing GraphGrammarEdit

WARNING: Name must use Python variable syntax

Name: pacGrammarScore

InitialAction: Enabled?

Rules: New / Edit / Delete — pacDie 1, pacEat 2, pacMoveRight 4, pacMoveUp 4

FinalAction: Enabled?

Load GG
Save GG
Generate latex document from GG
Generate GG code
Execute GG code

OK | Cancel

Editing GGruleEdit

Name (Python variable syntax required): pacDie
Order: 1
TimeDelay: 2
Subtypes Matching (if rule matches A then it also matches B if B has all attributes in A)
Condition: Edit — Enabled?
Action: Edit — Enabled?

LHS — pacManV3: New Edit | New Help | New gridNodeCenter | New pacmanV3 | New pacFo

RHS — pacManV3: New Edit | New Help | New gridNodeCenter | New pacmanV3 | New pac

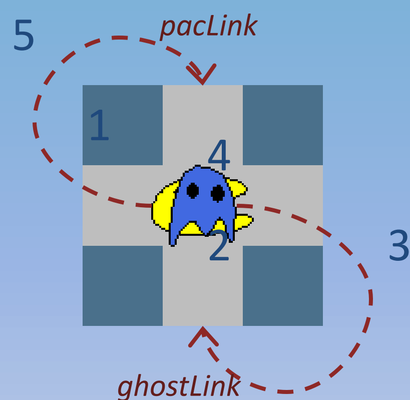
Editing 'Nonamed' (modified) | Editing transf. 'Nonamed' (not modified | Editing 'Nonamed' (modified) | Editing transf. 'Nonamed' (not modif

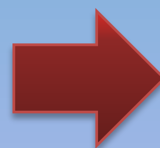
OK | Cancel

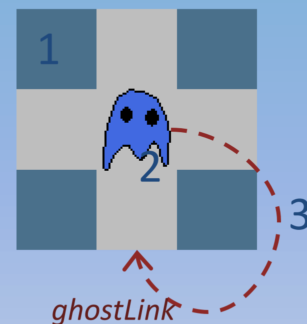
Editing 'Nonamed' (not modified) | pacGrammarScore_GG_mdl.py'

BUILDING EXAMPLE WITH ATOM^3

Build the Graph Grammar





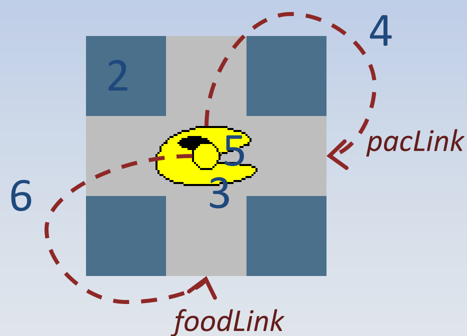


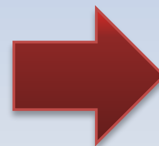
P 1

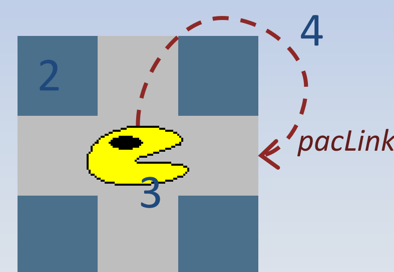
Your score <ANY> 1

Your score<SPECIFIED> 1

P 2



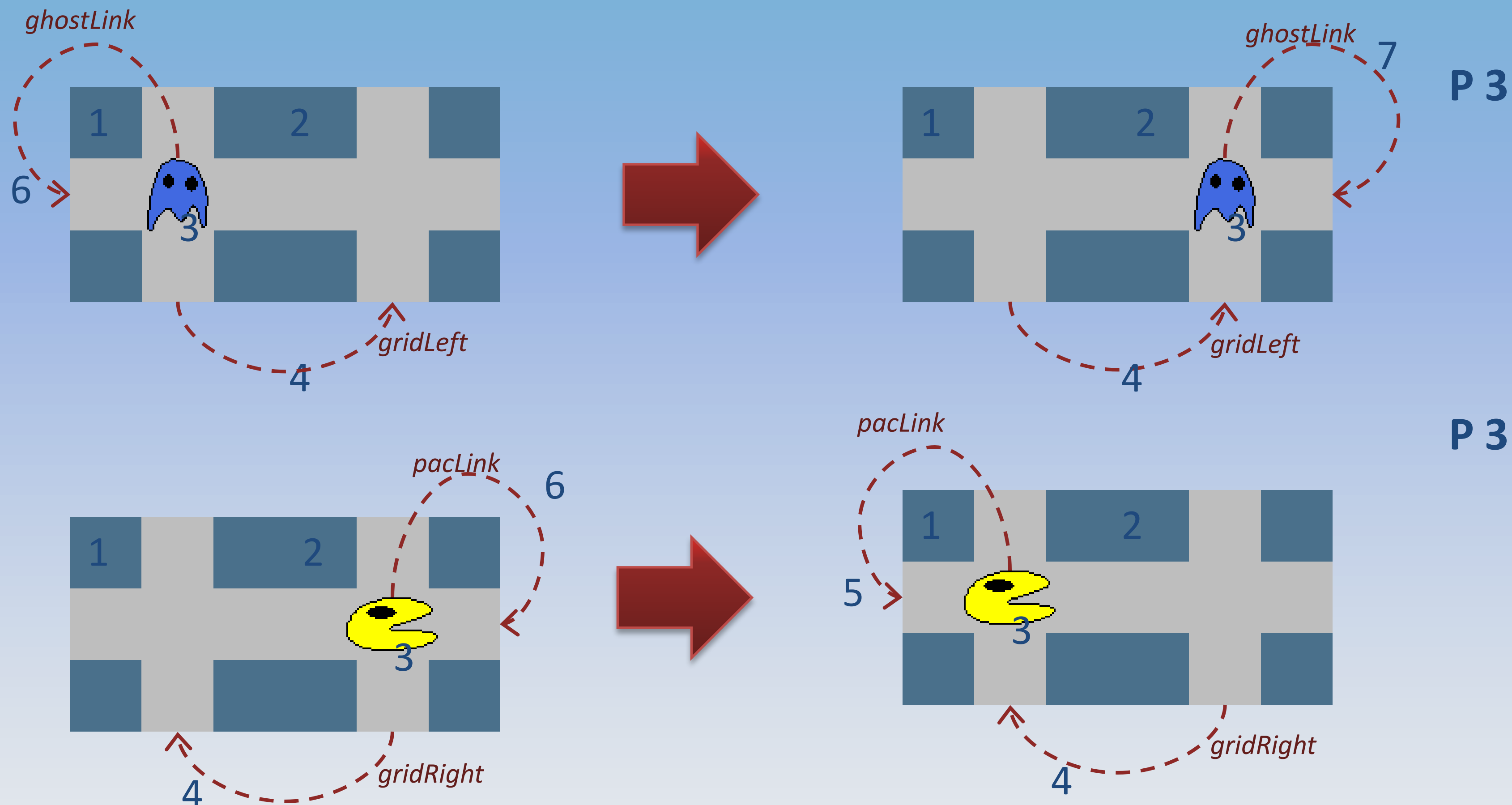




1: `return self.LHS.nodeWithLabel(1).score + 1`

BUILDING EXAMPLE WITH ATOM^3

Build the Graph Grammar



BUILDING EXAMPLE WITH ATOM^3

- Capture a trace of execution
  - 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
```

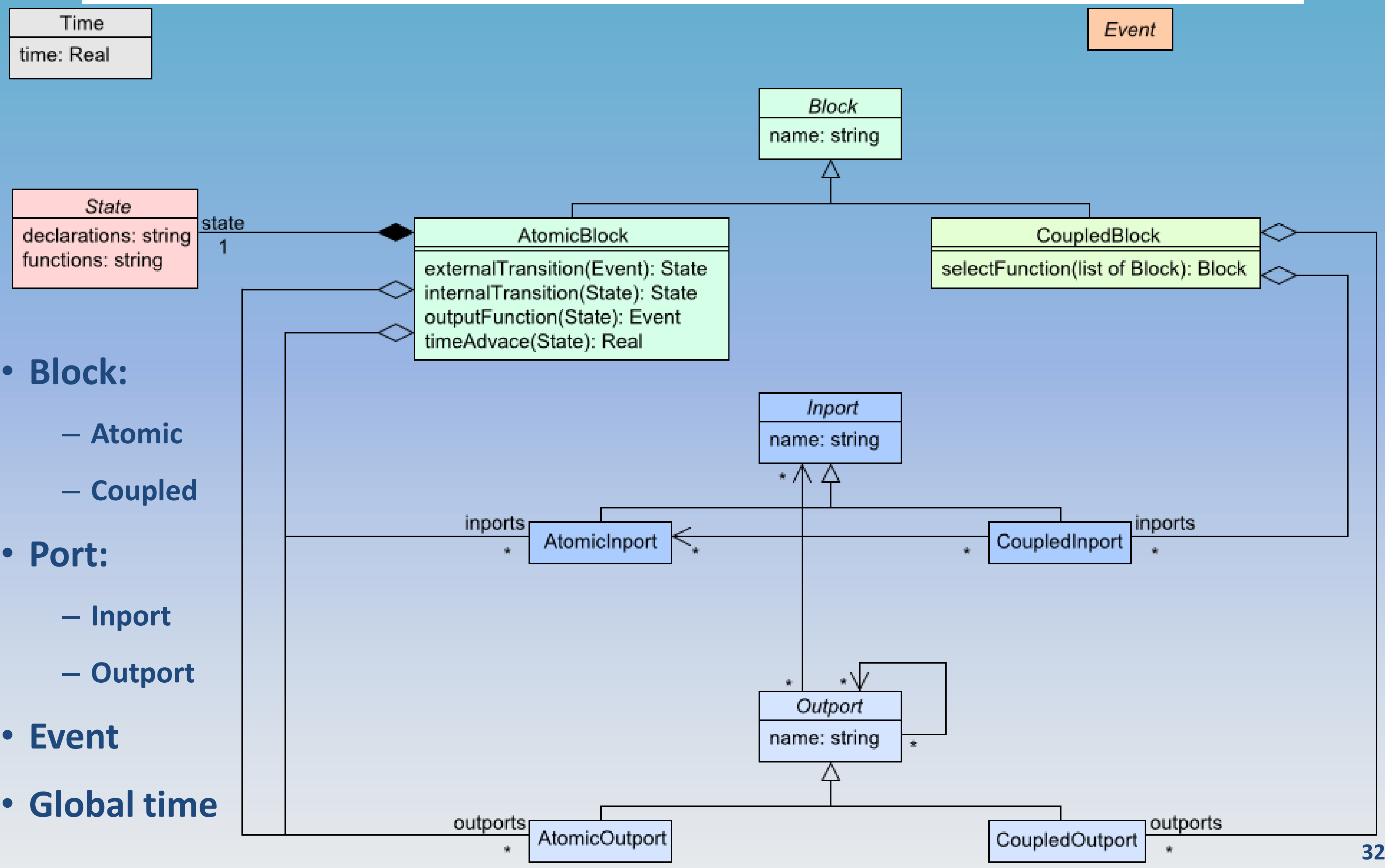
OVERVIEW

- In the context
- Existing Programmed Graph Rewriting Systems
- AToM^3's graph rewriting engine by example
- Modelled and Modular Timed Graph Transformation (MoTif): Mimic AToM^3 and beyond
- Conclusion and Future Work

OVERVIEW OF THE DEVS FORMALISM

- Bernard Zeigler, late ‘70s
- Basis for compositional modelling and simulation of discrete event systems
- Design, performance analysis and implementation

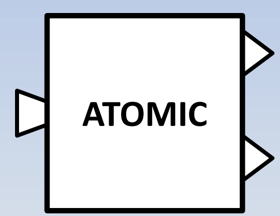
OVERVIEW OF THE DEVS FORMALISM



OVERVIEW OF THE DEVS FORMALISM

Atomic DEVS:

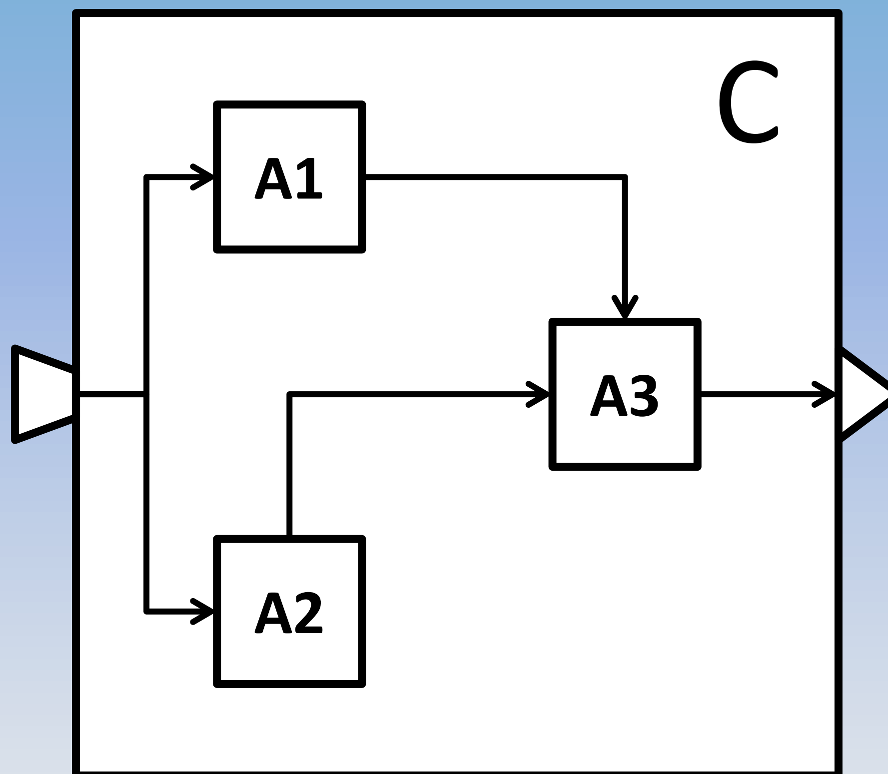
- Time Advance
- Output Function
- Internal Transition
- External Transition



X
x1
ϕ
tx
T
S
s
(s,0)
(s,e)
δ_ext((s,e),x)
ta(s)
δ_int(s)
tx
T
Y
y1
ϕ
λ(s2)
T

OVERVIEW OF THE DEVS FORMALISM

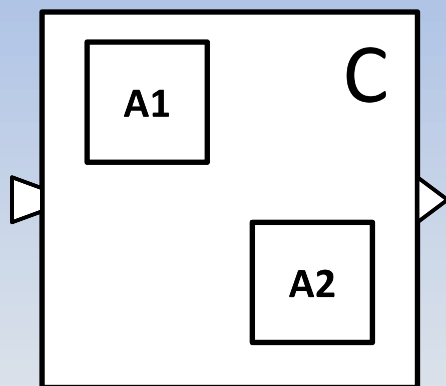
Coupled DEVS



OVERVIEW OF THE DEVS FORMALISM

Coupled DEVS:

– Select Function



S_1

s_{11} (s_{11}, t_1) $ta(s_{11})$

δ_int(s_{11})

s_{12}

t_1 T

S_2

s_{22}

δ_int(s_{21})

s_{21} (s_{21}, t_2) $ta(s_{21})$

t_2 T

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]
```

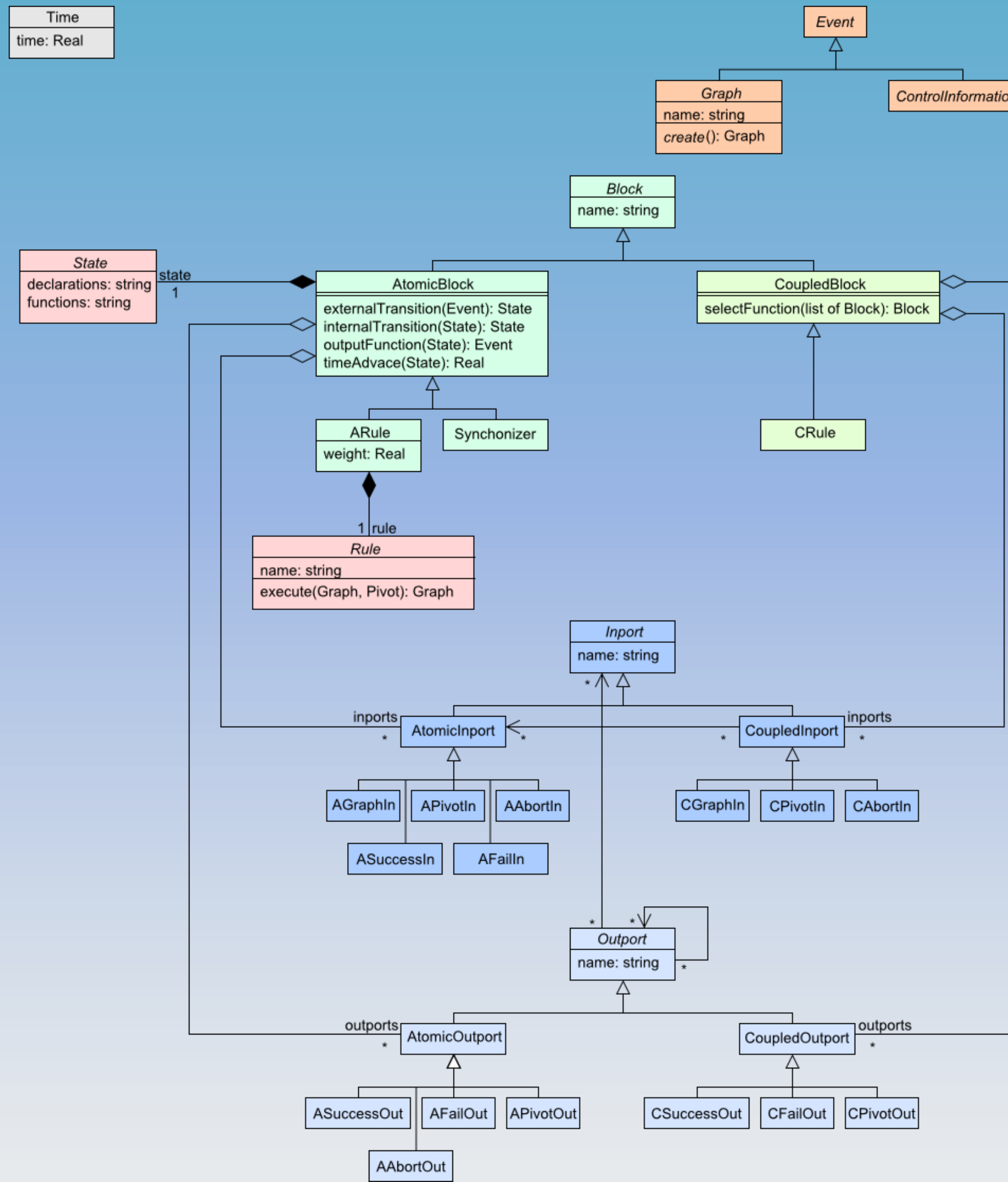
MOTIF

- DEVS blocks
  - Atomic block: encapsulate the graph rewriting rule
  - Coupled block: encapsulate a graph grammar
- Events
  - Inport: recieve the host graph
  - Outport(s): send the transformed graph

Programmed Graph Rewriting: MoTif

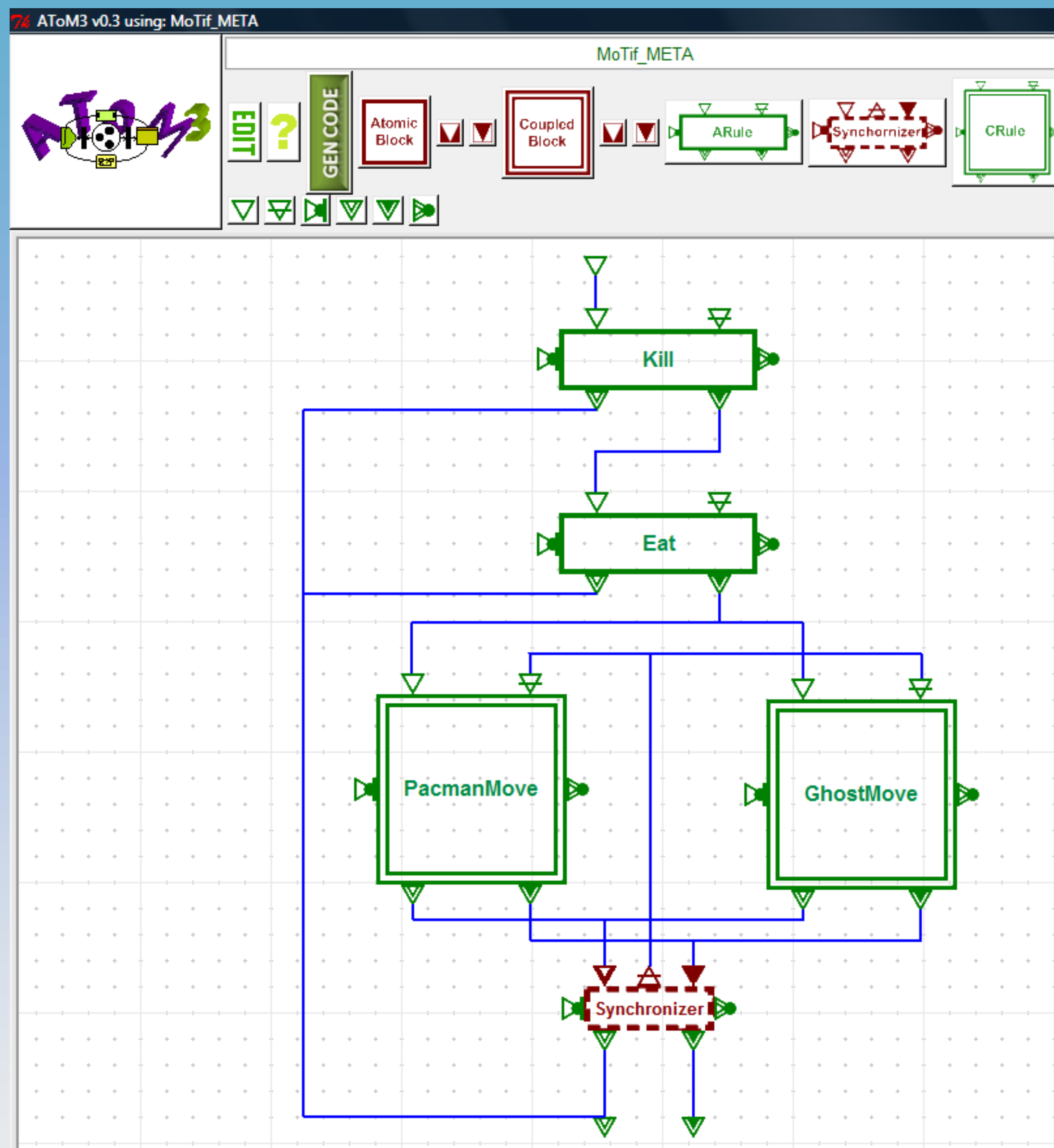
McGill

MOTIF META-MODEL



McGill

MOTIF MODELLING ENVIRONMENT

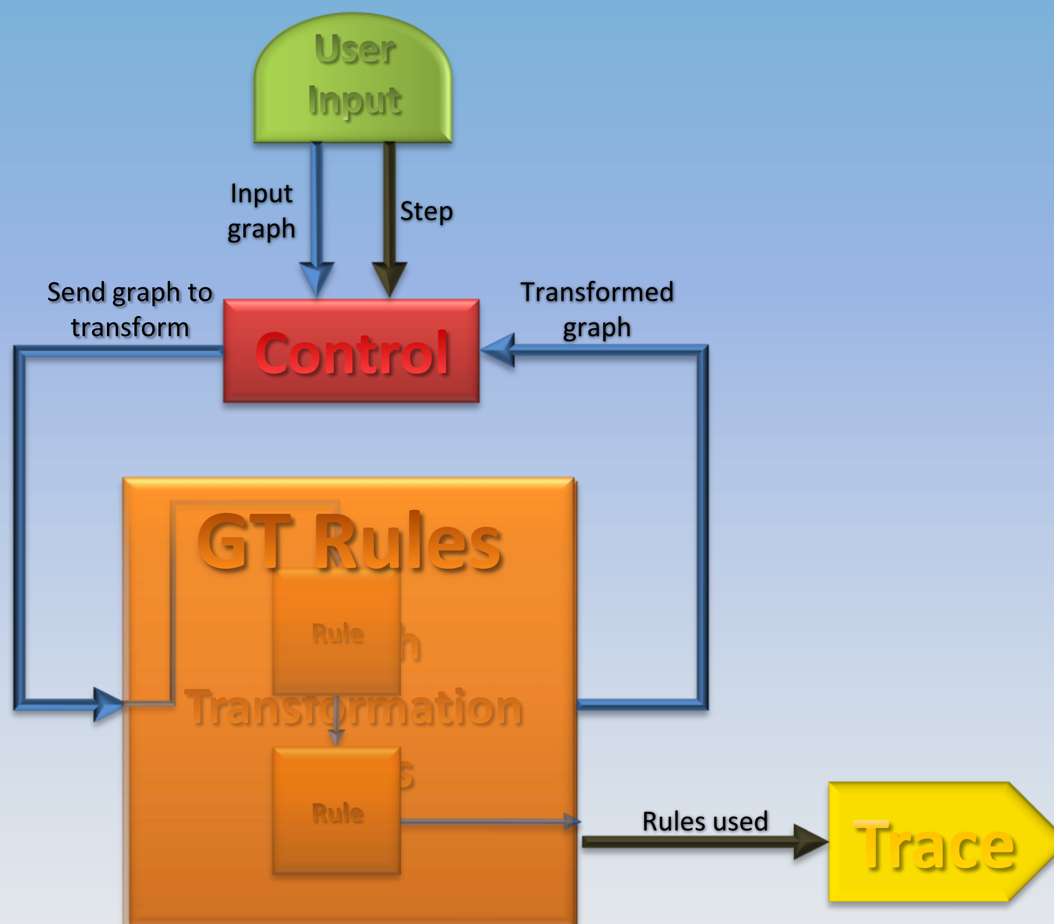


Programmed Graph Rewriting: MoTif

McGill

MOTIF

Model AToM³'s graph transformation engine

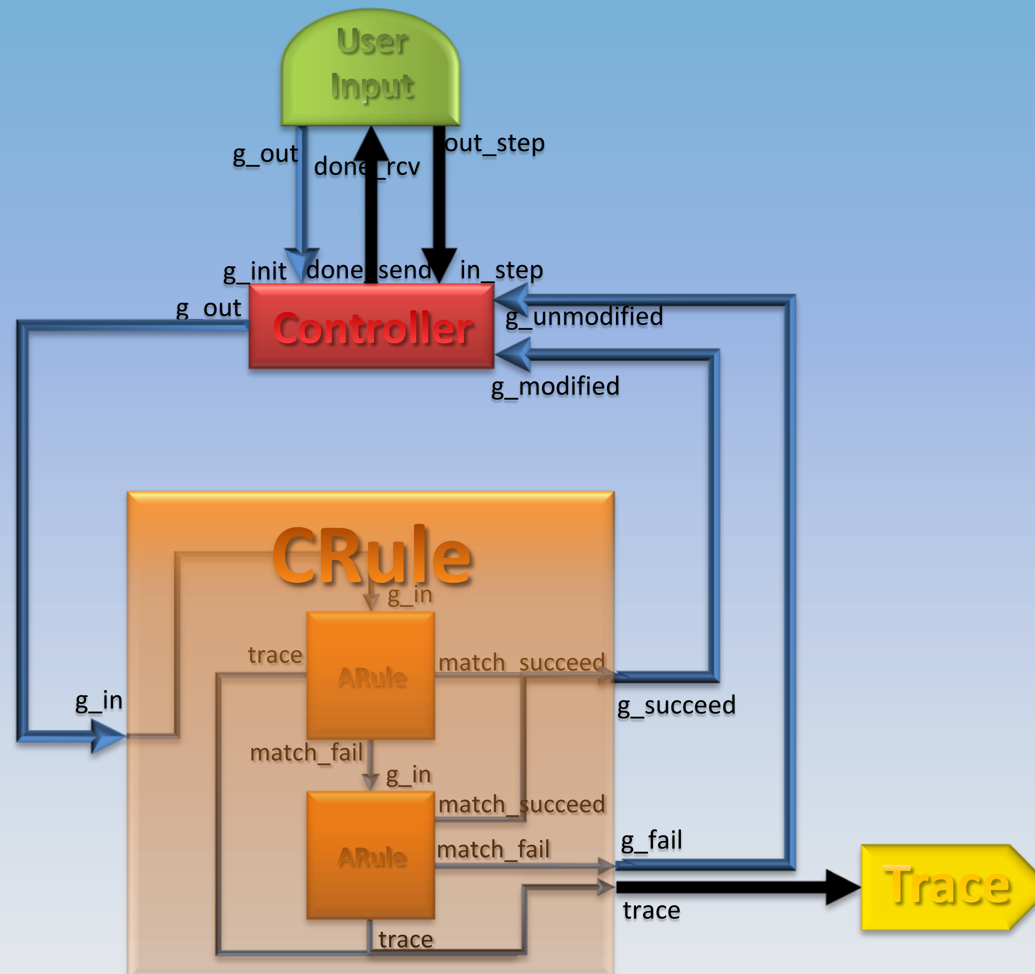


Programmed Graph Rewriting: MoTif

McGill

MOTIF

Graph transformation engine

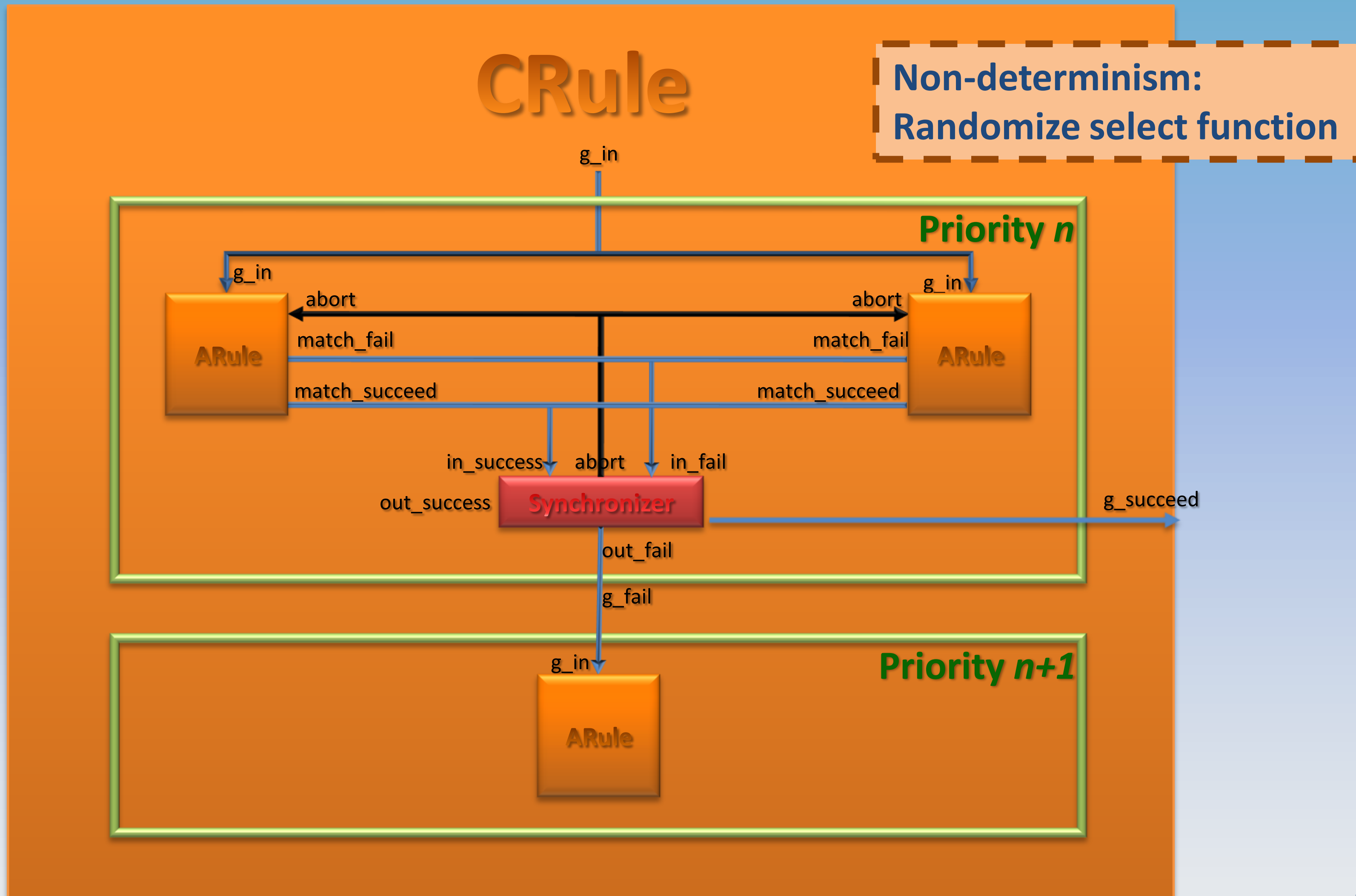


Programmed Graph Rewriting: MoTif

McGill

MOTIF

Managing priorities



MOTIF

Compiling the rules

Given the ASG of the model

- **Loading**

1. Build *N={label:[Nodes]}* the list of the potential nodes of the isomorphic subgraph with the LHS label as key
2. Initialize *M* empty

- **LHS Match**

3. From each root, perform an incremental DFS that finds the nodes (path) one by one
4. When path is complete, build *M={label:node}* of matched nodes
5. If M is incomplete: FAIL

- **RHS application**

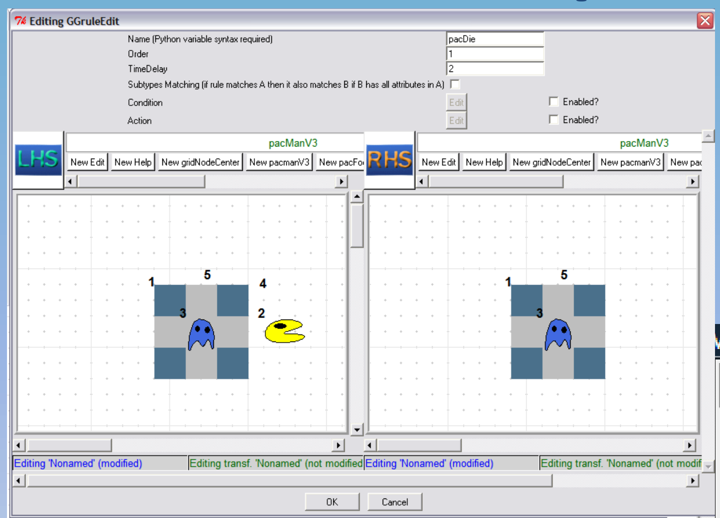
6. Remove nodes from the in/out-connections and from the graph dictionary
7. Create new nodes in the graph dictionary and connect them
8. Modify attributes
9. SUCCESS

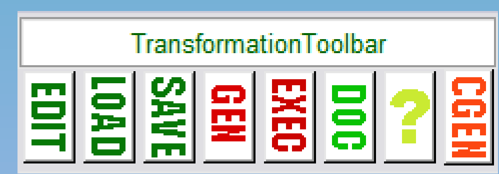
Programmed Graph Rewriting: MoTif

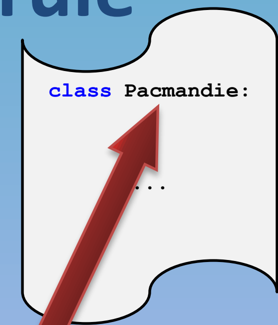
McGill

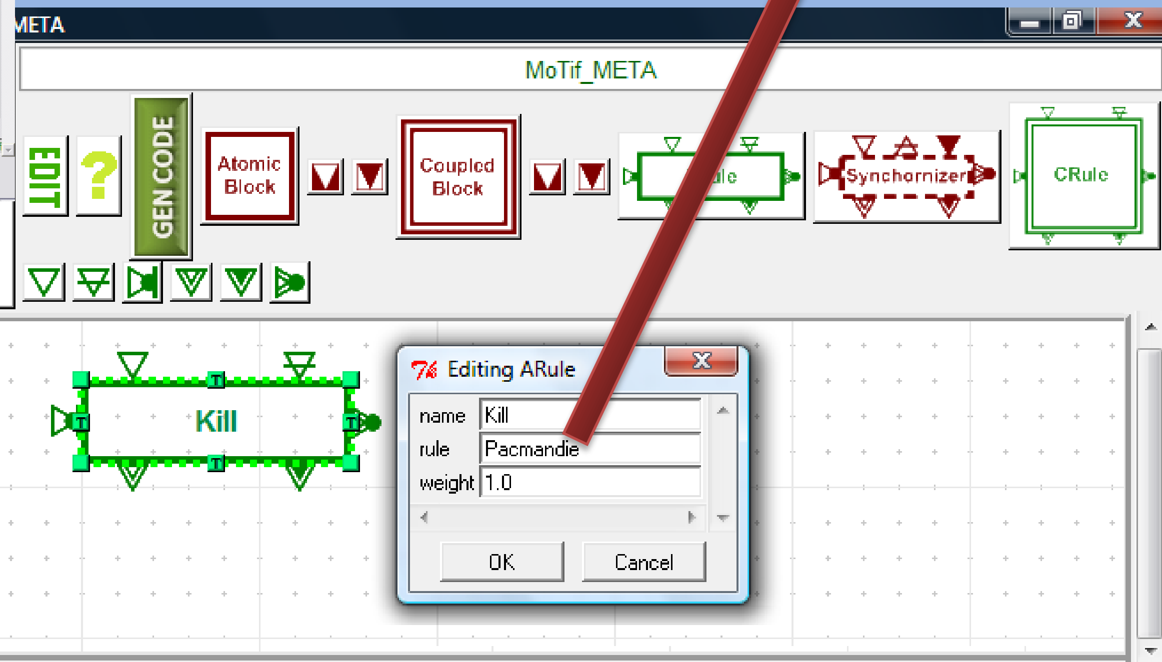
MOTIF

Use compiled version of AToM³'s rule









MOTIF

Generating pyDEVS code

Given the ASG of the model

- **Pre-generation**

1. Generate Executer for simulation and Event classes
2. Generate the Environment: User, Controller and interface with the model

- **Model to DEVS:** *Visitor Pattern*

3. For each Atomic block,
   Call the DEVSGenerator with parameters of the block (transitions, functions, inports, outports)
4. For each Coupled block,
   Keep information of the block (select, inports, outports)
5. For each Coupled block,
   Open the submodel and goto 3
   Call the DEVSGenerator with its information

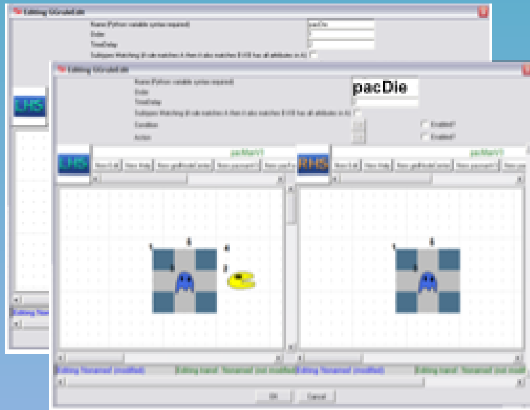
- **DEVS to code:** *Command Pattern*

6. Atomic block code generator
7. Coupled block code generator
8. Special blocks code generator
9. Use of predefined constants

McGill

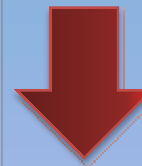
Programmed Graph Rewriting: MoTif

MOTIF

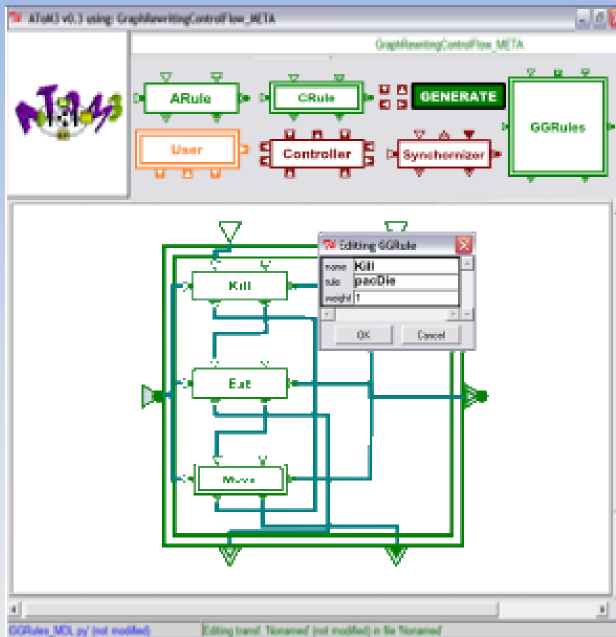


GENERATE

```
class pacDie:
  def execute(graph):
    ...
```



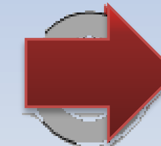
IMPORT



GENERATE

```
class GGRuleKill:
  def __init__(self):
    self.state = KillState('pacDie')
    self.graph_in = self.addInport()
    self.success_out = self.addOutPort

  def extTransition():
    graph = self.peak(self.graph_in)
    g = self.state.rule.execute(graph)
    ...
    return self.state

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

  def outputFnc():
    if (self.state.isMatched):
      self.poke(self.success_out, self.state.graph)
    ...

  def timeAdvance():
    return self.state.weight
```

SIMULATE



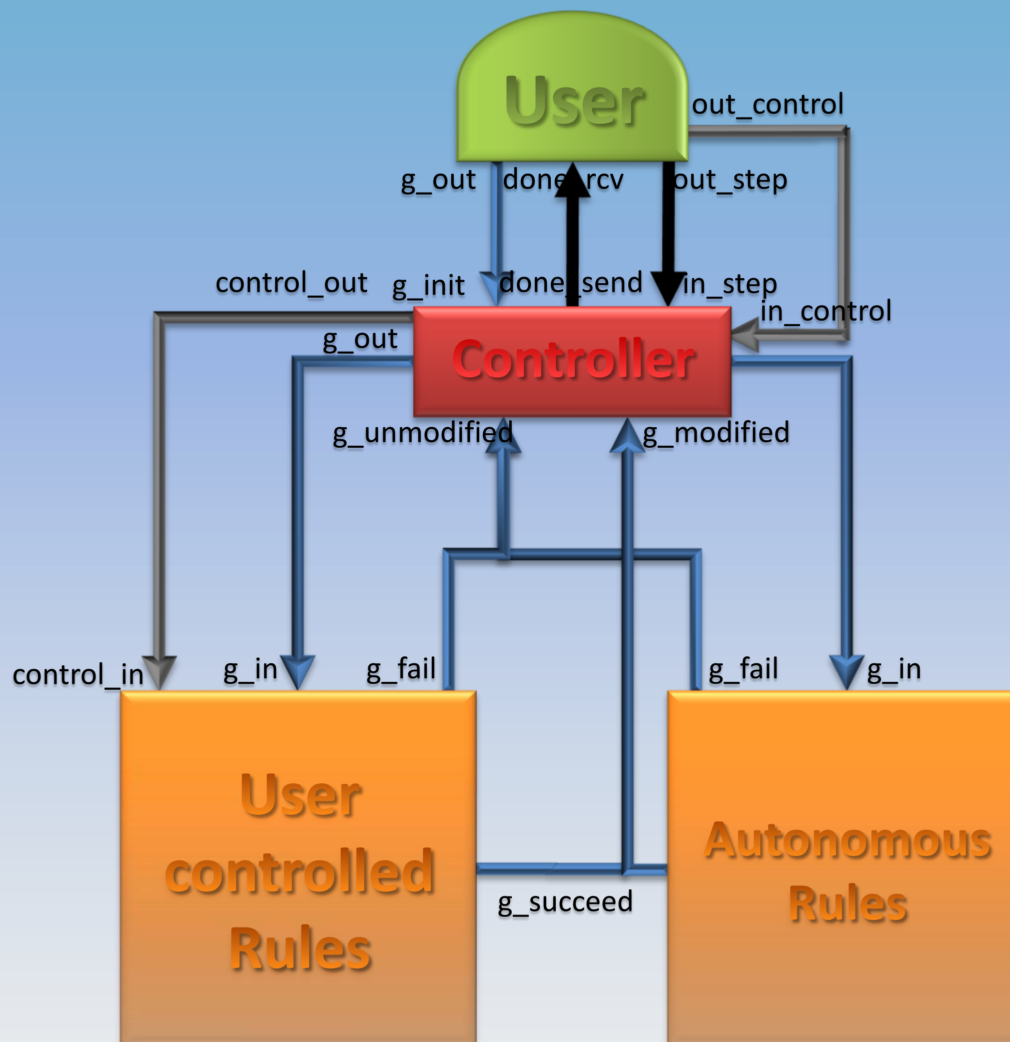
```
Rules Used
>1:  PacManUp
>2:  Eat
>3:  PacManLeft
>4:  Eat
>5:  PacManDown
>6:  Eat
>7:  PacManDown
>8:  Eat
>9:  GhostRight
>10:  GhostDown
>11:  Kill
>12:  GhostUp
>13:  GhostLeft
>14:  GhostUp
>15:  GhostLeft
>16:  GhostRight
```

Programmed Graph Rewriting: MoTif

McGill

MOTIF

Extension of AToM^3's graph transformation engine

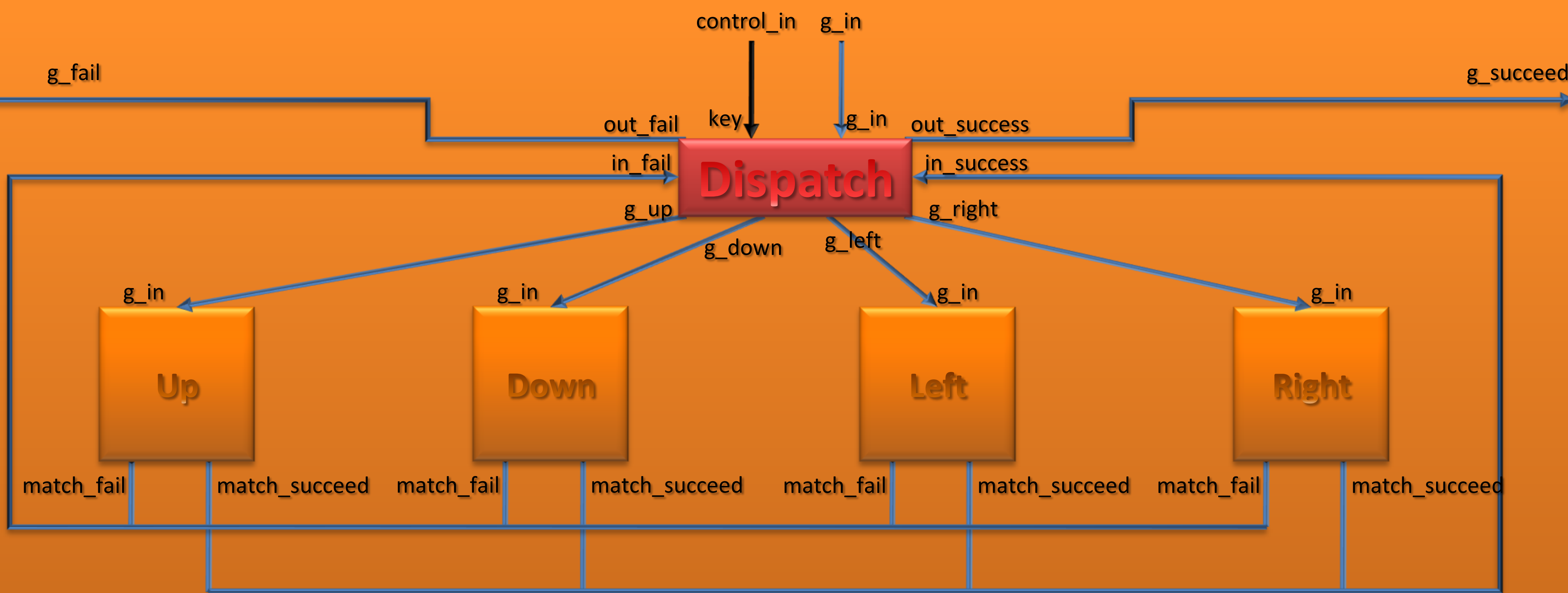


Programmed Graph Rewriting: MoTif

McGill

MOTIF

Extension of AToM^3's graph transformation engine

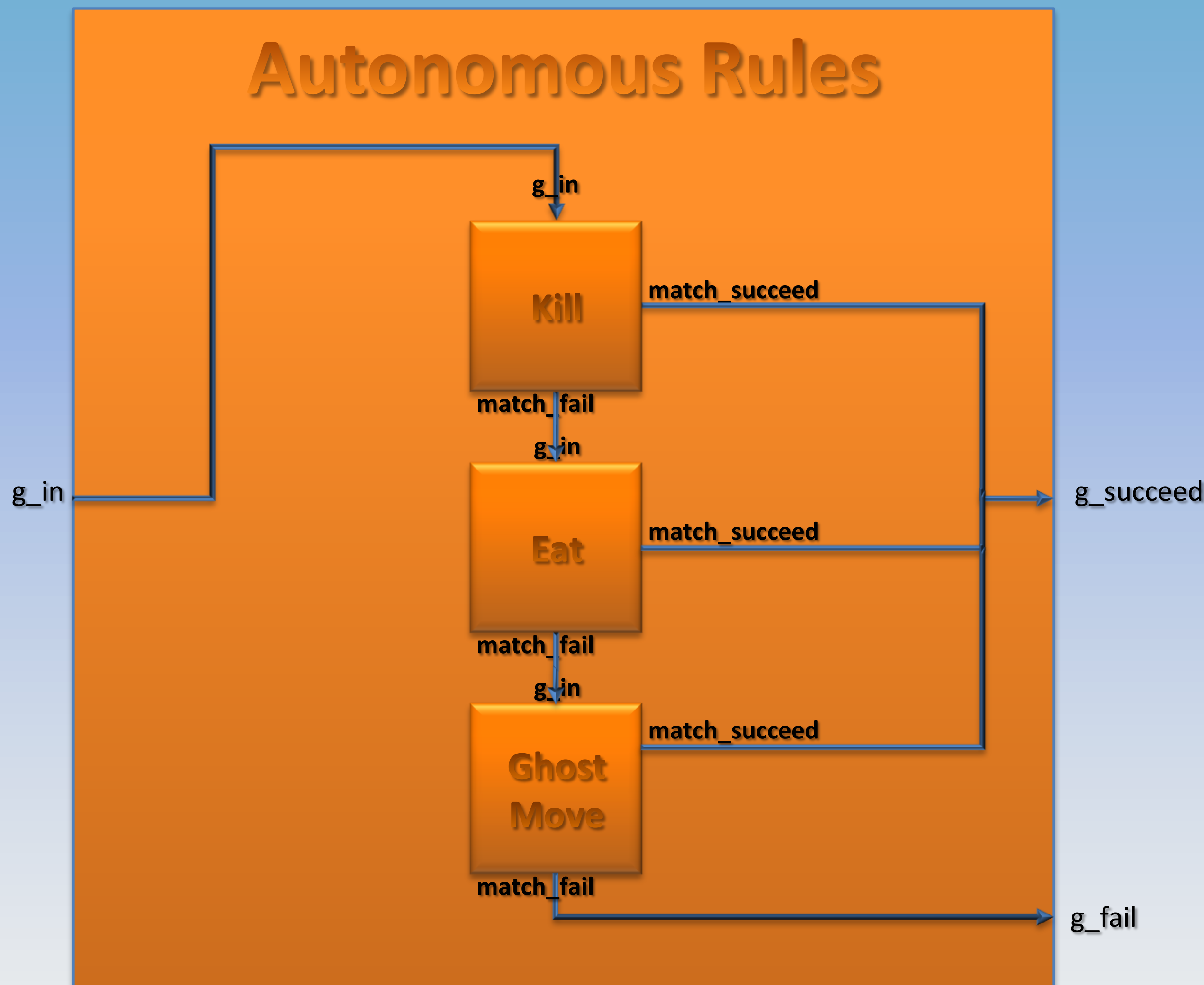


Programmed Graph Rewriting: MoTif

McGill

MOTIF

Extension of AToM3's graph transformation engine



MOTIF

Priorities revisited

- Controlled Transformations
  - Layered Transformations
    - Prioritized Transformations

OVERVIEW

- In the context
- Existing Programmed Graph Rewriting Systems
- AToM^3's graph rewriting engine by example
- Modelled and Modular Timed Graph Transformation (MoTif): Mimic AToM^3 and beyond
- Conclusion and Future Work

SUMMARY

Control flow structure properties satisfied

- ✓Sequence
- ✓Branching
- ✓Looping
- ✓Hierarchy + Modularity
- ✓Parallelism
- ✓Time

CONCLUSION AND FUTURE WORK

Parallelism

- DEVS is a sequential formalism
- Parallel-DEVS
- Kiltera (CSP-like languages)

IN COMPLETION FUTURE WORK

Time

- Metric, Statistics
- Timed graph transformation
- Real-Time DEVS
- Simulation…

COMPLETED FUTURE WORK

User - Events

- Event-driven Graph Rewriting
- Modelling of the user

➢ Web-based pacman game
- AJAX
- SVG
- Real-time

FUTURE WORK

Some Extensions

- Optimization on rule application
- Information on the flow
- Replace python code by... Statechart? Modelica? Kermeta?

MORE REFERENCES

13. Progres website: http://www.se.rwth-aachen.de/tikiwiki/tiki-index.php?page=Research%3A+Progres

14. Fujaba website: http://wwwcs.uni-paderborn.de/cs/fujaba

15. MOFLON website: www.moflon.org

16. VMTSwebsite: http://vmts.aut.bme.hu

Programmed Graph Rewriting: MoTif

McGill

Are we at the right level of abstraction?

Is it the way industry should go in model transformation?