MoTif: The Modular Timed Graph Transformation Language

Hans Vangheluwe
Eugene Syriani

Modelling, Simulation and Design Lab
McGill University
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
CONTEXT

- Meta-Model + Model Transformation + ...
- Model Transformation $\Rightarrow$ 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

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

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS
From UML to JAVA And Back Again (FUJABA) [4,5]


EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

MOFLON [6]

EXISTING PROGRAMMED GRAPH REWRITING SYSTEMS

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

RUNNING EXAMPLE WITH AToM³ [12]

Simplified PacMan formalism [13]

RUNNING EXAMPLE WITH AToM³

Build the Meta-Model of the PacMan formalism
RUNNING EXAMPLE WITH AToM³

Model the Graph Transformation
Graph Transformation Rules

RUNNING EXAMPLE WITH ATOM³

1: return self.LHS.nodeWithLabel(1).score + 1
RUNNING EXAMPLE WITH ATOM³

Graph Transformation Rules

1. ghostLink
2. gridLeft
3. gridRight
4. pacLink
5. gridRight
6. gridRight
7. ghostLink
RUNNING EXAMPLE WITH AToM³
RUNNING EXAMPLE WITH ATOM³

- Capture execution trace:
  - Keep log of used rules
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
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]
• **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 MODELLING ENVIRONMENT

- Atomic Rule
- Synchronizer
- Coupled Rule
- Atomic DEVS
- Coupled DEVS
- pyDEVS Generator
MODEL AToM³’S GRAPH TRANSFORMATION ENGINE
GRAPH TRANSFORMATION ENGINE
MANAGING PRIORITIES

Non-determinism: Randomize select function
class Kill(ARule):
    def __init__(self):
        ARule.__init__(self,
                       name='Kill')
        self.state = ARuleState(Pacmandie())
        def weightFunction(self):
            return 1.0

class Pacmandie:
    def match():
        ...
    def execute():
        ...

class Pacmanmove:
    def match():
        ...

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 ATOM$^3$’S GRAPH TRANSFORMATION ENGINE
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
Decider finds the next move for the ghost

Decider consumes time
MODELLING OF TRANSFORMATION: SYSTEM
MODELLING OF THE TRANSFORMATION: USERCONTROLLED \textit{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
• Next Mode event:
  - Cumulative
  - Roll-back
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
ONE STEP FURTHER

Time

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