COMP 763B - Modeling and Simulation Based Design

Rule Based Operational Semantics Specification in Ptolemy

Yanwar Asrigo

30th April 2008

Overview

- Recap: Ptolemy
- Problem Statement
- Implementation
- Testing

Ptolemy II [1]

heterogenous modeling and design

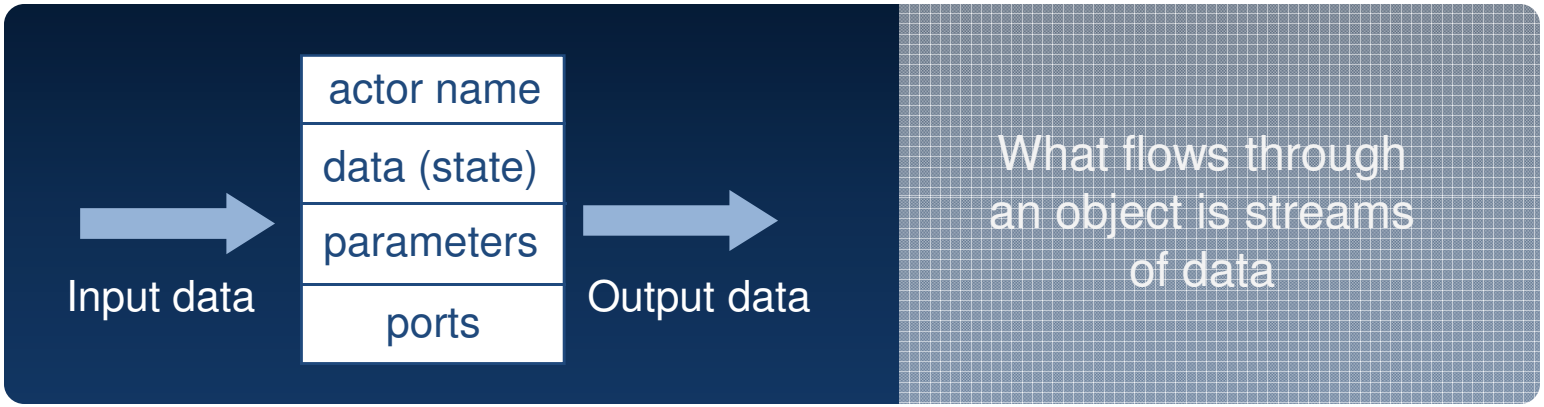
- Started at UC Berkeley by Prof. Edward Lee
- Studies modeling, simulation, and design of concurrent, real-time, and embedded systems
- Developed using Java and truly free software (GPL)
- Special features:
  - Actor Orientation + Domain
  - Hierarchically Heterogeneous Model

---

[1] Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Nuendorffer, S., Sachs, S., Xiong, Y. : Taming Heterogeneity – The Ptolemy Approach. Proceedings of the IEEE 91(1) (Jan 2003) 127-144.

Actor Orientation

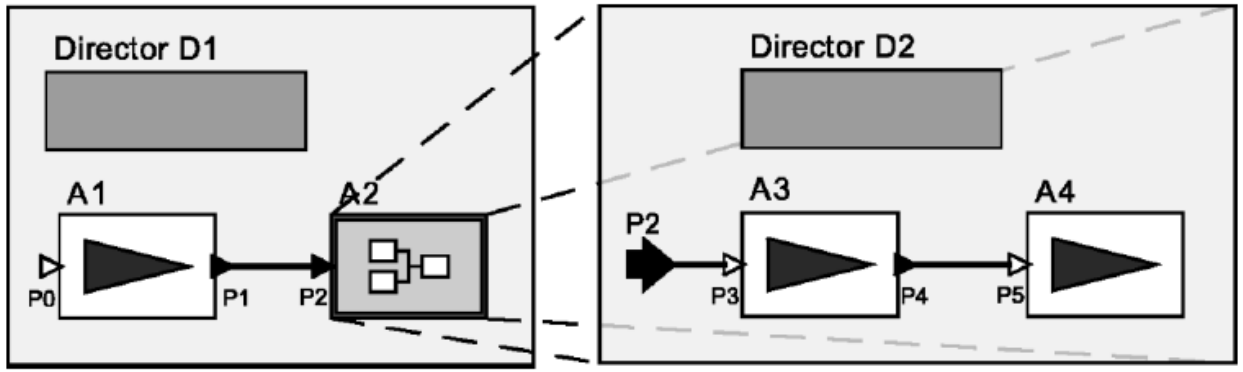
- Actors are concurrent components that communicate through ports by passing messages



- Actor-oriented view of a system decouples the **transmission of data** from the **transfer of control**.

Domains

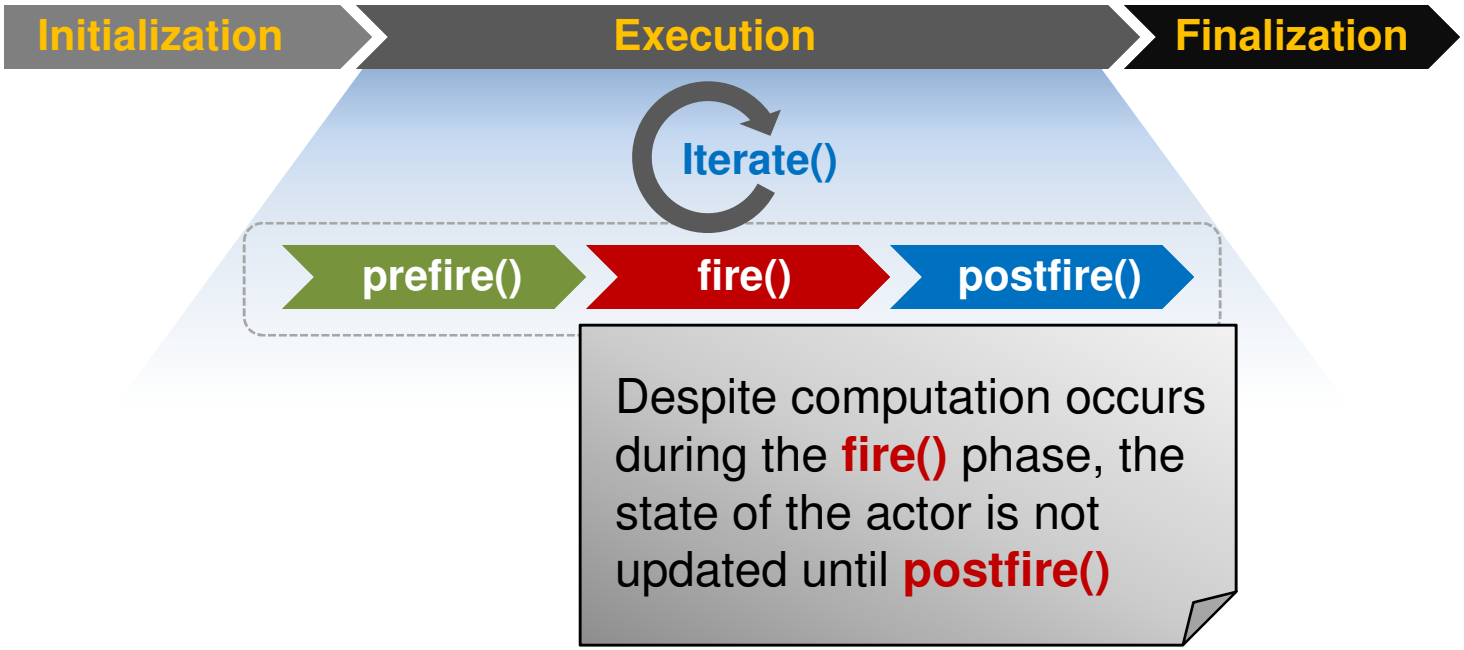
- Domains provide semantic models for component interactions. Domain is realized by:
  - **Director**: governs the execution of a composite entity
  - **Receiver**: implements the communication semantics



Director D1 Controls the execution order of actors A1 and A2

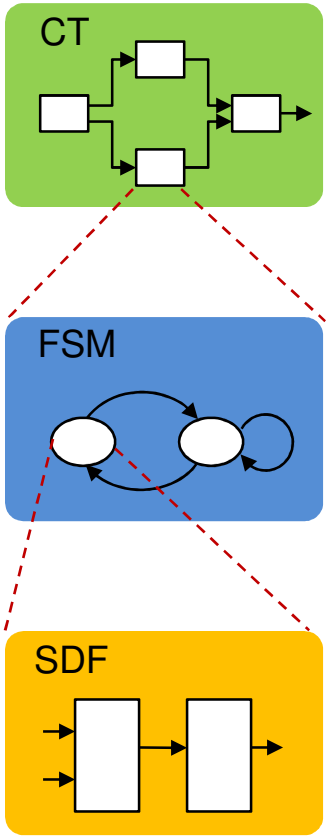
Director D2 controls the execution of A3 and A4 whenever A2 is executed.

Actor Execution



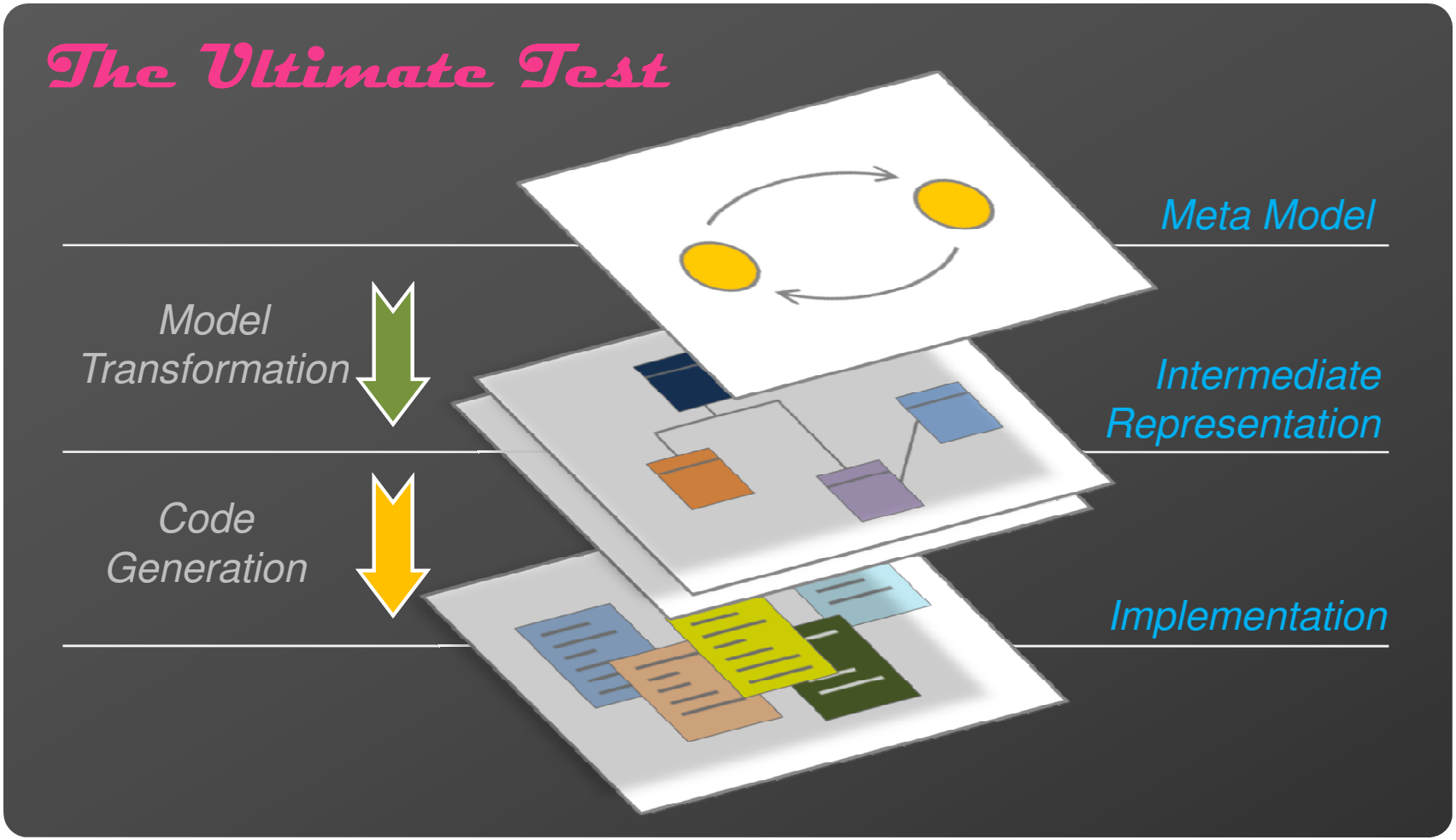
- The domain of a composite actor determines how the iteration of one actor is related to the iterations of other actors in the same composite

Hierarchical Heterogeneous Composition



- Divide a complex model into a tree of nested submodels
- Each level is composed of a network of interacting components
- Network at each level is **locally homogenous**, while allowing different interaction mechanisms to be specified at different levels in the hierarchy

What is Missing? Meta-Modeling!



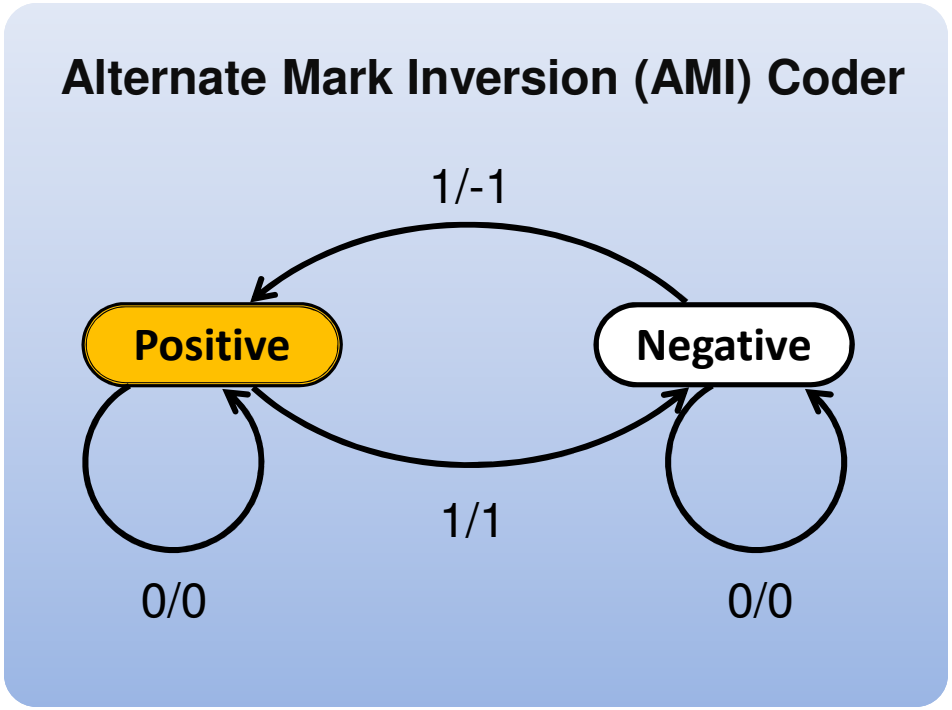
The Project

- Using external tool for Meta-modeling formalisms and automatically synthesize the codes for Ptolemy

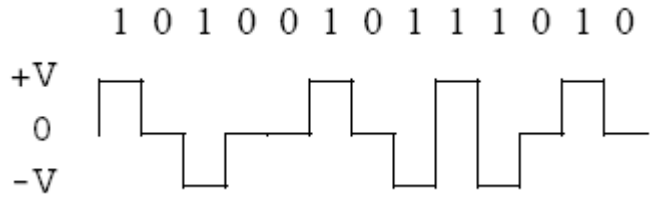
Specifically:

- Specify operational semantics using rules and compile these rules into Ptolemy's code (Java)
- Tools: AToM^3 and Motif
- Target Formalism: Finite State Automata / Machine

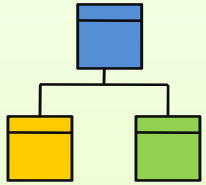
Finite State Automata (Example)



Input	Output
0	0
1	1
1	-1
1	1
0	0
1	-1

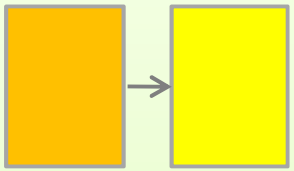


Specifying and Compiling Operational Semantics Rules in AToM^3



Meta-model of domain specific notation

Finite State Automata Meta-Model



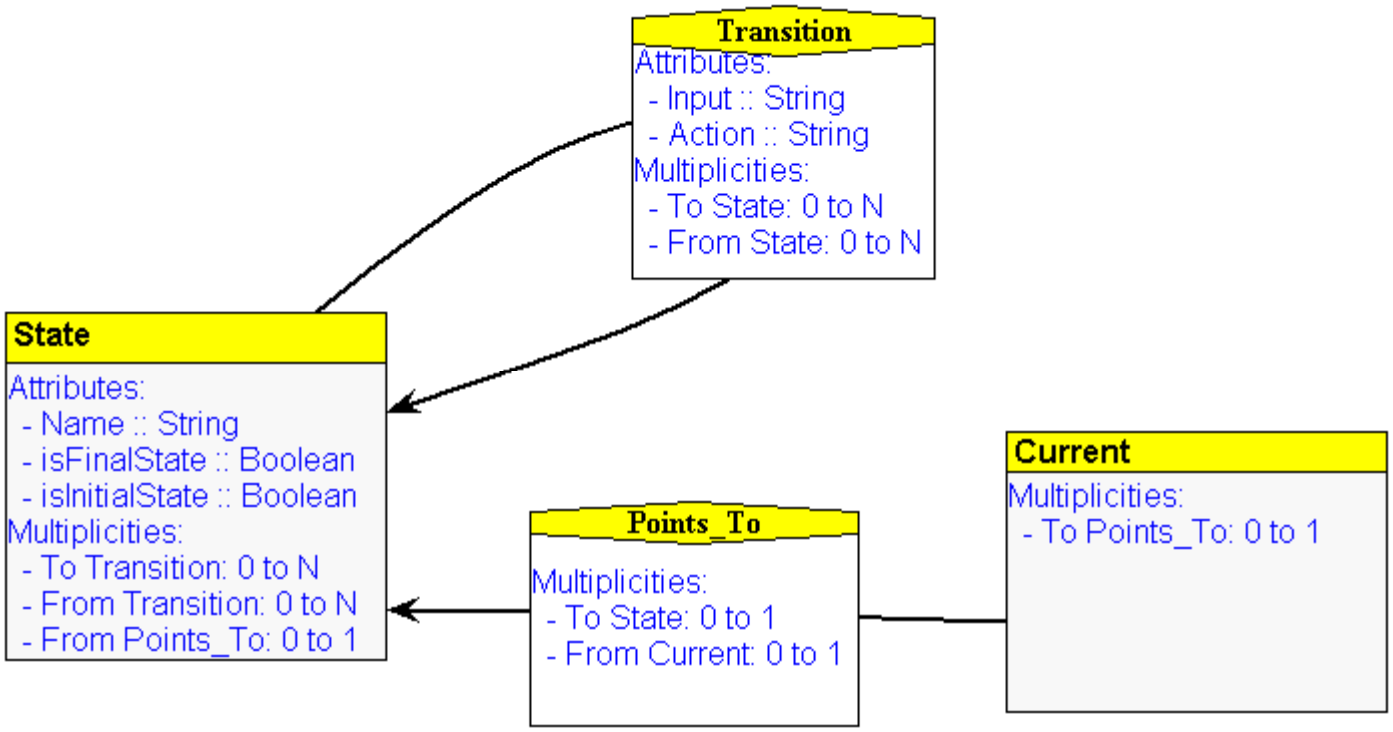
Specify operational semantics as graph transformation rule

Graph Grammar Rule defining FSM execution transformation

Compiled rule-based operational semantics

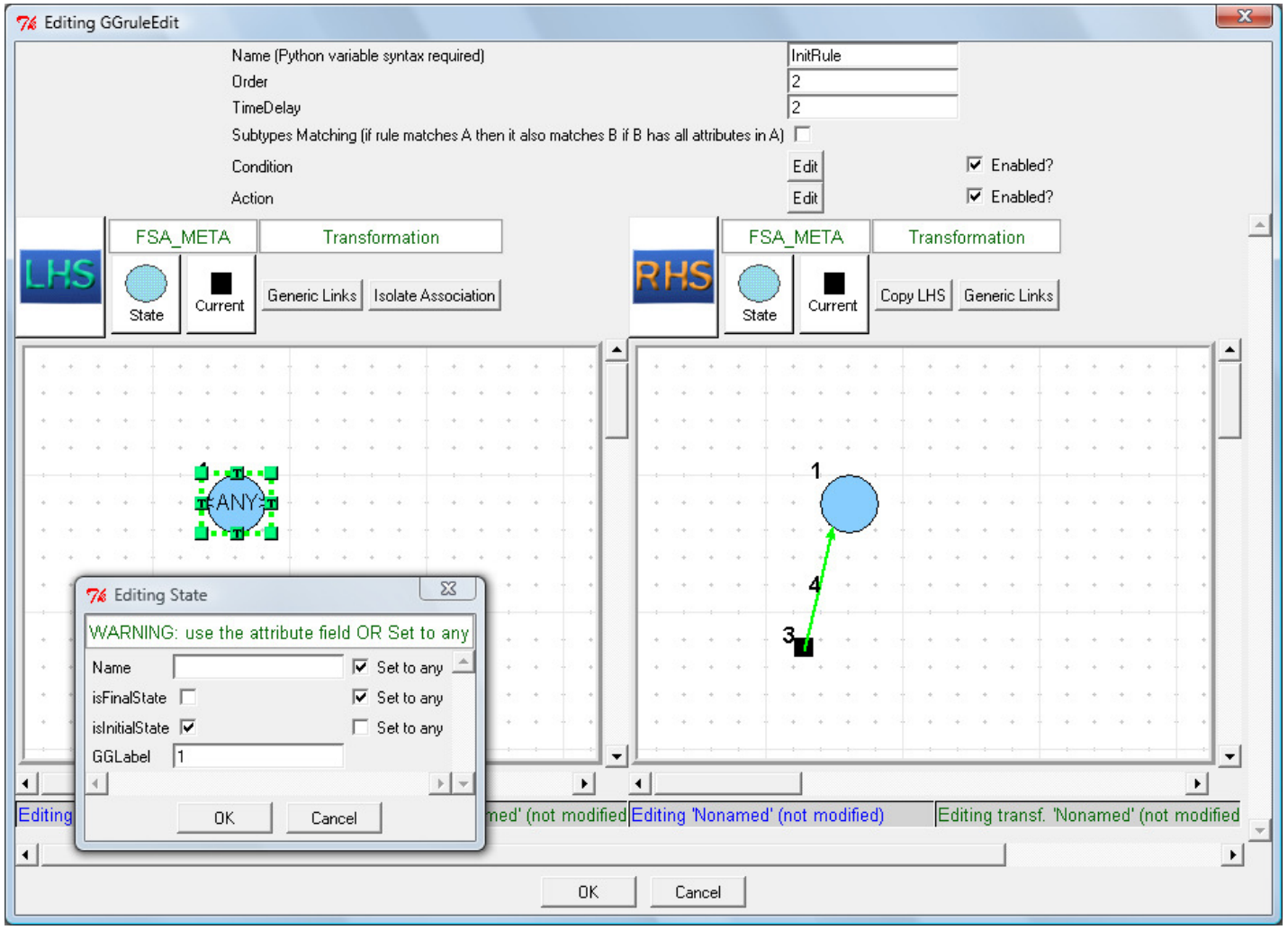
Compiled GGRule in Java

Finite State Automata Meta Model

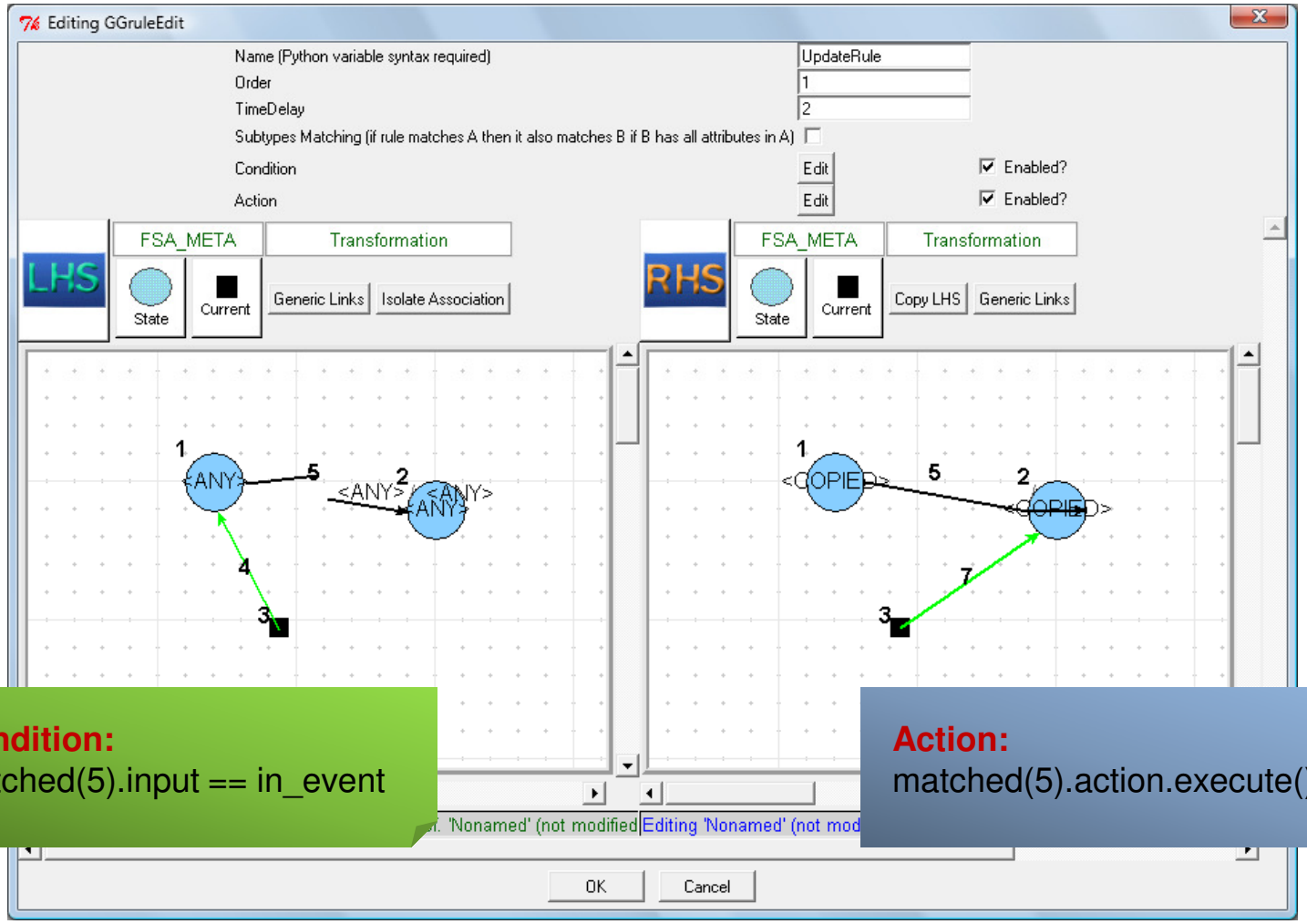


* This can be transformed to UML class diagram which eventually can be used to generate the code

Initialization Rule



Update Rule

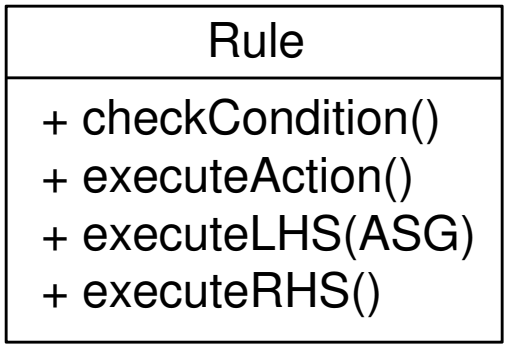


Condition:
matched(5).input == in_event

Action:
matched(5).action.execute()

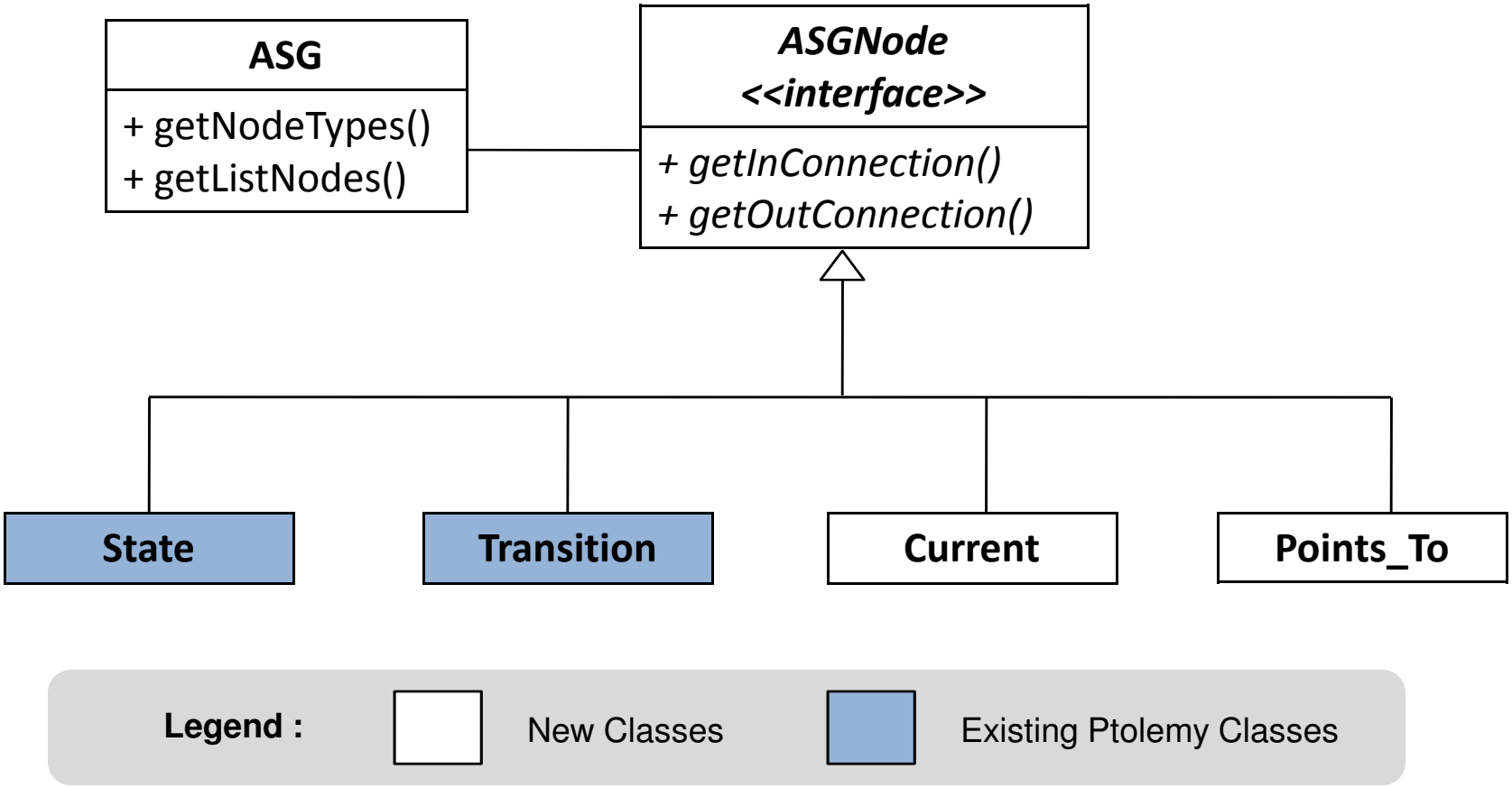
Motif [2] GG Compiler

- Used Motif GG compiler to extract and compile rule defined in AToM^3
- Modified to produce Java code suitable for Ptolemy
- Logic is mainly based on AToM^3 graph representation
  - Ptolemy data structure is modified to support this

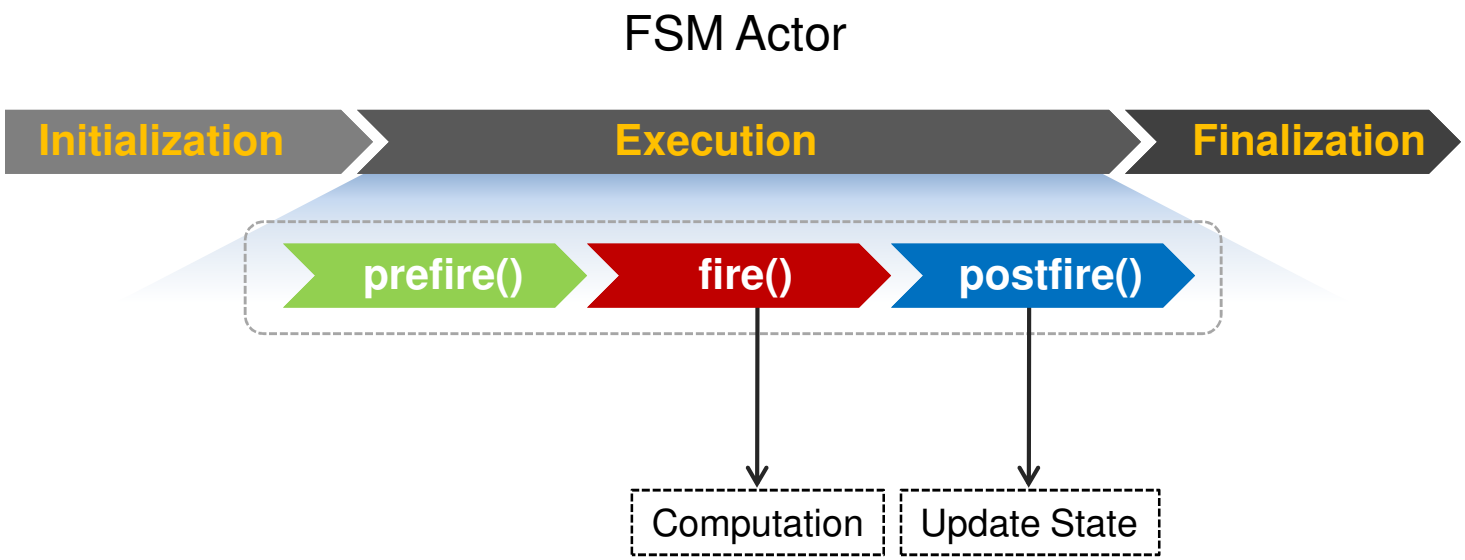


[2] Eugène Syriani and Hans Vangheluwe. Programmed Graph Rewriting with DEVS. In Manfred Nagl and Andy Schür, editors, Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007), Lecture Notes in Computer Science (LNCS). Springer-Verlag, October 2007. Kassel, Germany.

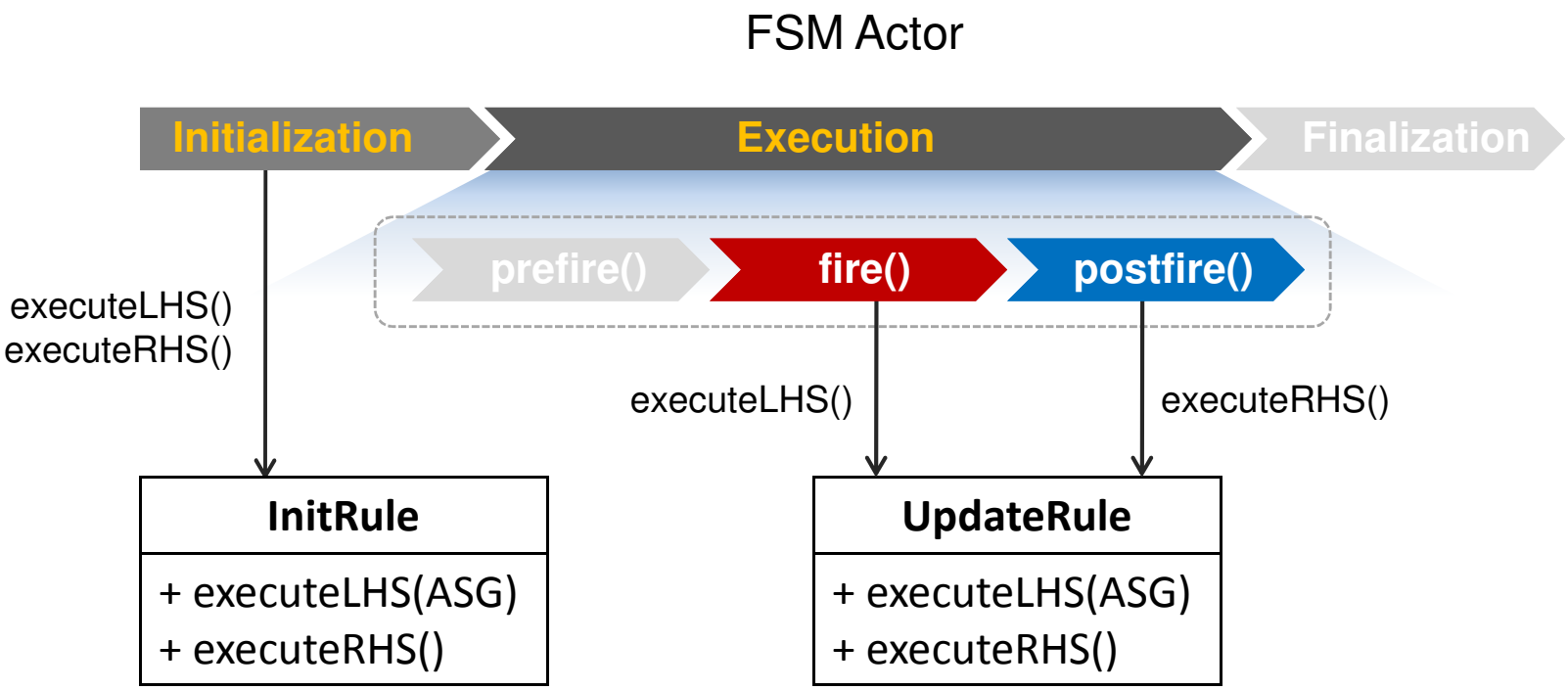
Modifications in Ptolemy (1)



Modifications in Ptolemy (2)

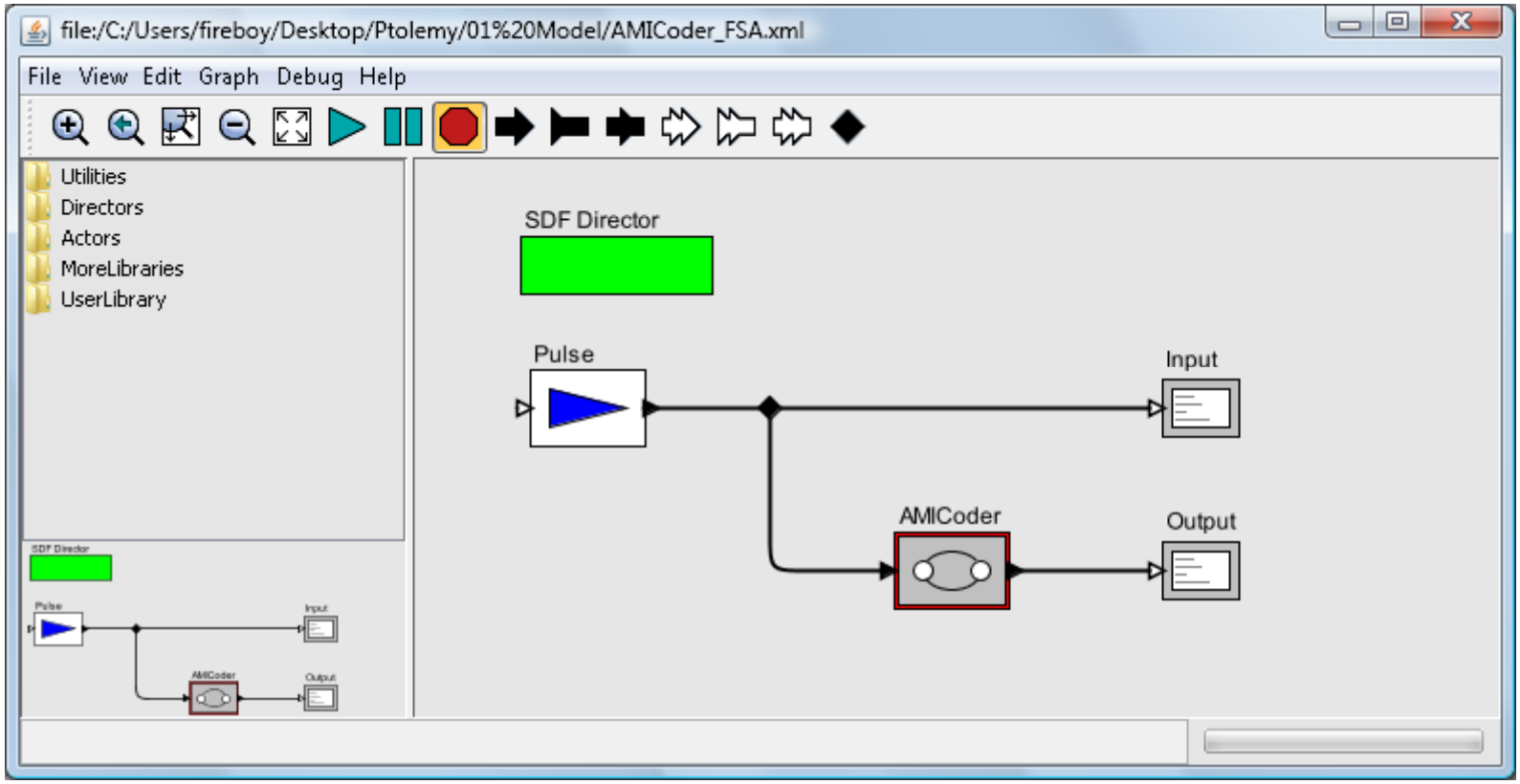


Modifications in Ptolemy (2)

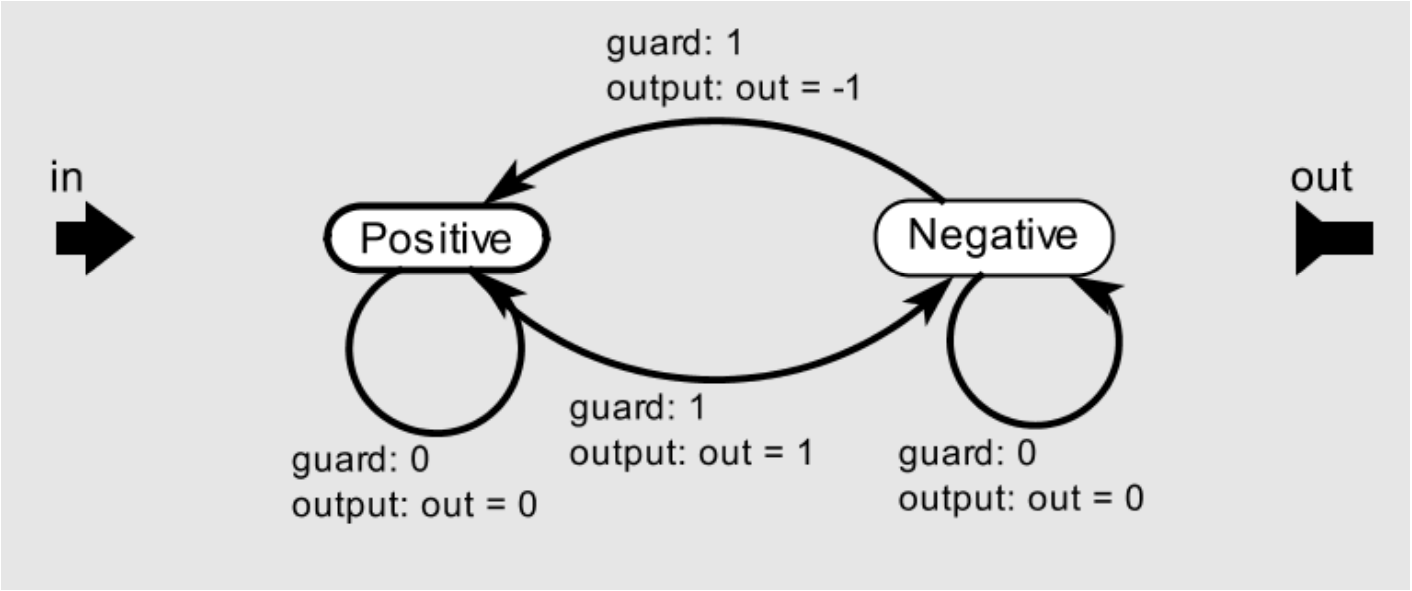


- **fire()** corresponds to the **left hand side matching**, and
- **postfire()** corresponds to **transforming the matched sub-graph to right hand side** of the transformation rule

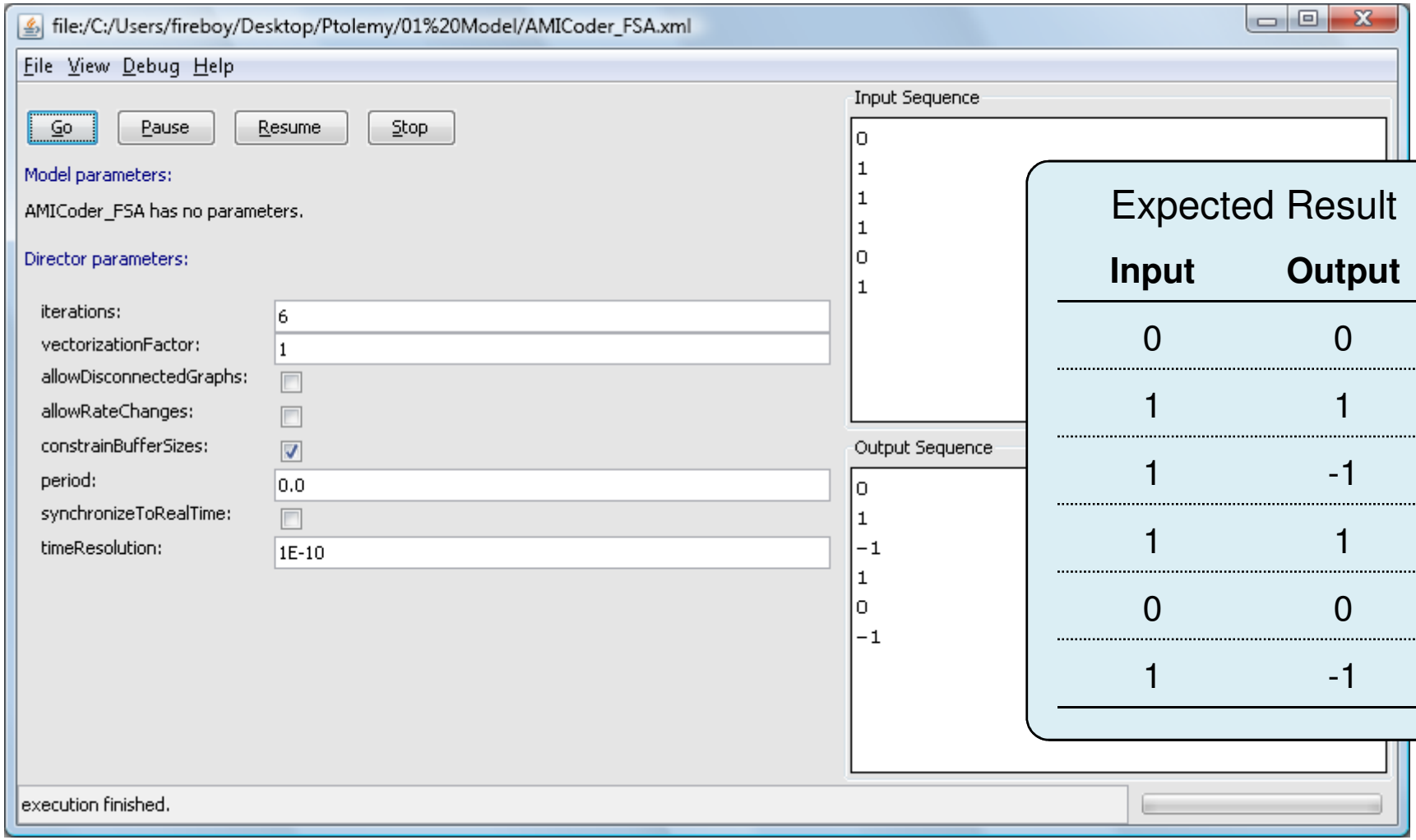
Testing – AMI Coder



Testing – AMI Coder



Testing – AMI Coder



Expected Result

Input	Output
0	0
1	1
1	-1
1	1
0	0
1	-1

Thank You