DEVS Standards Group meeting

Winter Simulation Conference 2001

Washington, DC

December 11, 2001

DEVS standardization: some thoughts

Hans Vangheluwe

Juan de Lara, Jean-Sébastien Bolduc, Ernesto Posse

Modelling, Simulation and Design Lab (MSDL)

School of Computer Science

McGill University

Montréal, Canada

hv@cs.mcgill.ca

http://moncs.cs.mcgill.ca

Presentation Overview

1. Previous experiences with Modelling/Simulation/Standardization
2. Standardizing . . . what ?
   (a) The DEVS formalisms
   (b) The DEVS model representation
   (c) The DEVS model-solver interface
   (d) The DEVS model libraries
3. Meta-modelling
   - Architecture: modelling formalism syntax and semantics
   - Examples of Meta-modelling in AToM^3
4. Meta-modelling syntax and semantics of xyz-DEVS

Previous experiences with Modelling/Simulation/Standardization

- Formalism – Modelling Language – Simulation Model – Libraries
- DAE++ – Modelica – DSblock – Modelica Standard Library
- PDE + DAE – MSL-USER – MSL-EXEC – WEST++ model library
- PDE + ODE + ALG – OOCSMP – Java – OOCSMP library
- Python-(classic)DEVS (with ports)
- Meta-modelling syntax and semantics of Causal Block Diagrams

Standardizing . . . what ?

1. The DEVS formalisms
2. The DEVS model representation
3. The DEVS model-solver interface
4. The DEVS model libraries

Standardizing the DEVS formalisms

Relationships between different variants of DEVS

1. Inheritance (specialization) – caveat: inheritance is also a transformation
2. Transformation (*e.g.,* onto Classic DEVS)
   Reasons for transformation:
   - conceptual: insight, proof of "equivalence" (morphism)
   - avoid building a new simulator. Automatically transform to formalism for which a (efficient) simulator exists.

Standardizing the DEVS model representation

For exchange and re-use

1. Between programs, agents, machines, . . .

   Needs to be platform neutral, efficiently machine readable and writable.

   Suggestion: XML.

2. Between humans

   Needs to be readable, expressive, compact.

   Suggestions: graphical (composition), textual (expressions, loops, scoping, inheritance).

3. Storage of large amounts of data models (trajectory formalism)

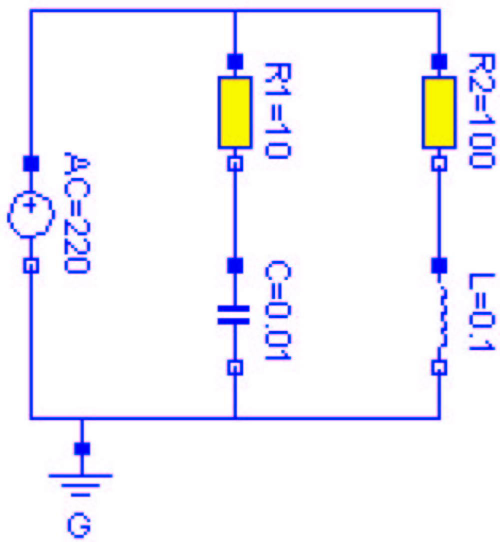
   Needs to be compact.

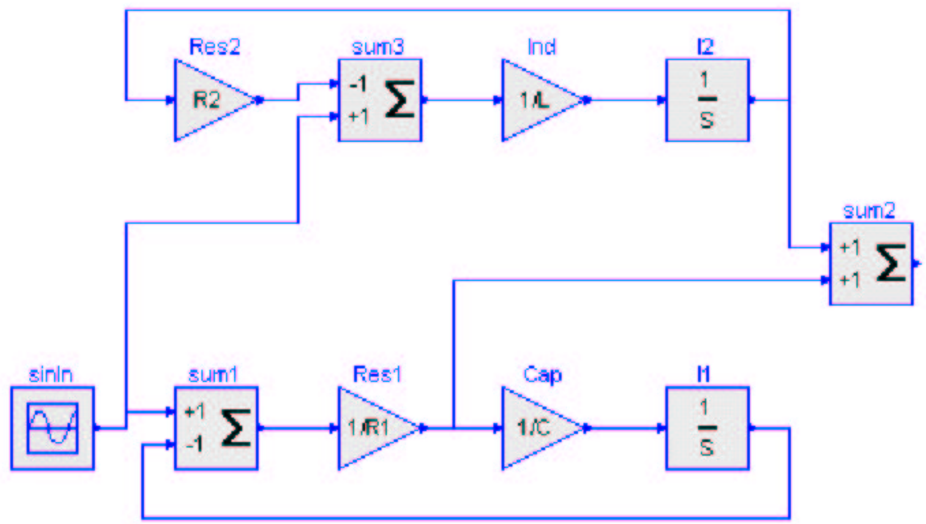
   Suggestions: binary (XDR, dbase). Least important issue.

OO Modelling in Modelica

- everything is a class
- inheritance hierarchy: from generic to specific

Electrical example: Modelica vs. Matlab/Simulink





Electrical Types

```
type Time = Real (final quantity="Time", final unit="s");
type ElectricPotential = Real (final quantity="ElectricPotential",
                               final unit="V");
type Voltage = ElectricPotential;
type ElectricCurrent = Real (final quantity="ElectricCurrent",
                             final unit="A");
type Current = ElectricCurrent;
```

Electrical Pin Interface

```
connector PositivePin "Positive pin of an electric component"
       Voltage v "Potential at the pin";
  flow Current i "Current flowing into the pin";
end PositivePin;
```

Electrical Port

```
partial model OnePort
  "Component with two electrical pins p and n
   and current i from p to n"
  Voltage v "Voltage drop between the two pins (= p.v - n.v)";
  Current i "Current flowing from pin p to pin n";
  PositivePin p;
  NegativePin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Electrical Resistor

```
model Resistor "Ideal linear electrical resistor"
  extends OnePort;
  parameter Resistance R=1 "Resistance";
  equation
    R*i = v;
end Resistor;
```

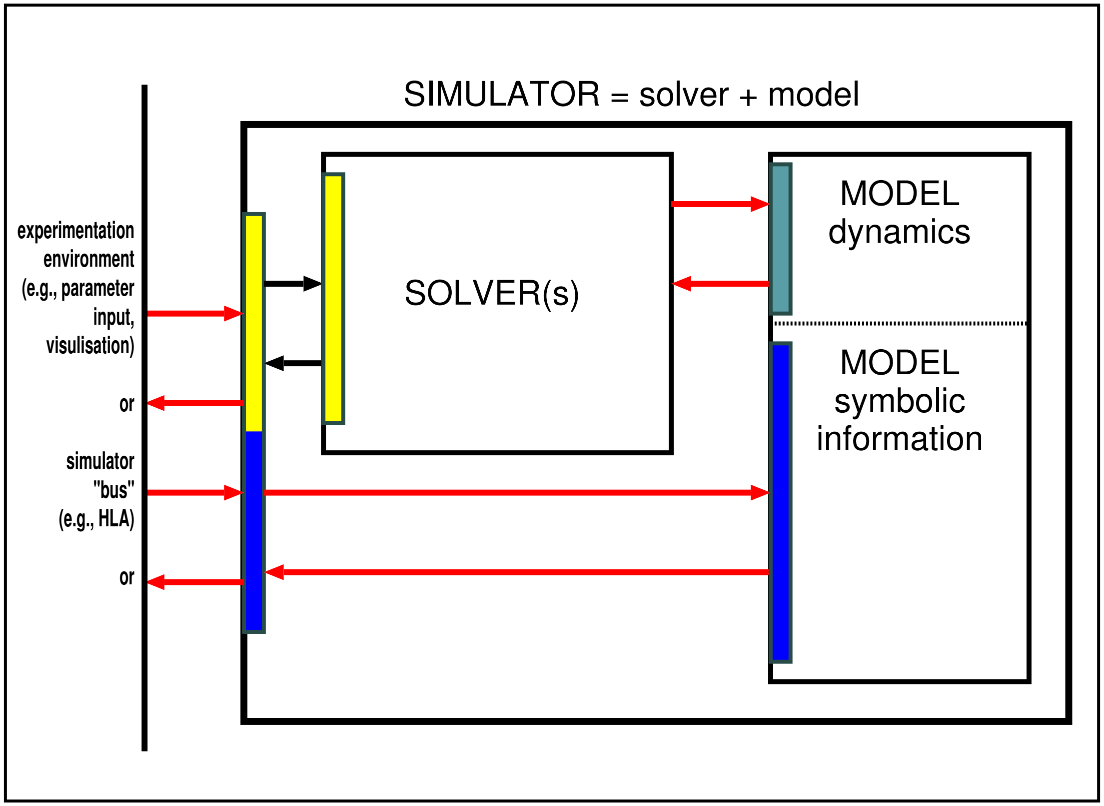
The circuit

```
model circuit
 Resistor  R1(R=10);
 Capacitor C(C=0.01);
 Resistor  R2(R=100);
 Inductor  L(L=0.1);
 VsourceAC AC;
 Ground    G;
equation
 connect(AC.p, R1.p);
 connect(R1.n, C.p);
 connect(C.n, AC.n);
 connect(R1.p, R2.p);
 connect(R2.n, L.p);
 connect(L.n, C.n);
 connect(AC.n, G.p);
end circuit;
```

Standardizing the DEVS model representation

- ability to reason about, manipulate model $\rightarrow$ model is **not** code
- language (C++, Java, . . .) independent ($x + y = z + 2$)

Standardizing the DEVS model-solver interface



Standardizing the DEVS model-solver interface

- Only the interface (API) is defined
- This allows for multiple language bindings
- Does the simulator correctly implement the formalism's semantics ? How to verify ?
  - formal proof (starting from an implementation): hard !
  - compare with automatically generated (from formal specification) reference implementation.

Standardizing the DEVS model libraries

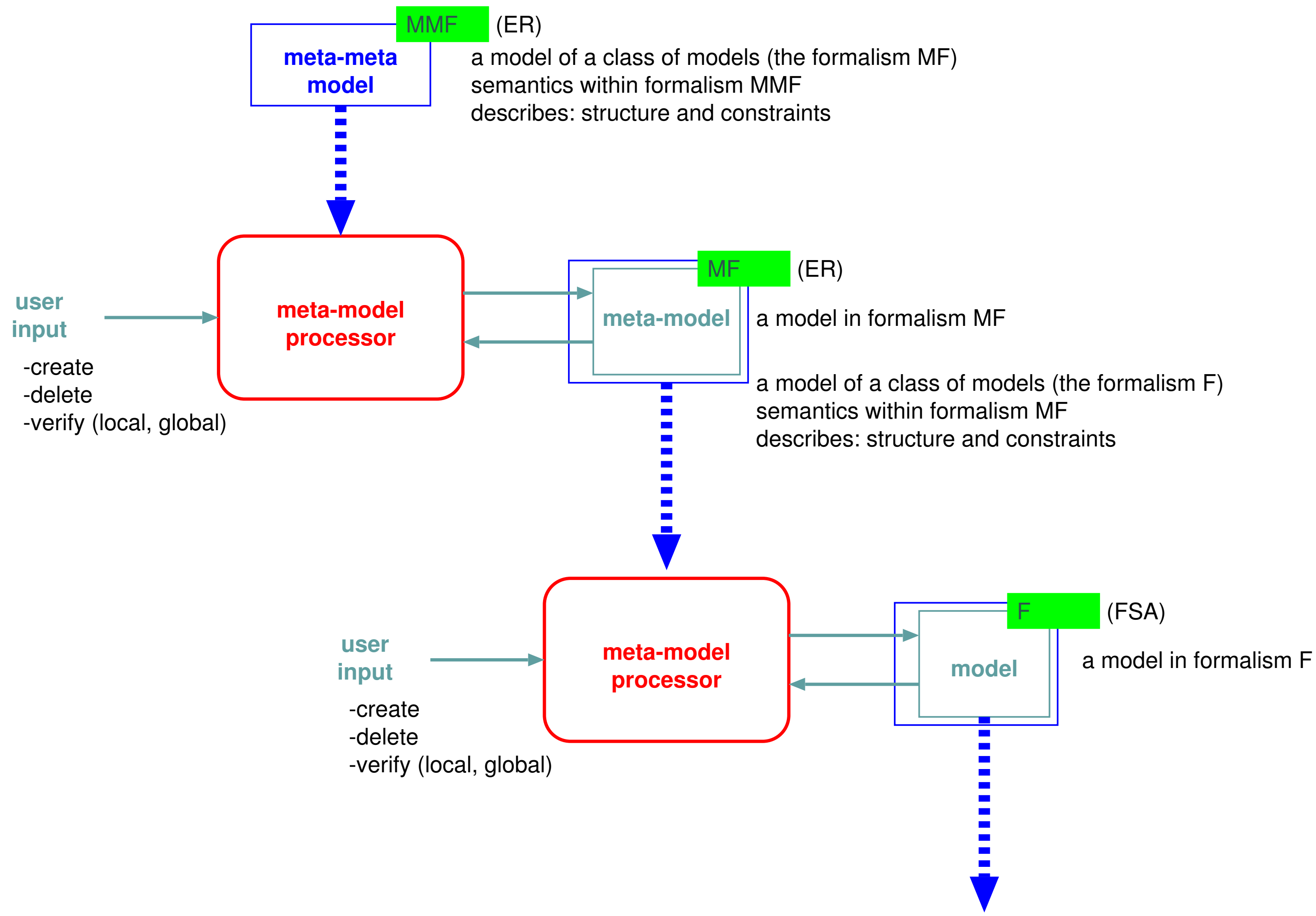
- success of language/standard depends on availability of standard libraries (in different application domains).
- success increases if re-use mechanisms are good (inheritance)
- Modelica, Extend, C++ vs. Java, ...

What is Meta-modelling ?

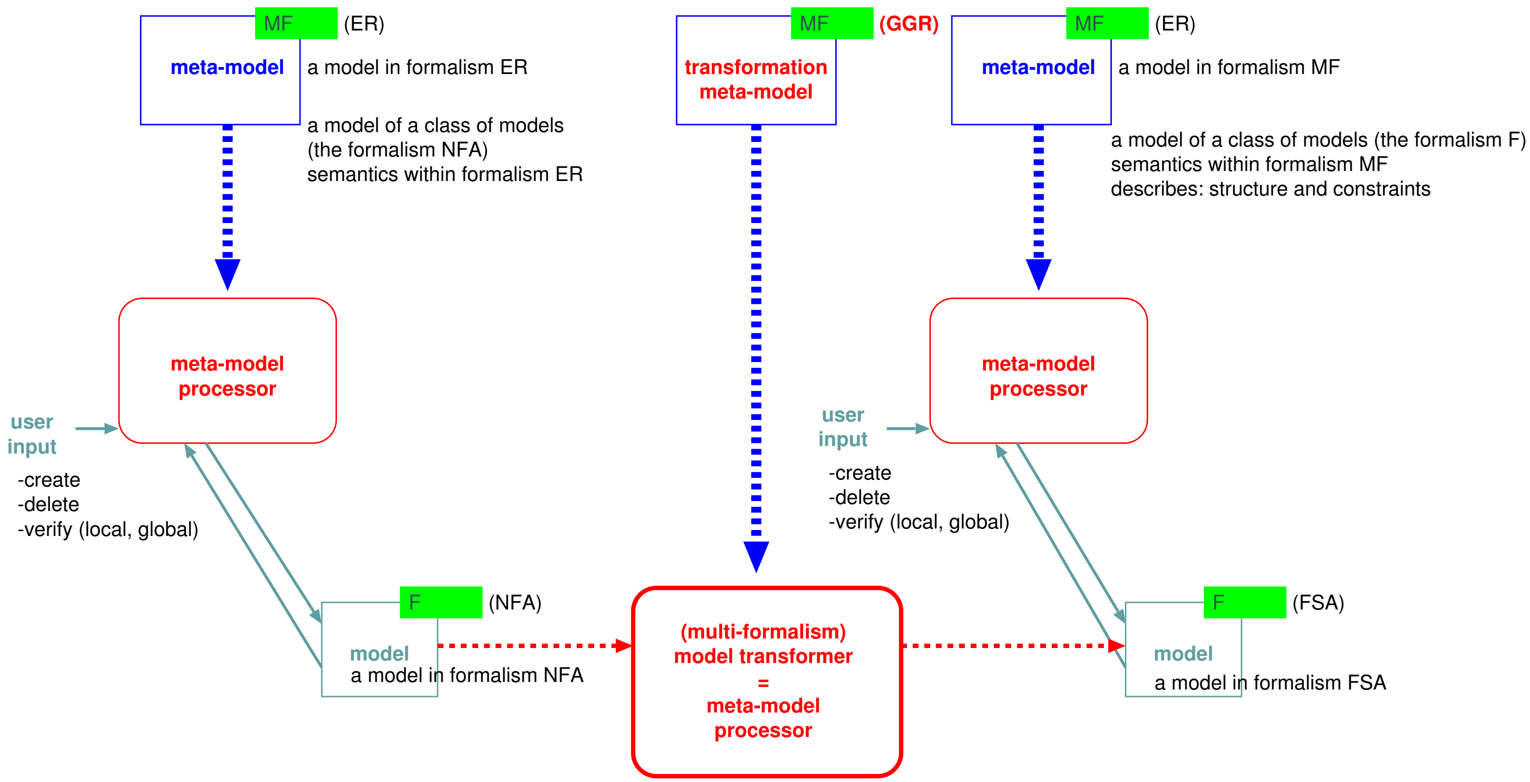
- A meta-model is **a model of** a modelling formalism
- A meta-model is itself a model. Its syntax and semantics are governed by the formalism it is described in. That formalism can be modelled in a meta-meta-model.
- As a meta-model is a model, we can reason about it, manipulate it, . . . In particular, properties of (all models in) a formalism can be formally proven.
- Formalism-specific modelling and simulation tools can *automatically* be generated from a meta-model (AToM^3).

- Formalisms can be tailored to specific needs by modifying the meta-model (possibly through inheritance if specializing).
- Semantics of new formalisms through extension or transformation (multi-formalism).

Meta-modelling architecture: syntax



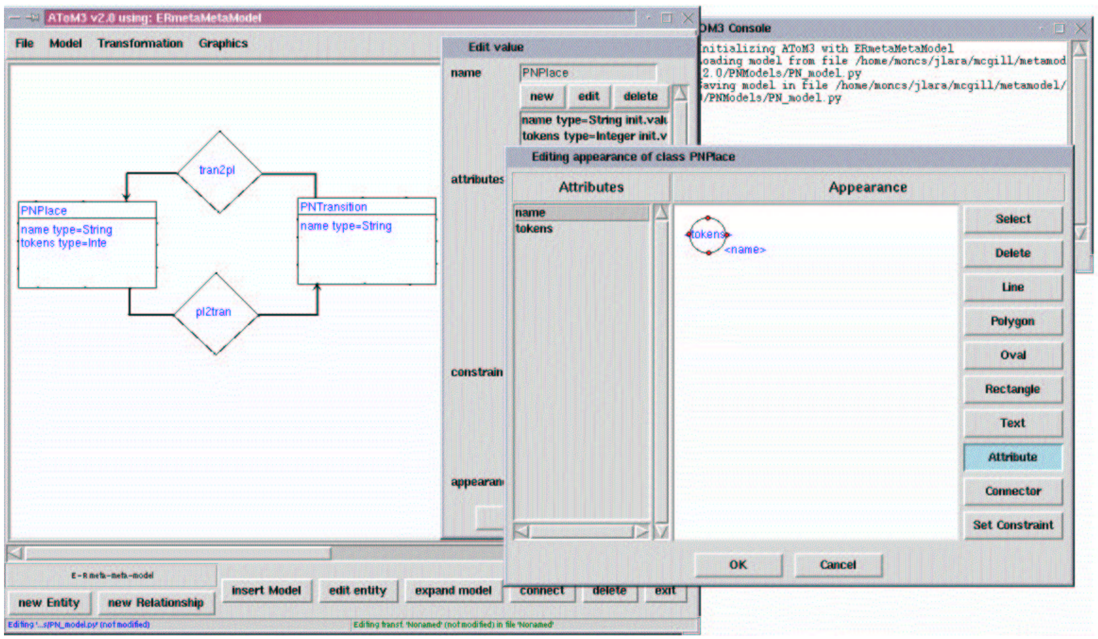
Meta-modelling architecture: transformation

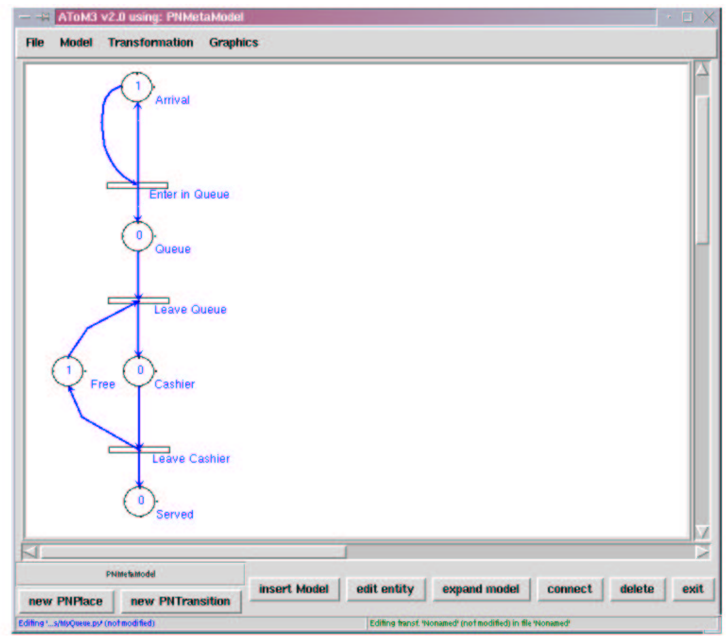


Examples of Meta-modelling in AToM^3

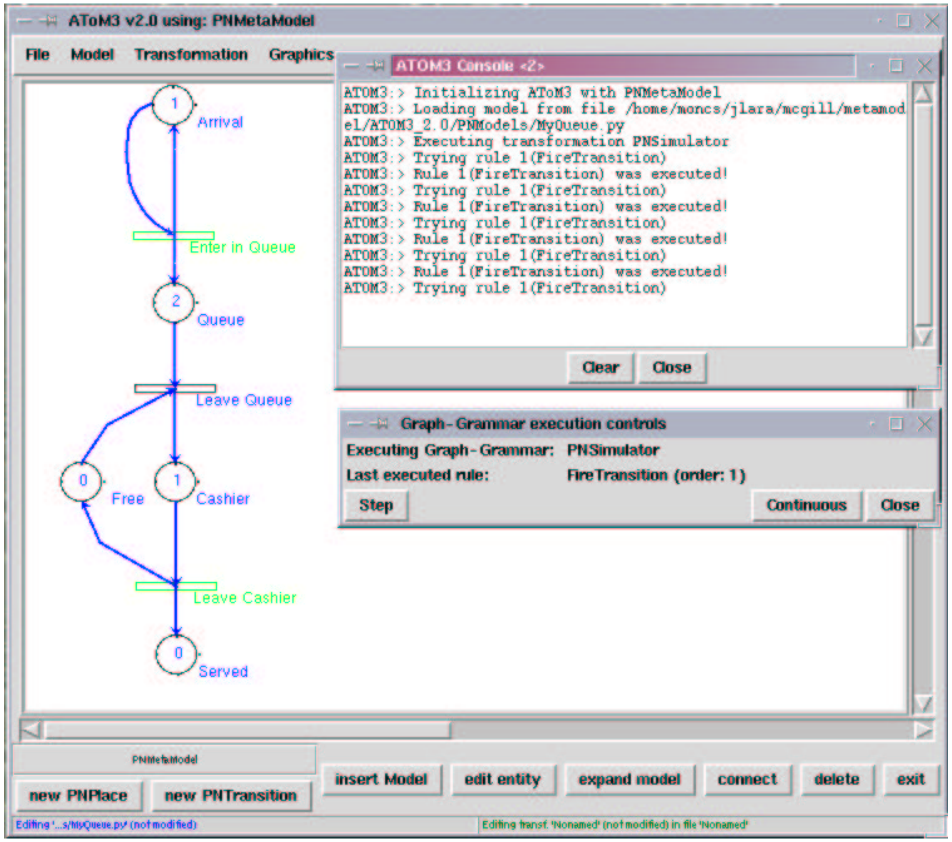
1. Petri Net Meta-model (syntax)
2. Petri Net Graph Grammar (operational semantics)
3. Petri Net Modelling and Simulation tool (reference implementation)
4. GPSS modelling environment (syntax only, semantics through code generation for existing, efficient GPSS simulator)
5. Other examples: NFA to DFA, Causal Block Diagrams, Data Flow Diagrams to Structure Diagrams, . . .

Petri Net Meta-model and generated tool

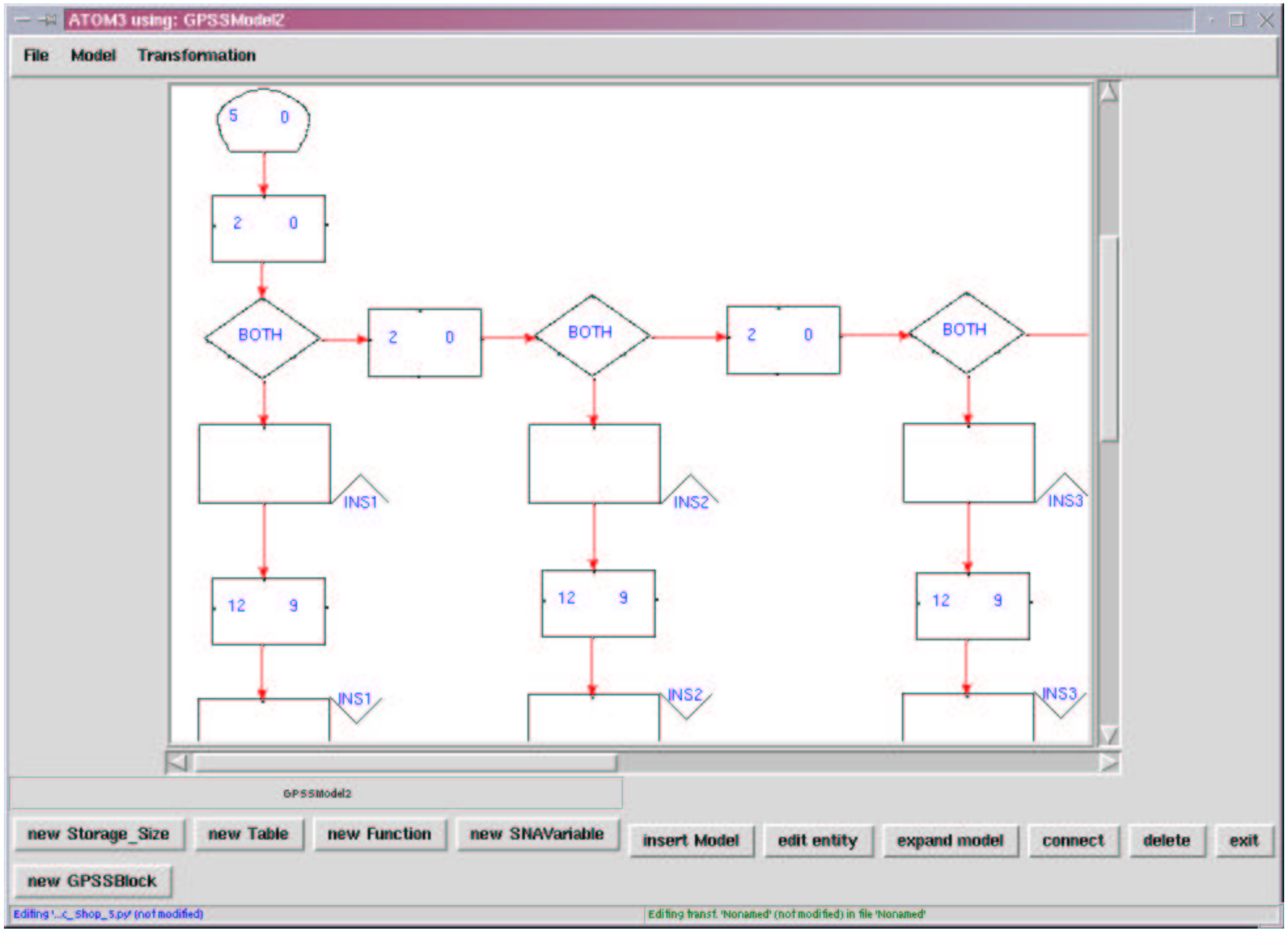




Generated Petri Net Simulator – reference impl



GPSS modelling environment



Generated Code

```
*                 Manufacturing Shop, Model 5
*                 G. Gordon
       SIMULATE
 L0    GENERATE   5,0                          CREATE PARTS
 L1    ADVANCE    2,0                          PLACE ON CONVEYOR
 L2    TRANSFER   BOTH,L3,CONV1                MOVE TO FIRST INSPECTOR
 L3    SEIZE      INS1                         GET FIRST INSPECTOR
 L11   ADVANCE    12,9                         INSPECT
 L14   RELEASE    INS1                         FREE INSPECTOR
 TAB   TABULATE   1                            MEASURE TRANSIT TIME
 L20   TERMINATE  1
 CONV1 ADVANCE    2,0                          PLACE ON CONVEYOR
 L5    TRANSFER   BOTH,L6,CONV2                MOVE TO SECOND INSPECTOR
 L6    SEIZE      INS2                         GET SECOND INSPECTOR
 L12   ADVANCE    12,9                         INSPECT
 L15   RELEASE    INS2                         FREE INSPECTOR
 L18   TRANSFER   ,TAB
 CONV2 ADVANCE    2,0                          PLACE ON CONVEYOR
 L8    TRANSFER   BOTH,L9,CONV3                MOVE TO THIRD INSPECTOR
 L9    SEIZE      INS3                         GET THIRD INSPECTOR
 L13   ADVANCE    12,9                         INSPECT
 L16   RELEASE    INS3                         FREE INSPECTOR
 L19   TRANSFER   ,TAB
 CONV3 TERMINATE  0
 1     TABLE      M1,5,5,10
       START      1000
       END
```

Meta-modelling syntax and semantics of xyz-DEVS

- Only connect ...
- Ernesto Posse: meta-modelling DEVS (variable structure, automatic bisimulation proofs)
- Jean-Sébastien Bolduc: mapping ODEs onto behaviourally equivalent DEVS
- Thierry Cornelis: meta-models $\leftrightarrow$ XML, MSL
- Hans Vangheluwe & Indrani A.V.: meta-model non-causal (Modelica) models