8 July 2008
Equation-based Object-Oriented Languages and Tools
Paphos, Cyprus

Multi-Paradigm Language Engineering and Equation-Based Object-Oriented Languages

Hans Vangheluwe

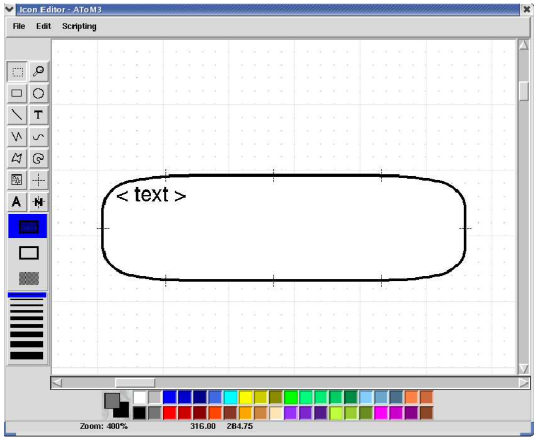
McGill

Modelling, Simulation and Design Lab (MSDL)
School of Computer Science, McGill University, Montréal, Canada

Overview

1. Multi-Paradigm Modelling (MPM)
2. Domain-Specific Modelling
3. Language Engineering and MPM Tools
4. MPM for EOOLT
5. EOOLT for MPM
6. Conclusions

Modelling a Variety of Complex Systems . . .



Multi-Paradigm modelling (minimize **accidental complexity)**

- at most appropriate **level of abstraction**
- using most appropriate **formalism(s)**
- with **transformations** as first-class models

Pieter J. Mosterman and Hans Vangheluwe.

Computer Automated Multi-Paradigm Modeling: An Introduction. Simulation 80(9):433–450, September 2004.

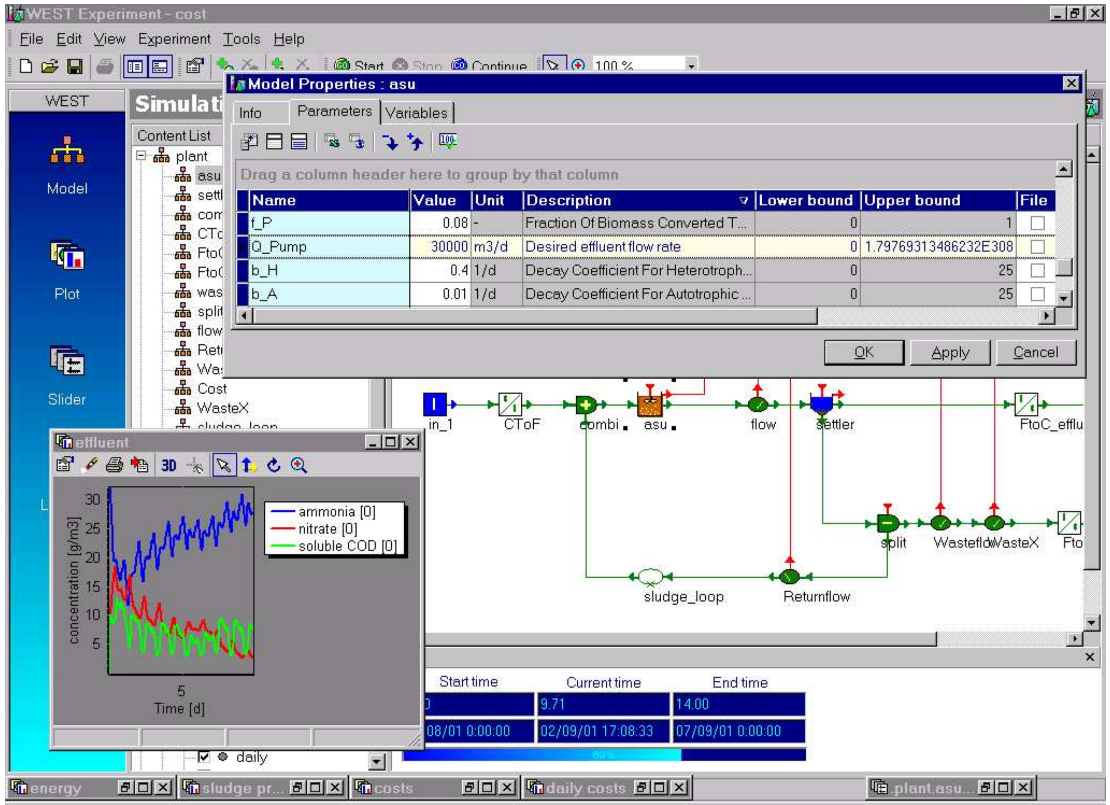
Special Issue: Grand Challenges for Modeling and Simulation.

Domain-Specific (Visual) Modelling: Waste Water Treatment Plants (WWTPs)

NATO's Sarajevo WWTP

`www.nato.int/sfor/cimic/env-pro/waterpla.htm`

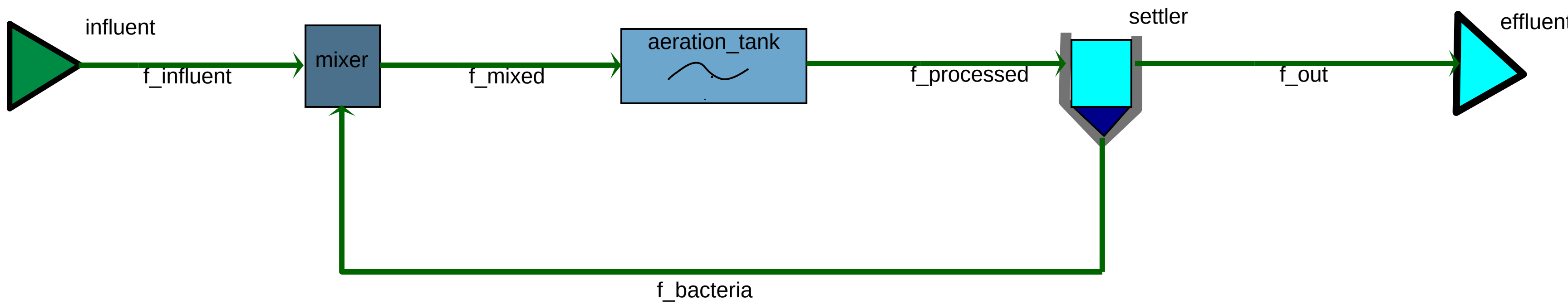
DS(V)M Environment



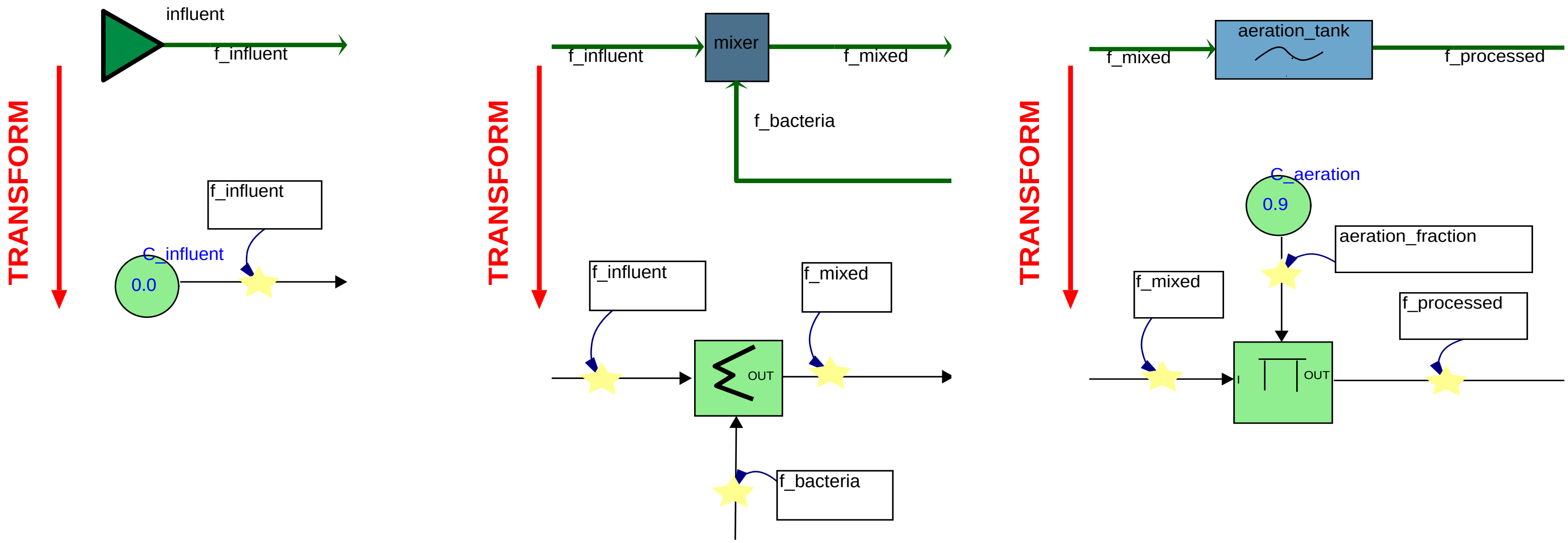
www.hemmis.com/products/west/

Henk Vanhooren, Jurgen Meirlaen, Youri Amerlinck, Filip Claeys, Hans Vangheluwe, and Peter A. Vanrolleghem. WEST: Modelling biological wastewater treatment. Journal of Hydroinformatics, 5(1):27-50, 2003.

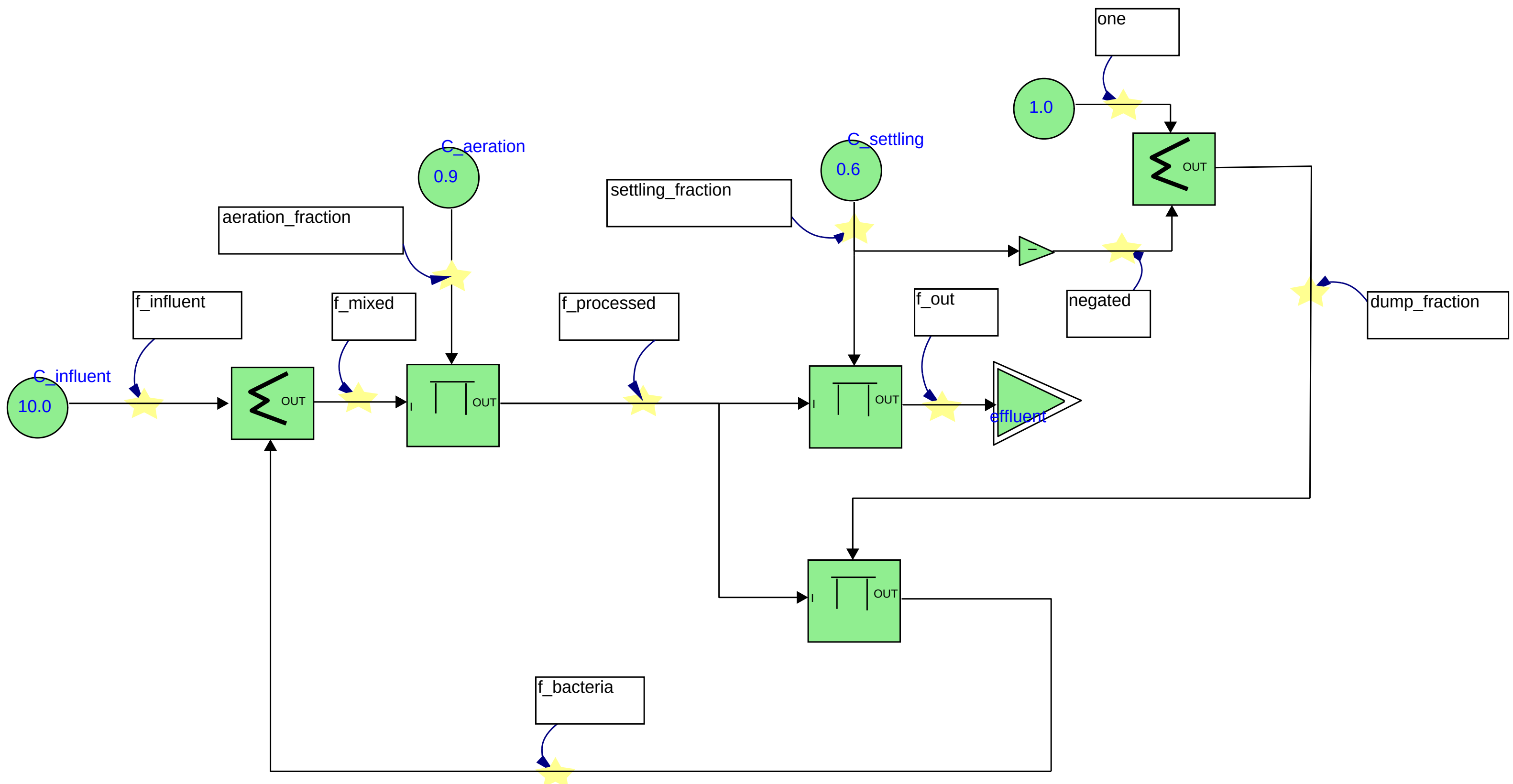
WWTP (domain-specific) model



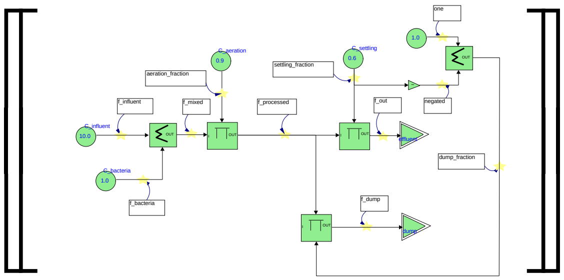
Transformation from WWTP to . . .



… its meaning (steady-state abstraction): Causal Block Diagram (CBD)



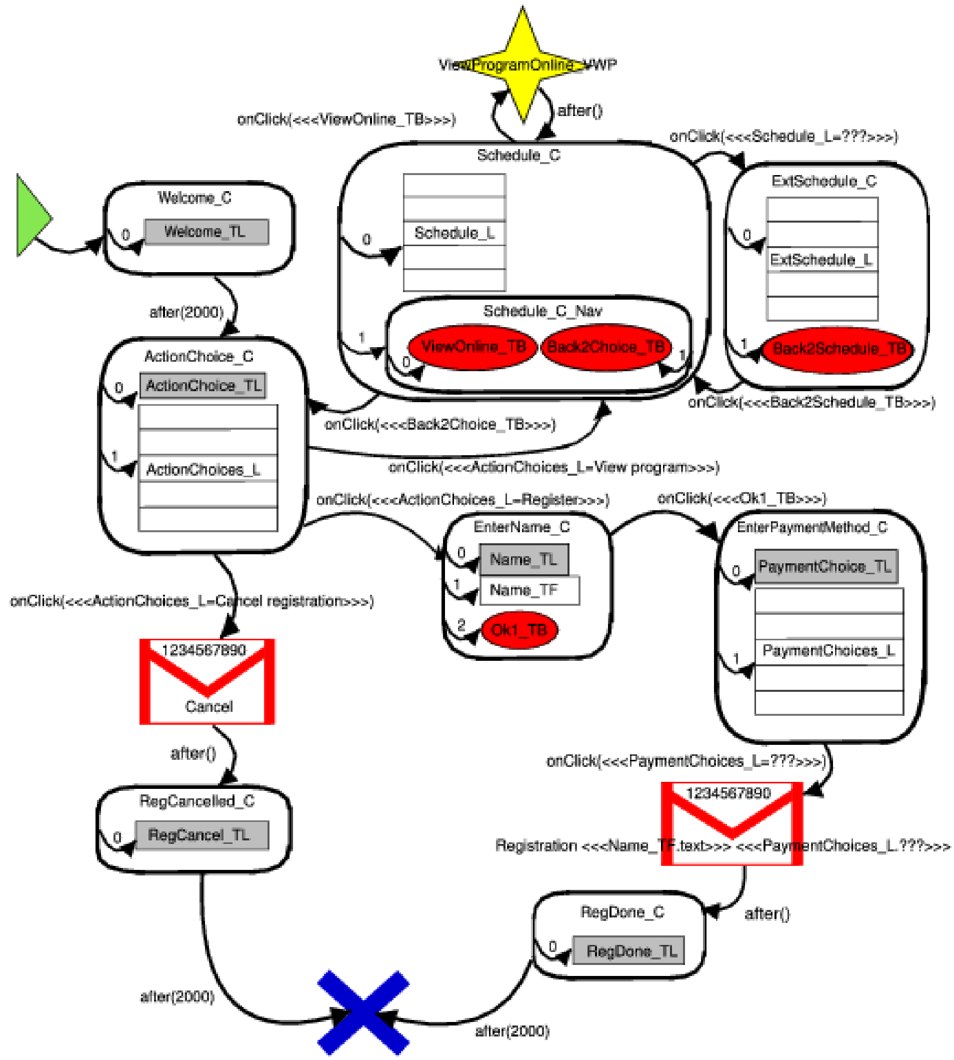
Meaning of the CBD



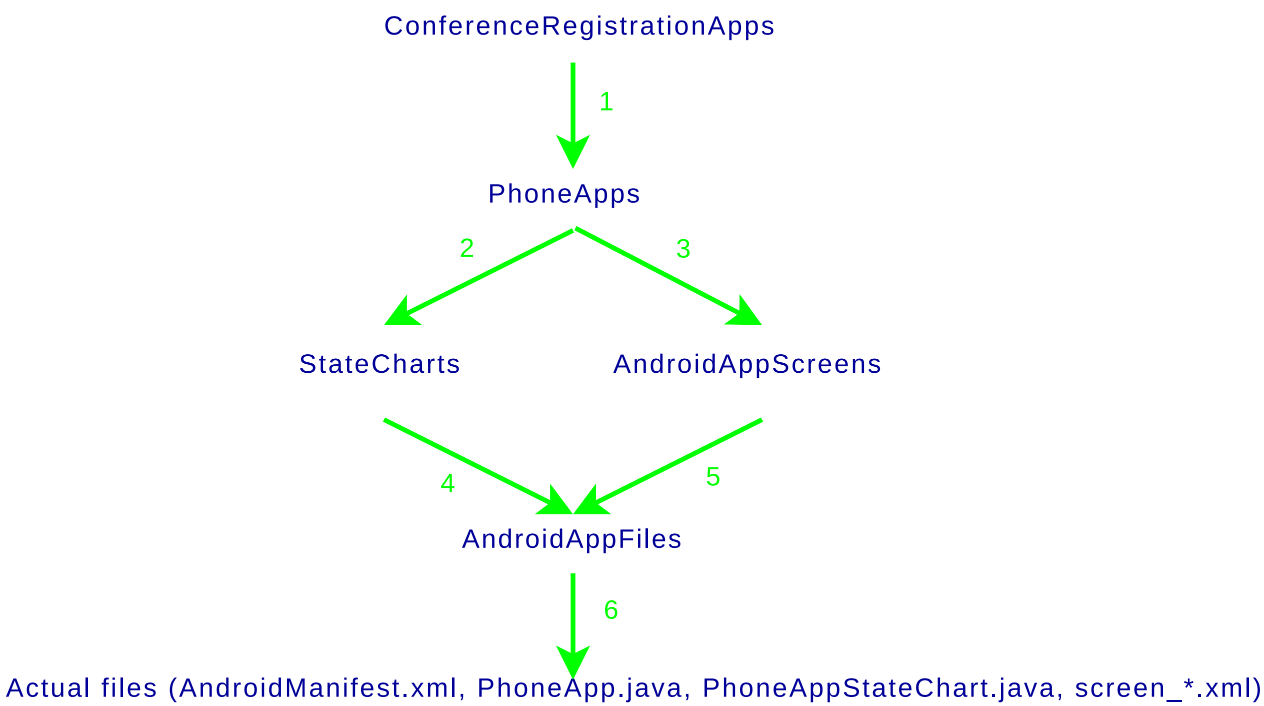
$$
= \left\{
\begin{array}{rcl}
f\_influent & = & C\_influent \\
f\_bacteria & = & C\_bacteria \\
f\_mixed & = & f\_influent + f\_bacteria \\
aeration\_fraction & = & C\_aeration \\
f\_processed & = & aeration\_fraction * f\_mixed \\
settling\_fraction & = & C\_settling \\
negated & = & -settling\_fraction \\
one & = & 1 \\
dump\_fraction & = & one + negated \\
f\_dump & = & f\_processed * dump\_fraction \\
f\_out & = & settling\_fraction * f\_processed
\end{array}
\right.
$$

DS(V)M example application: conference registration (Google Android)

DS(V)M example application, the PhoneApps Domain-Specific model



Only transform . . .



Why DS(V)M ?
(as opposed to General Purpose modelling)

- **match the user's mental model** of the problem domain
- **maximally constrain** the user (to the problem at hand)
  $\Rightarrow$ easier to learn
  $\Rightarrow$ avoid errors
- **separate** domain-expert's work
  from analysis/transformation expert's work

Anecdotal evidence of 5 to 10 times speedup

Steven Kelly and Juha-Pekka Tolvanen.

Domain-Specific Modeling: Enabling Full Code Generation. Wiley 2008.

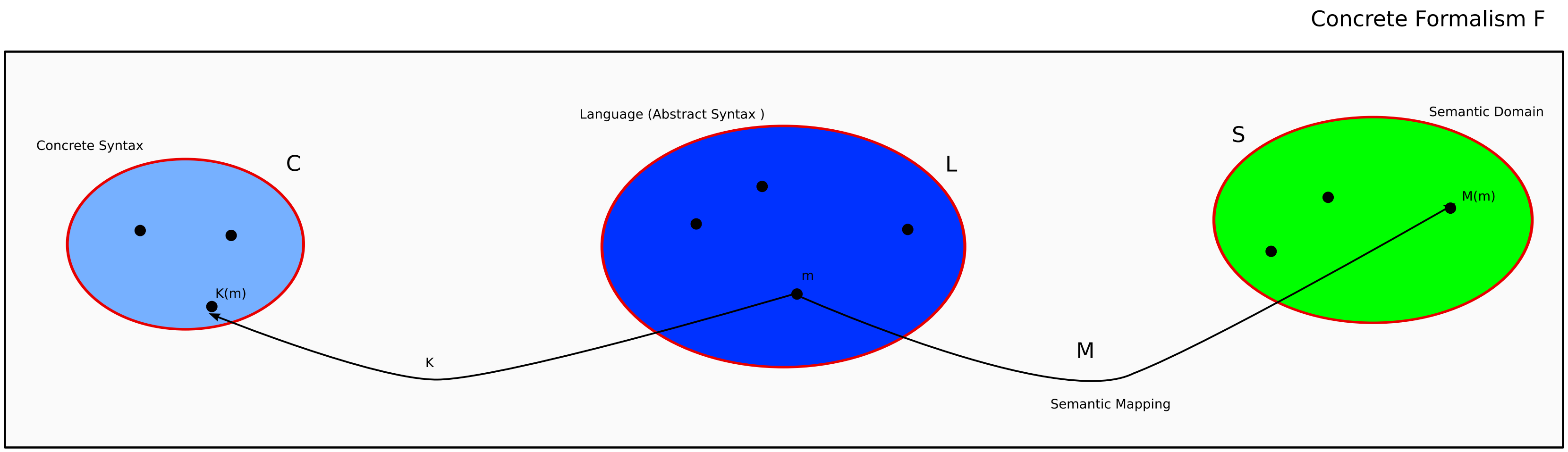
Building (DS)(V)M Tools Effectively . . .

- **development cost** of DS(V)M Tools may be prohibitive !
- we want to effectively (rapidly, correctly, re-usably, . . .)
  1. Specify DS(V)L **syntax**:
     - **abstract** $\Rightarrow$ **meta-modelling**
     - **concrete** (textual/visual)
  2. Specify DS(V)L **semantics**:
     transformation
  3. Model (and analyze and execute) model **transformations**:
     $\Rightarrow$ **graph rewriting**

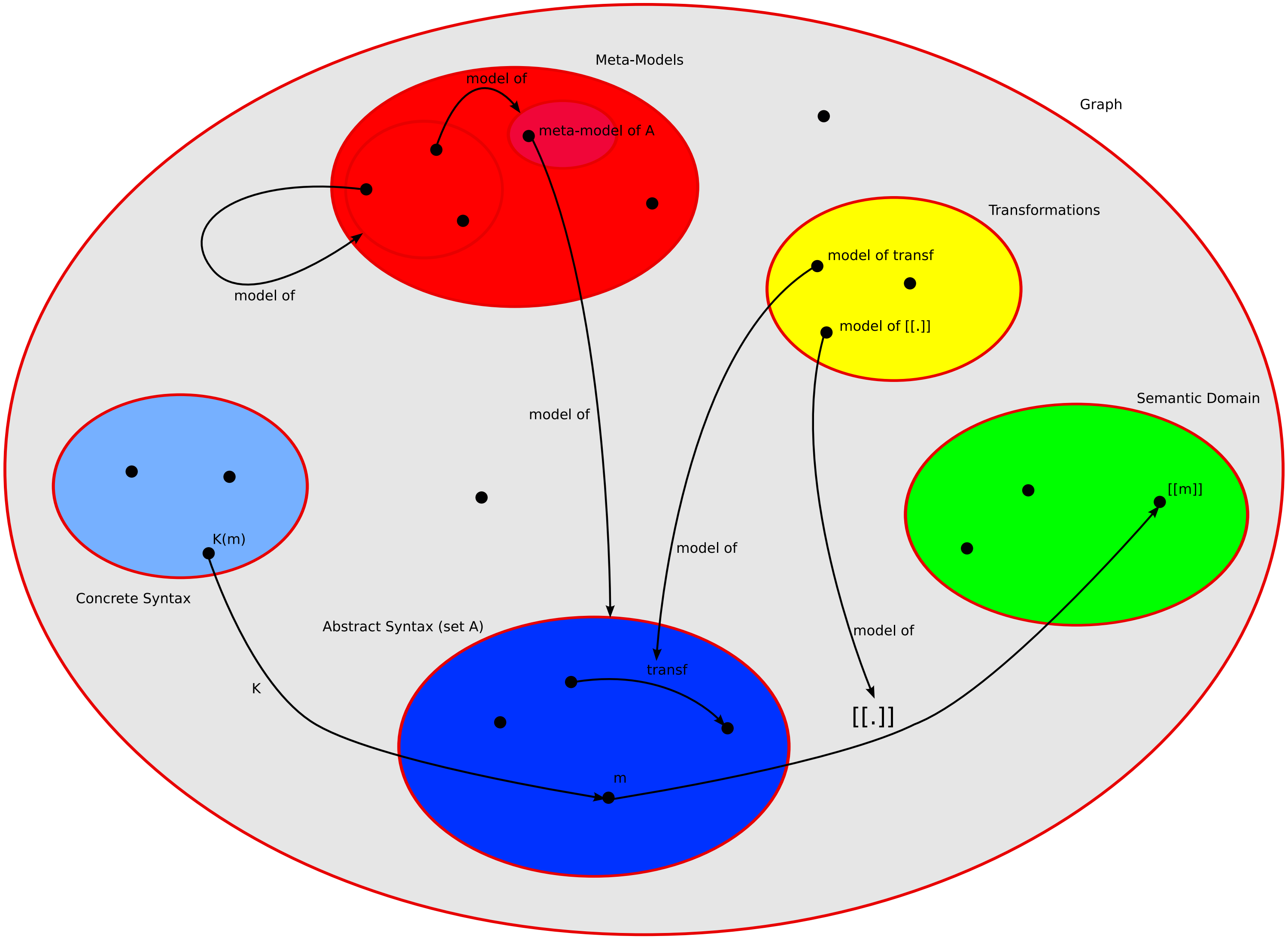
$\Rightarrow$ **model everything**

(in the most appropriate formalism,
at the appropriate level of abstraction)

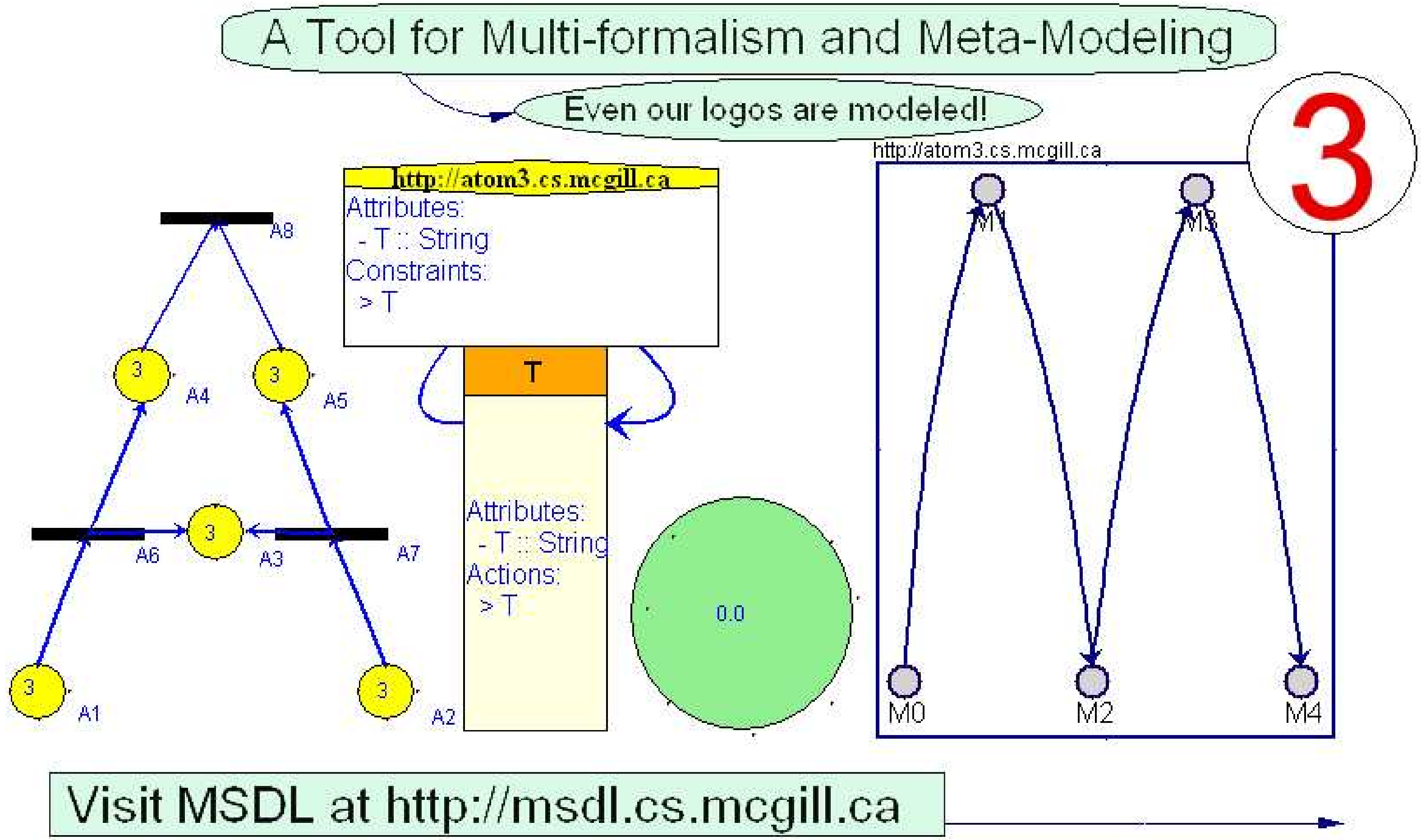
Dissecting a Formalism



Modelling Modelling Languages



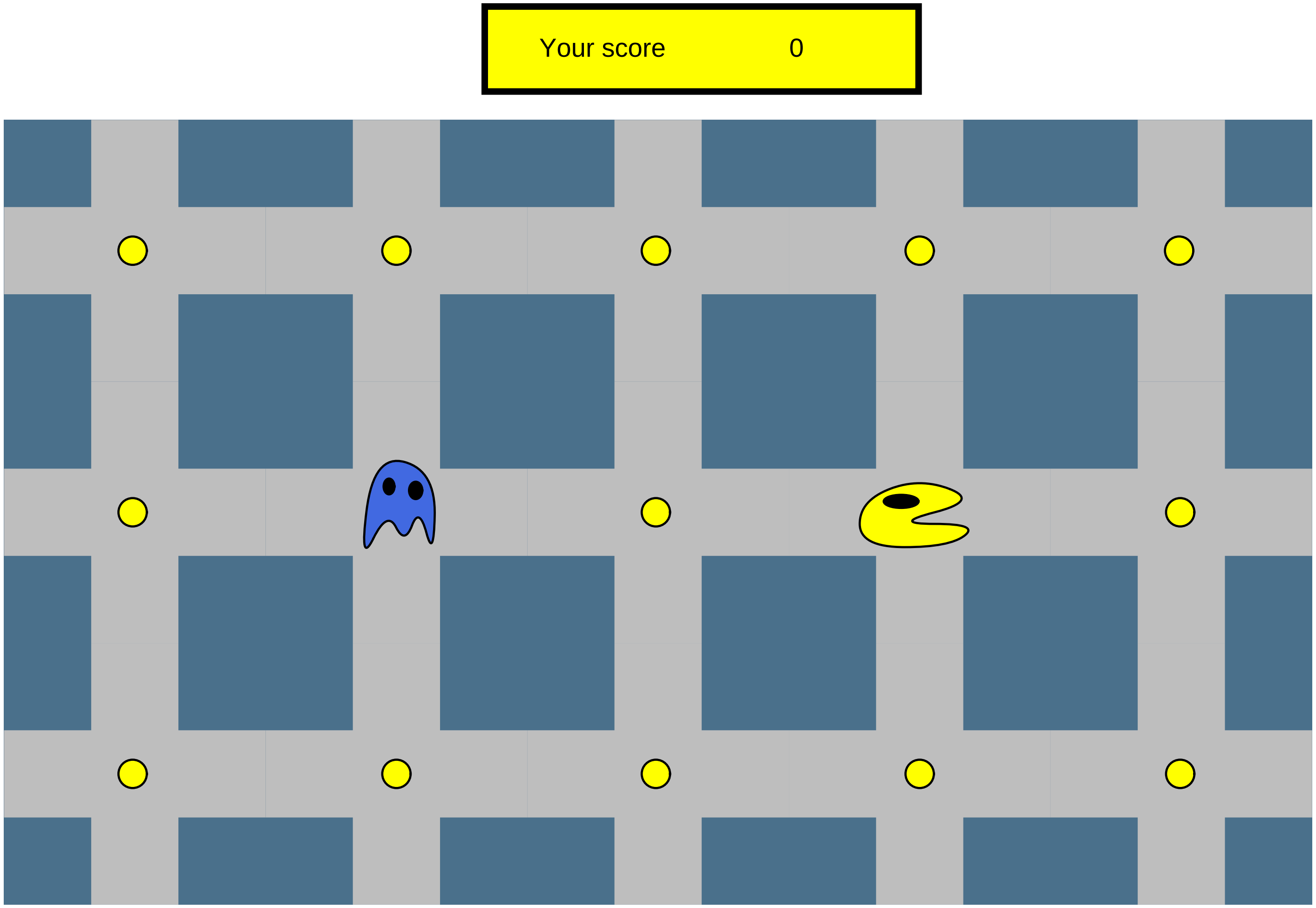
From now on: use AToM^3



Juan de Lara and Hans Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling.

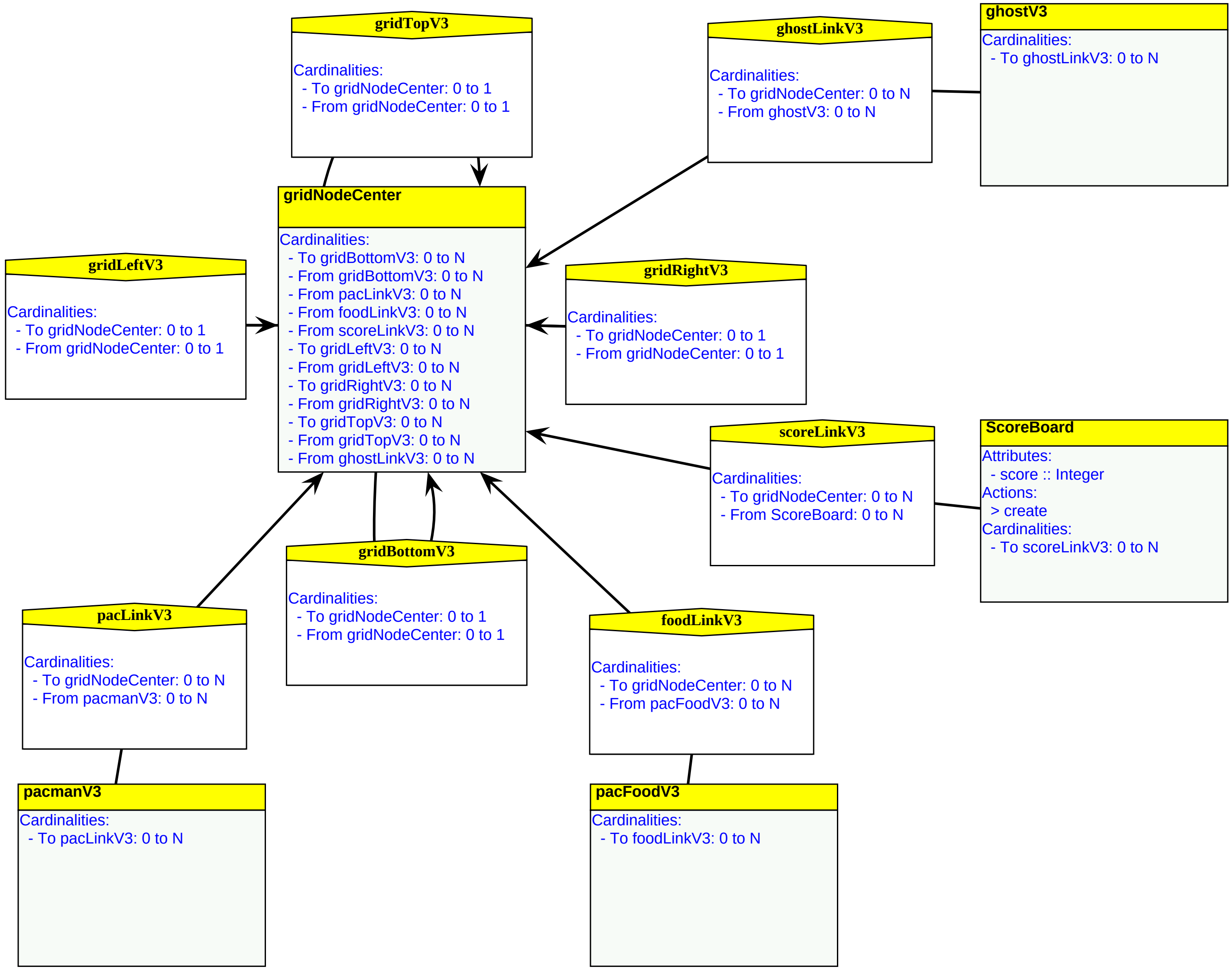
Fundamental Approaches to Software Engineering (FASE). LNCS 2306, pages 174 – 188, 2002.

A model in the PacMan Formalism

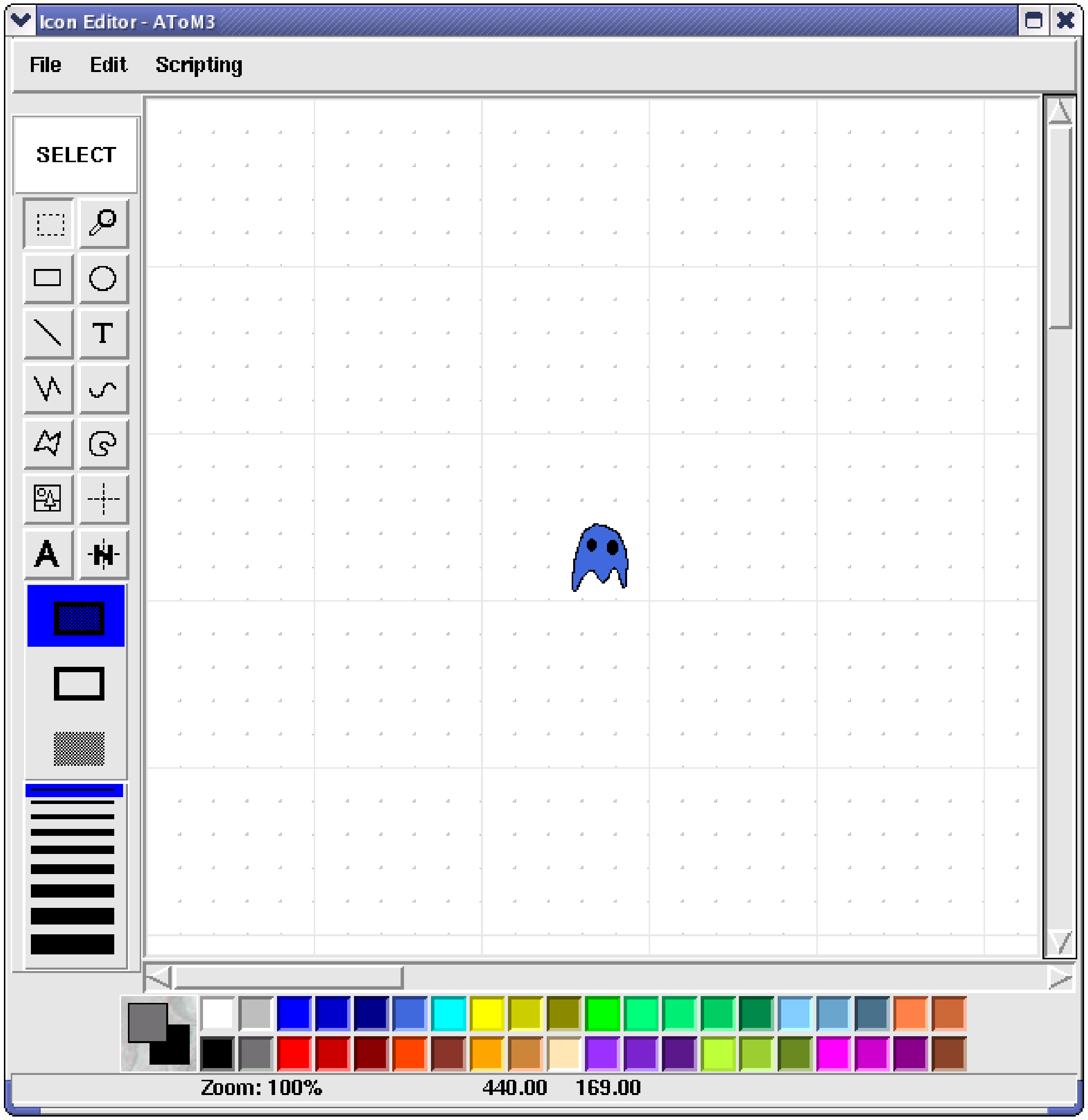


(thanks to Reiko Heckel)

Modelling Abstract Syntax (meta-model)



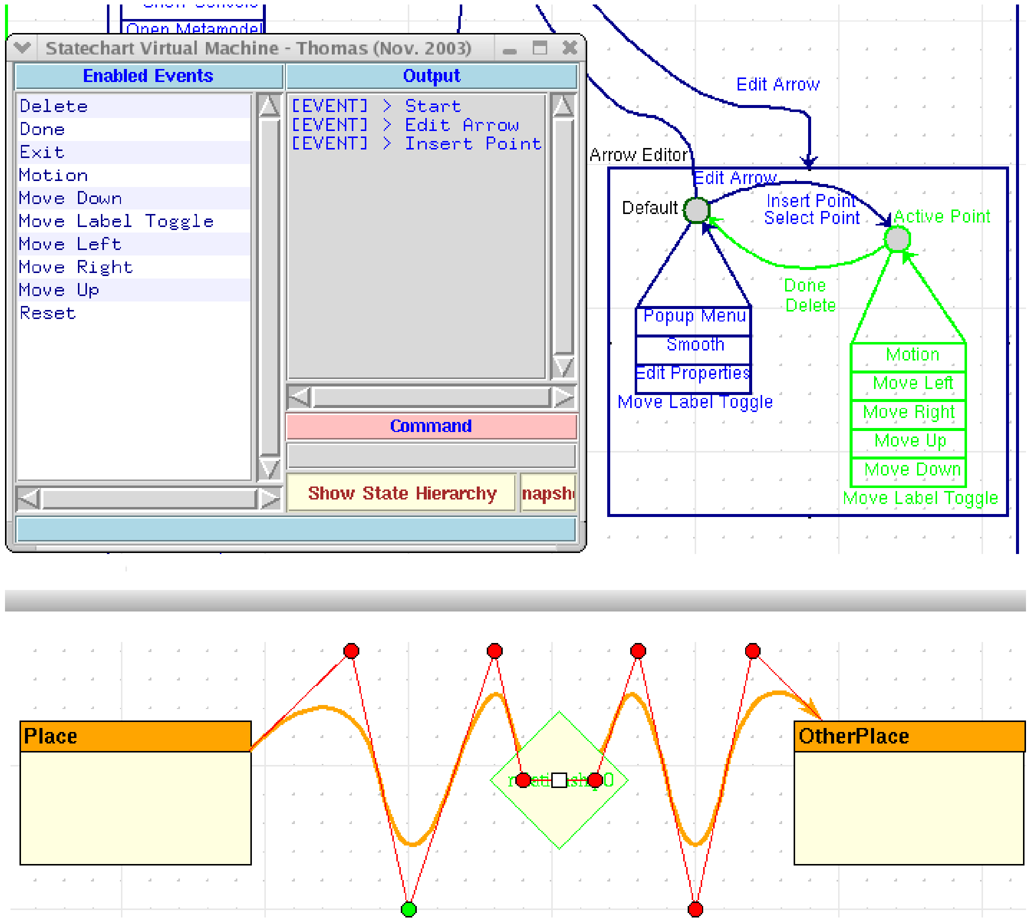
Modelling Ghost Concrete Visual Syntax



GhostLink Concrete Visual Syntax

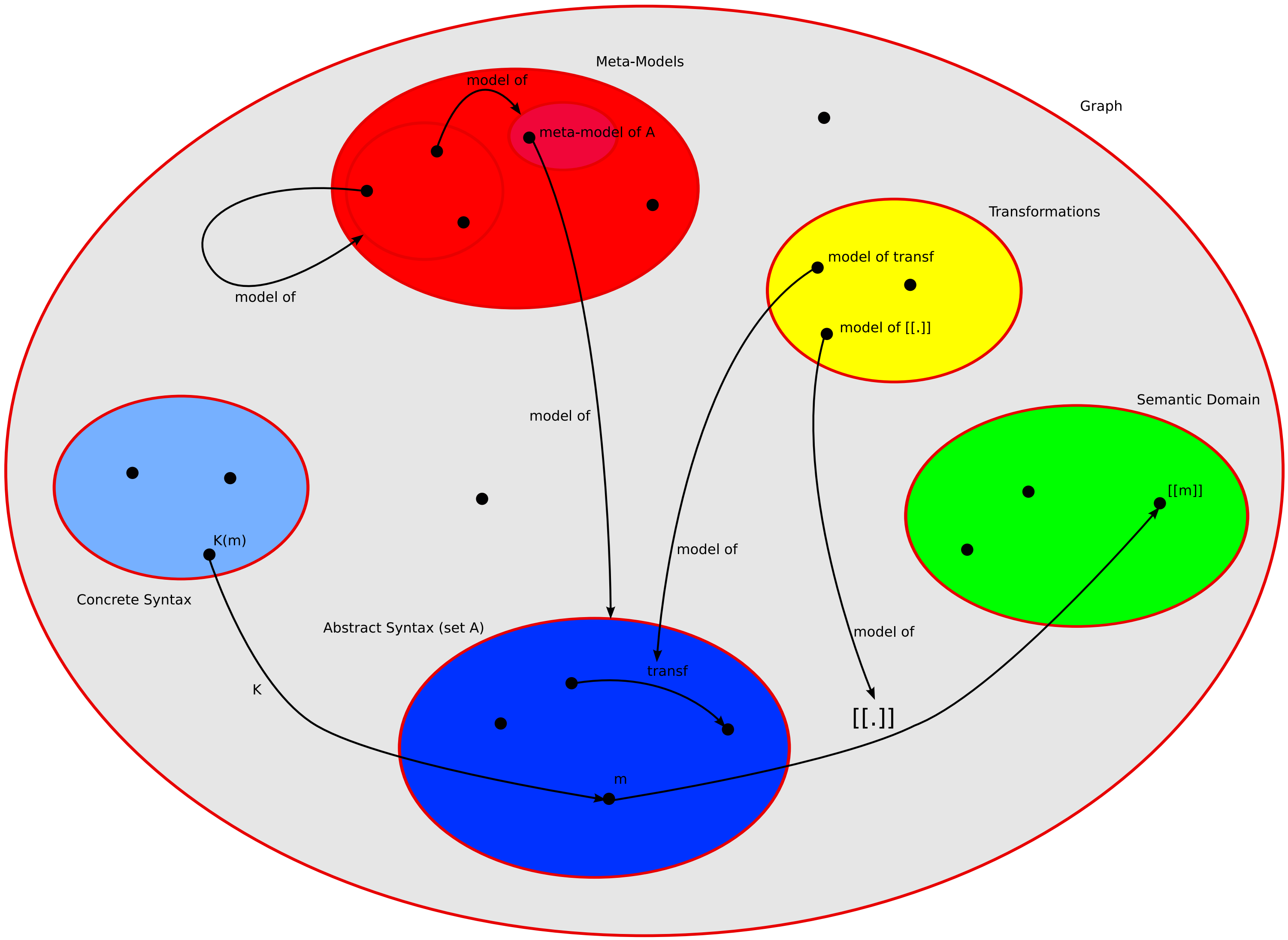
```
# Get n1, n2 end-points of the link
n1 = self.in_connections_[0]
n2 = self.out_connections_[0]

# g1 and g2 are the graphEntity visual objects
g0 = self.graphObject_   # the link
g1 = n1.graphObject_     # first end-point
g2 = n2.graphObject_     # second end-poing

# Get the high level constraint helper and solver
from Qoca.atom3constraints.OffsetConstraints import OffsetConstraints
oc = OffsetConstraints(self.parent.qocaSolver)

# The constraints
oc.CenterX((g1, g2, g0))
oc.CenterY((g1, g2, g0))
oc.resolve()
```

Model the GUI's Reactive Behaviour ! in the Statechart formalism (++)



challenge: find *optimal* formalism to specify GUI reactive behaviour

Modelling Modelling Languages



Specifying Model Transformations

What is the "optimal" formalism ?
Models are often graph-like $\Rightarrow$ natural to express model transformation by means of **graph transformation** models.

Tools:

GME/GReAT, PROGRES, Fujaba, AGG, AToM^3, GROOVE, ...
First three used in large industrial applications.

Ehrig, H., G. Engels, H.-J. Kreowski, and G. Rozenberg.

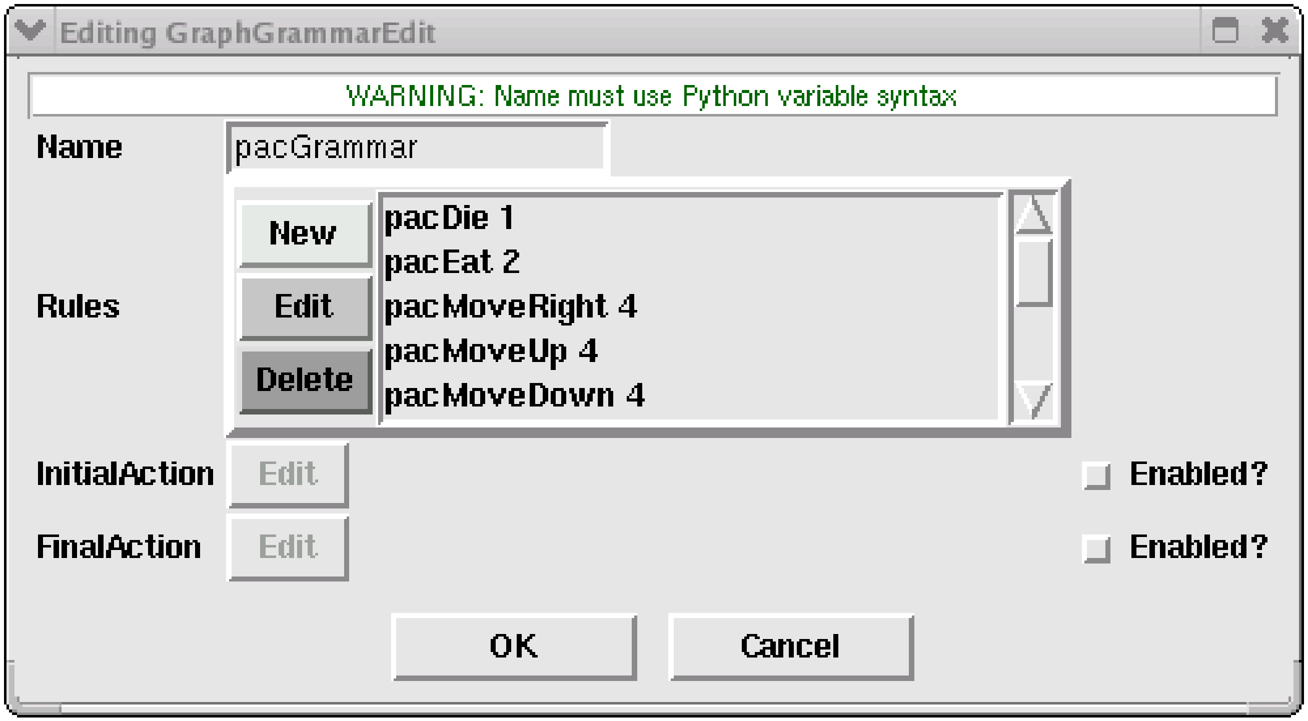
Handbook of graph grammars and computing by graph transformation.

1999. World Scientific.

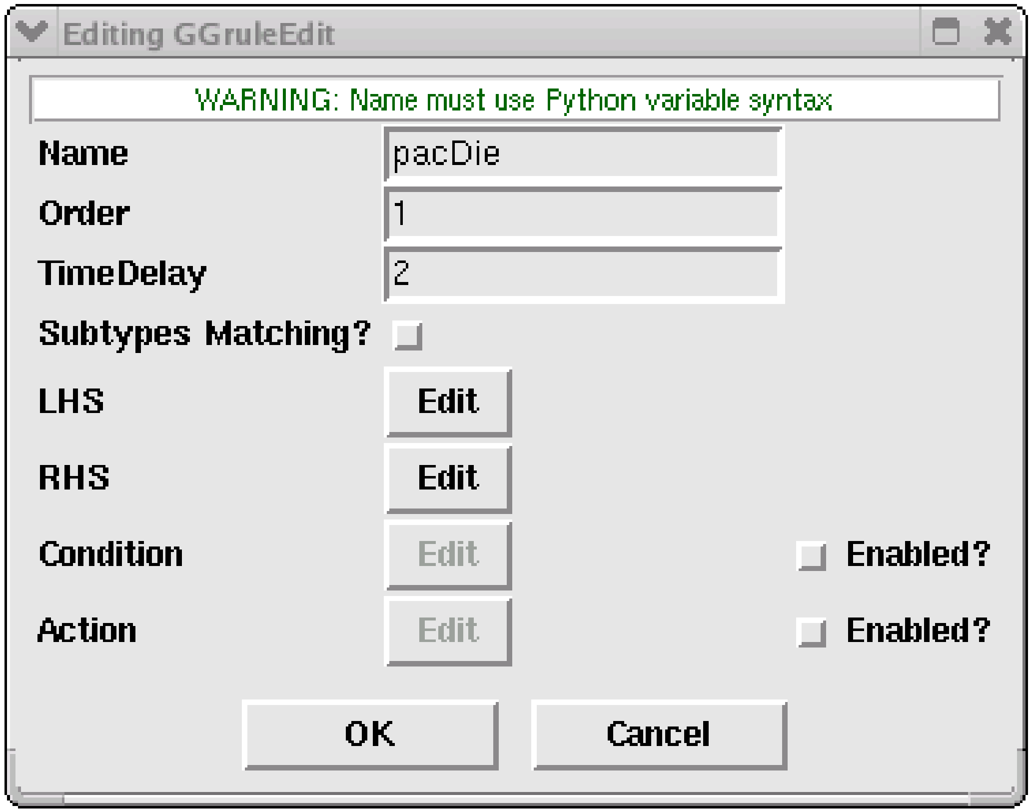
Modellling PacMan Operational Semantics using Graph Grammar models

note: for Denotational Semantics: map for example onto Petri Net

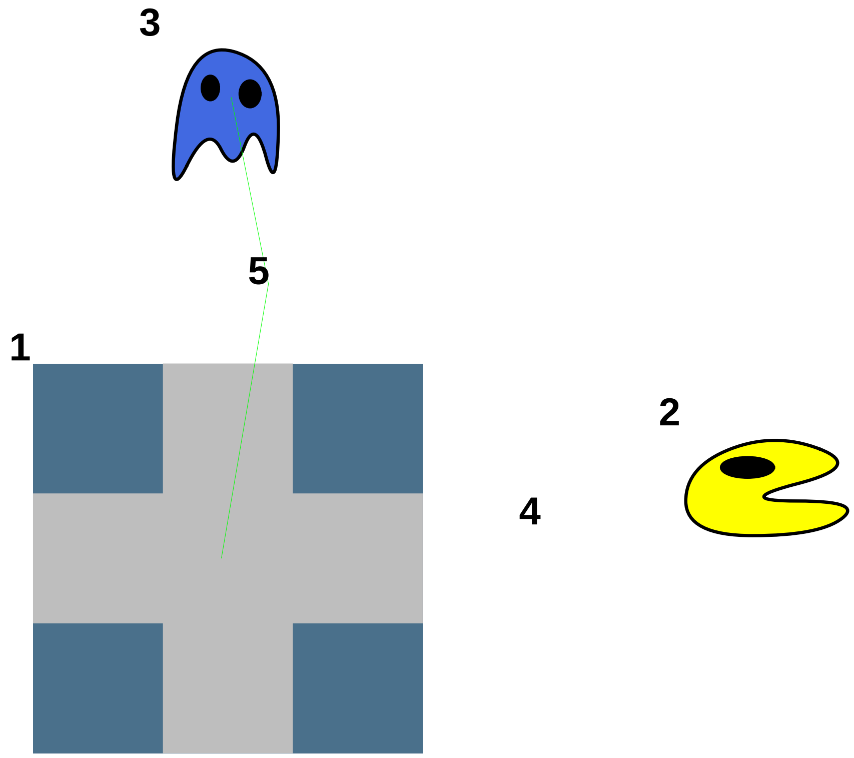
Model Operational Semantics using GT



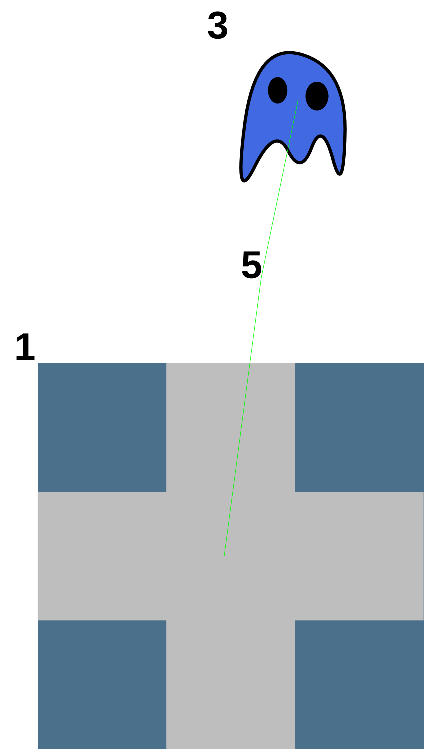
PacMan Die rule



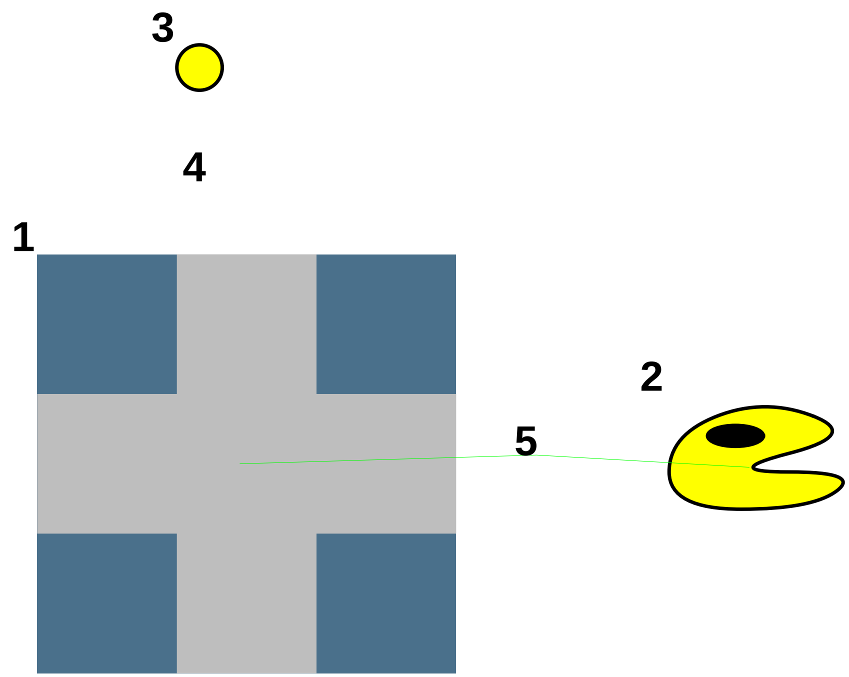
PacMan Die rule LHS



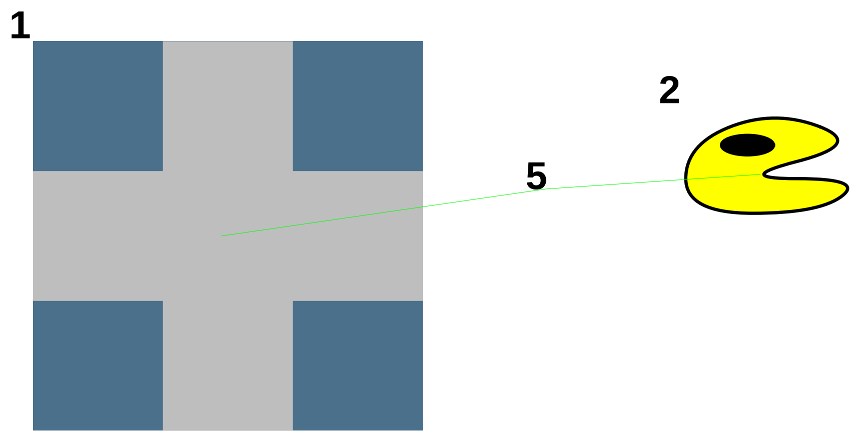
PacMan Die rule RHS



PacMan Eat rule LHS

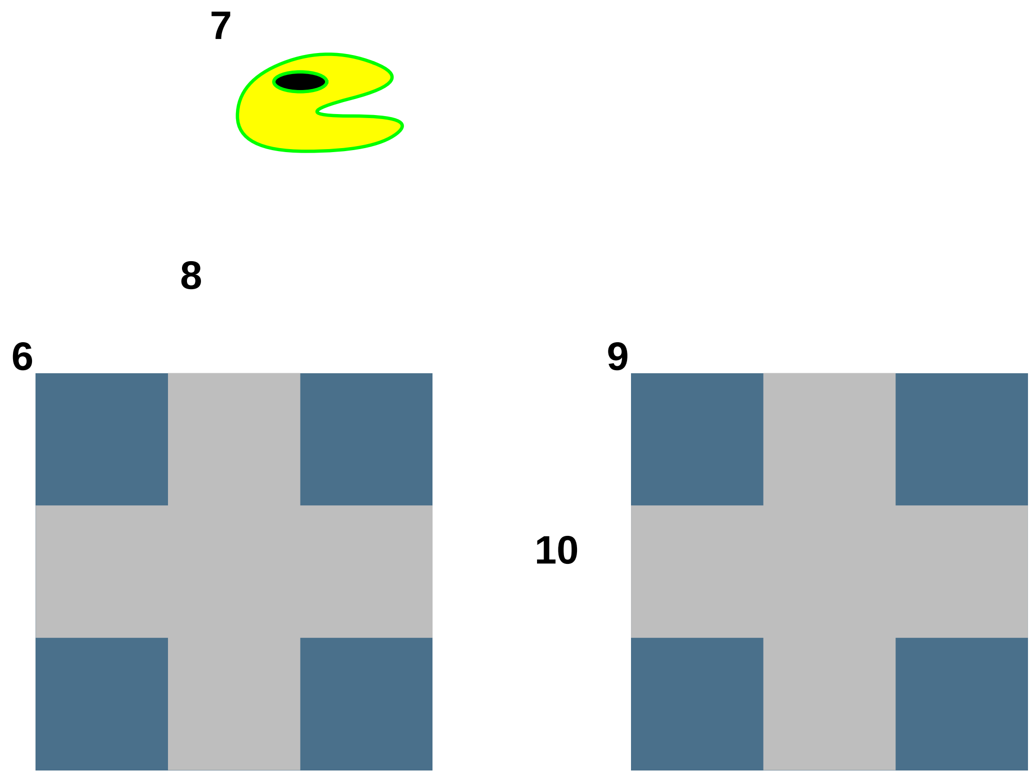


PacMan Eat rule RHS

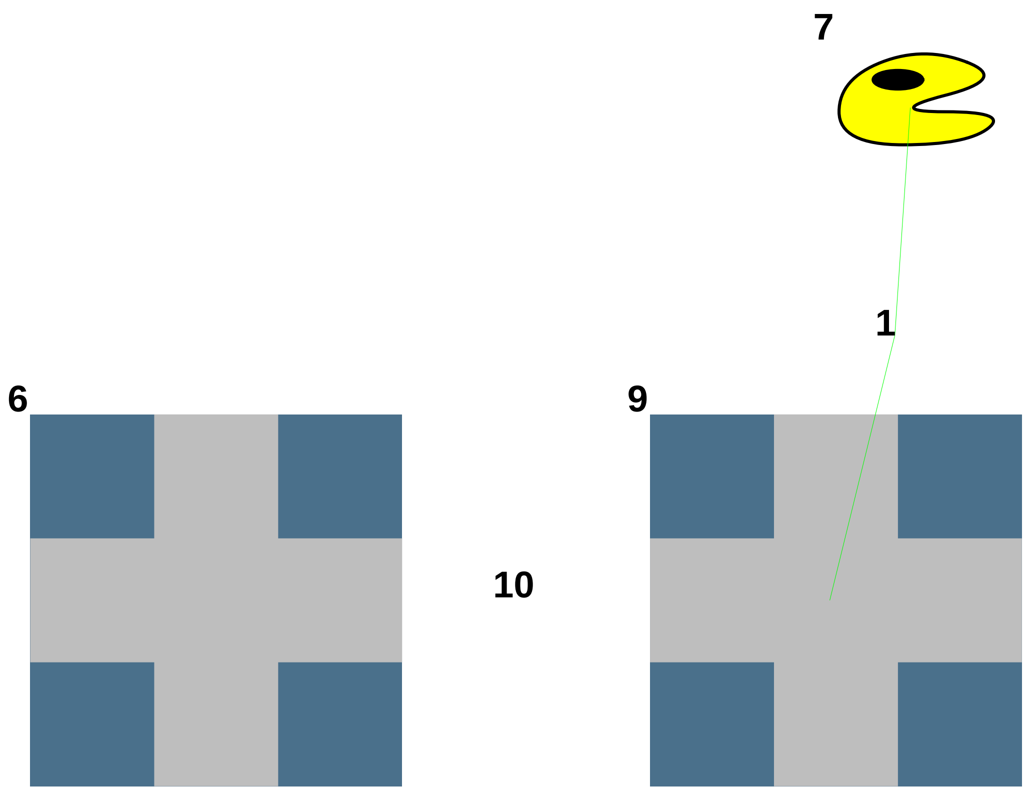



```
scoreBoard = None
scoreBoards = atom3i.ASGroot.listNodes['ScoreBoard']
if (not scoreBoards):
  return
else:
  scoreBoard = scoreBoards[0]
  scoreVal = scoreBoard.score.getValue()
  scoreBoard.score.setValue(scoreVal+1)
  scoreBoard.graphObject_.ModifyAttribute('score',scoreVal+1)
```

PacMan Move rule LHS



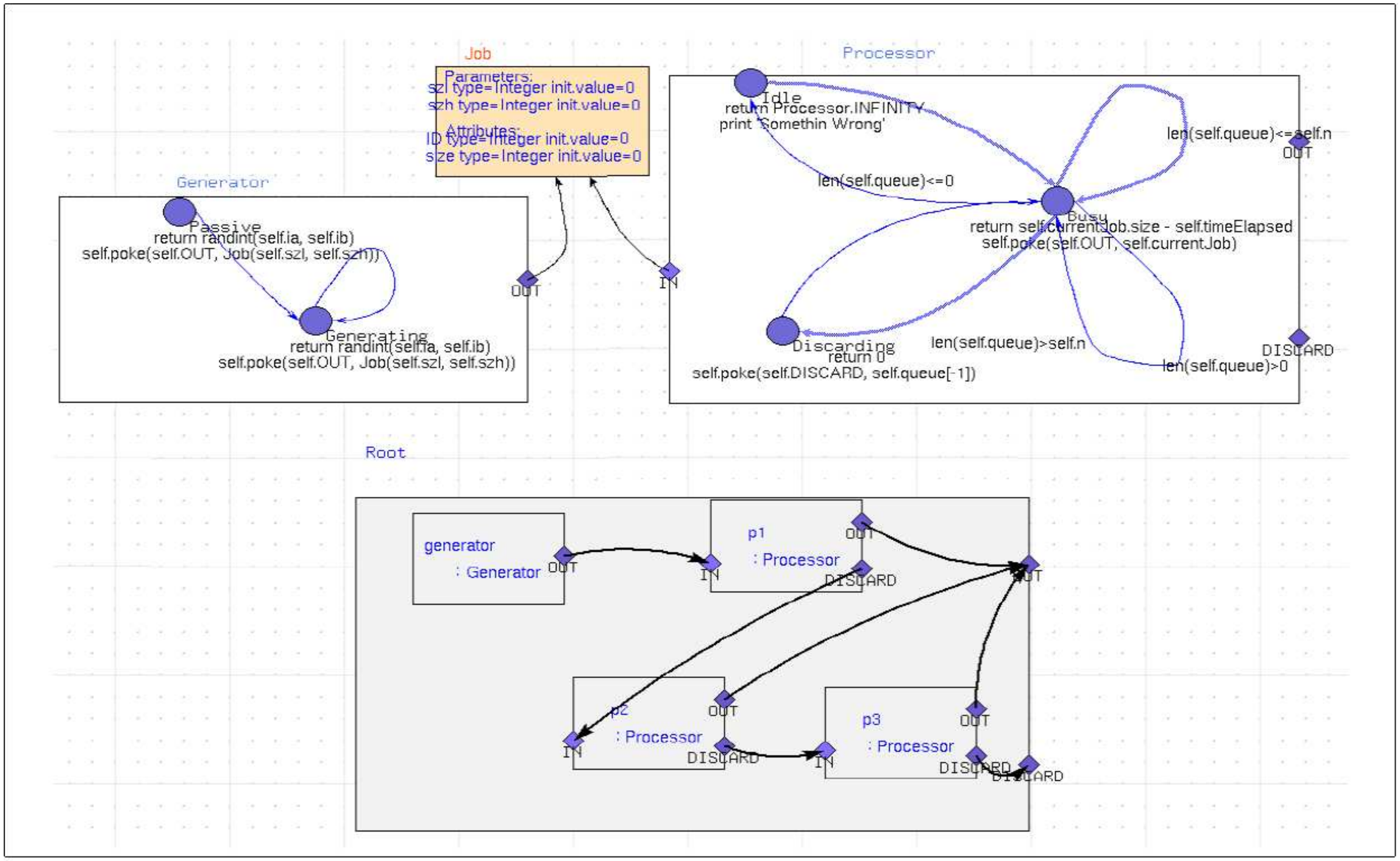
PacMan Move rule RHS



MPM for EOOLT

- Domain-Specific Languages (*e.g.*, WWTP):
  - model abstract syntax, including domain constraints
  - model concrete syntax
  - model mapping onto EOOL (note: need trace-ability)
- Rule-based specification of EOOLT model transformations
- Graph patterns for variable structure formalisms

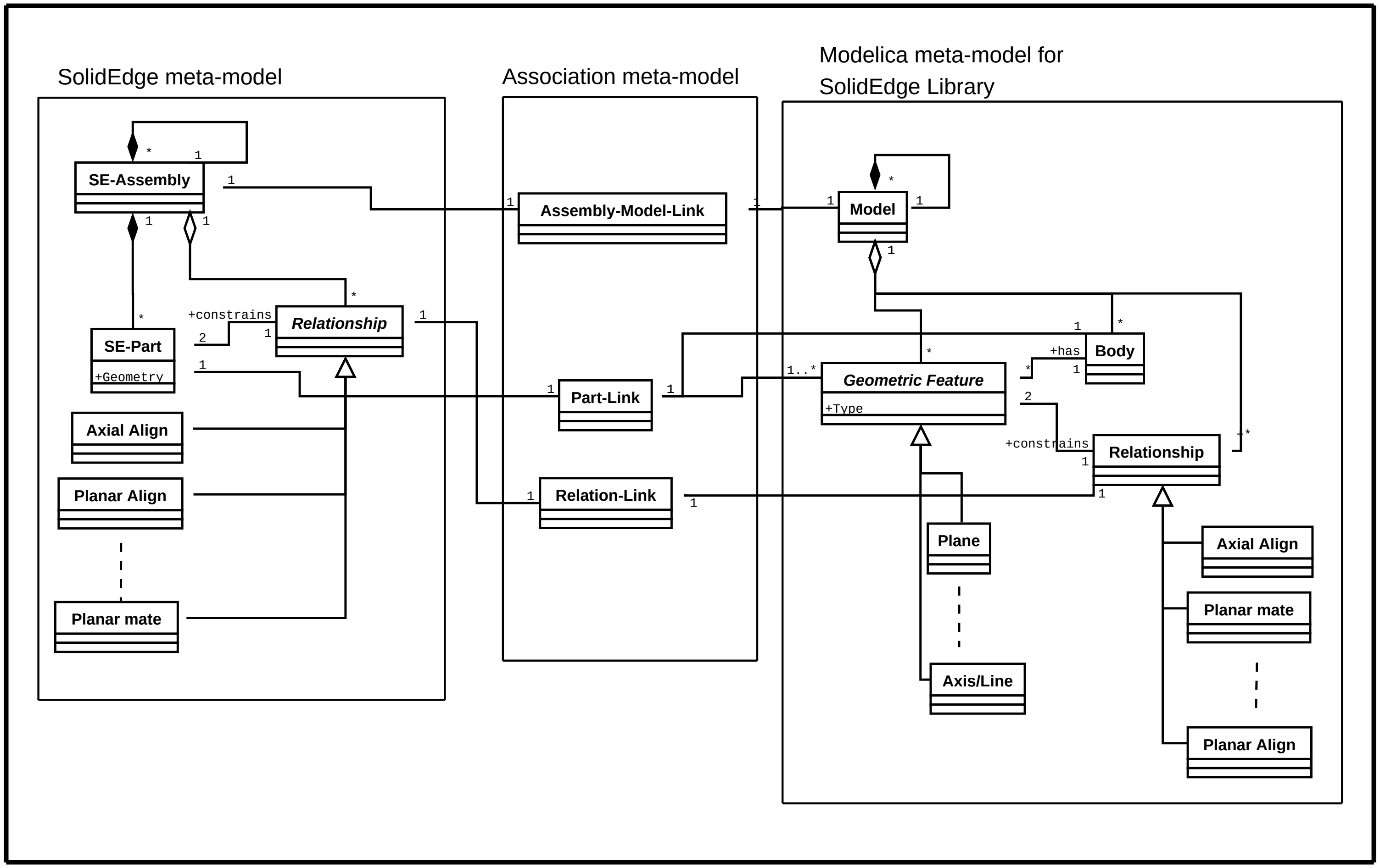
Visual Modelling Environment for DEVS



EOOLT for MPM

- add declarative constraint equations to (meta-)models
- model consistency, co-evolution (with TGGs)

Meta-triple/Triple Graph Grammar(TGG)



Andy Schürr. Specification of Graph Translators with Triple Graph Grammars.

LNCS 903. pages 151–163, 1994.

Conclusions

- Multi-Paradigm Modelling (MPM)
- Domain-Specific Modelling
- Language Engineering and MPM Tools
- MPM for EOOLT
- EOOLT for MPM

model **everything** !

Model Based Development: some Open Problems

1. deal with legacy models (code)
2. consistency (TGGs + modularity), multi-user modelling
3. multi-view modelling, (semantic) consistency
4. version control, (meta-) model evolution
5. trace-ability (backward links)
6. multi-formalism modelling
7. model refinement
8. automated design-space exploration
9. automated testing (of models and model transformations)
10. transformation models are first-class models $\Rightarrow$ higher-order transformation
11. deal with concrete syntax (arbitrary mix of textual, visual) in a unified manner
12. concrete visual syntax: web-based (SVG+Ajax)