DSM-TP 2010 | Domain Specific Modeling Theory and Practice

(Domain-Specific) Modelling Language Engineering

Hans Vangheluwe

5 September 2010, Lisboa, Portugal

Overview

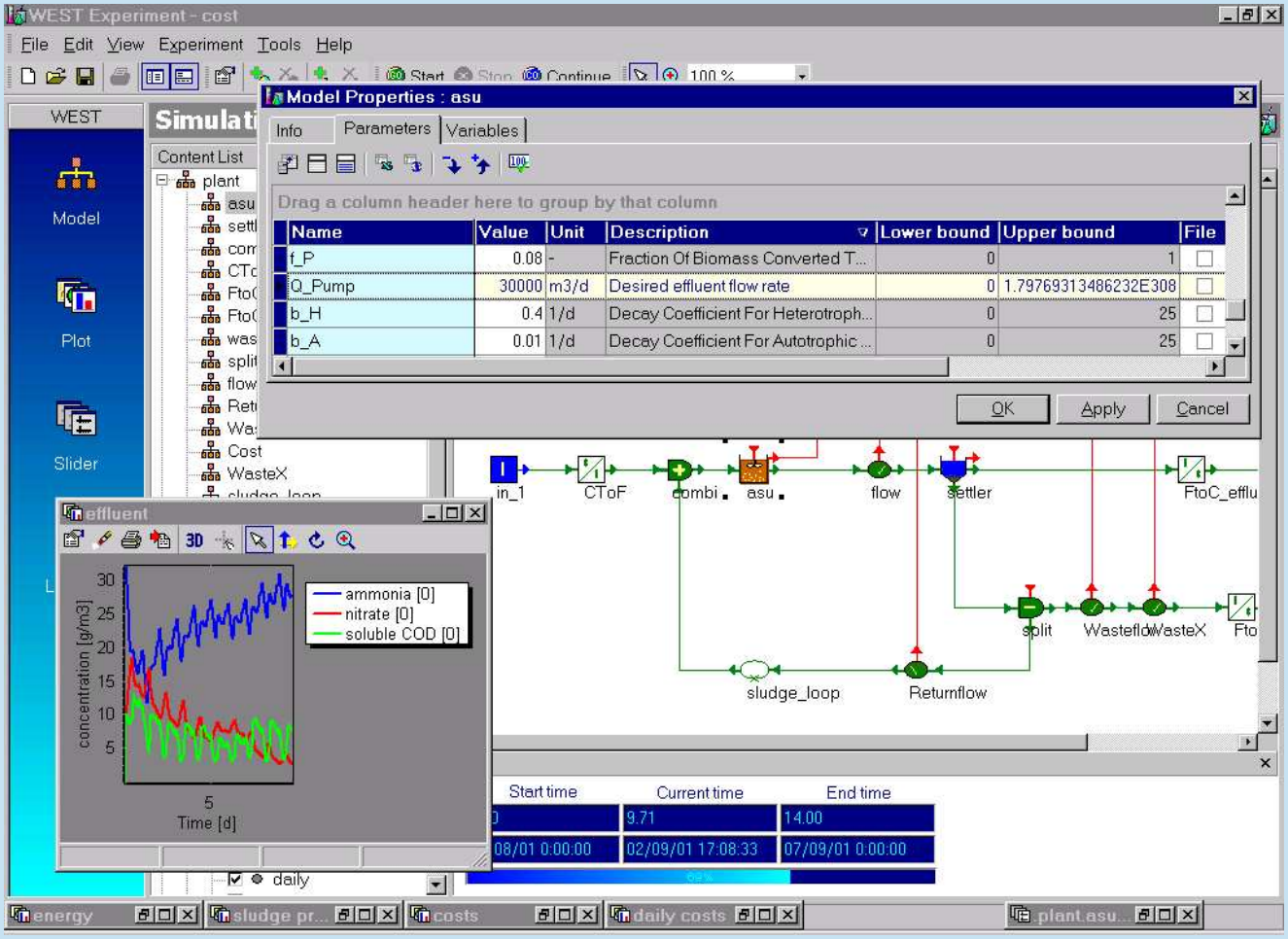
1. Domain-Specific (Visual) Modelling – DS(V)M
   - What/Why of DS(V)M (and DS(V)Ls) ?
2. Dissecting Modelling
3. Dissecting Modelling Languages
4. Building DS(V)M Tools Effectively
   1. Specifying **syntax** of DS(V)Ls:
      - **abstract** (**meta-modelling**)
      - **concrete** (textual–visual)
   2. Specifying DS(V)L **semantics**: **transformations**
   3. Modelling (and executing) **transformations**: **(rule-based) transformation languages**

Domain-Specific Modelling Example

NATO's Sarajevo WWTP

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

DS(V)M Environment



www.hemmis.com/products/west/

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

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

- **match the user's mental model** of the problem domain

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

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

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

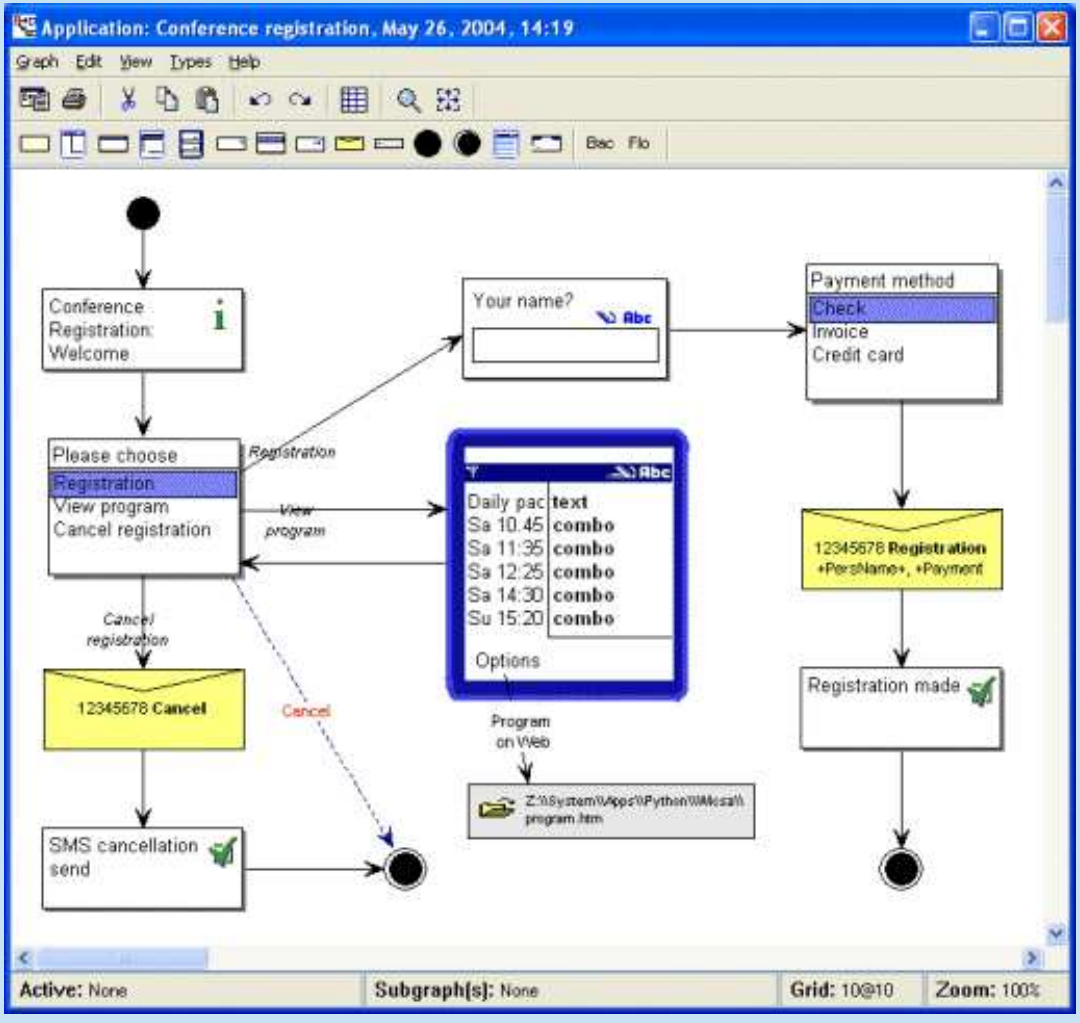
Steven Kelly and Juha-Pekka Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley, 2008.

Laurent Safa. The practice of deploying DSM, report from a Japanese appliance maker trenches. In Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06), pp. 185-196, 2006.

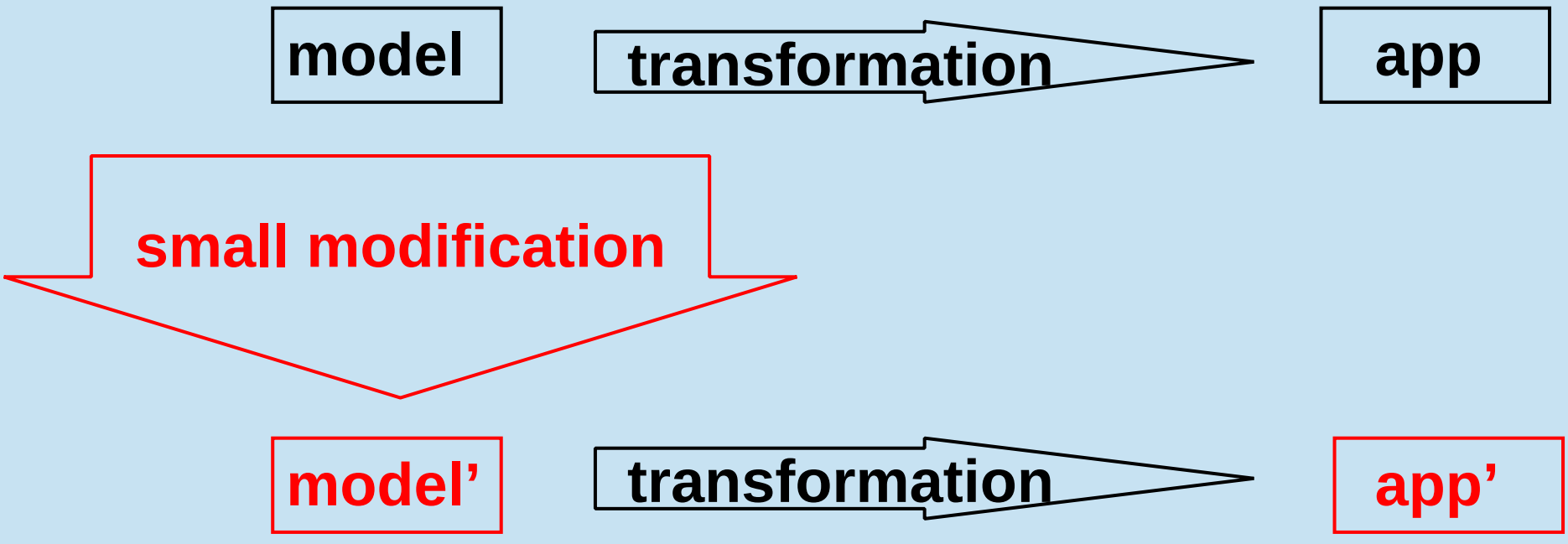
DS(V)M Example in Software Domain
smart phones, the application

MetaEdit+ (`www.metacase.com`)

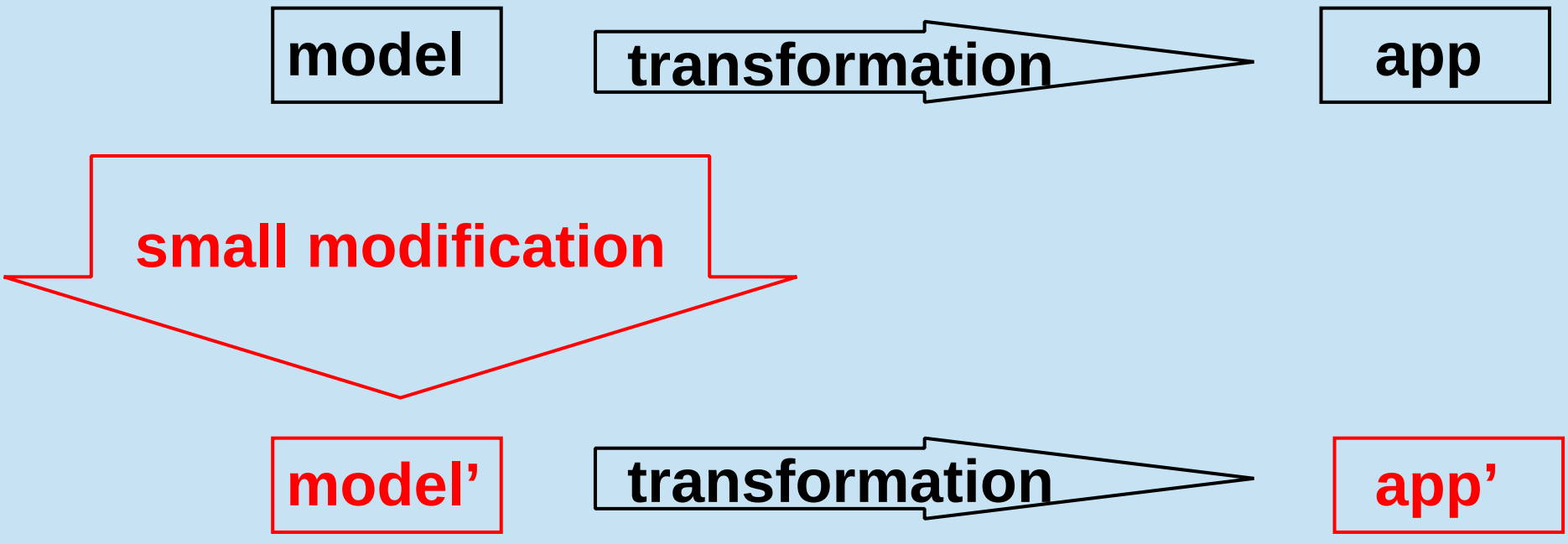
DS(V)M Example: smart phones, the Domain-Specific model



Model-Based Development: Modify the Model (e.g., based on feature model of product family)

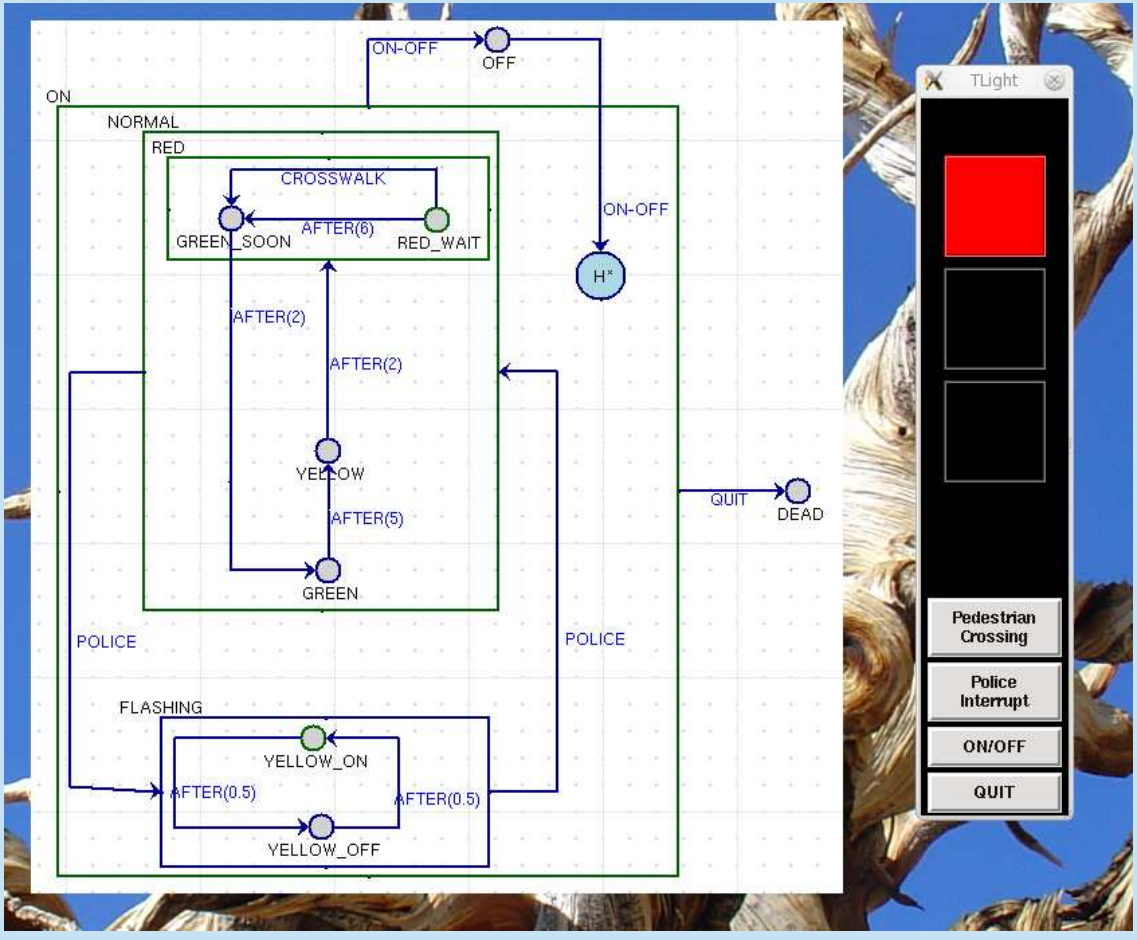


Model-Based Development: Modify the Model (e.g., based on feature model of product family)

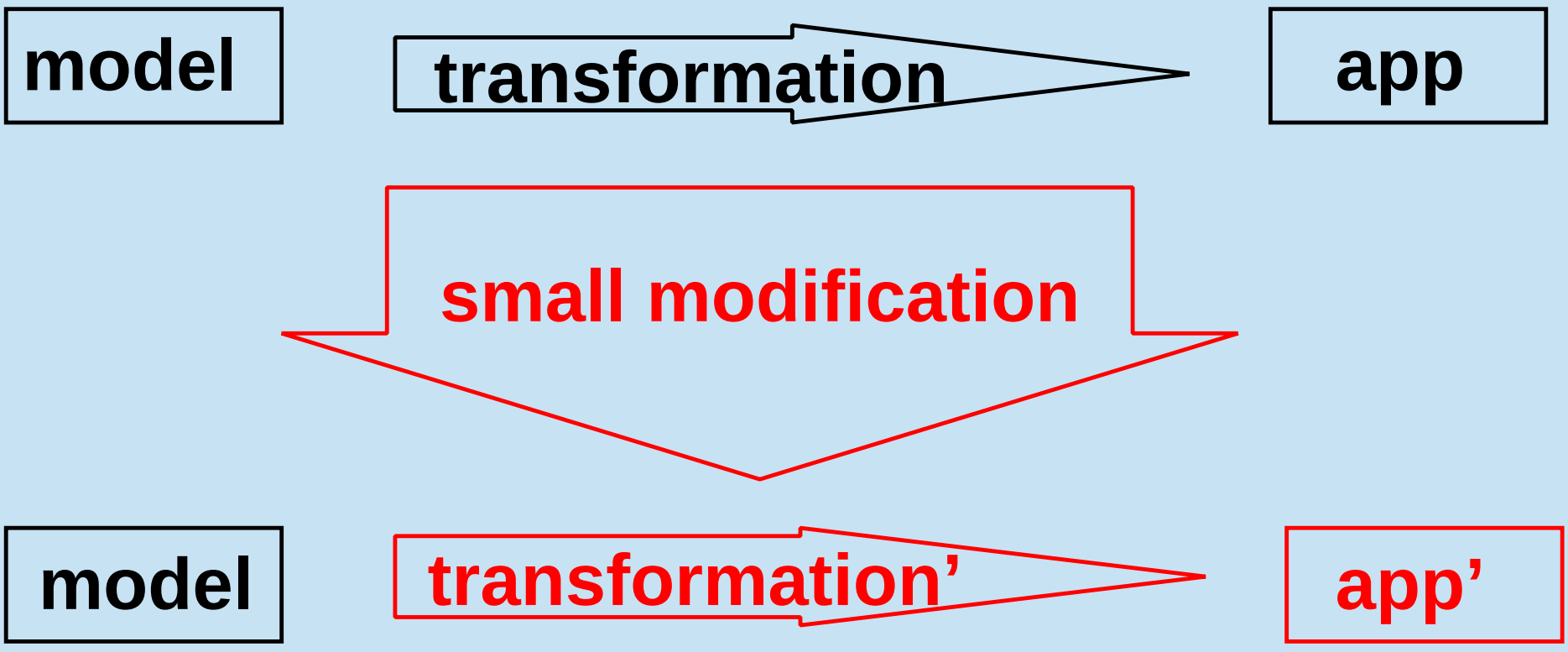


small **modification** in model may lead to large change in **app**
$\sim$ choice of formalism (e.g., Statecharts)

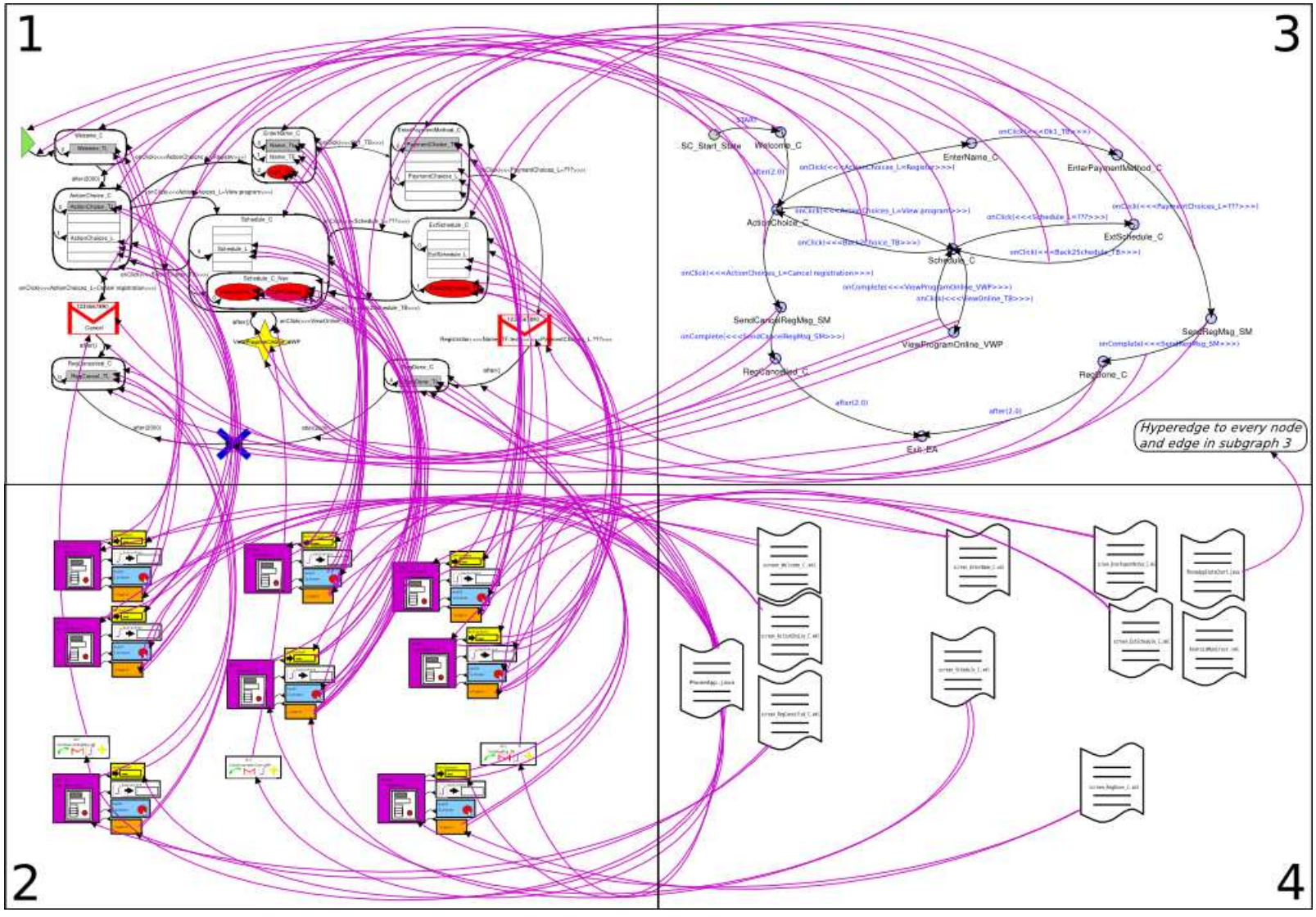
Statecharts



Model-Based Development: Modify the Transformation (e.g., target platform changes, or optimization)



Can be Multi-Step/Multi-Formalism



Building DS(V)M Tools Effectively . . .

- **development cost** of DS(V)M Tools may be prohibitive!
- $\Rightarrow$ need **Modelling Language Engineering**

Dissecting Modelling

Matters of (Meta-) Modeling

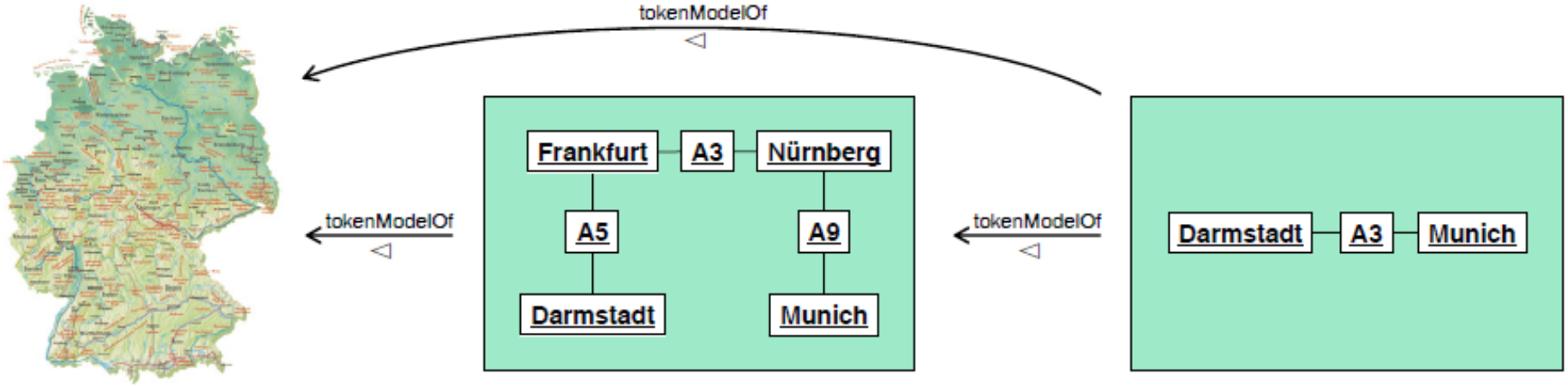
Thomas Kühne

Darmstadt University of Technology, Darmstadt, Germany
e-mail: kuehne@informatik.tu-darmstadt.de

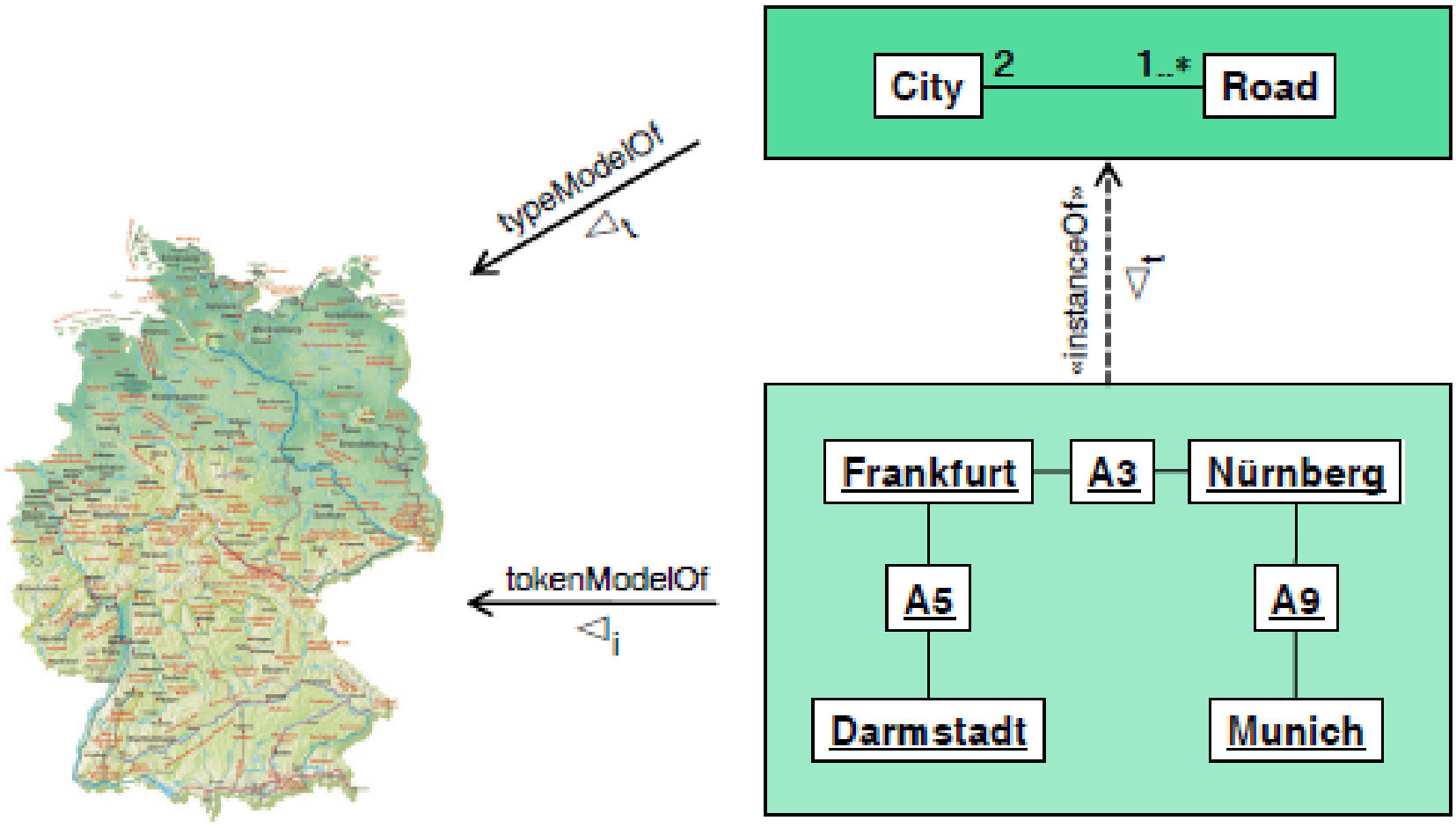
Model Features

mapping feature	A model is based on an original.[4]
reduction feature	A model only reflects a (relevant) selection of an original's properties.
pragmatic feature	A model needs to be usable in place of an original with respect to some purpose.

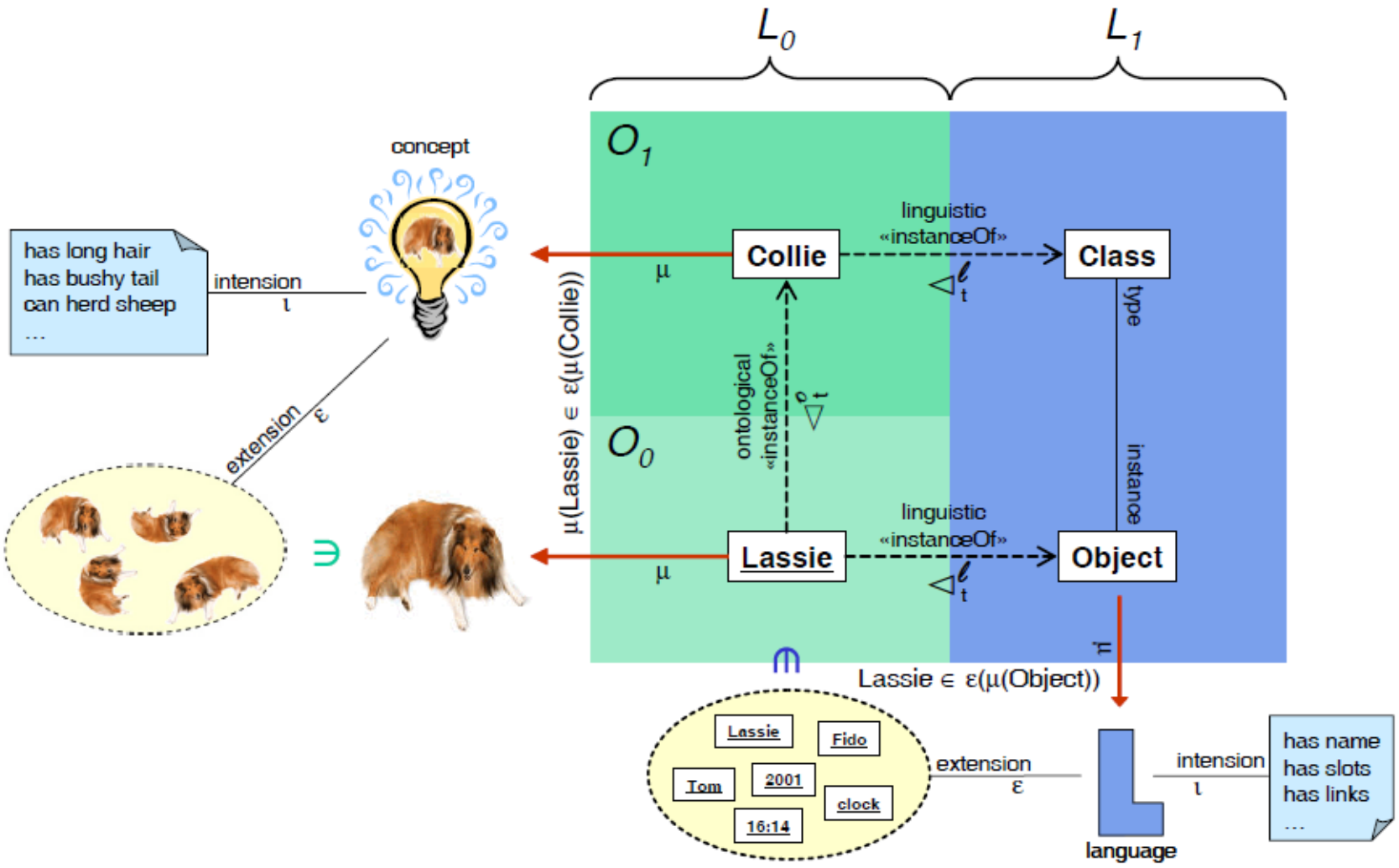
Token Models



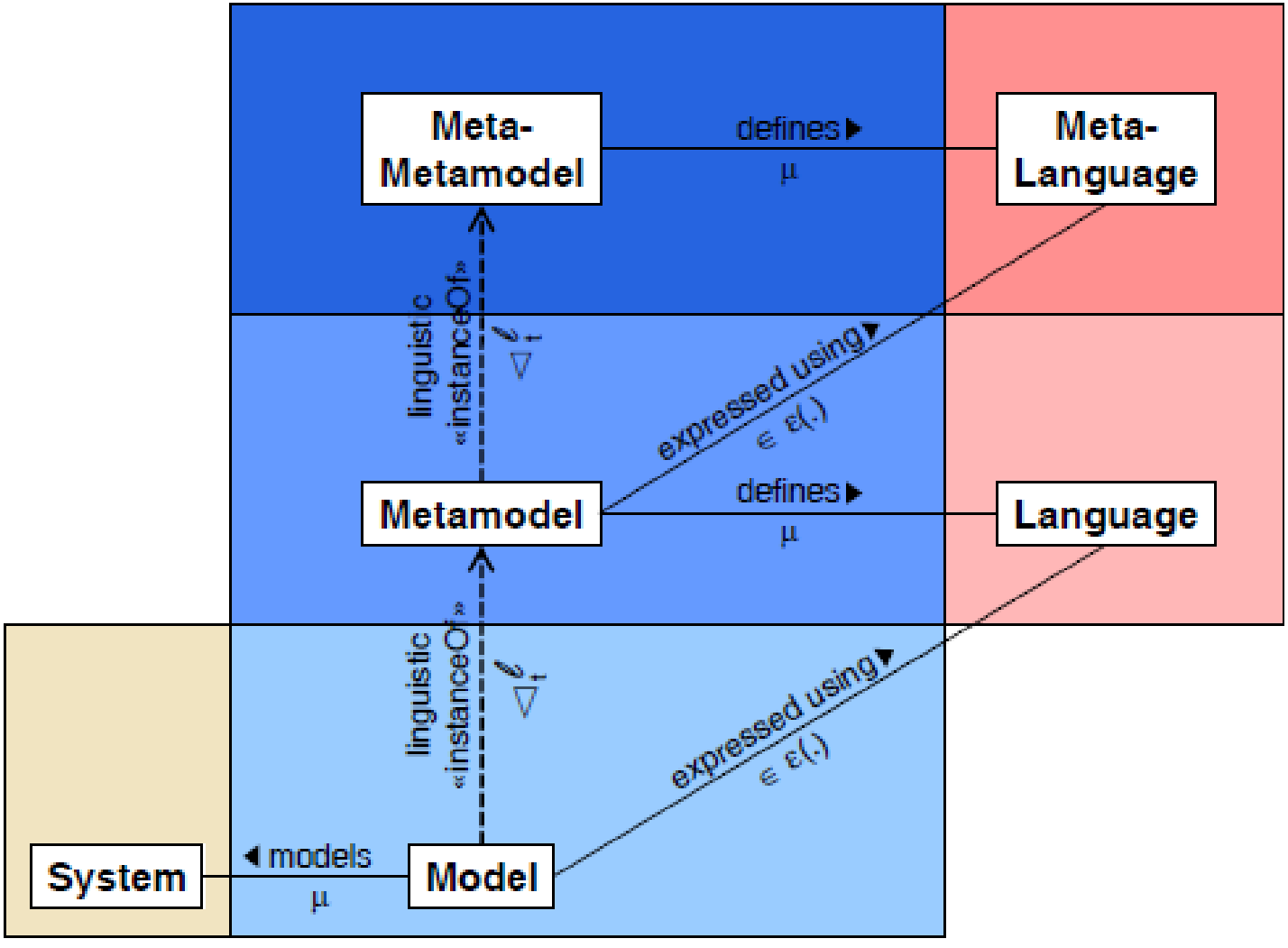
Rôles a Model may Play



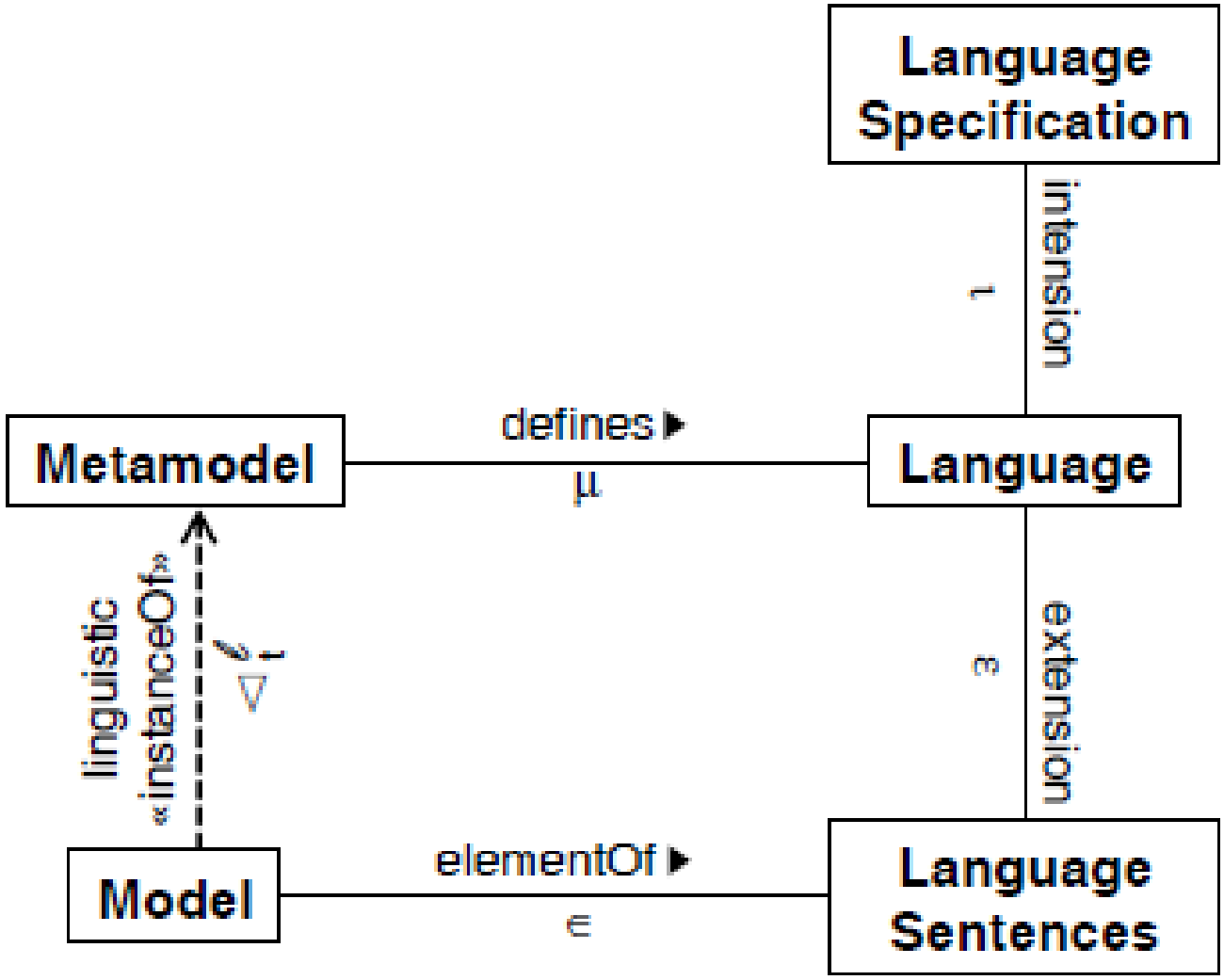
Ontological vs. Linguistic Instantiation



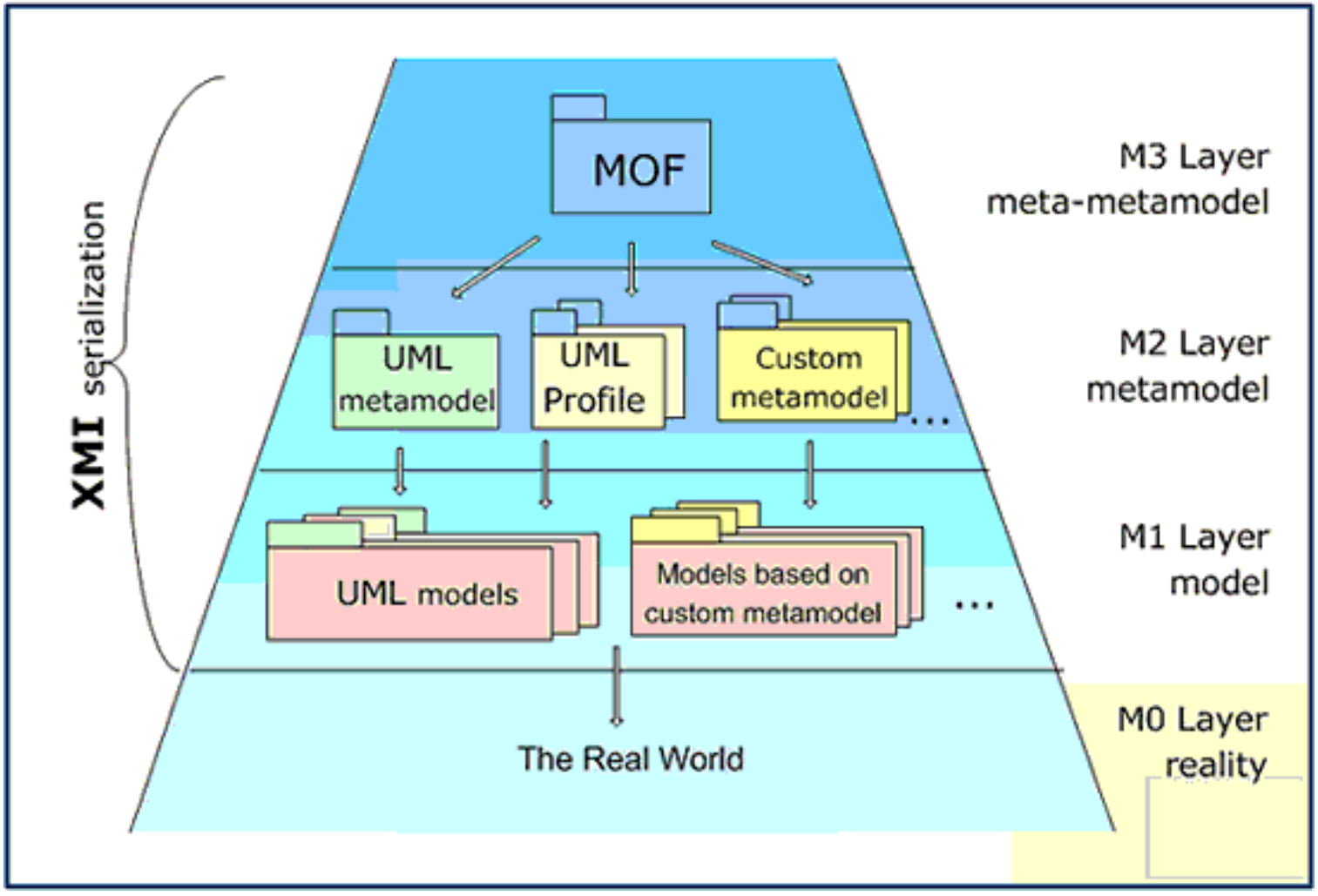
Language Definition Stack



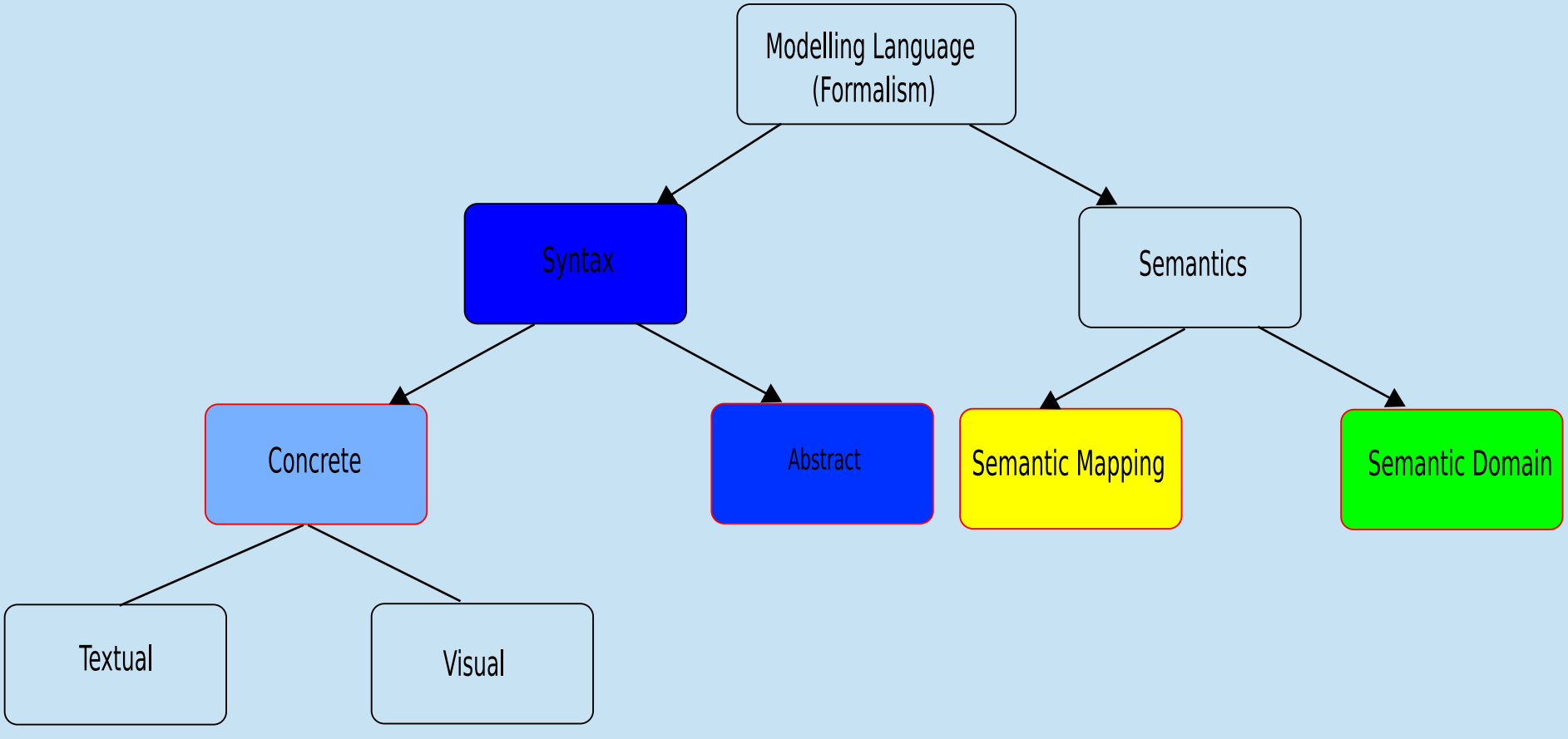
Meta-models as Language Definitions



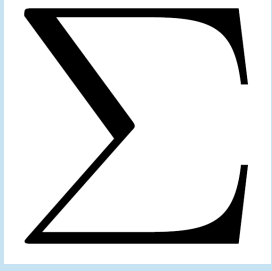
Meta-hierarchy – OMG's 4 Layer Architecture



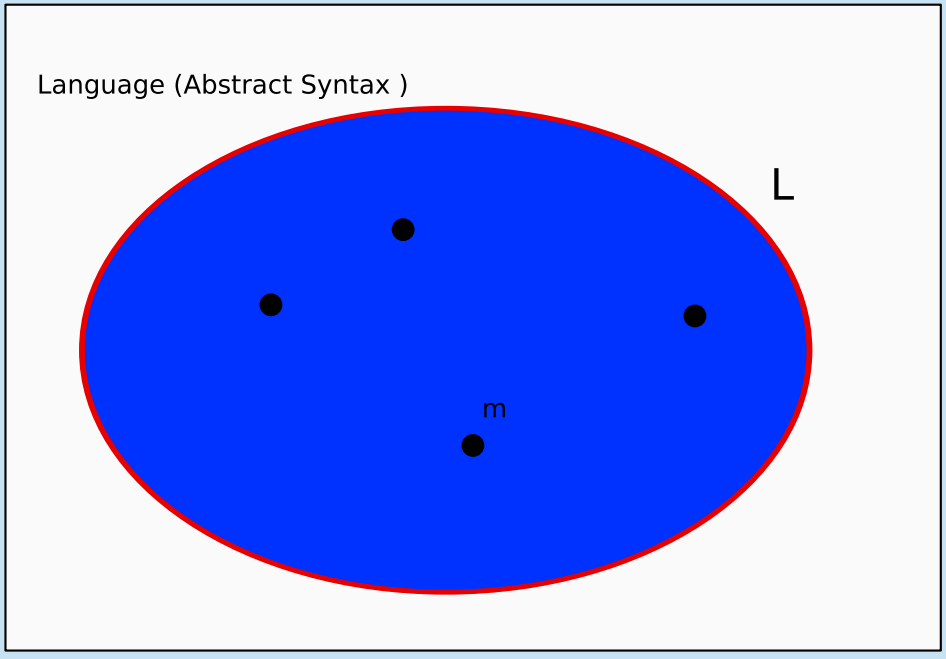
Dissecting a Modelling **Language**



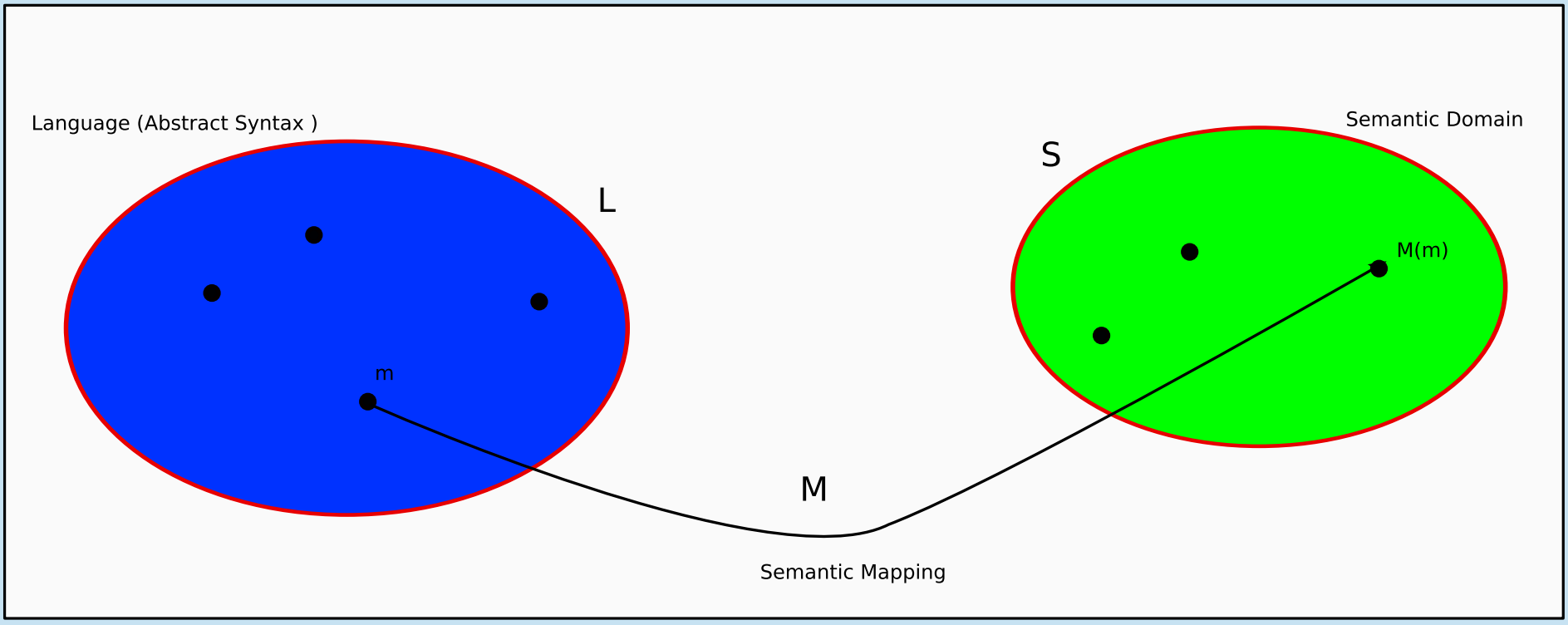
Deciding on terminology



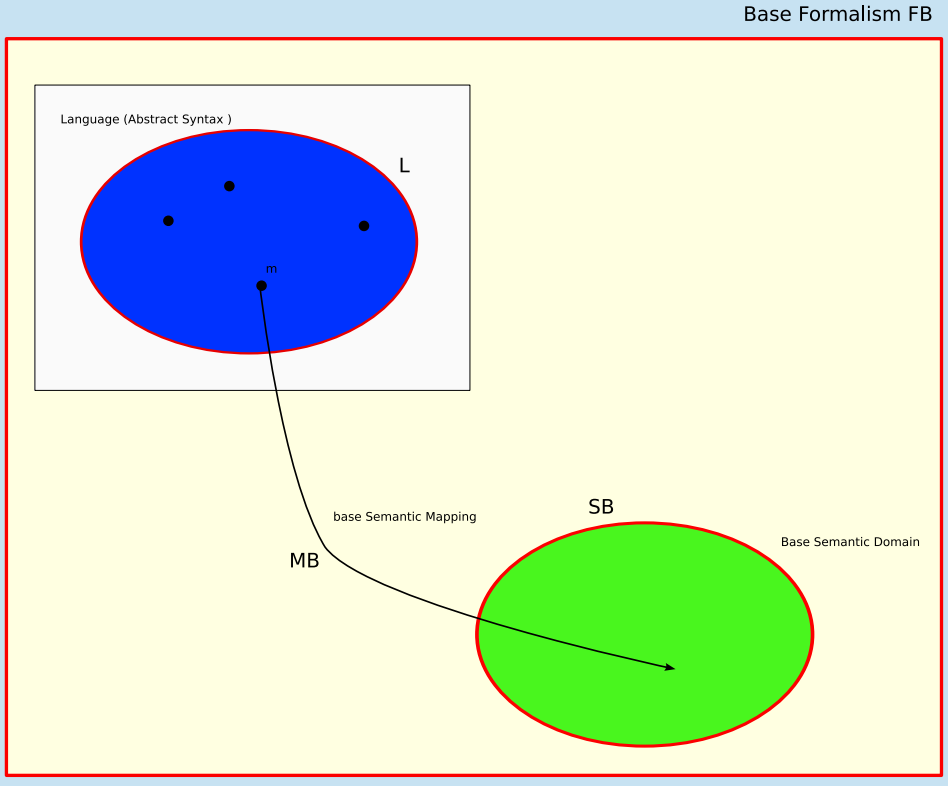
What's in a name ? **Language**



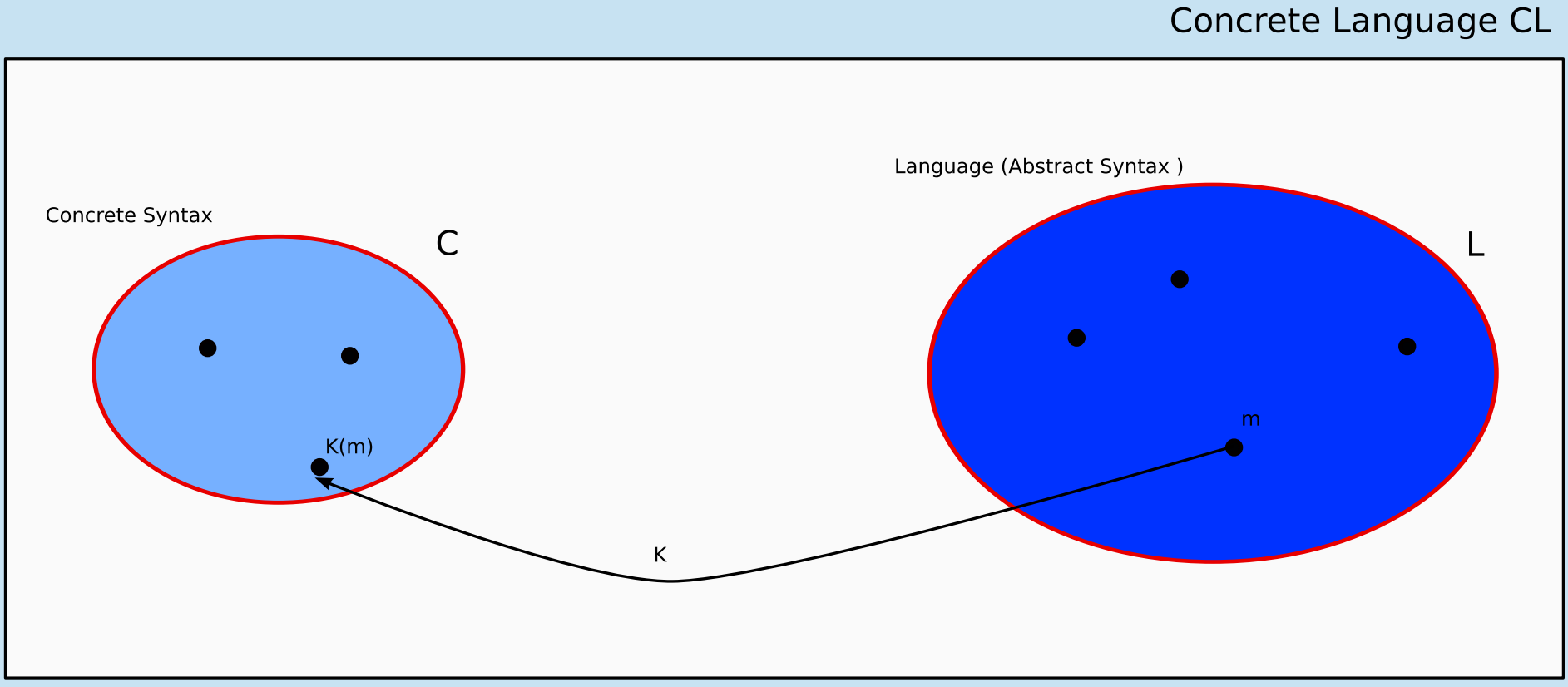
What's in a name ? **Formalism**



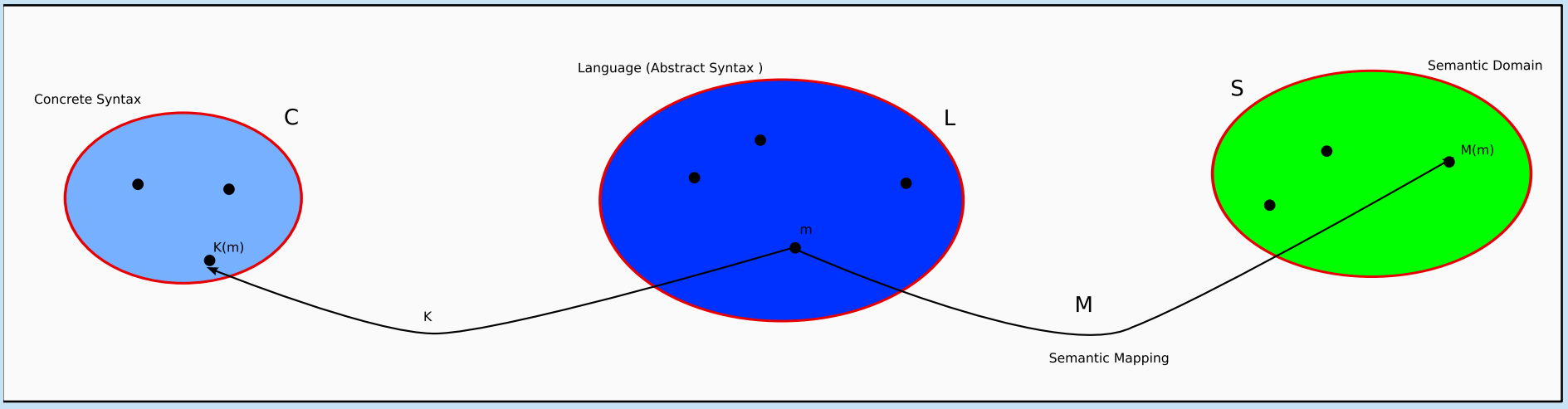
What's in a name ? **Base Formalism**



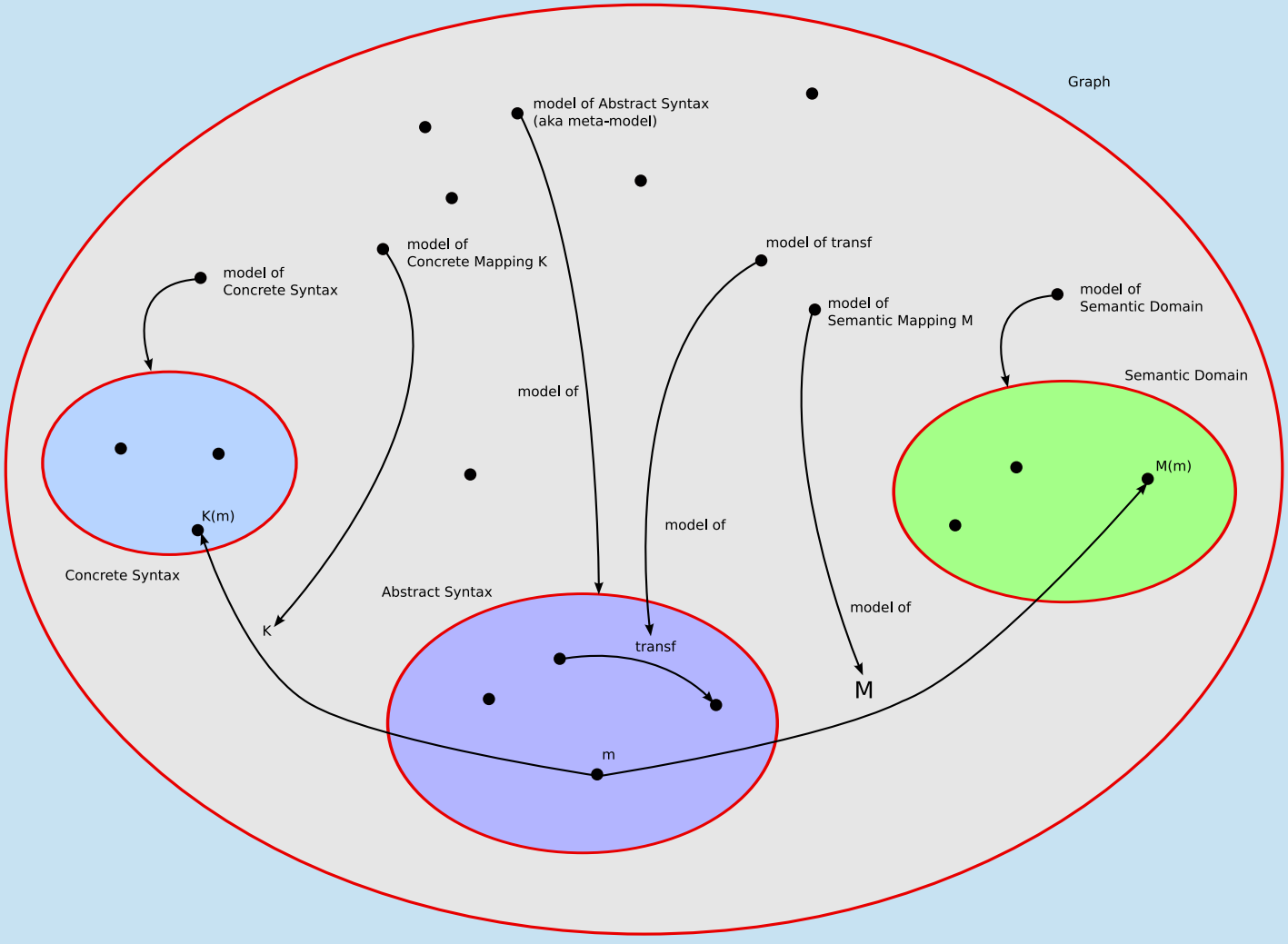
What's in a name ? **Concrete Language**



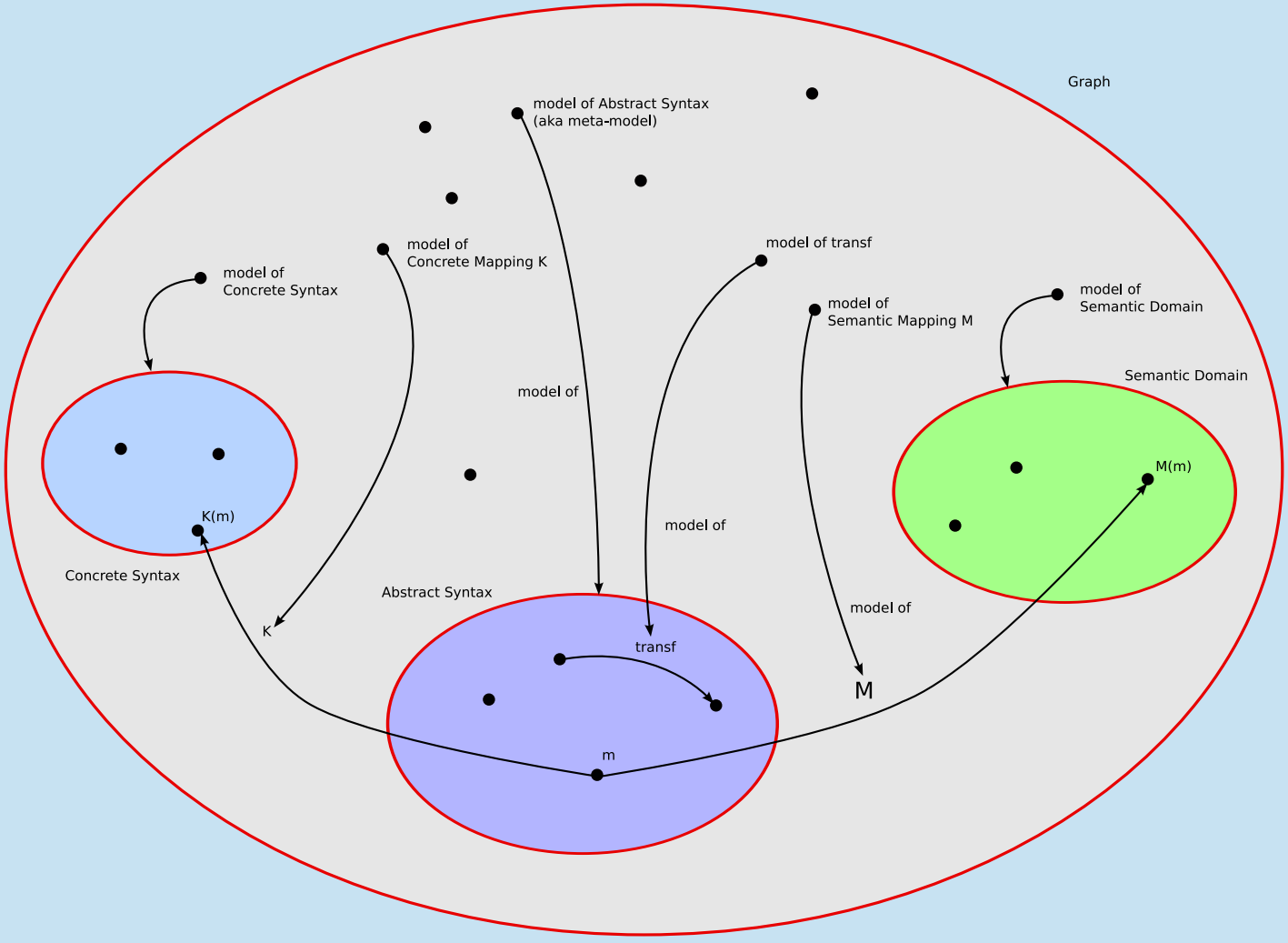
What's in a name ? **Concrete Formalism**



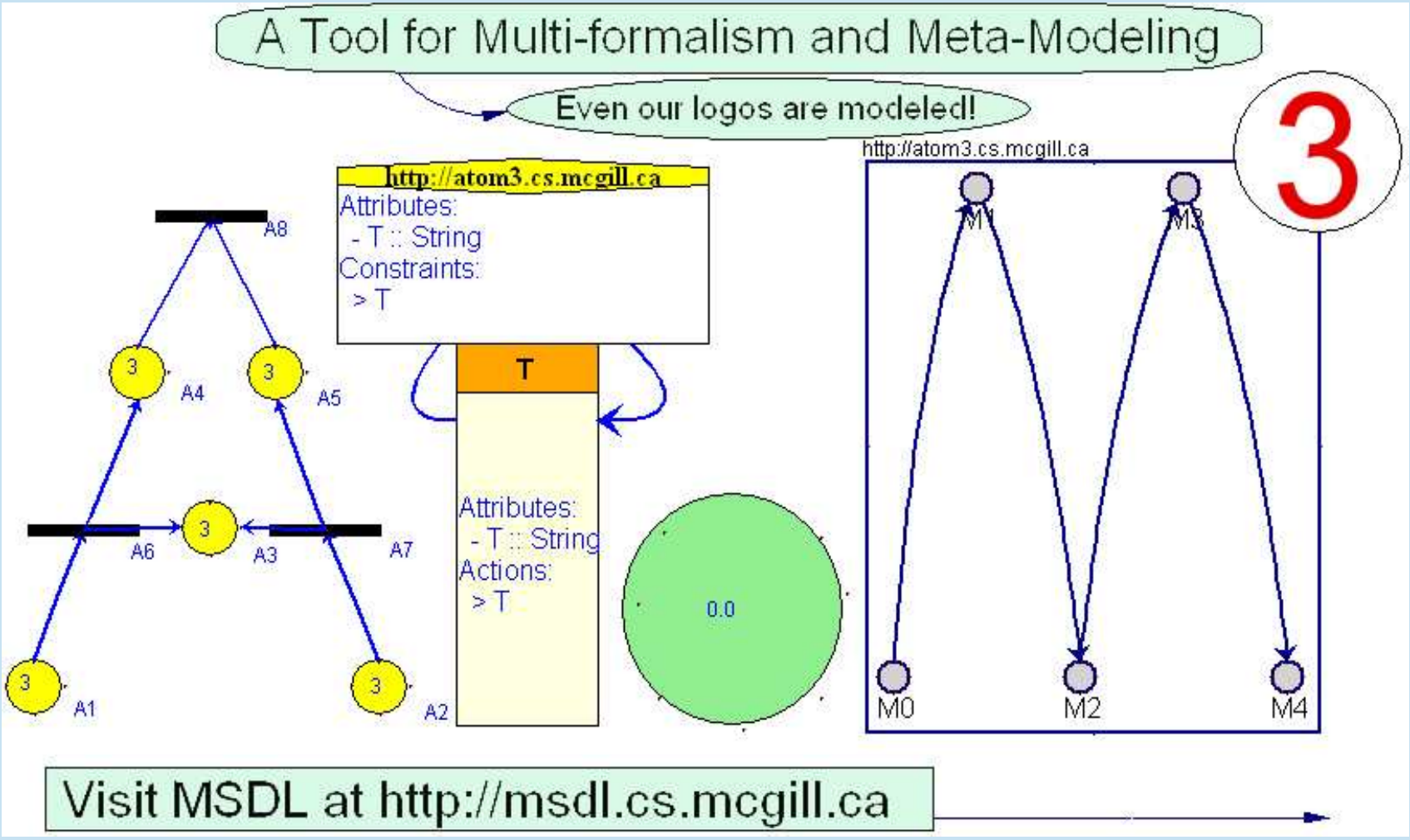
Modelling a Modelling Language/Formalism



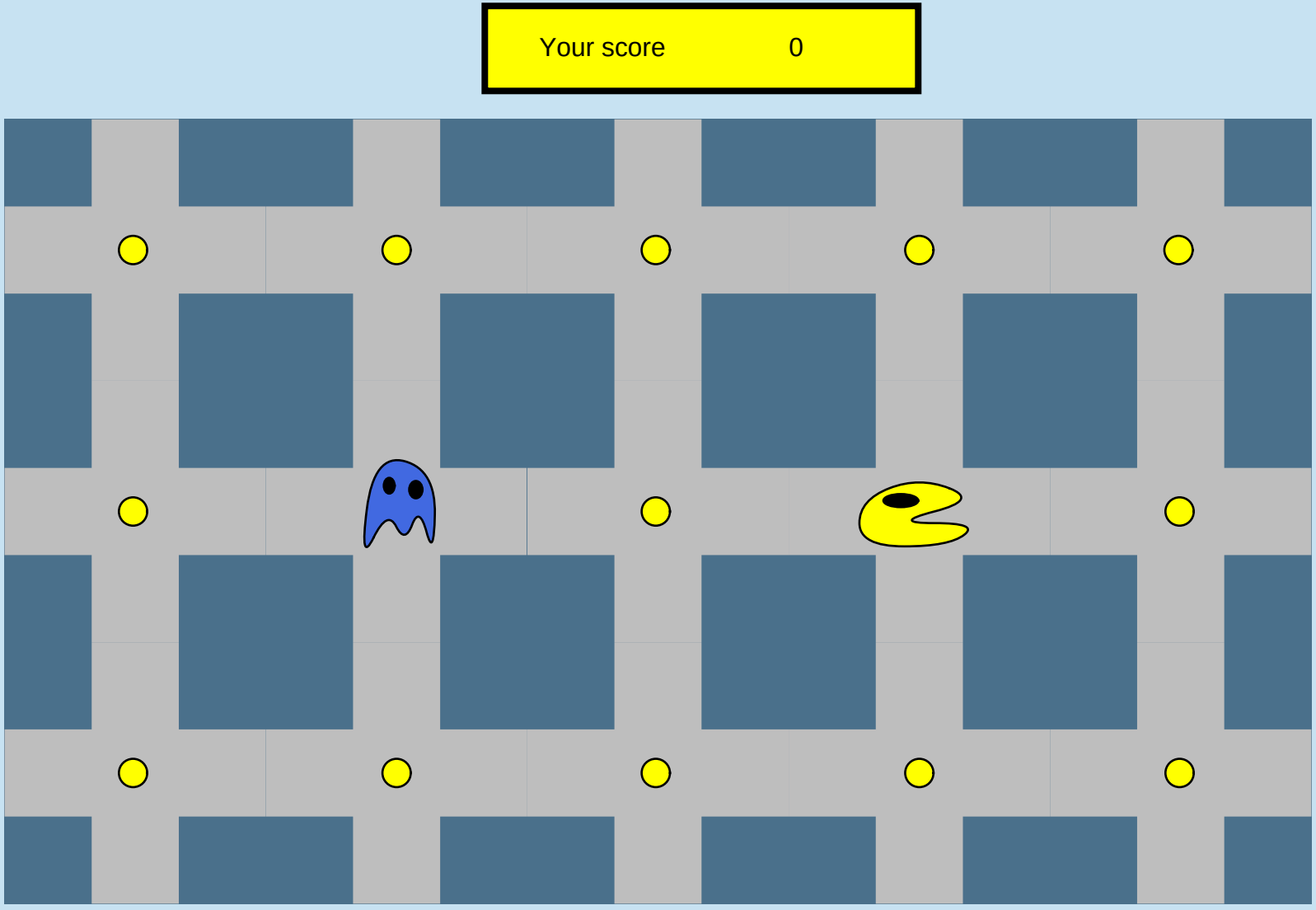
Sets of Models



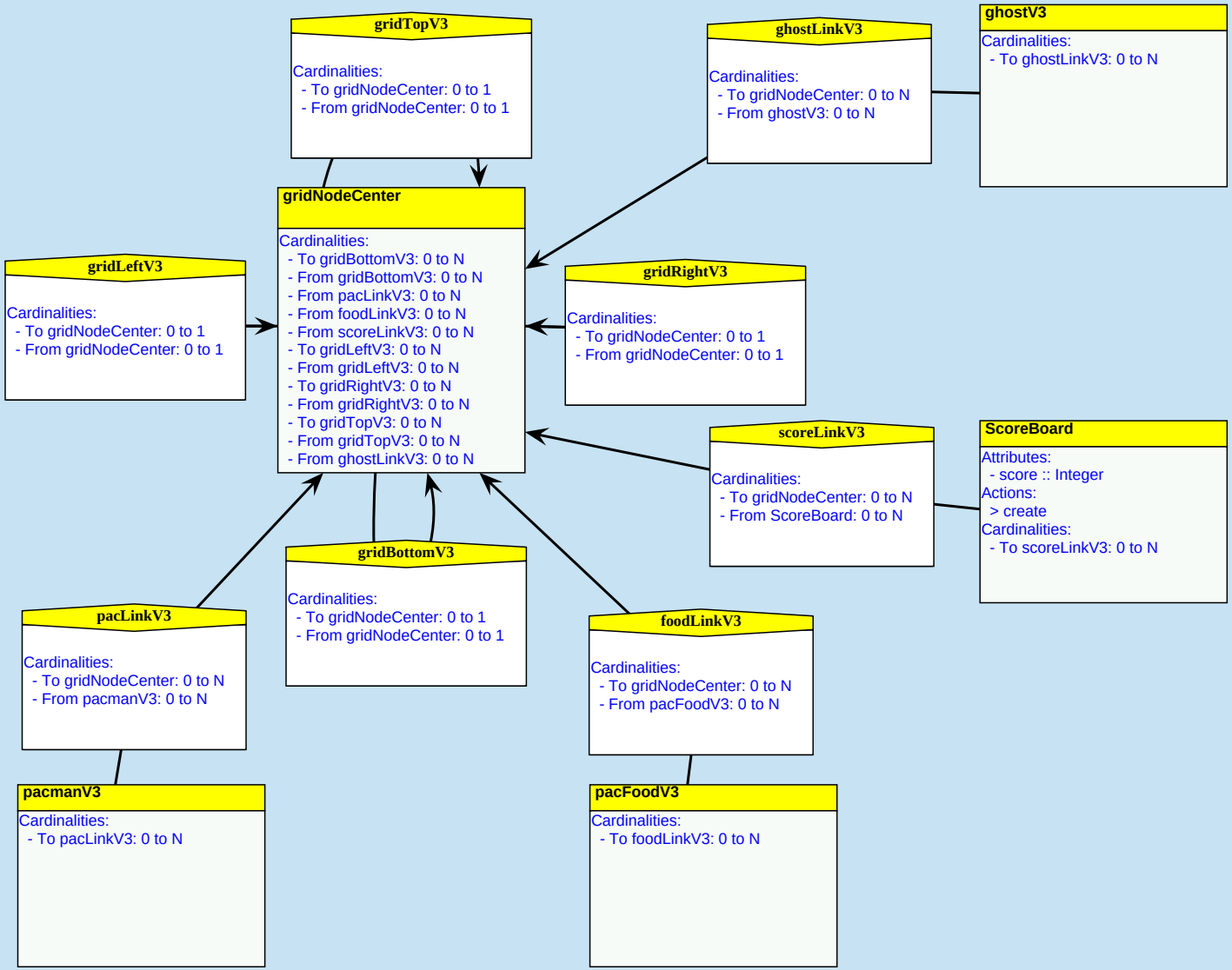
From now on: use AToM^3



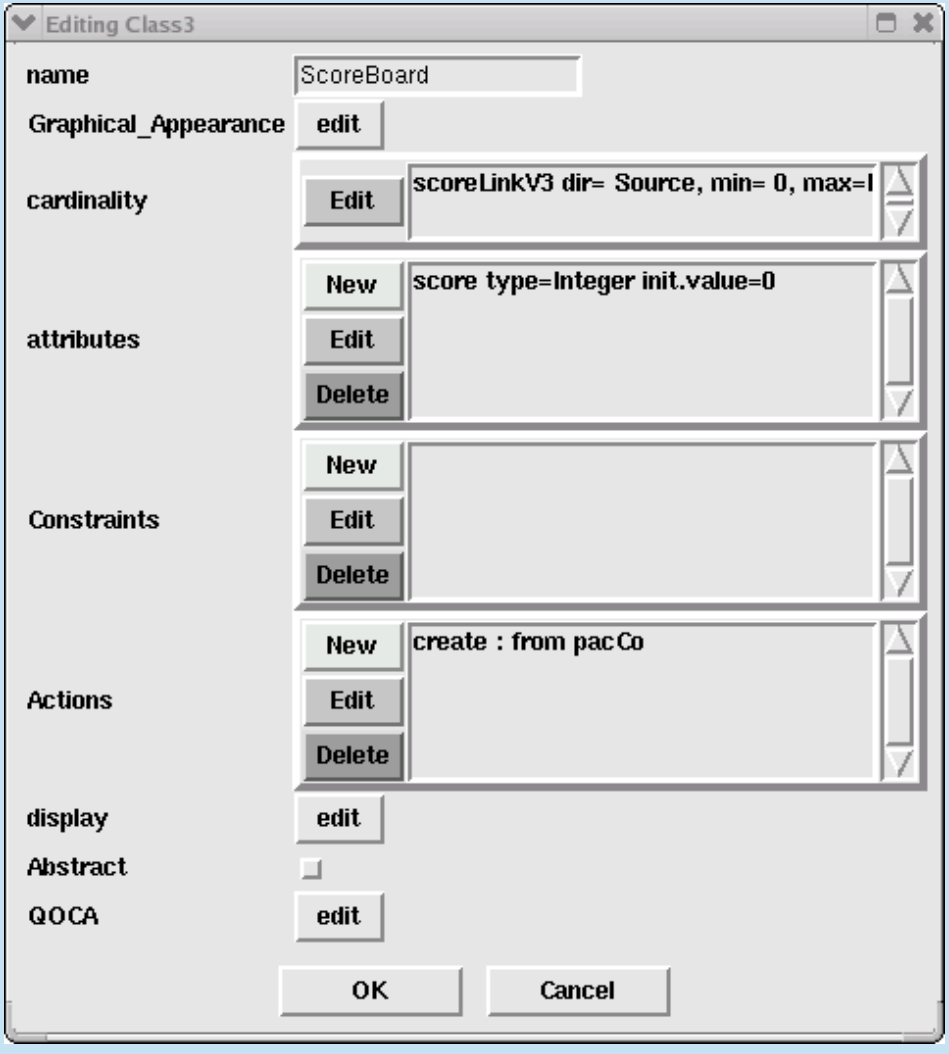
A model in the PacMan Formalism



Modelling Abstract Syntax (meta-model)



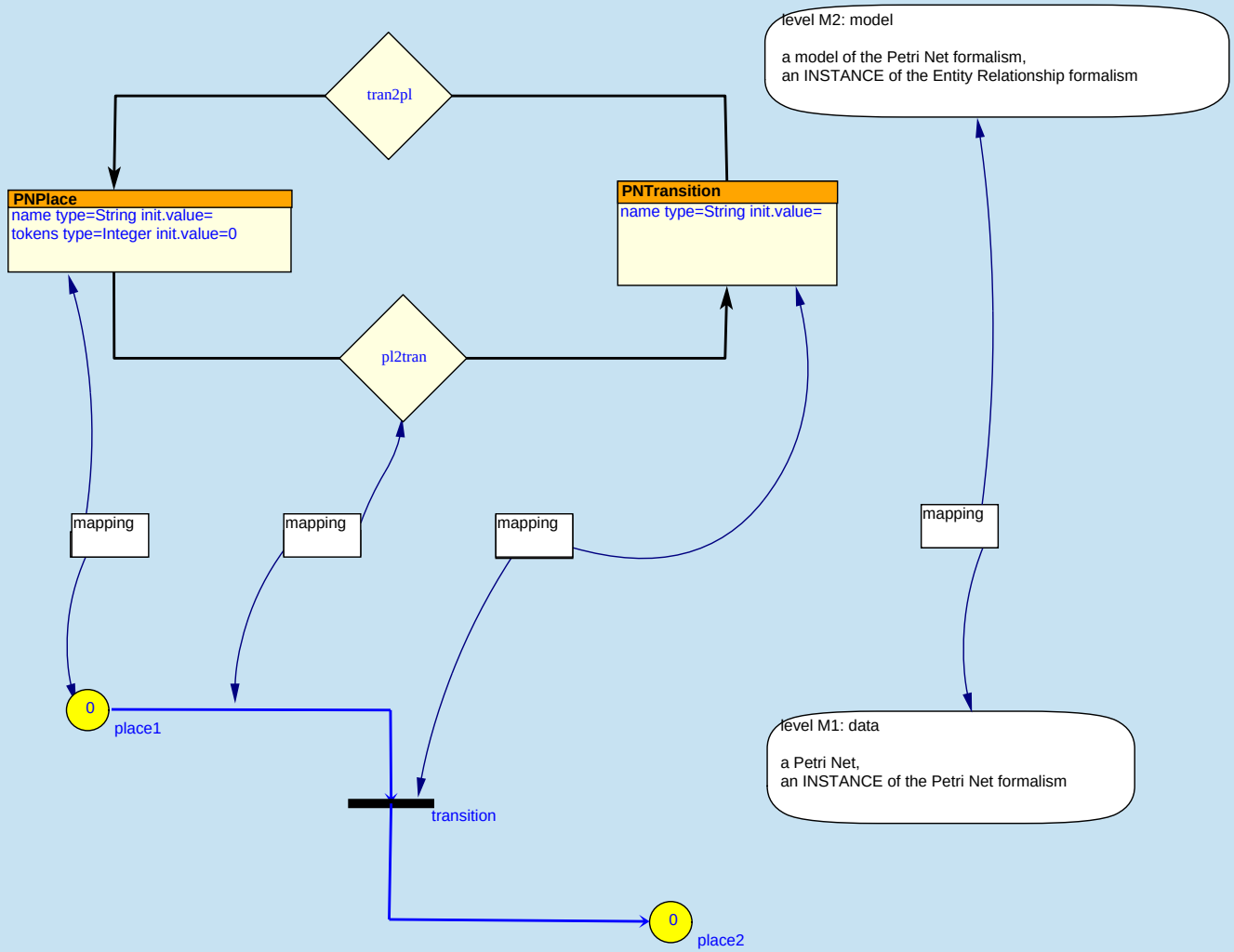
Modelling the Scoreboard Entity



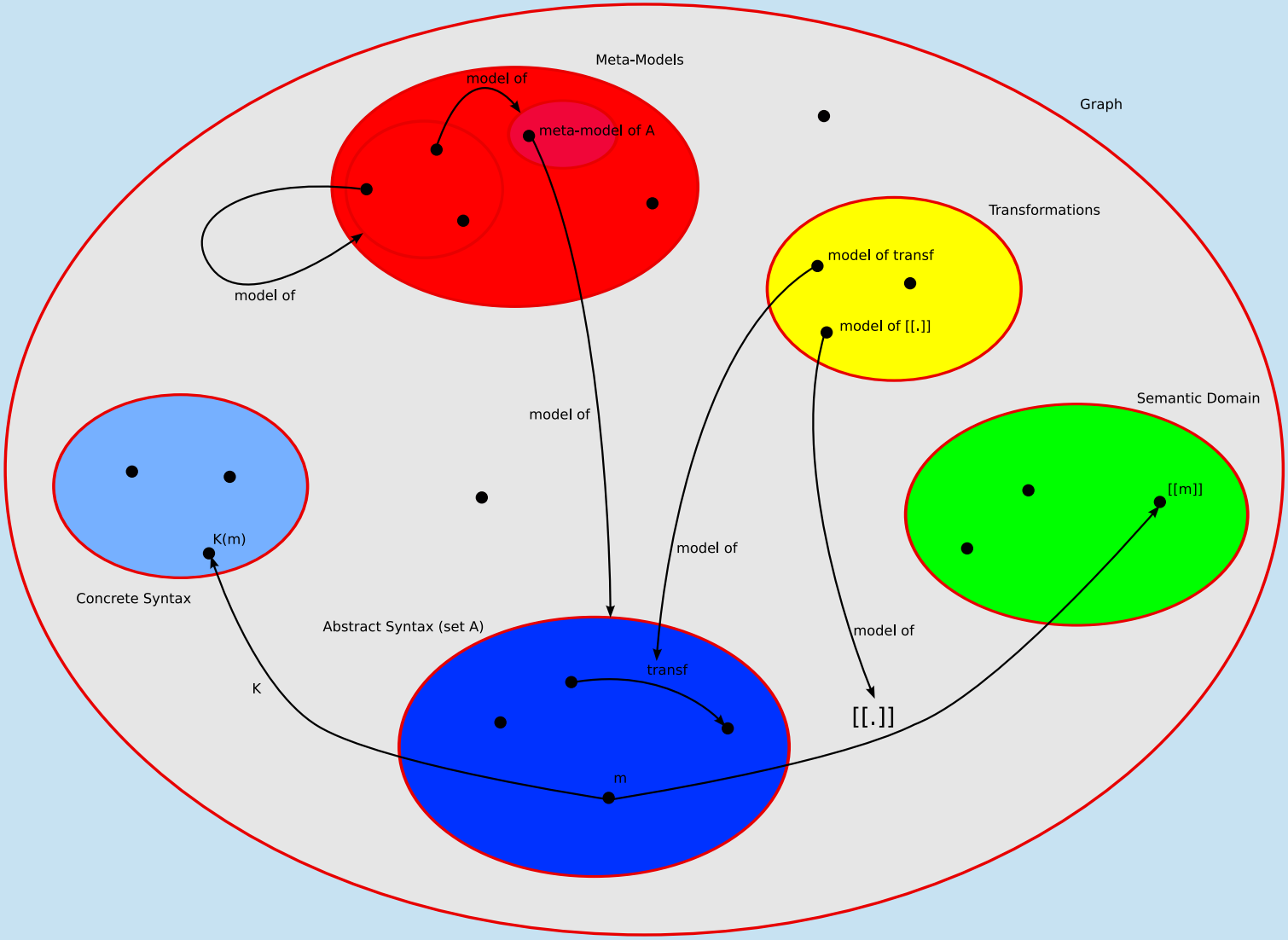
Synthesis of Code for Syntax-Directed Editing

```
class ScoreBoard(ASGNode, ATOM3Type): # Abstract Syntax only

 def __init__(self, parent = None):
    ASGNode.__init__(self)
    ATOM3Type.__init__(self)
    self.graphClass_ = graph_ScoreBoard
    self.isGraphObjectVisual = True
    self.parent = parent
    self.score=ATOM3Integer(0)
    self.generatedAttributes = {'score': ('ATOM3Integer' ) }
    self.directEditing = [1]

 def clone(self):
    cloneObject = ScoreBoard( self.parent )
    for atr in self.realOrder:
     cloneObject.setAttrValue(atr,self.getAttrValue(atr).clone())
    ASGNode.cloneActions(self, cloneObject)
    return cloneObject
```

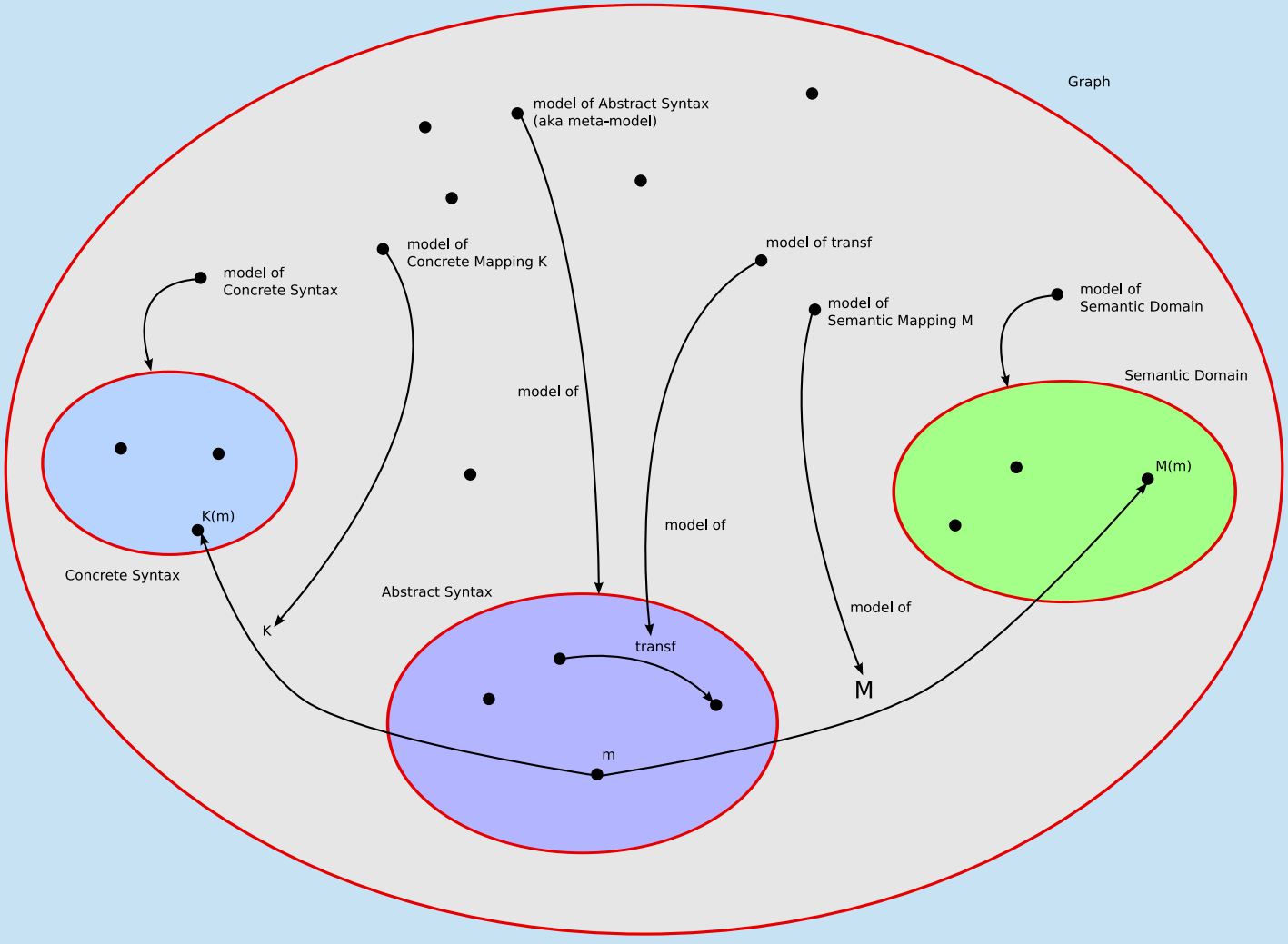
Meta-modelling: model-instance morphism



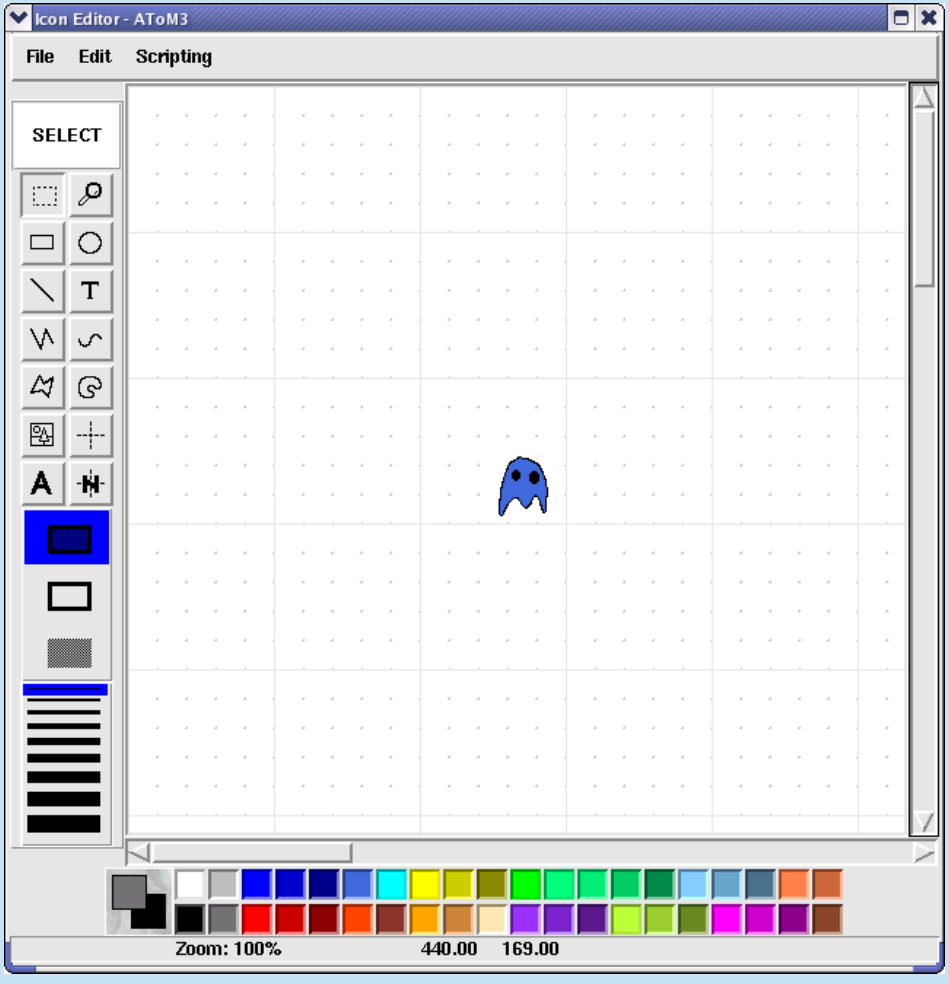
Meta-meta-. . . : Meta-circularity



Sets of Models: Modelling Concrete Syntax



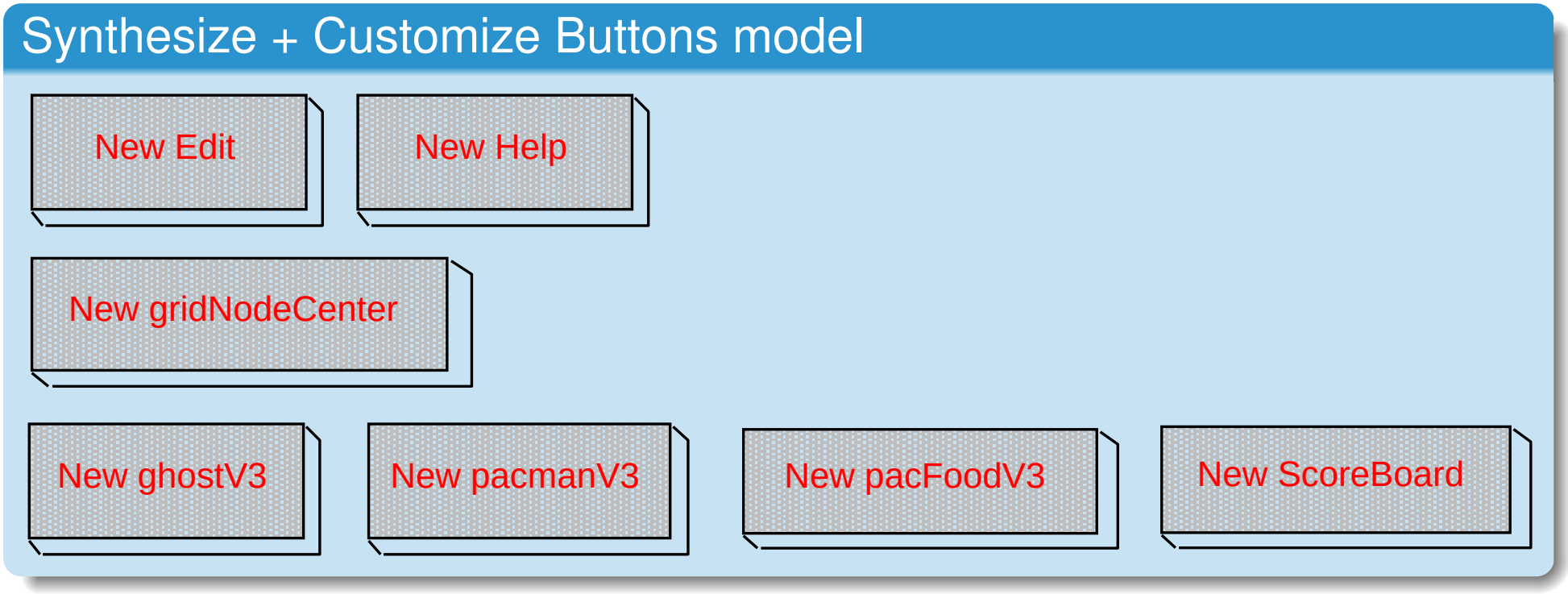
Modelling Ghost Class Instances Concrete Visual Syntax



Modelling PacFoodLink Association 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()
```

Synthesize + Customize Buttons model

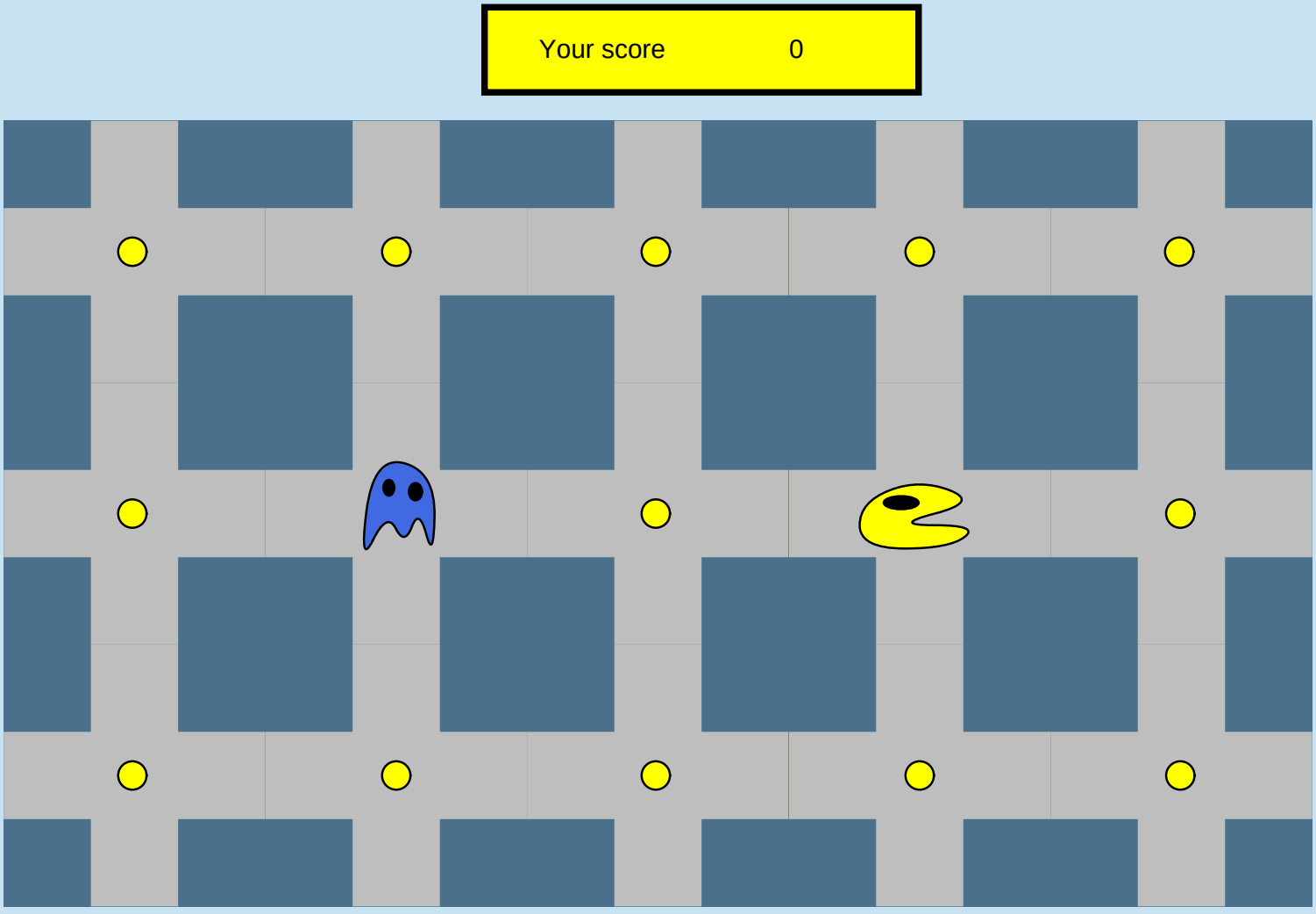


Note: create *vs.* execute

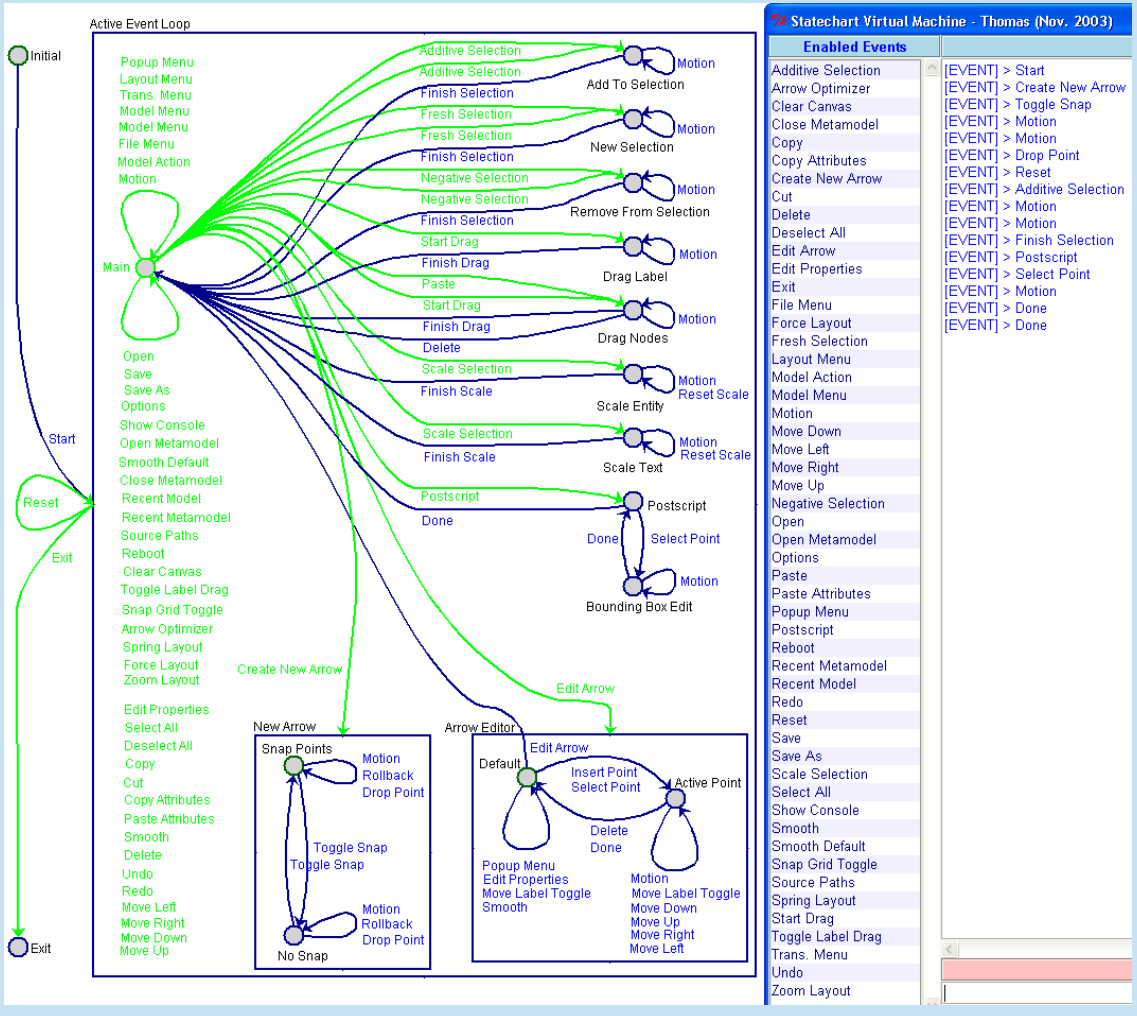
Default generated Buttons code for ghostV3

```
# This method has as parameters:
#  - wherex: X Pos. in window coordinates where user clicked.
#  - wherey: Y Pos. in window coordinates where user clicked.
newPlace = self.createNewghostV3 (self, wherex, wherey)\n'))
```

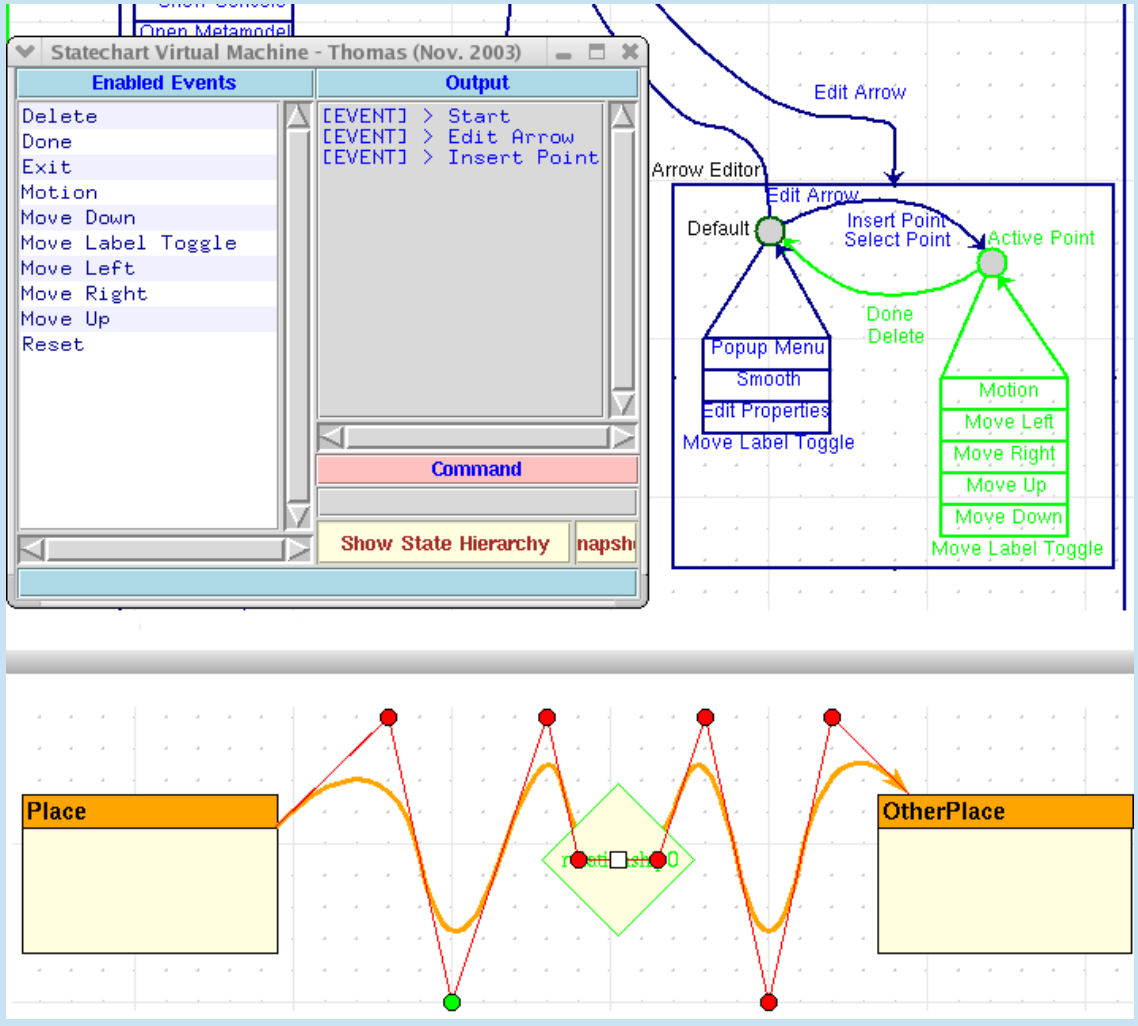
Can now build valid PacMan models ?



Model the GUI's Reactive Behaviour ! in the most appropriate formalism . . . Statecharts



The GUI's reactive behaviour in action

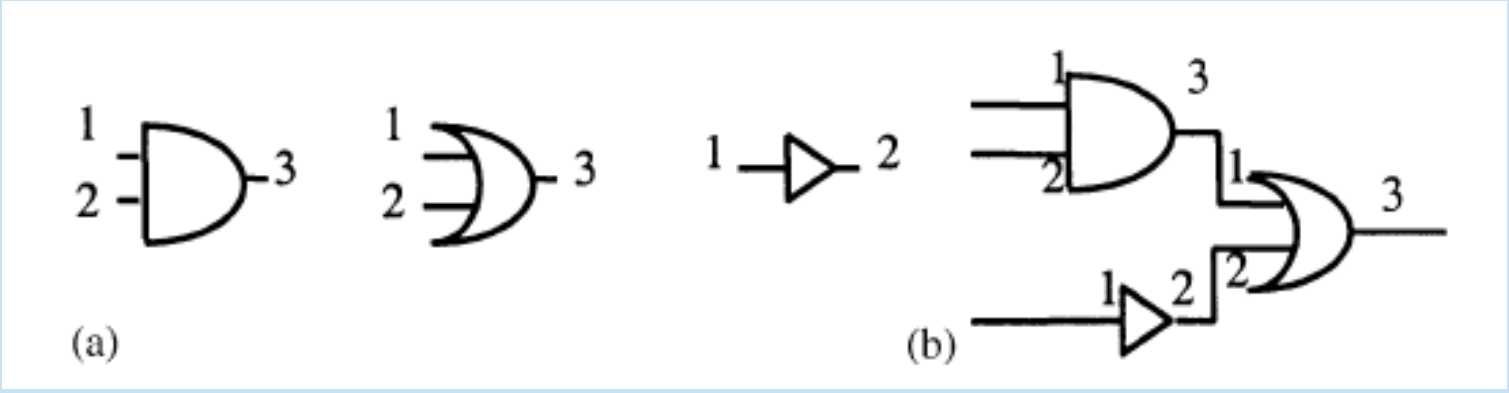


challenge: what is the *optimal* formalism to specify GUI reactive behaviour ?

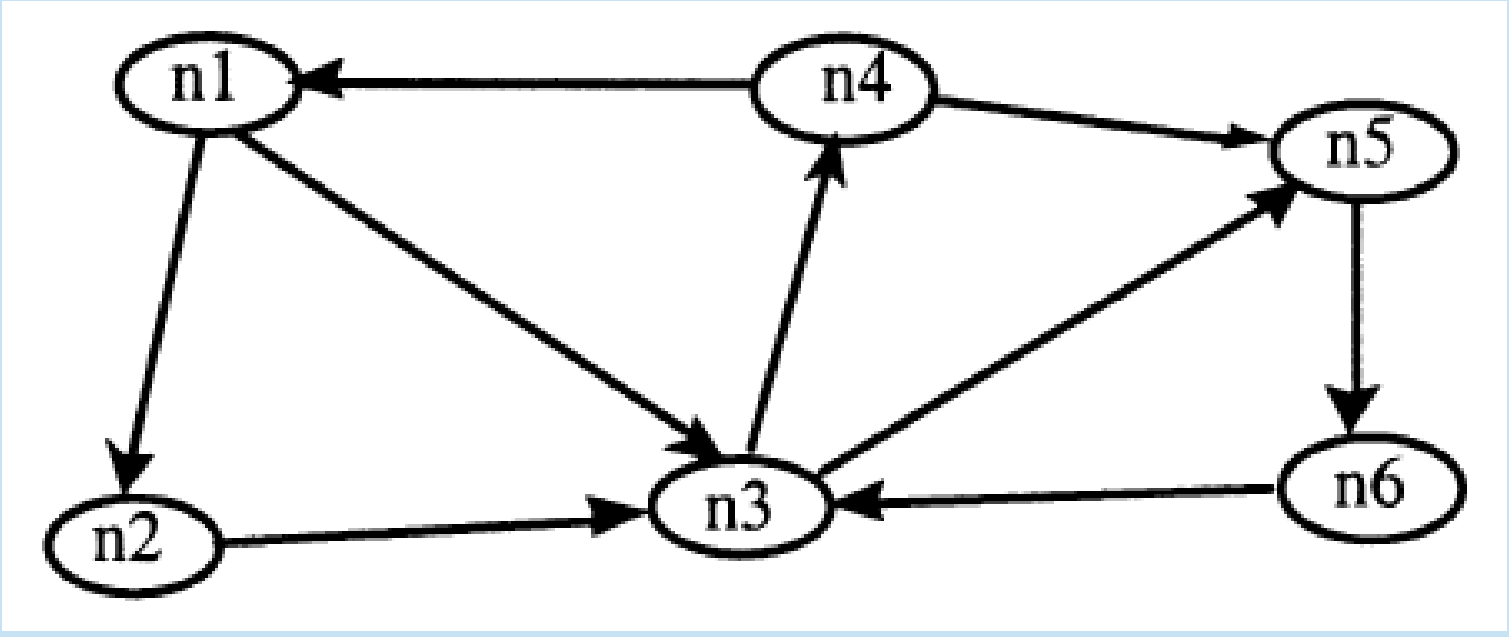
Concrete Visual Syntax

G. Costagliola, A. Delucia, S. Orefice and G. Polese.
A Classification Framework to Support the Design of Visual Languages.
Journal of Visual Languages and Computing, Volume 13, Issue 6, December 2002, pages 573-600.

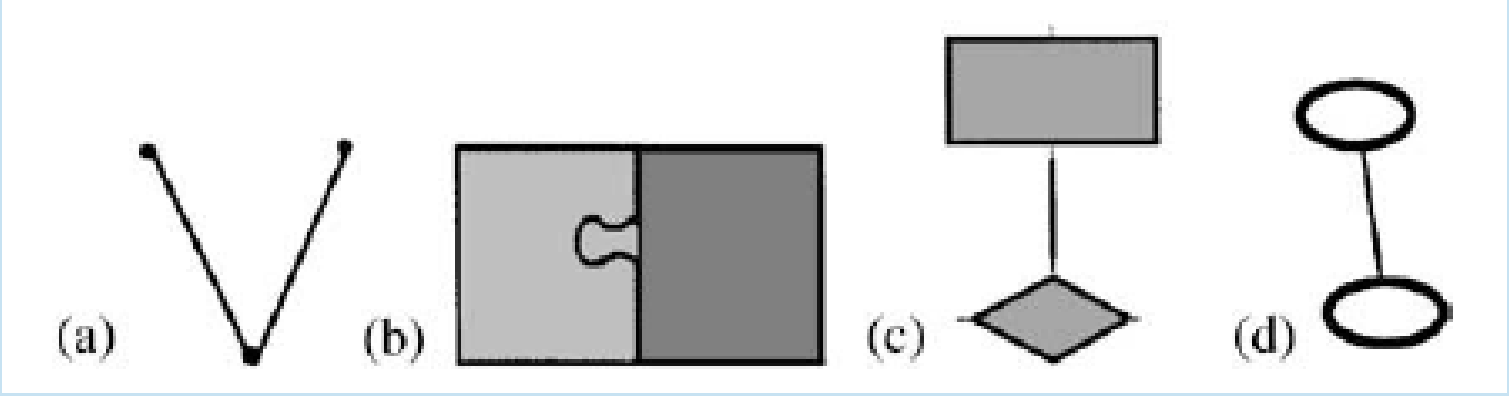
Plex



Graph



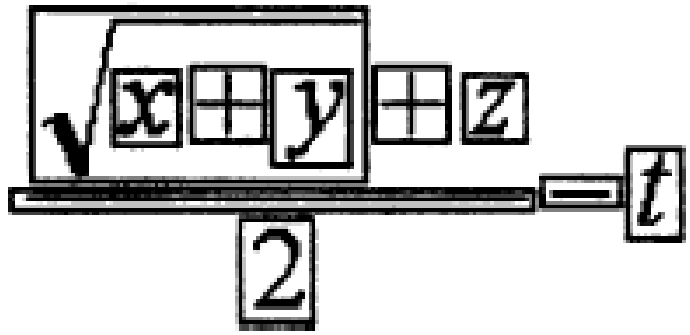
Connection Types



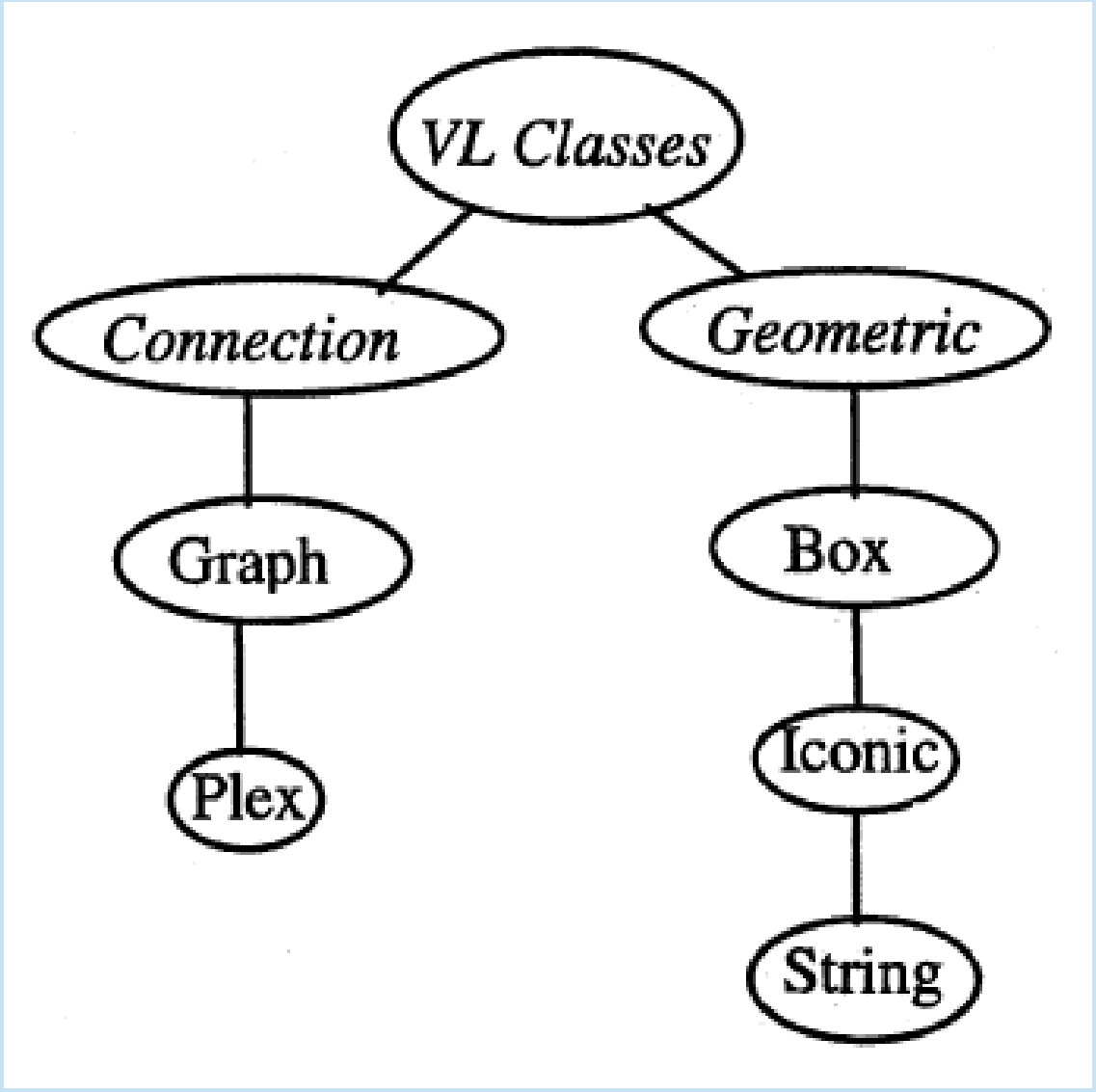
Iconic

Box

$$\frac{\sqrt{x+y}+z}{2}-t$$



Visual Language Classification



Hybrid Languages