NSERC Summer Research

26 August 2004

Domain-Specific Visual Modelling

Denis Dubé

School of Computer Science, McGill University, Montréal, Canada

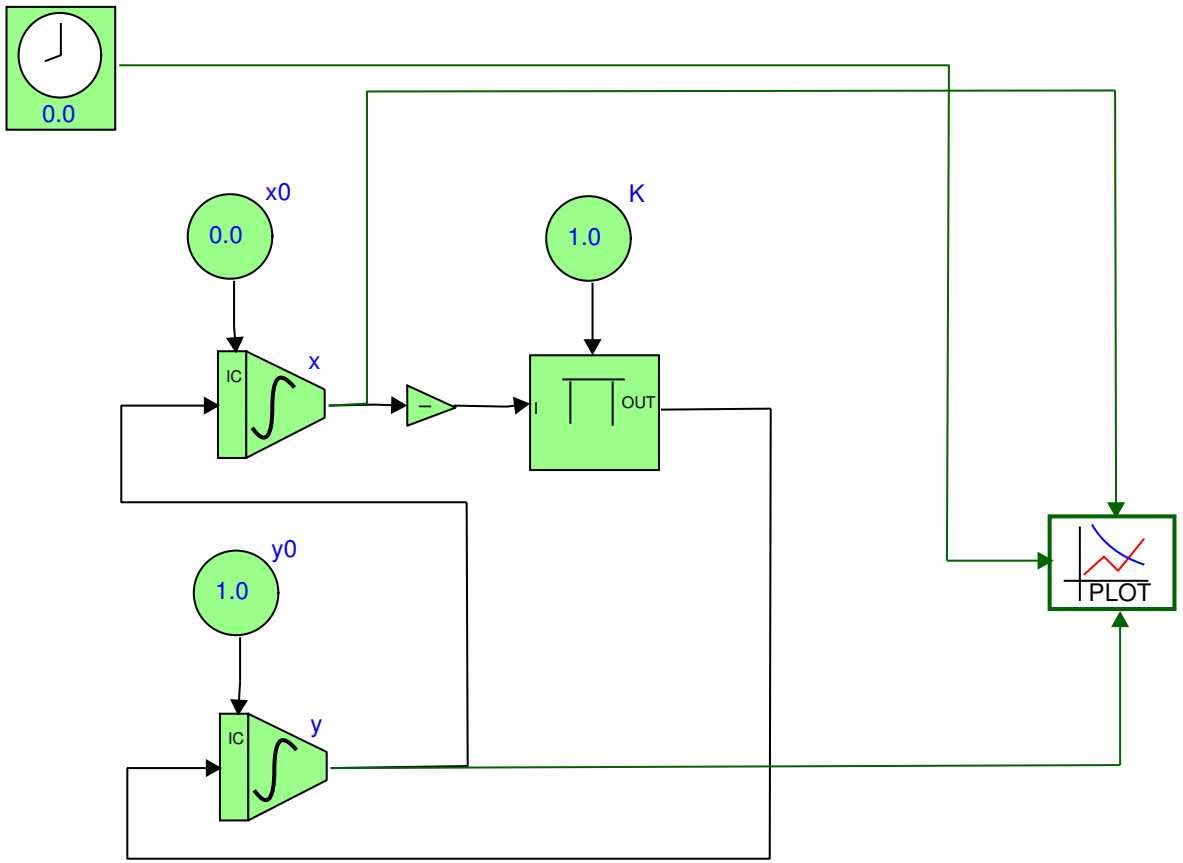
Modelling, Simulation & Design Lab (MSDL)

Domain-Specific Visual Modelling

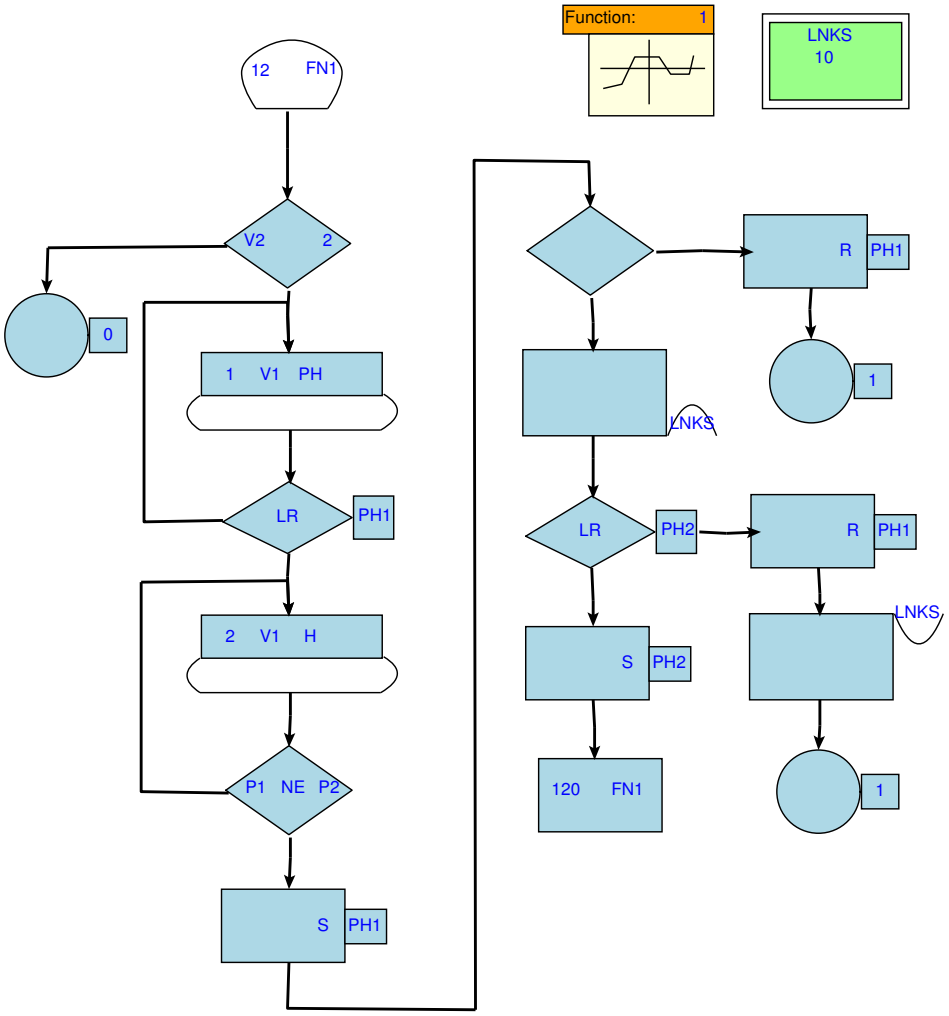
- Enables working directly with domain concepts
- High level of abstraction

- Some Examples:
  DCharts, StateCharts, Petri-Nets, GPSS, Timed Automata,
  Reachability Graphs, Causal Block

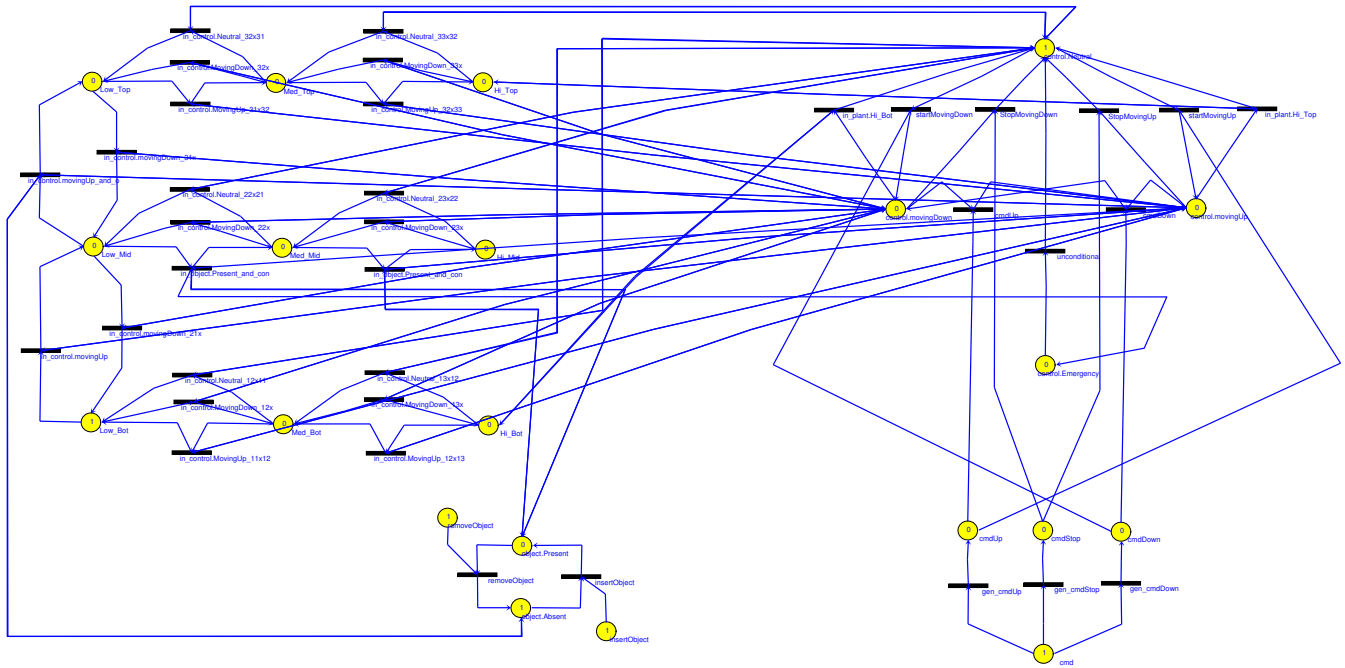
Causal Block Harmonic Oscillator



GPSS Telephone Simulation



Petri-Nets Power Window Controller



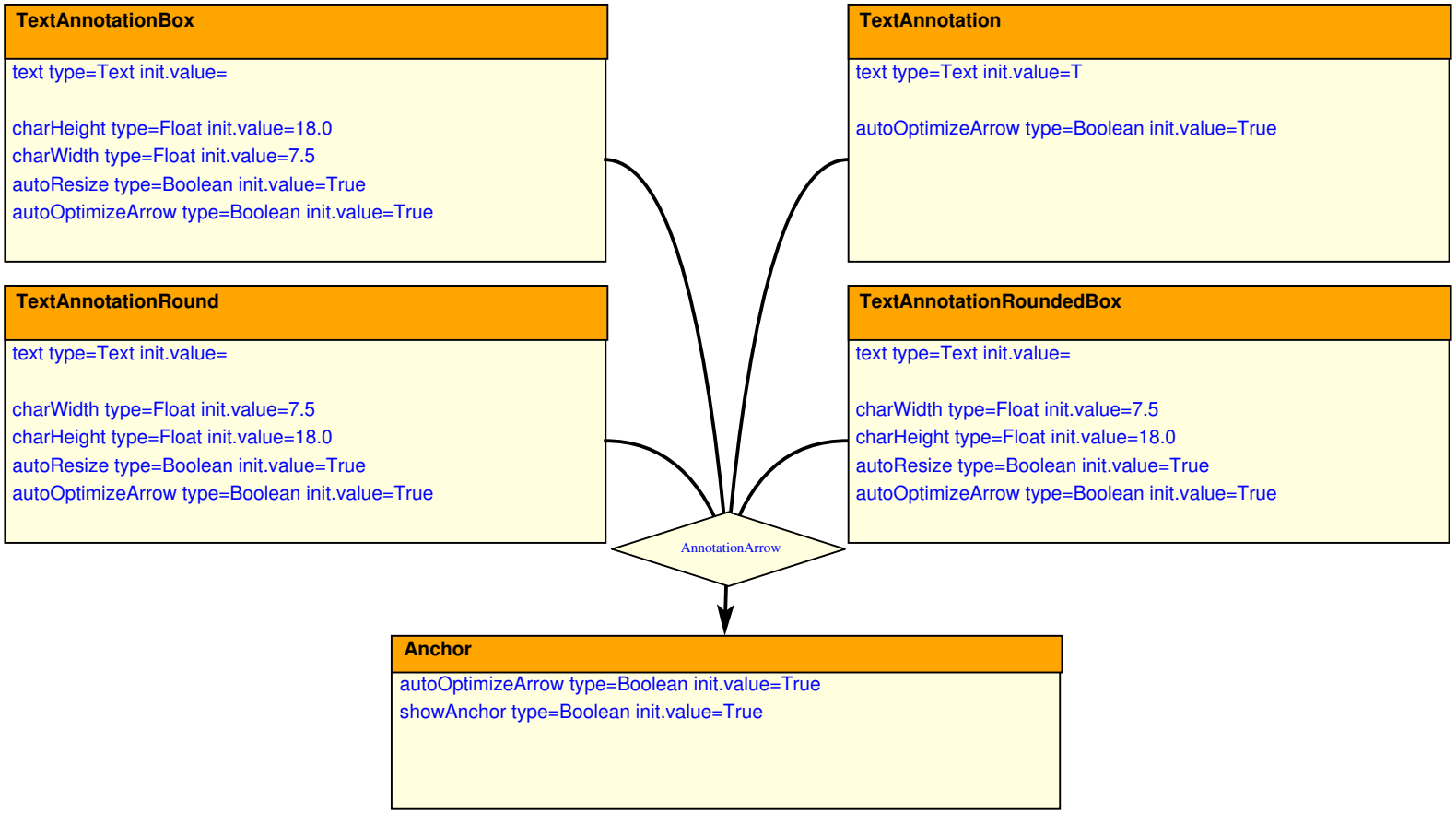
Domain-Specific Modelling Environment

- Meta-modelling specifies the *syntax* of domain specific modelling formalisms explicitly, in the form of a model
- Thus a meta-modelling tool allows domain experts to build a meta-model and *synthesize* a domain-specific modelling environment from it.
- One such tool is AToM^3 (A Tool for Multi-formalism Meta-Modelling), developed by the Modelling, Simulation and Design Lab

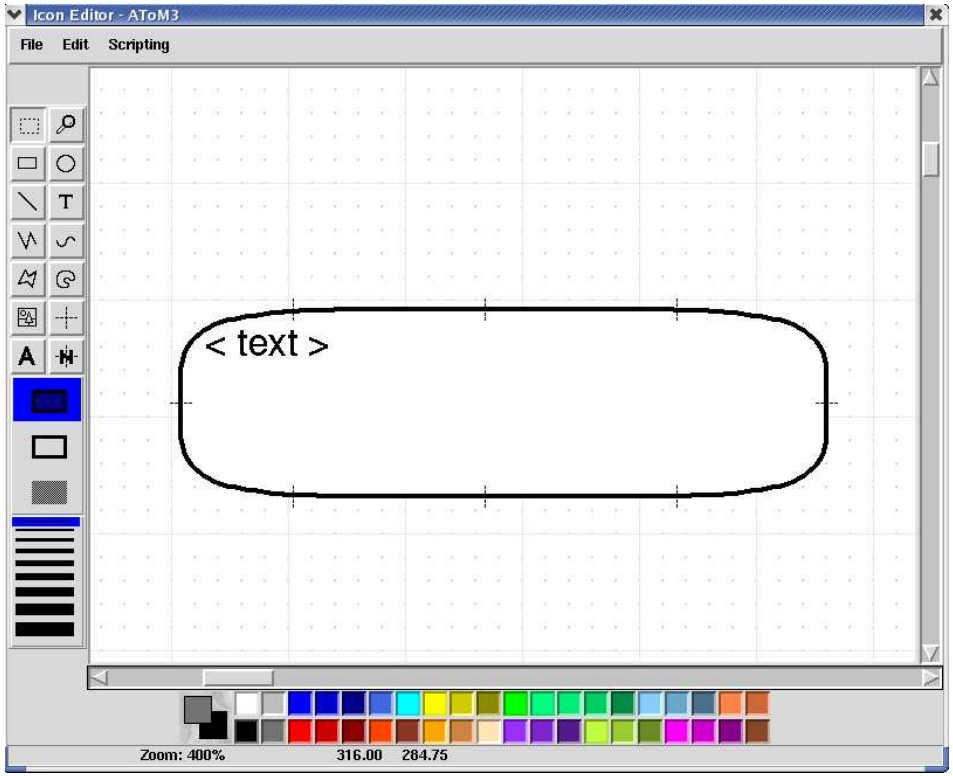
Visually Modelling The Syntax

- Enables intuitive creation of meta-models
- Visual entities are connected to denote relationships
- Dynamic visual attributes such as names can be set
- Dynamic pre/post conditions can be set to alter model behaviour

Annotation Meta-Model



Icon-Editor

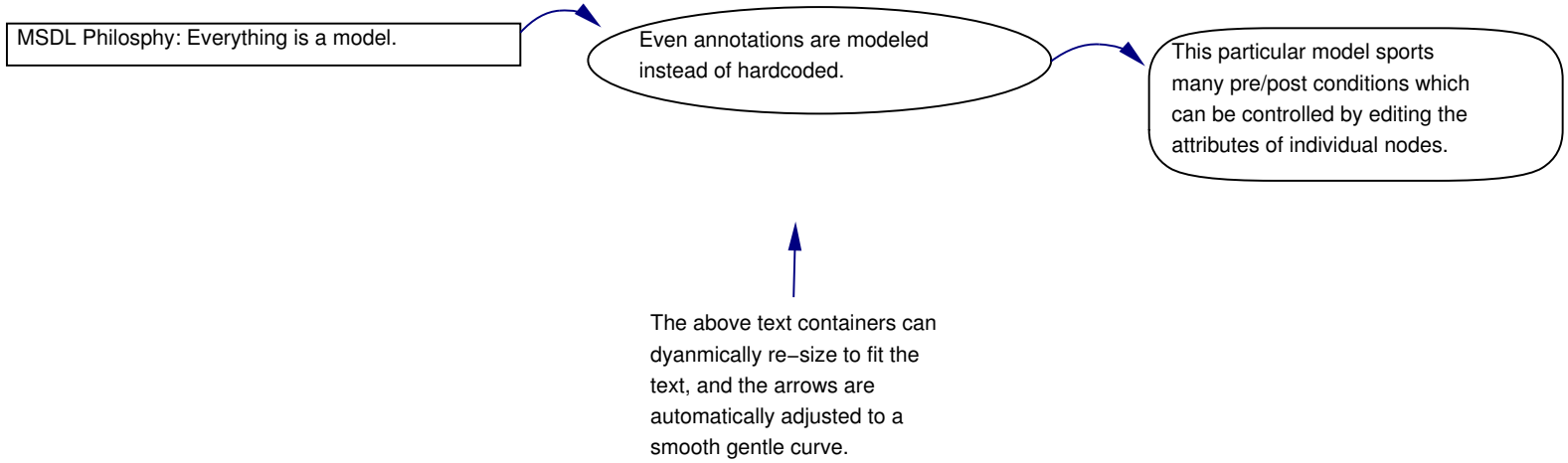


Francois Plamondon

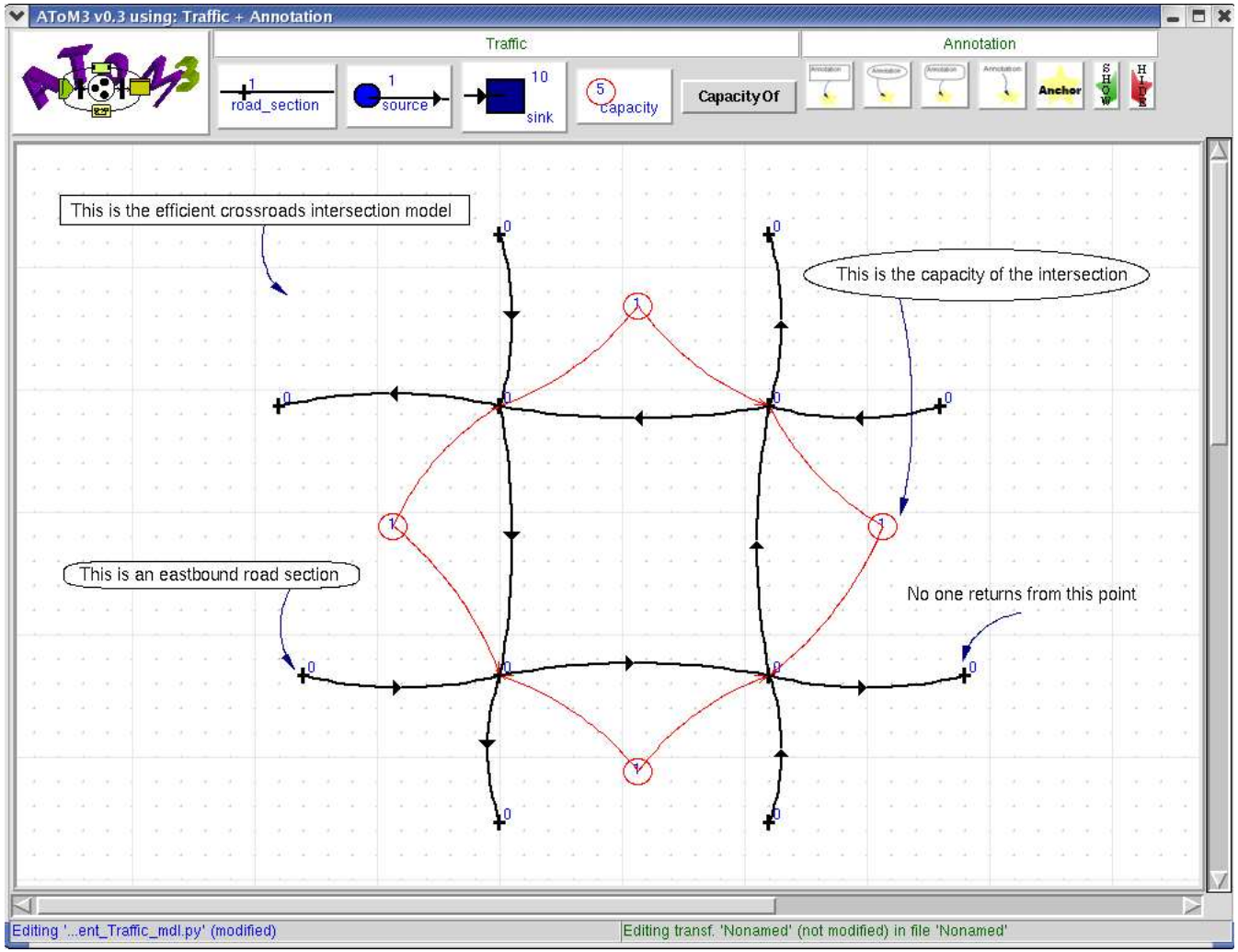
NSERC USRA, Summer 2003

http://moncs.cs.mcgill.ca/people/fplamo/summerwork.dtml

Annotation Model



Traffic + Annotation Model



Visual Modelling Environment

- Thus far, the visual meta-modeling process is described from a static point of view
- The key interactive components of a visual modelling environment are:

  i) Visual environment behaviour

  ii) Layout in static models, unchanging

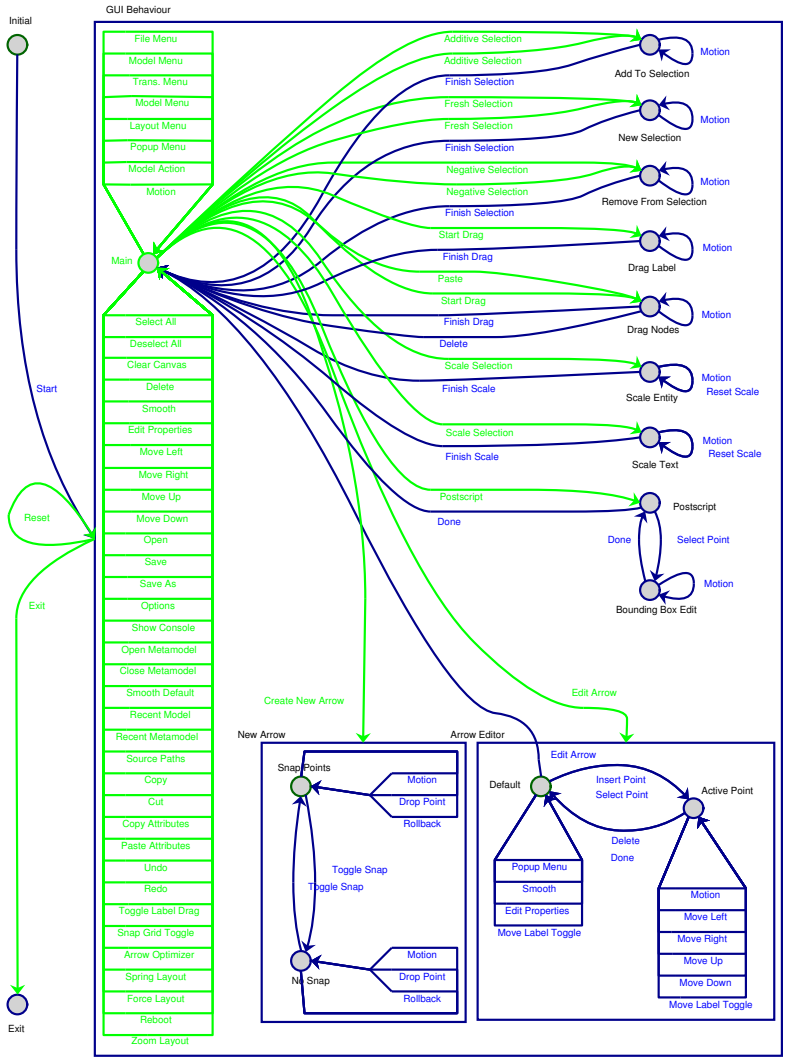
  iii) Layout in dynamic models, undergoing graph transformations

Visual Environment Behaviour

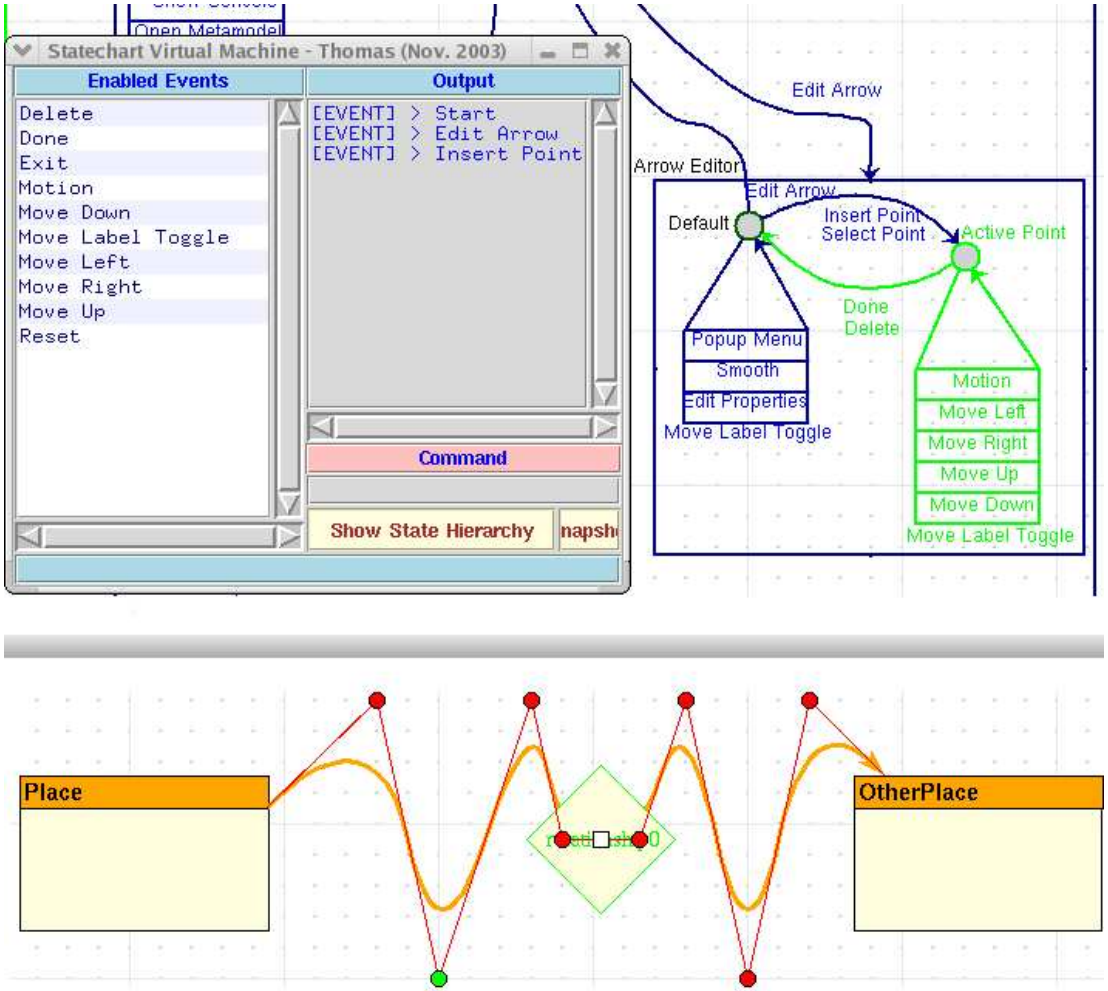
- Philosphy: ”Everything is modelled explicitly”
- The behaviour was modeled as a DChart, a form of StateCharts, that is in turn a form of finite state automata
- The model was then simulated with SVM to ensure correct behaviour
- Python code was generated from the model using SCC

DCharts, SVM, and SCC were developed in **Thomas Feng**’s M.Sc. thesis.

DCharts GUI Behaviour



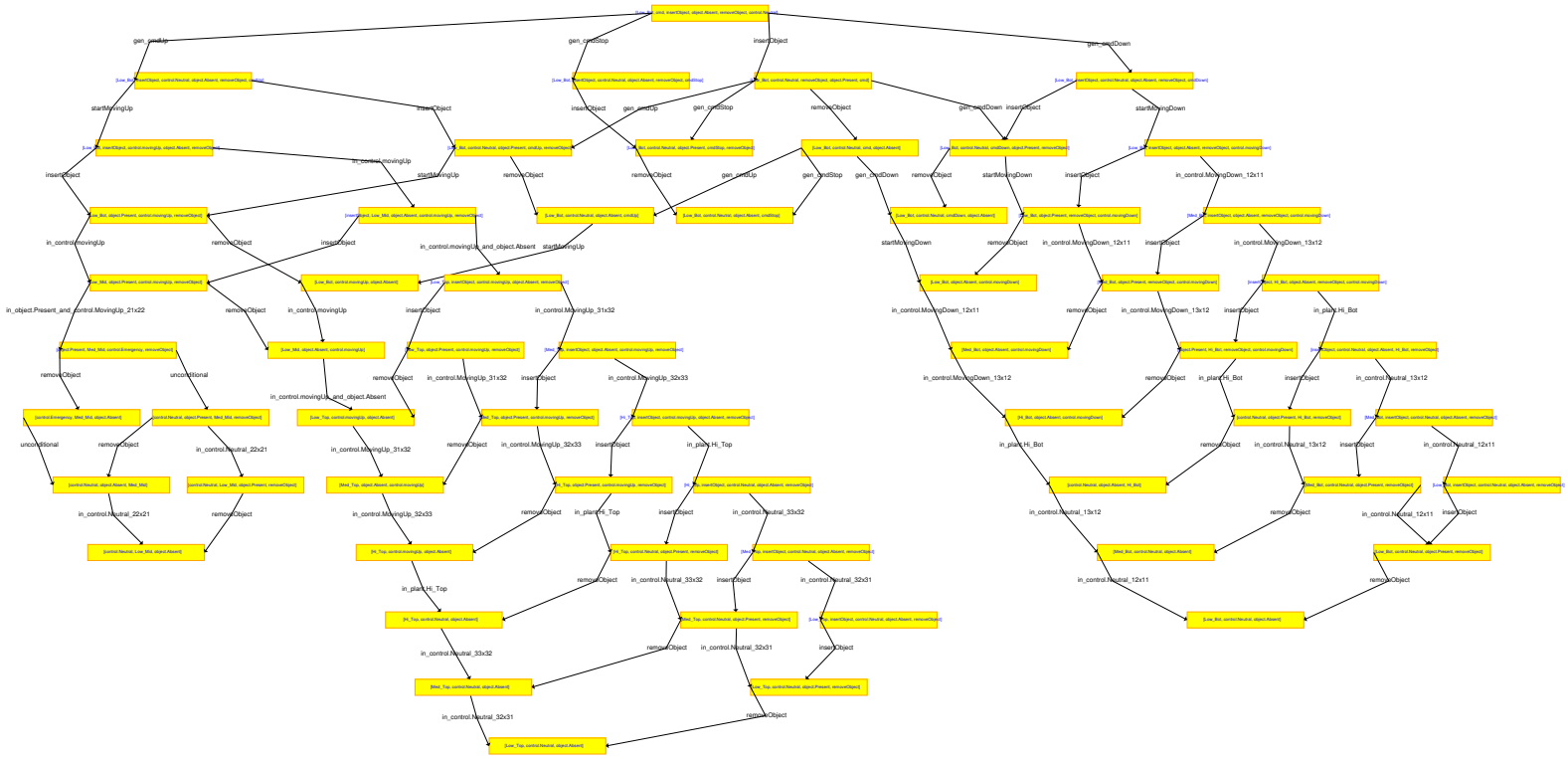
DCharts SVM vs In Action



Static Layout

- An extensive review of the existing litterature and tools was conducted
- In particular, one tool, yED, proved very powerful yet free to download
- Thus the ability to export AToM^3 models to several common graph languages was implemented
- The ability to export and import from yED, to preserve the AToM^3 model appearence, was also implemented

Reachability Graph Export/Import to/from yED



Static Layout

The following tools were directly integrated into AToM^3:

- Spring-based layout
  Simulates nodes and edges to create a layout
- Snap Grid
  Removes burden of aligning node and edge control points
- Automatic edge optimizer
  Removes burden of creating straight or gently curved arrows
- Interactive edge manipulation
  Eases the creation and modification of control points
  Removes burden of manually selecting connection ports

Static Layout

Additional minor yet useful tools:

- General manipulation of multiple nodes and edges at once
- Scaling of nodes and edge drawings
- Text scaling
- Global zooming
- Arbitrary relative label placement
- Cut, copy, and paste
- Undo and redo

Spring-based Layout

- This layout approach works by modelling:
  i) Each pair of connected nodes as being tied together by an ideal spring, with a given rest length
  ii) Each pair of unconnected nodes as electrical charges and thus exerting repulsive forces on each other
  iii) A friction force to limit the effect of repulsive forces
- Highly configurable and animated in real-time
- Can be applied to selectively (to sub-graphs)
- Effective on models that have a small/sparse structure
- Disadvantages: Does not minimize edge crossings and is vulnerable to local minima solutions

Dynamic Layout

A force-transfer based layout was implemented:

- Handles the overlap resulting from the manipulation of a node or a cluster of nodes
- i.e. this occurs when using graph grammars to transform one model into another
- Can be configured to work automatically in the background or applied directly to specific nodes and even edge control points
- Handles overlap by moving nodes just enough so that they no longer overlap

Future Work

- Need more tools to handle dynamic layout
- Linear constraints could allow the smooth transformation of one model into another by specifying distances and directonal relationships between inter-connected nodes
- Spring-based constraints could do the same, and far more effectively in the case of conflicting constraints, although performance in larger models will likely prove insufficient
- A hybrid of the two approaches will yield a powerful tool for handling dynamic model layout

sourceTree Contributions