NSERC Summer Research

26 August 2004

Domain-Specific Visual Modelling

Denis Dubé

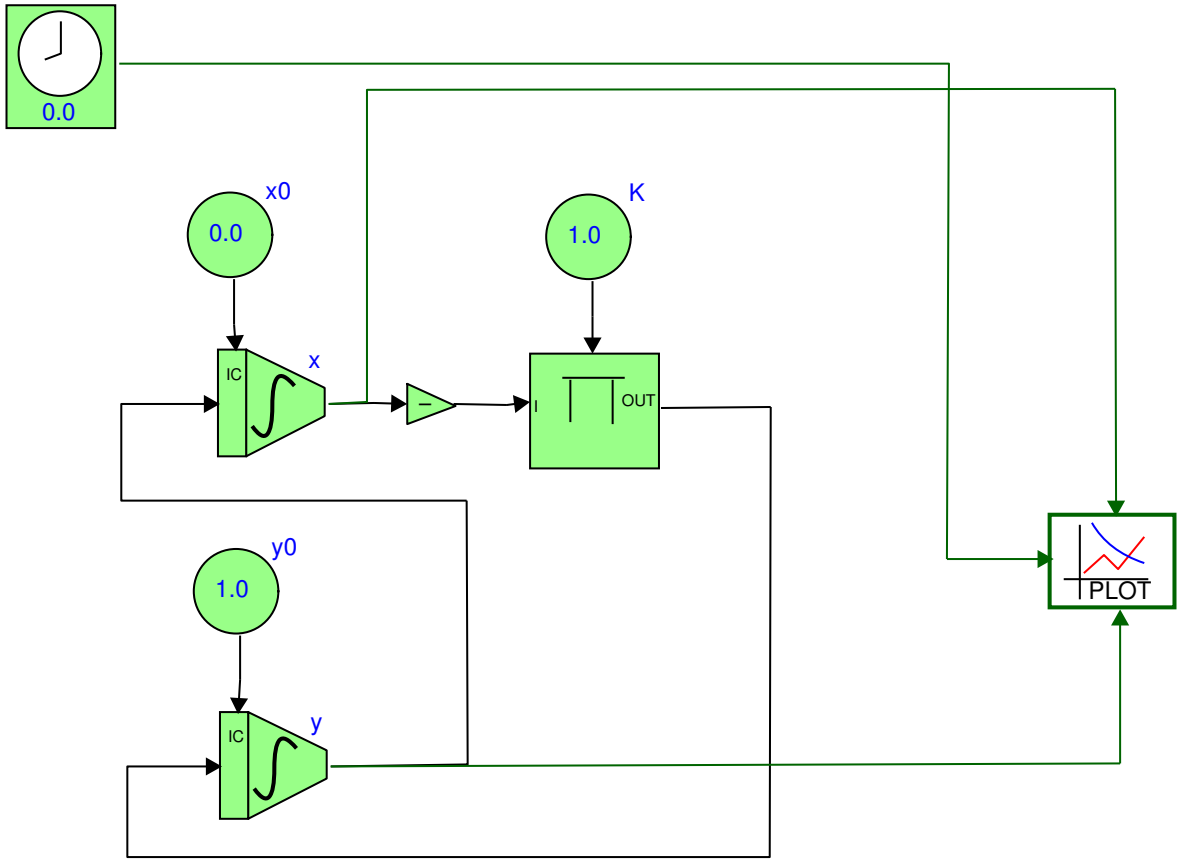
School of Computer Science, McGill University, Montréal, Canada

Modelling, Simulation & Design Lab (MSDL)

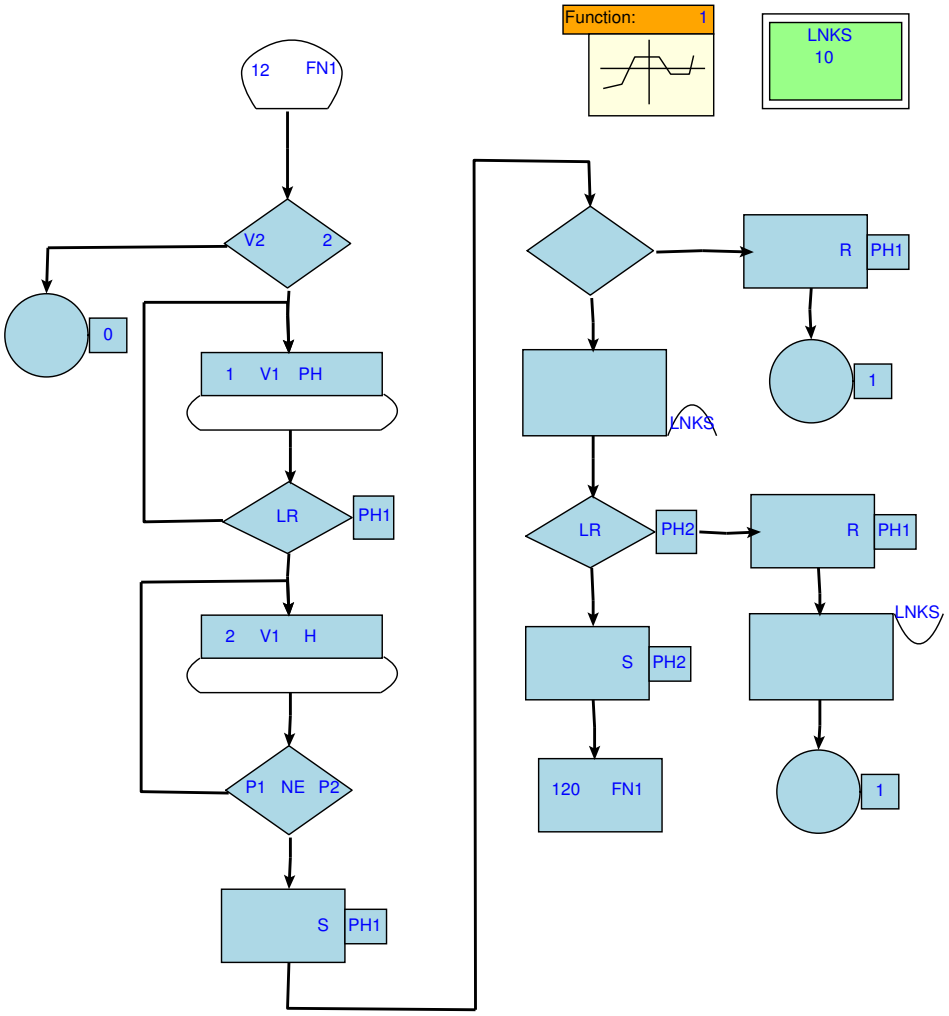
Domain-Specific Visual Modelling

- Enables working directly with domain concepts
- High level of abstraction
- Some Examples:
  DCharts, StateCharts, Petri-Nets, GPSS, Timed Automata,
  Reachability Graphs, Causal Block Diagrams

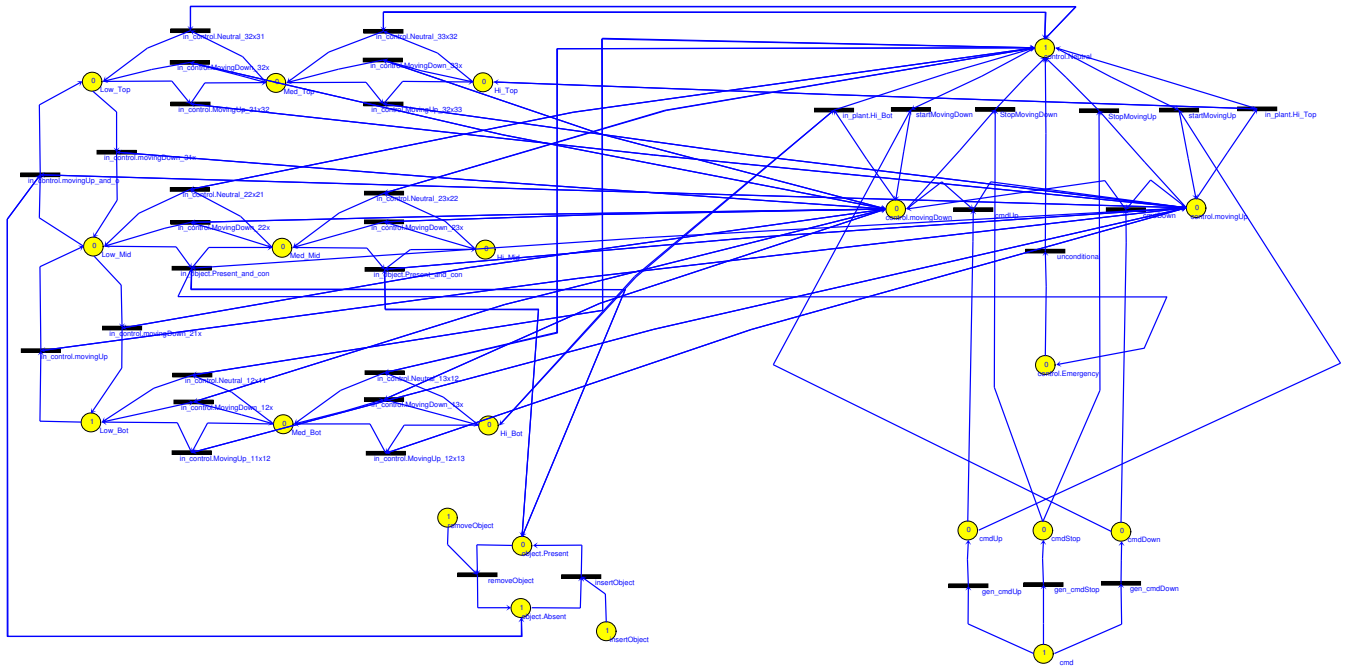
Causal Block Diagram | Harmonic Oscillator



GPSS Telephone Simulation



Petri-Nets Power Window Controller



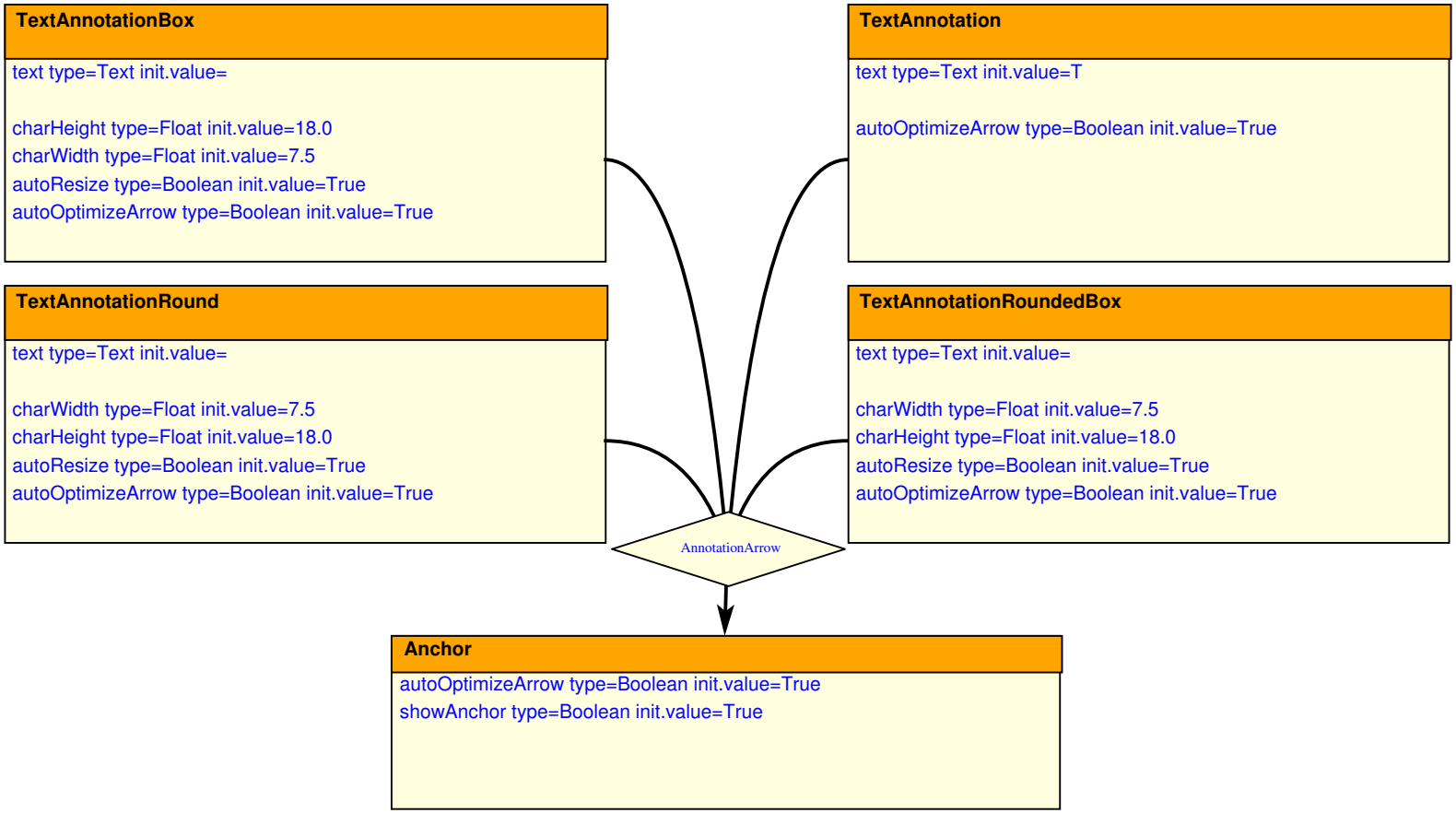
Domain-Specific Modelling Environment

- 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

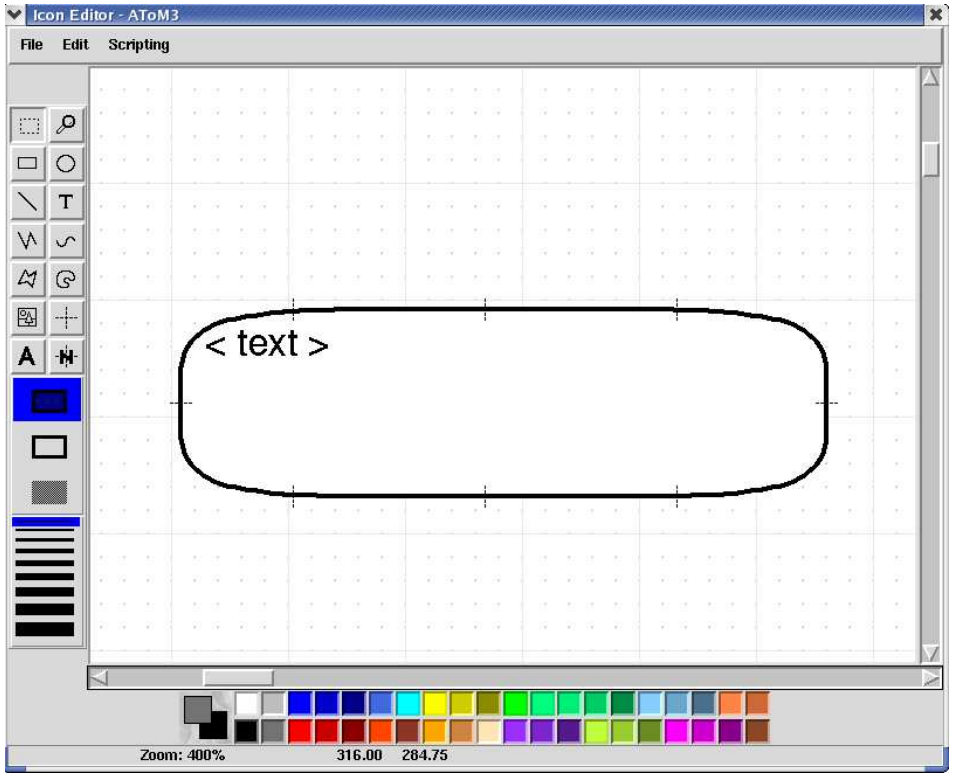
Visually Modelling The Syntax

- 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

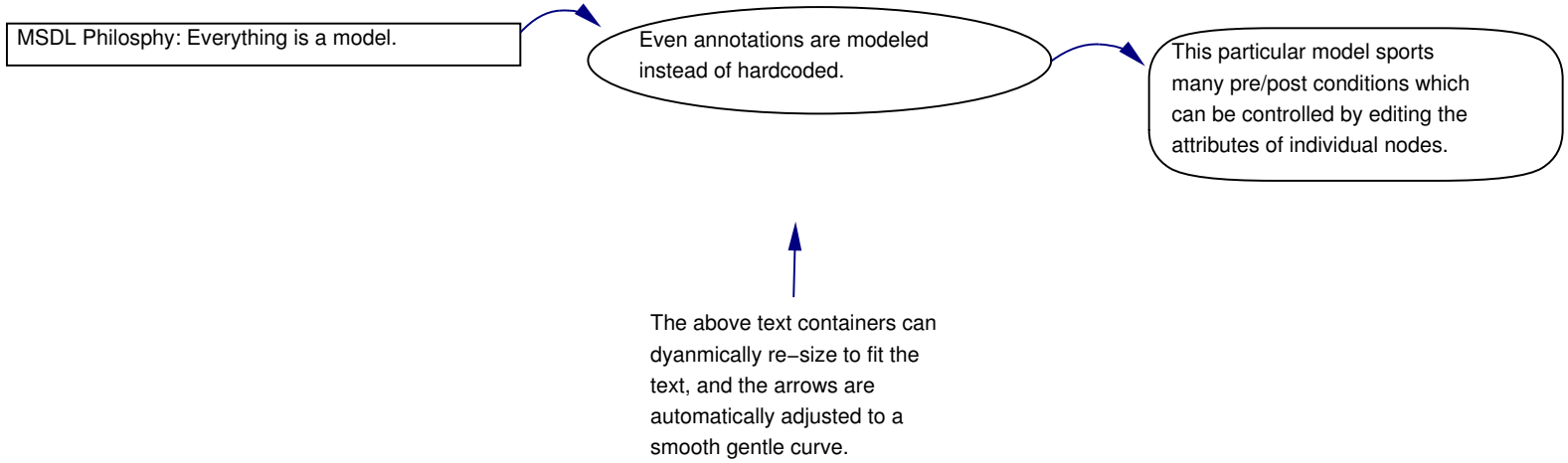


Francois Plamondon

NSERC USRA, Summer 2003

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

Annotation Model



Visual Modelling Environment

Thus far, the visual meta-modeling process is described from a static point of view

Some of the key components of an interactive visual modelling environment are:

1. Visual environment behaviour
2. Layout in static models, unchanging
3. Layout in dynamic models, undergoing graph transformations

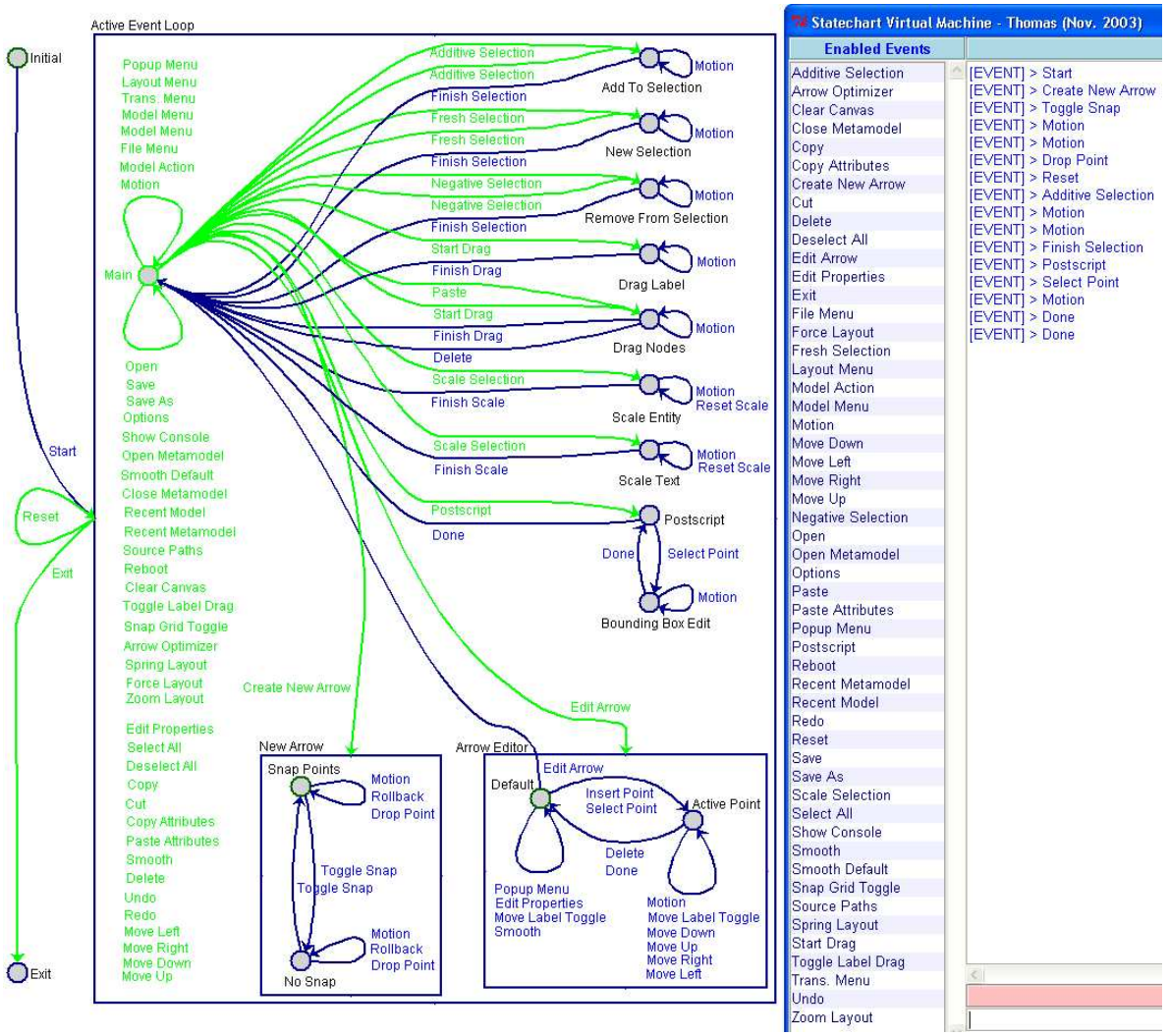
Visual Environment Behaviour

Philosphy: ”Everything is modelled explicitly”

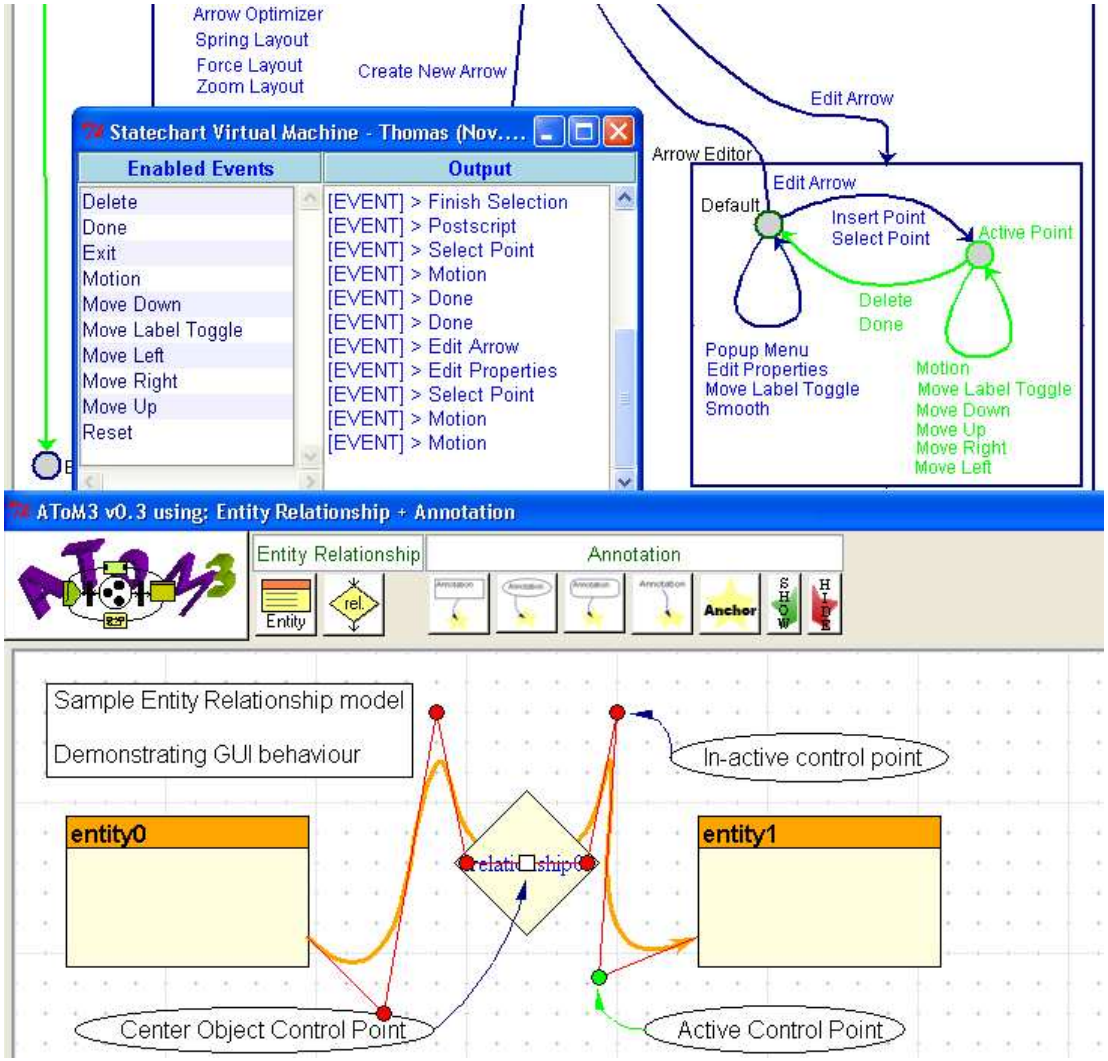
1. The behaviour was modeled as a DChart, a form of StateCharts, that is in turn a form of finite state automata
2. The model was then simulated with SVM to ensure correct behaviour
3. Python code was generated from the model using SCC

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

DCharts GUI Behaviour



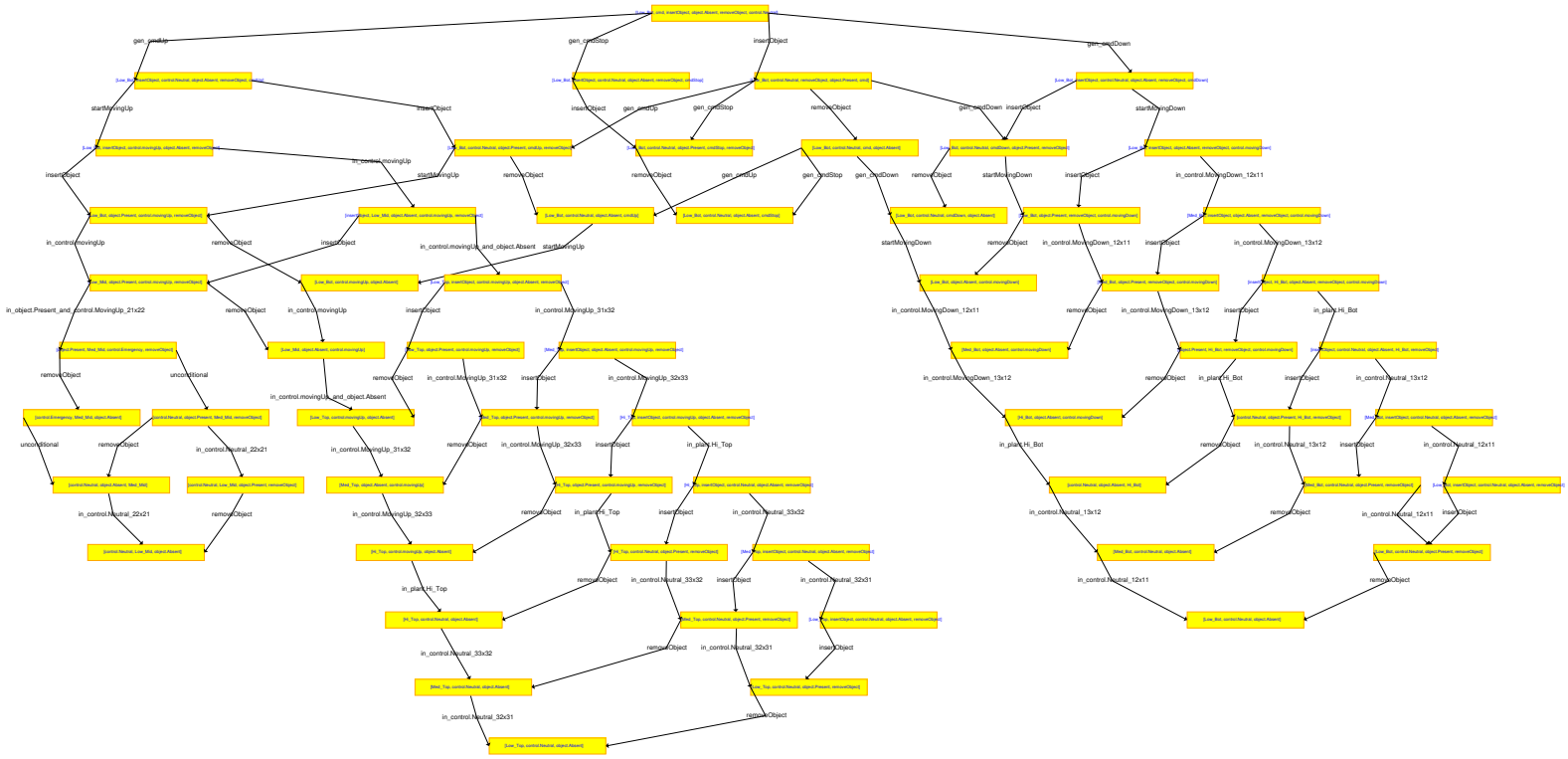
DCharts GUI Behaviour



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:

1. Each pair of connected nodes as being tied together by an ideal spring, with a given rest length
2. Each pair of unconnected nodes as electrical charges and thus exerting repulsive forces on each other
3. A friction force to limit the effect of repulsive forces

Spring-based Layout

Advantages:

- Highly configurable
- Animated in real-time
- Can be applied selectively (to sub-graphs)
- Quite effective on models that have a small/sparse structure

Disadvantages:

- $O(n^2)$ performance
- Does not minimize edge crossings
- 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
- Animated in real-time
- 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

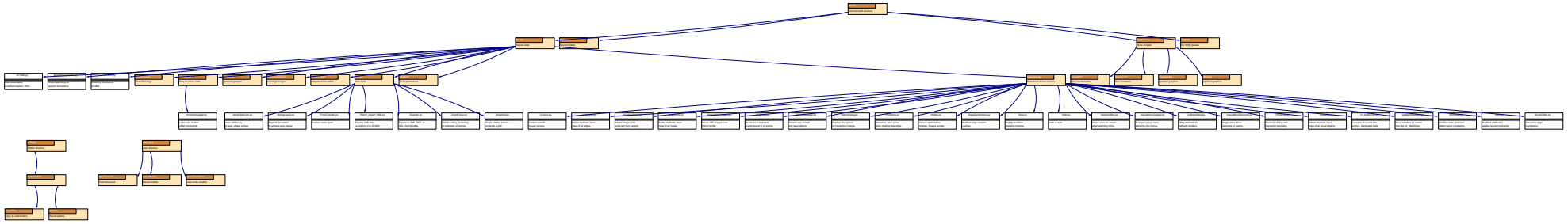
Future Work

Better tools to handle dynamic layout are needed.

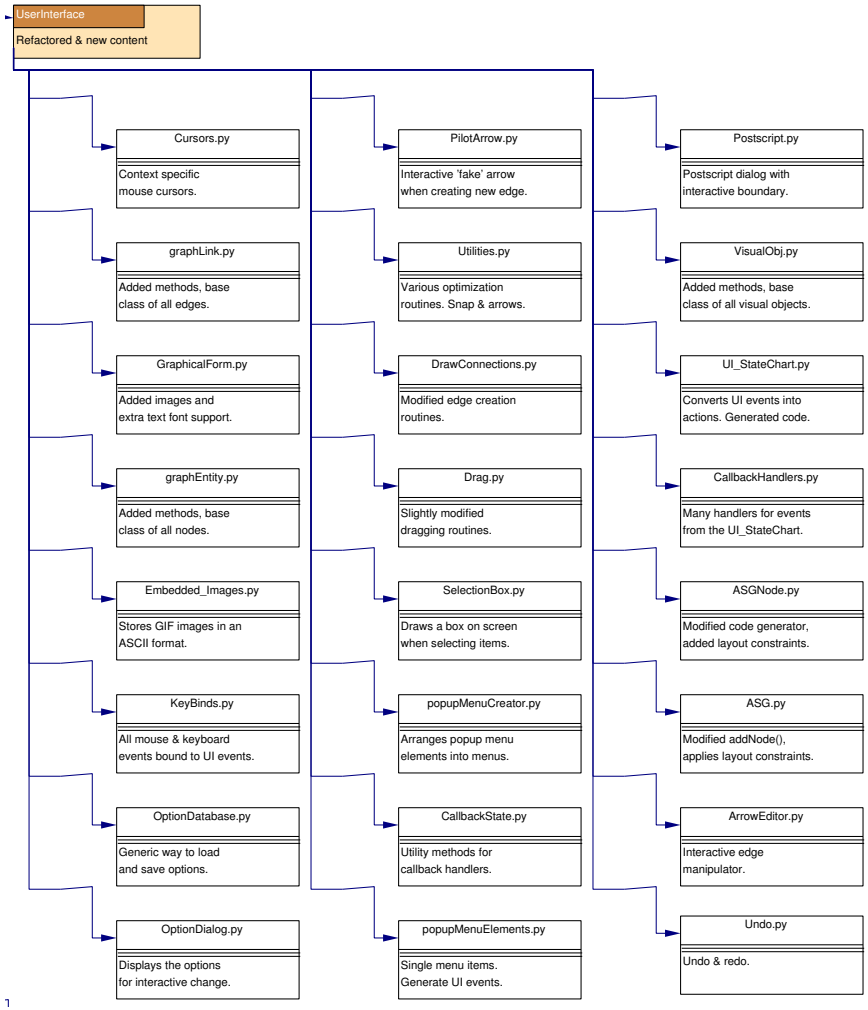
- A *linear constraints* approach would allow the specification of directional relationships and distances between nodes of a given type.
- As nodes were inserted/removed, the layout would shift to accomodate the change automatically or on demand.
- This would require layout specifications for each specific domain.

- Furthermore, a *spring-based* approach could be used to supplement the linear constraints in situations where constraints conflict.
- The springs would naturally find a compromise solution to the conflict, whereas in a pure linear constraint approach, one of the constraints must be dropped.

sourceTree Contributions Global



sourceTree Contributions GUI



sourceTree Contributions Layout