March 3, 2005

AToM3 Version 0.3
User Interface and Layout

Denis Dubé

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

Overview

- Motivation
- Explicit meta-modelling
- Icon editor
- UI behavior model
- Miscellaneous additions
- External tool layout
- Internal static model layout
- Future work

Motivation

- Increase productivity:

  Make it faster to create and manipulate models

- Increase readability:

  Make model properties more obvious by better graphical layout

- Decrease errors:

  Make the models and documentation produced more reliable

Explicit Meta-modelling

- 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

Explicit Meta-modelling

- 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 behavior

Explicit Meta-modelling

- The highest level Meta-model is the Entity Relationship model
- In theory:

  An Entity Relationship model can generate the meta-model Entity Relationship used to create the model in the first place (bootstrapping)

- In practice:

  This capability was just recently added to AToM^3 version 0.3

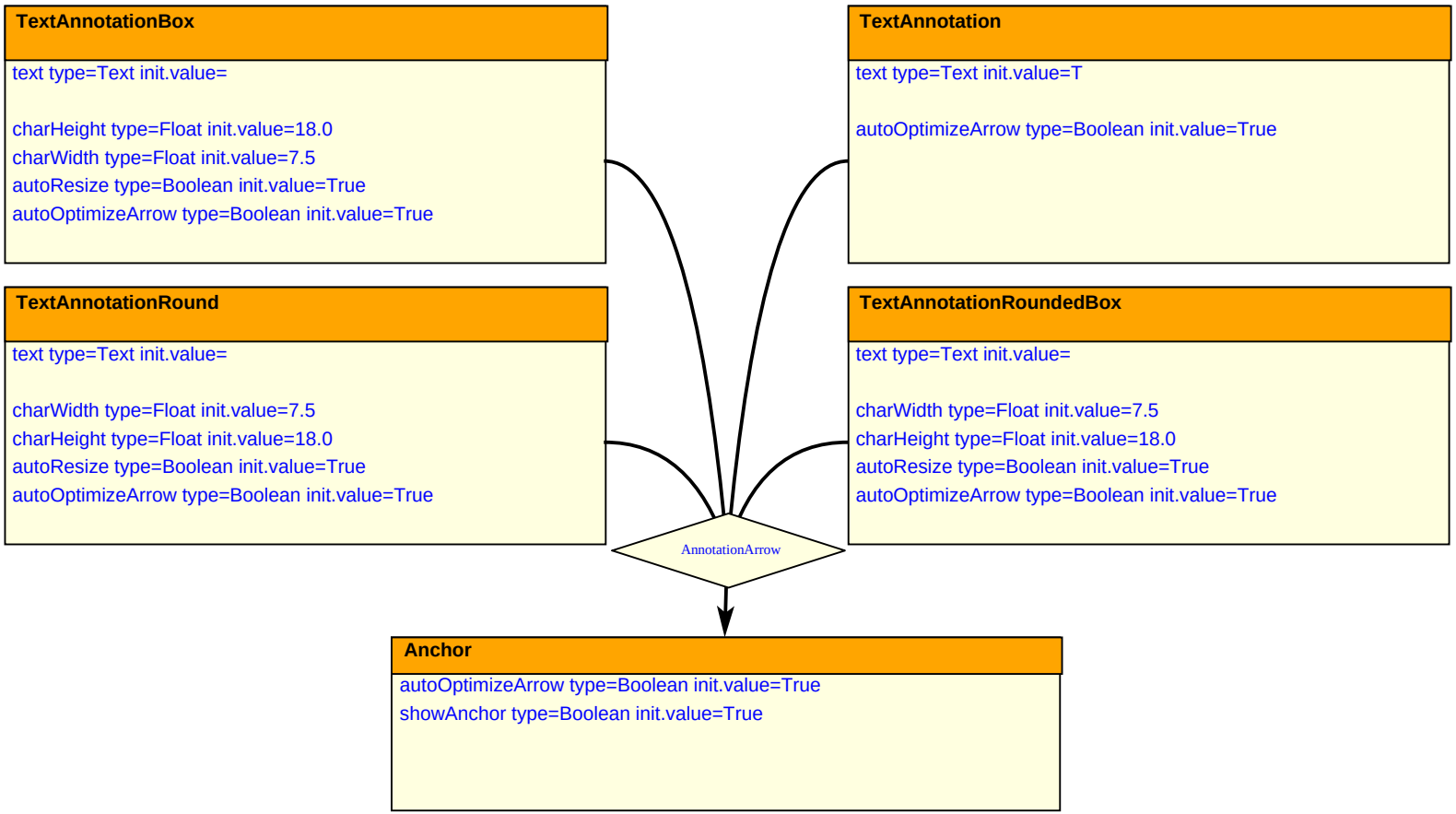
- Motivation:

  Makes it possible to easily extend the Entity Relationship model

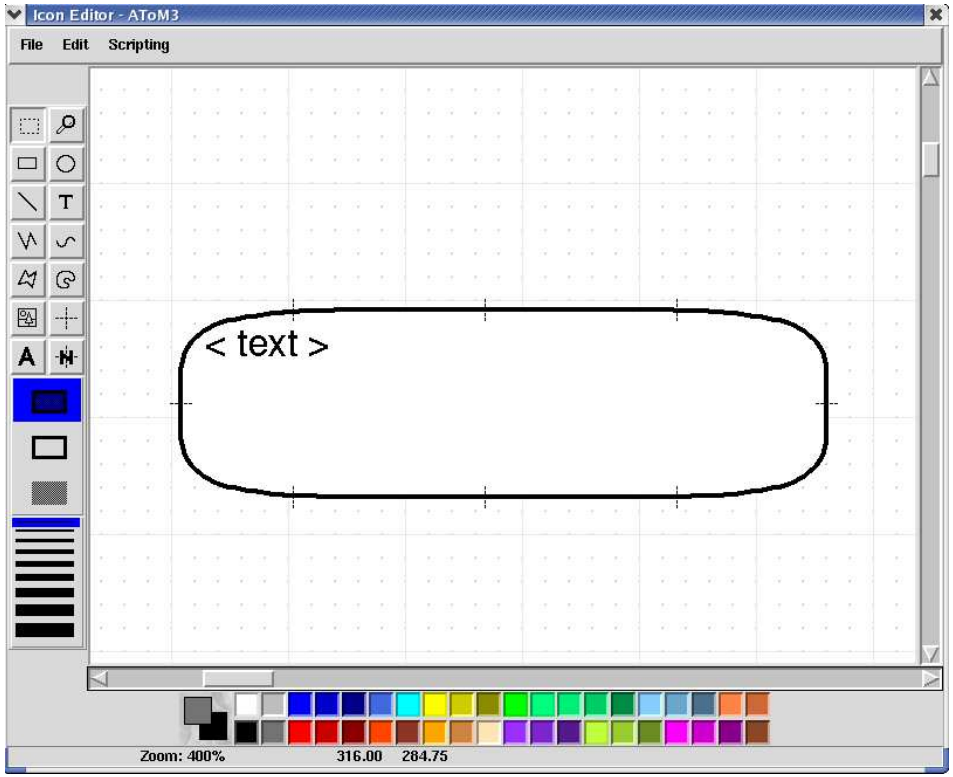
Overview

- Motivation
- Explicit meta-modelling
- **Icon editor**
- UI behavior model
- Miscellaneous additions
- External tool layout
- Internal static model layout
- Future work

Annotation Meta-Model



Icon-Editor

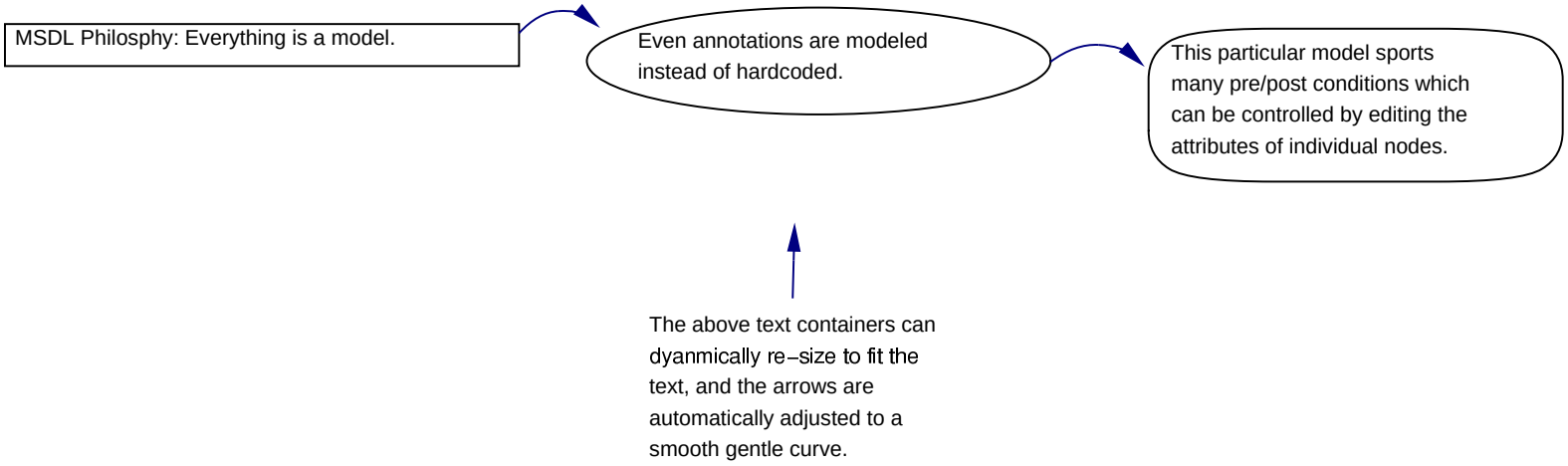


Francois Plamondon

NSERC USRA, Summer 2003

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

Annotation Model



Overview

- Motivation
- Explicit meta-modelling
- Icon editor
- **UI behavior model**
- Miscellaneous additions
- External tool layout
- Internal static model layout
- Future work

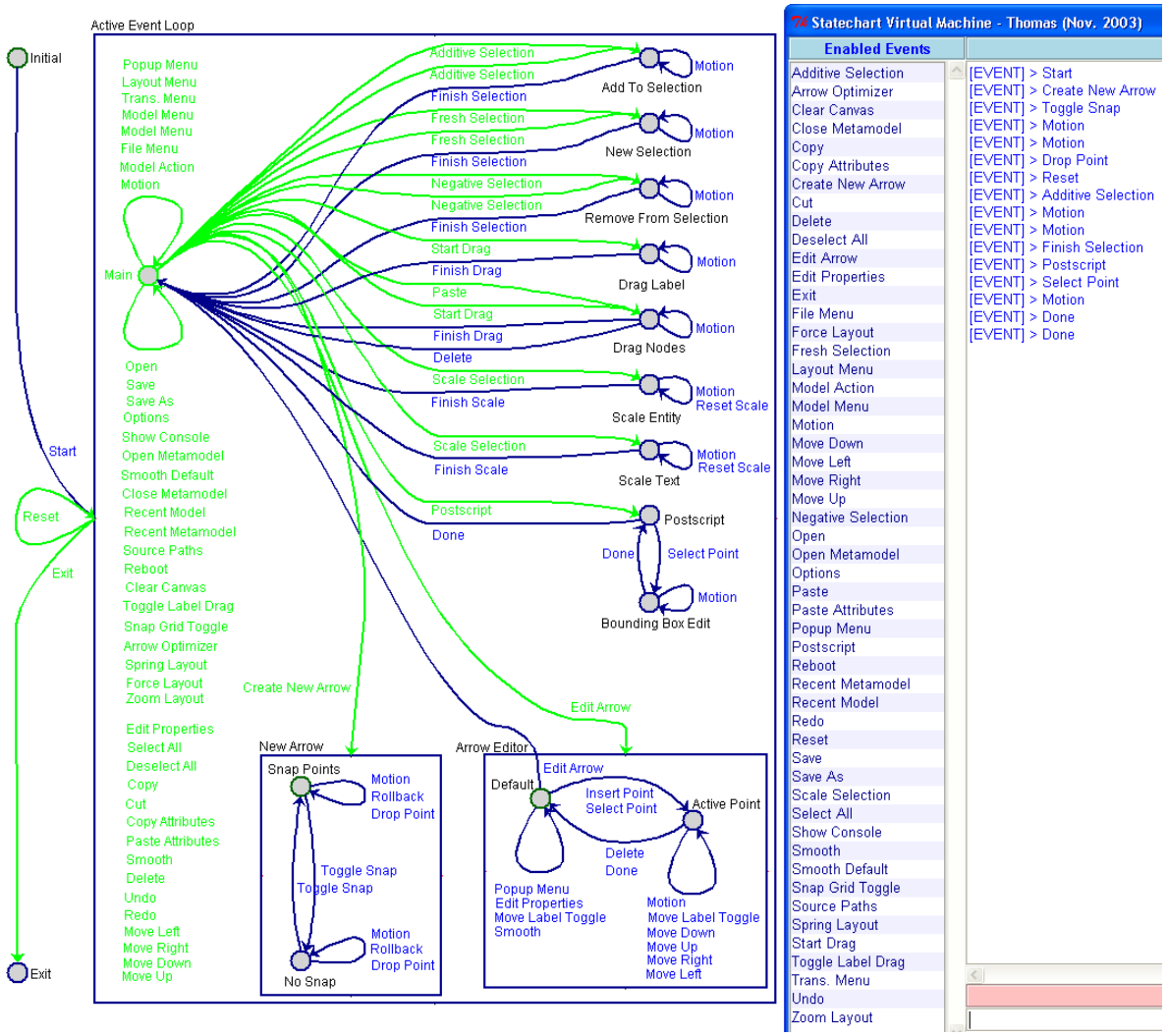
User Interface Behavior Model

Philosophy: ”Everything is modelled explicitly”

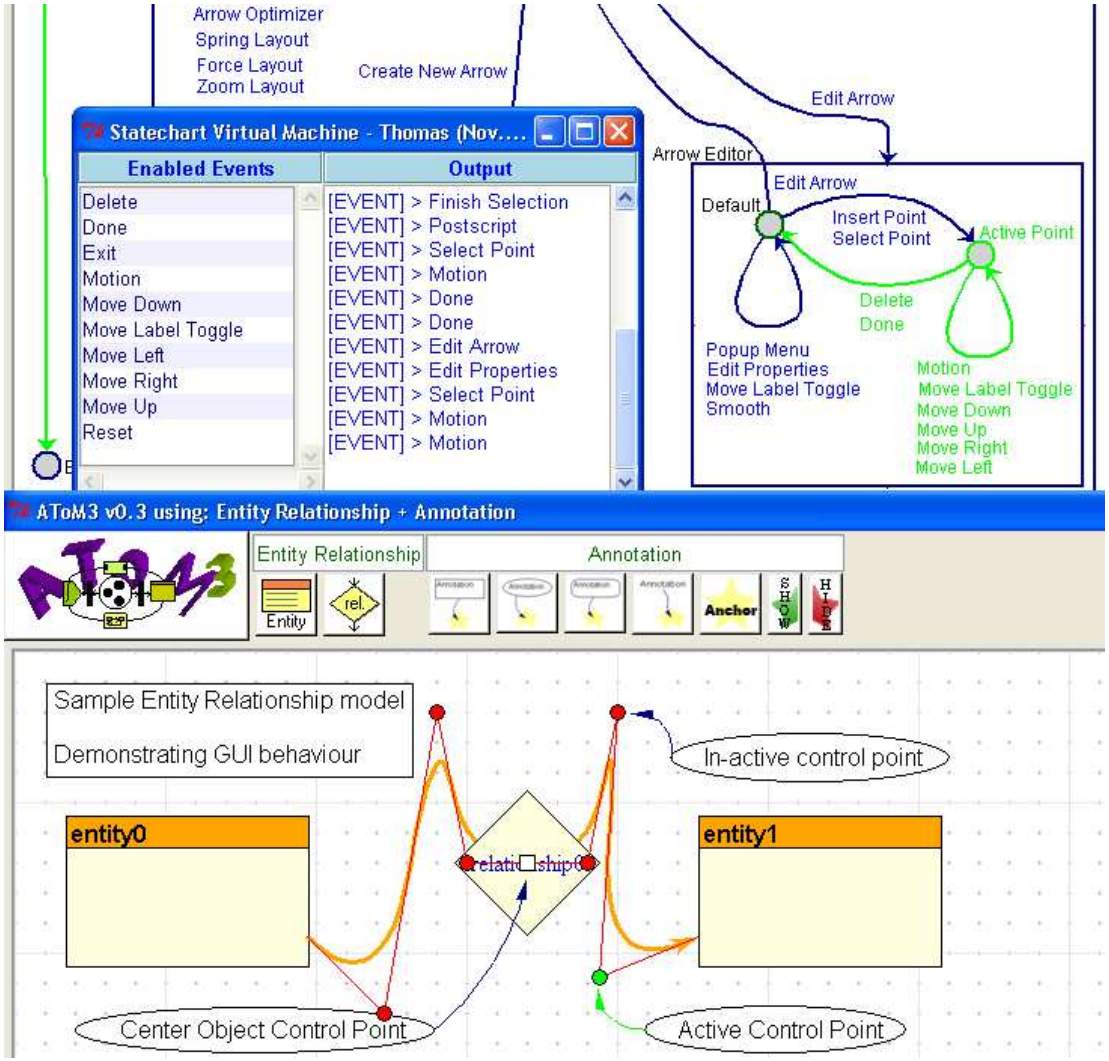
1. The behavior was modelled as a DChart, a form of StateCharts, that is in turn a form of finite state automata
2. The model was then simulated with SVM to ensure correct behavior
3. Python code was generated from the model using SCC

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

DCharts GUI Behavior



DCharts GUI Behavior



User Interface Behavior Model

Recent Addition: User-defined behavior model

1. User modifies the ’Central’ user interface statechart model
2. User generates a DES file from the model and then generates Python code using SCC (the custom version found in the Central Model Directory)
3. User opens AToM^3 options and specifies which file to use for the UI behavior

Future Work: A code generating keybinds to events model?

Overview

- Motivation
- Explicit meta-modelling
- Icon editor
- UI behavior model
- **Miscellaneous additions**
- External tool layout
- Internal static model layout
- Future work

Miscellaneous additions

- Graph grammar, automatic documentation
- Uncaught exception handling

  GUI warning

  Log file and support ticket
- Context sensitive popup menus
- Generalized option database

Miscellaneous additions

- 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 (nodes, links, and attributes)
- Undo and redo

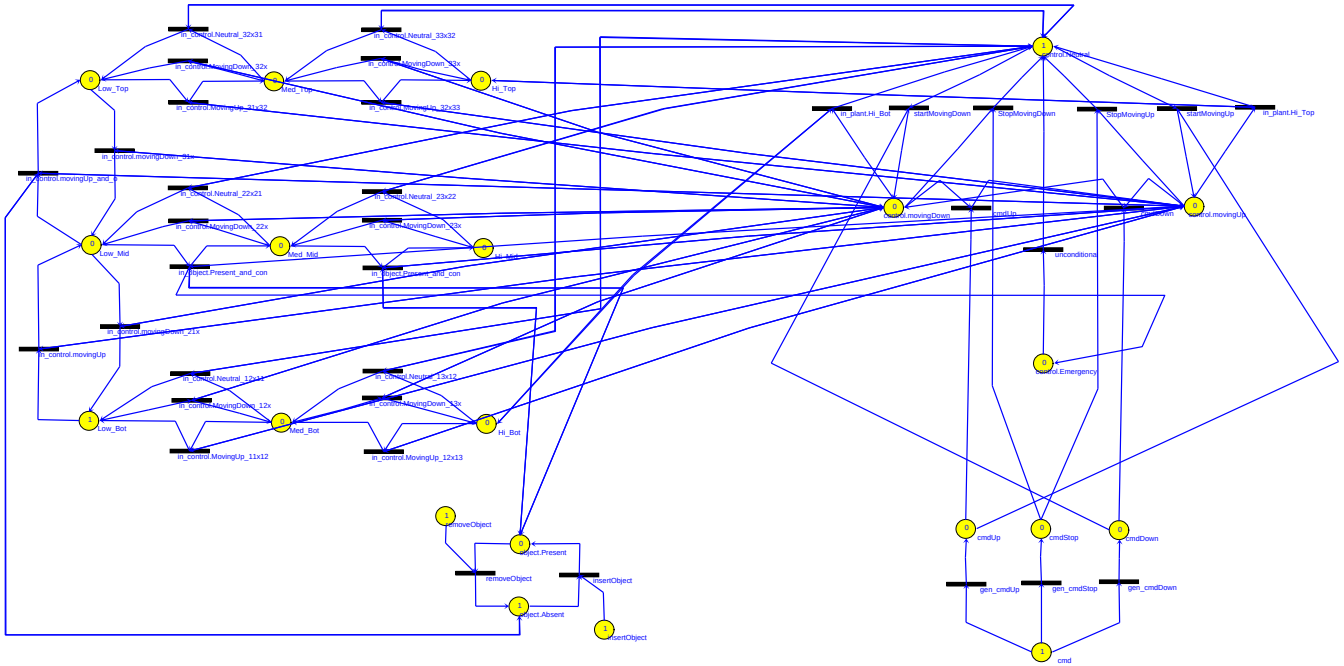
Overview

- Motivation
- Explicit meta-modelling
- Icon editor
- UI behavior model
- Miscellaneous additions
- **External tool layout**
- Internal static model layout
- Future work

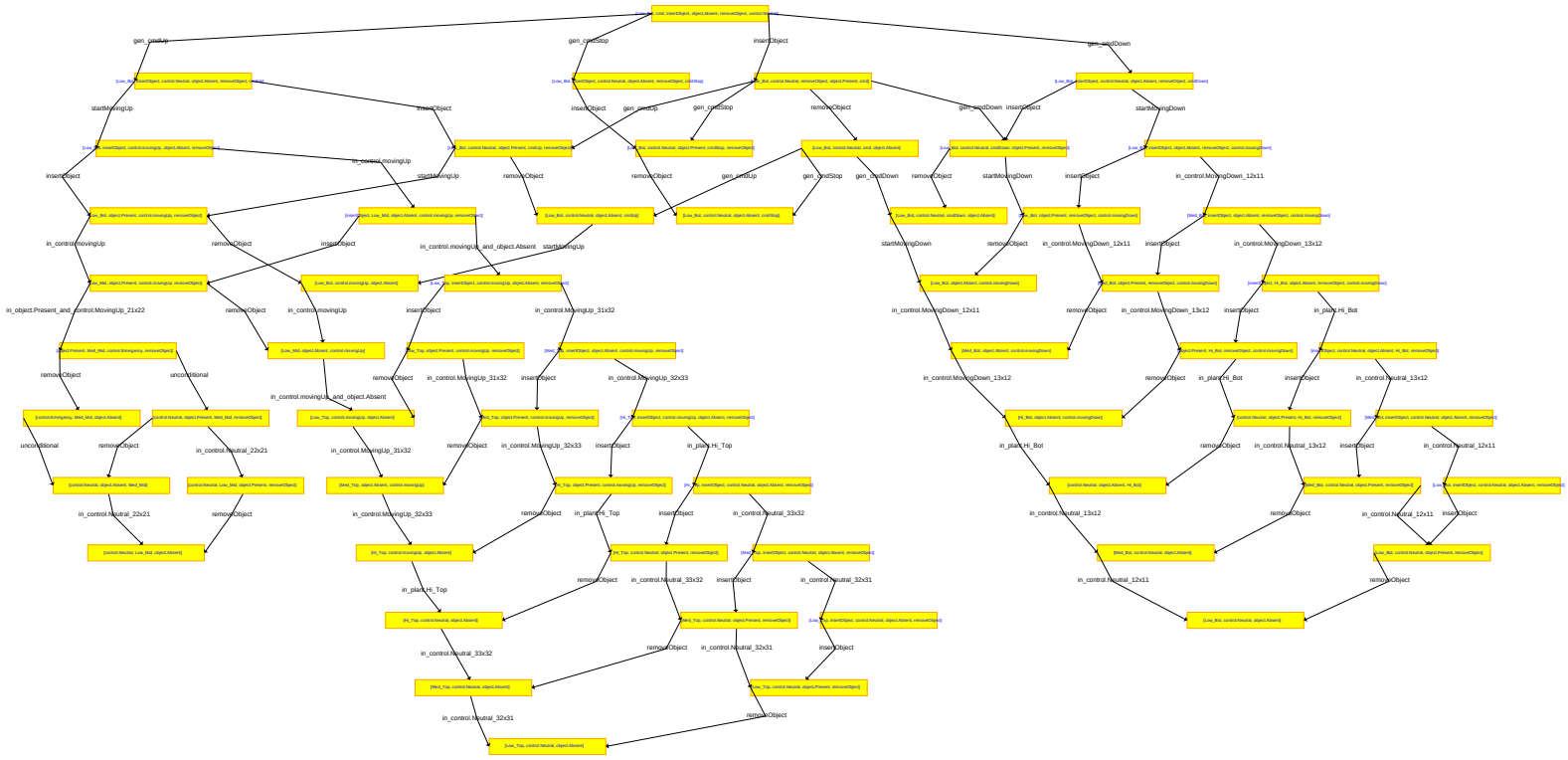
External tool layout

- An extensive review of the existing literature 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 appearance, was also implemented

Petri-Nets Power Window Controller



Reachability Graph Export/Import to/from yED



Overview

- Motivation
- Explicit meta-modelling
- Icon editor
- UI behavior model
- Miscellaneous additions
- External tool layout
- **Internal static model layout**
- Future work

Internal static model 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

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 (such as in graph grammar transformations) are needed!
- A *linear constraints* approach is being considered where in each formalism, graphical layout constraints would be explicitly defined for each syntax relationship

Future Work

Current Progress:

- Evaluated the DiaGEN approach, which use the QOCA constraint handling package, and found it rather lacking for non-trivial visual formalisms
- Currently evaluating the GenGED approach, which uses the PARCON constraint handling package