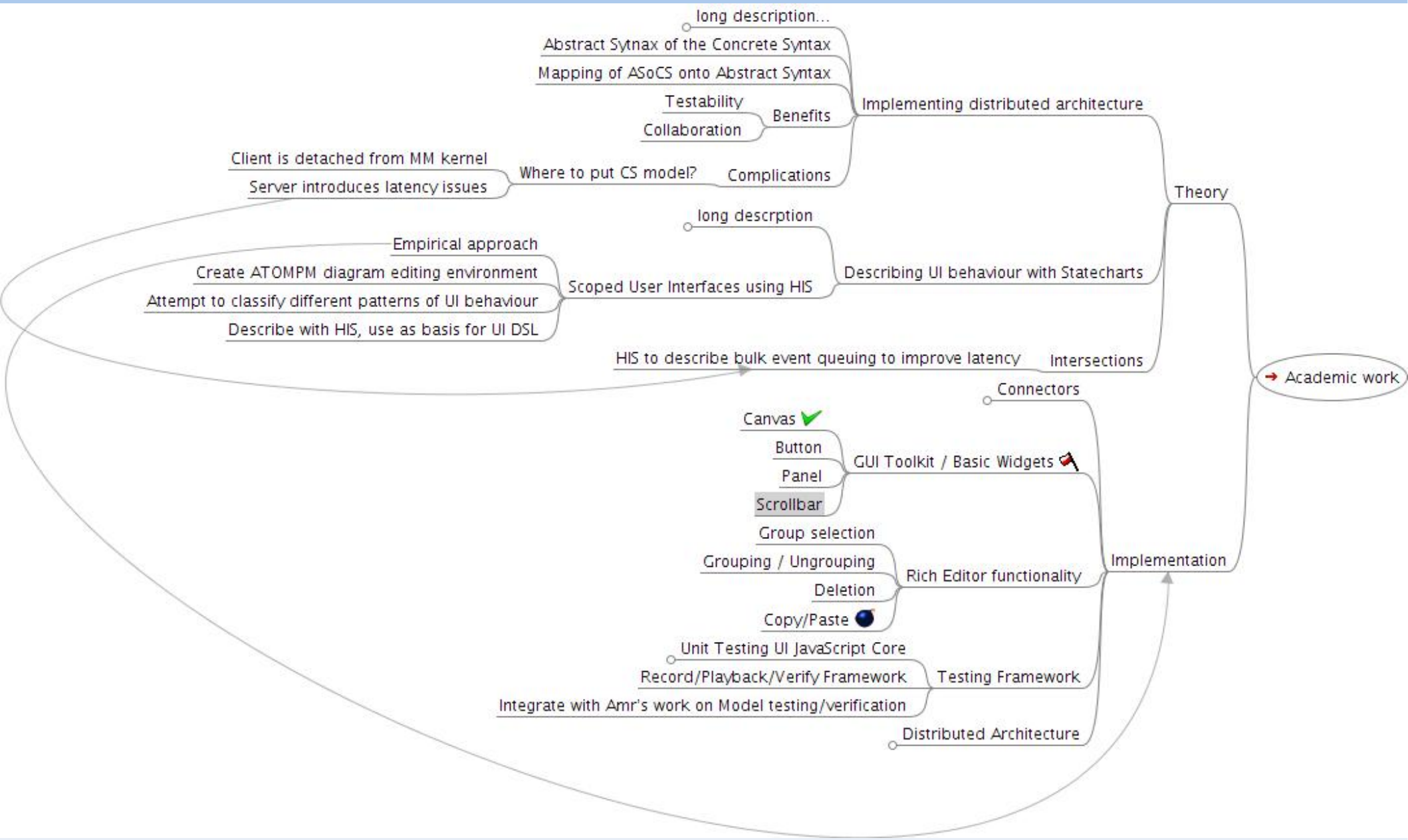
Modelling the Reactive Behaviour of Scoped User Interfaces with Hierarchically-linked Statecharts

Jacob Beard
McGill University Modelling, Simulation and Design Lab (MSDL)
08/27/2009

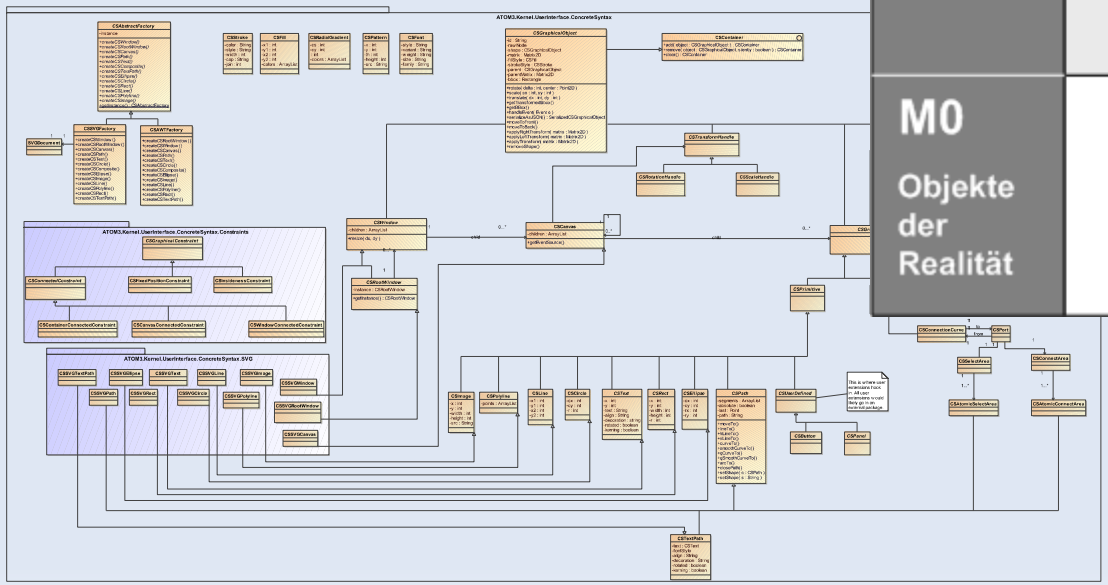
Challenges With UI Development

- Different kinds of desired behaviour:
  - autonomous
  - reactive
  - possibly real-time
- complex behavioural relationships
- difficult to minimize "accidental complexity"
- conventional code-centric approaches fall short

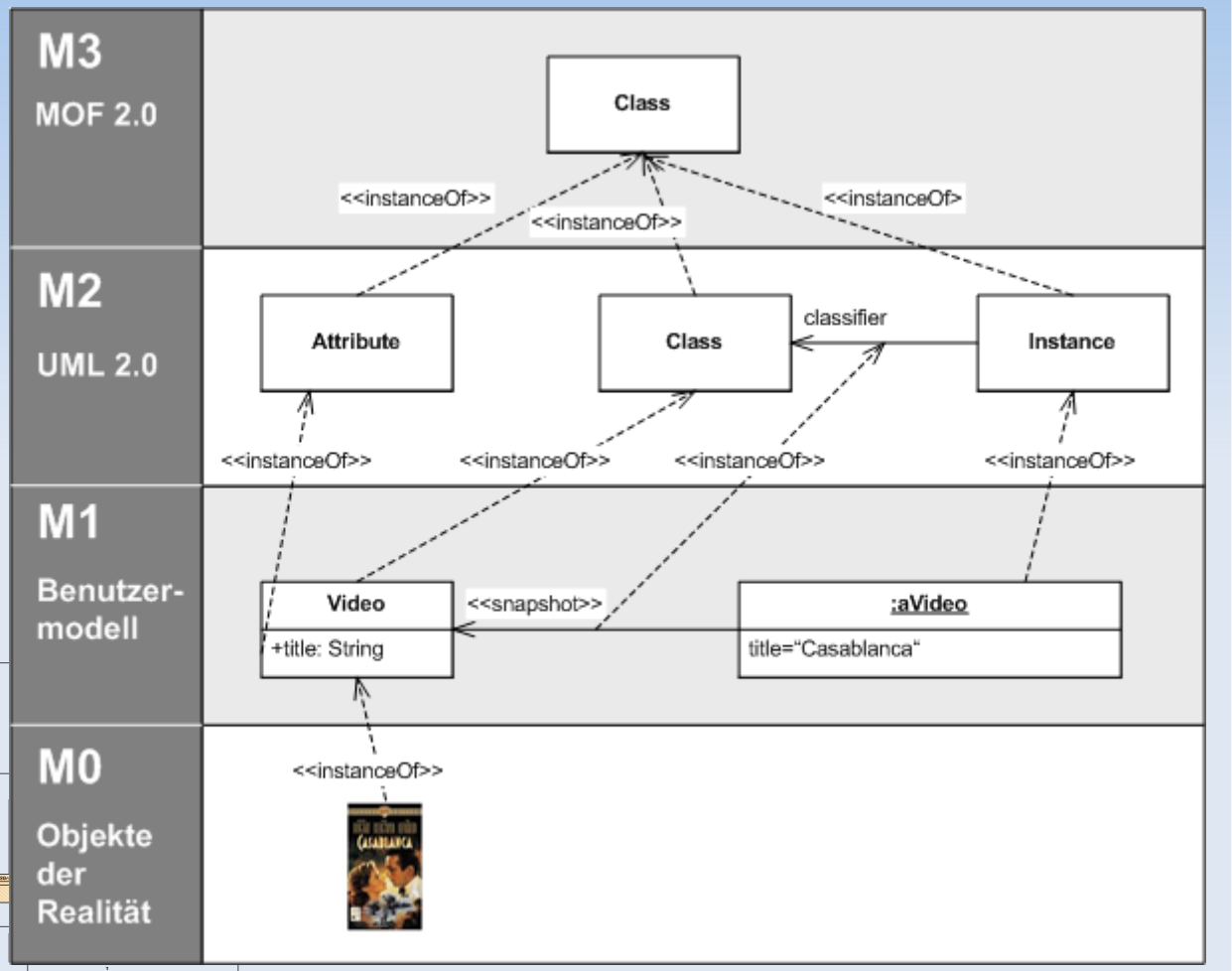
Big Picture


Academic work
Theory
Implementing distributed architecture
long description...
Abstract Sytnax of the Concrete Syntax
Mapping of ASoCS onto Abstract Syntax
Benefits
Testability
Collaboration
Complications
Where to put CS model?
Client is detached from MM kernel
Server introduces latency issues
Describing UI behaviour with Statecharts
long descrption
Scoped User Interfaces using HIS
Empirical approach
Create ATOMPM diagram editing environment
Attempt to classify different patterns of UI behaviour
Describe with HIS, use as basis for UI DSL
Intersections
HIS to describe bulk event queuing to improve latency
Implementation
Connectors
GUI Toolkit / Basic Widgets
Canvas
Button
Panel
Scrollbar
Rich Editor functionality
Group selection
Grouping / Ungrouping
Deletion
Copy/Paste
Testing Framework
Unit Testing UI JavaScript Core
Record/Playback/Verify Framework
Integrate with Amr's work on Model testing/verification
Distributed Architecture

Meta-Modelling the UI

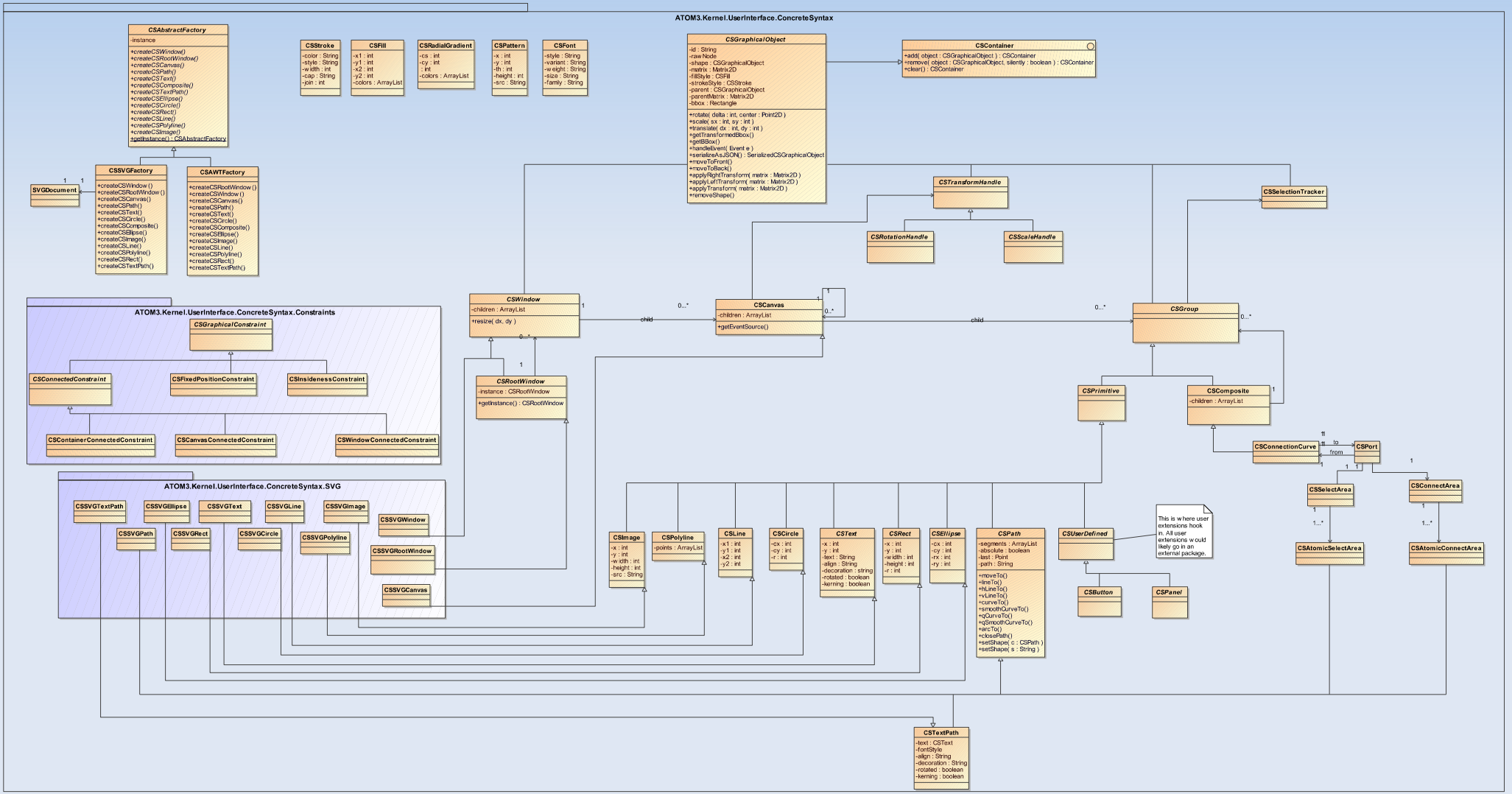


Abstract Syntax of the Concrete Syntax

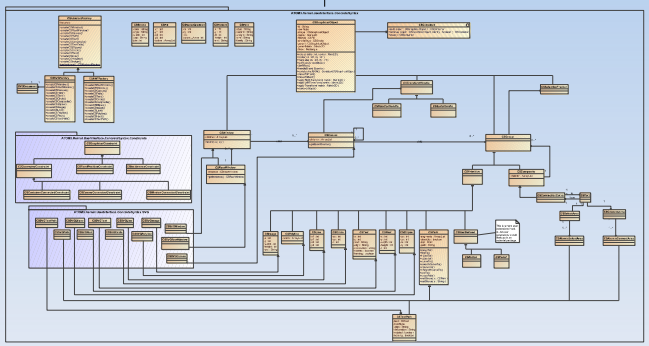


Source: http://en.wikipedia.org/wiki/File:M0-m3.png

Abstract Syntax of the Concrete Syntax



Meta-Modelling the UI (M1)



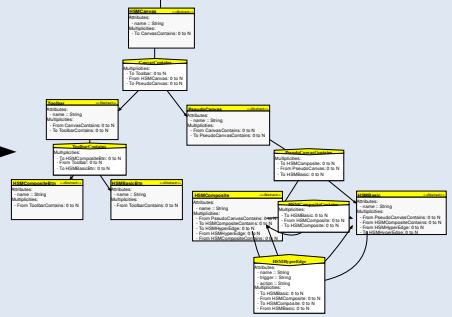
Abstract Syntax of Concrete Syntax

Mapping Metamodel

UML Metamodel

Formalism-specific Concrete Syntax Definition

Mapping onto the Abstract Syntax model

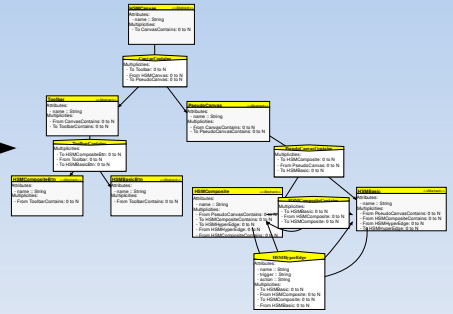


Abstract Syntax (Domain Model)

Meta-Modelling the UI (M0)

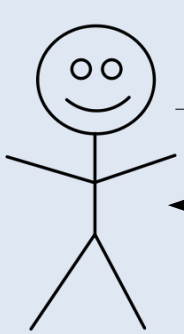
Formalism-specific Concrete Syntax Definition

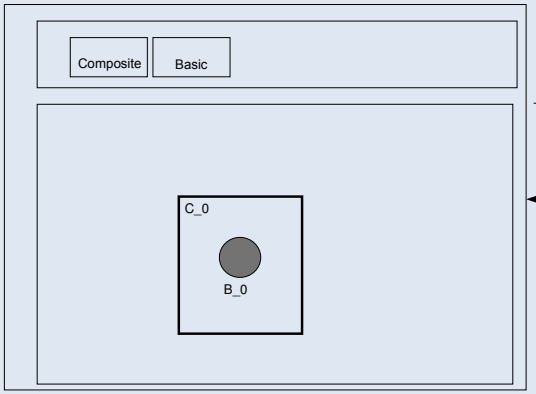
Mapping onto the Abstract Syntax model

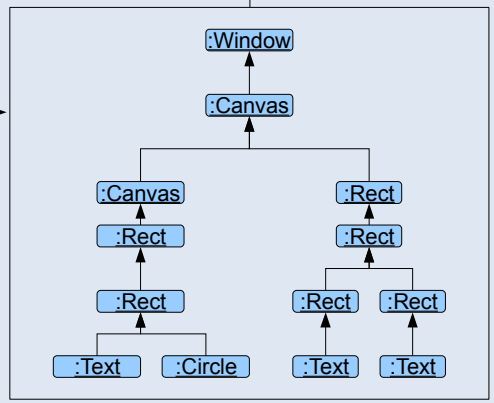


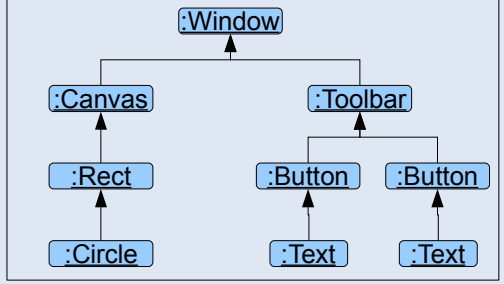
Abstract Syntax (Domain Model)

Two-Way Communication Between each Layer









User

Final UI

Concrete Syntax Model Instance

Abstract Syntax Model Instance

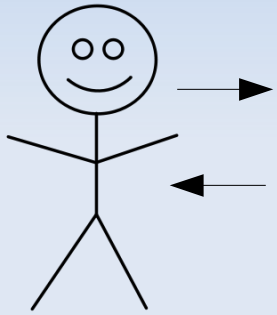
Model Instances as Web Services

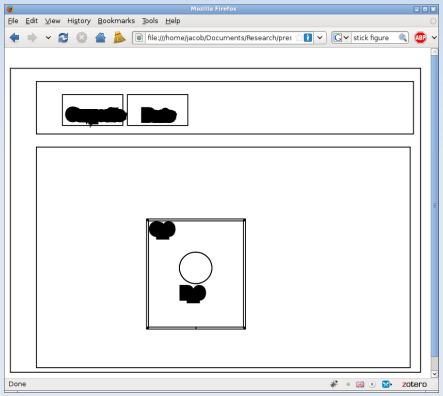
Client

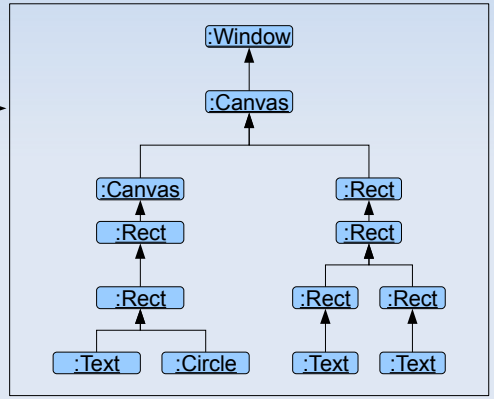
Server

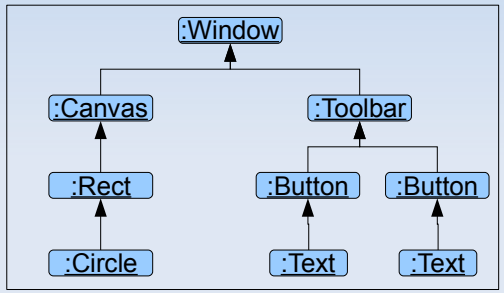
Server

XMLHTTPRequest









User

Final UI
(Browser-based)

Concrete Syntax
Model Instance

Abstract Syntax
Model Instance

Statecharts

- Use Statecharts to define UI behaviour
- In UML2, each Class may be associated with a statechart
- We define default UI behaviour in Abstract Syntax of Concrete Syntax
- Behaviour can be refined through class inheritance for domain-specific Concrete Syntax entities

Hierarchically-linked Statecharts

- Hierarchically-linked Statecharts (HIS) is a formalism for visually describing the structure and behaviour of Scoped UI's based on a combination of UML Class Diagrams and Statecharts.
- Class Diagrams used to describe permissible relationships between UI components.
  - Allows syntax-directed editing, conformance checks
- Statecharts used to encode reactive behaviour of individual visual entities.
- Objects are actors which encapsulate state and behaviour.
  - Communicate via message-passing interfaces.

HIS and Scoped UI

- Scoped User Interface:
  - reactive visual components (widgets) are hierarchically nested
  - at the highest level of the hierarchy, widgets exhibit general behaviour
  - deeper in the hierarchy, widgets have more specific behaviour
- Many real-world applications

HIS Workflow

- Language-engineering:
  - Model the Abstract Syntax using UML CD
  - Model the Concrete Visual Syntax using AsoCS
  - Specify mapping between AS and CS
  - Specify reactive behaviour using Statecharts
- Captures structure, behaviour and appearance of a visual language.
- Sufficient to allow the automatic synthesis of a language-specific modelling environment.

Benefits of Domain-Specific Modelling

- Domain- and formalism-specific modelling have the potential to greatly improve productivity:
  - match the user's mental model of the problem domain
  - maximally constrain the user (through the checking of domain constraints)
  - separate the domain-expert's work from analysis and transformation expert's work
  - are able to exploit features inherent to a specific domain or formalism.

Additional Benefits

- Testing
  - At each layer, events can be recorded, later on played back within testing harness.
  - Allows headless automated testing of UI.
  - Each layer can be simulated, thus automated.
- Collaboration
  - Centralized server = easy sync
  - Easy to hook multiple terminals to the same graphics server: many-to-one relationship

Implementation

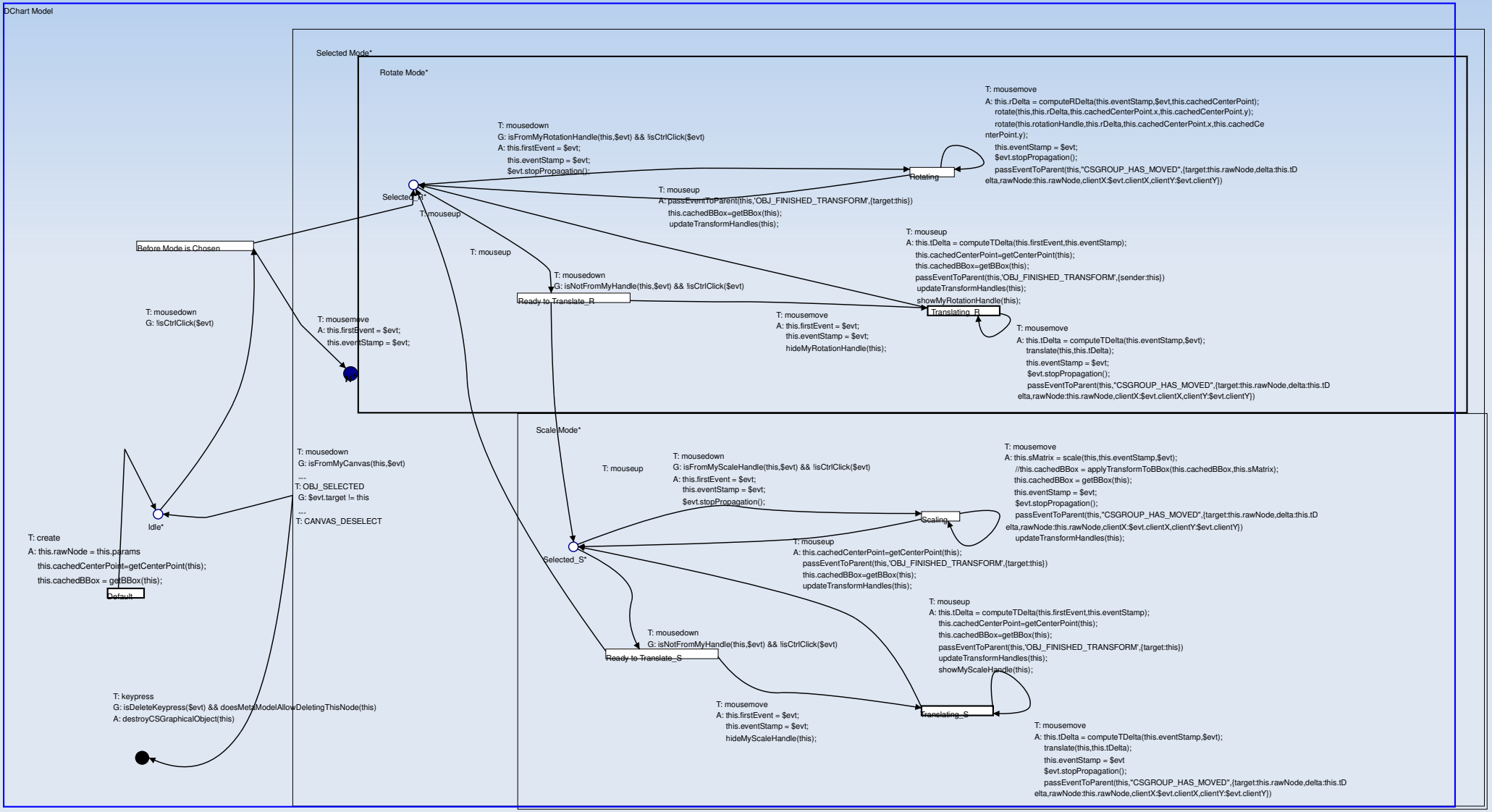
Current Status

- Have not had a compiler able to handle both Class Diagrams and Statecharts
- I have used
  - AToM3 DchartsV3 to model Statecharts
  - SCCJS to compile them to JavaScript constructor function artifacts
  - Annotated SVG for specifying Concrete Syntax
  - Handwritten “glue code”
- Raphael's plugin generates Abstract Syntax constraints from AToM3 Class Diagrams
  - Enables syntax-directed editing

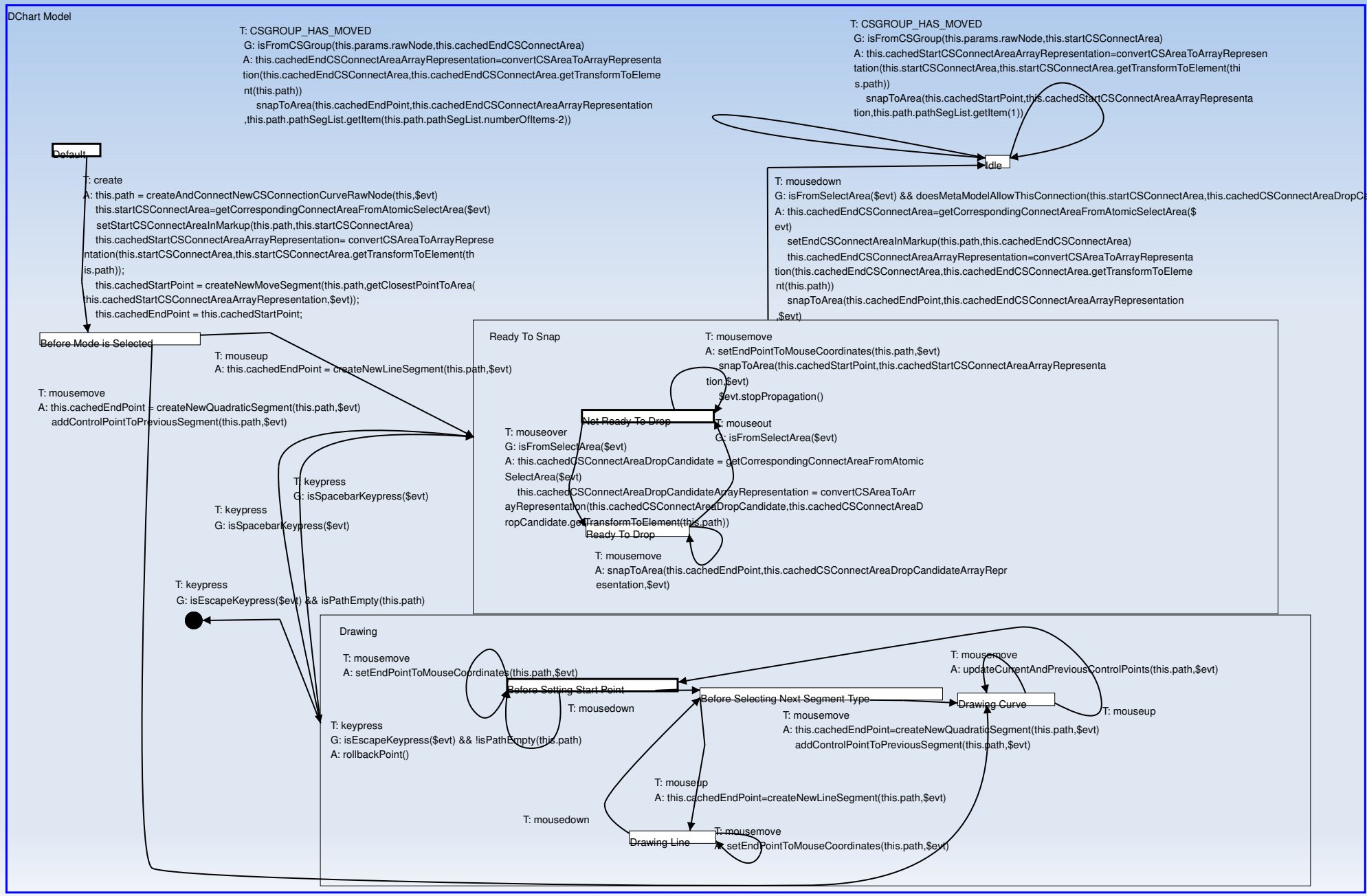
Demo

- Hosted live here
- UI behaviour based on inkscape:
  - Translate, rotate, scale
  - Drawing curves

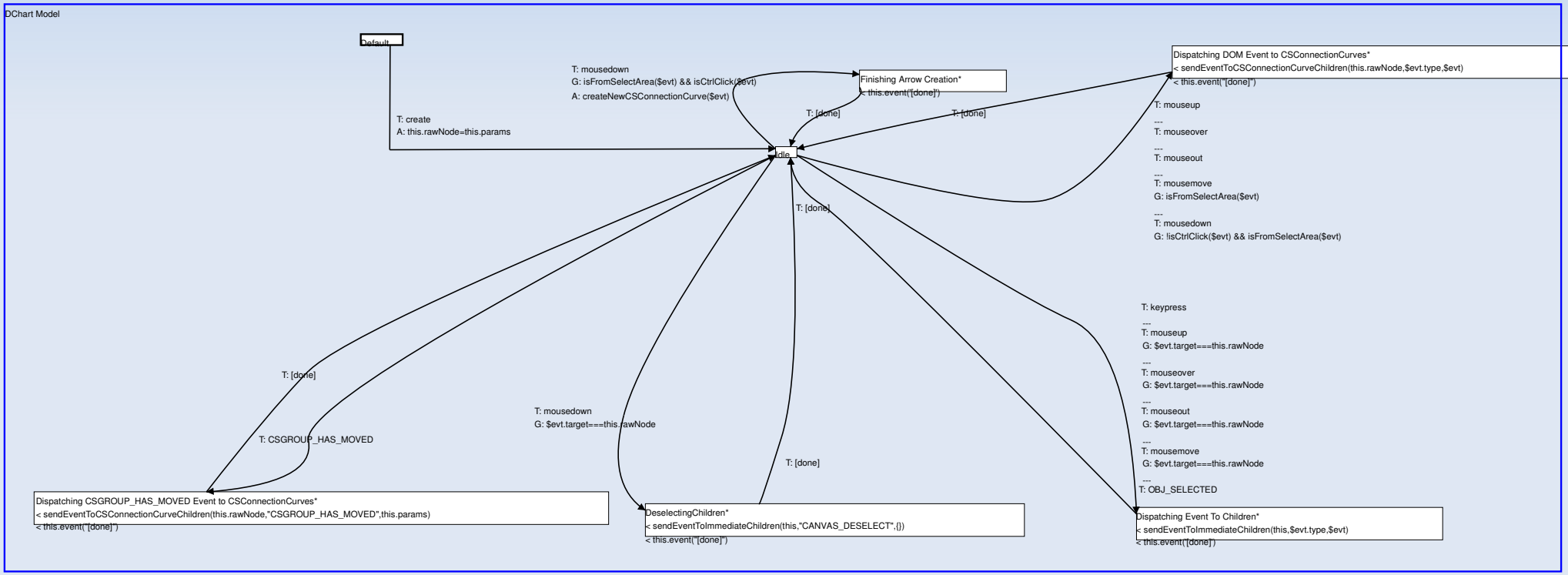
CSGroup Behaviour



CSConnectionCurve Behaviour



CSCanvas Behaviour



Future Work

- Ultimately: Web-based Statechart and Class diagram editor
  - Plug into compiler on the back end
  - Allow development of rich web-based UI
- SVG Open conference in October
  - Feasible to create a very good web-based statechart editor implementation.

Thank you for attending!