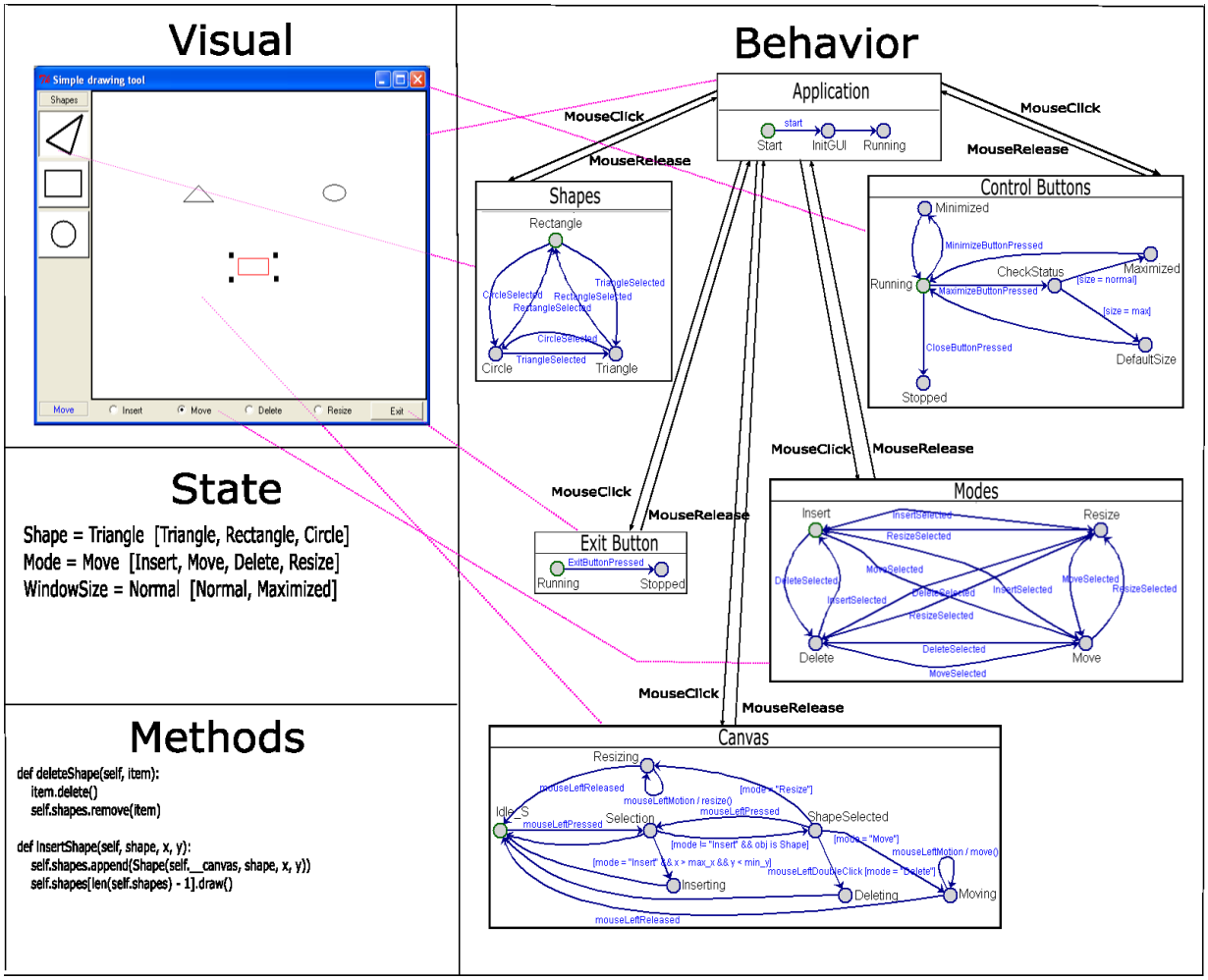
User Interfaces by Domain-Specific Visual Modelling

Interactive behavior is best modeled with Statecharts. But there is still a need for a formalism that encapsulates both the appearance and the behavior of GUIs' applications. Therefore, we are proposing XXXX.

XXXX is a Domain-Specific formalism for constructing GUIs. As shown below, the formalism consists of four parts: Visual, Behavior, States, and Methods.



1. The *Visual* part is where users can specify GUIs layout. This can be built using either pre-defined GUIs or customized GUIs. A drawing tool is available to assist users in drawing their customized GUIs.
2. The *Behavior* part is based on a hierarchy of communicating DCharts. When the user uses a pre-defined GUI, a DChart describing its default behavior will be loaded automatically. The user can later modify this behavior, if they wish. On the other hand, the behavior of the cusomized GUIs needs to be specified by the user. The top-level DChart is the one that

describes the behavior of the application's main window. When a model is simulated, the system first executes that top-level DChart. Since the main window knows where each GUI resides, it passes an event to the right descending DChart. A descending DChart would choose whether to handle that event locally or to pass it on to another DChart, and so on.

3. The *State* part keeps a memory of the application's data members. Some DCharts may need to use these variables in order to decide what to do at a particular moment.
4. The *Methods* part has all pieces of code that DCharts might need to call.

When generating code from this formalism, the four parts can be easily generated to Python files. The Icons Editor that François Plamondon wrote in 2003 can generate python code for a graphical representation. This tool will enable us to generate code for the *Visual* part. As for the *Behavior p*art, Thomas Feng's SCC can generate code from a DChart. Lastly, the *State* and *Methods* parts can be saved "as is" to a Python file. The formalism's code generator engine will glue these python files together and provide a main file to run.