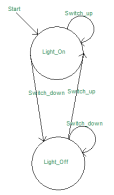
Ionospheric Radar Experiment Scenario Modeling

Natalie Larson, Vanderbilt University
MIT Haystack Observatory REU Summer 2011
Mentors: Bob Schaefer, Phil Erickson

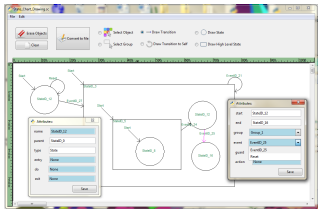
ABSTRACT

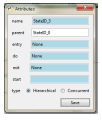
MIT Haystack's Atmospheric Sciences Group has for nearly 50 years operated the Millstone Hill upper atmospheric radar, focused on studies of the near-Earth space environment. The radar is a complex system, which can pose a challenge for experiment design. The aim of this project was to build a graphical user interface to allow a scientist to design and model scenarios of experiment flow through the Millstone Hill radar experiment chain. This end was achieved by creating an interface which allows a user to draw a series of shapes representing states and arrows representing transitions between states in order to create a Harel state chart. While the software meets this specific need it was also designed to be able to represent a wide variety of state charts for use in other future applications. The software supports such functions as moving objects, cutting, pasting, saving and opening files, undoing, redoing, adding and editing state and transition attributes, choosing from previously entered data, and performs a number of tasks to automatically ensure consistency within the state chart. When the state chart is complete the user may convert the drawing into a text document consisting of each state and each transition listed with its accompanying attributes. This file is then used as input to an auto-generating computer code tool designed to execute the scenario as a simulation on distributed processors. While open source software exists to translate code into state chart drawings, no low-cost readily available software could be found to translate state chart drawings into code. For this reason, and for the flexibility and control self-made software affords, the state chart drawing program was created from scratch.

PROGRAM USE



State Chart Example

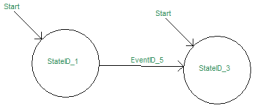




Attribute menu for super-states

- Circles represent states
- Rectangles represent states which are composed of other states and may be either hierarchical or concurrent, a convention adopted from Harel State Charts
- Hierarchical super-states contain groups of states which must be executed serially and which cannot be executed simultaneously or independently
- Concurrent super-states contain groups of states which may be executed independently of other groups of states
- The project makes a distinction between hierarchical and concurrent states in order to delineate states or groups of states which may be executed on separate computers and processed in parallel
- Hierarchical groups of states must have one and only one start state, while each state in a concurrent group of states must be a start state
- A super-state's type may be changed by right-clicking on the state and changing attributes in a pop-up menu; when this happens, the start arrows for all objects within the state are checked for consistency and reassigned if necessary
- The start state within a super state may be selected by entering the name of the start state in the super-state's pop-up menu, or by drawing a start arrow to the state; start arrows are rearranged if necessary to maintain consistency.
- Arrows represent transitions between states
- Attributes of states and transitions such as name, code to be executed when a state is entered, and a group name for events corresponding to a particular process may be entered by the user (by right-clicking on an event or state), though some attributes which are displayed in the attribute menus are made non-editable to ensure consistency between the attribute fields and the drawing (start and end fields for transitions, the parent field for states and super-states, and the type field for states)
- Those attributes are saved in a list corresponding to the object
- Lists of all circles, rectangles, arrows, and loops are used to facilitate the program
- When the drawing is complete, the "Convert to file" button prompts the user to enter a filename and then prints a specially formatted list of all states, transitions, and corresponding attributes which is translated by a different module into python code for directing performing tasks such as directing a radar antenna

SAMPLE DRAWING



-State 1 is a required, implicit concurrent base and transition 1 is a required, implicit start transition for State 1

-Events must be listed by group before listing states and transitions (a convention of the program reading the output file)

-States must be listed in order of parent (Hierarchical or Concurrent super-state), with parent states coming before child states

OUTPUT

STATECHART]
name = output_test

[OPTIONS]
Debug = True
Print = True
Genecode = True
GenCallbacks = True
Interpret = True
LimitAction = False
InsertCB =

[INITIAL]
filename = init_des.p4

[EVENTS_1]
base = 1
name = Group_1
event_1 = EventID_5

[STATE_1]
name = StateID_0
parent = None
type = ConcurrentState
entry = None
do = None
exit = None

[STATE_2]
name = StateID_1
parent = StateID_0
type = State
entry = None
do = None
exit = None

[STATE_3]
name = StateID_3
parent = StateID_0
type = State
entry = None
do = None
exit = None

[TRANSITION_1]
start = start
end = StateID_0
event = None
guard = None
action = None

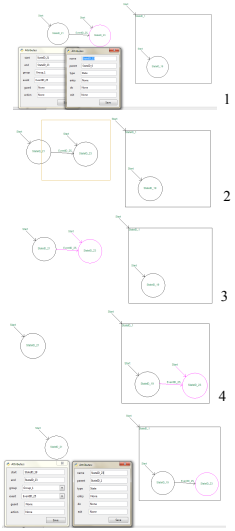
[TRANSITION_2]
start = start
end = StateID_1
event = None
guard = None
action = None

[TRANSITION_3]
start = start
end = StateID_3
event = None
guard = None
action = None

[TRANSITION_4]
start = StateID_1
end = StateID_3
event = Group_1.EventID_5
guard = None
action = None

SUPPORTED FEATURES

Moving objects: Supported features used to move objects include single or group object selection, relocation, automatic arrow length and connection point correction, automatic parent recognition for objects and start and end recognition for transitions, and automatic adjustment of number of start arrows based upon hierarchical or concurrent parent state status.



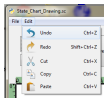
1. The parent fields of states and the start and end fields of transitions are sensed and automatically recorded upon any creation, move, or paste of a state or event. Note the parent of StateID_23 and start and end attributes of EventID_25 before the move.

2. Groups of objects may be selected using the Select Group button, which creates a rectangular drag tool. The tool captures all objects inside the selection rectangle. Single objects may be selected with the "Select Object" button, by clicking within a delta distance of any line comprising an object.

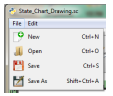
3. Selected objects are highlighted and can then be dragged.

4. The mouse button has not yet been released and the arrow (EventID_25) has overshot StateID_19.

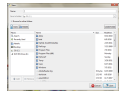
5. The mouse button has been released and the program has automatically corrected the length and attachment point of EventID_25, has deleted the start arrow pointing to StateID_23 since its super-state, StateID_1, is Hierarchical, and has updated the parent attribute of StateID_1 and the start and end attributes of Event ID_25.



Undo, redo, cut, copy, and paste and their keyboard shortcuts are supported. The delete key may be used to erase selected objects, instead of the Erase Objects button.



A dialog window allows the user to choose a location of and name for the drawing file. The name of the text output file is selected when the Convert to File button is clicked.



Users may enter new event and group names, or may select from all previously entered group and event names

FEATURE	ALGORITHM
Saving and opening files	Pickling: Python function that turns objects into byte streams
Keeping start arrows consistent	Find the smallest surrounding rectangle to find parent type (Hierarchical or Concurrent), update arrows according to type. Check for consistency every time an object is created, moved, pasted or erased. If parent is concurrent: delete all arrows and then reassign start arrows to all objects in the parent rectangle. If hierarchical: look for one start arrow, take the first one found and then erase all start arrows in the rectangle, excluding the first arrow found, else if no start arrow was found, assign a start arrow to the first object within the rectangle found.
Real-time drawing of arrows	Change endpoints of arrow to current cursor location and continually redraw the screen
Real-time drawing of rectangles, ability to drag a rectangle in any direction	Get vector made by start click and by current mouse locations; rotate it 45 degrees in each direction to form the rectangle, and take the uppermost, leftmost point, which (along with width and height) is the input to a draw rectangle function
Snap-to-fit arrows	If the point clicked is within a delta distance of a certain shape, create a vector from the object's center to the point clicked; if the object is a circle, scale the vector by the radius of the circle to obtain the snap-to point on the circle; if the object is a rectangle, determine which edge will be crossed by the vector and find the intersection of the vector and the line segment of the rectangle's edge.
Highlighting/Selecting objects	Is the click within a delta distance of any object's perimeter? Arrow: does the point lie within the rectangle created by delta d on either side or on either end of the line segment? Circle: does the point satisfy the equation of the circle, accounting for delta d? Rectangle: does the point lie within any of the four rectangles created by delta d on either end or on either side of each line segment?
Ensuring consistency of parent fields of states and start and end fields of transitions	Find the smallest surrounding rectangle to find an object's parent. Find objects whose coordinates (the bounding box of their coordinates) lie within a rectangle to find a rectangle's children. Update child-parent relationships every time an object is created, moved, pasted or erased.
Undo/Redo	Keep a set of super-lists to hold the current values of all variables needed to recreate the current drawing. Append new values to each of the lists any time an action is taken. Undo an action by moving backward through the set of lists; redo by moving forward.
Moving object(s)	For all highlighted objects, draw vectors from the original point clicked to all object parameters needed to draw the object (for example, radius, for a circle). As the cursor moves, update the object's parameters to be the current location of the mouse minus the vector.
Printing states so that each parent state is printed before its child	Use a depth-first-search: write the required base state first, then add its name to a list. While the list is not empty, pop the first name from the list and set a variable, "previous," to that name. Then, find all rectangles that list "previous" as a parent and write out their attributes. Add the names of these rectangles to the list. Print all circles afterward.

FUTURE WORK

-Print button, print preview button
-Add capability to represent events which are turned into messages and then back into events to facilitate distributed computing
-Ability to draw loops at any location on a shape
-Rotate event names so that they appear at the same angle as the arrow to which they correspond
-Add ability extend or contract lengths of arrows when objects or groups of objects are moved, if keeping all objects attached is desired
-Way to show movement that does not cause the screen to blink