Spring 2013

Entropy and State Visualization for Automation Design and Evaluation Prototyping Toolset

Rohit Deshmukh
San Jose State University

Entropy and State Visualization for Automation Design and Evaluation Prototyping Toolset

A Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Rohit Deshmukh

Spring 2013

SAN JOSE STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

Entropy and State Visualization for Automation Design and Evaluation

Prototyping Toolset

by

Rohit Deshmukh

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

Spring 2013

Dr. Soon Tee Teoh, Department of Computer Science

Date

Dr. Michael Feary, NASA Ames Research Center

Date

Dr. Mark Stamp, Department of Computer Science

Date

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research

Date

ABSTRACT

Entropy and State Visualization for Automation Design and Evaluation Prototyping Toolset

by

Rohit Deshmukh

Automation Design and Evaluation Prototyping Toolset (ADEPT) is a plug-in developed on the Eclipse Rich Client Platform(RCP). ADEPT can be used by domain expert designers to create and modify testable prototypes. The aim of the project is to enhance ADEPT by adding dynamic visualizations to the ADEPT user interface. Three types of visualizations are implemented in this project. Table view is helpful to view the hierarchy and nesting of Logic Tables. The State visualization displays all the states in a selected Logic Table. Entropy visualization is a subset of State visualization and displays limited number of states having lowest Entropy value.

ACKNOWLEDGEMENTS

I would like to say thank you to all the people who have helped me with my project. Thank you very much Dr. Michael Feary for giving me an opportunity to work on ADEPT and providing me with a prototype of an airplane built in ADEPT for testing the visualizations. I would like to thank my advisor Dr. Soon Tee Teoh for his guidance and his excellent insights throughout the duration of the project. I would like to thank Dr. Mark Stamp for his time and for agreeing to be a member of my project committee. I would like to thank my family and all of my friends for their encouragement and support throughout the Master's program.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Introduction

The Human Systems Integration Division is addressing the issue that current automation design and modeling tools do not provide adequate support for the specification of automation decision logic and user interaction with automation decision logic. Automation Design and Evaluation Prototyping Toolset (ADEPT) has been developed to address this need. ADEPT is a plug-in developed at NASA Ames Research Center. ADEPT allows domain experts to design and to rapidly create and modify testable prototypes. Computational Human-Computer Interaction (HCI) analyses have been integrated with the ADEPT design framework to provide designers without HCI expertise, the ability to evaluate HCI aspects of prototype designs. ADEPT is being developed to help identify the Human-Automation Interaction (HAI) vulnerabilities. ADEPT allows to create an iterative specification of decision logic of the automation being designed. An accurate and complete specification for a design prototype can be built using ADEPT. "In addition to the focus on specifying decision logic, the tool is intended to provide a platform for integrating HAI testing and analysis" [1].

The proposed project is an effort to improve the ADEPT plug-in. As part of the initial project work, the areas for improvements in ADEPT were identified. Some projects built in ADEPT, such as the auto-pilot interface design prototype for an airplane, resulted in multiple Logic Tables which made the project too complicated, to understand the Logic Tables. Logic Tables also support the nesting of tables which makes the Logic Tables more complex to understand. Moreover, if there are multiple collaborators using same ADEPT project, it is hard for all of them to follow the hierarchy of tables and logic specifications. Table View is a visualization that was created to eliminate this issue. Another important requirement for ADEPT is to have a state diagram that shows the possible states for each Logic Table. As the state diagram visualization is implemented it results in a large number of states being generated for any complex prototype design such as the auto-pilot design. This leads to a new visualization which displays only a few nodes of higher interest from each Logic Table.

The report focuses on the implementation of visualization techniques for the ADEPT plug-in. All software and tools used for this project are discussed in the following section. In addition, a detailed introduction to the ADEPT user interface is given. Next, the report briefs on the preliminary work done and the initial improvements made to the ADEPT user interface. The rest of the paper discusses, in detail, the design and the uses of all the three visualization techniques: Table-view, State-view, and Entropy-view. In the final section the report discusses the dynamic integration of the visualizations with the ADEPT plug-in. Finally, the report ends with a conclusion and references.

2. Software Tools Used

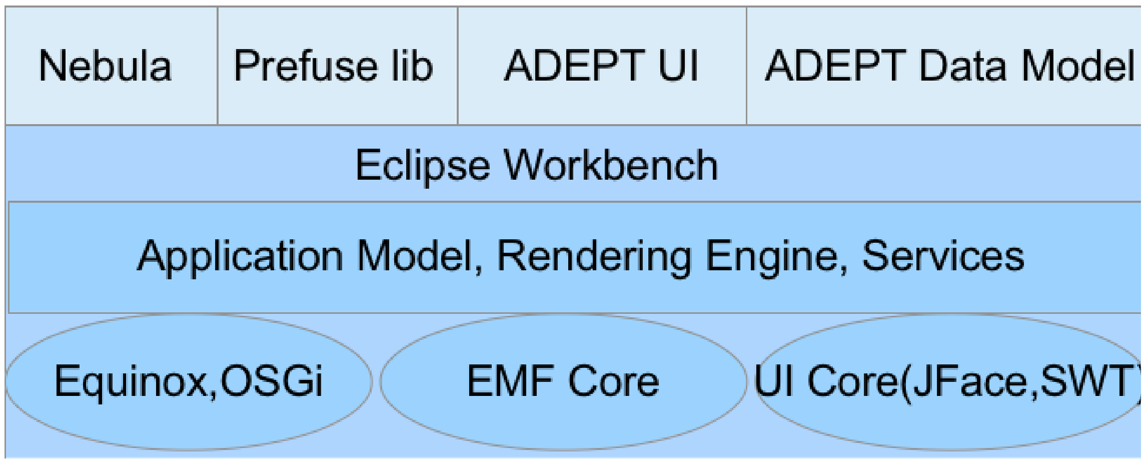


Figure 1: ADEPT plug-in with Eclipse architecture and other software tools used

2.1. Eclipse RCP

Eclipse is a special Integrated Development Environment (IDE) which can be used as a tool to create new applications using various high level programming languages such as Java, C++ and PHP [2]. Important aspect of the Eclipse IDE is that it can be extended by programmers, to create their own Rich Client Platform plug-ins specific to their project requirements, instead of reinventing the wheel [3]. Programmers can use the underlying Eclipse framework and build plugins which use the services provided by Eclipse and develop application specific interfaces that can be fully integrated with the Eclipse IDE.

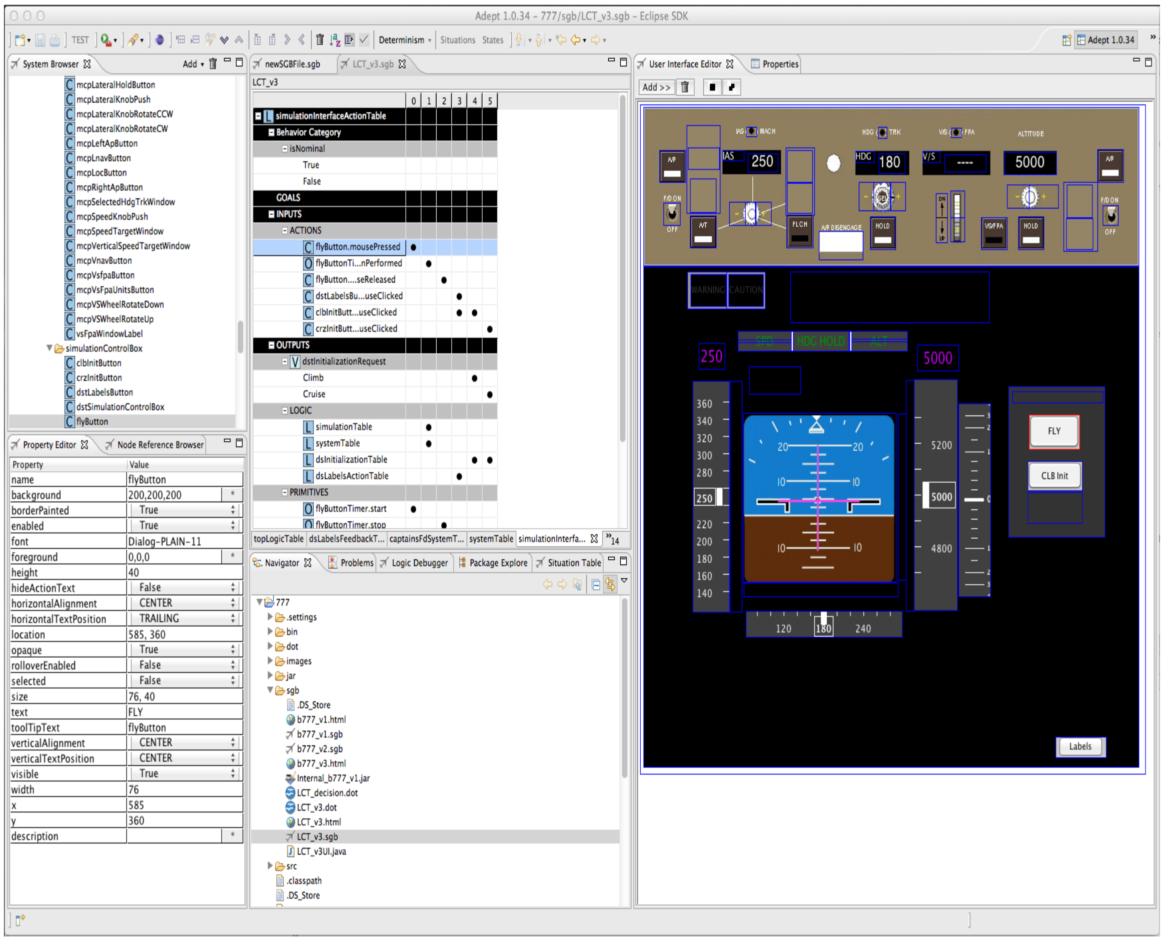
In this project Eclipse platform was used to build ADEPT plugin, which simplifies the task for domain experts to design and code the prototype without having them to write any high level language code. Figure1 above shows Eclipse framework outlined in dark blue color, contains all the core Eclipse tools provided by Eclipse as part of the Eclipse SDK. The contribution made towards the development of this project is shown in the top row highlighted in light blue color. We will briefly discuss each component in the following section.

2.2. ADEPT plug-in

My project aims to improve the user interface and add visualization capabilities to the ADEPT plug-in. We need to have a brief introduction to ADEPT in order to understand the project.

Figure 1 shows ADEPT integrated on top of the Eclipse Framework. The ADEPT plug-in contributes menus, views and its own editor to the Eclipse IDE. ADEPT uses many components of the underlying eclipse framework and adds its own functionality to form a fully integrated application for HAI testing and analysis.

The main User Interface of ADEPT constitutes of the following components User Interface Editor, Logic Editor, System Browser and Property Editor.



Property	Value
name	flyButton
background	200,200,200
borderPainted	True
enabled	True
font	Dialog-PLAIN-11
foreground	0,0,0
height	40
hideActionText	False
horizontalAlignment	CENTER
horizontalTextPosition	TRAILING
location	585, 360
opaque	True
rolloverEnabled	False
selected	False
size	76, 40
text	FLY
toolTipText	flyButton
verticalAlignment	CENTER
verticalTextPosition	CENTER
visible	True
width	76
x	585
y	360
description	

Figure 2: ADEPT plug-in the user interface

Figure 2 displays the user interface for the ADEPT plug-in. The main center panel in Figure 2 represents the Logic Editor; the panel to the right of Logic Editor is the User Interface Editor. System Browser is on the top left and Property Editor is at the bottom left in Figure 2. ADEPT offers a graphical user interface design capability with an automation behavior specification capability. ADEPT also provides an automatic code generator to enable domain expert designers to create testable software prototypes. The User Interface Editor provides a blank canvas to the user. The user can add graphic objects and controls to resemble the user interface of the prototype. Buttons, display labels and other widgets can be used as graphical objects. Graphical objects created in other applications can also be imported into ADEPT. The property browser facilitates to change the properties of the graphical objects. In addition the properties can be changed dynamically by dragging the objects into the Logic Editor, corresponding to the automation behavior. "The Logic Editor enables the designer to specify the decision logic and automation behavior of the device, the environment in which the device operates, as well as the behavior of the user interface objects on the user-interface, corresponding to reflect the current state of the device and environment" [1].

2.3. Nebula widgets toolkit

Nebula is a toolkit which provides a set of Widgets with special features that are not available with the standard widgets provided by the Java SWT Widgets.

Nebula is a collaborative effort of different Eclipse projects and individual programmers. The source code for the widgets is also available under the Eclipse Public License 1.0, this was the major factor in choosing Nebula widgets library for ADEPT. Moreover the Nebula widgets toolkit is build using the SWT graphics so there were no compatibility issues with other views in ADEPT. The Grid widget was custom tailored to create the main component of user interface which is the Logic Editor Table [6].

2.4. Prefuse

Prefuse is a library that provides a framework for building rich graphical visualizations. Prefuse provides Java 2D graphics interface to create graphic objects, which can be easily integrated in to ADEPT project. Prefuse is an open sourced project and the source code is available under terms of BSD license. The availability of source code provides flexibility to extend the interfaces and develop custom graphical objects. Prefuse follows the information visualization reference model as shown in figure 3 below.

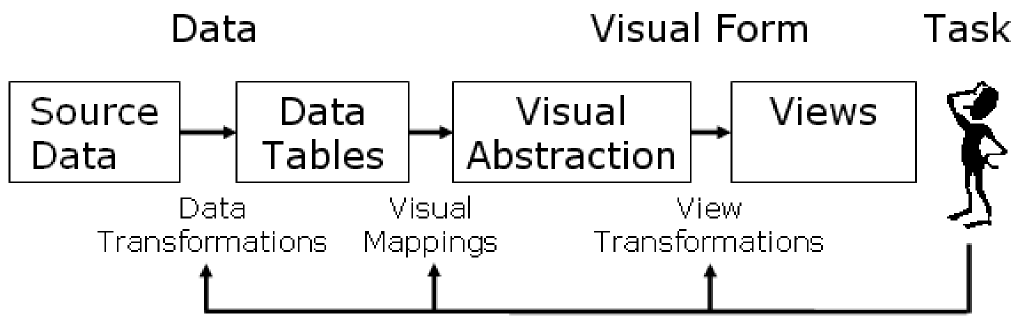
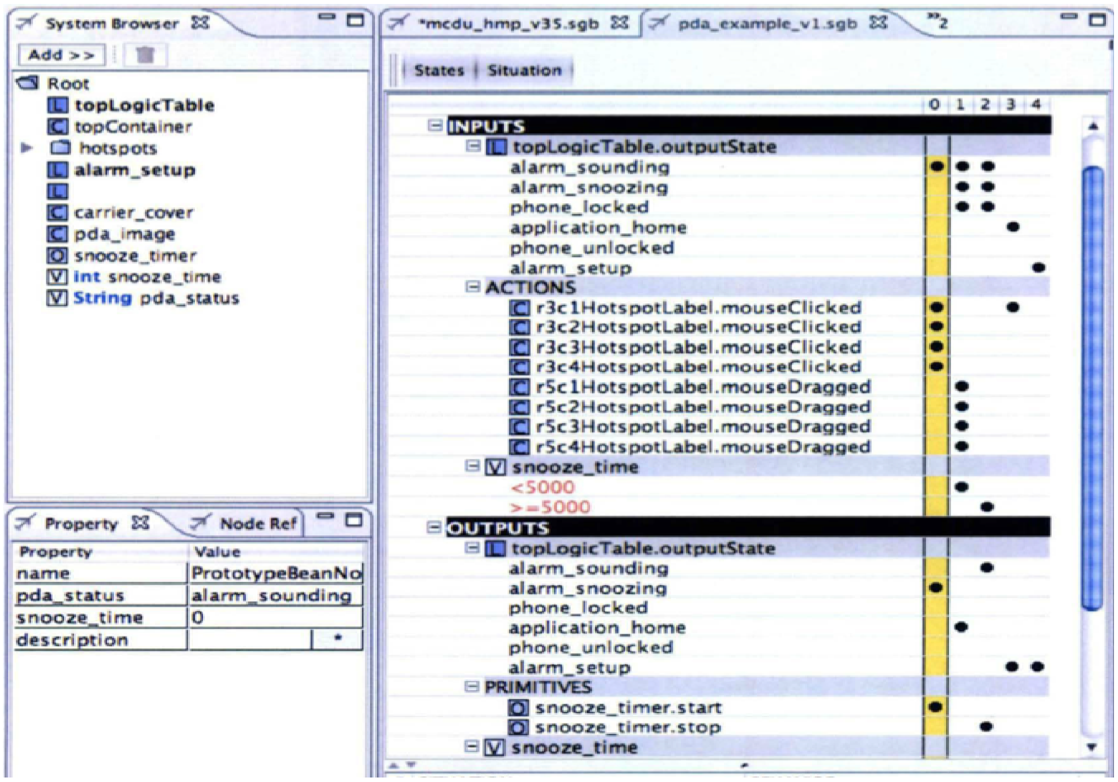


Figure 3: Information visualization reference model [7]

This architecture simplifies the data visualization process into simple individual steps. Data is transformed into set of Data Tables. In the next step a visual abstraction is created depending on the kind of visual object we want to use to display the data. The visual abstraction will contain the definition for the properties such as shape, color and position of the graphical object. This visual abstraction is then transformed to create interactive views. User has the freedom to modify at any intermediate step of the data, data tables or the visual abstraction process. Prefuse library is extensively used in the development of the three visualizations in this project.

3. Preliminary work done to improve the ADEPT plug-in

During the preliminary part of this project, necessary requirements to improve the user-interface for the ADEPT plug-in were identified. The Logic Table which is the main component of the IDE had issues with Drag and Drop functionality. The swing component used to design and build the Logic Editor Table, was not compatible with the new SWT framework that Eclipse uses extensively. The only way to rectify this issue was to replace the main component representing the Logic Table, with another widget that was compatible with rest of the Eclipse components and would support Drag and Drop functionality. We decided to use the Nebula project Grid widget which is a custom SWT widget. The Grid widget is a spreadsheet/table component that offers features not currently found in the base SWT Table [5]. The Grid widget was customized to add a column select focus control. In the ADEPT plug-in the column represents a situation, while the row specifies the state. It is important for the designer to visualize if either a row or a column is selected; even the toolbar Menus are activated depending on whether user selects a row or a column. To improve the action feedback of row/column selection a focus column was added to the original widget. When a column is selected it has a focus unlike other widgets, where focus is used only to highlight a row selection. Another major change necessary was to replace the System Browser view which is basically a tree structure also supported by a swing tree widget. The swing widgets also caused flickering problems during resizing and moving. Hence it was necessary to replace both these swing components with suitable widgets.



System Browser | Add >>

- Root
  - topLogicTable
  - topContainer
  - hotspots
  - alarm_setup
  - carrier_cover
  - pda_image
  - snooze_timer
  - int snooze_time
  - String pda_status

Property | Node Ref

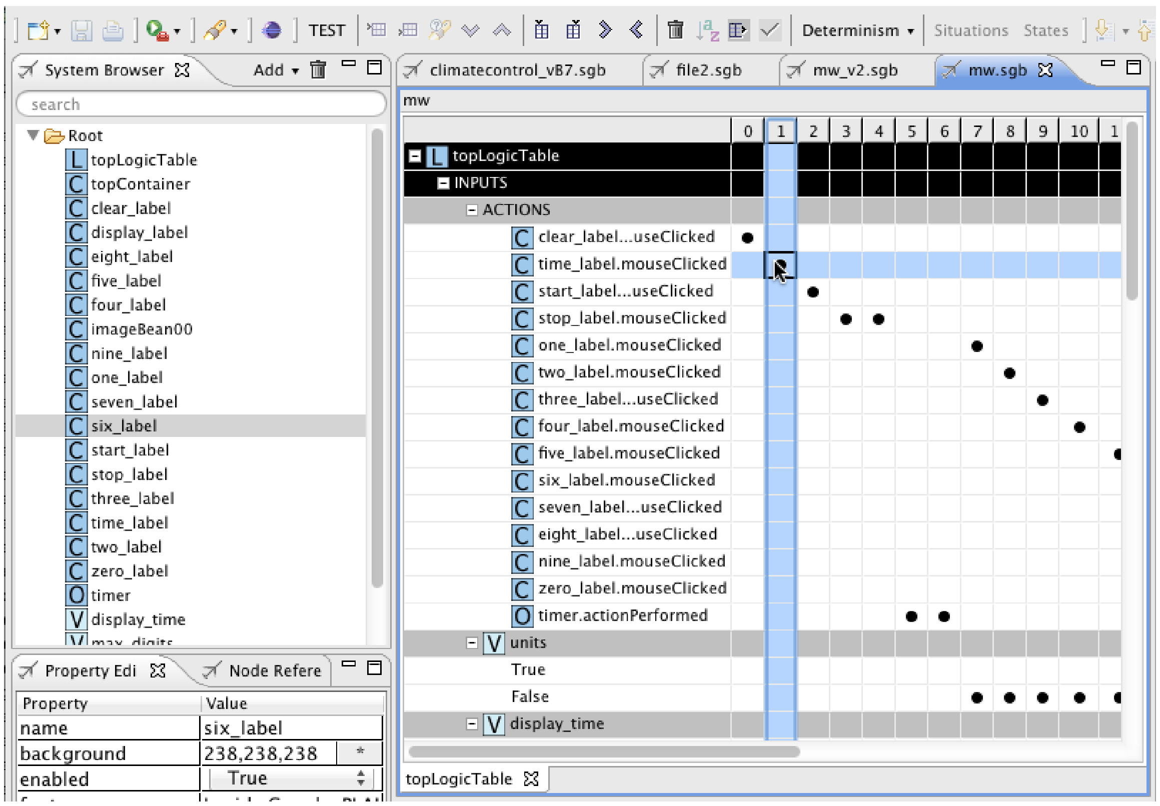
Property	Value
name	PrototypeBeanNo
pda_status	alarm_sounding
snooze_time	0
description	

*mcdu_hmp_v35.sgb | pda_example_v1.sgb

States | Situation

	0	1	2	3	4
INPUTS					
topLogicTable.outputState					
alarm_sounding	●	●	●		
alarm_snoozing		●	●		
phone_locked		●	●		
application_home				●	
phone_unlocked					
alarm_setup					●
ACTIONS					
r3c1HotspotLabel.mouseClicked	●			●	
r3c2HotspotLabel.mouseClicked	●				
r3c3HotspotLabel.mouseClicked	●				
r3c4HotspotLabel.mouseClicked	●				
r5c1HotspotLabel.mouseDragged		●			
r5c2HotspotLabel.mouseDragged		●			
r5c3HotspotLabel.mouseDragged		●			
r5c4HotspotLabel.mouseDragged		●			
snooze_time					
<5000		●			
>=5000			●		
OUTPUTS					
topLogicTable.outputState					
alarm_sounding			●		
alarm_snoozing	●				
phone_locked					
application_home		●			
phone_unlocked					
alarm_setup				●	●
PRIMITIVES					
snooze_timer.start	●				
snooze_timer.stop			●		
snooze_time					

Figure 4: Old ADEPT plug-in



TEST | Determinism | Situations | States

System Browser | Add

search

- Root
  - topLogicTable
  - topContainer
  - clear_label
  - display_label
  - eight_label
  - five_label
  - four_label
  - imageBean00
  - nine_label
  - one_label
  - seven_label
  - six_label
  - start_label
  - stop_label
  - three_label
  - time_label
  - two_label
  - zero_label
  - timer
  - display_time

Property Edi | Node Refere

Property	Value
name	six_label
background	238,238,238
enabled	True

climatecontrol_vB7.sgb | file2.sgb | mw_v2.sgb | mw.sgb

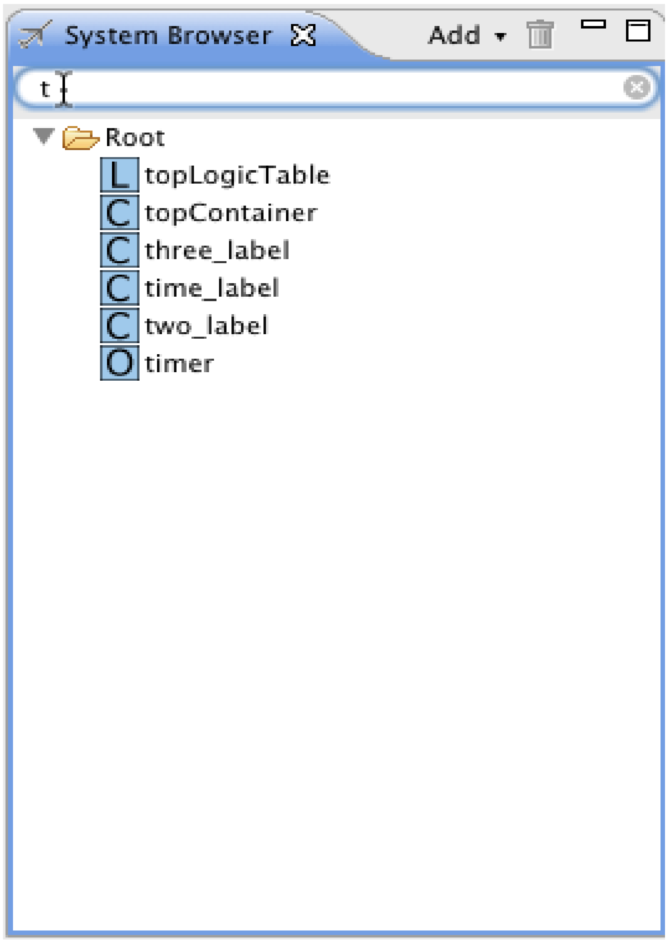
mw

	0	1	2	3	4	5	6	7	8	9	10	1
topLogicTable												
INPUTS												
ACTIONS												
clear_label...useClicked	●											
time_label.mouseClicked		●										
start_label...useClicked			●									
stop_label.mouseClicked				●	●							
one_label.mouseClicked								●				
two_label.mouseClicked									●			
three_label...useClicked										●		
four_label.mouseClicked											●	
five_label.mouseClicked												●
six_label.mouseClicked												
seven_label...useClicked												
eight_label...useClicked												
nine_label.mouseClicked												
zero_label.mouseClicked												
timer.actionPerformed						●	●					
units												
True												
False								●	●	●	●	●
display_time												

topLogicTable

Figure 5: New ADEPT plug-in with Nebula Grid widget demonstrating focus column and System Browser

The System Browser lists all the controls, Logic Tables, user objects and variables used in a project. In certain projects with many controls and variables it becomes hard to find a variable or an item under the System Browser. To search for a particular variable, the user had to visually scan and scroll the entire tree until the variable is found. A search feature within the System Browser was required to ease this task. In the new version of System Browser the search feature was implemented and the older swing widget was replaced by a JFace tree viewer. The two figures above Figure 4 demonstrates old ADEPT plug-in and Figure 5 displays the new ADEPT plug-in. System Browser is on the top left side in the plug-in. Figure 6 below highlights the search feature in action, for the System Browser. The search feature shows all the nodes from the tree that matches the string provided in the search box, while hiding all other nodes. The original tree is visible again by clearing off the search box.



Root
- topLogicTable
- topContainer
- three_label
- time_label
- two_label
- timer

Figure 6: Search Feature for System Browser

4. Logic Tables View

4.1. Need for Logic Tables View

ADEPT is used to design prototypes for applications with varying complexity of design, smaller prototypes could be built using only one Logic Table, but for larger applications with more than 100 user objects and controls, it is easier to define logic for behavior specification using multiple Logic Tables. These Logic Tables are created using System Browser and can be organized using folders. Organizing the Logic Tables, controls and variables in folders do not necessarily display the way these tables are activated. The behavior specifications specify the logic for activation of these Logic Tables. The Logic Table Editor is used to set and view this information, however only one Logic Table can be viewed at a given instance, inside an editor. Hence for prototypes consisting of multiple Logic Tables nested in a hierarchy, there is a need for designers to be able to view the activation sequence or dependency of Logic Tables. Also from usability studies performed on ADEPT, the participants felt that some work was needed to make the prototypes they designed understandable to others in their design group [1]. Hence a visualization was necessary to get a better insight on the activation of these tables and the hierarchy followed while building a prototype design in ADEPT.

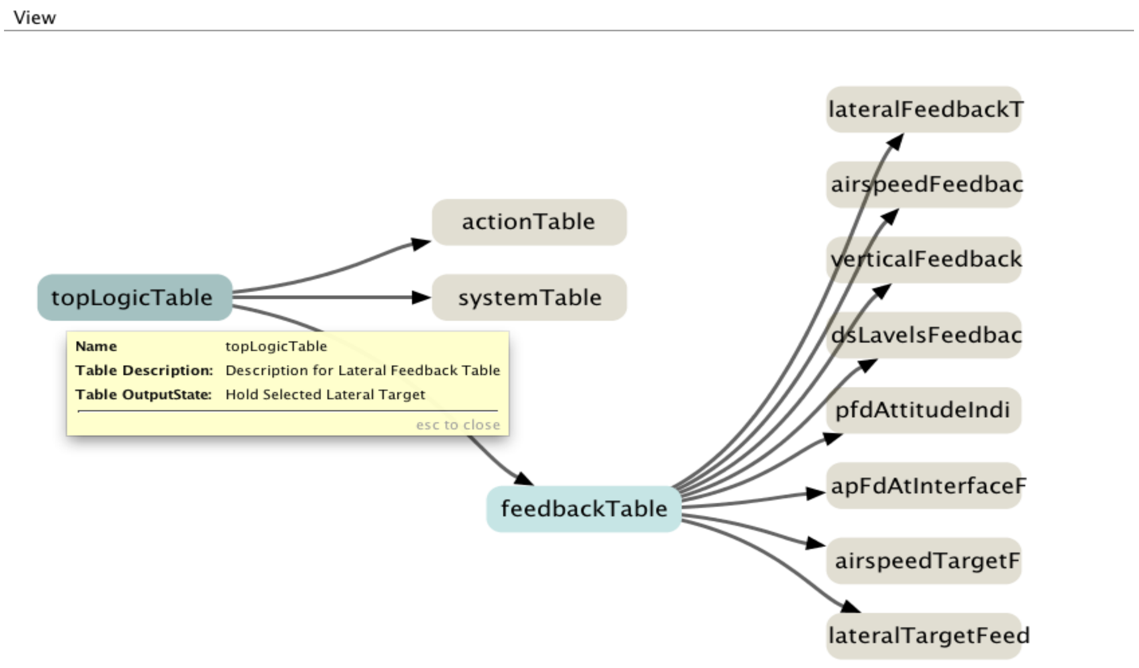
4.2. Design and Implementation

The requirement for this visualization was to display all the Logic Tables in a hierarchy showing how each table is activated. I used a Node-Link Tree layout for this visualization. The visualization displays a rooted tree such that each depth level of a tree is on a shared line. The orientation of the tree is set from left to right. The challenging step was to parse the ADEPT project file and extract the table structure.

ADEPT uses logic tables to create the logic for a prototype using 0 and 1 as logical operators, transforming this data into state diagram required a complex parser. Parser is an important part of the project, it helps in transforming tabular representation into graphical visualization. We implemented a parser which parses the ADEPT project file containing the logic. The parser for States View searches the ADEPT file for the TopLogicTable, which is a root by default for all prototypes build in ADEPT. Next the parser searches for the output Node and collects all logic nodes used recursively in a Logic Table, finally giving a tree structure. The following recursive algorithm shows the process of data collection for the Logic Tables view.

```
FindLogicNode(topLogicTable, root)
for each table search for the Output node
    for each Output node search for Logic Table
        if found add Logic Table as child to the root
        call FindLogicNode(newLogicTable, newParentnode)
```

Eventually we modify the parser for collecting data for other visualizations which is discussed in the later sections. The Figure 7 below shows the visualization. In this example we can easily notice that actionTable, systemTable and feedbackTable can be activated through the topLogicTable. When the user hovers over a node, the visualization displays important properties of the Logic Table such as default output-state for the table.



Name	topLogicTable
Table Description:	Description for Lateral Feedback Table
Table OutputState:	Hold Selected Lateral Target

Figure 7: Logic Tables View

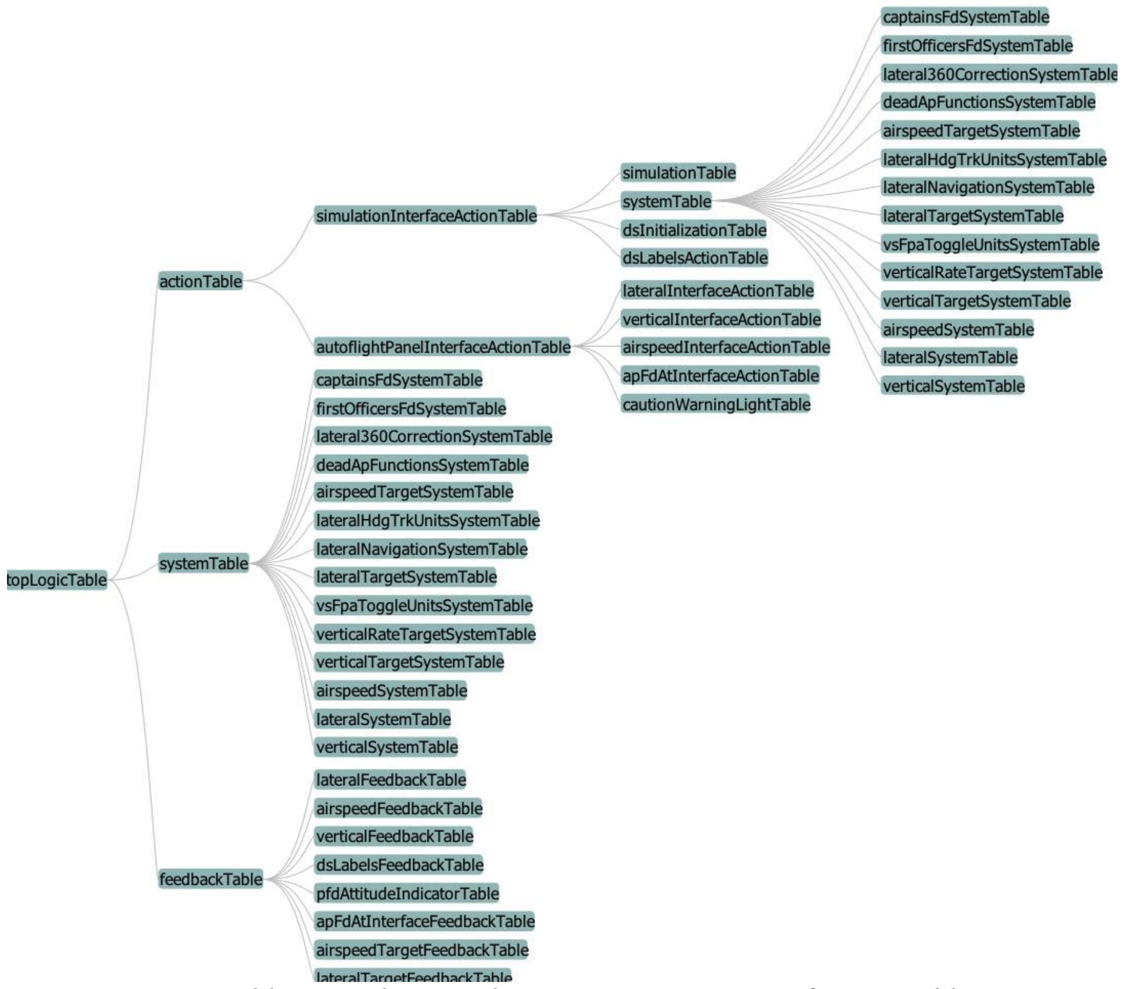

topLogicTable
actionTable
systemTable
feedbackTable
simulationInterfaceActionTable
autoflightPanelInterfaceActionTable
simulationTable
systemTable
dsInitializationTable
dsLabelsActionTable
lateralInterfaceActionTable
verticalInterfaceActionTable
airspeedInterfaceActionTable
apFdAtInterfaceActionTable
cautionWarningLightTable
captainsFdSystemTable
firstOfficersFdSystemTable
lateral360CorrectionSystemTable
deadApFunctionsSystemTable
airspeedTargetSystemTable
lateralHdgTrkUnitsSystemTable
lateralNavigationSystemTable
lateralTargetSystemTable
vsFpaToggleUnitsSystemTable
verticalRateTargetSystemTable
verticalTargetSystemTable
airspeedSystemTable
lateralSystemTable
verticalSystemTable
captainsFdSystemTable
firstOfficersFdSystemTable
lateral360CorrectionSystemTable
deadApFunctionsSystemTable
airspeedTargetSystemTable
lateralHdgTrkUnitsSystemTable
lateralNavigationSystemTable
lateralTargetSystemTable
vsFpaToggleUnitsSystemTable
verticalRateTargetSystemTable
verticalTargetSystemTable
airspeedSystemTable
lateralSystemTable
verticalSystemTable
lateralFeedbackTable
airspeedFeedbackTable
verticalFeedbackTable
dsLabelsFeedbackTable
pfdAttitudeIndicatorTable
apFdAtInterfaceFeedbackTable
airspeedTargetFeedbackTable
lateralTargetFeedbackTable

Figure 8: Logic Table View showing the entire tree structure of Logic Tables in a larger prototype

Figure 8 shows a fully expanded Logic Tables View, A copy of an image file, of a fully expanded view is also saved in the project for reference.

5. States Visualization

5.1. Need for States Visualization

ADEPT uses a tabular representation of a finite state machine. "In contrast to a typical state transition diagram, the representation used by ADEPT focuses more on presenting information about the situation (input combination) automation behavior (output combination), and less on presenting information about state transition in a summarized form information. This focus allows a more compact notation, which enables the designer to see more behaviors, making it easier to make a complete specification" [1]. Although the tabular representation is easier for the designers it might not be useful for other collaborators, from different groups who wish to analyze state transitions and research other aspects of the prototype being evaluated.

5.2. Design and Layout

The Logic Editor in ADEPT displays information for states and situations. The rows define state and the columns define a situation. The initial design of the visualization was to display all the situations from a Logic Table in a force directed layout. Since each situation consists of an input and an output, the idea was simply to display all the situations. Figure 9 below shows the initial implementation to view the situations. This design had following problems

- Repetition of nodes since we were using each situation to display a transition, many inputs and outputs were common for different situations.
- There were no transitions from one situation to another.

- For larger projects the visualization generated large number of floating nodes which was difficult to view.
- Overall the visualization generated large number of nodes with no feature to group them together in a hierarchy.

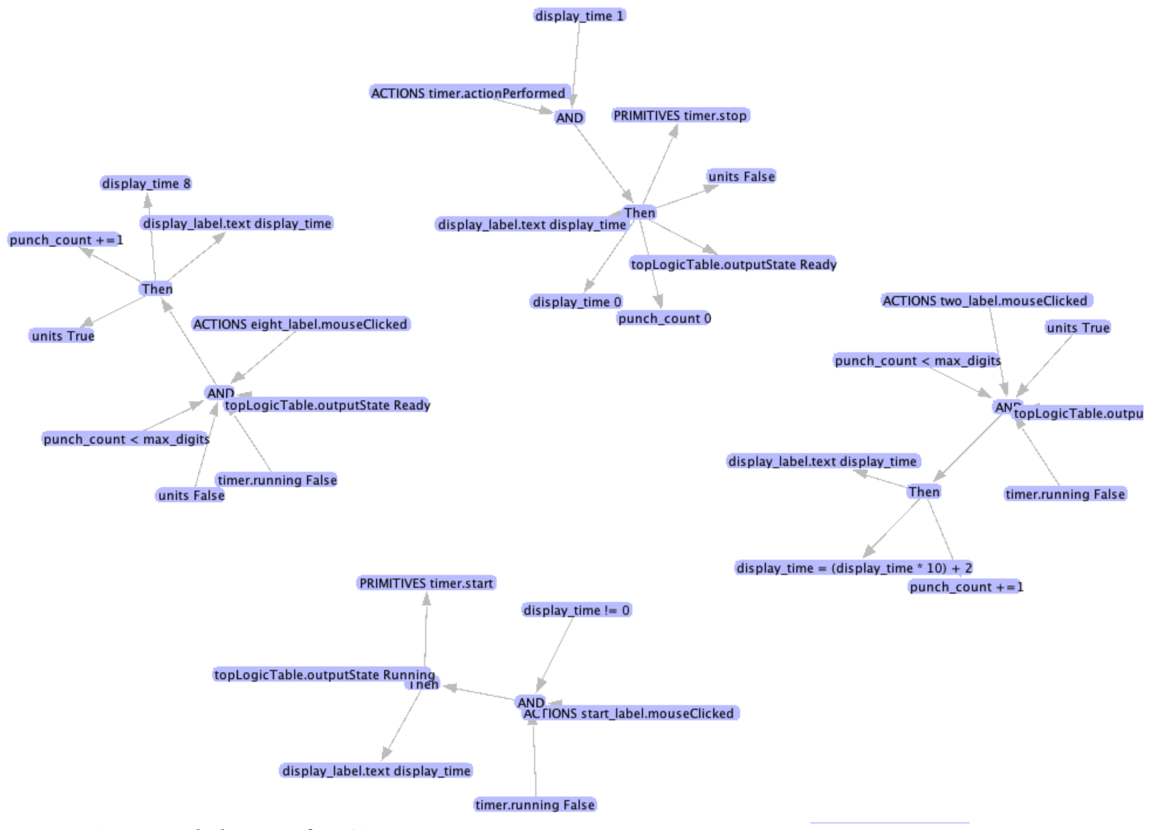

display_time 1
ACTIONS timer.actionPerformed
AND
PRIMITIVES timer.stop
units False
Then
display_label.text display_time
topLogicTable.outputState Ready
display_time 0
punch_count 0
display_time 8
display_label.text display_time
punch_count +=1
Then
units True
ACTIONS eight_label.mouseClicked
AND
topLogicTable.outputState Ready
punch_count < max_digits
timer.running False
units False
ACTIONS two_label.mouseClicked
units True
punch_count < max_digits
AND
topLogicTable.outpu
display_label.text display_time
Then
timer.running False
display_time = (display_time * 10) + 2
punch_count +=1
PRIMITIVES timer.start
display_time != 0
topLogicTable.outputState Running
Then
AND
ACTIONS start_label.mouseClicked
display_label.text display_time
timer.running False

Figure 9: Initial design for States View

The initial design was helpful and led towards a more hierarchical approach and use of animation since the number of nodes increased drastically for complex projects.

5.3. Aggregates and Animation

Prefuse provides an interface to create graphical objects for representing collections in visualizations. Aggregates are used to represent visual items that belong to a particular group. In order to better visualize state transitions, an aggregate is used to represent one situation. A situation in ADEPT is represented by a column, therefore adding all items in a column from the Logic Editor formed an aggregate.

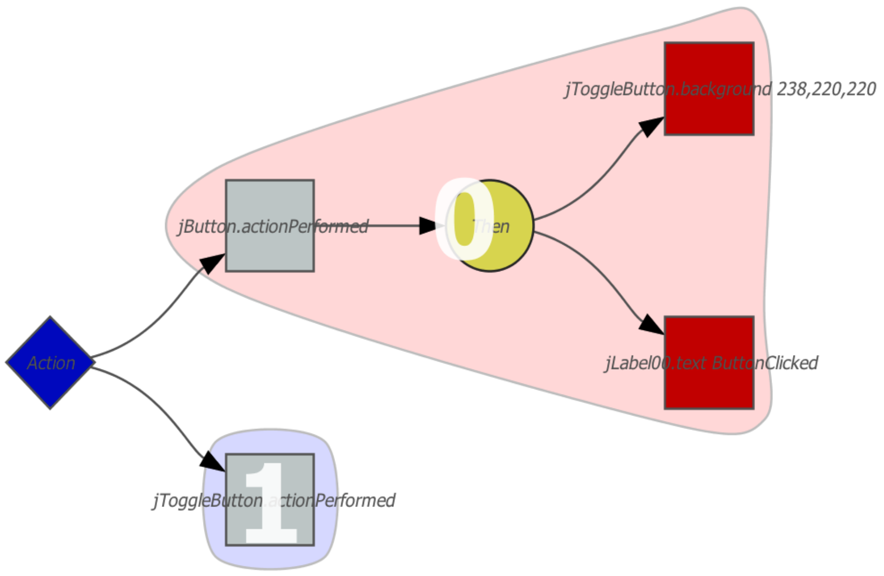

jToggleButton.background 238,220,220
jButton.actionPerformed
0
Then
Action
jLabel00.text ButtonClicked
1
jToggleButton.actionPerformed

Figure 10: Aggregate and decorator example

Layout of the aggregate is created using a convex hull that surrounds all the items in an aggregate. Figure 10 demonstrates an aggregate that has 4 items. Decorators are used for each aggregate to display the situation number. Animation was added so that we can show and hide an aggregation depending on the user interaction. Animation was primarily added because we do not want to overwhelm the user by displaying all the states and situations at once. Figure 10 above also shows two aggregates one which is in focus displaying all the nodes within, and another one which shows only one node. When the user clicks on the node with situation 1, aggregate for that node comes into focus and animates showing all the nodes in that situation, while all other aggregations which are out of focus collapse into a single node. This approach allows the user to focus on one situation at a time.

5.4. Implementation

The most important classes created to implement State View are

- AggregateLayout.java

  This class is provided with the Prefuse toolkit. The class calculates a convex hull to be displayed. Run() method in this class was modified to customize the implementation for States visualization. Changes to the Run() method include displaying all items in an aggregate which is in focus and for other nodes, displaying only one item inside the aggregate.

- AggregateDragControl.java

  This class is an implementation of drag control for an aggregate. The dragging of an aggregate is achieved by moving all the elements inside an aggregate.

- StateToolTip.java

  A tooltip class to display important properties related to the visual items in the visualization.

- StatesView.java

  The main class responsible to create the table and display a visualization in a view. The visualization uses a Force Directed Layout, that positions graph elements based on a physics simulation of interacting forces; by default, nodes repel each other, edges act as springs, and drag forces (similar to air resistance) are applied[10] .

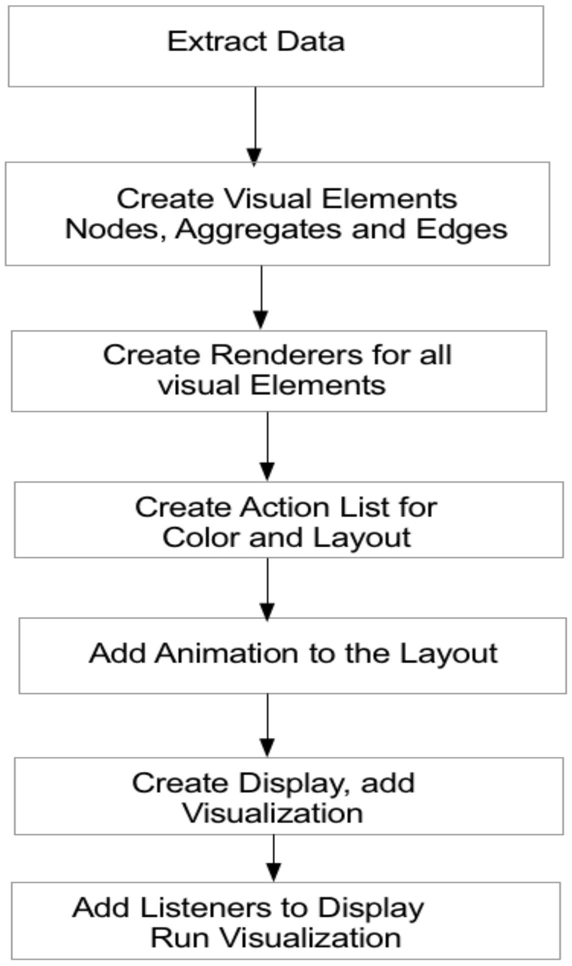

Extract Data
Create Visual Elements Nodes, Aggregates and Edges
Create Renderers for all visual Elements
Create Action List for Color and Layout
Add Animation to the Layout
Create Display, add Visualization
Add Listeners to Display Run Visualization

Figure 11: Steps followed to create States View Visualization

- StatesParser.java

  The parser class to create tree-paths for the States visualization. The StatesParser creates an XML file for each Logic Table. The parser first identifies all the states from the Outputs section in the Logic Table. The Parser then creates transitions based on the specified Logic and actions from the Inputs section of the Logic Table.

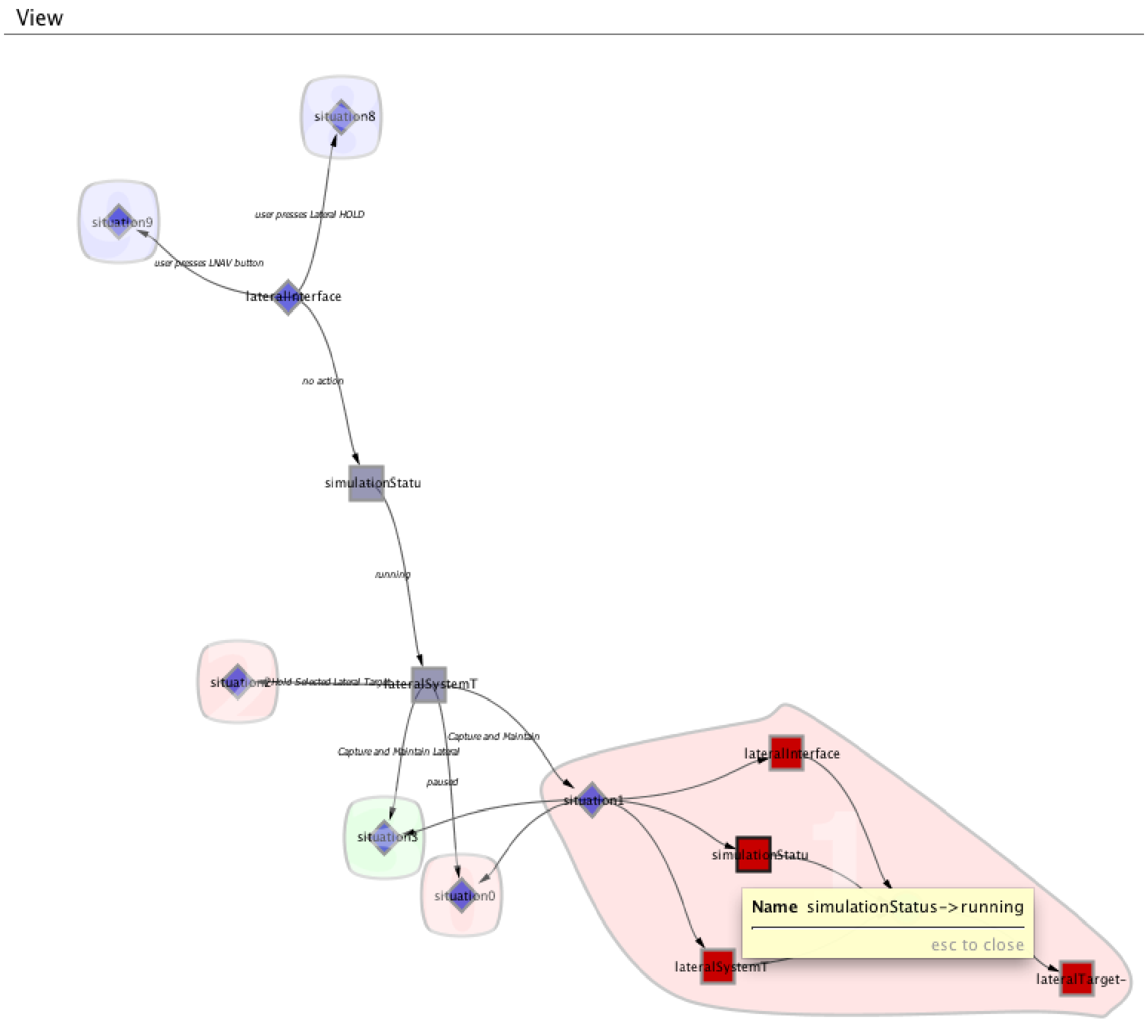


Figure 12: States Visualization example

Figure 12 above shows an example of States visualization. The different shapes are used to differentiate between various elements from the Logic Tables Editor. Square shape is used to show a state defined in a logical table. We use shape of diamond for the “OR” and the “AND” conditions from the Logic Table. “OR” and “AND” conditions can be further distinguished using yellow color for the “OR” and green color for the “AND” condition. Different colors are used randomly for the aggregates representing one State. Actions are listed on the Edges.

5.5. Uses for States Visualization

Currently, ADEPT uses tabular representation of the States and Situations which is primarily used to specify behavioral specifications. The States View visualization makes it easier to view the state diagram for a project. The added functionality to pan and zoom helps the users to easily navigate through all the states. The states view can be used in the Debug mode also, where it can highlight the current state and the transition which can be very helpful for other collaborators to understand the prototype under test.

6. Entropy

6.1. Entropy

Entropy is defined as a measure of uncertainty in the field of information theory. We follow the approach of Shannon's definition of Entropy in this Project. Shannon's entropy measures the possible value of information available in a message. For example consider an example of coin toss. In case of a fair coin the probability for heads or tails is same so the entropy is highest. Since we cannot predict the outcome both sides have equal probability. Similarly, In case of a fair die there are 6 equally possible outcomes. Such a die roll will have 6 bit of entropy. If we replace a die with an unfair die having the letter 1 on each of the 6 sides of the die, then the die roll has zero entropy, since we can perfectly predict the outcome for each roll.

The main step for this visualization is to calculate entropy for each action in the Logic Editor Table against all other states, to find nodes with highest conditional probability. A graph is drawn for nodes with low entropy values to filter states that are relatively more predictable to be executed in a prototype design. Another easy example to summarize is a prototype of a tape recorder. The entropy calculated for the on-off switch would be low since no other action can be performed without switching it on, so the entropy for that transition would be lowest in comparison to all other actions in the prototype design. Entropy has important characteristics which further substantiate it as a reasonable measure of choice or information [14]. Entropy is zero only if we are certain of the event's outcome. The highest entropy is generated when the probabilities of all possible outcomes are equal. The entropy increases with uncertainty logarithmically.

6.2. Entropy Calculation

We apply Shannon's Entropy calculation to the Situations and States in ADEPT's Logic Editor. In the Logic Editor the state of a Situation is either 1 or 0. We calculate the probabilities of all actions using the Logic Editor.

Suppose p1,p2,p3...,pn are all probabilities of occurrence for n possible situations. Then entropy H is given by,

$$H = -\sum p_i \mathrm{Log}_2\, p_i \qquad [14]$$

Entropy calculation to create the tree for visualization is explained in the block diagram shown below in Figure 13. The tree structure created after adding all these paths is saved in an XML file. The visualization class uses the XML file to generate nodes and transitions for displaying the Entropy visualization.

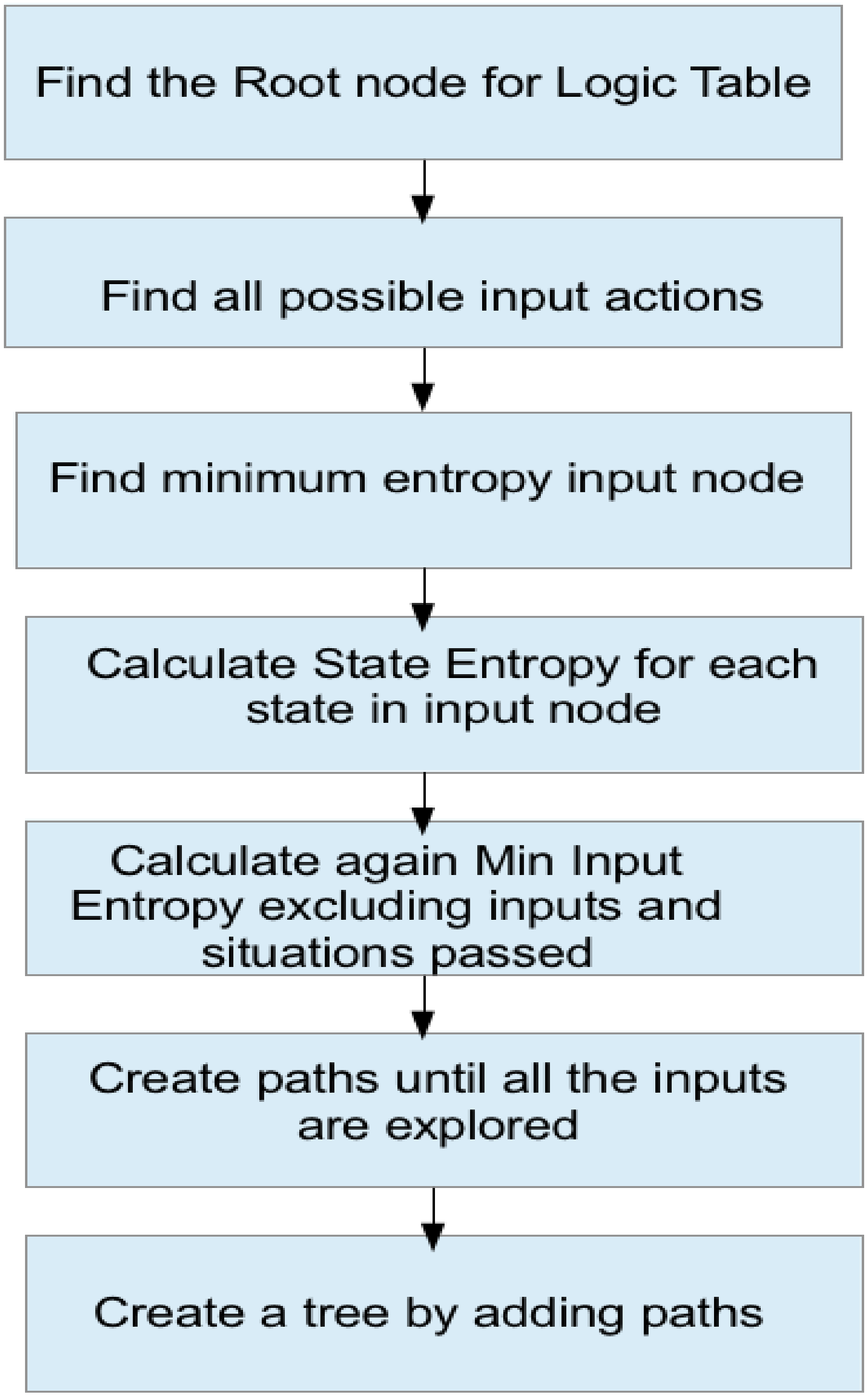

Find the Root node for Logic Table
Find all possible input actions
Find minimum entropy input node
Calculate State Entropy for each state in input node
Calculate again Min Input Entropy excluding inputs and situations passed
Create paths until all the inputs are explored
Create a tree by adding paths

Figure 13: Creating Entropy Visualization tree

6.3. Entropy Calculation Algorithm Analysis

In order to calculate minimum Entropy Input node, we find the Input node with least Entropy. This is calculated using State Entropy formula. The state Entropy for an input is calculated by finding the number of active situations for that state and Situation Entropy for given state. State entropy of an input is calculated by,

$$\text{State Entropy} = \left(\frac{ActiveSituationcount}{AllSiutationsCount}\right)*(Situation\ Entropy)$$

To calculate Situation Entropy we use following formula,

$$\text{Situation Entropy} = -\sum_{Situation=0}^{n}\left(\frac{1}{X_i}\right)\log_2\left(\frac{1}{X_i}\right)$$

where x is the Active Situation Count for a given Input state, which is extracted from the Logic Editor. If we have N input states and M situations, we calculate the State Entropy for all N input nodes given M situations. The time complexity is given by O(n*m), since this is the largest term in the above Algorithm.

Following table shows execution time captured in milliseconds for varying number of Input nodes and Situations.

Total Input Nodes	15	25	29	67	92	11
Situations	6	26	28	22	24	16
Time in Milliseconds	0	3	4	5	11	1

Table 2: Execution time calculated for Entropy Calculation Algorithm

6.4. Entropy Visualization

A Node link tree layout is used for the Entropy visualization. We use the Prefuse library to generate the visualization. We follow the same steps described in Figure 11. to render the visualization, except no animation is needed for Entropy visualization. Figure 14 below shows an Entropy visualization for *airSpeedTargetSystemTable* a Logic Table from an auto pilot prototype built in ADEPT.

Example 1

The Logic Table has total of 25 states and 7 situations. The number of

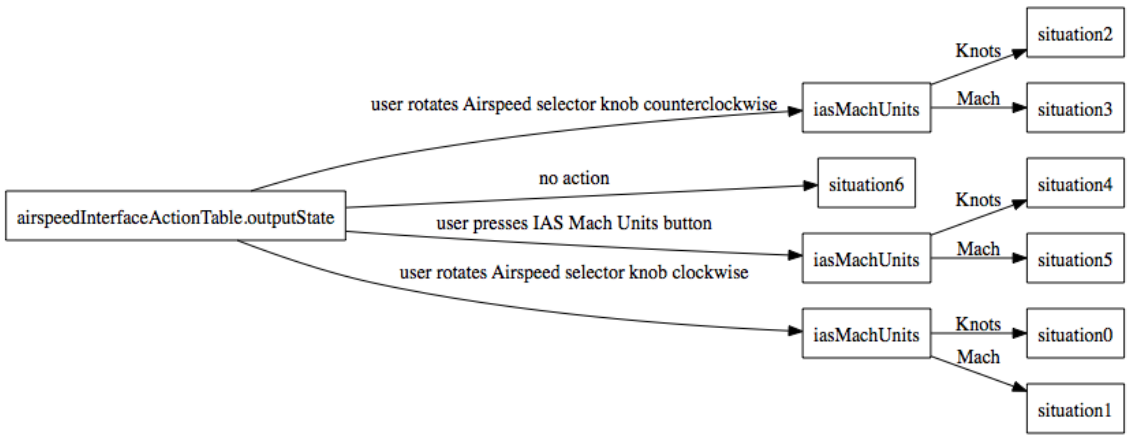

airspeedInterfaceActionTable.outputState
user rotates Airspeed selector knob counterclockwise
no action
user presses IAS Mach Units button
user rotates Airspeed selector knob clockwise
iasMachUnits
situation6
iasMachUnits
iasMachUnits
Knots
Mach
situation2
situation3
Knots
Mach
situation4
situation5
Knots
Mach
situation0
situation1

Figure 14: Entropy for Air speed target system table from an airplane design prototype

nodes and transitions appearing depends on the prototype design or the logic editor. The higher the uncertainty, higher is the entropy value for that state. We are interested only with the states having low entropy value, hence leading us to states having higher conditional chance of occurrence. We display only those states and transitions with minimum entropy.

Example 2

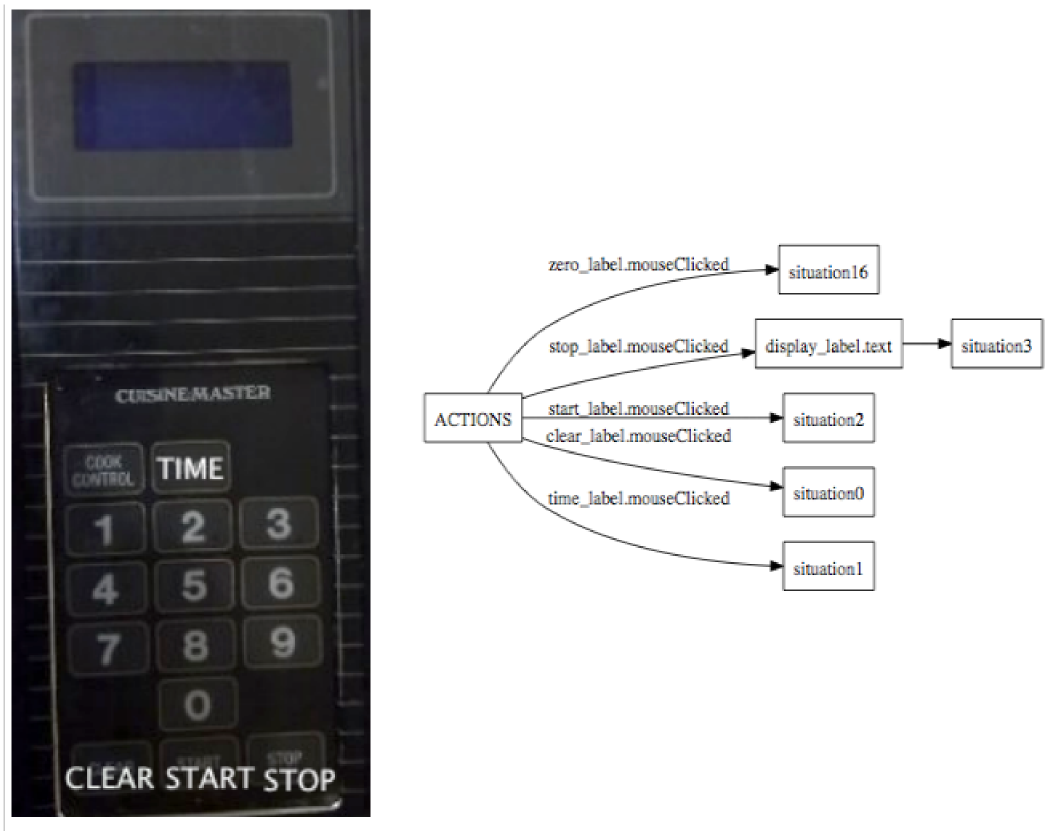


Figure 15: Entropy Visualization example for a microwave prototype design

Figure 15 above shows another example of Entropy visualization. The above prototype is for an interface of a microwave. This visualization shows very few nodes compared to the number of situations it has in the prototype design which is 26. Most of the situations were added to enter the numeric entry for time. We observe that the Entropy calculation eliminates all the situations representing time entry for each numerical key except for zero key and display all other situations and actions.

6.5. Uses of Entropy Visualization

Entropy visualization helps in eliminating all the possible transitions from the States View and shows only the states with higher importance. The Entropy visualization calculates Entropy values and helps the user on focusing on the important actions and transitions in the Logic Table. Thus reducing number of nodes and transitions to be displayed. This approach might be useful for users who are trying to understand or modify the prototype design created by someone else in ADEPT. Since they do not have to analyze each and every state or transition. The visualization helps by giving a quick overview of possible actions that can lead to a particular important situation which has a higher probability of execution. The Entropy visualization also supports zoom-in, zoom-out and zoom to fit actions.

7. Integration with ADEPT plug-in

The visualization is integrated with the ADEPT plug-in using the underlying Eclipse framework. A separate visualization view is created in the ADEPT plug-in, to display the visualizations. The view can be arranged next to the Logic Editor Table. A Listener Interface in ADEPT is used by the visualization and the XML writer class to be notified for file change events. Each time there is a change in a file the listener notifies the XML writer object as well as the visualization object. The XML writer then parses the ADEPT project file, and stores the parsed data for the visualizations in individual XML files for each visualization. In the next step the visualization class reads the data from the XML files, to create layouts and rendering for the visualizations. The View shows visualization for the current table selected in the Logic Editor Table. The states visualization can also be used with ADEPT in TEST mode. In ADEPT the user can evaluate and test a prototype design using the TEST mode. States view is integrated with ADEPT such that it can dynamically receive node change events. States view shows dynamic transitions by highlighting the nodes, when the prototype is in the TEST mode.

8. Conclusion

The project's main goal was to improve the user interface of ADEPT. We improved the adept user interface by adding new components to the ADEPT UI and the visualizations helped collaborators gain more insight of the prototypes being developed on ADEPT. The initial implementation of nebula grid widget for Logic Editor Table was successful and very critical for the project. The grid now supports drag and drop functionality for all user controls. The column focus control used for column selection helps users easily distinguish the column selection operation from a row selection operation. The System Browser was implemented again using newer widgets thus eliminating issues with drag and drop compatibility with the earlier swing version. The search feature improves the usability of System Browser by giving quick list of variable, logic table or user controls being searched. Finally, we have implemented and integrated the three visualizations into ADEPT plug-in. These visualizations help collaborators and domain experts to quickly understand logic programmed for the prototype being designed. Some complex prototypes built in ADEPT have multiple logic tables, Logic Table View is very helpful to view the activation hierarchy for such complex prototypes. The activation hierarchy is a key factor to understand a multiple logic table prototype, as it shows how the specified logic is executed for individual logic tables. States visualization displays all the states and transitions for selected table in the Logic Editor. The main feature for States visualization view is use of aggregates and animation. Aggregates helps users to focus on a selected group of interested situations or states. Animation helps by hiding or showing only few nodes of interest. States visualization is also helpful for debugging a project. The state and transition is highlighted to indicate a transition in state. Finally, we use Entropy visualization to display situations and actions with higher importance. Entropy visualization is a subset of States visualization and shows only the nodes with higher importance. We calculate Entropy values for all nodes using their conditional probabilities. Entropy visualization helps the user to identify and focus on nodes which are relatively important in a prototype.

9. References

[1] Michael Feary.(2010), *'A toolset for supporting iterative human automation interaction in design'*, *NASA Ames Research Center, Tech. Rep. 20100012861*.

[2] Jeff McAffer and Jean-Michel Lemieux.(Oct 2005). *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications. Addison-Wesley Professional.*

[3] Eclipse RCP Tutorial
http://www.vogella.com/articles/EclipseRCP/article.html

[4] Heer, J., S.K. Card, J.A. Landay. prefuse: A Toolkit for Interactive Information Visualization. *CHI 2005,* Portland, OR, 421-430.

[5] Nebula project

http://www.eclipse.org/nebula/

[6] Nebula project wiki

http://wiki.eclipse.org/Nebula/restructure#Nebula_Project_Restructuring_Proposal

[7] Prefuse

http://www.prefuse.org/

[8] Chi, E. H. (2000). A taxonomy of visualization techniques using the data state reference model. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on* (pp. 69-75). IEEE.

[9] Heer, J. ; Card, S. K. DOITrees revisited: scalable, space-constrained visualization of hierarchical data. Proceedings of the Working Conference on Advanced Visual Interfaces 2004; 2004 May 25-28; Gallipoli; Italy. NY: ACM Press; 2004; 421-424.

[10] Prefuse API Documentation
http://prefuse.org/doc/api/

[11] Ihara, Shunsuke (1993). *Information theory for continuous systems*. World Scientific. p. 2. ISBN 978-981-02-0985-8.

[12] http://en.wikipedia.org/wiki/Entropy_(information_theory)

[13] Robert M. Gray(July 2009) , *Entropy and Information Theory*. Springer-Verlag

[14] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, *5*(1), 3-55.