

# ***Spider Control***

***By Duncan Strand***

*dstrand@zebs.org.uk*

## Table of Contents

TABLE OF CONTENTS.....	2
INTERFACE DESIGN.....	3
OBJECT-ORIENTED DESIGN .....	4
JAppFrame .....	4
JEditTree.....	4
JSpider.....	4
JSpiderNode.....	4
CONCLUSION.....	5
APPENDIX A SOURCE CODE.....	6

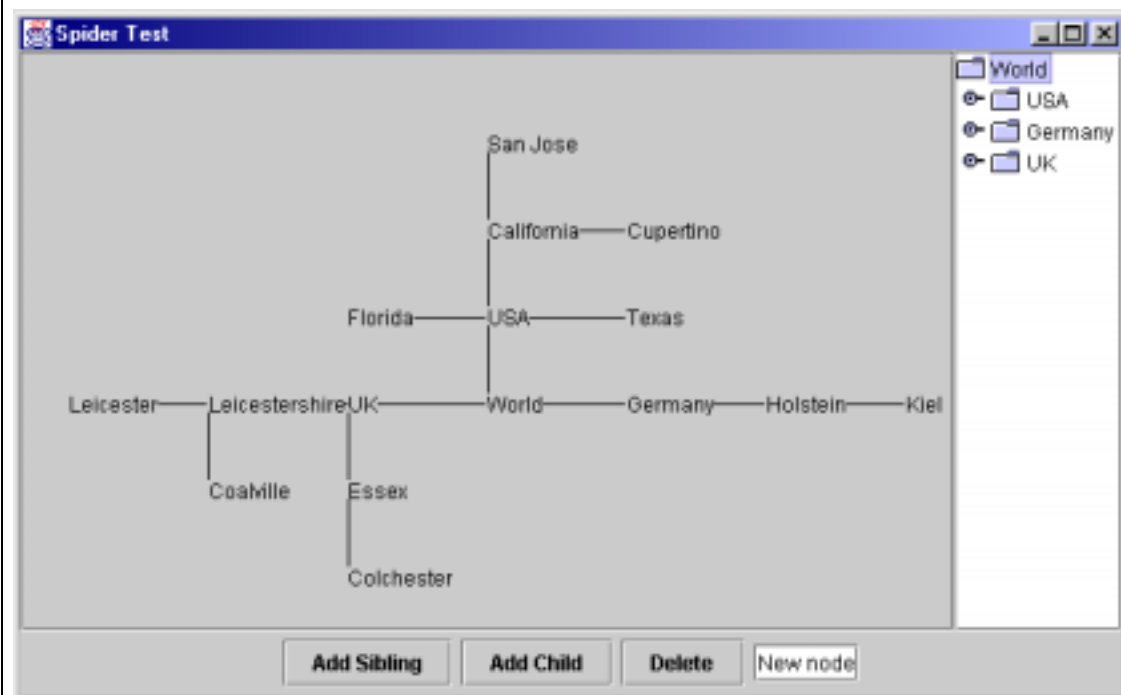
## Interface design

My main aim with the interface was to keep it simple and ensure that the data was clearly presented to the user.

That said there is a big difference between this component and how it should ideally work. There are several issues with this component:

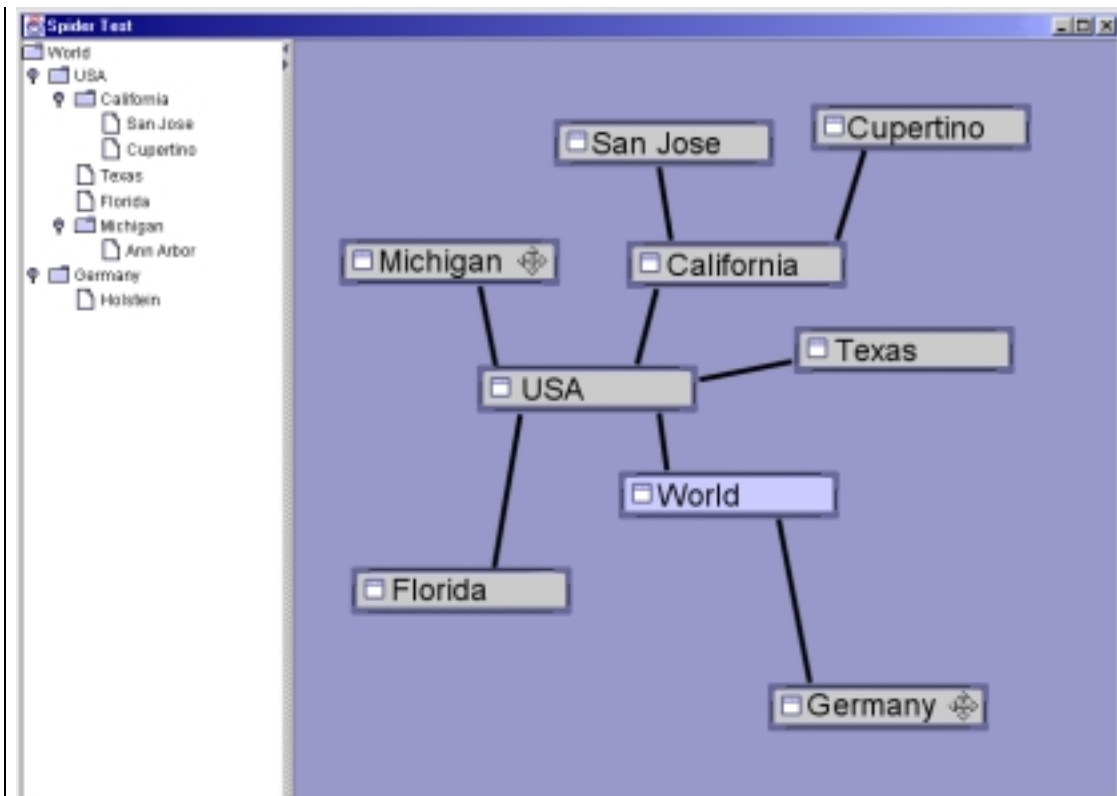
- With the exception of the root node, all nodes will only display 3 child nodes.
- It can get stuck in an endless loop looking for a suitable position to draw a child node.
- It's not interactive. If the user wants to change the display they can't, the only way the user can change the contents of the control is via the tree control.

The spider control can be seen in the image below. It uses a Vector to store the positions where each node is drawn to ensure that no two nodes are drawn on top of each other. As mentioned this control can get stuck looking for somewhere to draw a new node. In the image added a new child now to the UK node would cause this. This is because there is nowhere for the node to be drawn, with a little extra time the control could be expanded to choose a random position when this circumstance occurs and draw the node there.



If the tree model becomes too deep – that is any node has a large number of children then the control will still draw them but they will not be displayed until the user has resized the program's main window.

Ideally the user should have the ability to manipulate the appearance of the control and the data it contains. This mock-up image helps illustrate what would be a good spider control.



In this picture, on the left is the tree control, and on the right the spider control. The tree has been expanded to show all the data in the tree model. However the spider control hasn't. The child nodes of "Germany" and "Michigan" have not been displayed. In this hypothetical control any node with un-displayed child nodes have a little graphic indicating that there is more data to be displayed. The root node is highlighted in blue.

## Object-Oriented Design

This spider control is very simplistic and uses a small number of classes, four in total.

### *JAppFrame*

This class extends the JFrame class and implements the ActionListener interface. The user interface is created and managed by this class. It created the tree model and when creating the spider and tree controls passes the model to the constructors.

### *JEditTree*

This class extends JEditTree and implements the ActionListener, and TreeModelListener interfaces. Currently this class serves no purpose, but could potentially be expanded to customise the manner in which the tree control operates – possibly even to allow the tree control to be manipulated by the spider control.

### *JSpider*

This class extends the JComponent class and implements the ActionListener, and TreeModelListener interfaces.

### *JSpiderNode*

This class draws the nodes text to the display; it could be expanded to draw the lines linking the node and its parent.

## Conclusion

With a great understanding of the JFC (Java foundation Classes) of which Swing is part, a much more capable spider control could be developed as I have shown in the body of this report.

# ***Appendix A***

## ***Source Code***