Software Specification

Dynamic Visualization of State Transitions

Alex Aalbertsberg (s1008129)

February 1, 2015

Contents

1 Introduction

This document will describe a software system design for an application that can dynamically receive information from applications that generate state diagrams about transitions that take place during a simulation. The application will take the information about these transitions and visually represent them, so as to create a more clear full picture of the situation for the end user.

There are several tools in existence that generate state diagrams according to complex grammars/alphabets. These diagrams are often not as intuitive to read for an end user, whereas a graphical representation of the situation would provide a lot of clarity as to what the diagram actually does. Therefore, the idea behind this application is to visually represent the current situation of a state diagram as it runs through simulation. This should help the end user get a clear picture of what is actually happening, as opposed to merely seeing the relatively unintelligible state transitions.

2 Requirements

The final product must:

1. Be able to create an initial visual representation of a state diagram from the tool that generated said diagram.

2. Receive information about state transition steps from any state diagram tool.

3. Interpret the received information and translate the abstract syntax that is received to a graphical representation AKA the concrete syntax.

4. Support a preset schema of commands and respective parameters that allows client applications to perform GUI operations according to what happens in the state transitions.

5. Update the visual representation (concrete syntax) according to the transition information that has been received.

3 Technology

3.1 Design Choices

This section will describe the technology that has been used to create the final product. The decisions to utilize the technology listed below were made during the early stages of the project.

Criteria	Java	C#
GUI Support	Yes, in Java 8 (JavaFX)	Yes
Networking Support	Yes	Yes
Multi-OS	Yes	Limited*

Table 1: Comparison of Programming Languages

Based on the above criteria, the decision was made to write the application in Java 8 and uses the native library JavaFX for user interfacing. Java has been chosen in order to maintain as much scalability as possible.

Criteria	JSON	CSV	XML
Overhead	Small	Small	Large
Language Support	Open-source	None	In-built
Parseability	Easy	Medium	Easy

Table 2: Comparison of Transport Protocols

Based on the above criteria, the decision

was made to let communication take place by sending JSON objects over an established TCP connection. XML's large overhead and no support existing for parsing CSV data left JSON to be the best option. The application therefore adopts the client-server paradigm as its communication protocol. This means that client applications (i.e. the application simulating the state diagram) will connect to the server application and send JSON objects containing information about the steps that have been taken, so that the server application may perform actions that correspond with the received information. You will find a description of currently supported commands and their parameters below. The library that has been used for JSON is the one from www.json.org. Their source code for supporting JSON is freely available on their website.

3.2 Application Design Specifics

The server application itself will consist of several classes:

1. The Main class. This class initializes the GUI and TCP Server, so that commands may be interpreted and updated on the user interface.
2. The CommandProcessor class. This class deciphers the JSON data sent to the TCP server, interprets and then executes the specified command.
3. The AbsCommand (abstract) class. This class is a framework for the implementation of any command. It only contains a constructor that requires a JSONObject, and a performCommand method that will execute whatever the JSON data tells it to do.
4. All classes that specify a certain command derive from the AbsCommand class.
5. The SocketListener class. This class listens for incoming data on a certain port and sends the received information through to the CommandProcessor.
6. The JSONObject class. This class encapsulates an associative set of objects that can easily be sent over a TCP connection.

The client application will have to connect to the server application's IP address and port number on which it is listening. Subsequently, the client will be able to send JSONObjects over this established connection, which the server application will parse. The parsed parameters should then lead to a desired result on the GUI.

3.3 Commands and Parameters

Several initial commands and their parameters that are supported by the application are as follows:

1. init. This command allows the user to set initial GUI/application values.
   (a) GUI width.
   (b) GUI height.
   (c) Element scale.
2. set. This command allows the user to create elements on the GUI.
   (a) Identifier.
   (b) X-axis position.
   (c) Y-axis position.
   (d) Width.
   (e) Height.
   (f) Color.
   (g) Shape.
3. move. This command allows the user to move existing GUI elements.

   (a) Identifier.
   (b) New X-axis position.
   (c) New Y-axis position.

4. remove. This command allows the user to remove an existing GUI element.
   (a) Identifier.

4 Test results

For the sake of properly testing the final product, a couple of (commonly used) diagrams will be defined for testing purposes:

1. The wolf-goat-cabbage problem
2. An elevator with x floors
3. Dining philosophers
4. Christmas tree

4.1 Wolf-Goat-Cabbage Problem

This problem deals with the following: A man is standing on the bank of a river with a wolf, a goat and a cabbage. The man wants to cross the river with all three of them. However, his boat will only allow him to take one of the three with him at a time. The other two will have to wait. When the man is not there, the wolf is capable of eating the goat, and the goat is capable of eating the cabbage.

For this problem, both a success and a failure scenario have been created. These scenarios seem to work fine. Of course, the success scenario first places the goat on the other bank, as the wolf and the cabbage share no interaction that would hinder progress. The man could then take either the wolf or the cabbage to the opposite bank (it does not matter which one) and takes the goat back to the first bank. Then, the man will take the wolf or the cabbage (whichever he did not take with him previously). Now, the goat will be the only one left on the first bank, allowing a simple cross to complete the problem.

5 Related work

JTorX - a tool for Model-Based Testing.

Other related work includes Visualization of State Transition Graphs by van Ham et al. at the University of Eindhoven. This research pertained to modelling state diagrams in 3D, so that users could get a better view of the overall structure of such a diagram.