VMWare Virtual Network Visualizer

Software Design Document

Version 2.0

October 25, 2012

Team
–Intentionally Left Blank–

Pierce Trey, Everett Bloch,
Jonathan Lamb, Daniel Palmersheim

Prepared For
UI CS 383—Software Engineering I
Instructor: Paul Oman, Ph.D.
Fall 2012

SDD template acquired from Jacksonville State University

Table of Contents

List of Figures

1. Introduction

1.1 Purpose

The purpose of this document is to lay out a detailed description of the technical design of team Intentionally Left Blank's VMWare Virtual Network Visualizer. This includes the design of the data structures, the methods used for accessing and manipulating these structures and the Graphical User Interface that provides the useful, visual representation of the virtual network to the user—the ultimate purpose of this program.

1.2 Scope

This project is being developed using the Spiral Waterfall design methodology. This product utilizes the Java Run-time Environment to provide a multi-platform application that creates a graph of a virtual network with the aid of the Java Universal Network/Graph library. There is an underlying skeleton of code that ties together the reading in of information about the virtual network, the storage of that information in a system of nodes and edges and the display of this information on a screen with the ability for the user to interact with it.

1.3 Glossary

ILB - Intentionally Left Blank

GUI – Graphical User Interface

JDK – Java Developers Kit

JRE – Java Run-time Environment

JUNG – Java Universal Network/Graph library.

XML – Extensible Markup Language

SRS - Software Requirements Specification

IP - Internet Protocol

Node - A vertex on a graph that corresponds to a machine in a network

Edge - A relationship between two nodes (connectivity)

Weight - The distance between two nodes (length of an edge)

Capacity - The bandwidth/speed of an edge

1.4 References

Software Requirement Specification (SRS) - Intentionally Left Blank

1.5 Overview of Document

The remainder of this document contains the technical description of the systems that this application can be deployed on, the structure of the underlying program code, data structures and GUI, as well as a description of how this program meets all of the required specifications from the SRS.

2.0 Deployment Diagram

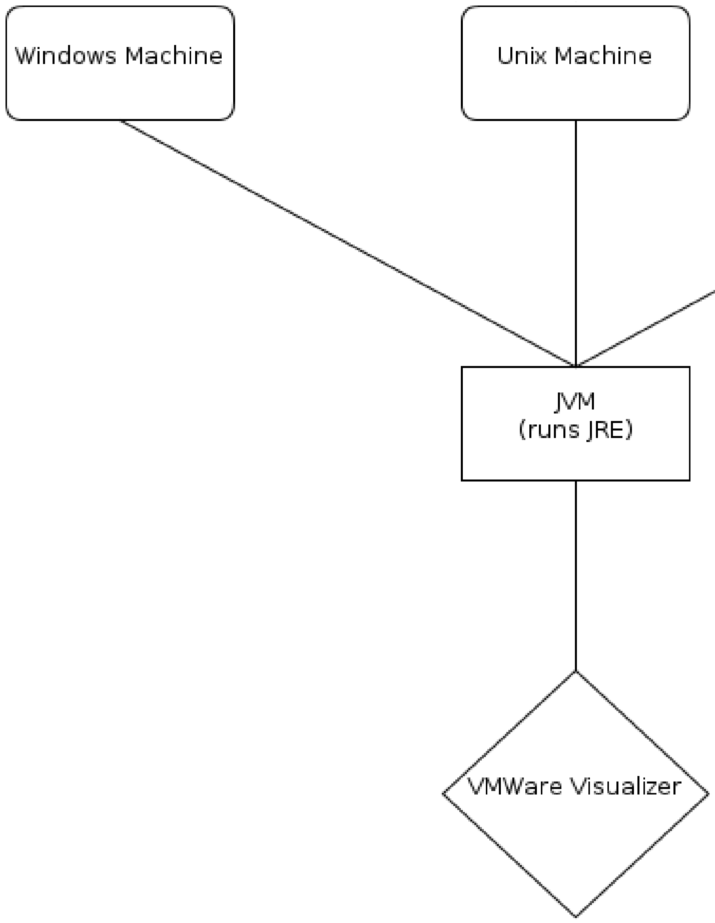

Figure 2.1 - Deployment Diagram

This software is capable of running on any system with JRE 7+ running on it. This allows for increased portability, and reliable, consistent operation on any of these systems. The product will be packaged as a .jar file along with all necessary external libraries. Copies of the documentation shall also reside in this zipped distribution.

3.0 Architectural Design

3.1 Data Logger Class

This object is a way to log to a file from all classes. This should make debugging easier as it isn't very helpful to print things to stdout. messages are appended to the log file. Their class name and message level are appended to a timestamp use the appropriate level for your log info/debugging/testing: severe, warning, and info. In addition this Class provides a session log which provides logged information of the current session. This acts as a loading screen upon start up, and a log window for the user during run time.

- *Severe Messages*: These messages are meant to log critical errors in the code. This method accepts a string describing the name of the class that this error occurred in, as well as a description of the error.
- *Warning Messages*: These messages are meant to log errors that may or may not cause a critical fault in the system. The purpose of these warnings is to alert the coder of potential problems during the design and testing phase of development.
- *Info Messages*: These messages are meant to provide general information about correctly working procedures. These messages are useful for ensuring proper functionality during testing, and they may assist in narrowing down the location of an errors by eliminating properly working functionality.

3.2 Network Data Payload Class

This class is responsible for pre-processing the input data in the form of a node IP address, a list of neighboring IP addresses, and lists of capacities and weights corresponding to the list of neighboring IP addresses stored within an xml structure. The purpose of this class is to construct a representation of edges and nodes in such a way that back-linking is prevented, as well as to make available the data in such a way that it can simply be be piped in to a Graph object. Each element will contain, at minimum, those properties. This class provides a method to input each element iteratively. Upon completion of adding nodes, this class will then, upon command, process each path/edge into a linked list as well as generate all necessary nodes, again into a linked list. Once that has successfully completed, the data stored in the class can then be piped into a Graph object, with the supplied method. The linked list is used to keep a dynamic list of edges, which in turn are used to keep track of which nodes need to be created in order to prevent two nodes from linking to each other, or back-linking. Note that if anything fails during any of these processes the program will log the error and terminate. This is the only non-fail safe step in the pipeline, which is due to the critical necessity of the expected output to the graph.

3.3 Network Node Class

This class is used for the JUNG Graph Class to specify the node information. The data fields inside each node store information about the nodes on the network, such as the node ID, the IP address of the node, and a list of nodes connected to the current node. The class contains a constructor that takes the node ID as well as the IP address of the node and stores it in a new instance of the class, as well as a method that adds to the node's list of neighbors. The class also contains various string methods that return the data contained in the node for GUI display purposes.

3.4 Network Path Class

This class is used for the JUNG Graph Class to specify the path information. The data fields inside each path store information about the ID, capacity and weight of the path. The class contains a constructor that takes the path ID as well as the capacity and weight of the path and stores it in a new instance of the class, as well as methods that allow for the changing of the capacity and weight, as well as methods that allow for retrieving the information about these data fields (both for algorithmic calculations and GUI display purposes).

3.5 User Interface Class

This class contains all of the variables and methods that create the underlying graph from the input file, and display the visual representation of that graph using the JUNG framework. This includes inserting the edges and nodes into JUNG's built-in data structures as well as the physical layout and operations of the GUI. This class also contains methods to perform both Weighted and Unweighted shortest path algorithms over selected nodes in the network graph for analysis purposes.

3.6 Entry Class

This class contains is the main method of the program that initializes the process of the VMWare Virtual Network Visualizer. This takes care of instantiating the Data Logger, Network Data Payload, and User Interface (GUI) Classes.

3.7 XMLSAXParser Class

This class is used for opening and processing the input file containing the network configuration to be visualized. This class uses the SAX xml parser that is built into the java libraries.

4.0 Use Case Realizations

4.1 Network Surveying (addressed in SRS 3.3.1):

The visual graph that is produced within this program is used to realize this use-case. The JUNG framework allows the user to not only see a nicely laid-out graph, but it also allows for them to manually manipulate how the graph is viewed by rotating, zooming, skewing, highlighting and moving individual nodes. This ability to manipulate the graph allows for the customization of each graph for each intended surveying purpose. The ability to export graphs (in their current, manipulated state) allows for the future presentation of analyses performed on networks by taking snapshots of the graph at different stages in order to analyze the effects of potential changes to the virtual network for research or other purposes.

4.2 Network Management (addressed in SRS 3.3.2):

Aside from the visual aspect of analyzing a network as mentioned in the previous section (which plays a significant part in managing and improving a network), the algorithmic capabilities of this program are of special use in managing a virtual network. Finding the shortest path between two nodes on the network can allow for the optimization of the structure of a network so two nodes that are farther away from each other on the network don't experience poor service times due to the network's makeup. Ensuring all nodes are as efficiently connected as possible can aid in the overall efficiency of the network.

5.0 User Interface Design

The GUI for the Visualizer was created to be sensible and straightforward. The main functionality is found on the main screen, utilizing clearly labeled buttons and menus. A menu bar at the top provides access to extra functionality that would have been cumbersome to have in the main window. The program is full-screen by default and components within the GUI are designed to be re-sized as is appropriate with the window. The program was designed to adhere to a typical design you might see with a consumer product, so that the user has the potential to figure out all of the features without having to consult some sort of user manual first.

6.0 Help System Design

The program help system shall consist of a pdf version of the user manual. This is to be accessible from within the program (via a standard pdf viewer), and searchable (as pdfs are searchable).

This Page Intentionally Left Blank