CSET 3600 Project – Group 5

GUI for [V-NetLab](http://www.eng.utoledo.edu/~wsun/papers/ncisse05.pdf) Integration

Group members:

1. Taylor Hunt
2. Benjamin Kania
3. Sean Morris
4. Steven Murphy
5. Jeremy Ziehr

Table of Contents

[2 Problem Definition 4](#_Toc446359207)

[3 Background 4](#_Toc446359208)

[4 Group Introduction 5](#_Toc446359209)

[5 Plan 6](#_Toc446359210)

[6 CSET 3600 Use Cases – Group 5 7](#_Toc446359211)

[6.1 Use Case 1 8](#_Toc446359212)

[6.2 Use Case 2 10](#_Toc446359213)

[6.3 Use Case 3 11](#_Toc446359214)

[6.4 Use Case 4 12](#_Toc446359215)

[6.5 Use Case 5 13](#_Toc446359216)

[7 Project Plan 14](#_Toc446359217)

[8 UML Diagram 15](#_Toc446359218)

[9 Task List 16](#_Toc446359219)

[10 Code Review (Preliminary) 17](#_Toc446359220)

[10.1 Support Environment 17](#_Toc446359221)

[10.2 Architecture & Implementation 17](#_Toc446359222)

[10.3 Code Description 17](#_Toc446359223)

[10.3.1 Main Function 17](#_Toc446359224)

[10.3.2 VM Class 17](#_Toc446359225)

[10.3.3 HUB Class 17](#_Toc446359226)

[10.3.4 Data Class 18](#_Toc446359227)

[10.3.5 FileParser Class 18](#_Toc446359228)

[10.3.6 RootLayout FXML & Controller Class 18](#_Toc446359229)

[10.3.7 GuiOverview FXML & Controller Class 18](#_Toc446359230)

[10.3.8 VMEditDialog FXML, HUBEditDialog FXML, and Controller Class 18](#_Toc446359231)

[10.3.9 GUI FXML & Controller Class 18](#_Toc446359232)

[10.4 Code Ownership 18](#_Toc446359233)

[10.5 Future Work 19](#_Toc446359234)

CSET 3600 Project Proposal – Group 5

# Problem Definition

The project will implement a graphical network configurator using for [V-NetLab](http://www.eng.utoledo.edu/~wsun/papers/ncisse05.pdf), a tool for creating isolated virtual networks for network experiments. The project goal is to design and develop a graphical user interface (GUI) network configurator. The GUI will allow users to create a network topology, and specify parameters for the hosts and network devices (i.e. hubs). Users of the GUI should be able to save network configurations with the ability to open and continue working on the configuration. Users of the GUI should also have the ability to check for correctness of the network specifications. Once correctness has been verified, the user will have the ability to save the specification into a configuration file for V-NetLab to read.

# Background

V-NetLab was developed to make to logically isolate virtual networks of computers and simplify administration by automating the startup and shutdown of the virtual network. This improvement was created due to the physical limitations of settings up a dedicated physical isolated network for each student to experiment and test while learning. V-NetLab realized six to eight dedicated virtual computers for each student, upwards to fifty students, on inexpensive hardware.

The development environment is required to be an object-oriented programming language with Java being the preferred language.

Group 5 will leverage GitHub to communicate and version control all code and documents. We will also leverage the Blackboard group site for additional collaboration when needed.

# Group Introduction

Group 5 has five members; Taylor Hunt, Benjamin Kania, Sean Morris, Steven Murphy, and Jeremy Ziehr.

**Taylor Hunt** is a junior in the Computer Science and Engineering program at University of Toledo with an interest in databases.

**Benjamin Kania** is a senior in the Computer Science and Engineering Technology program at University of Toledo. Benjamin has used a variety of programming languages and has an interest in programming and running PC games.

**Sean Morris** is a senior in the Computer Science and Engineering Technology program at University of Toledo. Sean has experience with many scripting languages like bourne shell, korn shell, sed\awk, and PowerShell along with experience in object-oriented languages like C++ and Java in an academic environment.

**Steven Murphy** is a senior in the Computer Science and Engineering Technology program at University of Toledo. Steven has experience with HTML and PHP programming languages in addition to object- oriented languages C++ and Java.

**Jeremy Ziehr** is a senior in the Computer Science and Engineering Technology program at University of Toledo.

# Plan

High-level Milestones:



|  |  |  |
| --- | --- | --- |
| **PROJECT DETAILS** | |  |
| **DATE** | **Owner** | **MILESTONE** |
| 1/21/2016 | N\A | Project Start |
| 2/5/2016 | Sean | Project Proposal |
| 2/15/2016 | Steven | Use Cases |
| 2/26/2016 | Sean | Project Plan |
| 3/7/2016 | Jeremy | UML |
| 3/14/2016 | Taylor | Task List |
| 3/21/2016 | Group | First Code Review |
| 4/1/2016 | Ben | Test Suite |
| 4/8/2016 | Taylor | Updated Task List |
| 4/15/2016 | Group | Second Code Review |
| 4/25/2016 | Group | Final Report \ Code |
| 4/29/2016 | N\A | Project End |

A detailed task list with associated owners will be developed with the Project Plan.

# CSET 3600 Use Cases – Group 5

Use Cases can be thought of as a collection of possible scenarios related to a particular goal. The cases can be used to model the goals of the actor’s interactions and the systems response. Each of our use cases will have their own page or more and the following format:

1. **Use Case**: Title for the case
2. **Actors**: The person, group, or system that will the case has be implemented for
3. **Goal**: Outcome expected based on the Case
4. **Preconditions**: What has happened to cause the Case
5. **Trigger**: The action which causes the Case
6. **Scenario:** The written outline of the Use Case
7. **Exceptions:** Something that could cause a different outcome or error from the expected one.

The Use Cases below are an initial design for the project and it should be noted this is a living document. As this project progresses through each step to completion they will most likely change or be added too.

1. User Starts from an existing Design/File
2. User Starts a New Network Design
3. User Adds or Deletes a Switch to the Panel
4. User Adds or Deletes a Computer to the Panel
5. User Saves a Network Design to a file

## Use Case 1

**Use Case 1**: *User Starts from an existing Design/File*

**Actors:** GUI User

**Goal:** The user will be able open and continue from a previously saved file for a V-Net Lab application.

**Preconditions:** The user wants to add to a previous design.

**Trigger:** The user selects button to open the file.

**Scenario:**

1. The System displays the general GUI with a “button” to select for an existing V-Net design.
2. The user selects a button for opening an existing file.
3. File format is a “.txt” file.
4. As long as the file exists the system will verify that the format is correct and its setup is without errors.
5. The network will be named after the file’s name.
6. The network IP and subnet’s are verified to be the correct
7. The networks design will be populated on the panel. All connections and images will be represented.
8. The user will then be able to add, subtract, and save a new file for their network design (Described in Use Case 3, 4, and 5)

**Exceptions:**

1. The file is not of the correct format .txt. The system should check this and notify the user of the error and not proceed with the mapping of the network to the GUI’s panel.
2. If the file setup is not in the correct design and will generate errors when setting up the network. User is notified that there are errors in the design and nothing is put on the GUI’s panel.
3. Errors in the networks IP or subnet addresses the user is notified of the errors and nothing is put on the GUI’s panel

## Use Case 2

**Use Case 2**: *User Starts a New Network Design*

**Actors:** GUI User

**Goal:** The user will be able to start a new design.

**Preconditions:** The user wants to add to a previous design.

**Trigger:** The user opens the file.

**Scenario:**

1. The System displays the general GUI with a button to start a “New design”.
2. The user selects a button for the “New Design”.
3. System will prompt the user with the network starting private IP address and subnet mask.
4. The user will be prompted for a General name of the network.
5. If the user gives a proper private IP address and subnet mask the panel will now be opened to the user.
6. Switches, Hubs and Computer terminals can be added or deleted from the GUI’s Panel (Covered in Use Cases 3 and 4).
7. When finished or upon closing the design will be saved (Described in Use Case 5).

**Exceptions:**

1. If there is a design currently on the GUI’s panel then it is saved to a file before a new design is started.
2. If the IP address is incorrect and it is not a private address then the user is notified of the error and is not allowed to continue without correcting the error.

## Use Case 3

**Use Case 3**: *User Adds or Deletes a Switch on the GUI’s Panel*

**Actors:** GUI User

**Goal:** The user will be able to add or delete a switch

**Preconditions:** The user want to add switch to a new or previous design.

**Trigger:** The user drags a new switch to the panel

**Scenario:**

1. The System displays the general GUI with a button to add a “New Switch”.
2. The user selects a button for the “New Switch” then drags it to their wanted location.
3. System will prompt the user on what IP address and subnet mask the switch should have.
4. If the user gives a proper private IP address and subnet mask the system will allow any connections to be made.
5. Only connections that will be allowed will be within the proper IP and subnet mask
6. If the user wants to delete then when they select a previously added switch and then select a button for delete.
7. Upon deleting a switch image, IP address, and connections are removed from the panel. Both connections and IP addresses will now be available for future switches.
8. When finished or upon closing the design will be saved (Described in Use Case 5).

**Exceptions:**

1. If the IP address or Subnet is incorrect then the user is notified of the error and is not allowed to continue without correcting the error.

## Use Case 4

**Use Case 4**: *User Adds or Deletes a Computer on the GUI’s Panel*

**Actors:** GUI User

**Goal:** The user will be able to add or delete a computer

**Preconditions:** The user want to add a computer network to a design.

**Trigger:** The user drags a new computer to the panel

**Scenario:**

1. The System displays the general GUI with a button to add a “New Computer”.
2. The user selects a button for the “New Computer” then drags it to their wanted location.
3. System will prompt the user on what IP address and subnet mask the switch should have.
4. If the user gives a proper private IP address and subnet mask the system will allow any connections to be made.
5. Only connections that will be allowed will be within the proper IP and subnet mask
6. If the user wants to delete a computer, they select a previously added computer then the button to delete.
7. Upon deleting a computer image, IP address, and connections are removed from the panel. Both connections and IP addresses will now be available for future connections.
8. When finished or upon closing the design will be saved (Described in Use Case 5).

**Exceptions:**

1. If there is a design currently on the GUI’s panel then it is saved to a file before a new design is started.
2. If the IP address is incorrect and it is not a private one then the user is notified of the error and is not allowed to continue without correcting the error.

## Use Case 5

**Use Case 5**: *User Saves a Network Design to a file*

**Actors:** GUI User

**Goal:** The user want’s to save their network design

**Preconditions:** The user has developed a new or made changes to an old design

**Trigger:** The user closes or selects to save current design

**Scenario:**

1. The System displays the general GUI with a button to “Save”.
2. The user selects a button for “Save” or selects to close the GUI.
3. System will now save the file based on the name give when creating the network design. (Refer to Use Case 2)
4. If the user just choses to close the file is saved automatically to the network name given.
5. The file saved will be in line with the given format for V-Net Lab applications

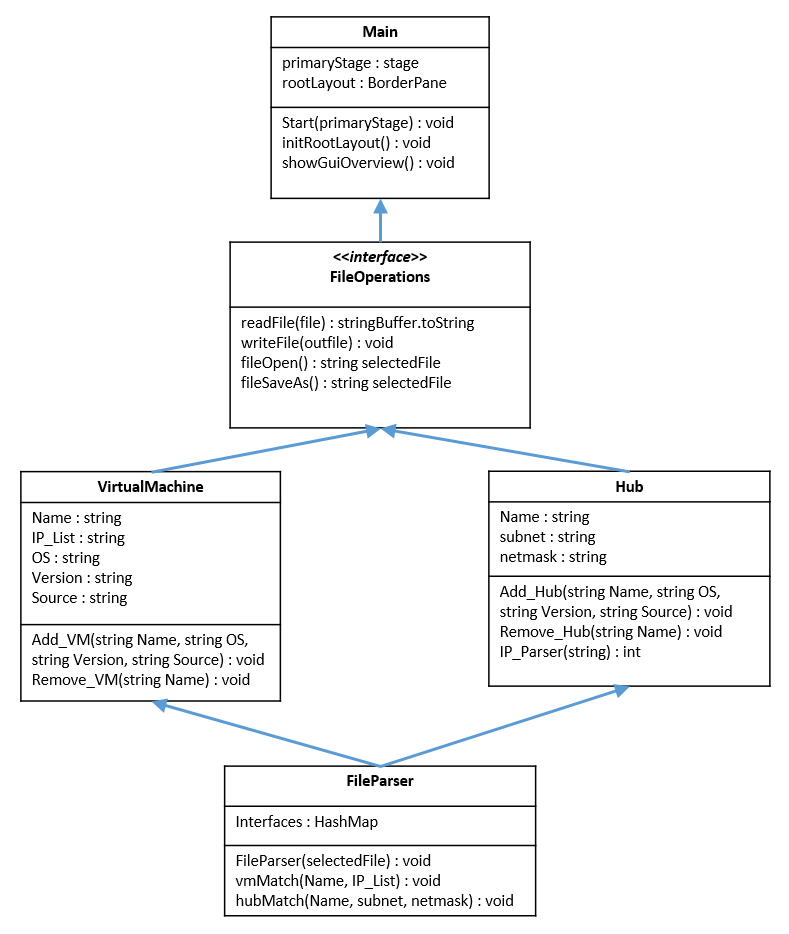
**Exceptions:**

1. If there is a name currently saved in the folder then the file is over written

# Project Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Start** | **End** | **Assigned To** | **Description** |
| 1/21/2016 | 1/21/2016 | Team | Project Kick-off |
| 1/21/2016 | 2/5/2016 | Sean | Project Proposal |
| 2/5/2016 | 2/15/2016 | Steve | Develop Use Cases |
| 2/5/2016 | 3/7/2016 | Sean | Design & Develop GUI layout |
| 2/19/2016 | 3/14/2016 | Steve | Design & Develop logic for network devices |
| 2/26/2016 | 3/7/2016 | Jeremy | Develop UML for project |
| 3/1/2016 | 3/7/2016 | Taylor | Design & Develop logic for reading and writing configuration file |
| 3/7/2016 | 3/14/2016 | Taylor | Develop Task List for team |
| 3/7/2016 | 3/21/2016 | Steve, Taylor, Sean | Integrate different methods for complete prototype |
| 3/14/2016 | 4/1/2016 | Ben | Develop Test Suite |
| 3/21/2016 | 3/21/2016 | Team | First Code Review |
| 3/21/2016 | 4/1/2016 | Steve, Taylor, Sean | Update code based on First Code Review feedback |
| 4/1/2016 | 4/8/2016 | Taylor | Update Task List |
| 4/1/2016 | 4/8/2016 | Ben, Jeremy, Taylor | Perform Testing based on Test Suite |
| 4/1/2016 | 4/25/2016 | Sean | Design graphic elements for GUI |
| 4/8/2016 | 4/25/2016 | Steve, Taylor, Sean | Update code based on Test Suite results |
| 4/15/2016 | 4/15/2016 | Team | Second Code Review |
| 4/15/2016 | 4/25/2016 | Steve, Taylor, Sean | Update code based on Second Code Review feedback |
| 4/15/2016 | 4/25/2016 | Team | Final Report \ File Code submitted |
| 4/18/2016 | 4/22/2016 | Ben, Jeremy, Taylor | Perform Final Testing based on Test Suite |
| 4/29/2016 | 4/29/2016 | Team | Project End |

# UML Diagram



# Task List

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Project Tracker | |  |  |  |  |
| [Click to view the Setup sheet.](#Setup!A1) |  |  |  |  |  |
|  |  |  |  |  |  |
| Task | Category | Assigned To | Start | Finish | Notes |
| Task 1 | Completed & Submitted UML | Jeremy | 2/26/2016 | 3/8/2016 | Jeremy submitted as well as sent an email copy to the group |
| Task 2 | Complete prototype | Steve, Taylor, Sean | 3/7/2016 | 3/21/2016 | Need to setup a time to meet |
| Task 3 | Develop Test Suite | Ben | 3/14/2016 | 4/1/2016 |  |
| Task 4 | Code Review | All | 3/21/2016 | 3/21/2016 | Will this be another skype/phone conference or in person? |
| Task 5 | Update code | Steve, Taylor, Sean | 3/21/2016 | 4/1/2016 |  |
| Task 6 | perform 1st test in Test Suite | Ben, Jeremy, Taylor | 4/1/2016 | 4/8/2016 |  |
| Task 7 | Design Graphic Elements of GUI | Sean | 4/1/2016 | 4/25/2016 |  |
| Task 8 | Update code based on 1st test | Steve, Taylor, Sean | 4/8/2013 | 4/25/2016 |  |
| Task 9 | Second code review | All | 04/15/2016 | 04/15/2016 |  |
| Task 10 | Update based on 2nd review | Steve, Taylor, Sean | 04/15/2016 | 04/25/2016 |  |
| Task 11 | Final report/file code submit | All | 04/15/2016 | 04/25/2016 |  |
| Task 12 | Final testing from test suite | Ben, Jeremy, Taylor | 04/18/2016 | 04/22/2016 |  |

|  |
| --- |
|  |

# Code Review (Preliminary)

Preliminary code review is comprised of two projects where elements of each will be used to develop the final product. 00\_Project is a prototype for a text editor user interface and aaa\_project is a prototype for user interface intended to be implemented. However, aaa\_project currently reads XML files and not the CFG file provided for reference.

## Support Environment

Netbeans 8.0.2 is the IDE used by the group for development along with JRE 8 Update 40 and above. JavaFX has been enabled in the Netbeans IDE to aide in the view (user interface) creation. SceneBuilder 8.0 is used to develop the FXML file format. More information on SceneBuilder can be found at <http://gluonhq.com/open-source/scene-builder/>.

## Architecture & Implementation

The architecture intended for this project is the Model-View-Controller (MVC) design pattern. This pattern allows us to separate the logic in each element.

In our project, the Model is represented by the Main, VM, HUB, FileParser, and Data classes. The View section represents the FXML, the graphical elements that comprise the various user interface elements. The Controller is represented by the labeled Controller files associated with each FXML file. In the project file, the Main section is contained at the root level, the VM, HUB, FileParser, and Data classes are contained in the Model subfolder, and the FXML and related Controller files are located in the View subfolder.

## Code Description

### Main Function

The code for the Main function contains logic for processing the configuration file and related input and output along with initializing the Stage or primary view for the user.

### VM Class

The VM class uses a basic getter and setter format to allow for calling or setting the appropriate value. The values are name, OS, source, version, and interfaces.

### HUB Class

The HUB class is similar to the VM class, it uses getters and setters for each value. The values in the HUB class are name, subnet, netmask, and interface.

### Data Class

The Data class is used to set define the patterns in the configuration file that make a VM or HUB. Regular expressions are used to determine the name of the device and its related information. A LinkedHashMap is used to call a node and its related information.  
\*Code logic is referenced from Garrett Rowell and used only for prototyping\*

### FileParser Class

The FileParser class reads the selected file and determines if the device is a VM or HUB using the node regular expression in the Data class. It then uses the other regular expressions in the Data class to determine the respective information and set the values for the device. There are sections for the determining the values of all the Ethernet IP’s and Interface ID’s using the regular expression lookup in the Data class.   
\*Code logic is referenced from Garrett Rowell and used on only for prototyping\*

### RootLayout FXML & Controller Class

The RootLayout FXML and related RootLayoutController file define the root scene for the user interface. It consists of the menu bar and related logic when interfacing with the menu options (i.e. handling the File, Open or Save As menu button).

### GuiOverview FXML & Controller Class

The GuiOverview and GuiOverviewController file provide the section where all devices and related information can be view and edited. The intention is for the view on the left to list every device in from the configuration file and the view on the right to display the details of the device via an ObservableList. A user is able to Add, Edit, or Delete devices on this view.

### VMEditDialog FXML, HUBEditDialog FXML, and Controller Class

The VMEditDialog, HUBEditDialog, and VMEditDialogController are the views the user interacts with when adding or editing a device. Currently there are logic issues for switching between devices but the code will allow for creating a new VM.

### GUI FXML & Controller Class

The GUI and GUIController file define the text area for the editing the configuration file.

## Code Ownership

The user interface and file parsing portions are owned by Sean Morris. The HUB, VM, and Network Interface classes are owned by Steven Murphy. Taylor Hunt is assisting both Sean and Steven with logic on method detail when appropriate.

## Future Work

Further effort is needed to build a unique parser. In addition much work is needed to identify a user interface model and related logic. The group will be meeting regularly to refine the prototype in a more mature product over the next several week. Once we have settled on a direction the UML will be updated accordingly.

There is still discussion on the best way to manage the VM and HUB information are reflected by Steven’s comments below. The current structure is using two arrays lists to track all of the objects created from the VM and HUB classes. All object additions and deletions occur from these array lists. An array list is used because of the ability to add/remove to the list without having to define the set size. The UML will be updated shortly to reflect this. For future development we will use another array list to track all of the connections. Connections are checked during each object creation if connections are found then they are entered into the list. This will allow for a user to enter several Hub’s or VM’s without the need to have it corresponding connection.

Functions in the classes that need to be added:

1. Each class will have an IP parser and verification.
   1. This will be used to verify if the format and if the IP is unique.
2. Name verification
   1. This will be used to verify if the name is unique
3. Connections check, after each object addition
   1. Each object will be checked if they have any connections