Detailed Design

Description Report
Table of Contents
1. INTRODUCTION ............................................................................................................. 4
  1.1. Problem Definition .................................................................................................. 4
  1.2. Purpose ................................................................................................................... 4
  1.3. Scope ....................................................................................................................... 4
  1.4. Overview ................................................................................................................ 4
  1.5. Definitions, Acronyms, Abbreviations ..................................................................... 5
  1.6. References .............................................................................................................. 5
2. SYSTEM OVERVIEW ...................................................................................................... 5
3. DESIGN CONSIDERATIONS ............................................................................................ 6
  3.1. Design Assumptions, Dependencies and Constraints ............................................... 6
       3.1.1. Performance Requirements .............................................................................. 6
       3.1.2. Time Constraints ............................................................................................ 6
       3.1.3. Software Constraints ...................................................................................... 6
       3.1.4. Hardware Constraints .................................................................................... 6
  3.2. Design Goals .......................................................................................................... 7
4. DATA DESIGN ................................................................................................................ 8
  4.1. Data Description ..................................................................................................... 8
  4.2. Data Dictionary ....................................................................................................... 9
5. SYSTEM ARCHITECTURE ............................................................................................. 11
  5.1. Architectural Design ............................................................................................... 11
  5.2. Description of Components ................................................................................... 12
       5.2.1. SNMP Agent .................................................................................................. 12
       5.2.2. Simulation Engine ........................................................................................ 14
       5.2.3. Map Controller .............................................................................................. 16
       5.2.4. Display Management ...................................................................................... 18
       5.2.5. Monitoring Component .................................................................................. 20
       5.2.6. Client Application ........................................................................................ 21
  5.3. Design Rationale .................................................................................................... 22
6. USER INTERFACE DESIGN .......................................................................................... 23
  6.1. Overview of User Interface ..................................................................................... 23
  6.2. Screen Images ......................................................................................................... 24
  6.3. Screen Objects and Actions .................................................................................... 25
7. DETAILED DESIGN ................................................................. 27
   7.1. Class Diagram of the System .............................................. 27
   7.2. Component Details ............................................................ 28
       7.2.1. SNMP Agent .......................................................... 28
       7.2.2. Simulation Engine ..................................................... 30
       7.2.3. Map Controller ........................................................ 34
       7.2.4. Display Management ................................................. 37
       7.2.5. Monitoring Component .............................................. 38
       7.2.6. Client Application ................................................... 40
8. LIBRARIES AND TOOLS .......................................................... 41
9. TIME PLANNING .................................................................... 42
   9.1. Term 1 Gantt Chart ............................................................ 42
   9.2. Term 2 Gantt Chart ............................................................ 43
10. CONCLUSION ........................................................................ 44
1. INTRODUCTION

1.1. Problem Definition

SNMP-based monitoring systems are generally used to query the situation of the devices on a network. In case of a failure, the SNMP agent running on the device automatically sends a message to the monitoring system to notify them.

Network administrator may need to create some error to test the network or monitoring system. However, creating a real error on a real device may not be easy without causing any damage to the device or interrupting the people who actually use that device. So, testing the network is an issue that is hard to resolve without simulation environment \(^1\).

1.2. Purpose

This document is written for the developers who will implement the system in order to describe general design features of the system in brief.

1.3. Scope

This document is intended to provide information about designed structure of SNMP simulator which will be produced by AjanSimit. All the data structures and components are described with the specific and complicated details through this document.

1.4. Overview

The rest of this document is separated into 7 main sections.

In System Overview section, general description of AjanSimit SNMP Simulator software system including its functionalities, objectives and benefits are provided.

Under Design Considerations section, the assumptions, dependencies and constraints related to design of the system is given.

In next 3 sections, different views of the design are described in manner of data, architecture and user interfaces.

In the seventh section, estimated scheduling is offered by using Gantt Chart.

The last section summarizes main points described in this document.
1.5. Definitions, Acronyms, Abbreviations

- GUI: Graphical User Interface
- HTTP: Hypertext Transfer Protocol
- MIB: Management Information Base
- MVC: Model-View-Controller
- SNMP: Simple Network Messaging Protocol
- SRS: Software Requirements Specifications
- SDD: Software Design Description
- TCP: Transmission Control Protocol
- UDP: User Datagram Protocol
- XML: Extensible Markup Language

1.6. References

[3] Internet Assigned Numbers Authority (IANA),
   http://www.iana.org/assignments/port-numbers

2. SYSTEM OVERVIEW

The system consists of an implementation of a virtual network, an SNMP agent which
runs on each device of a virtual network and sends trap messages in case of failure, a
monitoring component which records and logs virtual network and simulation activity, a
simulation mechanism for network and finally a GUI for the user to manage the whole
system.

An important detail about the system is that the whole system itself is also connected to
a real SNMP agent. In other words, the trap messages which nodes of the virtual network
might send will be able to be received by an external SNMP-based network monitoring
system. This way, the system can co-operate with real SNMP management systems.
3. DESIGN CONSIDERATIONS

3.1. Design Assumptions, Dependencies and Constraints

3.1.1. Performance Requirements

Only one user can work with the system at a time and only one simulation can be run by the system. The system can handle about 150 nodes simultaneously. The system should be able to send at least 5 trap messages to the destination address within 1 second time.

3.1.2. Time Constraints

The schedule of the project is determined beforehand and it will take about four months to finalize the project successfully. All group members have been working synchronized together. Since the main implementation of the project will be held in the second term, a task distribution and the needed time duration for these tasks are stated in the project schedule.

Gantt chart of the project is given in the Project Schedule section.

3.1.3. Software Constraints

The programming technologies to be used are:

• Adobe FLEX for client side development

• Oracle Java and JSP for server side development

• SNMP Java libraries for network-based development

3.1.4. Hardware Constraints

The hardware requirements are determined by the technologies to be used during development and specified below.

So, the minimum system requirements for the client to run Adobe FLEX are:

For Windows Client:

• Intel® Pentium® III processor

• Microsoft® Windows® XP Home, Professional, or Tablet PC Edition with Service Pack 2 or

3, Windows Server® 2003, Windows Vista® Home Premium, Business, Ultimate, or
Enterprise (including 64-bit editions) with Service Pack 1, or Windows 7

- 512MB of RAM

**For Macintosh Client:**

- Intel Core™ Duo or faster processor
- Mac OS X v10.4.9 or later
- 512MB of RAM

**For Linux Client:**

- Intel Pentium III processor
- Fedora Core 12, Ubuntu 9.10, or openSUSE® 11.2
- 1GB of RAM

### 3.2. Design Goals

The following goals are expected to be achieved for the end product:

**Portability:** Our project is designed to be portable since the used applications are Java and Flex, both of which are platform independent.

**Realistic:** System should be capable of working in real time. The probability of device failure and thus producing traps accordingly should be realistic.

**Reliability:** The resulting application should perform and maintain its functions in routine circumstances, as well as unexpected circumstances. It is expected that there will be no consistency problems. In result product, there should be a mechanism to keep consistency between client application and the server. Therefore several tests will be applied in order to provide reliability.

**Usability:** The software will be used by users who are network designers and experts. So making it user-friendly and easy to learn for new users is not a priority but still it is aimed not to have a steep learning curve.
4. DATA DESIGN

4.1. Data Description

User level authentication will be used for system. Login component stores user information in a hash file.

Map Controller manages virtual map files. These xml format files contain information about nodes in a network such that each map file represents a virtual network. Whenever the user makes a change on network design, Map Controller updates the corresponding virtual map file. If the user starts a simulation, Simulation Engine request current virtual map file from Map Controller.

When building and updating of a map file, Map Controller applies files which contains information about MIBs and node types which can be devices or software.

During simulation, Simulation Engine maintains a list of simulated nodes. This data structure initially contains node information retrieved from virtual map file which Map Controller sends to Simulation Engine at the beginning of simulation.

Simulation Engine sends a notify message to Monitoring Component and SNMP Agent when an error is triggered in one of the nodes. Monitoring Component also receives the event notifications related to simulation itself such as start and end time of simulation. Monitoring Component logs the notification it received into a log file. When a user demands a report, Display Management requests this report from Monitoring Component. Monitoring Component uses log files to produce requested report.

When an error is occurred on a node, Simulation Engine sends a notification to SNMP Agent which sends a trap message to a specified destination address.
4.2. **Data Dictionary**

The class diagram of the system (Figure 8) is provided in section 7.1.

All the system entities mentioned in this document are listed below:

**ClientApplication Component**

Connection::Connect(serverAdd: InetAddress): void
Connection::Disconnect(void): void
GUI::Login(username: String, password: String): boolean
GUI::Logout(void): void
Main::SendReports(report: String): void

**DisplayManagement Component**

DisplayController::SendChanges(operation: enum, params: Object[]): void
DisplayController::DisplayReport(report: String): void

**MapController Component**

MIBHandler::getMIB(nodeTypeID: int): List<String>
MIBHandler::parseMIB(nodeTypeID: int): String
NodeInfo::GetNodeType(nodeTypeID: int): String
RequestHandler::ApplyChanges(operation: enum, params: Object[]): Object
RequestHandler::MapRequest(void): Xml
VirtualMap::InsertNode(nodeID: int, nodeType: int): void
VirtualMap::RemoveNode(nodeID: int): void
VirtualMap::getNode(nodeID: int): Xml
VirtualMap::ChangeNodeProp(nodeID: int, propName: String, newValue: String): void
VirtualMap::InsertConn(connID: int, connType: int, nodeID1: int, nodeID2: int): void
VirtualMap::RemoveConn(connID: int): void
VirtualMap::GetConn(connID: int): Xml
Monitoring Component

LogHandler::LogMessage(errorType: enum): void
LogHandler::PrepareReport(reportType: enum): String
LogHandler::AddLogEntry(errorType: enum): void
LogHandler::ParseLogEntry(void): String
ReportHandler::DemandReport(reportType: enum): String

SNMPAgent Component

SNMPTrapSender::SendTrap(port: int, message: String, ip: InetAddress): void
Notifier::Notify(errorType: int): void
Notifier::ReadyToSend(traps: Queue, isReady: boolean): void

SimulationEngine Component

Master::OperateSimulation(action: enum): void
Master::ExtractNextNode(CurrentMap: Xml, NodeNumber: int): Xml
Master::FillNodes(CurrentMap: Xml): void
NodeThread::CheckProbability(void): void
NodeThread::ProbabilityFunction(errorCount: int, interval: int): void
NodeThread::NonVitalErrorAction(void): void
NodeThread::VitalErrorAction(void): void
SimClock::StartTime(void): void
SimClock::PauseTime(void): void
SimClock::StopTime(void): void
SimClock::ResumeTime(void): void
SimClock::SetTime(newTime: int): void
SimClock::GetCurrentTime(void): Time
SimClock::TimeMode(type: int): void
SimClock::SetTimeInterval(milliseconds: int): void
SimClock::GetTimeInterval(void): int
SimClock::IncrementElapsedTime(void): void
SimClock::GenerateTick(void): void
5. SYSTEM ARCHITECTURE

5.1. Architectural Design

The system is designed as web application which will take input from user via a FLEX application running on the client side. Rest of the system will be deployed on the server side. The system architecture is designed in MVC pattern (Figure 1).

MVC is a software architecture pattern which isolates the application logic from user interface permitting modularity, independent development and maintenance. MVC pattern suggest that the system is separated into 3 subsystems. Model handles application logic and exposes application functionalities. View renders interface communicating with user and sends user gestures to the Controller. Controller takes user inputs and maps them with updates of model by making method calls on model objects.[4]

In this project, subsystem View encapsulates FLEX application and will be deployed on the client side. Subsystem Controller encapsulates Display Management component. Controller is responsible for taking user inputs over a web server and interpreting them and making necessary calls on other components. Subsystem Model consists of Map Controller,
Simulation Engine, SNMP Agent and Monitoring Component. Model will store and maintain virtual maps, handles simulation activities and logging and sending trap messages in case of an error occurrence.

5.2. Description of Components

5.2.1. SNMP Agent

5.2.1.1. Process Narrative

SNMP Agent is responsible for creating trap messages and sending them to the destination IP addresses. When an error occurred in one of the virtual nodes, Simulator Engine notifies SNMP Agent about error and related node.

5.2.1.2. Interface Description

The component sends network packets using TCP or UDP. Content of these packets is constructed using SNMP. Since all the packets which the component sends contain trap messages, SNMP Agent uses only destination port number 162 which is default port number for SNMP trap messages [3].

5.2.1.3. Component Processing Detail

SNMP Agent uses incoming information to construct SNMP trap packets. SNMP Agent has networking ability since it sends packets to the network ports. The destination IP address or addresses are provided to the SNMP Agent with incoming notification messages. If more than one error notification comes to the SNMP network at the same time, SNMP Agent puts them into a queue structure and constructs corresponding trap messages repeatedly until queue is empty.
5.2.1.4. Dynamic Behavior of the Component

The sequence diagram of the component is given below (Figure 2). The component consists of 2 classes, namely Notifier and Agent. Notifier class takes notification message, extract data from it. Agent handles sending trap messages.

![Sequence diagram of SNMP Agent Component](image)

Figure 2 - Sequence diagram of SNMP Agent Component
5.2.2. Simulation Engine

5.2.2.1. Process Narrative

Simulation Engine is the component that handles simulation activities. It maintains simulation clock, simulates the errors and notifies the other components about its activities. Simulation Engine uses a probability function to determine a node has an error or not.

5.2.2.2. Interface Description

At the beginning of each simulation, Simulation Engine requests a virtual map from Map Controller component. It also receives messages from Display Management related to simulation operations such as start or pause simulation operations etc. It sends a notification message to the SNMP Agent if an error occurs in one of the nodes. It also sends messages to the Monitoring component which are related the simulation activities.

5.2.2.3. Component Processing Detail

Simulation Engine initiates a new thread for each node. At each tick of clock, each thread runs its probability function. Probability function decides if an error occurred on this node. If so, it also decides type of the error. When an error occurs, node thread decides whether the error is vital or not. That is, vital error stops the work of node immediately. If the error is non-vital, the node continues to run. In case of a non-vital error, the node thread sends a notification message to the SNMP Agent with necessary information to construct a trap message. In case of a vital error, the notification message goes both SNMP agent and Monitoring Component. Then the thread stops working. If no error occurs, the node thread waits next tick of clock.
5.2.2.4. Dynamic Behavior of the Component

Figure 3 - Sequence diagram of Simulation Engine component
5.2.3. Map Controller

5.2.3.1. Process Narrative

Map Controller is the component responsible for keeping track of the network creation and manipulation operations. It organizes the current virtual map file so that each change in the design of the virtual network is reflected to this file. When requested from other components, this component retrieves the required information from the virtual map file. When user wants to save the design, this controller gives power to the user to save or load any virtual map files. There are also NodeType file and MIB files. However these two files will be read-only by the Map Controller and other component will directly access these files.

5.2.3.2. Interface Description

When user starts designing network, Display Management component notifies Map Controller for each change in design. Then the required changes are reflected to the virtual map file. When user wants to save the network design, the request is also made by Display Management component from the Map Controller component where it is handled.

During simulation, Simulation Engine requests the node and the connection information corresponding to simulated part of the virtual network and Map Controller responds that request with the related information obtained from current virtual map file.

5.2.3.3. Component Processing Detail

Map controller keeps track of changes in network design by making I/O operations on virtual map file. These operations are inserting a node, deleting a node, changing information on a node in the xml file. Java XML libraries will be used to accomplish these XML operations. This component also makes I/O operations on NodeType file which is also xml file and MIB files which are txt files when the related method is called.
5.2.3.4. **Dynamic Behavior of the Component**

Apply changes method resides in Request Handler class of this component to reflect the changes of designed network. This method is called by Display Management component.

The RequestMap method in Request Handler class of Map Controller component is to be called by Simulation Engine component. This method causes returns the xml node containing the node, connection and the configuration of the nodes for simulated part of the network.

![Sequence Diagram of Map Controller Component](image)

**Figure 4 - Sequence Diagram of Map Controller Component**
5.2.4. Display Management

5.2.4.1. Process Narrative

Display management is the component responsible with getting the user related tasks and send the related calls to the related components. To be more specific, when the user makes any change on the network design, makes any simulation operation, requests any report related to the running or accomplished simulation, or even logs in to the system or logs out from the system, the system will be aware of that by this component.

5.2.4.2. Interface Description

The connection of the client application is directly to this component. Display Management component is the interface of the system for the client application.

Logging into system and logging out of the system are handled internally by this component.

Changes related to the network design are sent to the Map Controller component where they will be handled.

Operations related to the simulations are redirected to Simulation Engine where the simulation takes place.

Any report demands including performance report or simulation log will be requested from Monitoring Component.

5.2.4.3. Component Processing Detail

It is responsible for checking the incoming request from client application in case of security vulnerabilities. Keeps track of the user, when it logs in, when it logs out, and when it makes any request. If the user is not logged into the system, this component will drop the incoming requests from that remote user.

This component is not a part of Model subsystem. So there is not much job of this component related to the network design, simulation tasks or any other activity in application logic.

The only duty of this component is to call ApplyChanges method of the Map Controller component with the parameter list when anything changes on the design, to call
OperateSimulation method of Simulation Engine component to trigger it to do the operations requested by the user, to call the DemandReport method of the Monitoring Component to make it to prepare a report for the client request and sent is back to the client application where these methods are initially called.

5.2.4.4. Dynamic Behavior of the Component

![Sequence Diagram of Display Management Component](image)

Figure 5 - Sequence Diagram of Display Management Component
5.2.5. Monitoring Component

5.2.5.1. Process Narrative

Monitoring Component is the component responsible to log the operations to Log file and to retrieve them from the file whenever they are needed to be a report.

5.2.5.2. Interface Description

When an event is happened where the event is either a trap sent event or any event related to the simulation itself, this component is triggered by Simulation Engine.

Any request from Display Management component is handled by ReportHandler class of this component. The report is created by using the methods in LogHandler class to get the required information from the log files. The results are sent via the methods residing in Display Management component.

5.2.5.3. Component Processing Detail

When some notifications received by the monitoring component from Simulation Engine, the user creates a log file about simulation and this happen while the simulation is being done. That is, whenever an error occurred in one of the virtual nodes, the Simulation Engine notifies it to Monitoring Component and then it sends to Log File. In addition, when the Display Management wants any report about errors or operations from Monitoring Component, it receives from Log File and reports it to Display Management.

5.2.5.4. Dynamic Behavior of the Component

![Sequence Diagram of Monitoring Component]

Figure 6 - Sequence Diagram of Monitoring Component
5.2.6. Client Application

5.2.6.1. Process Narrative

The user has a web based application to do all network design and simulation operations on Internet.

5.2.6.2. Interface Description

All the use cases mentioned in SRS Document will be applicable to this application. Display Management component sends the retrieved results, reports and updates to this application. GUI receives them and reflect the incoming changes to the user and notifies back Display Management when user changes something on the application.

5.2.6.3. Component Processing Detail

There is a Flex Application on the Web Server and the user can reach that application via Internet by providing the IP number of the server on a web browser. Then the login screen appears to allow user to be logged in to the system.

After a successful login operation, user will see the main screen of the application where all the use cases can be followed by clicking on the right menu item. There is no algorithmic job of the client application. The only thing is to provide user a generic screen to model a network and to change it as he wants. The connection between the client application and the Display Management component will stay alive until user logs out or closes the browser.

5.2.6.4. Dynamic Behavior of the Component

Client Application waits for user action. When a user action occurs, Client Application takes it as an event.

![Figure 7 – Sequence Diagram of Client Application Component](image-url)
5.3. Design Rationale

MVC pattern is widely used in web based projects. MVC is chosen as system architecture since it separates application logic and user interface, which provides more cohesion and less coupling as good software engineering practice. By using this architecture, our 3 subsystems are almost isolated from each other.

We assign each major activity to a distinct component. Thus, Virtual network handling, simulation, SNMP communication, user communication, logging are all separated to different components. Furthermore, we minimized the interaction between components.

All components except Display Management have limited relation to other component. However, Display Management which is a part of controller subsystem has to communicate other components frequently. We discussed that whether this might causes problem. Because, if Display Management fails or runs with low performance, remaining parts of the system might be affected. Yet, still we decided on this architecture. Otherwise, interconnections between components could quickly overgrow so management and maintenance of the system would be impossible.
6. USER INTERFACE DESIGN

6.1. Overview of User Interface

The very first screen that the user is faced with is the Login Screen (Image 1). The user is asked to provide a user name and password to securely login to the system. By doing that every operation on the server is logged for that specific user.

If the user does not provide a correct username and password combination, system will decide the required action to be taken for that user. If he entered a valid username but wrong password for 5 times, the system will send a notification to the real user that his account is trying to be used by the others, and the password change is advised.

After a successful login, the user of the Client Application (Image 2) is able to create a virtual network by using the network nodes and the connections residing on the right hand side of the application. There are some network node types available such as routers, switches, hubs, repeaters, end-devices and each of these node types includes some specific device or software list in it and this list will get populate as the AjanSimit SNMP Simulator develops.

The user selects the desired node from a corresponding node type group and drags it to the network design area indicated with a light blue color on the screen and drops it. Then user applies this strategy to the other network nodes and creates connections between these nodes by selecting appropriate connection type from the list on the right hand side and clicks the two nodes on the draw area one by one. Then the user right clicks on the connection to set the required configurations on the connection. This is also valid for the network nodes. If the user right clicks on a network node, the properties of a selected node will display a popup window to set and save the changes in configurations of those nodes.
6.2. Screen Images

Image 1 - Screenshot of the Login Screen

Image 2 - Screenshot of the Main Screen of Client Application
6.3. Screen Objects and Actions

In the “File” menu, there are 4 options: “New”, “Load”, “Save” and “Exit”.

“New” menu item is to start a new network design. All the process made so far is cleared after an approval window is accepted. There is also an option to save the current topology to a file.

“Load” menu item is to load a previously saved network design. All the configurations and settings will be loaded from a specified file. There is also an option to save the current topology to a file.

“Save” menu item is to save the current design to a specified file. If the file is specified before, the file is updated. No change is observed on the current design on the screen.

“Exit” menu item is to terminate the client application. User will be warned to approve this process. There is also an option to save the current topology to a file.

In the “Design” menu, there are 2 options: “Delete” and “Clear Screen”.

“Delete” menu item is to delete the selected nodes on the design. If no item is selected this menu item is in disabled state. An approval window appears to accept the operation.

“Clear Screen” menu item is to restart drawing with the same application settings. That is the difference this item from the “New” item in the “File” menu. An approval window appears to accept the operation.

In the “Simulation” menu, there are 4 options: “Start”, “Pause”, “Stop” and “Resume”.

“Start” menu item is to start the simulation process on the designed network. A window appears to set the parameters of the simulation before starting.

“Pause” menu item is to stop the running simulation temporarily. If some of the nodes are selected, it behaves like a partial pausing. If no simulation is running, this button is in disabled state.

“Stop” menu item is to stop the running or paused simulation permanently. If some of the nodes are selected, it behaves like a partial stopping. If there is no simulation running or paused, this button is in disabled state.
“Resume” menu item is to resume the paused simulation previously. If some of the paused nodes are selected, it behaves like a partial resuming. If there is no simulation paused, this button is in disabled state.

In the “Options” menu there is no menu item yet. There will be some items to set the properties of the application and they will be identified and included during development phase.

In the “Help” menu, there are 2 options: “Application Help” and “About AjanSimit”.

“Application Help” menu item pops a window to show the help file created for the user to show how to use the application. This help file will be included to the application at the end of the development period according to the final version of the application.

“About AjanSimit” menu item pops a window to give information about the group AjanSimit the producer and the coder of this project in brief.
7.1. Class Diagram of the System

Figure 8 – Class Diagram of the System
7.2. Component Details

7.2.1. SNMP Agent

SNMP Agent is responsible for creating trap messages and sending them to the destination IP addresses. When an error occurred in one of the virtual nodes, Simulator Engine notifies SNMP Agent about error and related node. The component sends network packets using TCP or UDP. SNMP Agent uses only destination port number 162 which is default port number for SNMP trap messages.

SNMP Agent consists of Notifier and SNMPTrapSender. Notifier decides when SNMPTrapSender will send the trap message and SNMPTrapSender sends the trap message to the external application. It takes the data and notification needed from the Simulation Engine to prepare trap messages.

SNMP Agent needs a network connection to send the packages to the expected network ports. It also needs SNMP libraries.

SNMP Agent uses incoming information to construct SNMP trap packets. The destination IP address or addresses are provided to the SNMP Agent with incoming notification messages. If more than one error notification comes to the SNMP network at the same time, SNMP Agent puts them into a queue structure and constructs corresponding trap messages repeatedly until queue is empty.

SNMP Agent allows communication with SNMP server.
7.2.1.1. **Notifier**

**METHODS:**

`readyToSend(Queue traps, Boolean isReady): void`

Checks if SNMPTrapSender is ready to send the trap message and if it is not, it puts the message number to queue

`notify(int message_number): void`

Tells SNMPTrapSender to send the trap message to the external application

**ATTRIBUTES:**

- `int message_number`
- `Queue trap_queue`
- `boolean isReady`

7.2.1.2. **SNMPTrapSender**

**METHODS:**

`sendTrap(int port, String message, InetAddress destination): void`

Sends the according trap message to the external trap handling application

**ATTRIBUTES:**

- `int trap_port`
- `int message_number`
- `String message`
- `InetAddress host_address`
7.2.2. Simulation Engine

Simulation Engine component is the module which deals with simulation activities. In fact, it contains the classes that perform simulation. Simulation engine keeps the threads that represent each node in the virtual network map. These threads run concurrently, periodically check if an error occurred in the nodes they represent.

Actions which Simulation Engine should perform come from Display Management component. The virtual map which the component simulated is provided by Map Controller component to the module. Records related to simulation are sent to the Monitoring component by the Simulation Engine. Finally, Simulation Engine is responsible to notify SNMP Agent component when one of threads decides that the node it represent has an error.

Simulation Engine consists of 3 classes. Below, each of these classes is explained specifically with the methods and attributes which they contained.

7.2.2.1. Master

Master class decides that what action will be done according to incoming messages. It creates and destroys instances of NodeThread class. It signals the SimClock class about its next action.

**Methods:**

OperateSimulation(enum action): void

Identifies type of the action, then calls necessary method.

ExtractNextNode(Xml CurrentMap, int NodeNumber): void

Extracts node information from a xml file.

FillNodes(Xml CurrentMap): void

Takes an xml file, and then uses it to created nodeThread objects.

**Attributes:**

Xml CurrentMap
### 7.2.2.2. **NodeThread**

It is inherited from Thread class. Each instance of this class represents a node in virtual map.

**METHODS:**

- **CheckProbability(void): void**
  
  Runs probability function

- **ProbabilityFunction(int errorCount, int interval): void**
  
  It is distribution function which decides whether an error occurred or not. It takes 2 arguments which indicate average number of errors occurred in given time interval in seconds.

- **NonVitalErrorAction(void): void**
  
  In case of a non-vital error, it is called

- **VitalErrorAction(void): void**
  
  In case of a vital error, it is called

**ATTRIBUTES:**

- Final String nodeID

- Final String nodeType

- List<String> ErrorTypesList

- int nodeStatus

- int LastOccuredError
7.2.2.3. SimClock

SimClock class determines the simulation time. In other words, it simulates time ticks in the system. It wakes up threads that are in sleep waiting the next tick of clock.

METHODS:

StartTime(void): void

Starts clock ticking at each time interval, sets status attribute to 1

PauseTime(void): void

Pauses clock ticking, sets status attribute to 0

StopTime(void): void

Pauses clock ticking and sets time to zero, sets status attribute to -1

ResumeTime(void): void

Resumes clocks ticking, set status attribute to 1

SetTime(int newTime): void

Sets elapsed time to newTime value which is in seconds

GetCurrentTime(void): Time

Gets current system time

TimeMode(int type): void

Changes mode real time or simulated time. In real time mode, at each second, clock tick is generated. In simulated time mode, clock tick is generated at specific time interval

SetTimeInterval(int miliseconds): void

Sets the timeInterval value in milliseconds. In real time mode, this value is always set to the 1000.
GetTimeInterval(\texttt{void}): \texttt{int}

Gets timeInterval value

IncrementElapsedTime(\texttt{void}): \texttt{void}

As long as status is 1, increments elapsed time at each timeInterval.

GenerateTick(\texttt{void}): \texttt{void}

Notifies all NodeThread threads at each increment of time

\textbf{ATTRIBUTES:}

\begin{itemize}
  \item \texttt{int ElapsedTime}
  \item \texttt{int status}
  \item \texttt{int mode}
  \item \texttt{int timeInterval}
  \item \texttt{Time currentSysTime}
\end{itemize}
7.2.3. Map Controller

Map Controller is the module responsible for keeping track of the network creation and manipulation operations. It manages the virtual map file which is the main file of the system. It can insert, delete and get nodes and connections from the map file.

There should be enough disk space available on the server for this component work with the xml file that can be huge for very large scale networks.

There is no subcomponent available in this component.

During simulation, Simulation Engine requests the node and the connection information corresponding to simulated part of the virtual network and Map Controller responds that request with the related information obtained from current virtual map file.

Apply changes method resides in Request Handler class of this component to reflect the changes of designed network. This method is called by Display Management component.

There are 3 files that this component manages:

- Virtual Map File
- MIB Files
- NodeType File

These files are shared for all the node threads created by Simulation Engine. For that reason the operations on these files should be thread safe, in other words, only one method can be used to access these files from any of the component.
7.2.3.1. **VirtualMap**

**METHODS:**

- `insertNode(int nodeID, int nodeType) : void`
  Inserts a node to the virtual map file

- `removeNode(int nodeID) : void`
  Removes a node from the virtual map file

- `getNode(int nodeID) : Xml`
  Gets the node from the virtual map file

- `changeNodeProp(int nodeID, String propName, String newValue) : void`
  Changes a property of a node in the virtual map file

- `insertConnection(int connID, int connType, int nodeID1, int nodeID2) : void`
  Inserts a connection to the virtual map file

- `removeConnection(int connID) : void`
  Removes a connection from the virtual map file

- `getConnection(int connID) : Xml`
  Gets the connection from the virtual map file

**ATTRIBUTES:**

- `Xml tempNode`

- `Xml tempConn`
7.2.3.2. RequestHandler

**METHODS:**

- `applyChanges(enum op, Object[] params) : Object`
  Makes required function calls to complete the requested operation

- `mapRequest(void) : Xml`
  Gets the complete virtual map file from VirtualMap

**ATTRIBUTES:**

- Enum Operation
- VirtualMap vMap
- MIBHandler mibHand
- NodeInfo nodeInf

7.2.3.3. NodeInfo

**METHODS:**

- `GetNodeType(int nodeTypeID): String`
  Seeks for the specified nodetype in NodeTypes file and returns the name

**ATTRIBUTES:**

- String fileURL

7.2.3.4. MIBHandler

**METHODS:**

- `GetMIB(int nodeTypeID): List<String>`
  Calls the parseMIB method iteratively to collect all the MIB information.

- `ParseMIB(int nodeTypeID): String`
  Reads one attribute from MIB file at each time

**ATTRIBUTES:**

- String fileURL
7.2.4. Display Management

Display Management is the module that takes user actions from client, then makes necessary calls accordingly. It is responsible sending and receiving packets from ports and sending messages to other components. It simply performs a mediator duty between server side and client side.

Display Management is deployed in the server side. All data sent or received through network, except SNMP communication, are transfer over Display Management. It communicates with Client Application via TCP.

When a message comes to Display Management from Client Application, Display Management classifies it. If it is related to virtual map design, Display Management transfers it to Map Controller component. If it is related to simulation operation, the message is sent to the Simulation Engine. Messages that are requesting a report is transferred to the Monitoring component.

Since Display Management provides networking capability, it should allocate some network resources. Since the server side will run as web application using HTTP, Display management uses port number 80 to transmit data to client application.

Display Management contains DisplayController class whose methods and attributes are explained below.

7.2.4.1. DisplayController

Methods:

SendChanges(enum op, Object[] params): void

Takes a user action from client application, decides which component will perform the requested action, then make necessary method call on related component.

DisplayReport(String report): void

Takes a report as string, then send it to the client application over network.
ATTRIBUTES:

Master SimEng

RequestHandler reqHand

ReportHandler repHand

7.2.5. Monitoring Component

This component is of the kind module. It is the component responsible to log the operations to log file and to retrieve them from the file whenever they are needed to be a report.

Mainly this component receives the notifications which other ones send. After receiving those messages (traps), it sends them to LogFile in order to get the reports about them.

Usually the notifications are assumed to be sent by Simulation Engine. Moreover, while simulating process a log file is supposed to be created too.

No subcomponents exist.

One of the most important collaboration of Monitoring Component with other components is that when it sends any trap or messages, it is triggered by Simulation Engine. In example, whenever an error occurred in one of the virtual nodes, the Simulation Engine notifies it to Monitoring Component and then it sends to Log file. In addition, every request from Display management component is handled by ReportHandler class of this component. The report is created by using the methods in LogHandler class to get the required information from the Log files. The result is sent via the methods residing in Display Management component.

7.2.5.1. ReportHandler

METHODS:

DemandReport(enum reportType): String

Triggers the LogHandler to prepare a report
**ATTRIBUTES:**

LogHandler logger

Enum reportType

String fileURL

**7.2.5.2. LogHandler**

**METHODS:**

`logMessage(enum errorType): void`

Generates the message to be logged by errorType

`addLogEntry(enum errorType): void`

Logs the created message to the log file

`parseLogEntry(void): String`

Parses the log file to iterate on the entries

`prepareReport(enum errorType): String`

Prepares a report from log files

**ATTRIBUTES:**

Enum errorType

String fileURL

Enum reportType
7.2.6. Client Application

Client Application is a subsystem of SNMP Simulator. It is a web based application to do all network design and simulation operations.

All the use cases mentioned in SRS Document will be applicable to this application.

The Flex Application on the web server can only be reached by providing the IP number of the server on a web browser. Also, the user must login via the login screen to be allowed into the system.

Client application consists of main program.

Client application sends the changes that user performs to Display Management and takes the reports of the events also from Display Management.

Client application needs to access the Flex Application on the Web Server so it needs internet connection.

After a successful login operation, user will see the main screen of the application where all the use cases can be followed by clicking on the right menu item. There is no algorithmic job of the client application.

Client application provides the user a generic screen to model a network and the allows the user to change it when needed.

**METHODS:**

(sendReports(String): void

Sends prepared reports to the client application

**ATTRIBUTES:**

DisplayController disp

Lets the client application call the methods inside DisplayController class
8. LIBRARIES AND TOOLS

- Adobe Flash Builder 4
- Adobe Flex 4
- Eclipse IDE
- Edraw Max Vector-Based Drawing Tool v5
- Free XML Libraries for Java
- Free SNMP Libraries for Java
- Java SE v1.6
- Microsoft Office Word
- Socket, Multithread and Server Tools and Libraries of Java
- UMLet - UML Tool for Fast UML Diagrams
- Virtual Boss Construction Scheduling Software
<table>
<thead>
<tr>
<th>Number</th>
<th>Task</th>
<th>Start</th>
<th>End</th>
<th>Duration</th>
<th>% Complete</th>
<th>2010</th>
<th>2011</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Oct</td>
<td>Nov</td>
</tr>
<tr>
<td>1</td>
<td>Project Management</td>
<td>26/10/2010</td>
<td>30/1/2011</td>
<td>97</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SNMP Research</td>
<td>12/11/2010</td>
<td>15/11/2010</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Data Object Determination</td>
<td>28/11/2010</td>
<td>2/12/2010</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Software Behavior Determination</td>
<td>3/12/2010</td>
<td>4/12/2010</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Planning</td>
<td>5/12/2010</td>
<td>5/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Design Constraints Specified</td>
<td>7/12/2010</td>
<td>9/12/2010</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>System Data Entities Designated</td>
<td>10/12/2010</td>
<td>11/12/2010</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Architectural Design</td>
<td>12/12/2010</td>
<td>15/12/2010</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Components Specified</td>
<td>16/12/2010</td>
<td>16/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SNMP Agent Detailed</td>
<td>17/12/2010</td>
<td>17/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Simulation Engine Detailed</td>
<td>18/12/2010</td>
<td>18/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Map Controller Detailed</td>
<td>19/12/2010</td>
<td>19/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Display Management Detailed</td>
<td>20/12/2010</td>
<td>20/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Monitoring Component Detailed</td>
<td>21/12/2010</td>
<td>21/12/2010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Client Application Logic</td>
<td>22/12/2010</td>
<td>23/12/2010</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>User Interface Design</td>
<td>24/12/2010</td>
<td>25/12/2010</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Detailed Time Planning</td>
<td>26/12/2010</td>
<td>27/12/2010</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Presentation</td>
<td>28/12/2010</td>
<td>3/1/2011</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Component Classes &amp; Attributes</td>
<td>28/12/2010</td>
<td>2/1/2011</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Virtual Map XML Template</td>
<td>3/1/2011</td>
<td>3/1/2011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Node Type XML Template</td>
<td>4/1/2011</td>
<td>4/1/2011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Data Objects Revised</td>
<td>7/1/2011</td>
<td>7/1/2011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Prototype Preparation</td>
<td>10/1/2011</td>
<td>24/1/2011</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Task</td>
<td>Start</td>
<td>End</td>
<td>Duration</td>
<td>% Complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Project Management</td>
<td>1/2/2011</td>
<td>28/6/2011</td>
<td>148</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Web Page Preparation</td>
<td>15/2/2011</td>
<td>18/2/2011</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SNMP Research</td>
<td>18/2/2011</td>
<td>23/2/2011</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SNMP Agent Development</td>
<td>24/2/2011</td>
<td>7/3/2011</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Alpha Release version 3</td>
<td>10/5/2011</td>
<td>10/5/2011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Beta Release</td>
<td>1/6/2011</td>
<td>1/6/2011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Final Release</td>
<td>21/6/2011</td>
<td>21/6/2011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10. CONCLUSION

This detailed design shows group AjanSimit’s approach to create SNMP network simulator with the guidance of Siemens Enterprise Communications. In this document, all of the details about the design of the project were given. The detailed overview of the project was given with technical specifications. Desired user interface is explained with 2 sample images. The architecture of the system and the technical design are explained in detail and are supported with sequence diagrams. Class diagram of the system is placed. Each component is explained in detail with all of the methods, method parameters, return values, used data structures. Finally, the project schedule is determined and shown in detail with Gantt Charts. This document was the latest step for the development which will take place in the second semester. With this report, what needs to be done in the implementation phase was made much specific and it also has helped us to see the difficulties that can arise when working on a large-scale software product.