Final Project Report

NET PROFILE SWITCHER 1.0

Software Engineering Project | CS3202 | September 20, 2014

Name : Madumal S.A.T.

Index No : 110339G

Date of Submission : 20/09/2014

Table of Content

Table of Content 2

Introduction 3

project Idea 3

What 'Net Profile Switcher 1.0' do 3

use Case Diagram 4

Use Case Realzation 5

Sequence Diagram of 'Change Network Profile' Use Case 5

Sequence Diagram of 'Add New Profile' Use Case 6

Literature Review 7

System Design-Diagrams & Justifications 9

Design Architecture of 'Net Profile Switcher 1.0' 10

Package Diagram of the Software Architecture Design 12

Important Implementation Details 15

Changing proxy settings 15

Prototypes of relevant Methods 15

Changing Network Settings 17

Prototypes of relevant Methods 17

Data Persistance 19

Prototypes of some methods accessing database 20

Taking Precautions for sql Injection attacks possible 21

NEtwork Status Change Detection 23

Evaluation - (Testing & Results) - Performance 25

Developer Testing 25

End User Testing 26

Performance 26

Conclusion 27

Risk List 28

References 30

Introduction

PROJECT IDEA

Idea Behind the 'Net Profile Switcher 1.0' is basically the concept of network profile management. Nowadays almost all the PC users work with several networks in their usual day-to-day routine. Since different networks that the users get connected to, have different configuration settings. Thus the users have to configure the system over and over each time they switch from one network to another which obviously is a tedious task. 'Net Profile Switcher 1.0' address this issue for Windows user.

What 'Net Profile Switcher 1.0' do

In this app, as the user connects to a network for the first time, the user has to do the proxy and other necessary network settings configurations manually. Well, if the network support automatic configuration detection or network setting configuration scripts then app will automatically acquire those information and will proceed according to them. Once configured for a particular network the app will keep those setting info. as a profile.

In the encounter of switching of networks the app will automatically changes the settings or the user just has to change/apply the settings.

USE CASE DIAGRAM

Hence the requirements of the 'Net Profile Switcher 1.0' are quite few in number, use case diagram is pretty simple. In the Following I have shown an abstract Use Case Diagram in the following. It shows the use case which has a significant impact on the primitive functionality of the application.

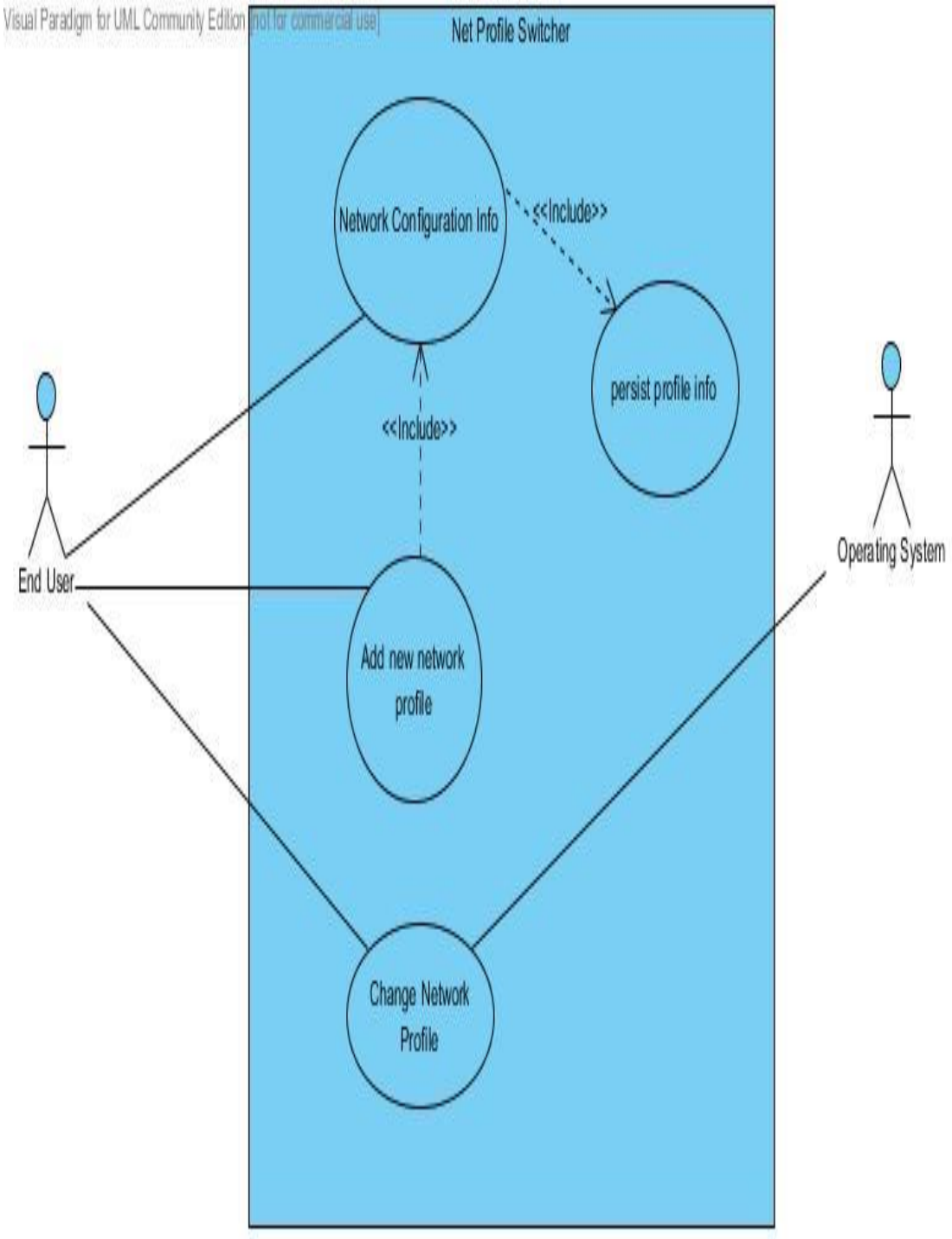

Figure 1.2.0

USE CASE REALZATION

For the Implementation of the use cases, they should be realized beforehand. With the intension of use case realization I have drawn the sequence diagram of some use cases having a significant impact on the application functionality. The desired functionality then can be implemented referring to those sequence diagrams.

Sequence Diagram of 'Change Network Profile' Use Case

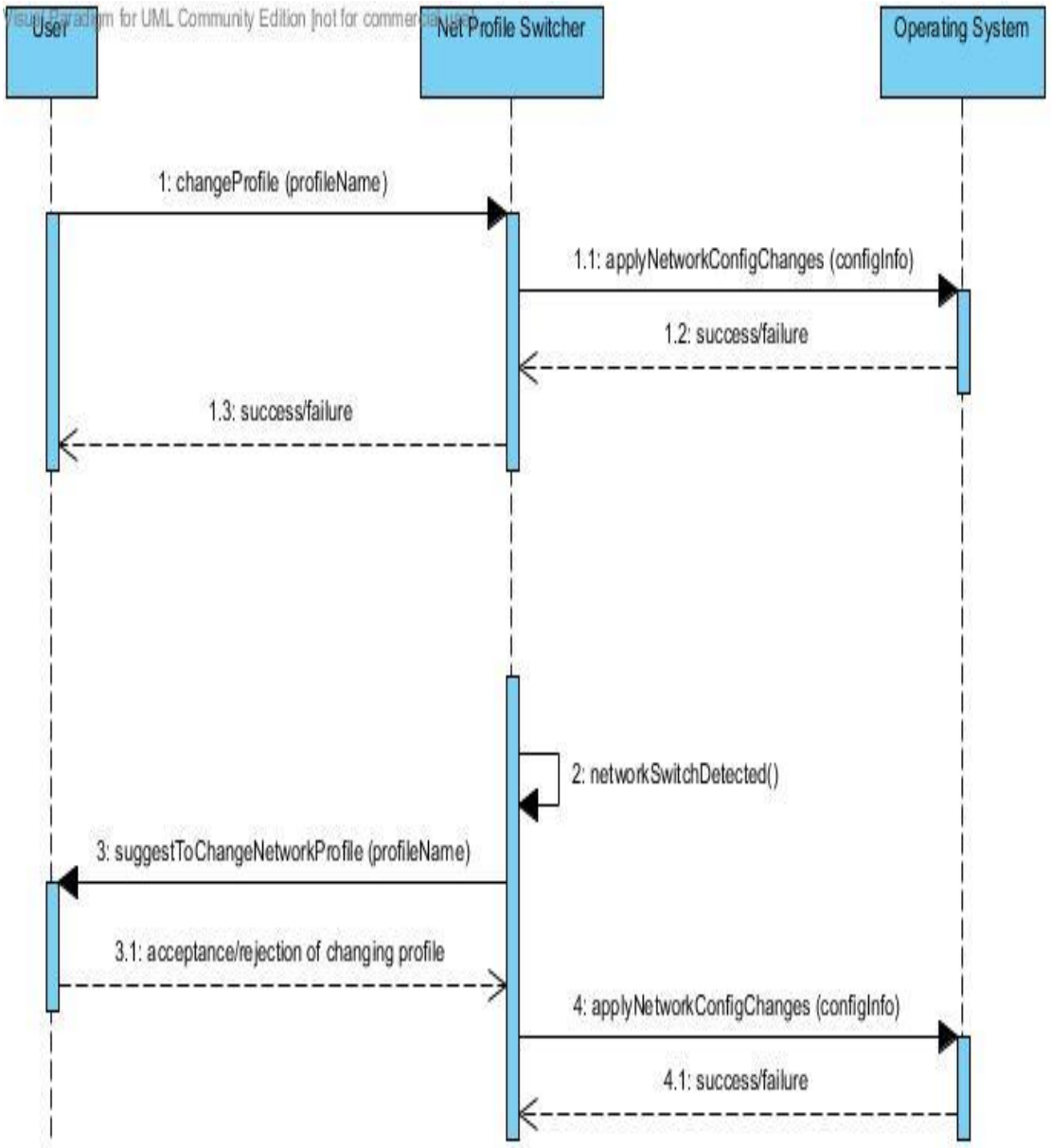

Figure 1.3.0

Sequence Diagram of 'Add New Profile' Use Case

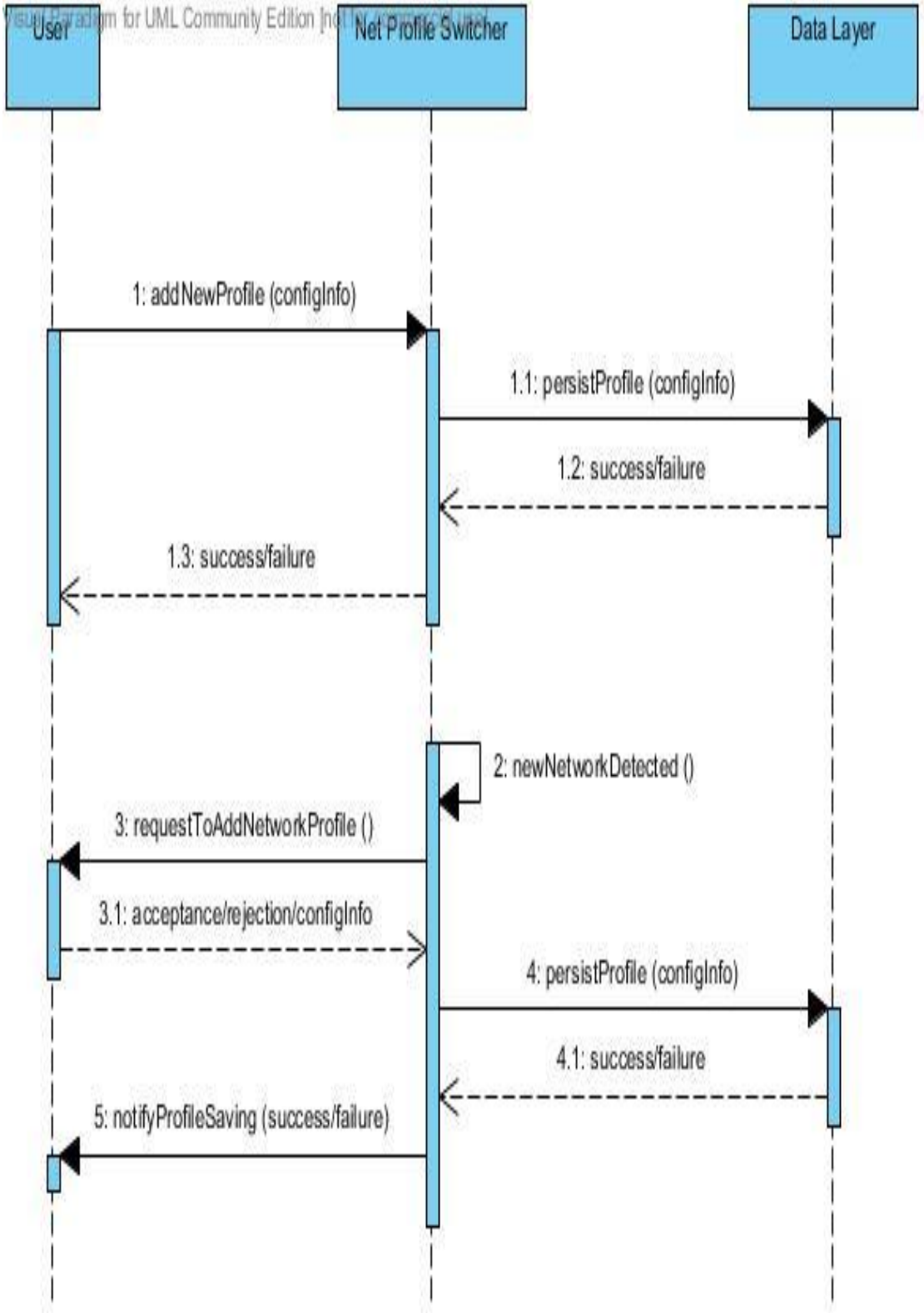

Figure 1.3.1

Literature Review

Network profile management is quite popular topic contemporarily. Significant number of software applications have been published. There are quite no. of software applications developed which can be rewarded as good. As computer literacy grow up around the world the usage of computers increases. The prevailing trend is having a laptop than a desktop computer. On the other hand most of the laptop pc users work with several networks in their day-to-day life and they normally have to switch between those network two or three times a day or even more. There the network profile management tools come into play.

As I have mentioned earlier there is a lot of network profile management tools. If I list some of the corporate names of those apps;

- Net Set Man [1]
- Control Plane [2]
- Quick Config [3]
- Net Profiles [4]
- Smart IP Profile [5]
- IP Switcher [6]
- Net Changer [7]
- Easy Net Switch [8]
- TCP Profile Manger [9]

When we analyze them, we can see some applications have a lot of features. They do almost everything network profile management. But there is a problem. The users should have a significant knowledge about the network connections, how computers work, and how to configure the software. 'Net Set Man' can be specify ad an example for this category. Working with those kind of software tools is time consuming, most users don't necessarily have the knowledge required, and most of the users don't need that much features in their usual routine. Besides those software applications are priced at a high value. Thus there is still an opportunity for a software which have the functional requirements of a network profile management tool but easy to use and available either at low price or for free.

Some of the specific software applications have some of the above mentioned requirements. But in their case user friendliness and convenient behavior is the issue. Most of those software applications are not up to the standards which make it difficult for user to use them. 'TCP/IP Manager' is can be given as an example for this category.

In network profile management tools, making network switches seamless is one of the main ideas. To achieve that automatic detection of network changes is essential. In some software tools they don't provide that feature which makes

the software simply useless because Windows platform provide the rest of the functionality itself. What the software tool has done was gathering those functionality to a one place. 'Mobile Net Switch' is an example for this category.

After analyzing these tools we can get into a conclusion. Although there are several network profile management tools available there is still a lack of a tool which provide core functionalities including automatic profile switching with simplicity where no expertise knowledge is required. In addition to that a tool either at low price or even further for free still has the demand. 'Net Profile Switcher 1.0' belongs to that category.

System Design-Diagrams & Justifications

At the Design Phase there were several considerations that needed in detail attention. Among them behavior of the app when connected the device is connected, when the device is offline, and at changes of network status. The above behavior is described in the following diagram.

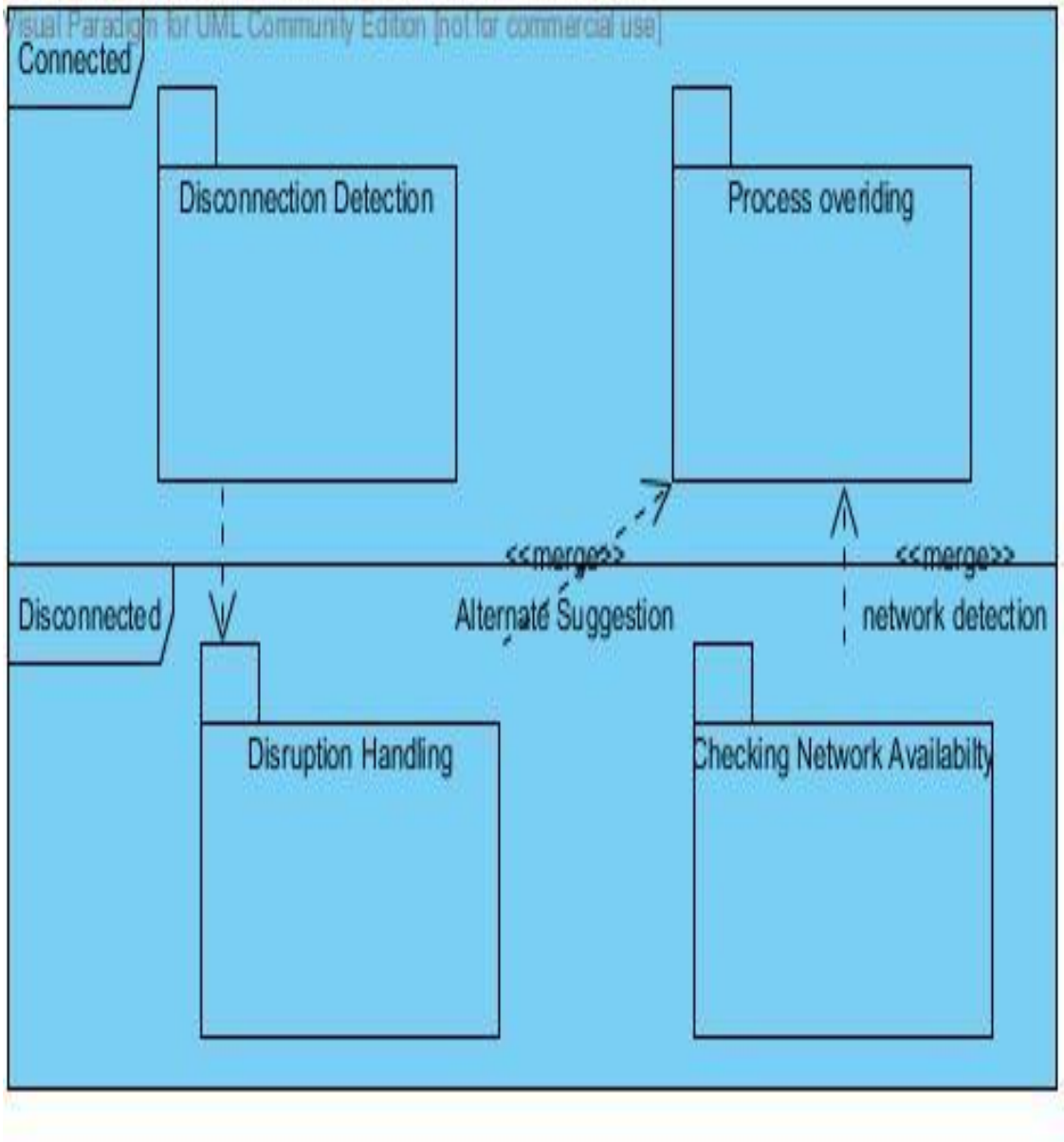

Figure 3.0

Design Architecture of 'Net Profile Switcher 1.0'

Design architecture of the 'Net Profile Switcher 1.0' is quite simple hence it is a stand-alone light-weight software system. Three tier architecture was used but there were some alterations were imposed with the intension of achieving security, efficiency, performance, and reliability. Software architecture diagram is shown below.

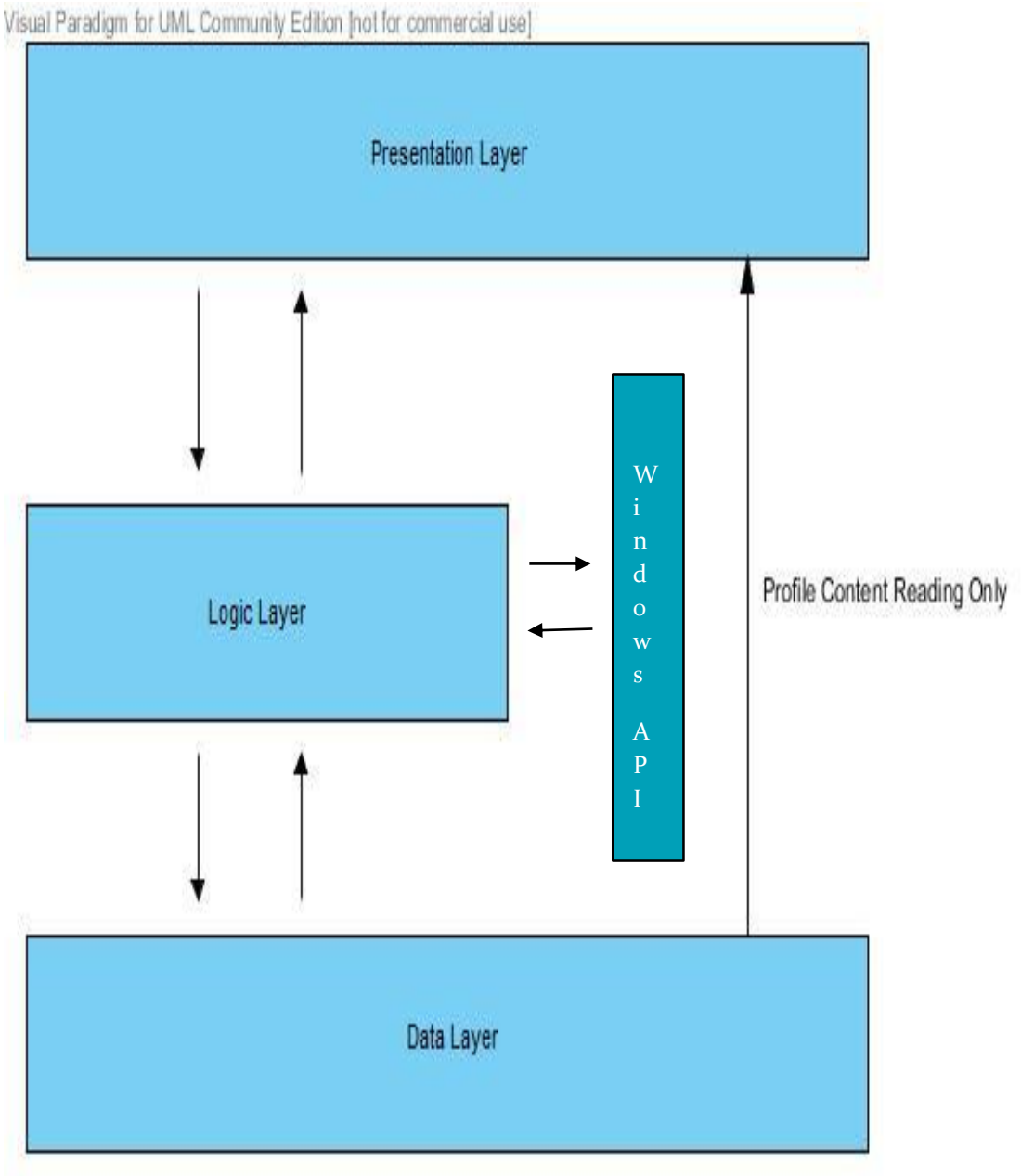

Figure 3.1

Direct Access to Data Layer from Presentation Layer

From Presentation layer Data layer is directly accessible. This decision was taken to enhance the performance and efficiency. But access privileges are restricted to read only to ensure integrity of stored data.

Only the Logic Layer Communicate with the Windows API

At the communication with the Windows API, the application should be careful otherwise security breaches can occur. To ensure security is guaranteed while communicating with windows API, Logic layer (where there is a reference monitor) is only have the access to Windows API.

NOTE:

Contemporarily these decisions may not seem to have a significant impact on the application or on the platform it runs. But in future upgrades, enhancements, and changes these design decisions will play a quite important and essential role.

Package Diagram of the Software Architecture Design

In the above three tier architecture (adopted version) of the 'Net Profile Switcher 1.0' is shown. But there it gives only an abstract idea of the design. In order to proceed with the implementation more detailed design diagram was required. So I drew a package diagram of the intended software architecture of the application which is shown below.

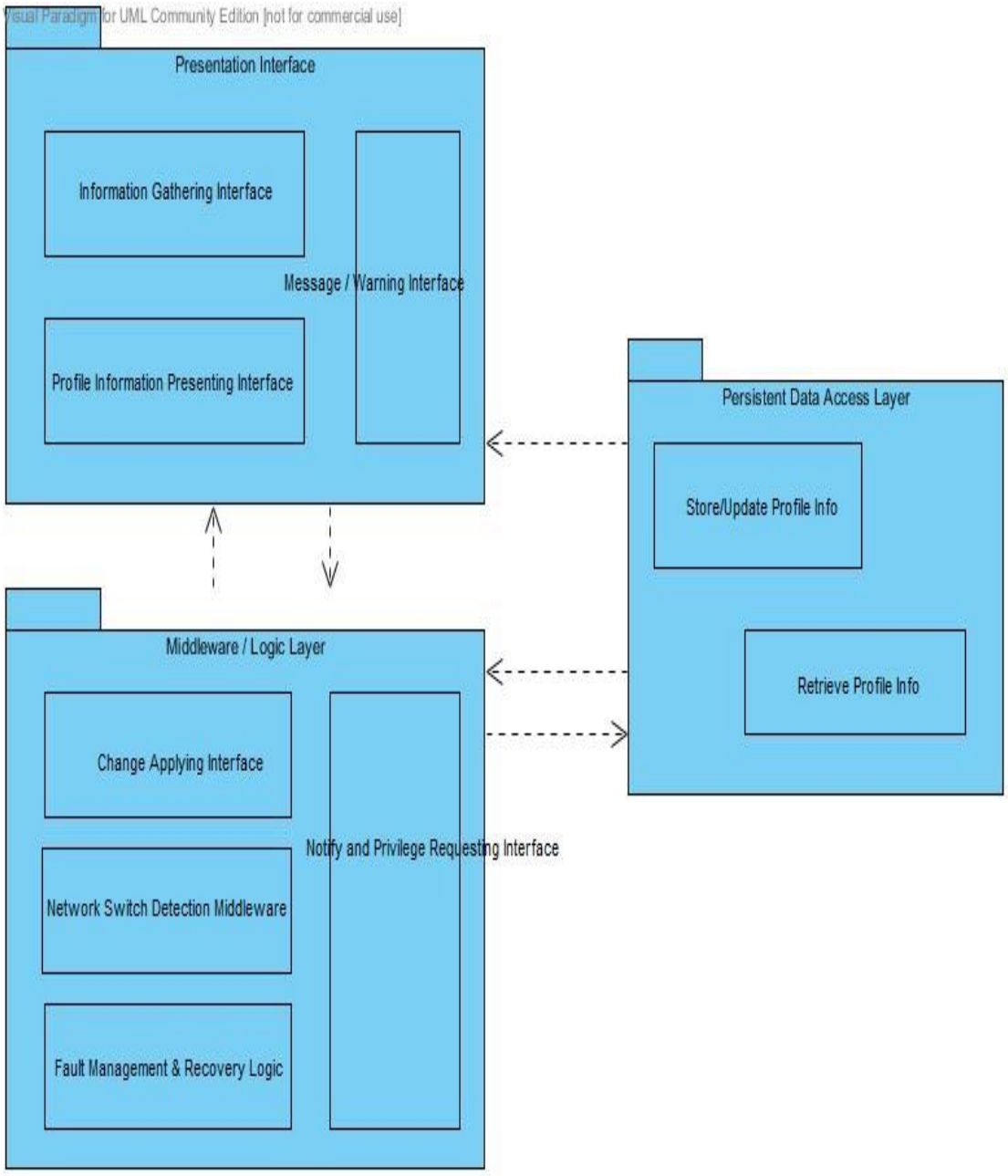

Figure 3.2

Presentation Interface

- Information Gathering Inter face
- Profile Information Presenting Interface

Presentation Interface Package there are two interfaces are shown as above. But physically (=in visibility) there is only one interface (GUI) which act as both 'Information Gathering Interface' and 'Profile Information Presenting' Interfaces according to the situation. In reality same GUI is used to gather information from the user and also to show the details of the persistently stored information. Although the GUI is same there are two different logical implementations for both of the interfaces. That's reason of adding above two interface sub-packages in 'Presentation Interface' main package.

- Message / Warning Interface

This one was a separate sub-package both in physical and virtual aspect. Basically Message Boxes are used to show messages, warnings and errors. This interface is used to get error free user inputs and to prevent user from doing unnecessary and disallowed caterings.

Middleware / Logic Layer

As the name suggests middleware layer or else logic layer handles almost all the communication with presentation, data, and windows API. There are several sub-packages providing different kind of functionality.

Change Applying Interface

When applying system network settings changes it happens via this sub-package. It is also a reference monitor which provide required security by enforcing restriction where it needed.

Network Change Detection Middleware

Some network status changes of the system (usually the PC with an OS) should be detected and necessary steps should be taken such as notifying the user, retrieving the data and etc. This process happens via this sub-package.

Functionality of other two sub packages are obvious as their name suggests.

Persistent Data Access Layer

This layer or main package consists of two sub-packages named 'Store/Update Profile Info' and 'Retrieve Profile Info'. Functionality is implied by the name of those two packages.

Important Implementation Details

In the 'Net Profile Switcher 1.0' Changing the Proxy Settings and Changing the Network Settings are some of the most significant functionalities. In the following I have shown the code segments which implement the above functionalities.

CHANGING PROXY SETTINGS

Prototypes of relevant Methods

```
///<summary>
///set the manually entered proxy setting in the registry
///</summary>
///<param name="http">httpProxyServer:port</param>
///<param name="secure">secure(https)ProxrServer:port</param>
///<param name="ftp">ftpProxyServer:Port</param>
///<param name="socks">sockProxyServer:port</param>
///<param name="byPassLocal">enable/disable by passing proxy for local
addresses</param>
///<param name="proxyExceptions">proxy exceptions</param>
///<param name="autoDetect">enable/disable automatic proxy detection</param>
public bool setManualProxy(bool sameproxy, string proxy, bool byPassLocal,
string proxyExceptions, bool autoDetect)


///<summary>
///set automatic proxy configuration using the given(by the URL) proxy
configuration script
///</summary>
/// <param name="ScriptURL">URL of automatic proxy configuration script</param>
///<param name="byPassLocal">enable/disable by passing proxy for local
addresses</param>
///<param name="proxyExceptions">proxy exceptions</param>
///<param name="autoDetect">enable/disable automatic proxy detection</param>
public bool setProxyScript(string ScriptURL, bool byPassLocal, string
proxyExceptions, bool autoDetect)
```

These two methods with the help of some more other methods allows the 'Net Profile Switcher 1.0' to apply system proxy changes. Applying changes is done by altering the 'registry keys' of Windows OS. Misuse or invalid configuration application to the system registry keys might lead to a malfunctioning of the system. In the worst case it could lead to a system crash. Thus Invalid setting application should be restricted.

'Microsoft.Win32' [10] library provides necessary mechanisms for sound and safe access altering of registry keys using registry class [11] of Win32 library. Therefore Microsoft Win32 library is used in 'Net Profile Switcher 1.0' to access. A code segment where registry keys are accessed is shown in the following

Code Segment Accessing System Registry Keys Using 'Micrsoft Win32' Library

```
using Microsoft.Win32;


//accessing registry key
RegistryKey registry =
Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\
Internet Settings", true);

//enabling proxy
registry.SetValue("ProxyEnable", 1);

//assignment of proxy server
registry.SetValue("ProxyServer", proxy);

#region proxy exception
if (byPassLocal)
{
	if (string.IsNullOrEmpty(proxyExceptions))
		registry.SetValue("ProxyOverride", "<local>");
	else
	{
		string exceptions = string.Concat("<local>;", proxyExceptions);
		registry.SetValue("ProxyOverride", exceptions)
	}
}
else
{
	if (!string.IsNullOrEmpty(proxyExceptions))
		registry.SetValue("ProxyOverride", proxyExceptions);
	else
		registry.SetValue("ProxyOverride", string.Empty);
}
#endregion
```

CHANGING NETWORK SETTINGS

Prototypes of relevant Methods

```
/// <summary>
/// provide the network card configuration of the specified Network Interface
Card (NIC)
/// </summary>
/// <param name="nicName">Name of the NIC</param>
/// <param name="ipAdresses">Array of IP</param>
/// <param name="subnets">Array of subnet masks</param>
/// <param name="gateways">Array of gateways</param>
/// <param name="dnses">Array of DNS IP</param>
public static void GetIP(string nicName, out string[] ipAdresses, out string[]
subnets, out string[] gateways, out string[] dnses)
```

```
/// <summary>
/// Set IP for the specified network card name
/// </summary>
/// <param name="nicName">Caption of the network card</param>
/// <param name="IpAddresses">Comma delimited string containing one or more
IP</param>
/// <param name="SubnetMask">Subnet mask</param>
/// <param name="Gateway">Gateway IP</param>
/// <param name="DnsSearchOrder">Comma delimited DNS IP</param>
/// <returns>int[3],[0]-setIp return, [1]-setGateway return, [2]-
setDNSsearchOrder return</returns>
public static int[] SetIP(string nicName, string IpAddresses, string
SubnetMask, string Gateway, string DnsSearchOrder)
```

```
/// <summary>
/// Enable DHCP on the NIC
/// </summary>
/// <param name="nicName">Name of the NIC</param>
/// <returns>setDHCP return</returns>
public static int SetDHCP(string nicName)
```

Above Methods provide necessary and sufficient functionalities to apply changes to Network Settings. Basically the IP Address, Subnet Mask, Gateway, DNS Server entries of a specified Network Interface Card (NIC). Application of these settings should be handle with care. Invalid and Incorrect entries would lead to malfunctioning of the communication with the network.

Avoiding or checking for Invalid entries is an important and desired requirement. Thus 'ManagementObject Class' [12] along with 'Win32_NetworkAdapterConfiguration Class' [13] is used for accessing and altering of network interface Card Settings. This combination provides sound and safe mechanisms for accessing and altering network interface card

configuration settings. It doesn't allow Invalid/Incorrect Entries for the network interface card. Instead of applying changes it returns an error at the encounter of invalid entry without applying changes.

In the following I have shown a code segment which is used to access and alter configuration settings of a specified network interface card.

A Code Segment Where NIC configuration Settings are Accessed

```
using System.Management;

//accessing Win32_NetworkAdapterConfiguration class
ManagementClass mc = new ManagementClass("Win32_NetworkAdapterConfiguration");
ManagementObjectCollection moc = mc.GetInstances();

foreach (ManagementObject mo in moc)
{
if (mo["Caption"].Equals(nicName))
{
       //accessing the properties of given NIC
       ManagementBaseObject newIP = mo.GetMethodParameters("EnableStatic");
       ManagementBaseObject newGate = mo.GetMethodParameters("SetGateways");
       ManagementBaseObject newDNS =
mo.GetMethodParameters("SetDNSServerSearchOrder");

       //changing the properties of the management object of the given NIC
       newGate["DefaultIPGateway"] = Gateway.Split(';');
       newGate["GatewayCostMetric"] = new int[] { 1 };

       newIP["IPAddress"] = IpAddresses.Split(';');
       newIP["SubnetMask"] = SubnetMask.Split(';');

       newDNS["DNSServerSearchOrder"] = DnsSearchOrder.Split(';');

       ManagementBaseObject setIP = mo.InvokeMethod("EnableStatic", newIP,
null);
       int IPRtn = (int)(UInt32)setIP["RETURNVALUE"];

       //applying setting changes
       ManagementBaseObject setGateways = mo.InvokeMethod("SetGateways",
newGate, null);
       int GatewayRtn = (int)(UInt32)setGateways["RETURNVALUE"];

       ManagementBaseObject setDNS = mo.InvokeMethod("SetDNSServerSearchOrder",
newDNS, null);
       int DNSrtn = (int)(UInt32)setDNS["RETURNVALUE"];

       int[] rtn = { IPRtn, GatewayRtn, DNSrtn };
       return rtn;
}
}
```

```
int[] error = {-1, -1, -1};
return error;
```

NOTE:

Before Coming to these method user inputs/stored data is refined at UI layer. User Input validation at user interface level is enforced in order to avoid incorrect and invalid inputs. Therefore frequency of occurring errors at this level is minimized significantly.

DATA PERSISTANCE

Storing entered network profile Information in a persisting storage is also a functional requirement of 'Net Profile Switcher 1.0'. Local, light-weight database is the perfect match for this app since it is a stand-alone software. The reasons for going for a database without using a text file or XML file are applying future upgrades or changes, and the chosen 'SQLite' [14] database is just also a light-weight file with .db extension which supports sql.

Three tables was designed and 'profileName' is set as the primary key of all three tables. The structure of the three tables are as follows.

Network Table

Name	Type
profileName	varchar(100)
nicName	varchar(200)
dhcp	tinyint(1)
manual	tinyint(1)
ipAddress	varchar(200)
subnetMask	varchar(200)
gateway	varchar(200)
dnsServer	varchar(200)

This table is used to store user entered values in the network page of 'Net Profile Switcher 1.0'

Proxy Table

Name	Type
profileName	varchar(100)
enableProxy	tinyint(1)
enableManual	tinyint(1)
sameProxy	tinyint(1)
http	varchar(100)
https	varchar(100)
ftp	varchar(100)
socks	varchar(100)
byPassProxyForLocal	tinyint(1)
enableScript	tinyint(1)
scriptURL	varchar(200)
clearAtActivation	tinyint(1)
autoDetect	tinyint(1)
enableException	tinyint(1)
exception	varchar(300)

This table is used to store user entered values in the proxy page of 'Net Profile Switcher 1.0'

Remark Table

Name	Type
profileName	varchar(100)
remark	varchar(500)

This table is used to store user entered values in the remark page of 'Net Profile Switcher 1.0'

Prototypes of some methods accessing database

```
/// <summary>
/// Insert the given networkpage details in to the network Table in database
/// </summary>
/// <param name="npd">NetworkPageDetails instance containig network page
details</param>
/// <returns>DatabaseMessage</returns>
public static DatabaseMessage Insert_to_networkTable(NetworkPageDetails npd)
```

```
/// <summary>
/// retreives the first row from the network Table which contains the given
'profileName'
/// </summary>
/// <param name="profileName">profileName of which profile network page details
desired</param>
/// <returns>NetworkPageDetails instance containing retreived network page
details</returns>
public static NetworkPageDetails Read_networkTable(string profileName)

/// <summary>
/// enum of message related to database connection and transactions
/// </summary>
public enum DatabaseMessage
{
       noProfileNameError,
       databaseError,
       proxyenableUnassignedError,
       successfulInsertion,
       insertionFailed,
       entryDoesNotExists,
       entryExists,
};
```

Several no. of methods are used to inorder to persist and retrieve information to and from the database. Enumerator named 'DatabaseMessage' is created to return value to inform the success or failure of the database transactions adhering to standards.

Taking Precautions for sql Injection attacks possible

With the intention of preventing sql injections, 'Prepared Statements' are used whilst preparing sql queries. An Example code segment which highlight using of prepared statements is shown below.

```
string ConString =
ConfigurationManager.ConnectionStrings["ConString"].ConnectionString;
SQLiteConnection conn = null;

try
{
conn = new SQLiteConnection(ConString);
conn.Open();

SQLiteCommand delCmd = new SQLiteCommand();
delCmd.Connection = conn;
delCmd.CommandText = "DELETE FROM network WHERE profileName = @profileName";
delCmd.Prepare();
delCmd.Parameters.AddWithValue("@profilename", profileName);
int affectedRows = delCmd.ExecuteNonQuery();
```

```
if (conn != null)
       conn.Close();

if (affectedRows > 0)
       return DatabaseMessage.entryExists;

return DatabaseMessage.entryDoesNotExists;
}


catch (SQLiteException ex)
{
if (conn != null)
       conn.Close();

return DatabaseMessage.databaseError;
}


finally
{
if (conn != null)
       conn.Close();
}
```

NETWORK STATUS CHANGE DETECTION

One of the major functionality of 'Net Profile Switcher 1.0' is automatic detection of switching of networks. 'Net Profile Switcher 1.0' detects networks switching, connection establishment to a network, disconnection from a network and take necessary steps. At a detection of switching networks and if the network profile already persist in the database then the app retrieves the information and pops up giving a notification to the user. Then the user just have to click the activate button if the user want to change the settings. If the profile doesn't exists the app pops up and request user to provide network configuration settings.

This Task was achieved using an even handler of 'System.Net' Library. At the startup of the app the event handler for network status change detection should get registered. It was done by adding the following code snippet in the constructor of the main Windows Form.

```
//registering for network availability change event triggers
NetworkChange.NetworkAvailabilityChanged += new
NetworkAvailabilityChangedEventHandler(NetworkChange_NetworkAvailabilityChanged
);
```

At an event trigger the following event handling method is called. It retrieves connectivity level of the connection and the SSID of the network to which the user is connected and passes those information to another custom defined method (named 'atNetworkCahnge') which proceed with notifying user and etc.

Network availability change event handling method

```
/// <summary>
/// Network availability change event handling method
/// </summary>
/// <param name="sender">sender object</param>
/// <param name="e">NetworkAvailabilityEventArgs</param>
private void NetworkChange_NetworkAvailabilityChanged(object sender,
NetworkAvailabilityEventArgs e)
{
       . . . . .

       switch (connectionProfile.GetNetworkConnectivityLevel())
       {
              case NetworkConnectivityLevel.None : continue;
              case NetworkConnectivityLevel.LocalAccess:
              {
                     //calling the atNetworkChange method by passing necessary
parameters
                     atNetworkChange(connectionProfile.ProfileName , "local");
                     return;
              }
              . . . .
       }
```

```
. . . .

}
```

Custom defined 'atNetworkChange' method

```
/// <summary>
/// at a network change this method is called by the event handling method
/// </summary>
/// <param name="profileName">SSID of the network</param>
/// <param name="accesssType">access type of the connected network</param>
private void atNetworkChange(string profileName, string accesssType)
{
if(string.IsNullOrEmpty(profileName))
{
       MessageBox.Show("Network status changed : Not connected to a network");
       return;
}
else if (ProfileSelection_ComboBox.Items.Contains(profileName))
{
       ProfileSelection_ComboBox.Text = profileName;
       this.Show();
       MessageBox.Show("Network Status Change Detected");
       return;
}
else
{
       ProfileSelection_ComboBox.Items.Add(profileName);
       ProfileSelection_ComboBox.Text = profileName;
       this.Show();
       MessageBox.Show("Network Status Change Detected");
       return;
}
}
```

Evaluation - (Testing & Results) - Performance

DEVELOPER TESTING

In the software evaluation criteria testing plays and major role. Testing can be classified as developer testing, user testing (alpha and beta testing) and etc. While developing this application two testing approaches were conducted along with debugging. First approach was unit testing which comes under developer testing. Unit testing allows to test as its name suggests methods and classes. The advantage is once the test cases are written, the used unit testing framework keep checking throughout the project whether there is a failure in written unit test cases. Therefore bugs can be found early and corrected them.

'NUnit' testing framework [15] was used at the development phase of the 'Net Profile Switcher 1.0'. In the Following I have shown a screenshot of running the written test cases in the 'NUnit' browser.

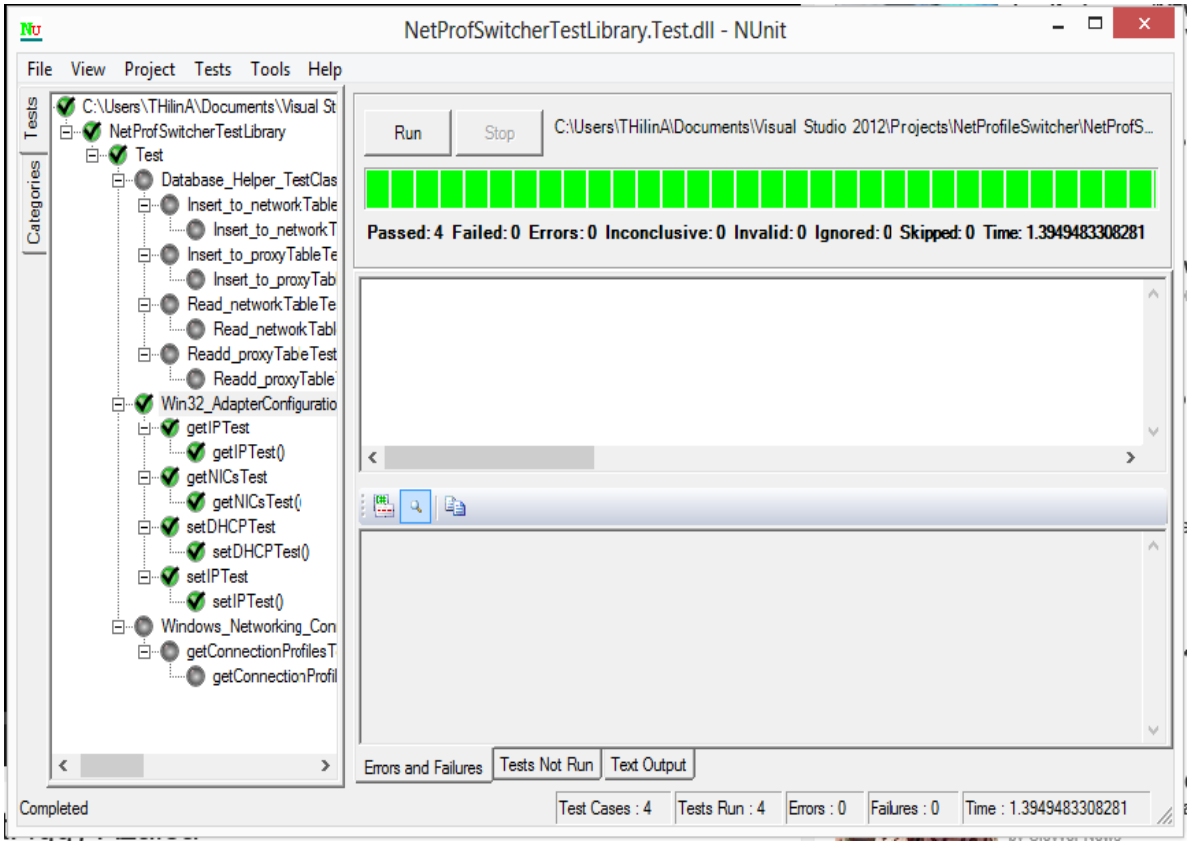

Figure 5.1.0

END USER TESTING

End user testing is also pretty important in software testing. Normally though there are three categories of end user testing such as developer level, alpha, and beta testing. In 'Net Profile Switcher 1.0' user testing happens throughout the project. In between the implementation obviously after implementing the core functionality several releases were made and given them for my friend to use. They used it and gave me feedback (verbal feedback) and do necessary changes according to them. It also helped me to improve simplicity and learnability of the software because in their feedback they also suggested many improvements on this software.

PERFORMANCE

Since the targeted platform for this app is Windows and it is for latest versions of windows, there was no big issue in performance hence the app only needs very less processing power. But there is a mechanism for detecting network connection establishment. Thus responsiveness should be evaluated. Having this idea the software application was tested in different Windows platforms with different hardware capabilities.

Performance of the software under different circumstances described in the above showed that the app is working fine and responsiveness is at the expected level.

Conclusion

'Net Profile Switcher 1.0' network profile management software tool is a special kind of software tool which provide core functionality of network profile management tool including automatic switching of network profiles as the connected network switches.

There were several issues before developing this software. I had a dwell whether it is a good idea to start developing this kind of software after getting to know already exist software tools similar to this one. After analyzing already existing software applications I got into a conclusion that there's still a high demand for 'Net Profile Switcher 1.0' as I have proven in the literature review.

There were several issues while implementing the app. Lack of APIs is one of the major issues. In the first phase I started to develop a windows store app but eventually I found that some required APIs are not supported for windows store app. This happens because at the first place I didn't do a feasibility report. After the encounter of this matter I conducted a feasibility study and started with a Windows Form where all the required APIs and libraries are available.

I have prepared a risk list for this particular project and it's shown below. It summarized the available risks and risks involved at the development.

RISK LIST

Risk Factors and Categories	L -Low Risk Evidence	M - Medium Risk Evidence	H - High Risk Evidence	Rating (HML)
Mission and Goal Factors				
Project Fit	User Requirements are the main concern. Directly supports user missions and goals.			Low
User Factors				
Customer Involvement			Minimal user involvement; variety of end user inputs is less	High
Customer Experience		End users have some level of experience with similar apps and have variety of suggestions		Medium
Customer Acceptance	End users accept the concept (Network Profiling) behind the project			Low
Customer Training Needs	Minimum level of end user training/experience required			Low
Project Size		Medium, moderate complexity, decomposable		Medium
Hardware Constraints			Significant hardware-imposed constraints; Windows PC Platforms only (except early versions)	High
Technology		Highly capable with existing technologies. As technology evolves changes in requirements occurs (Need of more functionality and less UI)		Medium
Supplied Components		Components working successfully under most circumstances		Medium
Requirements Stability	little or no change expected to approved set (baseline)			Low
Requirements Complete and Clear	all completely specified and clearly written			Low
System Testability			parts of the system bit hard to test in	High

			full and minimal planning being done for system testing	
Design Difficulty	well defined interfaces; design well understood			Low
Implementation Difficulty			Lack of Libraries and required APIs. Algorithms and/or design have elements somewhat difficult to implement.	High
Test Capability		Unit testing support with variety of unit testing frameworks. The design of the system constraints some tests at development phase. User validation and testing required		Medium
Functionality		good functionality, meets most customer needs		Medium
External Hardware or Software Interfaces			Several APIs along with some libraries required. Some API should be available at OS. Lack of these software interfaces may lead to catastrophic situations	High
Hardware Platform	stable, no changes expected, capacity is sufficient			Low
Change Implementation	in place can be responsive to customer needs			Low
Vendor Support	complete support at reasonable price and in needed time frame			Low

Table 7.1.0

References

[1] http://www.netsetman.com/

[2] http://www.controlplaneapp.com/

[3] http://intelloware.com/QuickConfig.aspx

[4] http://code.google.com/p/netprofiles/

[5] http://smartipprofiler.wordpress.com/

[6] http://www.eusing.com/ipswitch/fishelp.htm

[7] http://www.tyxsoft.com

[8] http://www.easynetswitch.com/

[9] http://www.devlogiciels.net

[10] http://msdn.microsoft.com/en-us/library/microsoft.win32(v=vs.110).aspx

[11] http://msdn.microsoft.com/en-us/library/microsoft.win32.registrykey(v=vs.110).aspx

[12] http://msdn.microsoft.com/en-us/library/System.Management(v=vs.110).aspx

[13] http://msdn.microsoft.com/en-us/library/aa394217(v=vs.85).aspx

[14] http://www.sqlite.org/

[15] http://nunit.org/