Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего профессионального образования

«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет . информационных технологий

Кафедра . прикладная математика .

Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_

Руководитель

проекта (работы) \_\_\_\_\_\_\_ Старолетов М.С.\_\_\_\_\_\_

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2013 г.

**Проектирование и реализация системы охранной сигнализации**

**с определением движения на объекте**

Пояснительная записка

к курсовой работе

по дисциплине Проектирование сетевых и многопоточных приложений

**КР 230105.16.000ПЗ**

Студент группы \_\_\_\_\_\_\_ПОВТ-81\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ненайденко А.С.\_\_\_\_

Руководитель

проекта (работы) \_\_ \_\_\_\_\_\_\_\_\_\_ Старолетов С.М.\_\_

Барнаул 2013

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Задание на курсовую работу**  Учебная дисциплина: Проектирование сетевых и многопоточных приложений  ФИО студента: Ненайденко Александр Степанович  Группа: ПОВТ-81  Задание: Проектирование и реализация системы охранной сигнализации с определением движения на объекте.  Руководитель: Старолетов С. М. | | | | | | | | | | |
|  |  |  |  |  | **КР 230105.16.000ПЗ** | | | | | |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подпись | Дата |
| Разработал | | Ненайденко А.С. |  |  | Проектирование и реализация системы охранной сигнализации с определением движения на объекте | Лит. | | | Лист | Листов |
| Пров. | | Старолетов С.М. |  |  |  | У |  | 2 | 33 |
|  | |  |  |  | АлтГТУ, ФИТ  *гр. ПОВТ-81* | | | | |
| Н.контролер | |  |  |  |
| Утв. | | Кантор С.А. |  |  |

**Аннотация**

В данной работе представлены сведения о проектировании и разработке приложения "Охранная сигнализация с определением движения на объекте". Приложение позволяет ставить на сигнализацию компьютеры в локальной сети, и сигнализировать серверу об обнаружении движения на объекте.

Программное обеспечение написано на языке программирования С# с использованием Microsoft Visual Studio 2012 и NET Framework 4.5. Для обнаружения движения камерой используется бесплатная библиотека AForge. Исходный код приложения, а также данный отчет представлены на прилагаемом CD диске.

**Содержание**

[Введение 5](#_Toc345686078)

[1 Постановка задачи 6](#_Toc345686079)

[1.1 Развернутая постановка задачи 6](#_Toc345686080)

[2 Проектирование программного обеспечения 7](#_Toc345686081)

[2.1 Диаграмма вариантов использования 7](#_Toc345686082)

[2.2 Диаграмма развертывания 8](#_Toc345686083)

[2.3 Временная диаграмма 8](#_Toc345686084)

[2.4 Диаграмма деятельности 9](#_Toc345686085)

[3 Описание программного обеспечения 10](#_Toc345686086)

[3.1 Общий подход к реализации 10](#_Toc345686087)

[3.2 Описание приложения 11](#_Toc345686088)

[Заключение 17](#_Toc345686089)

[Список использованных источников 18](#_Toc345686090)

[Приложение А Руководство пользователя 19](#_Toc345686091)

[Приложение Б Текст программы 21](#_Toc345686093)

**Введение**

В современном мире технические и технологические достижения стали неотъемлемой частью жизни. Один из важнейших показателей прогресса - это внедрение информационных технологий**,** которое повлекло за собой переход общества в новую фазу развития – так называемое «информационное общество». Развитие информационных технологий явилось результатом широкого использования компьютерной техники в разных сферах человеческой жизни.

Компьютерные технологии дают нам огромное количество преимуществ. Они удовлетворяет наши потребности, облегчают наш труд и повышают его эффективность. Компьютерная техника рационализирует деятельность человека, организует её в соответствии с целесообразностью. Она расширяет доступ к информации. Компьютеры освобождают нас от утомительной работы, выполняя с высокой скоростью разнообразные операции и эта скорость постоянно увеличивается.

Сфера применения компьютерных технологий очень широка и практически не ограниченна. В частности использование компьютеров, отлично подходит для реализации охранной системы, на основе web-камер. В таком приложении на компьютер охранника устанавливается серверная часть приложения, а на каждый компьютер работников в кабинетах устанавливается клиентская часть. Охранник удаленно, посредством программы, ставит все подключенные компьютеры на сигнализацию, и в случаи обнаружения движения на объекте программа сигнализирует ему об этом.

# 1 Постановка задачи

## 1.1 Развернутая постановка задачи

Спроектировать и реализовать систему охранной сигнализации с определением движения на объекте с помощью web-камеры.

Охранная сигнализация имеет 2 вида объектов – сервер и клиенты. Сервер установлен у охранника и он получает данные о срабатывании датчиков движения на объектах. Каждый датчик представляет собой программу на основе библиотеки для распознавания движения (например OpenCV), которая берет данные с видеокамеры и определяет, есть ли движение. И, если есть движение сигнализирует об этом охраннику.

# 2 Проектирование программного обеспечения

## 2.1 Диаграмма вариантов использования

На рисунке 2.1 представлена диаграмма использования для нашего приложения.

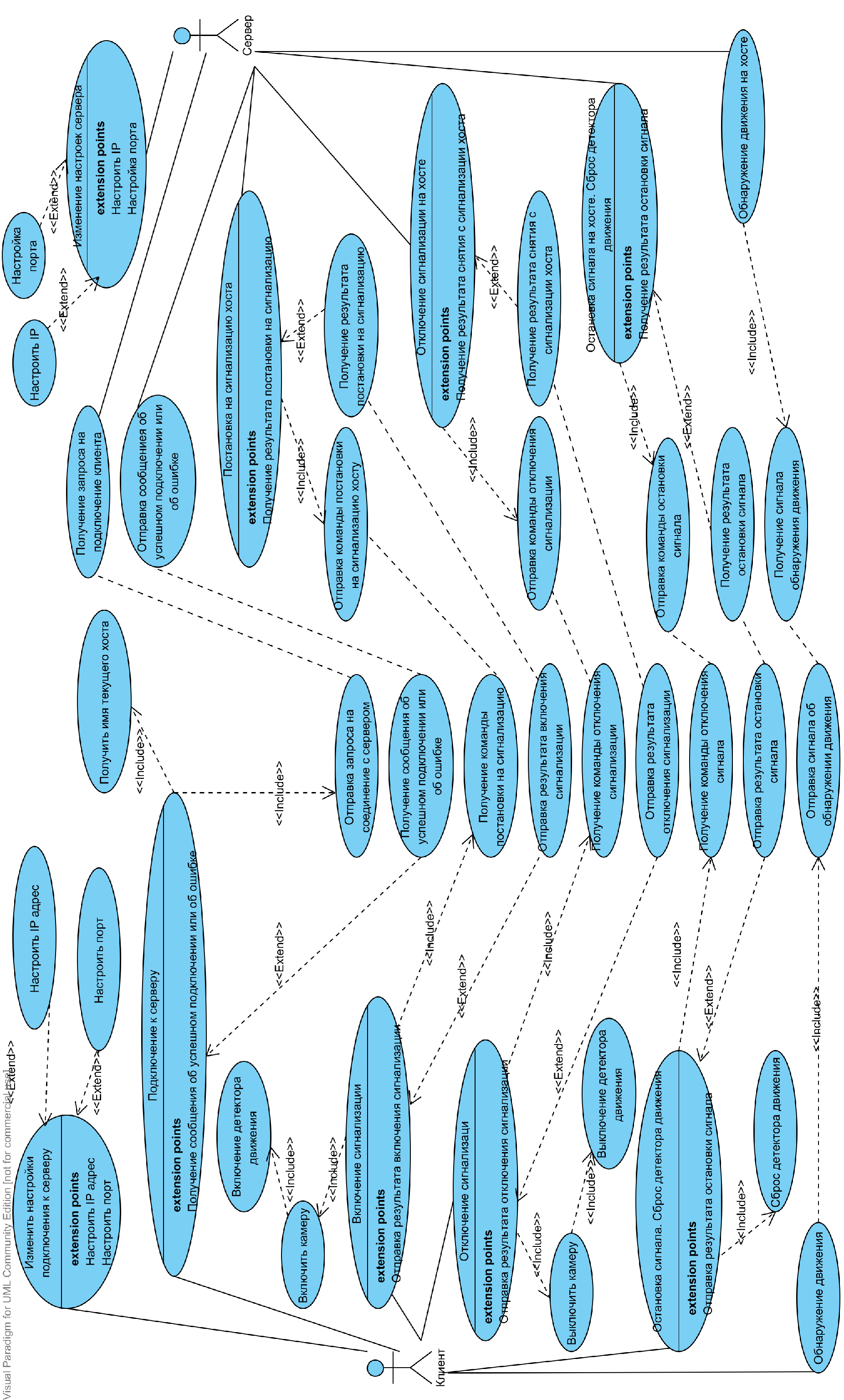


Рисунок 2.1 - Диаграмма вариантов использования

## 2.2 Диаграмма развертывания

На рисунке 2.2 представлена диаграмма развертывания приложения. В ней отраженны необходимые файлы и библиотеки для функционирования программы.

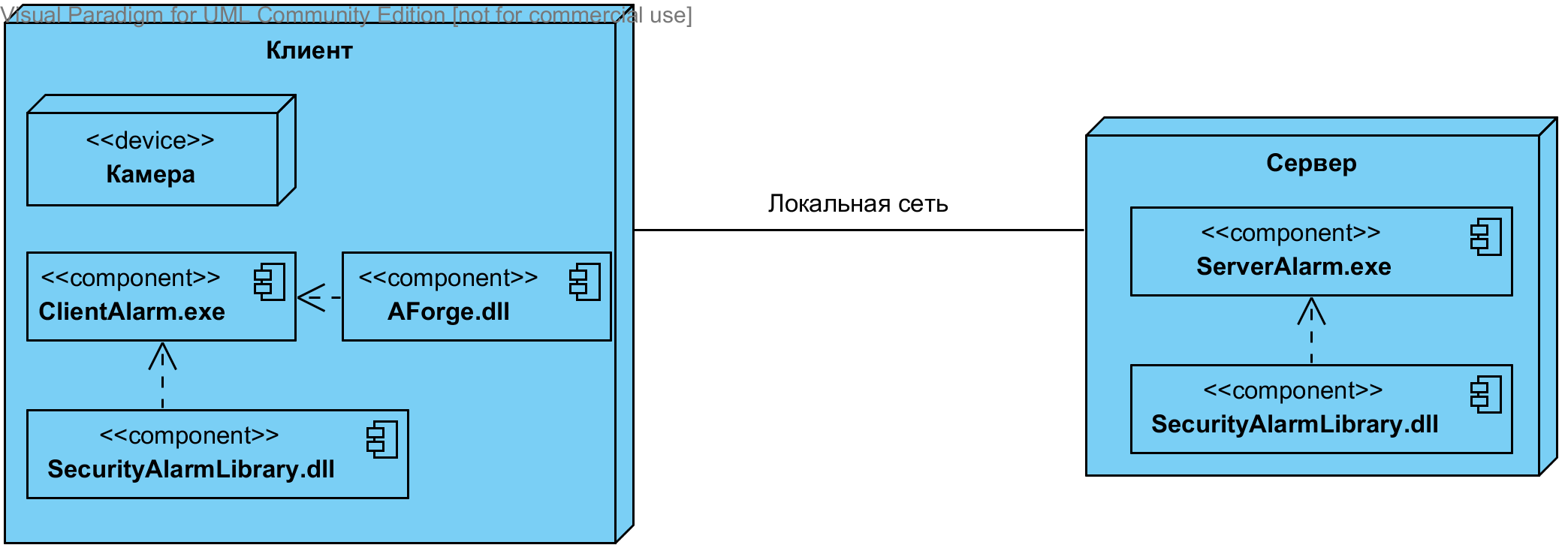


Рисунок 2.2 - Диаграмма развертывания

## 2.3 Временная диаграмма

На рисунке 2.3 представлена временная диаграмма. Данная диаграмма отображает наиболее полный вариант работы программы.

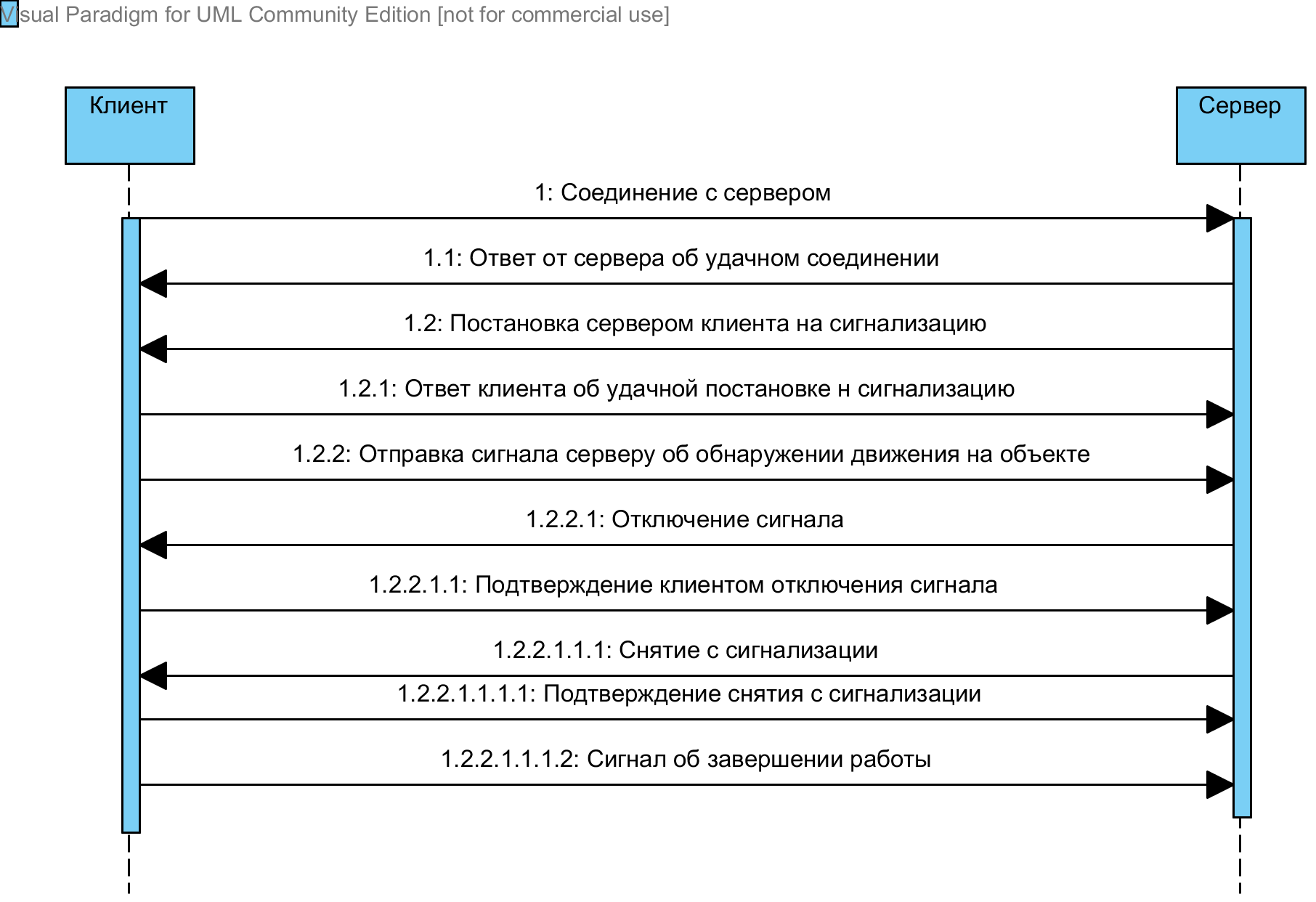


Рисунок 2.3 - Временная диаграмма

## 2.4 Диаграмма деятельности

На рисунке 2.4 представлена диаграмма деятельности.

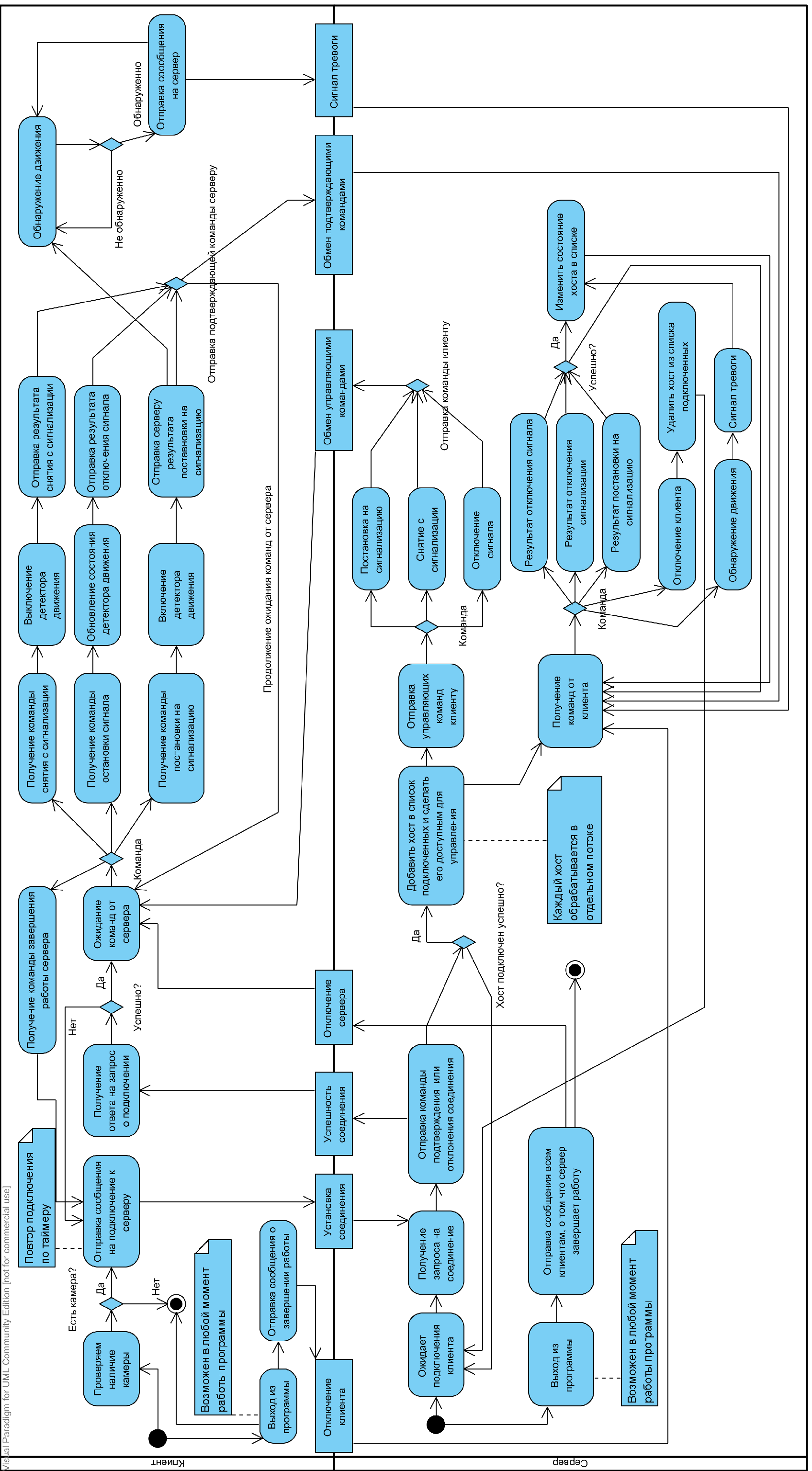


Рисунок 2.4 - Диаграмма деятельности

# 3 Описание программного обеспечения

## 3.1 Общий подход к реализации

Данный программный продукт представляет собой клиент-серверное приложение, написанное с использованием библиотеки .NET Framework 4.5 на языке C#. Для распознавания движения на объекте используется бесплатная библиотека AForge. Приложение были написано в среде Microsoft Visual Studio 2012 и протестировано в операционной системе Windows 7 64 bit и Windows Vista 32 bit.

Приложение было разделено на три части:

* серверная часть;
* клиентская часть;
* библиотека совместно используемых классов.

Серверная часть приложения отвечает за управлениями клиентами (постановка на сигнализацию, остановка сигнала, снятие с сигнализации). Сервер является многопоточным: взаимодействие с каждым клиентом происходит в отдельном потоке. Подключения принимаются также в отдельном потоке, а за отправку команд клиентам отвечает главный поток приложения.

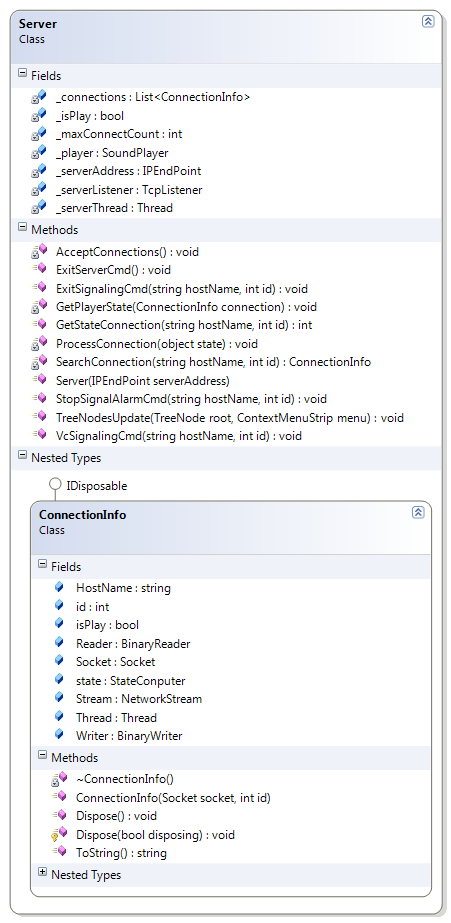
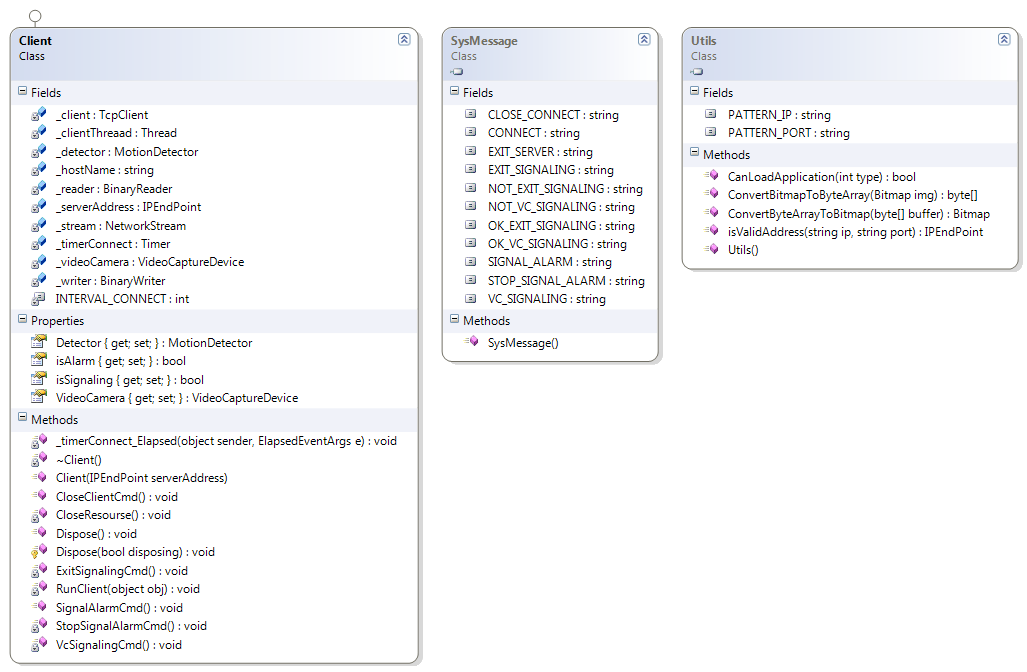
Клиентская часть приложения отвечает за обнаружение движения на объекте и отправку, в случаи обнаружения движения, этой информации серверу. Клиент является двух поточным: в первом потоке осуществляется работа GUI приложения, обнаружение движения и отправка информации на сервер, второй поток служит для принятия и обработки команд от сервера.

C# — язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. На C# легко реализуется графический пользовательский интерфейс, поэтому он был выбран для реализации приложения.

## 3.2 Описание приложения

На рисунке 3.1 приведена диаграмма основных классов, реализующих взаимодействие между сервером и клиентами.

Рисунок 3.1 - Диаграмма основных классов



Так как на диаграмме представлены поля и методы классов с указанием типов и модификаторов доступа, то ниже опишем их функциональное назначение.

Класс Client - класс реализующий клиента приложения.

Поля класса:

* private const int INTERVAL\_CONNECT - Константа, задающая через сколько тиков будет повторена попытка подключения к серверу;
* private TcpClient \_client - клиент для соединения с сервером;
* private Thread \_clientThreaad - поток в котором выполняется взаимодействие с сервером;
* private IPEndPoint \_serverAddress - адрес сервера;
* private String \_hostName - имя хоста;
* private NetworkStream \_stream - поток для передачи данных через сокеты;
* private BinaryReader \_reader - поток для чтения данных из сокета;
* private BinaryWriter \_writer - поток для записи данных в сокет;
* private System.Timers.Timer \_timerConnect - таймер, для повторного соединения;
* private VideoCaptureDevice \_videoCamera - видеокамера;
* private MotionDetector \_detector - детектор движения;
* public bool isSignaling - флаг, находится ли клиент на сигнализации или нет;
* public bool isAlarm - флаг, сработала ли тревога или нет.

Методы:

* public Client(...) - конструктор класса;
* void \_timerConnect\_Elapsed(...) - обработчик события на таймер;
* private void RunClient(...) - запуск клиента, выполняется в отдельном потоке;
* private void StopSignalAlarmCmd() - остановка сигнала сигнализации;
* public void SignalAlarmCmd() - включение сигнала сигнализации;
* private void VcSignalingCmd() - постановка на сигнализацию;
* private void ExitSignalingCmd() - снятие с сигнализации;
* public void CloseClientCmd() - отправка серверу сообщения о том что клиент завершает работу;
* private void CloseResourse() - освобождение ресурсов;
* public void Dispose() - уничтожение объекта;
* protected virtual void Dispose(...) - уничтожение объекта;
* ~Client() - деструктор.

Класс SysMessage - класс, представляет собой возможные варианты сообщений между клиентом и сервером.

Поля класса:

* public const String CONNECT - Соединение;
* public const String CLOSE\_CONNECT- Закрытие соединения;
* public const String VC\_SIGNALING - Включить сигнализацию на удаленном хосте;
* public const String EXIT\_SIGNALING - Выключить сигнализацию на удаленном хосте;
* public const String OK\_VC\_SIGNALING - Сигнализация успешно включена на удаленном хосте;
* public const String NOT\_VC\_SIGNALING - Сигнализация не включена на удаленном хосте;
* public const String OK\_EXIT\_SIGNALING - Сигнализация успешно выключена на удаленном хосте;
* public const String NOT\_EXIT\_SIGNALING - Сигнализация не выключена на удаленном хосте;
* public const String EXIT\_SERVER - Выключение сервера;
* public const String SIGNAL\_ALARM - Сигнал тревоги на сервер;
* public const String STOP\_SIGNAL\_ALARM - Отключить текущий сигнал тревоги и продолжить наблюдение;

Класс Utils - класс,содержит полезные функции.

Поля класса:

* public const string PATTERN\_IP - шаблон ip адреса;
* public const string PATTERN\_PORT - шаблон номера порта.

Методы класса:

* public static IPEndPoint isValidAddress(...) - проверка на корректность введеннного ip адреса и порта;
* public static byte[] ConvertBitmapToByteArray(...) - конвертирования картинки в массив байт;
* public static Bitmap ConvertByteArrayToBitmap(...) - конвертирование массива байт в картинку;
* public static bool CanLoadApplication(...) - проверка запущенно ли уже приложение или нет.

Класс Server - класс, реализующий функционал сервера.

Поля класса:

* private Thread \_serverThread - поток сервера;
* private IPEndPoint \_serverAddress - адрес сервера;
* private TcpListener \_serverListener - слушатель соединений;
* private List<ConnectionInfo> \_connections - список подключенных соединений;
* private int \_maxConnectCount - максимальное число соединений;
* private SoundPlayer \_player - плеер, для проигрывания сигнала тревоги;
* private bool \_isPlay - флаг, включен ли плеер или нет.

Методы класса:

* public Server(...) - конструктор класса;
* private void AcceptConnections() - принемает новые соединения;
* private void ProcessConnection(...) - обработка одного соединения;
* private ConnectionInfo SearchConnection(...) - поиск нужного соединения;
* public int GetStateConnection(...) - получить состояние хоста;
* public void VcSignalingCmd(...) - отправить команду на включение сигнализации;
* public void StopSignalAlarmCmd(...) - остановить сигнал тревоги;
* private void GetPlayerState(...) - получить состояние плеера;
* public void ExitSignalingCmd(...) - отправка команды отключения сигнализации;
* public void ExitServerCmd() - отправка команды всем хостам, что сервер завершает работу;
* public void TreeNodesUpdate(...) - обновление дерева соединений.

Класс ConnectionInfo - класс, отражающий одно отдельно взятое соединение с клиентом.

Поля класса:

* public Socket Socket - сокет;
* public Thread Thread - поток, в котором обрабатывается клиент;
* public String HostName - имя хоста;
* public NetworkStream Stream - поток для передачи данных через сеть;
* public BinaryWriter Writer - поток для записи;
* public BinaryReader Reader - поток для чтения;
* public int id - уникальный идентификатор соединения;
* public StateConputer state - состояние хоста;
* public bool isPlay - флаг, проигрывается ли плеер;

Методы класса:

* public ConnectionInfo(...) - конструктор класса;
* public override string ToString() - строковое представление объекта;
* public void Dispose() - уничтожение объекта;
* protected virtual void Dispose(...) - освобождение ресурсов;
* ~ConnectionInfo() - деструктор.

# Заключение

В результате работы над проектом были изучены возможности организации работы приложений по сети по протоколу TCP/IP на языке программирования C#. Также изучены возможности этого языка для реализации многопоточных приложений.

Все основные требования к приложению были реализованы.

В дальнейшем было бы не плохо организовать передачу видео серверу с каждого клиента в реальном времени и , если необходимо, его запись.

# Список использованных источников

1. Дейтел, Х. С# [Текст] : Учебный курс. / Дейтел Х., Дейтел П., Листфилд Дж., Нието Т., Йегер Ш., Златкина М. - СПб.: БХВ-Петербург, 2006. - 1056 с.: ил.
2. Шилдт, Г. C# [Текст] : Учебный курс. / Г. Шилдт – СПб: Питер, 2003 – 471 с.
3. Microsoft Developer Network Library [Электронный ресурс ] Режим доступа: http://msdn.microsoft.com/ru-ru/default.aspx

# Приложение А

# Руководство пользователя

Клиентская часть программы работает в фоновом режиме. Получить доступ к ней можно в системном трее, нажав на по иконке правой кнопкой мыши. В меню предложенны три пункта (Рисунок А.1):

* Настройка - служит для вызова формы настройки ip адреса и номера порта сервера (Рисунок А.2);
* Видео - просмотр захвата видео с камеры;
* Выход - выход из приложения.

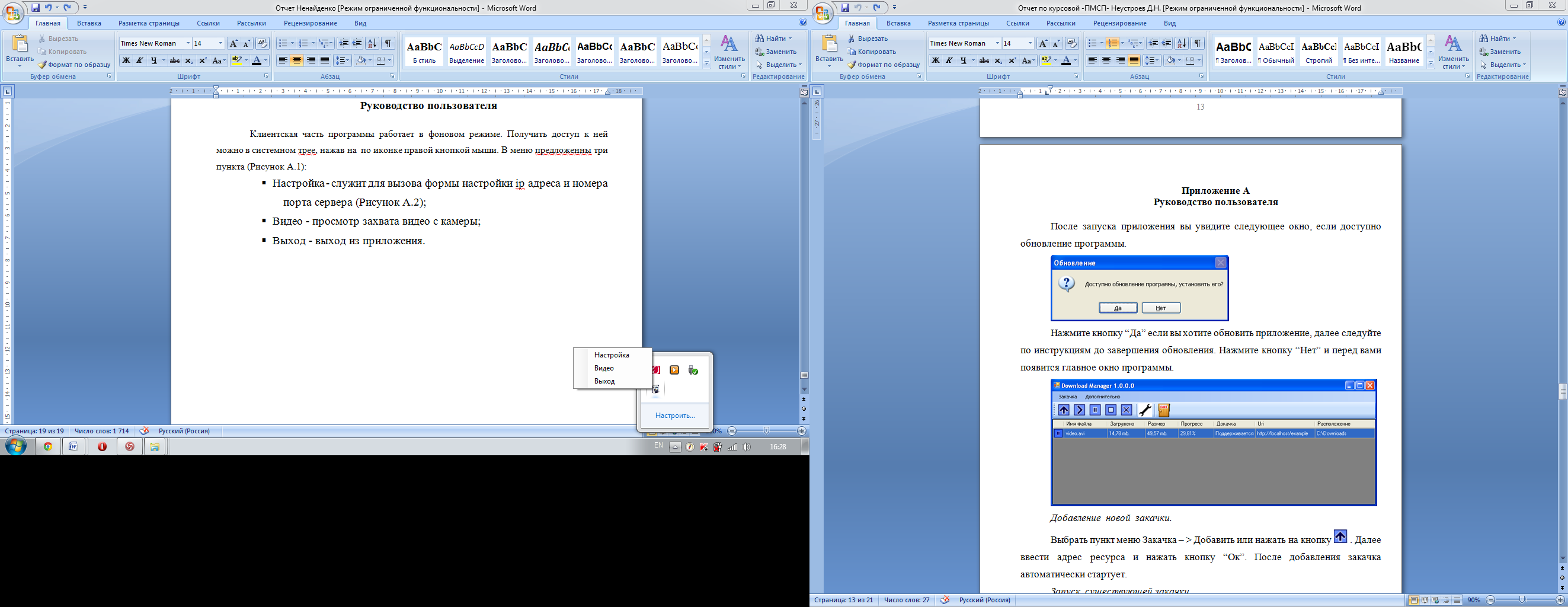


Рисунок А.1 - Меню клиента

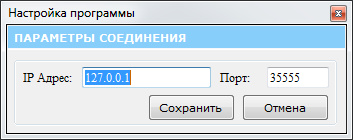


Рисунок А.2 - Настройка подключения к клиенту

Серверная часть приложения также имеет интуитивно понятный интерфейс (Рисунок А.3).

Иерархия меню представлена следующим образом:

* Меню - управление программой;
  + Настройка - служит для вызова формы настройки ip адреса и номера порта сервера;
  + Выход - выход из программы;
* Компьютер - управление отдельным выделенным хостом.
  + Поставить на сигнализацию;
  + Снять с сигнализации;
  + Отключить сигнал.

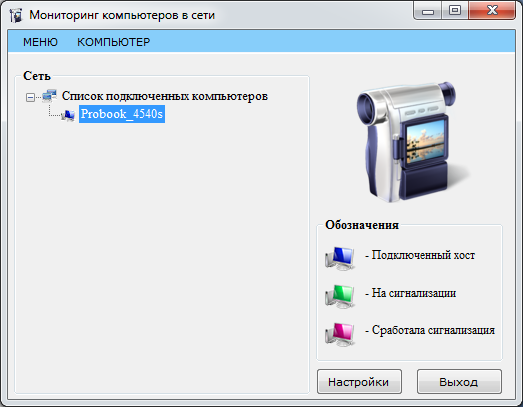


Рисунок А.3 - Главное окно серверного приложения

Текущее состояние хоста отображает картинка, условное обозначение которой также представлено на главной форме.

# Приложение Б

# Текст программы

**Сервер**

using System;

using System.Collections.Generic;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using System.IO;

using System.Windows.Forms;

using SecurityAlarmLibrary;

using System.Media;

namespace ServerAlarm

{

/// <summary>

/// Сервер

/// </summary>

public class Server

{

private Thread \_serverThread;

private IPEndPoint \_serverAddress;

private TcpListener \_serverListener;

private List<ConnectionInfo> \_connections;

private int \_maxConnectCount;

private SoundPlayer \_player;

private bool \_isPlay;

/// <summary>

/// Информация о соединении

/// </summary>

private class ConnectionInfo : IDisposable

{

/// <summary>

/// Состояния компьютера

/// </summary>

public enum StateConputer

{

Normal = 0,

OnSignal = 1,

SignalActiv = 2

}

public Socket Socket;

public Thread Thread;

public String HostName;

public NetworkStream Stream;

public BinaryWriter Writer;

public BinaryReader Reader;

public int id;

public StateConputer state;

public bool isPlay;

/// <summary>

/// Новое соединение

/// </summary>

/// <param name="socket">Сокет</param>

/// <param name="id">ID</param>

public ConnectionInfo(Socket socket, int id)

{

this.Socket = socket;

this.Stream = new NetworkStream(this.Socket);

this.Reader = new BinaryReader(this.Stream);

this.Writer = new BinaryWriter(this.Stream);

this.id = id;

this.isPlay = false;

state = StateConputer.Normal;

}

/// <summary>

/// Строковое представление

/// </summary>

/// <returns></returns>

public override string ToString()

{

if(!string.IsNullOrWhiteSpace(HostName))

return String.Format("{0}",HostName);

return "Определяется имя хоста...";

}

/// <summary>

/// Уничтожение объекта

/// </summary>

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

//Освобожение ресурсов

protected virtual void Dispose(bool disposing)

{

if (disposing)

{

if (Socket != null)

{

Socket.Dispose();

Socket = null;

}

if (Stream != null)

{

Stream.Dispose();

Stream = null;

}

if (Writer != null)

{

Writer.Dispose();

Writer = null;

}

if (Reader != null)

{

Reader.Dispose();

Reader = null;

}

}

}

//Деструктор

~ConnectionInfo()

{

Dispose(false);

}

}

/// <summary>

/// Новый сервер

/// </summary>

/// <param name="ipEndPoint">Структура: ip адрес и порт</param>

public Server(IPEndPoint serverAddress)

{

\_serverAddress = serverAddress;

\_connections = new List<ConnectionInfo>();

\_serverListener = new TcpListener(\_serverAddress);

\_player = new SoundPlayer(global::ServerAlarm.Properties.Resources.signalka5);

\_isPlay = false;

\_serverThread = new Thread(AcceptConnections) { IsBackground = true };

\_serverThread.Start();

}

//Принемаем новые соединения (обработку каждого помещаем в отдельный поток)

private void AcceptConnections()

{

\_serverListener.Start();

while (true)

{

Socket socket = \_serverListener.AcceptSocket();

\_maxConnectCount++;

ConnectionInfo connection = new ConnectionInfo(socket, \_maxConnectCount)

{

Thread = new Thread(ProcessConnection) { IsBackground = true }

};

connection.Thread.Start(connection);

lock (\_connections)

\_connections.Add(connection);

}

}

//Обработка отдельного соединения

private void ProcessConnection(object state)

{

ConnectionInfo connection = (ConnectionInfo)state;

try

{

bool flag = true;

do

{

string cmd = connection.Reader.ReadString();

if (cmd.Contains(SysMessage.CONNECT))

{

string[] str = cmd.Split(' ');

connection.HostName = str[1];

}

else

{

switch (cmd)

{

//Хост разрывает соединение

case SysMessage.CLOSE\_CONNECT:

flag = false;

break;

//Сигнализация на удаленном хосте включена успешно

case SysMessage.OK\_VC\_SIGNALING:

connection.state = ConnectionInfo.StateConputer.OnSignal;

break;

//Сигнализация на удаленном хосте не включена по кааким то причинам

case SysMessage.NOT\_VC\_SIGNALING:

connection.state = ConnectionInfo.StateConputer.Normal;

MessageBox.Show("Не удалось поставить на сигнализацию!",

connection.HostName, MessageBoxButtons.OK,

MessageBoxIcon.Error);

break;

//Сигнализация на удаленном хосте выключена успешно

case SysMessage.OK\_EXIT\_SIGNALING:

connection.state = ConnectionInfo.StateConputer.Normal;

break;

//Сигнализация на удаленном хосте выключена не успешно

case SysMessage.NOT\_EXIT\_SIGNALING:

connection.state = ConnectionInfo.StateConputer.Normal;

MessageBox.Show("Не удалось выключить сигнализацию!",

connection.HostName, MessageBoxButtons.OK,

MessageBoxIcon.Error);

break;

//Сработала сигнализация на удаленном хосте

case SysMessage.SIGNAL\_ALARM:

connection.state = ConnectionInfo.StateConputer.SignalActiv;

if (!\_isPlay)

{

if (!connection.isPlay)

\_player.PlayLooping();

connection.isPlay = true;

}

break;

}

}

} while (flag && connection.Socket.Connected);

}

catch (SocketException exp)

{

MessageBox.Show(string.Format("SocketException на хосте: {0} - {1}",connection.HostName, exp.Message));

}

catch(Exception exp)

{

MessageBox.Show(string.Format("Exception на хосте: {0} - {1}", connection.HostName, exp.Message));

}

finally

{

connection.Writer.Close();

connection.Reader.Close();

connection.Stream.Close();

connection.Socket.Close();

lock (\_connections)

\_connections.Remove(connection);

}

}

//Поиск соединения по id и номеру хоста

private ConnectionInfo SearchConnection(string hostName, int id)

{

lock (\_connections)

{

foreach (ConnectionInfo c in \_connections)

if (c.id == id && c.HostName == hostName)

return c;

return null;

}

}

/// <summary>

/// Отправить команду на постановку хоста на сигнализацию

/// </summary>

/// <param name="hostName">Имя хоста</param>

/// <param name="id">ID</param>

public int GetStateConnection(string hostName, int id)

{

ConnectionInfo connection = SearchConnection(hostName, id);

//Не нашли на выход

if (connection == null)

return 0;

//Устанавливаем состояние

switch (connection.state)

{

//На сигналке

case ConnectionInfo.StateConputer.OnSignal:

return 1;

//Сигнализация сработала

case ConnectionInfo.StateConputer.SignalActiv:

return 2;

}

//Обычное состояние

return 0;

}

/// <summary>

/// Отправить команду на постановку хоста на сигнализацию

/// </summary>

/// <param name="hostName">Имя хоста</param>

/// <param name="id">ID</param>

public void VcSignalingCmd(string hostName, int id)

{

ConnectionInfo connection = SearchConnection(hostName, id);

//Не нашли на выход

if (connection == null)

return;

//Если нашли, отправим соответствующую команду серверу

connection.Writer.Write(SysMessage.VC\_SIGNALING);

}

/// <summary>

/// Отправить команду на остановку сигнала

/// </summary>

/// <param name="hostName">Имя хоста</param>

/// <param name="id">ID</param>

public void StopSignalAlarmCmd(string hostName, int id)

{

ConnectionInfo connection = SearchConnection(hostName, id);

//Не нашли на выход

if (connection == null)

return;

GetPlayerState(connection);

connection.state = ConnectionInfo.StateConputer.OnSignal;

//Если нашли, отправим соответствующую команду серверу

connection.Writer.Write(SysMessage.STOP\_SIGNAL\_ALARM);

}

//Получить статус проигрования

private void GetPlayerState(ConnectionInfo connection)

{

bool flag = false;

lock (\_connections)

{

foreach (ConnectionInfo c in \_connections)

if (c.id == connection.id)

{

c.isPlay = false;

break;

}

foreach (ConnectionInfo c in \_connections)

if (c.isPlay)

flag = true;

}

if (!flag)

{

\_isPlay = false;

\_player.Stop();

}

}

/// <summary>

/// Отправить команду на снятие хоста с сигнализации

/// </summary>

/// <param name="hostName">Имя хоста</param>

/// <param name="id">ID</param>

public void ExitSignalingCmd(string hostName, int id)

{

ConnectionInfo connection = SearchConnection(hostName, id);

//Не нашли на выход

if (connection == null)

return;

GetPlayerState(connection);

connection.state = ConnectionInfo.StateConputer.Normal;

//Если нашли, отправим соответствующую команду серверу

connection.Writer.Write(SysMessage.EXIT\_SIGNALING);

}

/// <summary>

/// Команда всем хостам, о том что сервер выключается

/// </summary>

public void ExitServerCmd()

{

lock (\_connections)

{

foreach (ConnectionInfo c in \_connections)

c.Writer.Write(SysMessage.EXIT\_SERVER);

\_connections.Clear();

}

}

/// <summary>

/// Получить список подключенных хостов

/// </summary>

/// <param name="root">Корень дерева</param>

/// <param name="menu">Контекстное меню</param>

public void TreeNodesUpdate(TreeNode root, ContextMenuStrip menu)

{

//Получим те узлы которые уже есть в дереве

List<int> idxsTree = new List<int>();

foreach (TreeNode n in root.Nodes)

idxsTree.Add(Convert.ToInt32(n.Name));

List<int> idxsCon = new List<int>();

//Теперь добавляем новые узлы

lock (\_connections)

{

foreach (ConnectionInfo c in \_connections)

{

idxsCon.Add(c.id);

//Если уже такой узел есть

if (idxsTree.Contains(c.id))

//пропускаем

continue;

TreeNode node = new TreeNode(c.HostName, 1, 1)

{

Name = c.id.ToString()

};

node.ContextMenuStrip = menu;

root.Nodes.Add(node);

}

//Удалим страрые узлы

for (int i = 0; i < root.Nodes.Count; i++)

{

int id = Convert.ToInt32(root.Nodes[i].Name);

if (!idxsCon.Contains(id))

{

root.Nodes.Remove(root.Nodes[i]);

i--;

}

}

//Теперь у всех узлов нужно обновить состояние

foreach (ConnectionInfo c in \_connections)

{

string key = c.id.ToString();

TreeNode node = root.Nodes.Find(key, false)[0];

if (node == null)

continue;

//Учтем текущее состояние компьютера

switch (c.state)

{

//Обычное состояние

case ConnectionInfo.StateConputer.Normal:

node.ImageIndex = 1;

node.SelectedImageIndex = 1;

break;

//Стоит на сигнализации

case ConnectionInfo.StateConputer.OnSignal:

node.ImageIndex = 3;

node.SelectedImageIndex = 3;

break;

//Сигнализация сработала

case ConnectionInfo.StateConputer.SignalActiv:

node.ImageIndex = 2;

node.SelectedImageIndex = 2;

break;

}

}

}

}

}

}

using System;

using System.Net;

using System.Windows.Forms;

using SecurityAlarmLibrary;

namespace ServerAlarm

{

public partial class MainServerForm : Form

{

private const int INTERVAL\_UPDATE = 3000;

//Сервер

private Server server;

//Адресс сервера

private IPEndPoint serverAddress;

//Флаг закрытия

private bool isAppExit;

private ImageList TreeImageList;

public MainServerForm()

{

InitializeComponent();

treeView1.Nodes.Add(new TreeNode("Список подключенных компьютеров"));

TreeImageList = new ImageList();

TreeImageList.Images.Add(global::ServerAlarm.Properties.Resources.network);

TreeImageList.Images.Add(global::ServerAlarm.Properties.Resources.comp);

TreeImageList.Images.Add(global::ServerAlarm.Properties.Resources.comp\_activ);

TreeImageList.Images.Add(global::ServerAlarm.Properties.Resources.comp\_signal);

treeView1.ImageList = TreeImageList;

treeView1.ImageIndex = 0;

treeView1.SelectedImageIndex = 0;

this.isAppExit = false;

LoadSettings();

timer1.Interval = INTERVAL\_UPDATE;

timer1.Tick += UpdateTree;

}

private void LoadSettings()

{

//Загрузка настроек

String ip = global::ServerAlarm.Properties.Settings.Default.SERVER\_IP\_ADDRESS;

String port = global::ServerAlarm.Properties.Settings.Default.SERVER\_PORT;

serverAddress = null;

try

{

serverAddress = Utils.isValidAddress(ip, port);

if (serverAddress == null)

throw new FormatException("Не правильный формат ip адреса или порта");

server = new Server(serverAddress);

}

catch (FormatException fexp)

{

MessageBox.Show(fexp.Message);

}

}

//Обновление дерева клиентов

void UpdateTree(object sender, EventArgs e)

{

TreeNode root = treeView1.Nodes[0];

var sNode = treeView1.SelectedNode;

server.TreeNodesUpdate(root, nodeContextMenu);

treeView1.ExpandAll();

bool flag = false;

foreach (TreeNode node in root.Nodes)

if (node.Text == sNode.Text)

{

treeView1.SelectedNode = node;

flag = true;

}

if(!flag)

treeView1.SelectedNode = root;

}

//Загрузка программы, а вместе с ней и запуск сервера

private void MainServerForm\_Load(object sender, EventArgs e)

{

timer1.Start();

}

//Закрытие на крестик

private void MainServerForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!isAppExit)

{

e.Cancel = true;

this.Opacity = 0;

this.WindowState = FormWindowState.Minimized;

this.ShowInTaskbar = false;

}

else

e.Cancel = false;

Close();

}

//Выход (по контекстному меню)

private void toolStripMenuItem2\_Click(object sender, EventArgs e)

{

CloseProgramm();

}

//Настройка (по контекстному меню)

private void toolStripMenuItem1\_Click(object sender, EventArgs e)

{

ShowSettingsForm();

}

//Открыть программу

private void openToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (this.WindowState == FormWindowState.Minimized)

this.WindowState = FormWindowState.Normal;

this.Activate();

this.Opacity = 100;

this.ShowInTaskbar = true;

}

//Сворачиваем

private void MainServerForm\_MinimumSizeChanged(object sender, EventArgs e)

{

this.Opacity = 0;

this.WindowState = FormWindowState.Minimized;

this.ShowInTaskbar = false;

}

//Настройка

private void setToolStripMenuItem\_Click(object sender, EventArgs e)

{

ShowSettingsForm();

}

//Показать форму настройки

private void ShowSettingsForm()

{

bool flag = false;

//Запуск формы настроек

using (SettingsForm sForm = new SettingsForm())

{

//Если настройки изменили

if (sForm.ShowDialog() == DialogResult.OK)

{

flag = true;

}

}

if (flag)

{

CloseProgramm();

Application.Restart();

}

}

//Завершение работы с программой

private void CloseProgramm()

{

server.ExitServerCmd();

this.Dispose();

}

//Выход

private void clToolStripMenuItem\_Click(object sender, EventArgs e)

{

CloseProgramm();

}

//Поставить на сигнализацию

private void onSignalToolStripMenuItem\_Click(object sender, EventArgs e)

{

//Получим выделенный хост в дереве

TreeNode node = treeView1.SelectedNode;

//Если ничего не выделенно, или выделен корневой

if (node == null || node.Parent == null)

//На выход

return;

//Иначе отправляем команду

string hostName = node.Text;

int id = Convert.ToInt32(node.Name);

server.VcSignalingCmd(hostName, id);

}

//Снять с сигнализации

private void notSignalToolStripMenuItem\_Click(object sender, EventArgs e)

{

//Получим выделенный хост в дереве

TreeNode node = treeView1.SelectedNode;

//Если ничего не выделенно, или выделен корневой

if (node == null || node.Parent == null)

//На выход

return;

//Иначе отправляем команду

string hostName = node.Text;

int id = Convert.ToInt32(node.Name);

server.StopSignalAlarmCmd(hostName, id);

server.ExitSignalingCmd(hostName, id);

}

//Выбор хоста в дереве

private void treeView1\_AfterSelect(object sender, TreeViewEventArgs e)

{

//Если корневой то на выход

if (e.Node.Parent == null)

return;

HideMenuItems(e.Node);

}

//Открыть меню "Компьютер"

private void компьютерToolStripMenuItem\_Click(object sender, EventArgs e)

{

TreeNode node = treeView1.SelectedNode;

if (node == null || node.Parent == null)

{

//Заблокируем все элементы меню

onSignalToolStripMenuItem.Enabled = false;

notSignalToolStripMenuItem.Enabled = false;

stopSignalToolStripMenuItem.Enabled = false;

return;

}

HideMenuItems(node);

}

//Скрытие кнопок меню

private void HideMenuItems(TreeNode node)

{

string hostName = node.Text;

int id = Convert.ToInt32(node.Name);

int state = server.GetStateConnection(hostName, id);

//В зависимости от состояния блокируем некоторое меню

if (state == 1 || state == 2)

{

//Если на сигнализации или она сработала

onSignalToolStripMenuItem.Enabled = false;

notSignalToolStripMenuItem.Enabled = true;

if (state == 1)

stopSignalToolStripMenuItem.Enabled = false;

else

stopSignalToolStripMenuItem.Enabled = true;

}

else

{

//Если в обычном состоянии

onSignalToolStripMenuItem.Enabled = true;

notSignalToolStripMenuItem.Enabled = false;

stopSignalToolStripMenuItem.Enabled = false;

}

onSignalContextToolStripMenuItem.Enabled = onSignalToolStripMenuItem.Enabled;

noSignalContextToolStripMenuItem.Enabled = notSignalToolStripMenuItem.Enabled;

stopSignalContextToolStripMenuItem.Enabled = stopSignalToolStripMenuItem.Enabled;

}

//Поставить на сигнализацию из контекстного меню

private void onSignalContextToolStripMenuItem\_Click(object sender, EventArgs e)

{

onSignalToolStripMenuItem.PerformClick();

}

//Снять с сигнализации из контекстного меню

private void noSignalContextToolStripMenuItem\_Click(object sender, EventArgs e)

{

notSignalToolStripMenuItem.PerformClick();

}

//Отключить сигнал из контекстного меню

private void stopSignalContextToolStripMenuItem\_Click(object sender, EventArgs e)

{

stopSignalToolStripMenuItem.PerformClick();

}

//Открытие контекстного меню

private void nodeContextMenu\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

TreeNode node = treeView1.SelectedNode;

if (node == null || node.Parent == null)

return;

HideMenuItems(node);

onSignalContextToolStripMenuItem.Enabled = onSignalToolStripMenuItem.Enabled;

noSignalContextToolStripMenuItem.Enabled = notSignalToolStripMenuItem.Enabled;

stopSignalContextToolStripMenuItem.Enabled = stopSignalToolStripMenuItem.Enabled;

}

//Выбор хоста правой кнопкой мыши

private void treeView1\_NodeMouseClick(object sender, TreeNodeMouseClickEventArgs e)

{

if (e.Button == System.Windows.Forms.MouseButtons.Right)

treeView1.SelectedNode = e.Node;

}

//Отключить сигнал

private void stopSignalToolStripMenuItem\_Click(object sender, EventArgs e)

{

//Получим выделенный хост в дереве

TreeNode node = treeView1.SelectedNode;

//Если ничего не выделенно, или выделен корневой

if (node == null || node.Parent == null)

//На выход

return;

//Иначе отправляем команду

string hostName = node.Text;

int id = Convert.ToInt32(node.Name);

server.StopSignalAlarmCmd(hostName, id);

}

//Выход

private void button1\_Click(object sender, EventArgs e)

{

clToolStripMenuItem.PerformClick();

}

//Настройки

private void button2\_Click(object sender, EventArgs e)

{

setToolStripMenuItem.PerformClick();

}

}

}

using System;

using System.Net;

using System.Net.NetworkInformation;

using System.Text.RegularExpressions;

using System.Windows.Forms;

using SecurityAlarmLibrary;

using System.Net.Sockets;

namespace ServerAlarm

{

public partial class SettingsForm : Form

{

public SettingsForm()

{

InitializeComponent();

}

//Сохранить

private void button1\_Click(object sender, EventArgs e)

{

String ip = null;

String port = null;

try

{

if (comboBox1.SelectedIndex < 0)

throw new Exception("Не выбран ip адрес!");

ip = comboBox1.SelectedItem.ToString();

port = textBox2.Text.Trim();

//Проверка валидности введенных данных

IPAddress Addr = null;

if (!IPAddress.TryParse(ip, out Addr))

throw new Exception("Неверный формат IP адреса");

Regex regex = new Regex(Utils.PATTERN\_PORT);

Match match = regex.Match(port);

if (!match.Success)

throw new FormatException("Не правильный формат порта");

int Port = 0;

if (!int.TryParse(port, out Port))

throw new Exception("Не правильный формат порта");

//Сохраняем настройки в файл настроек

global::ServerAlarm.Properties.Settings.Default.SERVER\_IP\_ADDRESS = Addr.ToString();

global::ServerAlarm.Properties.Settings.Default.SERVER\_PORT = Port.ToString();

global::ServerAlarm.Properties.Settings.Default.Save();

this.DialogResult = System.Windows.Forms.DialogResult.OK;

MessageBox.Show("Настройки успешно сохранены! Программа будет перезапущенна!");

}

catch (Exception exp)

{

this.DialogResult = System.Windows.Forms.DialogResult.Cancel;

MessageBox.Show(exp.Message);

}

}

//Отмена

private void button2\_Click(object sender, EventArgs e)

{

this.DialogResult = System.Windows.Forms.DialogResult.Cancel;

this.Close();

}

//Закрытие на крестик

private void SettingsForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (this.DialogResult == System.Windows.Forms.DialogResult.OK)

return;

this.DialogResult = System.Windows.Forms.DialogResult.Cancel;

}

//Загрузка

private void SettingsForm\_Load(object sender, EventArgs e)

{

//Загрузка настроек

string ip = global::ServerAlarm.Properties.Settings.Default.SERVER\_IP\_ADDRESS;

comboBox1.Items.Clear();

foreach (NetworkInterface nic in NetworkInterface.GetAllNetworkInterfaces())

foreach (UnicastIPAddressInformation addrInfo in nic.GetIPProperties().UnicastAddresses)

if (addrInfo.Address.AddressFamily == AddressFamily.InterNetwork)

comboBox1.Items.Add(addrInfo.Address.ToString());

bool isSelected = false;

for (int i = 0; i < comboBox1.Items.Count; i++)

{

if (comboBox1.Items[i].ToString() == ip)

{

comboBox1.SelectedItem = comboBox1.Items[i];

isSelected = true;

}

}

textBox2.Text = global::ServerAlarm.Properties.Settings.Default.SERVER\_PORT;

if (!isSelected)

MessageBox.Show("Выбранный ранее ip адрес сейчас недоступен! Необходимо выбрать новый!");

}

}

}

**Клиент**

using System.Net.Sockets;

using System.Threading;

using System.Net;

using System;

using System.IO;

using SecurityAlarmLibrary;

using System.Windows.Forms;

using AForge.Video.DirectShow;

using AForge.Vision.Motion;

namespace ClientAlarm

{

/// <summary>

/// Клиент

/// </summary>

public class Client : IDisposable

{

private const int INTERVAL\_CONNECT = 5000;

private TcpClient \_client;

private Thread \_clientThreaad;

private IPEndPoint \_serverAddress;

private String \_hostName;

private NetworkStream \_stream;

private BinaryReader \_reader;

private BinaryWriter \_writer;

private System.Timers.Timer \_timerConnect;

private VideoCaptureDevice \_videoCamera;

private MotionDetector \_detector;

/// <summary>

/// На сигнализации или нет?

/// </summary>

public bool isSignaling { get; set; }

/// <summary>

/// Тревога?

/// </summary>

public bool isAlarm { get; set; }

/// <summary>

/// Камера

/// </summary>

public VideoCaptureDevice VideoCamera

{

get

{

return \_videoCamera;

}

set

{

\_videoCamera = value;

}

}

/// <summary>

/// Детектор движения

/// </summary>

public MotionDetector Detector

{

get

{

return \_detector;

}

set

{

\_detector = value;

}

}

/// <summary>

/// Новый клиент

/// </summary>

/// <param name="serverAddress">Адрес сервера</param>

public Client(IPEndPoint serverAddress)

{

\_serverAddress = serverAddress;

\_hostName = Dns.GetHostName();

\_timerConnect = new System.Timers.Timer(Convert.ToDouble(INTERVAL\_CONNECT));

\_timerConnect.Elapsed += \_timerConnect\_Elapsed;

\_clientThreaad = new Thread(RunClient) { IsBackground = true };

\_clientThreaad.Start();

}

//Повтор соединения по таймеру

void \_timerConnect\_Elapsed(object sender, System.Timers.ElapsedEventArgs e)

{

if (\_client.Connected)

return;

try

{

\_client.Connect(\_serverAddress);

}

catch (SocketException)

{

}

}

//Запуск работы клиента (в отдельном потоке принемаем данные)

private void RunClient(object obj)

{

while (true)

{

try

{

\_client = new TcpClient();

\_timerConnect.Start();

while (true)

if (\_client.Connected)

{

\_timerConnect.Stop();

break;

}

\_stream = \_client.GetStream();

\_writer = new BinaryWriter(\_stream);

\_reader = new BinaryReader(\_stream);

string helloStr = string.Format("{0} {1}", SysMessage.CONNECT, \_hostName);

\_writer.Write(helloStr);

try

{

bool flag = true;

do

{

//Считываем команды из буфера

string cmd = \_reader.ReadString();

switch (cmd)

{

//Включить сигнализацию

case SysMessage.VC\_SIGNALING:

VcSignalingCmd();

break;

//Выключить сигнализацию

case SysMessage.EXIT\_SIGNALING:

ExitSignalingCmd();

break;

//Выключение сервера

case SysMessage.EXIT\_SERVER:

CloseResourse();

flag = false;

break;

//Выключить сигнал тревоги и продолжить наблюдение

case SysMessage.STOP\_SIGNAL\_ALARM:

StopSignalAlarmCmd();

break;

}

} while (flag);

}

catch (SocketException)

{

MessageBox.Show("Соединение было неожиданно разорванно!");

CloseResourse();

}

}

catch (Exception exp)

{

CloseResourse();

MessageBox.Show(exp.Message);

}

finally

{

CloseResourse();

}

}

}

//Выключить сигнал тревоги и продолжить наблюдение

private void StopSignalAlarmCmd()

{

isAlarm = false;

if (\_detector != null)

\_detector.Reset();

}

/// <summary>

/// Отправка на сервер сигнала тревоги

/// </summary>

public void SignalAlarmCmd()

{

if (!isSignaling)

return;

isAlarm = true;

\_writer.Write(SysMessage.SIGNAL\_ALARM);

}

//Включить сигнализацию на видеокамере

private void VcSignalingCmd()

{

if (\_videoCamera != null)

if (!\_videoCamera.IsRunning)

\_videoCamera.Start();

//Если включили сигнализацию успешно отправим эту информацию серверу

if (\_videoCamera.IsRunning)

{

\_writer.Write(SysMessage.OK\_VC\_SIGNALING);

isSignaling = true;

isAlarm = false;

}

else

{

//Иначе нужно сообщить серверу о не удаче

\_writer.Write(SysMessage.NOT\_VC\_SIGNALING);

isSignaling = false;

isAlarm = false;

}

}

//Выключить сигнализацию на видеокамере

private void ExitSignalingCmd()

{

if (\_videoCamera != null)

if (\_videoCamera.IsRunning)

\_videoCamera.Stop();

//Если выключили сигнализацию успешно отправим эту информацию серверу

if (!\_videoCamera.IsRunning)

\_writer.Write(SysMessage.OK\_EXIT\_SIGNALING);

else

//Иначе нужно сообщить серверу о не удаче

\_writer.Write(SysMessage.NOT\_EXIT\_SIGNALING);

isSignaling = false;

isAlarm = false;

}

/// <summary>

/// Отправка на сервер команды об отключении клиента

/// </summary>

public void CloseClientCmd()

{

if(\_client != null && \_client.Connected)

\_writer.Write(SysMessage.CLOSE\_CONNECT);

}

//Закрытие ресурсов

private void CloseResourse()

{

//Закрытие соединения

\_writer.Close();

\_reader.Close();

\_stream.Close();

\_client.Close();

isAlarm = false;

isSignaling = false;

if (\_videoCamera.IsRunning)

\_videoCamera.Stop();

}

//Уничтожение объекта

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

//Освобождение памяти

protected virtual void Dispose(bool disposing)

{

if (disposing)

{

if (\_stream != null)

{

\_stream.Dispose();

\_stream = null;

}

if (\_reader != null)

{

\_reader.Dispose();

\_reader = null;

}

if (\_writer != null)

{

\_writer.Dispose();

\_writer = null;

}

if (\_timerConnect != null)

{

\_timerConnect.Dispose();

\_timerConnect = null;

}

}

}

//Деструтор

~Client()

{

Dispose(false);

}

}

}

using System;

using System.Net;

using System.Text.RegularExpressions;

using System.Windows.Forms;

using SecurityAlarmLibrary;

using AForge.Video.DirectShow;

using AForge.Vision.Motion;

namespace ClientAlarm

{

public partial class ClientSettingsForm : Form

{

//Адрес сервера

private IPEndPoint serverAddress;

//Клиент

private Client client;

//Флаг закрытия

private bool isAppExit;

private FilterInfoCollection videoDevices;

private VideoCaptureDevice videoCamera;

private MotionDetector detector;

public ClientSettingsForm()

{

InitializeComponent();

this.Opacity = 0;

this.WindowState = FormWindowState.Minimized;

this.isAppExit = false;

}

//Создание клиента

private void ClientSettingsFormExtracted()

{

//Загрузка настроек

textBox1.Text = global::ClientAlarm.Properties.Settings.Default.SERVER\_IP\_ADDRESS;

textBox2.Text = global::ClientAlarm.Properties.Settings.Default.SERVER\_PORT;

client = null;

serverAddress = null;

try

{

serverAddress = Utils.isValidAddress(textBox1.Text, textBox2.Text);

if (serverAddress == null)

throw new FormatException("Не правильный формат ip адреса или порта");

client = new Client(serverAddress);

if (videoCamera.IsRunning)

videoCamera.Stop();

client.VideoCamera = this.videoCamera;

client.Detector = this.detector;

}

catch (FormatException fexp)

{

MessageBox.Show(fexp.Message);

}

}

//Настройка программы

private void settingsToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (this.WindowState == FormWindowState.Minimized)

this.WindowState = FormWindowState.Normal;

this.Activate();

this.Opacity = 100;

}

//Выход из программы

private void closeToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (videoCamera!=null && videoCamera.IsRunning)

videoCamera.Stop();

client.CloseClientCmd();

this.Dispose();

}

//Сворачивание в трей при нажатии на крестик

private void ClientMainForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!isAppExit)

e.Cancel = true;

else

e.Cancel = false;

this.Opacity = 0;

this.WindowState = FormWindowState.Minimized;

}

//Разворот формы

private void ClientMainForm\_MaximumSizeChanged(object sender, EventArgs e)

{

//Загрузка настроек

textBox1.Text = global::ClientAlarm.Properties.Settings.Default.SERVER\_IP\_ADDRESS;

textBox2.Text = global::ClientAlarm.Properties.Settings.Default.SERVER\_PORT;

}

//Сохранить настройки

private void button1\_Click(object sender, EventArgs e)

{

//Проверим на валидность введенные значения

try

{

string ip = textBox1.Text.Trim();

string port = textBox2.Text.Trim();

Regex regex = new Regex(Utils.PATTERN\_IP);

Match match = regex.Match(ip);

if (!match.Success)

throw new FormatException("Не правильный формат ip адреса");

regex = new Regex(Utils.PATTERN\_PORT);

match = regex.Match(port);

if (!match.Success)

throw new FormatException("Не правильный формат порта");

IPEndPoint server = Utils.isValidAddress(ip, port);

if (server == null)

throw new FormatException("Не правильный формат ip адреса или порта");

serverAddress = server;

}

catch (Exception exp)

{

MessageBox.Show(exp.Message);

return;

}

//Сохраняем настройки в файл настроек

global::ClientAlarm.Properties.Settings.Default.SERVER\_IP\_ADDRESS = serverAddress.Address.ToString();

global::ClientAlarm.Properties.Settings.Default.SERVER\_PORT = serverAddress.Port.ToString();

global::ClientAlarm.Properties.Settings.Default.Save();

MessageBox.Show("Настройки успешно сохранены! Программа будет перезапущенна!");

if (videoCamera.IsRunning)

videoCamera.Stop();

client.CloseClientCmd();

Dispose();

Application.Restart();

}

//Не сохронять настройки

private void button2\_Click(object sender, EventArgs e)

{

global::ClientAlarm.Properties.Settings.Default.Reload();

Close();

}

//Показать запись с камеры

private void videoToolStripMenuItem\_Click(object sender, EventArgs e)

{

using (ClientShowVideoForm csvf = new ClientShowVideoForm(this.videoCamera))

{

csvf.ShowDialog();

}

}

//Загрузка формы

private void ClientSettingsForm\_Load(object sender, EventArgs e)

{

videoDevices = new FilterInfoCollection(FilterCategory.VideoInputDevice);

if (videoDevices.Count == 0)

{

MessageBox.Show("Камера не найдена!");

isAppExit = true;

Application.Exit();

}

videoCamera = new VideoCaptureDevice(videoDevices[0].MonikerString);

detector = new MotionDetector(new SimpleBackgroundModelingDetector(),

new MotionAreaHighlighting());

if (this.videoCamera.IsRunning)

this.videoCamera.Stop();

videoCamera.NewFrame += delegate(object send, AForge.Video.NewFrameEventArgs eventArgs)

{

//Отлавливаем движение

if (detector.ProcessFrame(eventArgs.Frame) > 0.02)

{

if (client == null)

return;

client.SignalAlarmCmd();

}

};

ClientSettingsFormExtracted();

}

}

}

**Библиотека классов**

using System;

namespace SecurityAlarmLibrary

{

/// <summary>

/// Сообщения для обмена между сервером и клиентом

/// </summary>

public class SysMessage

{

/// <summary>

/// Соединение

/// </summary>

public const String CONNECT = "CONNECT";

/// <summary>

/// Закрытие соединения

/// </summary>

public const String CLOSE\_CONNECT = "CLOSE";

/// <summary>

/// Включить сигнализацию на удаленном хосте

/// </summary>

public const String VC\_SIGNALING = "VC\_SIGNALING";

/// <summary>

/// Выключить сигнализацию на удаленном хосте

/// </summary>

public const String EXIT\_SIGNALING = "EXIT\_SIGNALING";

/// <summary>

/// Сигнализация успешно включена на удаленном хосте

/// </summary>

public const String OK\_VC\_SIGNALING = "OK\_VC\_SIGNALING";

/// <summary>

/// Сигнализация не включена на удаленном хосте

/// </summary>

public const String NOT\_VC\_SIGNALING = "NOT\_VC\_SIGNALING";

/// <summary>

/// Сигнализация успешно выключена на удаленном хосте

/// </summary>

public const String OK\_EXIT\_SIGNALING = "OK\_EXIT\_SIGNALING";

/// <summary>

/// Сигнализация не выключена на удаленном хосте

/// </summary>

public const String NOT\_EXIT\_SIGNALING = "NOT\_EXIT\_SIGNALING";

/// <summary>

/// Выключение сервера

/// </summary>

public const String EXIT\_SERVER = "EXIT\_SERVER";

/// <summary>

/// Сигнал тревоги на сервер

/// </summary>

public const String SIGNAL\_ALARM = "SIGNAL\_ALARM";

/// <summary>

/// Отключить текущий сигнал тревоги и продолжить наблюдение

/// </summary>

public const String STOP\_SIGNAL\_ALARM = "STOP\_SIGNAL\_ALARM";

}

}

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Net;

namespace SecurityAlarmLibrary

{

/// <summary>

/// Утилиты

/// </summary>

public class Utils

{

/// <summary>

/// Шаблон ip адреса

/// </summary>

public const string PATTERN\_IP = @"([01]?\d\d?|2[0-4]\d|25[0-5])\." +

@"([01]?\d\d?|2[0-4]\d|25[0-5])\." +

@"([01]?\d\d?|2[0-4]\d|25[0-5])\." +

@"([01]?\d\d?|2[0-4]\d|25[0-5])";

/// <summary>

/// Шаблон порта

/// </summary>

public const string PATTERN\_PORT = "^(([0-9]{1,4})|([1-5][0-9]{4})|(6[0-4][0-9]{3})|(65[0-4][0-9]{2})|(655[0-2][0-9])|(6553[0-5]))$";

/// <summary>

/// Проверка на валидность ip адреса и порта

/// </summary>

/// <param name="ip">Адрес</param>

/// <param name="port">Порт</param>

/// <returns></returns>

public static IPEndPoint isValidAddress(string ip, string port)

{

IPAddress ipAddress = null;

int portAddress = -1;

if (!IPAddress.TryParse(ip, out ipAddress))

return null;

if (!int.TryParse(port, out portAddress))

return null;

return new IPEndPoint(ipAddress, portAddress);

}

/// <summary>

/// Конвертирование Bitmap в Массив байт

/// </summary>

/// <param name="img">Изображение</param>

/// <returns></returns>

public static byte[] ConvertBitmapToByteArray(Bitmap img)

{

List<byte> buffer = new List<byte>();

using (MemoryStream stream = new MemoryStream())

{

img.Save(stream, System.Drawing.Imaging.ImageFormat.Bmp);

buffer.AddRange(stream.ToArray());

}

return buffer.ToArray();

}

/// <summary>

/// Конвертирование Массива байт в Bitmap

/// </summary>

/// <param name="buffer">Массив байт</param>

/// <returns></returns>

public static Bitmap ConvertByteArrayToBitmap(byte[] buffer)

{

Image image = null;

using (MemoryStream stream = new MemoryStream(buffer))

{

image = Bitmap.FromStream(stream);

}

return new Bitmap(image);

}

/// <summary>

/// Можно ли запустить приложение

/// </summary>

/// <param name="type">Тип 0 - сервер, 1 - клиент</param>

/// <returns></returns>

public static bool CanLoadApplication(int type)

{

string procName = null;

switch (type)

{

//Для сервера

case 0:

procName = "ServerAlarm";

break;

//Для клиента

case 1:

procName = "ClientAlarm";

break;

}

if (procName != null)

{

Process[] prs = Process.GetProcesses();

for (int i = 0; i < prs.Length; i++)

if (prs[i].ProcessName == procName)

return false;

}

return true;

}

}

}