УДК 004

Создание клиент-серверного приложения на языке программирования C# с интеграцией библиотеки WCF

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен процесс создания сетевого приложения для обмена сообщениями между пользователями. Приложения написано на языке C# с использованием библиотеки WCF. Результатом исследования будет являться приложение сетевого чата и описание его работы.
Ключевые слова: C#, WCF, графическое приложение

Creating a client-server application in the C# programming language with WCF library integration

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article describes the process of creating a network application for messaging between users. The applications are written in C# using the WCF library. The result of the study will be a network chat application and a description of its operation.
Keywords: C#, WCF, graphical application

1 Введение

1.1 Актуальность

Современные вычислительные системы оперируют технологией "клиент-сервер", где обязанности и нагрузка распределены между теми, кто предоставляет услуги (сервером) и теми, кто их использует (клиентом). Эта концепция базируется на программном обеспечении, размещенном на вычислительных машинах, взаимодействие между которыми осуществляется через сеть с использованием различных протоколов. Без данной технологии невозможно представить современное общество, так как она проникла во все сферы жизни человека – начиная от покупок в интернете и заканчивая научными конференциями. Яркими примерами таких программ служат социальные сети, где люди могут обмениваться друг с другом информацией и медиафайлами. Для объяснений принципов работы клиент-серверного приложения в данной статье будет реализовано и описано приложение, в котором пользователи смогут вести друг с другом переписку, то есть чат.

1.2 Обзор исследований

А.А. Вьюгина и А.А. Крошилина в своём труде рассматривают задачу визуализации работы клиент-серверной архитектуры [1]. В.Д. Хандогин описал реализацию передачи данных между клиентами и серверами через протокол передачи данных TCP\IP на клиент - серверной архитектуре [2]. Е.М. Гриценко, Е.Д. Иванов и П.С. Шаталов в своей работе представили краткий обзор серверной платформы Node.js, современных back-end технологий на примере веб-приложения и технические характеристики [3]. Е.И. Костин и И.В. Чумак рассмотрели службу WCF для использования в клиент-серверном приложении [4]. Е.А. Панкратова и А.М. Пчелко в своей статье описали возможности технологий P2P и WCF для создания распределенных систем [5].

1.3 Цель исследования

Целью исследование является создание клиент-серверного приложения для обмена сообщениями между пользователями с помощью языка программирования C# и библиотек WCF и WPF.

2 Материалы и методы

Для создания приложения используется язык программирования C# и его библиотеки WCF и WPF. В качестве IDE используется Visual Studio.

3 Результаты и обсуждения

Реализуемое приложение чата состоит из двух частей – серверной и клиентской частей. Для создания графического интерфейса используется модуль WPF.

В процессе создания проекта выбирается библиотека классов .NET Framework. В созданный проект добавляется сервис WCF. Название класса указывается «ServiceChat». При создании создаётся дополнительный класс интерфейса «IServiceChat». Данный класс представляет из себя описание возможностей WCF сервиса в проекте. В классе ServiceChat выполняется реализация этого интерфейса (рис. 1).

IServiceChat представляет собой контракт службы, который содержит операции для подключения клиента к чату, отключения клиента и отправки сообщений. IServerChatCallback определяет интерфейс обратного вызова для клиентов, чтобы сервер мог отправлять им сообщения. Атрибуты [ServiceContract] и [OperationContract] используются для определения контрактов и операций службы, а IsOneWay указывает, что операции могут быть односторонними, без ожидания ответа.

С помощью интерфейса была описана работа сервиса WCF, то есть серверная часть приложения. Также необходимо прописать дополнительный функционал. Этот функционал прописывается в классе ServiceChat – в том первоначально созданном классе.

```
[ServiceContract(CallbackContract = typeof(IServerChatCallback))]
Ссылок: 1
public interface IServiceChat
{
    [OperationContract]
    Ссылок: 1
    int Connect(string name);

    [OperationContract]
    Ссылок: 1
    void Disconnect(int id);

    [OperationContract(IsOneWay = true)]
    Ссылок: 3
    void SendMsg(string msg, int id);

}

Ссылок: 2
public interface IServerChatCallback
{
    [OperationContract(IsOneWay = true)]
    Ссылок: 1
    void MsgCallback(string msg);
}
```

Рисунок 1. Методы класса IServiseChat

В данном классе реализуется вышеописанный интерфейс и его методы, которые обрабатывают подключение новых пользователей, их отключение и отправку сообщений между ними. (рис. 2).

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
Ссылок: 1
public class ServiceChat : IServiceChat
{
    List<ServerUser> users = new List<ServerUser>();
    int nextId = 1;

    Ссылок: 1
    public int Connect(string name)
    {

        ServerUser user = new ServerUser() {
            ID = nextId,
            Name = name,
            operationContext = OperationContext.Current
        };
        nextId++;

        SendMsg(": "+user.Name+" подключился к чату",0);
        users.Add(user);
        return user.ID;
    }

    Ссылок: 1
    public void Disconnect(int id)
    {
        var user = users.FirstOrDefault(i => i.ID == id);
        if (user!=null)
        {
            users.Remove(user);
            SendMsg(": "+user.Name + " покинул чат",0);
        }
    }

    Ссылок: 3
    public void SendMsg(string msg, int id)
    {
        foreach (var item in users)
        {
            string answer = DateTime.Now.ToShortTimeString();

            var user = users.FirstOrDefault(i => i.ID == id);
            if (user != null)
            {
                answer += ": " + user.Name+" ";
            }
            answer += msg;
            item.operationContext.GetCallbackChannel<IServerChatCallback>().MsgCallback(answer);
        }
    }
}
```

Рисунок 2. Методы класса ServiceChat

Класс ServiceChat имеет атрибут [ServiceBehavior], который устанавливает режим одиночного контекста экземпляра службы

(InstanceContextMode.Single), что означает, что будет создан только один экземпляр службы для всех клиентов. Дополнительно создаётся класс ServerUSers. Класс содержит три свойства: ID для идентификации пользователя, Name для хранения его имени и operationContext для управления контекстом операции, что позволяет взаимодействовать с клиентами через их контексты операции. Этот класс используется для представления каждого пользователя в чате с его уникальным идентификатором, именем и контекстом операции.

Внутри класса создается список пользователей users и переменная nextId для уникальной идентификации пользователей. Метод Connect добавляет нового пользователя в список и отправляет уведомление о подключении всем клиентам. Метод Disconnect удаляет пользователя из списка при отключении и отправляет уведомление о его выходе. Метод SendMsg отправляет сообщение от одного пользователя всем другим пользователям в чате.

Для каждого пользователя используется экземпляр ServerUser, который хранит его идентификатор, имя и контекст операции. Методы в этом классе взаимодействуют с клиентами через их контексты операции, используя обратные вызовы для отправки сообщений.

Таким образом, этот функционал обеспечивает базовую функциональность для чата, где клиенты могут подключаться, отправлять сообщения и отключаться от чата.

Сервис WCF реализован. Далее нужно реализовать функционал хоста. Хост (сервер) будет представлять из себя консольное приложение. Для этого в проекте создаётся ещё одно решение уже с консольным приложением - ChatHost. Данный хост будет работать в фоне и обрабатывать логику сервиса. Для корректной работы хоста, нужно в его проекте добавить ссылку на библиотеку System.Service.Model, которая отвечает за работу WCF. Помимо библиотеки необходимо добавить ссылку на dll файл проекта сервиса.

В проекте хоста находится файл конфигурации, который нужно отредактировать (рис. 3).

Данный XML файл определяет настройки связей для обмена данными между клиентом и сервером по протоколу NetTcpBinding. Здесь заданы параметры, такие как адрес сервера (address), используемая привязка (binding), имя привязки (bindingConfiguration), контракт службы (contract), идентичность клиента и другие параметры. Этот файл используется для настройки и установки соединения между клиентом и сервером.

Далее реализуется функционал главного класса проекта хоста (рис. 4). В методе Main создается экземпляр ServiceHost для службы ServiceChat из пространства имен wcf_chat. Затем хост запускается с помощью метода Open(), и в консоль выводится сообщение о запуске. Приложение остается активным, ожидая ввода пользователя в консоли (Console.ReadLine()), что позволяет хосту работать и обслуживать запросы клиентов до завершения работы приложения.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8"/>
    </startup>

  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="mexBeh">
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="wcf_chat.ServiceChat" behaviorConfiguration="mexBeh">
        <endpoint address="" binding="netTcpBinding" contract="wcf_chat.IServiceChat"/>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8301"/>
            <add baseAddress="net.tcp://localhost:8302"/>
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>

</configuration>
```

Рисунок 3. Конфигурационный файл хоста

```
class Program
{
    Ссылок 0
    static void Main(string[] args)
    {
        using (var host = new ServiceHost(typeof(wcf_chat.ServiceChat)))
        {
            host.Open();
            Console.WriteLine("Хост стартовал!");
            Console.ReadLine();
        }
    }
}
```

Рисунок 4. Программная реализация хоста

Хост реализован и работает (рис. 5).

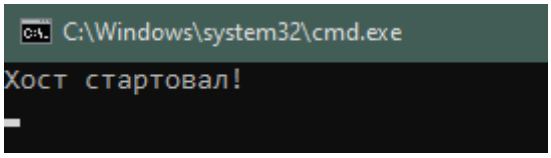

Рисунок 5. Проверка работоспособности хоста

Далее создаётся ещё одно решение в проекте уже на базе библиотеки WPF – ChatClient. Перед прописыванием функционала клиента, нужно добавить ссылки на сервис WCF и хоста.

После добавления ссылок следует добавление графических элементов в интерфейс: два TextBox, один из которых предназначен для ввода имени пользователя, а другой для ввода непосредственно сообщения; ListBox, в котором будут выводится текстовые сообщения, в том числе и сами сообщения пользователей; Button кнопка, при нажатии на которую будет происходить присоединение пользователя к чату (рис. 6).

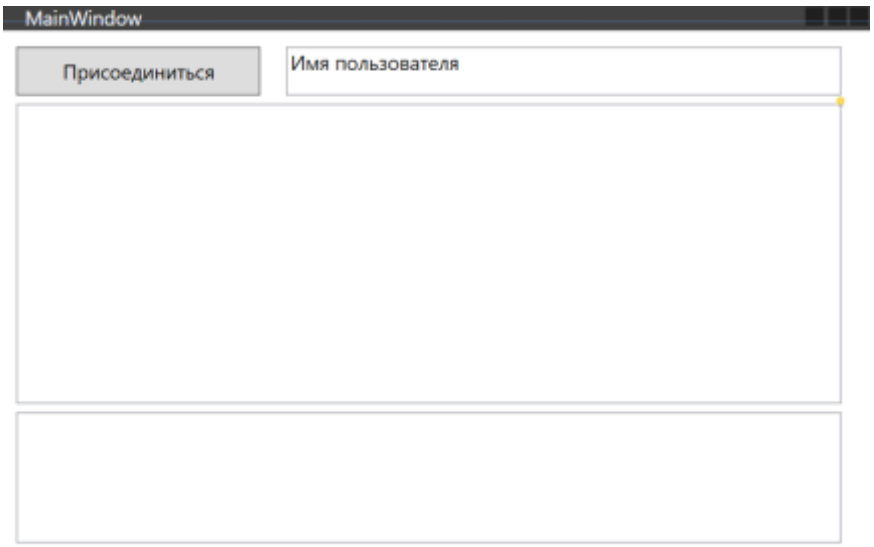

Рисунок 6. Внешний вид клиентской части приложения

Затем следует реализация функционала клиентской части (рис. 7 и 8).

```
public partial class MainWindow : Window, IServiceChatCallback
{
    bool isConnected = false;
    ServiceChatClient client;
    int ID;
    public MainWindow()
    {
        InitializeComponent();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {

    }

    void ConnectUser()
    {
        if (!isConnected)
        {
            client = new ServiceChatClient(new System.ServiceModel.InstanceContext(this));
            ID = client.Connect(tbUserName.Text);
            tbUserName.IsEnabled = false;
            bConnDicon.Content = "Отсоединиться";
            isConnected = true;
        }
    }

    void DisconnectUser()
    {
        if (isConnected)
        {
            client.Disconnect(ID);
            client = null;
            tbUserName.IsEnabled = true;
            bConnDicon.Content = "Присоединиться";
            isConnected = false;
        }

    }
```

Рисунок 7. Объявление класса MainWindow, переменных, конструктора и методов подключения и отключения пользователей от чата

Класс MainWindow реализует интерфейс IServiceChatCallback для обратного вызова от сервиса. В окне определены переменные для отслеживания состояния подключения, клиента чата и идентификатора пользователя. В конструкторе происходит инициализация компонентов.

Методы ConnectUser и DisconnectUser управляют подключением и отключением пользователя от чата. При подключении создается клиент чата (ServiceChatClient) с указанием текущего экземпляра MainWindow в качестве контекста обратного вызова. При отключении происходит закрытие соединения и возвращение элементов управления к исходному состоянию.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    if (isConnected)
    {
        DisconnectUser();
    }
    else
    {
        ConnectUser();
    }

}

Ссылок: 1
public void MsgCallback(string msg)
{
    lbChat.Items.Add(msg);
    lbChat.ScrollIntoView(lbChat.Items[lbChat.Items.Count-1]);
}

Ссылок: 1
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    DisconnectUser();
}

Ссылок: 1
private void tbMessage_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        if (client!=null)
        {
            client.SendMsg(tbMessage.Text, ID);
            tbMessage.Text = string.Empty;
        }
    }
}
```

Рисунок 8. Обработчики событий нажатия на кнопку и закрытия окна и методы обработки от сервиса

Обработчик события кнопки Button_Click переключает состояние подключения пользователя, вызывая соответствующие методы ConnectUser и DisconnectUser. Метод MsgCallback представляет собой реализацию обратного вызова от сервиса, добавляя полученное сообщение в список чата и прокручивая его вниз для отображения последнего сообщения. Обработчик события Window_Closing обеспечивает отключение пользователя при закрытии окна. Метод tbMessage_KeyDown реагирует на нажатие клавиши Enter при вводе текста сообщения, отправляя его через клиента чата.

Приложение готово к работе. Для этого осуществляется проверка на практике (рис. 9 и 10).

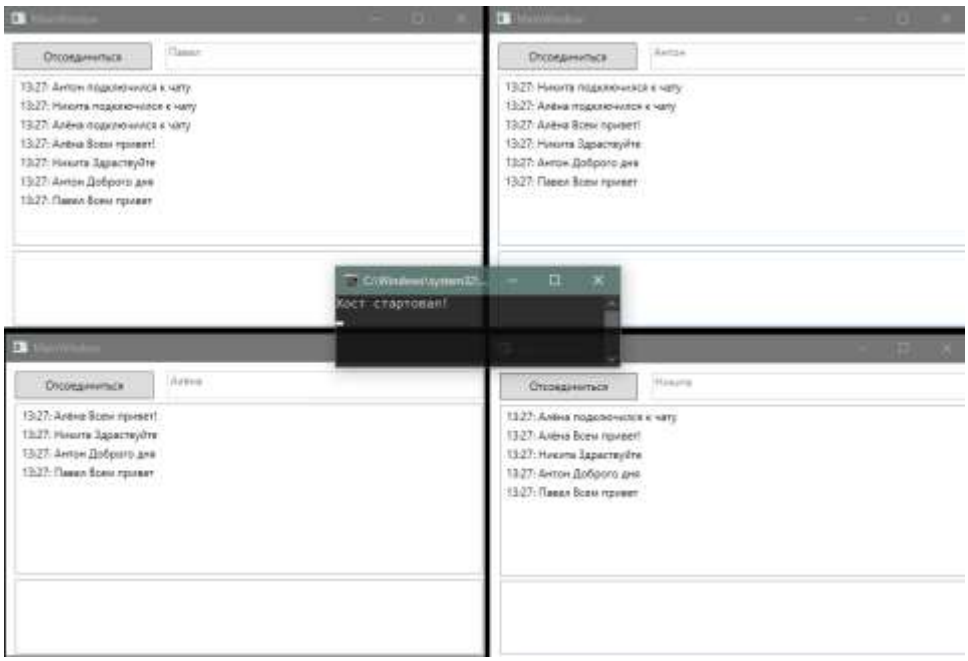

Рисунок 9. Запуск хоста, подключение клиентов и их переписка

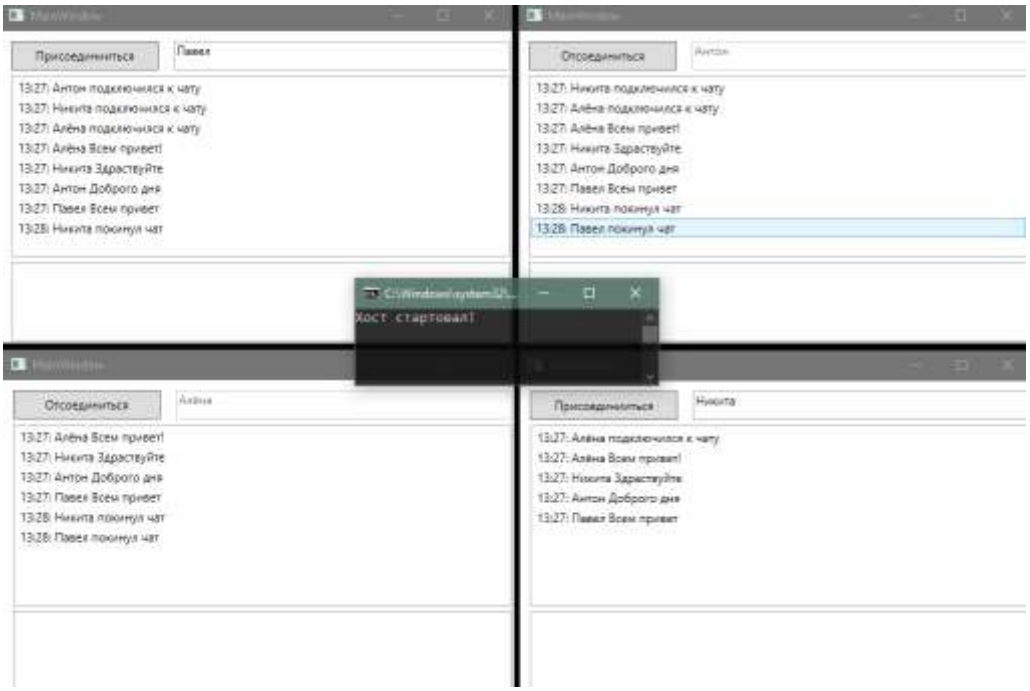

Рисунок 10. Пример работы программы при отключении от чата пользователя

Таким образом было реализовано клиент-серверное приложение на базе библиотек WCF и WPF. Функционал приложения был полностью описан и объяснён, также была показана работа приложения на практике.

Библиографический список

1. Вьюгина А.А., Крошилина А.А. Визуализация работы клиент-серверной архитектуры с использованием делегатов на языке C# // В сборнике: Методы и средства обработки и хранения информации. Межвузовский

сборник научных трудов. Под редакцией Б.В. Кострова. Рязань, 2022. С. 140-144.
2. Хандогин В.Д. Передача данных через протокол TCP IP // Системный администратор. 2022. № 1-2 (230-231). С. 128-131.
3. Гриценко Е.М., Иванов Е.Д., Шаталов П.С. Использование современных Back-end технологий разработки клиент-серверных приложений // В сборнике: Научные исследования: теория, методика и практика. Сборник материалов IV Международной научно-практической конференции. 2018. С. 213-214.
4. Костин Е. И., Чумак И. В. Создание службы WCF для использования в клиент-серверном приложении // Инновационные подходы в решении научных проблем. 2021. С. 121-126.
5. Панкратова Е. А., Пчелко А. М. Использование технологий P2P и WCF при разработке распределенных приложений // Информатика, математическое моделирование, экономика. 2013. С. 7-11.