Educational Research
Научно–методически статии

ПОДХОДИ ПРИ РАЗРАБОТВАНЕ НА ГРАФИЧЕН ИНТЕРФЕЙС ЗА РАБОТА С НЯКОЛКО ДОКУМЕНТА

Тодорка Терзиева
Пловдивски университет „Паисий Хилендарски“

Резюме. Известна е особената роля на задачите в обучението по информатика и информационни технологии – в частност, те самите могат да бъдат средство за обучение. В процеса на решаване на задачи обучаемите самостоятелно затвърдяват наученото, а също така откриват нови характеристики на изучаваните елементи чрез целенасочена активност от своя страна. В статията се представя проблемно ориентиран подход за обучение по учебната дисциплина „Създаване на графичен потребителски интерфейс C#“ на студенти от първи курс, специалност „Софтуерни технологии и дизайн“. Темата, която се разглежда, е свързана със стилове за дизайн на интерфейс и работа с няколко документа едновременно.

Keywords: graphical user interface, SDI and MDI applications

1. Въведение

Основен педагогически ефективен инструмент за организация на учебната дейност на студентите при изучаване на информатични дисциплини е създаването на проблемна ситуация. Това изисква нестандартно мислене, разбиране и систематизиране на познатите знания и умения. В същото време, разработваната задача, която се предоставя на студента на съответния етап от обучението, при липса на пълен обем от знания и понятия изисква пренебрегване на някои несъществени фактори на проблемната ситуация. Нивото на трудност на задачите трябва да съответства на нивото на подготовка на студентите. Следователно проблемната ситуация е необходимо да се дефинира по такъв начин, че да изисква базови знания от съответното ниво. При формулиране на условието на задачата трябва да има проблем, който не може да бъде решен чрез известните до този момент средства.

В статията се представя проблемно ориентиран подход за обучение по учебната дисциплина „Създаване на графичен потребителски интерфейс C#“ на студенти от първи курс, специалност „Софтуерни технологии и дизайн“. Темата, която

се разглежда, е свързана със стилове за дизайн на интерфейс и работа с няколко документа едновременно. Студентите имат базови знания по програмиране на C#. Те са изучавали методите и средствата на структурното програмиране и основи на обектноориентираното програмиране – базови алгоритмични конструкции, основни абстракции на типове данни и тяхното представяне, прилагане и анализ на основни алгоритми. Студентите имат малък практически опит от прилагане на езика C# и платформата .NET framework, като първи език за програмиране, изучаван в уводните курсове по програмиране. Целта на курса „Създаване на ГПИ (C#)“ е студентите да овладеят необходимите теоретични познания и да придобият практически умения за разработване на прозоречно базирани приложения с графичен потребителски интерфейс (ГПИ) на езика C#, като се използва интегрираната среда за разработка Microsoft Visual C#. Основно внимание се отделя на принципите за разработка на приложения с ГПИ и съществуващите за целта технологии. В курса се акцентира върху принципите на визуалното програмиране, обектите на ГПИ, обработка на изключения, работа с файлове и потоци, възможности за работа с графика в приложения с ГПИ, стилове за дизайн на ГПИ, работа с няколко документа едновременно (SDI и MDI приложения), връзка с бази от данни (Data Binding) и др.

2. Подходи при разработване на различни стилове за дизайн на ГПИ

Основните стилове за дизайн на потребителски интерфейс са следните:

– Интерфейс за единични документи – Single Document Interface (SDI);

– Интерфейс за множество документи – Multiple Document Interface (MDI)[1].

SDI е приложение, което се състои основно от една форма, съдържаща меню, например текстов редактор Notepad или Paint. Основните характеристики на SDI (фиг. 1) са следните:

– SDI е метод за организиране на приложения с ГПИ в отделни прозорци, които се управляват от операционната система отделно;

– всеки прозорец съдържа собствено меню или лента с инструменти;

– всички прозорци са независими един от друг.

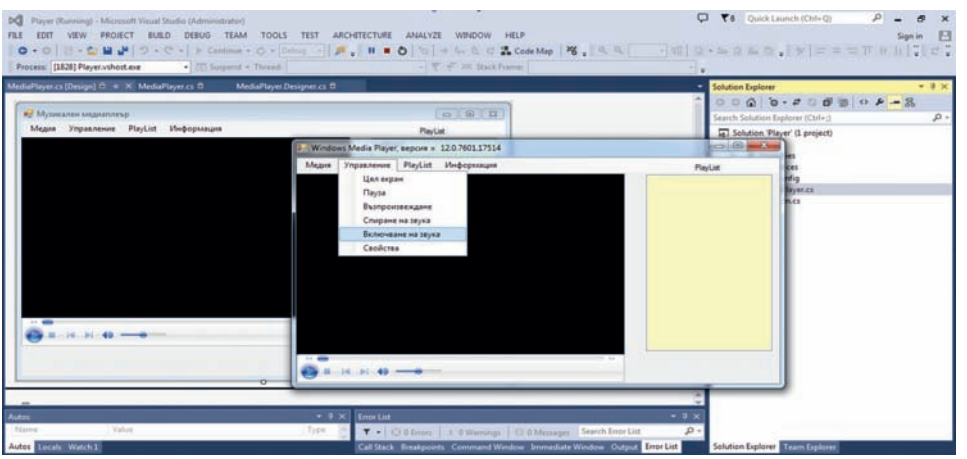
В някои случаи SDI може да има лента с инструменти и/или лента на състоянието. Въпреки че Notepad е текстово базиран, един SDI интерфейс може да бъде всеки вид приложение: текст, графики, таблици, Label, TextBox, ComboBox, ListView, TreeView, Button, MenuStrip, и др. Следователно, за да се създаде SDI, се започва от разработване на една нормална форма, добавя се меню към нея, което се конфигурира да изпълнява действията и командите, които искате. За изгражда-

не на SDI се използват класовете MenuStrip и Button GUI controls. За да се създаде друг документ от същия вид, потребителят трябва да отвори друг екземпляр на приложението.

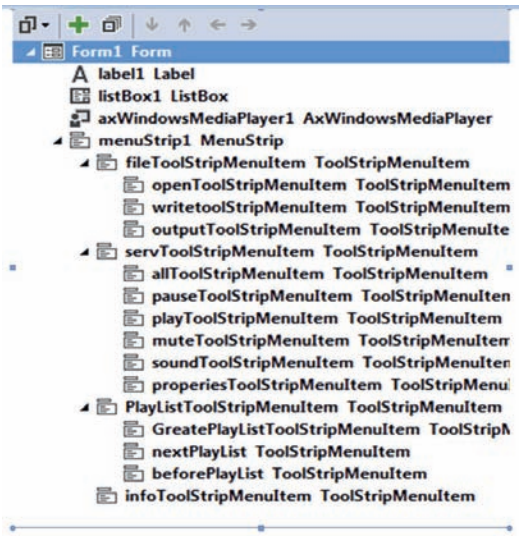
Ще разгледаме пример за създаване на интерфейс за единични документи, реализиращ мултимедиен плеър, съдържащ меню с основната функционалност на приложението. Като се използва съответната библиотека WMPLib[1], ще създадем приложение, което може да възпроизвежда различни мултимедийни файлове .AVI, .WAV, .MPEG, .WMA, .WMV и др.

1) Стартираме Visual Studio, създаваме нов проект с име Player. Кликваме на произволно място с десен бутон в Toolbox и избираме Choose Items à COM Components и поставяме отметка на Windows Media Player. Visual Studio.Net ще добави класа Windows Media Player.
2) Създаване на графичния интерфейс. Добавяме следните контроли: Windows Media Player с име axWindowsMediaPlayer1, за създаване на меню – menuStrip1, listBox1 – за създаване на PlayList.
3) Изграждаме йерархично меню за основната функцио-налност на приложението. На фиг. 1 са показани основните елементи и съответ-ните действия за всяко от тях на следващо ниво от йерархията (свойството DropDownItems).
4) Реализиране на функционалност.

```
// Програма, която реализира Windows Media Player 12
using System;
using System.Windows.Forms;
using System.ComponentModel;
```



Фигура 1. Проектиране на йерархично меню на WMP



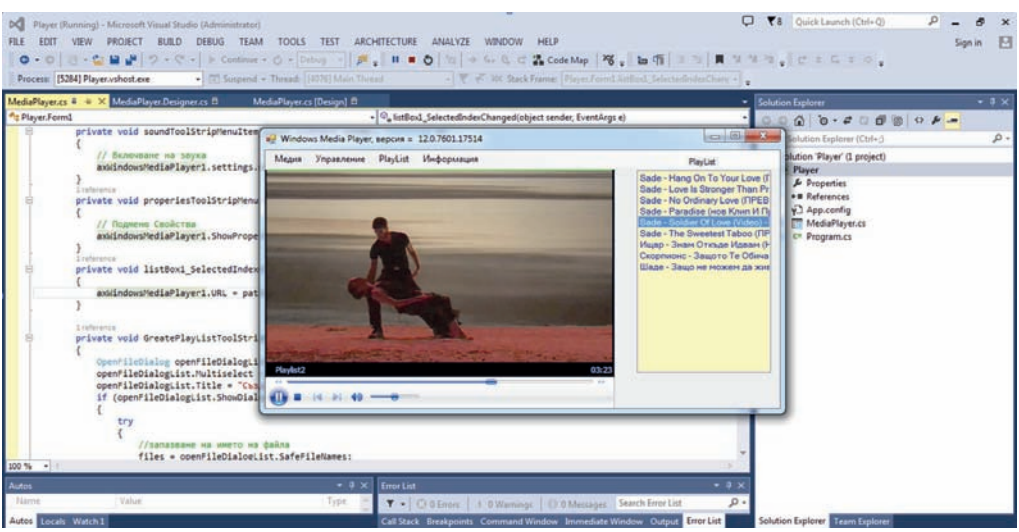

```
using WMPLib;
using System.IO;

namespace Player
{
  public partial class Form1 : Form
  {
     OpenFileDialog openFileDialogPlayer;
     public Form1()
     {     InitializeComponent();  }
     string[] files, paths;
private void Form1_Load(object sender, EventArgs e)
{
   openFileDialogPlayer = new OpenFileDialog();
   // Изписване на текущата версия на Player
   this.Text = „Windows Media Player, версия =  „ +
               axWindowsMediaPlayer1.versionInfo;
}
private void openToolStripMenuItem_Click(object sender, EventArgs
e)
{
```

```
      // Подменю за отваряне на файл
      openFileDialogPlayer.FileName = String.Empty;
      openFileDialogPlayer.InitialDirectory = „D:\\“;
      openFileDialogPlayer.Filter = „All Files(*.*)|*.*“;
      openFileDialogPlayer.FilterIndex = 2;
      openFileDialogPlayer.RestoreDirectory = true;
      if (openFileDialogPlayer.ShowDialog() == DialogResult.OK)
         try
         { // Името на файла се присвоява на плеера
           axWindowsMediaPlayer1.URL = openFileDialogPlayer.FileName;
           // Стартира се медийния файл
           axWindowsMediaPlayer1.Ctlcontrols.play();
         }
         catch (Exception ex)
          {  MessageBox.Show(„ Грешка! Не може да бъде прочетен такъв
файл!“ +        ex.Message,“Съобщение за грешка!“, MessageBoxButtons.
OK, MessageBoxIcon.Error);
         }
   }
   private void playToolStripMenuItem_Click(object sender, EventArgs
e)
   {
      // Подменю Play
      axWindowsMediaPlayer1.Ctlcontrols.play();
   }
         private  void  muteToolStripMenuItem_Click_1(object  sender,
EventArgs e)
      {
          axWindowsMediaPlayer1.settings.mute = true;
      }
     private void allToolStripMenuItem_Click_1(object sender, EventArgs
e)
      {   // Подменю за цял екран
          // Ако плеера е в състояние PLAY, може да се премине към ре-
жим - цял екран
           if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.
wmppsPlaying)
                   axWindowsMediaPlayer1.fullScreen = true;
      }
```

```
        private void outputToolStripMenuItem_Click(object sender,
EventArgs e)
     {
         // Изход от приложението
         Application.Exit();
     }
       private void pauseToolStripMenuItem_Click_1(object sender,
EventArgs e)
     {
         // Подменю Пауза
         axWindowsMediaPlayer1.Ctlcontrols.pause();
     }
       private void soundToolStripMenuItem_Click_1(object sender,
EventArgs e)
       {
           // Включване на звука
           axWindowsMediaPlayer1.settings.mute = false;
       }
        private void properiesToolStripMenuItem_Click(object sender,
EventArgs e)
       {
           // Подменю Свойства
           axWindowsMediaPlayer1.ShowPropertyPages();
       }
          private void listBox1_SelectedIndexChanged(object sender,
EventArgs e)
       {
           axWindowsMediaPlayer1.URL = paths[listBox1.SelectedIndex];
       }
          private void CreatePlayListToolStripMenuItem_Click(object
sender, EventArgs e)
       {
           OpenFileDialog openFileDialogList = new OpenFileDialog();
           openFileDialogList.Multiselect = true;
           openFileDialogList.Title = „Създаване на PlayList“;
           if (openFileDialogList.ShowDialog() == DialogResult.OK)
           {
  try
  {     //запазване на името на файла
```

```
                        files = openFileDialogList.SafeFileNames;
                        //запазване на целия път до файла
                        paths = openFileDialogList.FileNames;
                        for (int len = 0; len < files.Length; len++)
                        {  //Добавяне на песен към списъка
                            listBox1.Items.Add(files[len]);    }
                }
                catch (Exception ex)
                {
                  MessageBox.Show(„ Грешка! Не е избран медиен файл!“
+ ex.Message,
                                „Съобщение за грешка!“, MessageBoxButtons.
OK, MessageBoxIcon.Error);
                }
            }
        }
      private void infoToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            MessageBox.Show(„Това е музикален видео плеър.“);    }
        }
    }
```



Както видяхме, за да се създаде едно SDI приложение, се създава нова форма, добавя се меню и се създава подходящ интерфейс, който да позволи на потребителя да реализира функциите на приложението. Изграждането на едно MDI приложение изисква повече стъпки[2].

MDI приложенията поддържат работа с няколко документа едновременно, като всеки документ се показва в свой собствен прозорец, разположен във вътрешността на главния прозорец (Наков, 2007). MDI е вид ГПИ, който представлява един прозорец – родител (container), който е контейнер за други прозорци (child windows) в определено приложение. Само прозорецът-родител притежава меню или лента с инструменти. MDI позволява на дъщерните прозорци (child windows) да вграждат други прозорци вътре в тях, както и създаване на сложни вложени йерархии. MDI контейнери (MDI parents) са форми, които съдържат други форми. За да укажем, че една форма е MDI контейнер, задаваме нейното свойство *IsMdiContainer = true***.** Тези форми обикновено имат меню Window за смяна на активната форма (на свойство MdiWindowListItem се задава стойност windowToolStripMenu). MDI формите (MDI children) се съдържат в контейнер-формата. За да укажем, че една форма е MDI форма, задаваме на свойство *MdiParent = <контейнер>,* където контейнер е MDI форма, която е означена като контейнер.

Основните предимства на MDI са:

– Прозорците наследници (child windows) се управляват лесно от една родителска (container) форма.
– Едно меню и лента с инструменти може да бъде споделено с други прозорци.
– Възможността да се работи с множество документи от един прозорец на същото приложение.
– Чрез затварянето на родителския прозорец (container) потребителят затваря и другите дъщерни прозорци (child windows) (Skeet, 2013).

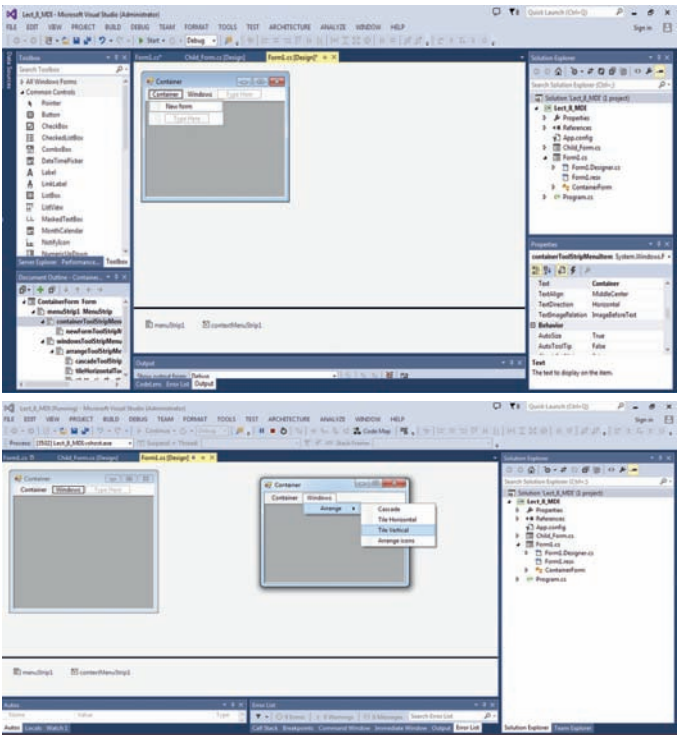
Ще разгледаме пример за създаване на стандартно MDI приложение, който обикновено се използва за въведение в разработване на интерфейс за работа с множество документи[2]**.**

1) Създаваме стандартно C# Windows Forms Application.

2) На първо място, трябва да се създаде форма *контейнер (container)*. Първата форма се превръща в контейнер само чрез промяна на свойството *IsMdiContainer = True.*

3) Добавяме нова форма към проекта. Тя ще бъде *дъщерна форма (child form)*.

4) След като е създадена новата форма, трябва тя да бъде извикана от родителската форма. Ще създадем меню чрез контрола MenuStrip**.** Добавяме към менюто основен елемент **Container** и един поделемент **New form** (фиг. 3).



5) За да отворим елемента **“New form”** от родителската форма, добавяме следния код към “New form” menu item:

```
private void newFormToolStripMenuItem_Click(object sender,
EventArgs e)
{
//Декларираме нова форма като Child_Form
        Child_Form childform = new Child_Form();
//Задаваме главната форма като родител container
        childform.MdiParent = this;
        //Показване на дъщерната форма
        childform.Show();
}
```

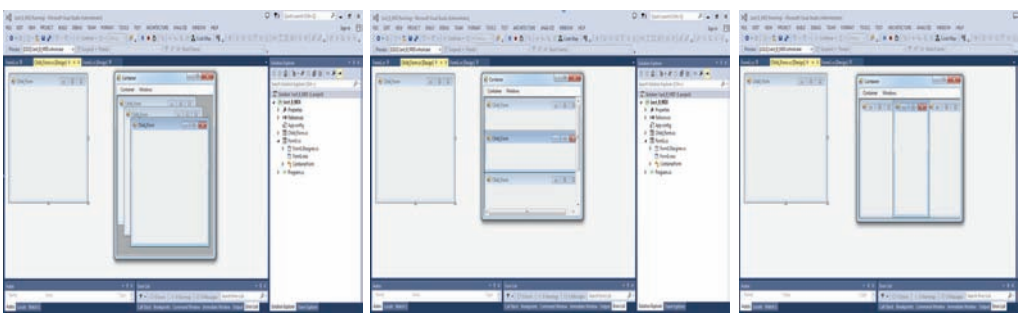
6) Ако компилираме приложението и кликнем няколко пъти върху елемента от менюто “New form”, ще получим няколко нови дъщерни форми вътре в основната форма.

Операционната система позволява на потребителя да избира между четири различни начина на подреждане. Например можете да позиционирате документите като вертикални колони, като хоризонтални редове, каскадно или като икони. За да реализира това, класът Form предоставя метод, наречен **LayoutMdi**. Синтаксисът му е следният:

```
public void LayoutMdi(MdiLayout value);
```

Методът LayoutMdi() изисква един аргумент, който е член на MdiLayout enumeration. Членове на това множество са *Cascade*, *TileHorizontal*, *TileVertical*, и *ArrangeIcons*.

7) Ще добавим към основното меню няколко допълнителни опции, които са необходими, за да визуализираме подреждането на дъщерните форми вътре в родителската форма (фиг. 4).



8) Добавяне на програмен код за различните начини на подреждане:

- Каскадно подреждане (Cascade) (фиг. 5)

```
private void cascadeToolStripMenuItem_Click(object sender, EventArgs
e)
        {    this.LayoutMdi(MdiLayout.Cascade);    }
```

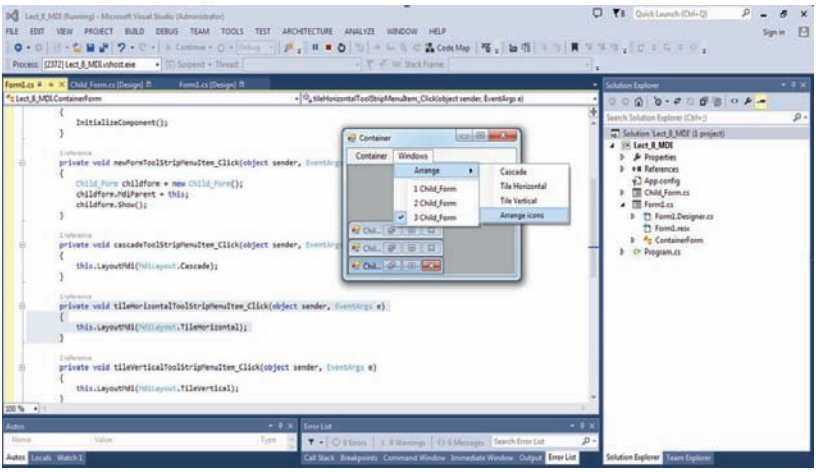
- Хоризонтално подреждане (Tile Horizontal) (фиг. 6)

```
private void tileHorizontalToolStripMenuItem_Click(object sender,
EventArgs e)
        {    this.LayoutMdi(MdiLayout.TileHorizontal);   }
```

- Вертикално подреждане (Tile Vertical) (фиг. 7)

```
private void tileVertikalToolStripMenuItem_Click(object sender,
EventArgs e)
    {    this.LayoutMdi(MdiLayout.TileVertical);   }
```

- Подредени икони (Arrange Icons) (фиг. 8)



```
private void arrangeIconsToolStripMenuItem_Click(object sender,
EventArgs e)
    {   this.LayoutMdi(MdiLayout.ArrangeIcons);  }
```

Оформлението на Arrange Icons е достъпно само за минимизирани дъщерни форми.

9) Ако потребителят отвори много форми деца, то става по-трудно да се придвижват между тях. За да се визуализират всички форми на свойството MdiWindowListItem на менюто (menu strip), задаваме windowToolStripMenuItem, така че всички отворени прозорци ще бъдат изброени в меню Window (фиг. 8).

Ще разгледаме пример за създаване на MDI интерфейс в Microsoft Visual Studio 2013.

Да се разработи приложение с ГПИ за регистриране и следене на успеха от следването на студент. Системата да включва:

– стартова форма и форма с данни за автора на системата;

– форма за въвеждане и корекция на личните данни на студентите и тяхната адресна регистрация;

– форма за регистриране на факултета, специалността, курса, групата и факултетния номер за всеки студент;

– форма за доходите на студент – стипендия (ако има), допълнителни доходи, такса за следване и извеждане на крайната сума.

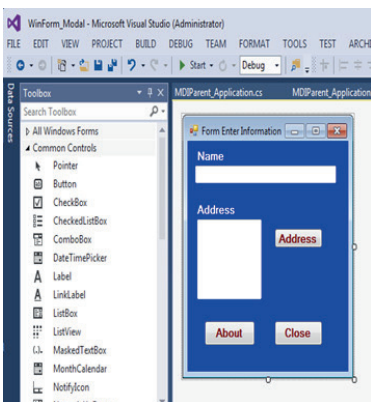
– отделните дейности да се оформят като избори в меню.

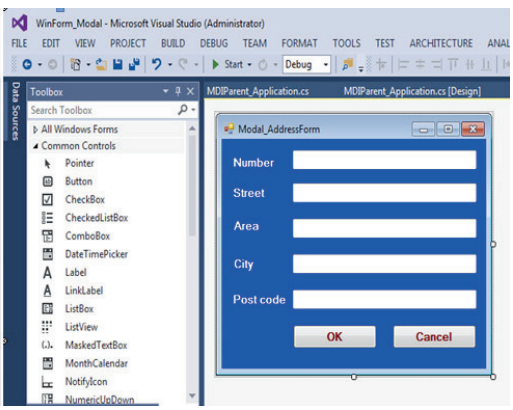
Приложението ще съдържа основно меню, което се намира във формата-контейнер.

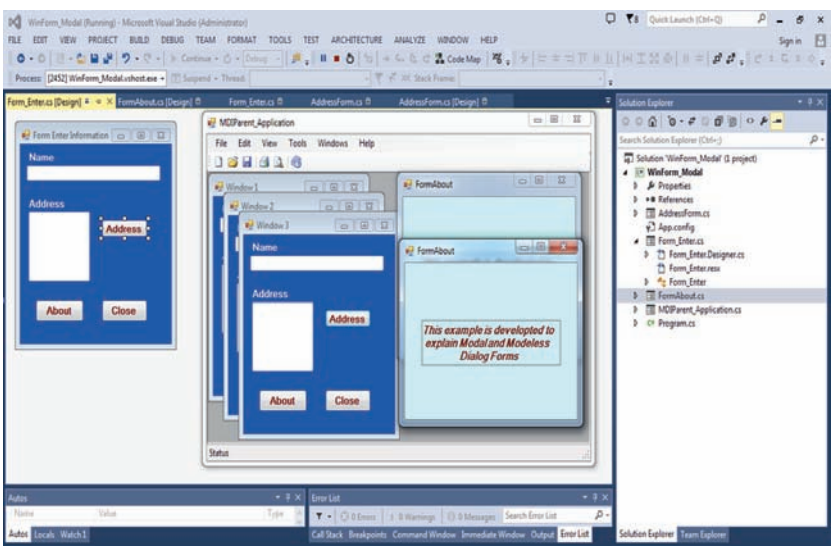
1) Създаваме нов проект New Project, Name = MDI_Application...

2) Добавяме нова форма – кликваме с десен бутон върху MDI_Application на Solution Explorer и избираме Add → Windows Form. Задаваме Name = Form_Enter (фиг. 9).

3) Добавяме нова модална форма – Name = AddressForm (фиг. 10), която ще се извиква при кликване върху бутон Address от първата форма за въвеждане на информация Form_Enter.







4) Добавяме програмен код към бутон Address от Form_Enter:

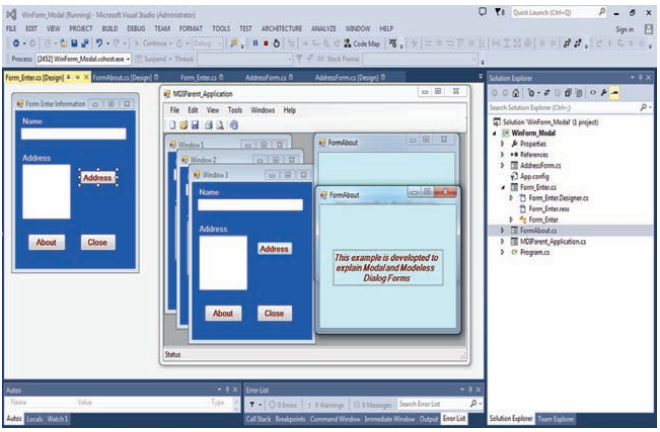
```
private void buttonAdr_Click(object sender, EventArgs e)
{    //създаване на модална форма
     AddressForm AdresForm = new AddressForm();
        if (AdresForm.ShowDialog()==DialogResult.OK)
               textBoxAdres.Text = AdresForm.Address;
        else
               textBoxAdres.Text="-----";
 }
```

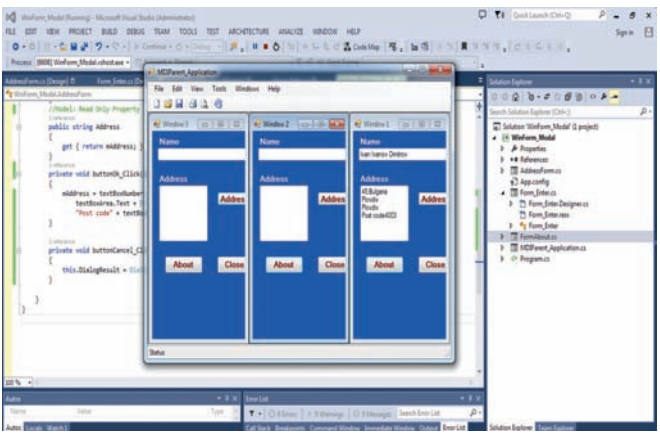
AddressForm е модална форма, защото при показването си прави неактивни всички останали форми на приложението и позволява достъпът до тях единствено след своето затваряне (фиг. 11). Модалността може да се задава първоначално, но не може да се променя, след като формата е вече показана.

5) За диалог между двете форми добавяме следния програмен код към AddressForm:

```
public partial class AddressForm : Form
{   //Модална форма за диалог между две форми
  private string mAddress;
  public string Address
  {
     get { return mAddress; }
  }
  private void buttonOk_Click(object sender, EventArgs e)
  {
     mAddress = textBoxNumber.Text + „;“ + textBoxStreet.Text +
Environment.NewLine +
      textBoxArea.Text + Environment.NewLine + textBoxCity.Text
+ Environment.NewLine +
            „Post code“ + textBoxPost.Text;
     this.DialogResult = DialogResult.OK;
   }
   private void buttonCancel_Click(object sender, EventArgs e)
   {
      this.DialogResult = DialogResult.Cancel;  }
```

6) Добавяме нова немодална форма – Name = FormAbout (фиг. 12), която ще се





извиква при кликване върху бутон About (за извеждане на информация) от първата форма за въвеждане на данни за студент Form_Enter. Немодалните форми се използват, когато е нужно няколко форми да са видими и достъпни едновременно на екрана.

7) По аналогичен начин създаваме другите две форми за въвеждане на служебна информация за студент и въвеждане на доходи.

8) За да добавим нова форма контейнер от главното меню PROJECT → Add New Item..., избираме от списъка *MDI Parent form*. Задаваме Name = MDIParentApplication и избираме Add. Свойството Text = MDIParent_Application. Автоматично се създава меню с всички базови действия и съответната функционалност.

9) Ще направим следните промени в кода:

```
public partial class MDIParent_Application : Form
{
   private int childFormNumber = 1;
   public MDIParent_Application()
   {
       InitializeComponent();
       ShowNewForm(null, null);
   }
   private void ShowNewForm(object sender, EventArgs e)
   {   //извикване на формата за въвеждане на име и адрес на студент
       Form_Enter childForm = new Form_Enter();
       childForm.MdiParent = this;
       childForm.Text = „Window „ + childFormNumber++;
       childForm.Show();
   }
```

От Solution Explorer кликваме два пъти върху Program.cs или с десен бутон избираме View Code, за да укажем приложението, което ще се стартира.

```
static void Main()
{
     Application.EnableVisualStyles();
     Application.SetCompatibleTextRenderingDefault(false);
     Application.Run(new MDIParentDocument());
}
```

След стартиране на приложението получаваме резултата, показан на фиг. 12. Средата автоматично създава меню във формата контейнер. То съдържа всички основни дейности, които се реализират от едно йерархичното меню, лента с инструменти и статус лента. Към автоматично генерирания програмен код са направени промените, показани по-горе. Лесно може да се направят корекции в менюто, за да се реализира исканата функционалност, така че да се адаптира това MDI приложение към разработвания проект.

Заключение

Една от целите на обучението по програмиране е студентите да усвоят теорията и да я прилагат, като решават практически проблеми. Тази цел може да бъде постигната чрез система от задачи (Анева, 2011; Гроздев & Гъров, 2008; Angelova & Rahnev, 2009). Известно е, че под система от задачи се разбира методически обоснована съвкупност от задачи, осигуряваща постигането на планирани от

обучението резултати (Grozdev, 2007). Всяка задача от една такава система носи определена информация, тясно свързана с изучаването на теоретичния материал, и има определено място и предназначение, като задачите се подреждат в нарастваща сложност. Групата задачи за прилагане на нови знания и умения трябва да бъде такава, че да се създаде представа за границите на приложимост на изучавания елемент, както и за типичните му приложения. Формирането на умения за прилагане на изучавания материал е на различни равнища – разпознаване и възпроизвеждане, съществено преобразуване на усвоеното, анализиране, оценяване, създаване. Целта на някои от задачите е да подпомогнат формирането на знания и умения на различни нива, други са предназначени за самостоятелна и колективна работа и създават условия за рационална обратна връзка.

БЕЛЕЖКИ

1. http://msdn.microsoft.com/en-us/library
2. http://www.csharpkey.com/visualcsharp/sdimdi/creation.htm

ЛИТЕРАТУРА

Анева, С., (2011). Реализиране на стандартни Windows приложения, съдържащи менюта чрез средата Visual C# при изучаване на събитийно програмиране в средното училище, *Сборник доклади на Национална научна конференция „Образованието в информационното общество*“, Пловдив, 311 – 320.

Гроздев, С. & Гъров, К. (2008). За системите от опорни задачи при подготовката за участие в олимпиади по информатика. Комбинаторни обекти и алгоритми. *Сборник доклади на 37 Пролетна конференция на СМБ*, Математика и математическо образование, Боровец, 304 – 311.

Гъров, К. (2010). За задачите в обучението по информатика и информационни технологии, *Сборник доклади на Национална конференция „Образованието в информационното общество“*, Пловдив, 27 – 28.05.2010, 95 – 101.

Наков, С. и колектив. (2007). *Програмиране за .NET Framework*. Том 2, В. Търново, Фабер.

Рахнев, А. (2010). *Интензификация на обучението по програмиране чрез използване на информационни технологии*, Хабилитационен труд за присъждане на научното звание „професор“, София.

Angelova, E. & Rahnev, A. (2009). Boosting Teaching and Learning Efficiency in Training Teachers of Information Technology, *Scientific Works, Plovdiv University*, vol. 36, book 3, Mathematics, 5 – 18.

Grozdev, S. (2007). *For High Achievements in Mathematics. The Bulgarian Experience (Theory and Practice).* Sofia: Association for the Development of Education.

Deitel, P. & Deitel, H. (2011). *C# 2010 for programmers*, 4-th ed., Pearson Education, Inc.

Sharp, J. (2010*). Microsoft Visual C# 2010 Step By Step*. Microsoft Press.

Skeet, J. (2013) *C# in Depth*, Third Edition, Manning Publications Co.

Troelsen, A. (2012). *Pro C# 5.0 and the .Net 4.5 Framework.* 6th Edition. Apress.

REFERENCES

Aneva, S., (2011). Realizirane na standartni Windows prilozheniya, sadarzhashti menyuta chrez sredata Visual C# pri izuchavane na sabitiyno programirane v srednoto uchilishte, *Sbornik dokladi na Natsionalna nauchna konferentsiya „Obrazovanieto v informatsionnoto obshtestvo"*, Plovdiv, 311 – 320.

Grozdev, S., Garov, K. (2008). Za sistemite ot oporni zadachi pri podgotovkata za uchastie v olimpiadi po informatika. Kombinatorni obekti i algoritmi. *Sbornik dokladi na 37 Proletna konferentsiya na SMB, Matematika i matematichesko obrazovanie*, Borovets, 304 – 311.

Garov, K. (2010). Za zadachite v obuchenieto po informatika i informatsionni tehnologii, *Sbornik dokladi na Natsionalna konferentsiya „Obrazovanieto v informatsionnoto obshtestvo*", Plovdiv, 27 – 28.05.2010, 95 – 101.

Nakov, S., i kolektiv. (2007). Programirane za .NET Framework. Tom 2, V. Tarnovo, Faber.

Rahnev, A. (2010). *Intenzifikatsiya na obuchenieto po programirane chrez izpolzvane na informatsionni tehnologii*, Habilitatsionen trud za prisazhdane na nauchnoto zvanie "profesor", Sofiya.

Angelova, E., Rahnev, A. (2009). Boosting Teaching and Learning Efficiency in Training Teachers of Information Technology, *Scientific Works*, Plovdiv University, vol. 36, book 3, Mathematics, 5 – 18.

Grozdev, S. (2007). *For High Achievements in Mathematics. The Bulgarian Experience (Theory and Practice)*. Sofia: Association for the Development of Education.

Deitel, P. & Deitel, H. (2011). *C# 2010 for programmers*, 4-th ed., Pearson Education, Inc.

Sharp, J. (2010). *Microsoft Visual C#* 2010 Step By Step. Microsoft Press.

Skeet, J. (2013) *C# in Depth*, Third Edition, Manning Publications Co.

Troelsen, A. (2012). *Pro C# 5.0 and the .Net 4.5 Framework.* 6th Edition. Apress.

APPROACHES IN DEVELOPING GUI AND WORKING WITH SEVERAL APPLICATIONS

Abstract. Tasks have a very specific role in the learning of informatics and information technologies. In particular, they can be educational tools within themselves. In the process of solving a problem, students consolidate their knowledge and also discover new characteristics of previously studied elements through activity on their part. The paper presents problem-oriented approach for teaching the course "Creating a GUI C #" for first-year students, specialty "Software technologies and design". This article emphasis on a methodological approach for teaching styles of interface design and working with several applications at the same time.

✉ **Dr. Todorka Terzieva, Assist. Prof.**
Faculty of Mathematics and Informatics
Plovdiv University "Paisii Hilendarski"
236, Bulgaria Blvd.
4003 Plovdiv, Bulgaria
E-mail: dora@uni-plovdiv.bg