

SOLUTIONS FOR DATA BINDING AND PROGRAMMATICALLY MANAGING RIBBON CONTROL OBJECTS BY DEVELOPING CUSTOM TASK PANES THAT INCORPORATE LANGUAGE-INTEGRATED QUERY (LINQ) INSTRUCTIONS

ALEXANDRU PÎRJAN^{1*}
DANA-MIHAELA PETROȘANU²

ABSTRACT:

Within this research, we have developed a series of solutions for data binding ribbon control objects and manipulating data sources by developing and integrating custom task panes that incorporate Language-Integrated Query (LINQ) instructions within the Office 2013 suite by using the development environment Visual Studio Tools for Office (VSTO) 2013. The solutions are designed to automate the activities carried out within office computer based information systems, by extending the default functionality of the Office 2013 suite, through the development of Component Object Model Add-Ins.

KEYWORDS: VSTO 2013, COM Add-Ins, LINQ, ribbon control, Custom Task Panes.

1. Introduction

The development environment VSTO 2013 facilitates the development of powerful Office solutions and makes it possible to approach various and complex technical aspects, such as mobile communications, data mining, cryptography [1], providing a complex set of development tools to the programmers [2].

In this paper, we have developed a complex solution for data binding to external data sources that consists in: creating using the development environment VSTO 2013 of a new tab in Word 2013, entitled *SURSE DE DATE*, within which we have incorporated a new group entitled *Utilizarea si Manipularea Surselor de Date*. This group contains an *editBox* ribbon control in which the user can insert the desired phone number, that will be validated before being effectively stored within the *editBox*; a *dropDown* ribbon control that will be binded to the data field *Nume* of a database created in *Oracle 12c Sql*Plus*, that contains the names of the persons and their associated phone numbers; the *dropDown* ribbon control will display only the names of the persons from the database, previously sorted in an ascending order.

After selecting a name from the *dropDown* ribbon control, the previously extracted data set from the database will be queried and we will use the LINQ language in order to sort the employees in an ascending order and the *editBox* ribbon control that displays the phone number will be updated with a new value, corresponding to the selected person

^{1*}Corresponding author. PhD. Lecturer ,Faculty of Computer Science for Business Management, Romanian-American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania, E-mail: alex@pirjan.com

² PhD. Lecturer ,Department of Mathematics-Informatics, University Politehnica of Bucharest, 313, Splaiul Independentei, district 6, code 060042, Bucharest, Romania, E-mail: danap@mathem.pub.ro

from the database. The newly created group will also contain a *checkBox* ribbon control whose selection will determine the instantiation of a *UserControl* form that will be encapsulated afterwards in the *CustomTaskPanels* collection associated with the document.

The task pane offers the possibility to visualize the content of the table from the database, to access the previous, the following, the first and the last records and also to delete the current record or a certain record corresponding to a person's code. Using the developed form one can edit and update the existing data within the data source, can add new records and can create queries based on the persons' names, having also the possibility to programmatically create a table within the Word document that will display the content of the database. Our developed solution allows that all the editing operations performed upon a person's data, to be reflected in the data source according to the contents of the new *DataSet* object belonging to the task pane and automatically updates the other *DataSet* object corresponding to the *Ribbon* bar in order to update the values of the *dropDown* list and the phone number, thus reflecting the new content of the database.

The advantages and features offered by *VSTO 2013* represent strong reasons that have determined us to approach, using this development environment, a high interest and actual theme regarding the data binding of ribbon control objects and their programmatically control, developing the solutions using the latest technologies available today.

2. Solutions for validating the contents of the *editBox* ribbon control object

In order to develop the solution, we have first created using *Microsoft Visual Studio Ultimate 2013* a new *Word 2013 Add-in* project, entitled *Surse de Date*. Within the project we have added a new *Ribbon (XML)* element entitled *Ribbon1* and we have overridden the *CreateRibbonExtensibilityObject()* function according to the procedure that we have previously depicted in [3]. We have created the elements corresponding to the extension of the existing *Ribbon* bar according to the techniques described in detail in our previous work [4]: a new tab element entitled *Surse de Date*, a group within this tab entitled *Utilizarea si Manipularea Surselor de Date*, an *editBox* control entitled *SMSNumar*, a *dropDown* list control entitled *ListaNume*, a *checkBox* control entitled *CasetaPanoul*.

In order to configure the default value of the *editBox* we have created within the *Ribbon1.cs* file the function *afiseazaNumarCurent* that we have previously declared using the XML attribute *getText* within the *Ribbon1.xml* file. This function takes as an argument an *IRibbonControl* object that represents the object that triggered the event, meaning the *editBox* object and returns a string object representing the value that will be displayed within the *editBox* when the module is loaded for the first time.

The *editBox* also allows the user to insert values using the keyboard. In order to validate these values, we have developed the *stocheazaNumar* function that is referred in the XML code using the attribute *onChange*. The developed function takes as arguments an object of *IRibbonControl* type and an object of type *String*. The function checks if the telephone number starts with the character "+" and that the remainder of the 11 characters are digits.

One important problem that we had to overcome was related to the situation when after the validation checks, we had to stop the execution, for example the situation when for the

phone number one would input letters instead of digits, because after the execution had been stopped, the *editBox* would have continued to display the input letters, even if the error message has been displayed to the user and the value has not been accepted.

In order to overcome this problem, we have developed and implemented an invalidation solution of the ribbon control object. When the *Component Object Model Add-In* is loaded, it automatically calls the previously described function *afiseazaNumarCurent*, but it does so only at the moment when the module is loaded. Afterwards, the state of the ribbon control object is stored in the cache area of the *Office* suite. By using the invalidation method corresponding to the *Ribbon* bar extension and to the unique identifier of the *editBox* object, we have requested the *Word* application to delete from the cache area the state of the ribbon control object having the identifier *SMSNumar* and to recall the *afiseazaNumarCurent* function as if the module had been loaded for the first time. Therefore, if the input value does not comply with the validation constraints, the *editBox* displays the default phone number. We have implemented the *stocheazaNumar* validation function using two approaches: a classical approach and a modern one.

The classical approach verifies first that the value input by the user exists (the string contains elements) and that it does not contain only white spaces. If the validation constraint is satisfied, the function continues to convert the string into an array of characters, it verifies whether the element from the first position is different from "+", case in which the user is asked to insert a valid value.

If the first element satisfies the validation condition, the array is traversed starting from the second position, element by element, comparing for each of the characters the UNICODE of the digits in order to verify that the input value consists only of numbers.

The modern approach consists in using *lambda* expressions in order to validate the phone number. We have checked first that the input value is not composed only of white spaces or is void. After this constraint is satisfied we have programmed the function to check if the first item is the "+" character, by directly calling the *ElementAt(index)* method that we have been able to call as the *String* class implements the *IEnumerable* interface.

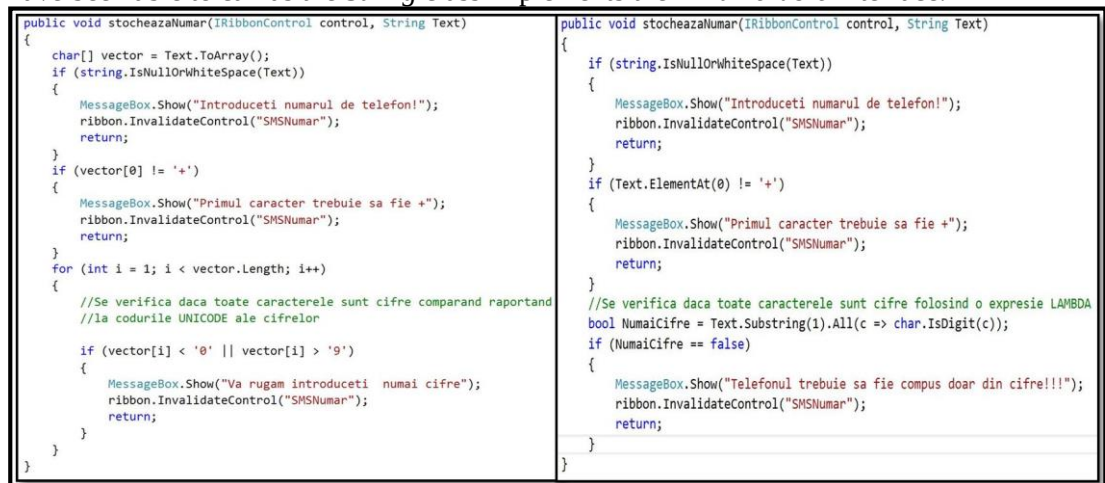


Figure 1. A comparison between the validation solutions through the classical approach (left) and the modern one (right)

In order to verify if starting from the second character, the input string consists only of numbers, we have used the *All* method corresponding to the *String* class, using directly as an argument of the method a *lambda* expression, in order to verify that each character is a digit, the returned result being stored in a *Boolean* variable. When comparing the two implementation methods, one can easily remark the readability and especially the flexibility of the validation solution that uses *lambda* expressions (**Figure 1**).

3. Solutions for data binding ribbon control objects

In the next step we have created the table entitled *AgendaTelefonica* using *Oracle 12c Sql*Plus* and we have populated it with six records (**Figure 2**).

```
CREATE TABLE AgendaTelefonica
(CodPersoana NUMBER(4) constraint AN_PK primary key,
 Nume VARCHAR2(70) constraint AN_NUME_NN not null,
 TelefAng VARCHAR2(30) default 'Telefon Necompletat');
```

Figure 2. The Oracle 12c Sql*Plus code for creating the *AgendaTelefonica* table

The table has the following structure: *CodPersoana* (primary key) of type *NUMBER* (4); *Nume* of type *VARCHAR2*(70) and a validation constraint that does not accept *null* values; *TelefAng* of type *VARCHAR2*(30). If the user does not input any phone number, the DBMS will automatically introduce the value "*Telefon Necompletat*".

In order to create the link with the data base within the *VSTO* project, we have added a *DataSet* element within the project and within this element we have created a *TableAdapter* object through which we have configured the connection parameters to the Oracle data base and we have selected to have available all the records from the table *AgendaTelefonica*.

From this moment on, within the project, the classes *DataSet1* and *AgendaTelefonicaTableAdapter* are available to be used for accessing and modifying the data source (**Figure 3**).

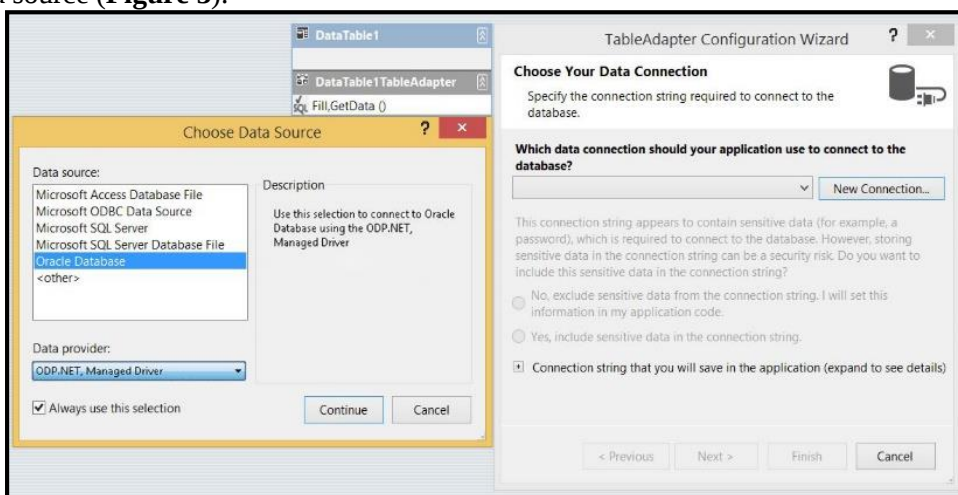


Figure 3. Specifying the connection to the data base by using an object of TableAdapter type

We have used the *DataSet1* class based on which we have created an instance entitled *SetulDeDate*, an instance that contains a table of *DataTable* type whose structure is identical to the one of the *AgendaTelefonica* table from the *Oracle 12c* database. In order to be able to modify the data source, we have created an object of *AgendaTelefonicaTableAdapter* type entitled *TabelaAgendaAdapter* and a temporary table of *DataTable* type that we will use in order to sort the contents of the database before displaying it in the *dropDown* list.

We have inserted in the *SetulDeDate* instance the records of the *AgendaTelefonica* table using the *Fill* method of the *TabelaAgendaAdapter* object. We have used the *LINQ* language in order to define the query that will sort the rows of the table from the data set alphabetically by the name of the person. This query is about to be run subsequently, being stored at the beginning in an implicit variable type entitled *sortQuery* (**Figure 4**).

```
TabelaAgendaAdapter = new DataSet1TableAdapters.AgendaTelefonicaTableAdapter();
TabelaAgendaAdapter.Fill(SetulDeDate.AgendaTelefonica);
//am folosit limbajul LINQ pentru a defini interogarea
var sortQuery = from inregistrare in SetulDeDate.AgendaTelefonica
                 orderby inregistrare.Nume
                 select inregistrare;
SetulDeDate.EnforceConstraints = false;
temporar = SetulDeDate.AgendaTelefonica.Copy();
temporar.Rows.Clear();
```

Figure 4. Defining and storing the query that is about to be run subsequently in order to sort alphabetically the data set by the person's name

In the moment when the *DataSet's* content will be linked with the *dropDown* list from the *Ribbon* bar, an association must exist between the values of the primary key of the *AgendaTelefonica* table that are being numbered starting with the digit zero and the index of the elements from the *dropDown* list that are also numbered starting from zero.

This is the reason why, in order to sort by the name of the person and for this sorting to be reflected also in the *dropDown* list of the *Ribbon* bar, the contents of the *DataSet* will have to be sorted using a temporary table, so that after the records have been sorted in an alphabetical order by the name, the values of the primary keys must be reconstructed starting with the value zero. Thus, the first record that is sorted alphabetically by name will have for the primary key field *CodPersoana*, the value zero, the next record will have a value of 1, etc. Otherwise, even if the data set is sorted alphabetically by name, in the drop-down list the values will be displayed in an unsorted order as the link with the data set within the project represents a correspondence between the primary key value of the record and the index value of the *dropDown* list.

Since there is no direct method to copy the *AgendaTelefonica* table's structure of the *SetulDeDate* data set, we have used the *Copy* method of the *AgendaTelefonica* object corresponding to the *SetulDeDate* instance in order to initially copy both the structure and the content of the table, in a temporary one. Afterwards, we have deleted the records from the temporary table using the *Clear* method of the *Rows* collection of the temporary table. In this moment, we have run the previously defined *LINQ* query and thus we have sorted

the records alphabetically by the name of the person, that we have reinserted in the initial table by reconstructing the primary key values starting with the value zero (Figure 5).

```
short nouacheie = 0;
/*momentul in care am rulat propriu-zis interogarea
 * si am reconstruit valoarea cheilor primare*/
foreach (var element in sortQuery)
{
    DataRow inregistrare = temporar.NewRow();
    inregistrare["CodPersoana"] = nouacheie;
    inregistrare["Nume"] = element.Nume;
    inregistrare["Telefon"] = element.Telefon;
    temporar.Rows.Add(inregistrare);
    nouacheie++;
}
```

Figure 5. Running the previously defined query stored in the implicit variable *sortQuery* and reconstructing the primary key values starting with the value zero

In this point we have deleted the records from the *AgendaTelefonica* table of the *SetulDeDate* data set but we have preserved its structure. We have updated using the *Update* method of the *TabelaAgendaAdapter* object the content of the *AgendaTelefonica* table from the *Oracle 12c* database with the current content of the project's data set table. In this moment, the content of the *AgendaTelefonica* table from the *Oracle 12c* database is deleted but its structure is preserved.

We have reinserted the records from the sorted temporary table in the initial *AgendaTelefonica* table from the data set *SetulDeDate* of the project and after having finalized the operation we have applied the *Update* method of the *TabelaAgendaAdapter* object in order to propagate the changes to the *Oracle 12c* database. In this moment, the *AgendaTelefonica* table from the *Oracle 12c* database contains the records sorted by name in an alphabetical order with the reconstructed values of the primary keys starting with the value zero for the first record (Figure 6).

In this moment we have built the *getCount* function within the *Ribbon1.cs* file, the function having been declared using the XML attribute *getItemCount* within the *Ribbon1.xml* file. The function takes as a parameter an *IRibbonControl* object and returns an integer representing the number of items of the *dropDown* list. Using the *Count* property of the *Rows* collection of the *AgendaTelefonica* data table of the data set, *SetulDeDate*, we have assured that the *dropDown* list contains as many items as there are in the database.

```
//Se sterge continutul tabelii AgendaTelefonica din cadrul
//setului de date SetulDeDate, nu si structura acesteia
foreach (DataSet1.AgendaTelefonicaRow vechiul in SetulDeDate.AgendaTelefonica)
{
    vechiul.Delete();
}
//Am actualizat tabela AgendaTelefonica din baza de date Oracle cu actualul
// continut al tabelii AgendaTelefonica din cadrul setului de date al proiectului
//Practic in acest punct continutul tabelii din baza de date Oracle este sters
TabelaAgendaAdapter.Update(SetulDeDate.AgendaTelefonica);
//Se reintroduc inregistrările din tabela sortată în tabela inițială AgendaTelefonica
//din cadrul setului de date SetulDeDate al proiectului
foreach (DataRow rs in temporar.Rows)
{
    DataSet1.AgendaTelefonicaRow inregReintrodusa =
        SetulDeDate.AgendaTelefonica.NewAgendaTelefonicaRow();
    inregReintrodusa.CodPersoana = Convert.ToInt16(rs["CodPersoana"]);
    inregReintrodusa.Nume =
        rs["Nume"].ToString();
    inregReintrodusa.Telefon = rs["Telefon"].ToString();
    SetulDeDate.AgendaTelefonica.AddAgendaTelefonicaRow(inregReintrodusa);
}
//Se actualizează continutul tabelii AgendaTelefonica din cadrul bazei de date Oracle
//astfel încât în acest moment baza de date conține înregistrările sortate și
//cu cheile primare reconstruite începând cu valoarea zero
TabelaAgendaAdapter.Update(SetulDeDate.AgendaTelefonica);
```

Figure 6. Reinserting the sorted records in the *Oracle 12c* database with the reconstructed values of the primary keys

In order to determine the names of the items from the *dropDown* list we have developed the following solution: we have created the *getLabel* function within the *Ribbon1.cs* file, function that we had previously specified using the XML attribute *getItemLabel*. The function takes two arguments: an *IRibbonControl* object that represents the *dropDown* list, an integer variable representing the index of the item and returns a string that represents the name of the item corresponding to the index. We have used the *FindByCodPersoana* method, of the *AgendaTelefonica* data table of the data set *SetulDeDate* from the project in order to search in the table of the *Oracle 12c* database for the record that has the value of the primary key *CodPersoana* equal with the value of the list's index. Once the record has been identified, the function returns the name of the person that will be displayed as an element within the *dropDown* list.

In order to update the value of the edit box *editBox* with the phone number of a specific person whose name has been selected from the dropdown list, we have developed and implemented the following solution: we have created the *SelectareNum* function within the *Ribbon1.cs* file, a function that we have previously declared using the *onAction* XML attribute. The function takes three arguments: an *IRibbonControl* object, a string type variable representing the name of the element from the dropdown list and an integer variable representing the index of the selected item from the dropdown list. Using the *FindByCodPersoana* method corresponding to the *AgendaTelefonica DataTable* object we have identified in the *Oracle 12c* database the record that has the respective primary key, we have updated the content of the *numar* variable that stores the default phone number that is being returned by the *afiseazaNumarCurent* function when the module is loaded. In order to delete the content of the *editBox* from the *Office* application cache we have used the invalidation method corresponding to the *Ribbon* bar extension (**Figure 7**).

In this moment, we have extended the *Ribbon* bar of the *Word* application through a Component Object Model Add-In with a new tab menu entitled *SURSE DE DATE* that contains a group labeled *Utilizarea si Manipularea Surselor de Date*. Within this group we have developed an *editBox* ribbon control that displays a default phone number and also allows the user to edit it directly from the keyboard (the input number must comply with the validation rules, the first character must be "+", followed by 11 digits), a *dropDown* ribbon control that contains as many items as the number of records of the *AgendaTelefonica* data table of the *Oracle 12c* database and a *checkBox* ribbon control labeled *Acceseaza Baza de Date*. As soon as the user has selected a name from the *dropDown* list ribbon control, the respective person is searched in the table of the *Oracle 12c* database.

After the person has been identified, the corresponding phone number is returned and the *editBox* ribbon control is automatically updated in real time (**Figure 8**).

```
//Funcția returnează numărul de elemente ale listei derulante din cadrul extensiei Ribbon
0 references
public int getCount(IRibbonControl control)
{
    return SetulDeDate.AgendaTelefonica.Rows.Count;
}

//Funcția returnează denumirea elementelor listei derulante din cadrul extensiei Ribbon
0 references
public string getLabel(IRibbonControl control, int index)
{
    DataSet1.AgendaTelefonicaRow inregistrarea;
    inregistrarea = SetulDeDate.AgendaTelefonica.FindByCodPersoana((short)index);
    return inregistrarea.Nume;
}

//Funcția actualizează conținutul casetei de editare cu numărul de telefon al persoanei
// selectate din lista derulanta. Am folosit soluția de invalidare pentru a solicita
// platformei Office să steargă din zona de memorie tampon valoarea stocată și să reapele
// funcția afișeazăNumarCurent creată anterior
0 references
public void SelectareNume(IRibbonControl control, string selectedId, int selectedIndex)
{
    DataSet1.AgendaTelefonicaRow inregistrarea;
    inregistrarea = SetulDeDate.AgendaTelefonica.FindByCodPersoana((short)selectedIndex);
    numar = inregistrarea.Telefon;
    ribbon.InvalidateControl("SMSNumar");
}
}
```

Figure 7. Dynamically constructing the content of the dropdown list according to the database and updating in real-time the content of the *editBox* ribbon control according to the person's name from the dropdown list

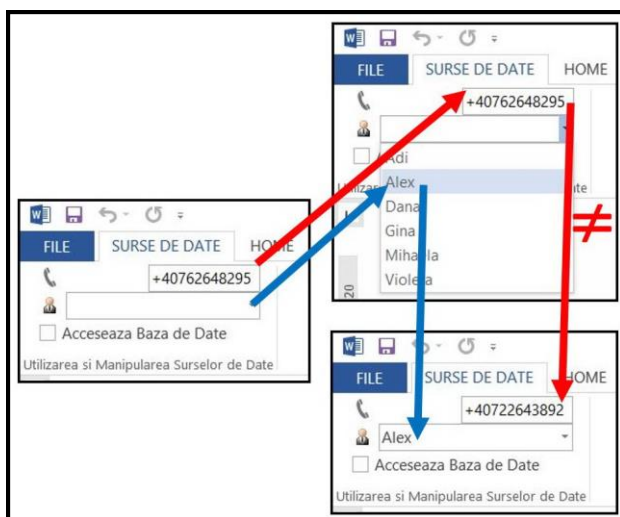


Figure 8. An example of how the developed solution works in order to dynamically build the content of the *dropDown* ribbon control and retrieve the phone number from the table of the *Oracle 12c* database directly in the Ribbon bar extension

In the next section, we have developed and implemented the associated action of the *checkBox* ribbon control that has the identifier *CasetaPanoul*.

4. Solutions for developing custom task panes within the documents

After selecting the *checkBox* ribbon control, a custom task pane that incorporates an *UserControl* object will be created. The custom task pane will be added to the *Office*

document's *CustomTaskPanels* collection and offers the possibility to access the contents of the *AgendaTelefonica* table from the *Oracle 12c* database (**Figure 9**).

Figure 9. Developing a *UserControl* object that is about to be incorporated in a custom task pane that will be added in the *CustomTaskPanels* collection of the *Office* application

We have first added within the document a new *UserControl* object within which we have developed and implemented the following elements:

- An object of type *label* entitled *label1*, that displays the text *Cod* followed by a *TextBox* object entitled *CodPersoana*, having a font size of 13.8 pt;
- An object of type *label* entitled *label2*, that displays the text *Nume* followed by a *TextBox* object entitled *NumePersoana*, having a font size of 13.8 pt;
- A third object of type *label* entitled *label3*, that displays the text *Telefon* followed by a *TextBox* object entitled *TelefonPersoana*, having a font size of 13.8 pt;
- A *RichTextBox* object entitled *InregCurentaDinTot* through which we are displaying the index of the current record from the database that is being displayed with the blue color within the task pane and the total number of records of the *Oracle 12c* database table with the red color. The font has a size of 12 pt and we have set the *BorderStyle* property to the value *None* in order to suppress the displaying of the border around the object;
- An object of type *label* entitled *label4*, that displays the text *Introduceți codul / numele persoanei*, having a font size of 10.2 pt;
- A *TextBox* object entitled *Cautare*, designed to facilitate the input of the person's code, name or phone number in order to be used in the subsequent querying and filtering operations;

- 10 objects of type *button* within which we have developed and implemented the following actions: accessing the previous, next, first and last records; deleting the current record; deleting in accordance with the person's code that has been input in the *TextBox* object entitled *Cautare*; updating the editing operations that have been performed within the form in the *AgendaTelefonica* data table from the *Oracle 12c* database; adding a new record; searching for the person's name that has been input in the *TextBox* object entitled *Cautare*; generating programmatically a table that contains the values of the *AgendaTelefonica* data table from the *Oracle 12c* database, directly within the Word document;
- A check box entitled *checkBox1* followed by a *TextBox* object entitled *Activează Opțiunile de Filtrare*, having a font size of 9 pt. If the check box was selected, two objects of type *button* that initially weren't visible (had the *Visible* property set to *FALSE*), become available. These two buttons offer the possibility to: filter the records within the database whose name contains the text input in the *TextBox* object entitled *Cautare*; filter the records within the database whose name starts with the text input in the *TextBox* object entitled *Cautare*;
- A *groupBox1* element whose border is not visible. This element groups the previously created *checkBox1* and the two filtering buttons.

In order to incorporate the *UserControl1* object in a custom task pane that will become available in the active window of the *Office* application, we have developed and implemented the following solution: we have accessed the *ThisAddIn.cs* main file of the module within which we have created, based on the previously developed class *UserControl1*, an instance entitled *controlul*. The class contains the form that has been created according to the previously mentioned specifications.

We have developed a function entitled *adaugaPanouPersonalizat* that instantiates the custom task pane and adds it in the *CustomTaskPanes* collection of the *Office* application. We have identified using the *ActiveWindow* property of the current application, the current window of the *Word* application that we will be using when creating the custom task pane. We have declared an object of *CustomTaskPane* type entitled *panoulMeu* and we have used the *Add* method corresponding to the *CustomTaskPanes* object in order to add the newly created task pane in the custom task panes collection of the *Office* application, specifying as parameters: the *UserControl* object that will be incorporated in the custom task pane, the title of the custom task pane *Baza de Date* and the application's window in which the custom task pane will be incorporated. We have also specified the docking position by setting the *DockPosition* property with the value *msoCTPDockPositionRight* in order to dock the custom task pane on the right side of the application's window (**Figure 10**).

```
controlul = new UserControl1();
Microsoft.Office.Tools.CustomTaskPane panoulMeu;
panoulMeu = CustomTaskPanes.Add(controlul, "Baza de Date", fereastra);
panoulMeu.Width = 500;
panoulMeu.Visible = true;
panoulMeu.DockPosition = MsoCTPDockPosition.msoCTPDockPositionRight;
panoulMeu.VisibleChanged += panoulMeu_VisibleChanged;
```

Figure 10. The solution for incorporating the *UserControl1* object within a custom task pane and adding it to the *CustomTaskPanes* collection of the *Office* application

In order to data bind the *TextBox* objects (*CodPersoana*, *NumePersoana*, *TelefonPersoana*) to the contents of the *Oracle 12c* database table records, we have selected one by one the three *TextBox* objects of the form, we have accessed sequentially the objects' properties and within the section *DataBindings* we have selected the data source from within the *DataSet* project. The *VSTO* development environment has automatically generated a new instance of the dataset that we have renamed it *SetulPanoului*, a new object of *TableAdapter* type (entitled *agendaTelefonicaTableAdapter*) in order to manage the link between this new dataset and the *AgendaTelefonica* data table from the *Oracle 12c* database and a *BindingSource* object entitled *agendaTelefonicaBindingSource* that simplifies a lot the process of managing the link between the data source and the form's content. Therefore, in this moment, there are two instances available into the project: an instance entitled *SetulDeDate*, that is available at the level of the *Ribbon* extension and an instance entitled *SetulPanoului*, that is available at the level of the newly developed custom task pane.

5. Conclusions

The *VSTO 2013* development environment offers a wide range of ribbon control objects and efficient ways to programmatically control them, facilitating the development of complex solutions for the office computer based information systems. *VSTO 2013* offers multiple advantages arising from the integration of the *.NET* development framework, offering the possibility to develop robust projects by using *Visual Studio* to program the software solutions. One of the most important benefits that *VSTO 2013* brings, consists in the possibility of extending the default features of the *Office* suite, depending on the users' specific needs and requirements. Analyzing the solutions that we have developed and implemented for data binding ribbon control objects and for programmatically controlling them, by developing custom task panes, we can conclude that *VSTO 2013* has brought significant improvements to the previous versions, representing a powerful and useful tool in the development of custom tailored *Office* solutions.

6. Acknowledgement

This paper presents a series of results obtained within the project "Practical School: Innovation in Higher Education and Success on the Labour Market", POSDRU/156/1.2/G/132920, implemented by the Romanian-American University in partnership with the Center for Legal Resources. The project is co-financed from the European Social Fund through the Sectorial Operational Programme Human Resource Development 2007-2013.

References

- [1] Tăbușcă A., A new security solution implemented by the use of the multilayered structural data sectors switching algorithm (MSDSSA), *Journal of Information*

Systems & Operations Management, vol.4, no.2, Ed. Universitară, 2010, ISSN 1843-4711

- [2] Carter E., Lippert E., Visual Studio Tools for Office 2007: VSTO for Excel, Word, and Outlook, Publisher: Addison-Wesley Professional, 2009, ISBN-10: 0321533216, ISBN-13: 978-0321533210
- [3] Pîrjan A., Solutions for repurposing the default actions and states of the Office controls through Component Object Model Add-Ins, Journal of Information Systems & Operations Management, Vol. 9, Nr. 1/2015, pp. 136-146, ISSN 1843-4711
- [4] Pîrjan A., Petroșanu D.M., Solutions for developing and extending rich graphical user interfaces for Office applications, Journal of Information Systems & Operations Management, Vol. 9, Nr. 1/2015, pp. 157-167, ISSN 1843-4711