Some Considerations on Using Data Structures Combined with TreeView Control

Professor, Ph.D. Mihaela MUNTEAN, lecturer, Ph.D. Mirela VOICU
West University of Timisoara, ROMANIA

Lists and binary trees structures represents a higher programming technique level. In RAD environments TreeView control is a powerful tool to see data in a hierarchical fashion or to observe the data structures functionality. We manage to transform the TreeView component in a strong data structures mastering control and suggest its using in application development. We now present, choosing Delphi for implementation, some techniques to illustrate the presented above.

1 Introduction

A data structure is a construct that you can define within a programming language to store a collection of data for easy access and frequent manipulation. It must provide the structural order for organizing data, while maintain access flexibility to allow fast and easy movement from one data item to another.

A well-organized data structure can speed up the computational process – the algorithm specified by the program – by a significant amount of running time. Linked lists and tree data structures are flexible dynamic data structures representing a powerful tool for organizing data objects based on keys.

On the other hand, the TreeView control manages to display a set of objects as an indented outline based on their logical hierarchical relationship. In Delphi, the TreeView component is an instance of the TTreeView class and each node is an instance of the TTreeNode class. A TreeView's list of TTreeNode objects is maintained by its Items property. This property is itself an instance of the TTreeNodes class.

We succeed to transform the TreeView control from a simple outlining tool in a strong *data structure mastering control.* This interface component will be used in manipulating a set of dynamic linked lists, the paper suggesting some proper programming techniques. In addition, the TreeView control will take over, all the modifications made on a binary tree structure.

2. Operating with list data structures through a TreeView Control

TreeViews are powerful tools if you need to see data in a hierarchical fashion. Objects classified in categories, groups, subgroups and assortments can be displayed within a TreeView control by extracting the necessary information from a specific text file – Figure 1.

All object assortments are stored in a table data structure; each table element contains an object code and two pointers to the first and last element of the corresponding object operations dynamic list.

The entire outline is load and saved in a form of tabbed text to disk using the TreeView's LoadFromFile( ) and SaveToFile( ) methods. Each tab represents one level of indent in the TreeView's hierarchy. The node's contextual menu establishes the potential operations at each TreeView's level and initiates the corresponding processing sequence on the defined data structures (1).

```
Type
    TOperation = record
                 ...
                 end;
    TPointNode = ^Node;
    TNode = record
              Op : TOperation;
              Next : TPointNode;
             end;                          ( 1 )
    TObj = record
            Code : String[30];
            First, Last : TPointNode;
           end;
    Public
    Objects : array [1..Nrmax] of TObj;
```

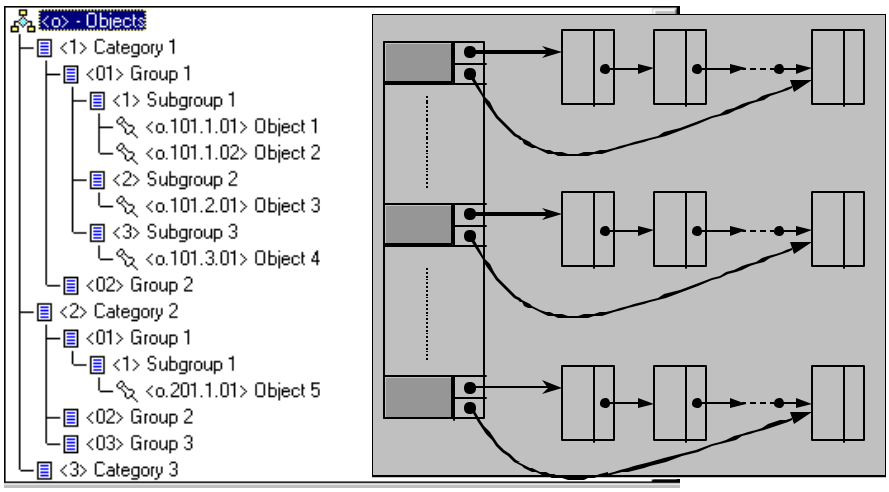

Fig. 1. The TreeView control as a data structure mastering control

The TreeView's content saving is followed by the list data structures storage to disk in a distinct file (2).

```
Type
TOperationFileRec=record
                    Code : String[30];
                    Op : TOperation;  ( 2 )
                  end;
Public  F : File of TOperationRecFile;
```

We start by detailing the *new object inserting mechanism*. The new object represents a new node in the TreeView control (3) and a new position in the table data structure. The new child node is added within the selected subgroup by using the AddChild( ) method with specifying the node's label.

```
procedure TMainForm.AddObjectClick(Sender:
                                   TObject);
Var DIO : TEditObjDIO;
    S : String;
    Aux : TOperationFileRec;
begin
  DIO.HintsActive := HintsActive;
  DIO.Add := True;
  if EditObjDlgRun(DIO) then with Tree do
    begin  S:='.'+DIO.Code;
           S:='.'+GetCod(Items[
              SelectedItem].Text)+S;  ( 3 )
           S:=GetCod(Items[SelectedItem].
              Parent.Text)+ S;
           S:='o.'+GetCod(Items[
              SelectedItem].Parent.Parent.
              Text)+S;
           if CanAdd(S) then
             begin
               Items[SelectedItem].Expand;
               SelectedItem:=
                 AddChild(SelectedItem,'*'+
                       '<'+S+'>'+DIO.Name);
               Aux.Code := S;
               Aux.Op := DIO.Op;
               NewObject(Aux);
             end;
    end
end;
```

At the same time the corresponding operations list must be initialized by creating it's first item (4).

```
procedure TmainForm.NewObject(R : TOperation-
FileRec);
begin Inc(Nr);
      With Objects[Nr] do
        Begin Code := R.Code;
              New(First);              ( 4 )
              First^Op := R.Op;
              First.Next := nil;
              Last := First;
        end;
end;
```

A new operation recording requires object localization in the table data structure and inserting a new element in the suitable operations list (5).

```
procedure  TMainForm.AddOperationClick(Sender:
TObject);
var
    DIO : TEditOperDIO;
    Poz : Integer;
Begin
  with Tree do
   begin
    DIO.HintsActive := HintsActive;
    DIO.Obj:= Copy(Items[SelectedItem].
                   Text,2,255);
    if GetCurentObjPos(Poz) then
     begin
      DIO.PozInObj := Poz;       ( 5 )
      if EditOperDlgRun(DIO) then
       if DIO.Op.Data<=Objects[Poz].Cap^.
                       Op.Data then
         MessageBox3SD('Er',ICON_ERROR,'',
                       '&Ok','',HintsActive)
       else
         InsertOper(Poz, DIO.Op);
     end;
    end;
end;
```

The new item will be inserted after an indicated node ***P*** by respecting the imposed sorting criteria (6).

```
procedure TMainForm.InsertOper(Poz : Integer;
Op : TOperation);
var P, Q : TPointNode;
begin
 if Poz>0 then
  begin
   PositionForData(Poz, Op.Data, P);
   New(Q);
   Q^.Op := Op;
   Q^.Next := P^.Next;
   P^.Next := Q;                          ( 6 )
   ....
  end;
end;

procedure TMainForm.PositionForData(Poz : Integer; Data : TDateTime; var P : TPointNode);
var
  found : Boolean;
begin
  P := Objects[Poz].Cap;
  found := False;
  while (P^.Urm<>nil) and not found do
   begin
    found := P^.Urm^.Op.Data>=Data;
    if not found then P := P^.Urm;
   end;
end;
```

The entire data structures manipulation is initiated through and controlled by the TreeView control. The little sample code presented above is sufficient to illustrate the TreeView's use for mastering data structure.

3. Managing binary tree data structures with a TreeView control

Now, we present some methods, which concern the TreeView using in the binary tree dynamical data structure visualization.

We can add an item in the TreeView and a new node in the binary tree, simultaneously, as in the following situation: we consider a *simple binary tree* described with the data structures (7):

```
type
point = ^nod;
nod = record
        number: integer;              ( 7 )
        left,right : point;
      end;
```

with the unit variables (8)

```
var
 Form2 : TForm2;
 p : point;                           ( 8 )
 r1: ttreenode;
```

We get use the following program:

```
procedure   TForm2.Button1Click(Sender:   TObject);
procedure CreateTree(var t:point;
                     r:ttreenode);
var x: integer;
begin
 x := strtoint(inputbox('','Value',''));
 if x<>0 then
  begin
   {new node in binary tree data structure}
   new(t);
   t^.number:=x;
   {a new item in TreeView1}           ( 9 )
   r:=treeview1.Items.addchildobject(
                          r,inttostr(x),t);
   createtree(t^.left,r);
   createtree(t^.right,r);
  end
 else t:=nil;
end;

begin
 {tree root}
 new(p);
 p^.number:=strtoint(inputbox('',
                    'Value:',''));
 {first item in TreeView1}
 r1:=treeview1.Items.addobject(
                 nil,inttostr(p^.number),p);
 createtree(p^.left,r1);
 r1:=treeview1.Items[0];
 createtree(p^.right,r1);
 treeview1.FullExpand
end;
```

In Figure 2 we present the result in execution for a certain binary tree data structures.

Another method used to view a binary tree dynamical data structure using a TreeView control, is the following: firstly we operate on the data structure (create, new node insertion, node deleting). Secondly, we represent in the TreeView the data structure current form (we traverse the data structure and we add a new item in TreeView).

For exemplify these methods we consider a ordered binary tree dynamical data structure described with (10):

```
type
  point = ^nod;
  nod = record                         ( 10 )
          number : integer;
          left,right : point
        end;
```

and with the unit variables (11):

```
var
 l,z:integer;
 p:point;                             ( 11 )
 r1:TTreeNode;
```

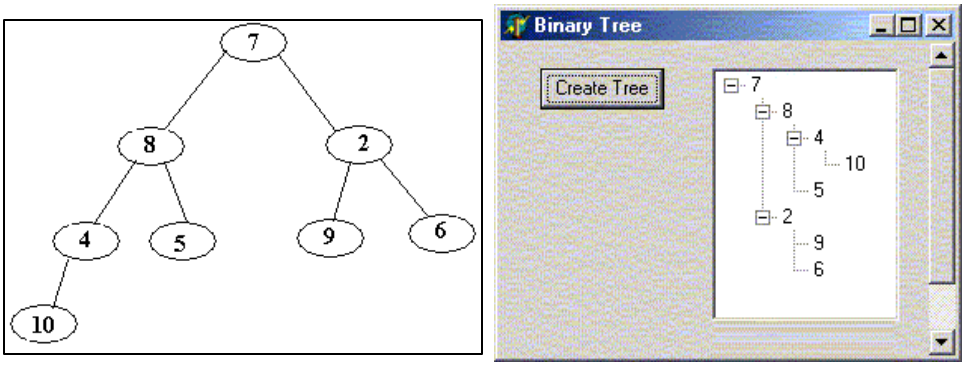

Fig. 2. Data structure and its representation in TreeView

For the binary tree visualization in TreeView – Figure 3, we use the following program:

```
procedure TForm1.Button2Click(Sender:
                                TObject);
procedure preorder(var root:point;
                   r:TTreeNode);
begin
 if root<>nil then
  begin
   r:=treeview1.Items.AddChildObject(
              r,inttostr(root^.number),root);
   preorder(root^.left,r);
   preorder(root^.right,r);
  end;
end;
begin                                    ( 12 )
 treeview1.Items.Clear;
 { first item in TreeView1}
 r1:=treeview1.Items.AddObject(nil,
                    inttostr(p^.number),p);
 preorder(p^.left,r1);
 r1:=treeview1.items[0];
 preorder(p^.right,r1);
end;
```

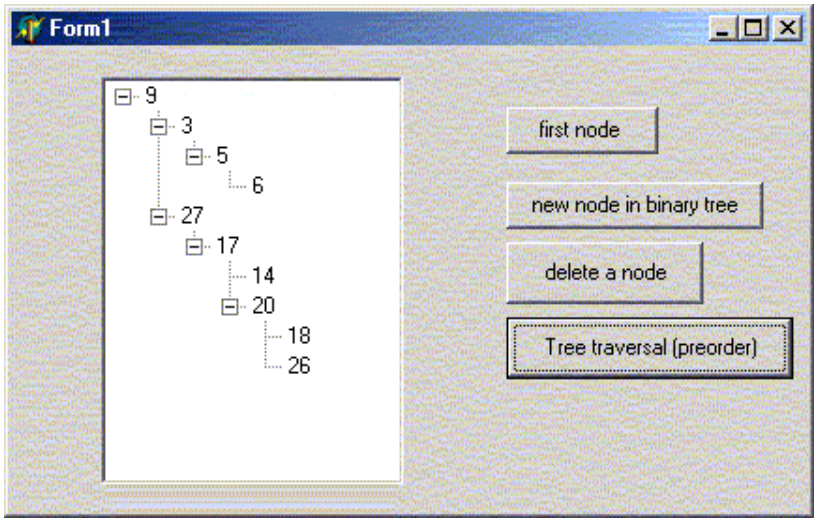

Fig. 3. Ordered binary tree visualization with TreeView

4. The TreeView data structure mastering control

Through the TreeView control users can establish the desired operations on a data structure. On the other hand, all data structure modifications are reflected by the interface control. Therefore, we define a new component - the *TreeView data structure mastering control* – as an interface for managing data structure - Figure 4.

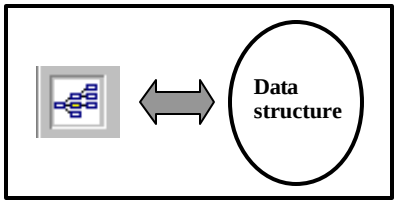

Fig. 4. The TreeView data structure mastering control

Programming our own methods for operating with the data structure through the classical TreeView component, we can define the *principal properties and methods for the proposed TreeView*. These are based on the native TreeView control properties and methods added with the adequate code for implementing the corresponding data structure operations.

5. Conclusions

A TreeView data structure mastering control can be implemented in all RAD environments that support applications development based on dynamic linked data structure algorithms. The data access flexibility that allows rapid movement from one data item to another is visually strengthened by the TreeView's facilities.

References

1. Cretu, V., *Structuri de date si tehnici de programare*, vol. I, Editura "Orizonturi universitare", Timisoara, 2000
2. Muntean M., *Using dynamic object lists in economic applications design*, Bucharest, 2001
3. *** Delphi 6, Development Guide