Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 7 (16): 2226-2240, 2007
ISSN 1812-5654

Logical Graphics Design Technique for Drawing Distribution Networks

Mansoor Al-a'Ali
Department of Computer Science, University of Bahrain, P.O. Box 32038, Bahrain

Abstract: Electricity distribution networks normally consist of tens of primary feeders, thousands of substations and switching stations spread over large geographical areas and thus require a complex system in order to manage them properly from within the distribution control centre. We show techniques for using Delphi Object Oriented components to automatically generate, display and manage graphically and logically the circuits of the network. The graphics components are dynamically interactive and thus the system allows switching operations as well as displays. The object oriented approach was developed to replace an older system, which used Microstation with MDL as the programming language and ORACLE as the DBMS. Before this, the circuits could only be displayed schematically, which has many inherent problems in speed and readability of large displays. Schematic graphics displays were cumbersome when adding or deleting stations; this problem is now resolved using our approach by logically generating the graphics from the database connectivity information. This paper demonstrates the method of designing these Object Oriented components and how they can be used in specially created algorithms to generate the necessary interactive graphics. Four different logical display algorithms were created and in this study we present samples of the four different outputs of these algorithms which prove that distribution engineers can work with logical display of the circuits which are aimed to speed up the switching operations and for better clarity of the display.

Key words: Electricity distribution network, distribution control center, schematic display, display algorithms, Delphi components

INTRODUCTION

Electricity Distribution Systems distribute electricity to a wide variety of customers such as homes, factories, hospitals, hotels, etc. This is achieved through a set of complex circuits spreading all over a town or a country feeding individual substations. Each substation feeds a specific area or a specific hospital or any consumer of electricity. Distribution control engineers are responsible for the distribution control and take a variety of actions to ensure that the distribution is maintained. They achieve this by having a system which enables them to keep track of the flow of electricity, the live and dead circuits, the isolated stations, the on and off switches, etc. Any switching to on or to off of any switch has an impact on the flow of electricity and must be counteracted to make sure that supply is maintained. Control engineers manage such interrelated tasks by having a supporting computer system to keep track of the flow and the status of the circuits and stations. For example, if the engineer wants to isolate a station for maintenance he would reroute electricity from some other source and these actions are registered in the system for monitoring and control.

Current computer systems for managing control centre operations are based on user drawn schematic diagrams coupled with a database to enable the engineer to implement the operations and to keep track of all the vents taking place by other engineers. These systems suffer from serious drawbacks in terms of speed efficiency and display clarity. Since the system is dealing with graphics drawn using a package such as AutoCAD or Microstation and processed by another application system written say in C and the data is stored in a database system such as ORACLE; this takes more time than acceptable to the engineer and some switching operations may take minutes. This speed problem may endanger the filed workers and may cause the supply to be cut off from consumers for many minutes. Another drawback to schematic diagrams is that they are difficult to trace because substations are manually squeezed inside empty slots of the diagram and slowly the diagram becomes unreadable. In some cases the engineers may insert a substation far away from where it should be because there is no space to insert next to the substation feeding it. Therefore, if the engineer wants to trace a circuit he may find himself spanning for long period and by then he would have forgotten where he came from.

We have devised a logical approach to storing, drawing and tracing circuits whereby the engineer does not have to manually insert the graphics and other details of a substation, but instead, he only needs to inter information in the database and the graphics are logically

generated. Our approach allows the user to carry out operations such as putting a switch to off or isolating a station or isolating a cable and the result of this action is a logically generated circuit graphics through appropriate algorithms, (Fig 16 and 17). These logically generated circuit diagrams are well organized and the spacing is cleverly generated to occupy the minimum space which makes tracing a simple, accurate and fast task. Circuits now do not spread over many screens and require a lot of scrolling to trace. Therefore, our logical approach resolves the two main drawbacks of schematic diagrams which are speed and space.

We have conducted a number of studies on the issues relating to the problems of the system at the ministry of electricity distribution control centre at UmElhasam, Bahrain (Al-A'ali, 1999, 2006). These studies were mainly focused at speed and display problems.

Research on automating the Electricity Distribution Control Centers and their activities of monitoring and controlling electricity distribution networks has received some attention in the research literature (Pitrone, 2006; Baxevanos and Labridis, 2007; Manjunath and Mohan, 2007). Most of the research is focused on post fault reconfiguration (Kashem *et al*., 2000; Ming-Yang *et al*., 2005; Carvalho *et al*., 2006). However, no research was directed to the issue of the speed of displaying the graphical distribution of the network which will enable the control engineer to visually study the situation and take the appropriate action. Most of the systems available are simple schematic displays.

Clavijo *et al*. (2001) describe a distributed system based on CORBA technology to provide real-time visual feedback to operators of large supervision systems, but no attention was given to the issue of speed or the quality of graphics display of the circuits. Carvalho *et al*. (2006) proposed an approach to operational planning and expansion planning of large-scale distribution systems.

Herrell and Bekar (1998) proposed a method of modeling of distribution systems in PCS. Yeh and Tram (1997) present an integrated solution for computerized distribution planning in a Geographic Information System (GIS) context, a synergy that magnifies the data accessibility between load forecasting and feeder planning tools, sealing the traditional gap between long-term and short-term distribution system planning. According to these authors, this approach enables the distribution system studies in a GIS context, to best assist utility planners in deciding where and when the customers will grow and how to expand the system facilities to meet the demand growth. Wainwright (1997) present a Distribution Engineering Geographical Information System as a major program of network information data capture at all voltage levels from the customer service cable right through to the primary supply in-feed; a suite of software for viewing the data in the office and in the field; tools for network optimization and a range of automation and information processing systems which assist in the operation and maintenance of the distribution network.

Yeh and Tram (1997) present an integrated solution for computerized distribution planning in a Geographic Information System (GIS) context, a synergy that magnifies the data accessibility between load forecasting and feeder planning tools, sealing the traditional gap between long-term and short-term distribution system planning. According to these authors, this approach enables the multi-year distribution system studies in a GIS context, to best assist utility planners in deciding where and when the customers will grow and how to expand the system facilities to meet the demand growth. Wainwright (1997) presents a distribution engineering geographical information system as a major program of network information data capture at all voltage levels from the customer service cable right through to the primary supply in-feed; a suite of software for viewing the data in the office and in the field; tools for network optimization; and a range of automation and information processing systems which assist in the operation and maintenance of the distribution network. Lestan and Gorisek (1997) present a form of a distribution network automation utility experience. Gorisek (1997) discussed some of their experience with the automation of distribution network utility. Li *et al*. (1999) propose an algorithm and implementation of an incident based connectivity trace system for distribution network. Carvalho *et al*. (1999) proposed an approach to operational planning and expansion planning of large-scale distribution systems.

Some studies were conducted on the issues relating to the problems speed efficiency of displaying distribution networks (Al-A'ali, 1999, 2006). After thoroughly searching the literature, we have found no other reported attempts to present some solutions to the issue of speed problems of distribution control systems. We have found no published work about the problems and methods of display of the distribution network. Therefore, the approach presented in this study and in other studies by the author represents a new direction into the way electricity distribution systems can be managed by the engineers from control centres in a logical way which makes the switching operations easy and quick to do.

This research presents the technical aspect of these logically generated circuit diagrams through the use of Object Oriented technique of the Delphi development platform. To our knowledge there is no reported research on the issue of logically generating electricity distribution networks except those by the author (Al-A'ali, 1998, Al-A'ali, 2006). This paper shows in a step by step way how the graphics of the distribution network can be generated as components of Delphi. These components are then organized in a logical display from the connectivity information stored in the database.

ELECTRICITY DISTRIBUTION SYSTEMS

An Electrical Distribution Network System is a collection of electrical circuits consisting of primary stations and substations linked via cables. Stations are linked to each other through switches, which are fixed in the stations. Every station may contain one or more switches and has one or more bus-bars, where every bus-bar can be considered as a station by itself. Switches are directly linked to cables. Primary stations are the sources that feed other stations, except in some rare situations (parallel connections). Substations are the stations that are fed by primary stations. Switches on primary stations are called circuit breakers. All connecting stations linked to a circuit breaker represent an electrical circuit. Electrical circuits form the Electrical Distribution Network (Fig. 1).

Electricity distribution networks normally consist of thousands of primary stations (feeders), substations and switching stations spread over large geographical areas and thus require a complex system in order to manage them properly in the control center and in the field. However, electricity network circuits which are part of the overall network, can sometimes be spread over large geographical areas and the control engineer can either zoom in or zoom out on these circuits which normally makes the circuits either unreadable, in the case of zooming out, or only one part is readable in the case of zooming in.

The normal tasks of a distribution control centre engineer are based on a large schematic display of the network (Fig. 2), which may comprise tens of thousands of stations, switching stations and supply stations (primaries). Previous distribution network was digitized using Microstation. These networks were then programmed to carry out many of the control centre operations such as adding new stations, retiring old stations, tracing the flow of a given circuit, isolating a station for maintenance, isolating a complete circuit for

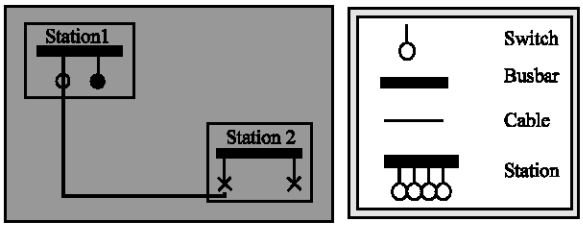

Fig. 1: The components of a circuit

maintenance, redirecting the flow of electricity, etc. These operations require decision making capabilities which can be provided either by the engineer at the control centre or by the system. The distribution network is all interconnected and the engineer can zoom in or zoom out to identify the particular circuit or station of interest. But once the engineer zoomed in or out on a given area of the display, he would be faced with a manually drawn network which would have grown over the years and would have no logical display pattern and thus difficult to follow which would in turn either slow or hinder the decision making process. Since the control centre engineer would normally be working on more than one region or circuit and controlling a number of teams out there in the field, he would find the illogical schematic display of great hindrance. The sample network shown in Fig. 2 was taken from the Distribution Control Centre at UmElhasam in Bahrain, which was drawn using Microstation and programmed using MDL tool and ORACLE as the database. Microstation contains the coordinates of each object in the network such as switches, bus-bars and cables, whilst ORACLE keeps information about the connectivity of the circuits and the status of each station or circuit if it is on or off and the direction of flow of electricity. The ORACLE-Microstation schematic display system known as EDMS, suffered from two main problems: 1) a very large confusing schematic display comprising over ten thousand stations and 2) speed problems where one circuit may sometimes take up to two minutes to display because of the architecture of the system which was based on an ORACLE database coupled with Microstation and MDL through an ODBC.

These monitoring and control activities of a distribution control centre include switching the flow from on to off and vv, preparing switching plans such as those required for isolating stations or circuits for maintenance, adding and removing substations, etc. All these activities and more have been incorporated in the Hi-Tech EDMS system.

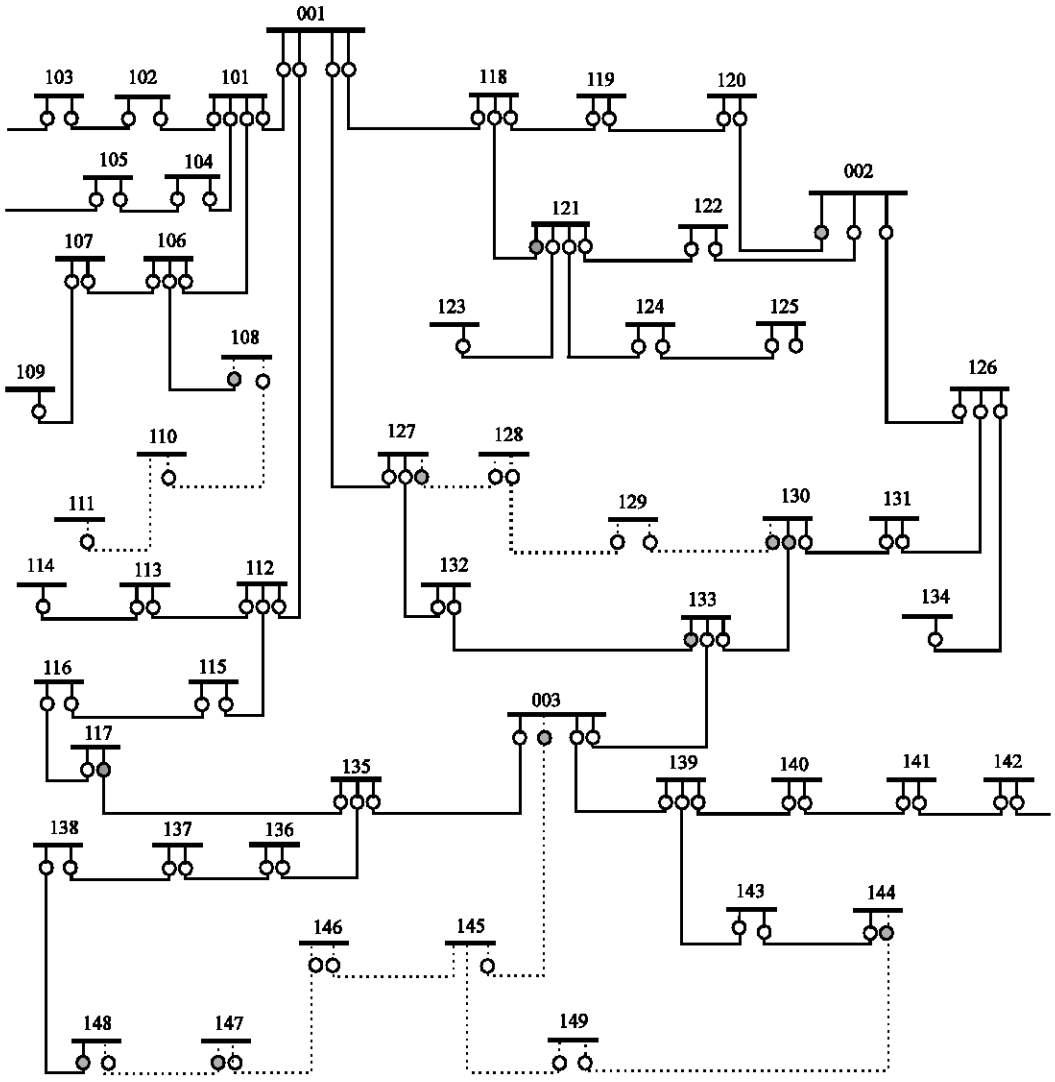

Fig. 2: A sample of a schematic distribution network digitised manually using Microstation

GENERATING CIRCUIT GRAPHICS COMPONENTS

In this project, various components were created in Delphi, each representing a different device in the network. These components are Switches, Cables, Bus-bars and Stations, all of type TGraphic. A station is constructed of five switches and one or more bus-bars. Cables link stations to each other.

As an example, one of the components created in the project will be illustrated in details. The others follow the same technique, except for the station component that needs some additional features and procedures that will be discussed laters. Figure 1 shows the different elements of the network that were created as new components in this project.

The graphic component presented in this chapter is TS, that is, the switch component.

Creating a graphic component requires three steps:

- Creating and registering the component
- Publishing inherited properties
- Adding graphic capabilities

Creating and registering the component: The following general procedure for creating a new component was followed:

- A new unit is created and called Oswitch.
- A new component type is derived and called TS, descended from TGrphicControl.

- TS is registered on the Samples page of the component palette.

Figure 3 shows the component Switch
The resulting code looks like listing 1:

```
<<Listing 1
    unit Oswitch
    interface
uses SysUtils, WinTypes, WinProcs, Messages, Classes,
Graphics, Controls, Forms;
type
    TS = class(TGraphicControl)
    end;
procedure Register;
implementation
procedure Register;
begin
    RegisterComponent('Samples', [TS]);
end;
end.
```

Publishing inherited properties: Once you derive a component type, you can decide which of the properties and events declared in the protected parts of the ancestor type you want to make available to the users of the component. TGrphicControl already publishes all the properties that enable the component to function as a control; all you need to publish is the ability to respond to mouse event.
For the TS control, one property is needed only:

```
type
    TS = class(TGraphicControl)
    published
        propertyOnClick;
        end;
```

The switch control now makes the On click interaction available to the users.
Other properties are also needed such as,

```
published
    property Hint;
    property ShowHint;
    property Visible;
    property PopUpMenu;
```

Adding graphic capabilities: Once the graphic component is declared and any inherited properties are published, graphic capabilities can be added to distinguish between components.

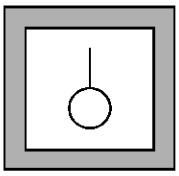

Fig. 3: The component switch

Two steps are always performed when creating a graphic control:

- Determining what to draw
- Drawing the component image

Determining what to draw: A graphic control generally has the ability to change its appearance to reflect either a dynamic condition or a user-specified condition or both. In general, the appearance of a graphic control depends on some combinations of its properties. To give the shape, control a property to determine the shape it draws, add a property called TstatusType which requires three steps:

- Declaring the property type
- Declaring the property
- Writing the implementation method

Declaring the property type: For the switch state, an enumerated type is needed to show the color and repaint the switch in its new state. The following type definition is added above the TS control object's declaration:

```
type
TstatusType = (stOn, stOff, stEarth);
TstatusType = class(TGraphicControl)
```

Declaring the property: To declare a property, usually a private field is used to store the data for the property and then methods for reading and/or writing the property value are declared too.

For the TS control, a field that holds the current shape must be declared and then a property that reads that field and writes through a method call is declared. The following declarations are added to TS:

```
type
    TS = class(TGraphicControl)
    private
        FStatus : TStatusType;
        procedure SetStatus(Value : TstatusType);
    published
    property status :TStatusType read FStatus write
SetStatus default stOn;
    end
```

Writing the implementation method: When either the read or write part of a property definition using a method call instead of directly accessing the stored property data, you need to implement these methods. The implementation of the SetStatus method is added to the implementation part of the unit:

```
procedure TS.SetStatus(Value :TstatusType);
 begin
    if FStatus <> Value then
    begin
       Fstatus := Value;
       case FStatus of
            StOn : Brush.color := clWhite;
            StOff : Brush.color := FOffColor;
        end;
        Invalidate;
       end;
end;
```

Figure 4 shows the switch status in both cases, on and off. By default, the switch takes the on status, but when the user changes its value in the object inspector, it repaints itself in black to represent the off status. Other variables declared in the same way as FStatus are as follows:

```
type
   TS = class(TgraphicControl)
   private
                FlastColor : TColor;
                Fselected : Boolean;
                Fpen: TPen;
                Fbrush: TBrush;
             Fcode : String;
                FSTationCode : String;
                FbusCode : String;
                FoffColor : TColor;
                FstationType :TSTation;
 procedure SetOffColor (Value : TColor);
 procedure SetCode(Value : String);
 procedure SetStationCode(Value : String);
 procedure SetBusCode(Value : String);
 procedure SetStationType(Value : TStation);
 procedure SetSelected(Value : Boolean);
 public
     property Selected : Boolean read FSelected write
     setSelected default False;
     property Code : String read FCode write SetCode;
     property StationCode : String read FStationCode
     write setStationCode;
     property BusCode : String read FBusCode write
```

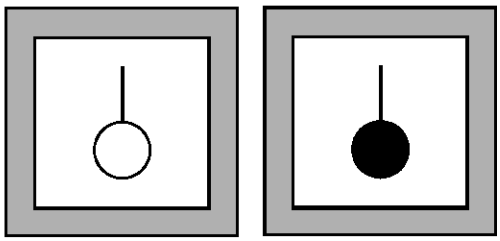

Fig. 4: Switch on/off status

```
     SetBusCode;
 published
     property offColor : TColor read FOffColor write
setOffColor;
end;
```

Overriding the constructor and the destructor: In order to change the default property values and initialize owned objects for the component, the inherited constructor and destructor must be overridden.

Changing default property values: In this example the switch control sets size to a rectangle with a height of 20 and width equal to 10 pixels. It is done as follows:

•

```
type
     TS = class(TGraphicControl)
     public
               constructor Create(AOwner :TComponent);
               override;
     end;
```

- The height and width properties are redeclared with their new default values, see code below.

```
type
     TS = class(TGraphicControl)
     published
               property Height default 20;
               property width default 10;
     end;
```

- The new constructor in the implementation part of the unit looks like this:

```
constructor TS.Create(AOwner: TComponent);
begin
     inherited Create(AOwner);
     Height :=20;
     Width := 10;
end;
```

Publishing the pen and brush: By default, the canvas has a thin, black pen and a solid, white brush. To enable developers using the shape control to change those aspects of the canvas, objects must be provided for them to manipulate at design time, then copy those objects into the canvas when painting. Objects such as an auxiliary pen or brush are called owned objects because the component owns them and is responsible for creating and destroying them. Managing owned objects requires three steps:

- Declaring the object fields
- Declaring the access properties
- Initializing owned objects

Declaring the object fields: For each object owned by a component, it must have an object field declared for it in the component. The field ensures that the component always has a pointer to the owned object so it can destroy the object before destroying itself. In general, a component initializes owned objects in its constructor and destroys them in its destructor. Fields for pen and brush objects to the TS control are added:

```
type
    TS = class(TGraphicControl)
    private
            Fpen : TPen;
        Fbrush : TBrush;
    end;
```

Declaring the access properties: You can provide access to the owned objects of a component by declaring properties of the type of the objects. That gives developers using the component a way to access the objects either at design time or at run time. In general the read part of the property just references the object field, but the write part calls a method that enables the component to react to changes in the owned object.
The following lines are added to the code :

```
type
    TS = class(TGraphicControl)
    private
        procedure SetBrush(Value : TBrush);
        procedure SetPen(Value : TPen);
    published
        property Brush : TBrush read Fbrush write
    SetBrush;
        property Pen : Tpen read Fpen write SetPen;
    end;
```

Then SetBrush and SetPen are written in the implementation part of the unit:

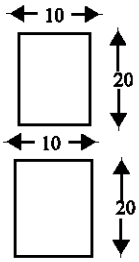

Fig. 5: The boundaries of the switch component

```
procedure TS.SetBrush(Value : TBrush);
begin
    FBrush.Assign(Value);
end;

procedure TS.SetPen(Value : TPen);
begin
    FPen.Assign(Value);
end;
```

Initializing owned objects: If you add objects to your component, the component's constructor must initialize those objects so that the user can interact with the objects at run time. Similarly, the component's destructor must also destroy the owned objects before destroying the component itself.
Steps are as follows:

- Constructing the pen and brush to the switch control constructor:

```
constructor TS.Create(AOwner : TComponent);
begin
    inherited Create(Aowner);
    Width := 10;
    Height := 20;
    Pen := TPen.Create;
    Brush := Bbrush.Create;
end;
```

Figure 5 shows the boundaries of the component switch with height = 20 and width = 10 pixels.

- Adding the overridden destructor to the declaration of the component object

```
type
        TS = class(TGraphicControl)
    public
        constructor create(Aowner : TComponent);
override;
    destructor destroy; override;
    end;
```

- Writing the new destructor in the implementation part of the unit

```
destructor TS.Destroy;
begin
    Fpen.Free;
    Fbrush.Free;
    inherited Destroy;
end;
```

Setting owned objects' properties: As one last step in handling the pen and brush objects, we need to make sure that changes in the pen and brush cause the switch control to repaint itself. Both pen and brush objects have OnChange events, so we can create a method in the switch control and point both OnChange events to it.

The following are added to the switch control and the component's constructor is updated to set the pen and brush events to the new method:

```
type
    TS = class(TGraphicControl)
    published
        procedure StyleChanged(Sender : TObject);
    end;

    implementation

constructor TS.Create(AOwner : TComponent);
begin
            inherited Create(Aowner);
            Width := 10;
            Height := 20;
            Pen := TPen.Create;
            Fpen.OnChange := StyleChanged;
            Brush := Bbrush.Create;
            Fbrush.OnChange := StyleChanged;
    end;

    procedure TS.StyleChanged(Sender : TObject);
    begin
        Invalidate.True;
    end;
```

With these changes, the component redraws to reflect changes to either the pen or the brush.

Drawing and refining the component image: The essential element of a graphic control is the way it paints its image on the screen. The abstract type TGraphicControl defines a virtual method called Paint that you override to paint the image on the control.
The Paint method for the switch control needs to do several things:

- Use the pen and brush selected by the user
- Use the selected shape
- Adjust coordinates

Overriding the Paint method requires two steps:

- Adding Paint to the component's declaration,

```
type
    TS = class(TGraphicControl)

    Protected
        procedure Paint; override;
    end;
```

- Writing the Paint method in the implementation part of the unit,

```
    procedure TS.Paint;
    begin
        With canvas do
        begin
            case FstationType of
                        StSubStation :
                        begin
                        Pen := Fpen;
                        Brush := Fbrush;
                        if status = stOff then
                        Brush.Color := offColor;

            Ellipse(0, 10, 10, 20);
            MoveTo(5, 0);
            LineTo(5, 10);
        end;
        StPrimary :
        begin
                        Pen := Fpen;
                        Brush := Fbrush;
                        if status = stOff then
                        Brush.Color := offColor;

                        MoveTo(0, 10);
                        LineTo(10, 20);
                        MoveTo(10, 10);
                        LineTo(0, 20);
                        MoveTo(5, 0);
                        LineTo(5, 15);
            end;
```

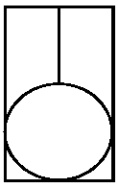

Fig. 6: Shape and pixels of the switch component

The resulting shape of the switch is depicted as in Fig. 6. It consists of a 10 pixels long line and a circle adhered to that line with a radius of 10 pixels.

By doing the previous steps, the switch control is ready to be implemented and used.

As seen before, the programming part of this project is done mainly using the Cable and Bus-bar components. These components were designed and implemented to be used by any other application independently from this project. This chapter will explain how to use these components.

INSTALLING THE COMPONENTS

Although we used Station and Cable components, the child components (Switch and Bus-bar) can be used too. Child components mean that their properties are inherited from their parents. In this case the parent is the component Station. Firstly, you must install these components in Delphi to appear in the Visual Component Library, or VCL, together with Delphi's components.

- You must have the following files:
  - oswitch.pas
  - cable.pas
  - busnar.pas
  - pstation.pas
  - untGrid.pas.

- Save these files in one directory.
- Run Delphi if it is not running.
- To install the components into Delphi use one of the following methods:

  - Pick Component | Install component from the menu.
  - Specify the files to be installed (not necessary all of them).
  - By pressing the button OK, the following Dialog box (Fig. 14) will appear showing the files you have selected. The path in this case is D:\ours\components (you can put these components in any other directory).
  - Finally, press Compile and save your work. The new components will appear in a new palette called Circuits in your VCL.

Using the components: After installing the components in Delphi, you can use the new components exactly as if they came with Delphi. You can put them in any form, modify their properties at Design time or call their methods at design time by attaching codes for their events. The key properties and methods for each component will be described in the next sections.

Switch component: The design time properties will appear in the Object Inspector as in Fig. 6. You will be familiar with most of these properties except offColor, StationType and Status. OffColor refers to the color of the switch when it is in its off status. StationType is a field to determine the type of the station whether it is a primary station or a substation and Status specifies the status of the switch whether it is on, off or earth. To change their values at design time you just write the new value or pick it from a dropdown list. During run time, for example, you can attach the following code to the OnClick Event for Button1 in order to change the status from stOn to stOff:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    S1.Status := stOff;
end;
```

Cable component: Cable component is nothing more than a line that can be drawn vertically or horizontally. The property that controls this is called HV and has two values (tV, tH). Place a Cable in a form and trigger this value to see the result. Figure 7 shows the cable with HV set to tV, while Fig. 8 shows the same cable with HV set to tH.

Bus-bar component: The Bus-bar is a simple rectangle. Its properties are a subset of properties defined in the Delphi's TShape Component. No special properties are defined (Fig. 8).

Station component: Station component is the most important object in this project. Fig. 9 shows the component at design time.

The Object Inspector in Fig. 9 shows the Station properties. As you notice, five of these properties are themselves objects and can be expanded one level more to set their properties. These objects are S1, S2, S3, S4 and S5. These objects are identical to each other and have the same set of properties. Each of them is an ordinary switch.

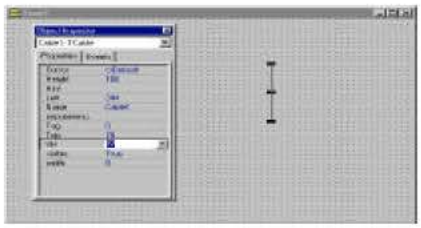

Fig. 7: The vertical element of the cable component

Fig. 8: The horizontal element of the cable component

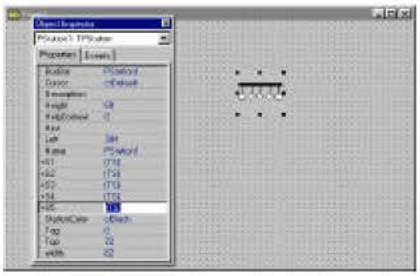

Fig. 9: The station component with switches at design time

Therefore, expanding any of them will give the same set of properties defined for the switch component. You can set the Station properties at design time using the Object Inspector as described in previous sections. An example of using this component at run time is to change the Station's Color. To change the stationColor property at run time, you may write the following piece of code:

PStation1.StationColor := clRed;

Unit untgrid: This unit includes many important procedures that can be used altogether with the previous components to produce meaningful circuits. This unit is designed to produce circuits at run time. The most important procedures of this unit will be described in the following sections.

Note that this section describes how to use the procedures and functions only and not how to create them. Note also that, to use any of the following

Fig. 10: A station at run time (with switches)

procedures, untGrid must be added in the Uses part of the unit that will use the untiGrid. For example, you may write:

Uses untGrid;

Then, you must have a variable of type TGrid which is defined in untGrid. The following procedures and functions are methods of this variable. You may write the following code to create such variable:

Var myGrid : TGrid;

Creatstation function: The syntax of the function is as follows:

function CreateStation(Par : TWinControl; h, v : integer; code : string; Color : TColor): integer;

This function is used to create a station during run time (Fig. 10). Par is the parent that this station will be drawn on. Variables h and v determine the position to draw on. Code is a string that is used to specify the name of the station. Color is the color of the station used in drawing. It retains an integer that is used as a handle for this station. This integer is used in other procedures within untGrid unit to refer to the same procedure.

To create a station, prepare a form with a button and attach the following code to the On Click event for that button:

x : = myGrid.CreateStation(form1, 2, 2, 'Station1', clBlack);

This statement will draw a black station called Station 1 on form 1, at row 2 and column 2. The variable X is an integer that will hold the retained integer. As shown in Fig. 10, the station initially consists of a bus-bar only. To add switches you have to use SetSwitch function as described in the next section. Figure 10 shows a station with two switches, one switch status is on and the second is off. Figure 11 shows a stations without any switches.

Fig. 11: A station at run time (without switches)

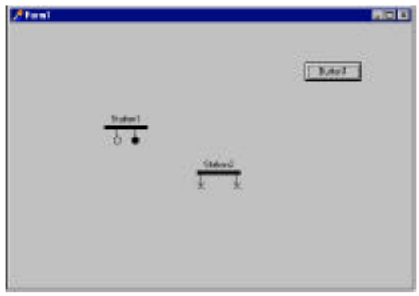

Fig. 12: Generated stations

Setswitch procedure:

The syntax is:
procedure SetSwitch (Index, Order : integer; status : TStatusType; ST : TStation; SwitchCode,BusCode, StationCode : String; M : TPopUpMenu);

This function is used to set the status of a switch in a specific station. Index is an integer that refers to a station, as described in the previous section. Order is an integer form 1 to 5 that determines the location of the switch relative to this station. Status is a variable of type TstatusType that determines the status of the switch to be drawn (on, off or earth). ST is a variable of type Tstation that determines the type of the station; if the station is primary, the switch will be drawn as an 'X' shape otherwise, it will be drawn as a circle. SwitchCode, BusCode and StationCode are strings that specify the name of the Switch, Bus-bar and Station, respectively. M is a popup menu that will be displayed whenever the switch is clicked. As an illustrating example, adding two switches to the previous station is done by adding the following code:

myGrid.setSwitch(x, 2, stOn, stSubStation, 'Switch2', 'BusCode', 'stnCode', M1);
myGrid.setSwitch(x, 4, stOff, stSubStation, 'Switch4', 'BusCode', 'stnCode', M1);

The first statement will create a switch in the second position (relative to the station) with ON status, while the second statement will create a switch in the forth position with OFF status (Fig. 11).

Linkstations procedure:
procedure LinkStations(Index1, Index2, O1, O2: integer; Par : TWinControl; CableCode : integer; Color : TColor; M : TPopUpMenu);

This procedure is used to link any two stations determined by Index1 and Index2. Par is the parent (background), which the link will be drawn on. CableCode

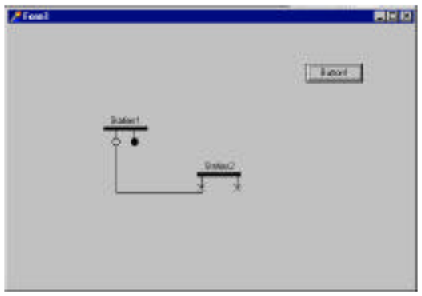

Fig. 13: Generated connected stations

is an integer that uniquely identifies the cable. Color is the color of the cable that will be used in drawing. M is a popup menu that will be displayed if the cable is clicked. To test this procedure, we'll use the previous example in the last section. First we need to create another station and then link between the two stations. The following lines of code must be added:

y: = myGrid.CreateStation (form1, 4, 3, 'Station2', clBlack);
myGrid.setSwitch(y, 1, stOn, stPrimary, 'Switch1', 'BusCode', 'stnCode', M1);
myGrid.setSwitch(y, 5, stOn, stPrimary, 'Switch5', 'BusCode', 'stnCode', M1);

Figure 12 shows form1 after adding a second station by writing the above three lines.

Now, the link between these two stations is performed by writing the following statement:
myGrid.LinkStations(x, y, 2, 1, form1, 1234, clBlack, M1);

This statement will link the two stations. The end points of the link are switch 2 in the first station and switch 1 in the second station. The final result is shown in Fig. 13.

Delphi does not support the feature of drawing a single line that is not straight. The cable that linked

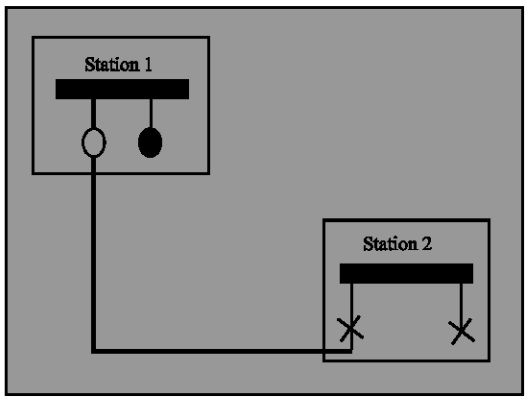

Fig. 14: Stations connectivity

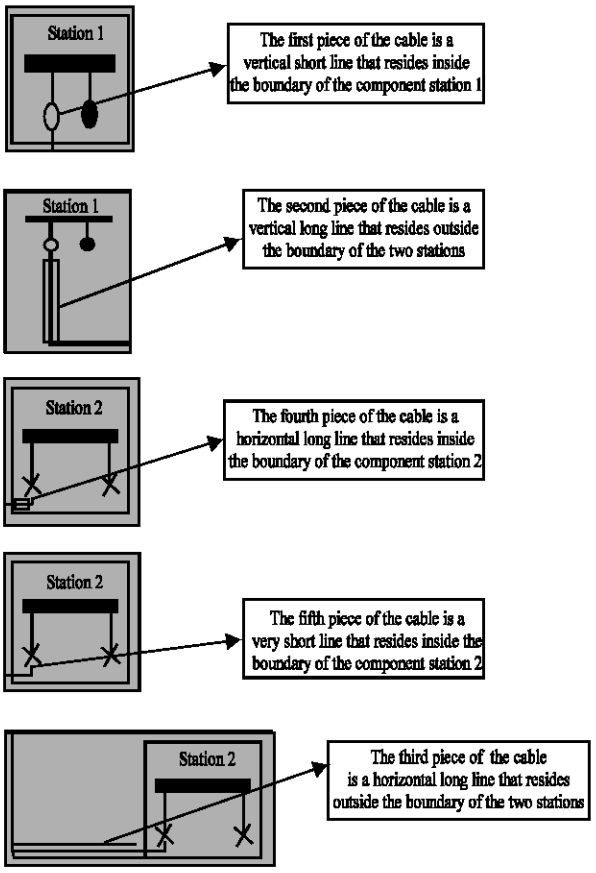

Fig. 15: Logically generating cable drawing

station 1 and station 2 in the previous example consists of 5 separate pieces of the same object, which is the cable object.

Figure 14 shows the link between the two stations. From the user point of view, the connection is one single cable, but from the technical point of view, the cable is constructed of five separate cables joined together through special mapping functions. The following figures make the picture clearer (Fig. 15).

Figure 16 presents the four approaches especially developed to logically generate and draw the different circuits. These four different approaches are based on

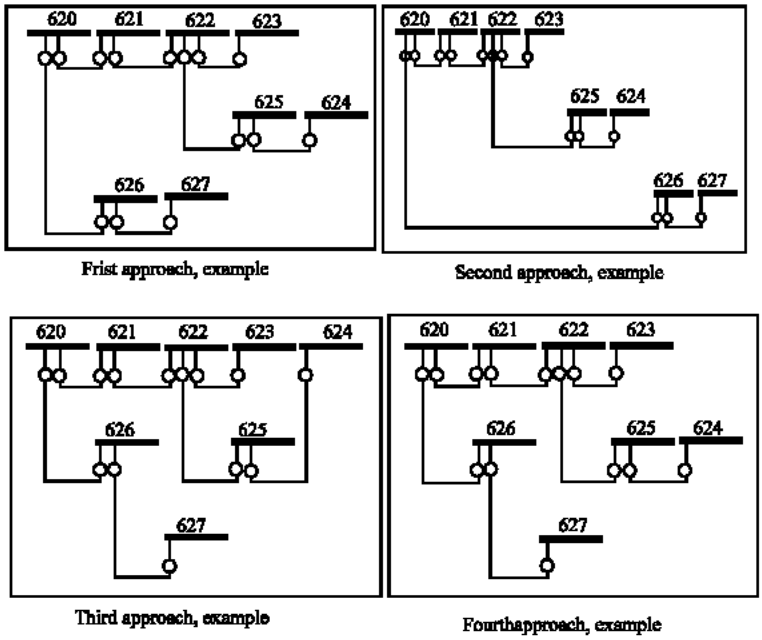

Fig. 16: Four different techniques for logically generating and drawing circuits

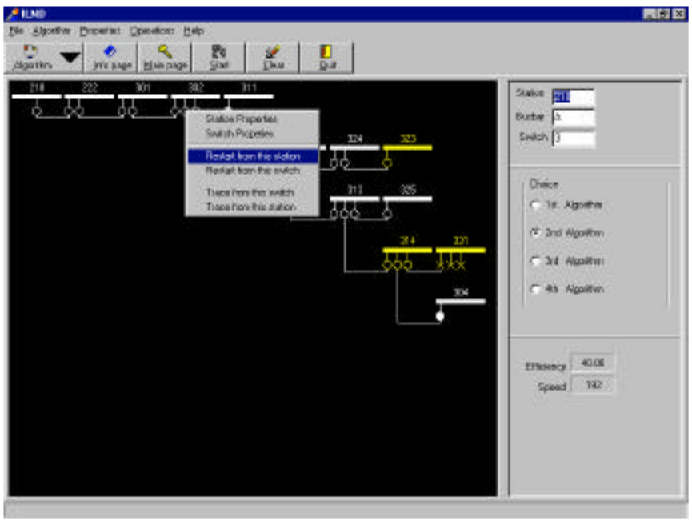

Fig. 17: A snapshot of a screen showing a logically generated circuit

four different algorithms which utilize space and speed up the drawing. Figure 17 shows a snapshot of a screen showing a logically generated and drawn circuit.

IMPLEMENTATION OF THE FULL SYSTEM IN DELPHI

A number of procedures and operations are used to perform different network operations on the Electrical Distributed Management System. These networks were all programmed using Delphi to carry out many of the control centre operations such as adding new stations, retiring old stations, tracing the flow of a given circuit, isolating a station for maintenance, isolating a complete circuit for maintenance, redirecting the flow of electricity, etc. These operations require decision making capabilities which can be provided either by the engineer at the control centre or by the system. It is beyond the scope of this article to explain how each of these procedures was designed and programmed.

The grid: All the procedures that are used to carry out the various network operations on the network are performed logically on a grid and physically on a panel. The grid is defined as a two-dimensional matrix initially set to empty. Cells are marked as non-empty if they are occupied by an element on the panel. Two procedures are used to perform the mapping function between the grid cells and the panel pixels. To draw an element on the panel, its position is marked first on the grid and then the mapping function is used to get the corresponding x-y coordinates on the panel and finally draw the element at that position.

Draw procedure: Starting from any switch, station or a primary, this procedure draws the complete circuit of the given station to the open points ('off' switches) or to the connecting primary station. Firstly, the starting station is taken as an argument. All the 'on' switches in this station are pushed in the stack. Then repeatedly an element of the stack is popped and the connecting station of the switch in that element is looked up. At this stage both stations are drawn by applying one of the algorithms. Then all of its out (on) switches are pushed into the stack and so on. This operation of pushing on the stack continues until reaching an open point (off switch) or a primary station (circuit breaker). The procedure terminates when the stack is empty.

Four different approaches to display the network were developed and tested. Approach 1 drew the circuits with the least space utilization. Space utilization was improved with approaches two, three and four consecutively. The effect on speed of display and operations was minimal, but space utilisation was drastically improved. The readability of these logical displays were shown to control engineers and it was decided that approach three was the most readable by them although approach four gave the maximum possible space utilization. Figure (Fig. 17) shows a snapshot of how one of the network displays is actually shown in Delphi.

CONCLUSIONS

Schematic displays are tedious to follow by the control engineers since stations are spread over large geographical areas and the process of zooming in and zooming out is not of great help. The tedious process of adding new stations or retiring unwanted stations becomes very easy when using the logical display rather than the schematic display. This paper proposed and demonstrated new techniques for logically generating and displaying electricity distribution network circuits using Delphi components. The new approach proved to be more efficient in terms of speed, readability and in carrying out routine operations on the network. The system allows dynamic control over the network by allowing operations on the switches.

REFERENCES

Al-A'ali, M., 1999. The effect of different data structures on speed in an electrical distribution network. DISTRIBUTEC Conference, 1999, UK.

Al-A'ali, M., 2006. Improving the speed of electrical distribution network systems by improving data representation techniques. WSEAS Trans. Circuits Syst., 5: 1124-1131.

Baxevanos, I.S. and D.P. Labridis, 2007. Implementing multiagent systems technology for power distribution network control and protection management. IEEE Trans. Power Deliv., 22: 433-443.

Carvalho, P.M.S., L.M.F. Barruncho and L.A.F.M. Ferreira, 1999. An evolutionary approach to operational planning and expansion planning of large-scale distribution systems. 1999 IEEE Transmission and Distribution Conference, 1999.

Carvalho, P.M.S., L.A.F.M. Ferreiraa and L.M.F. Barrunchob, 2006. Optimization approach to dynamic restoration of distribution systems. Int. J. Electrical Power Energy Sys.,

Clavijo, J.A., M.J. Segarra, C. Baeza, C.D. Moreno, R. Sanz, A. Jimenez, R. Vazquez, F.J. Diaz and A. Diez, 2001. Real-time video for distributed control systems. Control Engineering Practice, 9: 459-466.

Gorisek, J., 1997. Distribution network automation utility experience. 10th International Conference on Power System Automation and Control. PSAC'97, pp: 103-108.

Herrell, D. and B. Beker, 1998. Modeling of power distribution systems in PCS. IEEE 7th Topical Meeting on Electrical Performance of Electronic Packaging, pp: 159-162.

Li, H., Hsiao-Dong Chiang, W.G. Gale and J.T.F. Bennett, 1999. An incident based connectivity trace system for distribution network: Algorithm and implementation. 1999 IEEE Power Engineering Society Summer Meeting.

Ming-Yang, H., C.S. Chena and C.H. Linb, 2005. Innovative service restoration of distribution systems by considering short-term load forecasting of service zones. Int. J. Elect. Power Energy Syst., 27: 417-427.

Kashem, M.A., G.B. Jasmon and V. Ganapathy, 2000. A new approach of distribution system reconfiguration for loss minimization. Elect. Power Energy Syst., 22: 269-276.

Lestan, D. and J. Gorisek, 1997. Distribution network automation utility experience, 10th Int. Conference on Power System Automation and Control. PSAC'97, pp: 103-118.

Manjunath, K. and M.R. Mohan, 2007. A new hybrid multi-objective quick service restoration technique for electric power distribution systems. Int. J. Elect. Power Energy Syst., 29: 51-64.

Pitrone, N., 2006. Computer based tools for distribution network automation. Proceedings of the 6 th IASTED International Conference on European Power and Energy System, 161-166.

Wainwright, I., 1997. Engineering the benefits of a geographical information system: The business case for GIS (Conference Paper). Wainwright, I. IEE Colloquium on Engineering the Benefits of Geographical Information Systems (Digest No.1997/105). IEE, London, UK., pp: 1-7.

Yeh, E.C. and H. Tram, 1997. Information integration in computerized distribution system planning. IEEE Trans. Power Syst., 12: 1008-1013.