Managing Semantics of Graphic Components Through Remodeling Traditional Display Files

Dr. Hari Ramakrishna,
CBIT/ Computer Science and Engineering, Hyderabad, India
Email: dr.hariramakrishna@rediffmail.com

***Abstract*—Management, reuse and customization of semantics of graphic components through remodeling traditional display file models and graphic frameworks is presented. All these models are well tested in several graphic applications. A model debug driver tool application is adopted for the presentation of these models. Typical C++ code segments and Microsoft based visual studio application outputs are presented. These models are more generic and they can be used in any application domain using any language and development environment that support basic graphic primitives. The traditional ways of handling display files are also presented.**

***Index Terms*—Display files, Semantic graphic behavior, graphic components, graphic frameworks, debug driver tool, Microsoft graphic applications, object oriented models and frameworks. Dynamic display files.**

I. INTRODUCTION

Software industry is looking for rapid application development mechanisms with client orientation and short time span delivery, increasing quality and with-standing rapid changes in technology and requirements.

In this connection exporting third party tools plays a major role. Using third party tools decreases testing time. Development of frameworks for increasing degree of reuse has become an important focus. Customization of such frameworks as per client requirements increases importance of frameworks. Frameworks are different from libraries; client code is embedded in frameworks whereas client code includes and calls libraries. [1]

Display file concepts are traditionally used in several graphic models. Geometry of a graphic system is stored as a set of graphic instructions in display file. Display file interpreter recognizes these commands and generates graphics as per the client requirements.

Present day graphic and CAD systems are used at advanced levels. Present graphic systems demand implementation, and simulation of graphic components such that they mimic original real time components. Such requirements demand managing semantics of graphic components. These components should be in a position to implement their behavior, communicate with other components of the system and generate required graphics dynamically. These requirements demand remodeling of traditionally used graphic display file concepts.

II. TRADITIONAL DISPLAY FILE CONCEPTS

Display files store graphic behavior of elements of any graphic or GIS systems in a specific defined structure. The display file for general purpose interactive graphics software is divided into a set of segments such that each segment corresponds to a component of the overall display file. For example, in a building graphics information system, each civil engineering building element is treated as a segment. In other words, windows, doors, racks etc, which are known as civil engineering building elements, are stored in the display file as graphics segment. Sets of attributes are associated with each segment. All these attributes of segments are stored in segment-table.

The information that must be associated with each segment and how the information might be organized are important in understanding display file concepts. Each segment has its own unique name, and it can be referred with that name. For performing some segment operations like changing the visibility of segment, distinguishing the segment from other segments is required. When referring to a display file segment, set of display file instructions that belong to that segment are required. This may be determined by knowing where the display file instructions for that segment begin, and how many of them are there in its specific display file. For each segment, we need some way of associating its display file position information and its attribute information with its name. Sample display file attributes are listed in table 1.

Segmentation can be managed through a set of procedures to create, open, close and transform a segment. Sample user-routines needed to manage segments are listed in table 2.

The object-oriented dynamic display file models presented in this paper do not need implementation of segmentation requirements. The objects take care of

segmentation requirements as each graphic segment is defined as an object.

III. DISPLAY STRUCTURE MODELS

In the structure of the traditional display file, each display file command contains two parts: operation code (opcode), and operands. Opcode indicates what kind of command it is. Operands refer to required coordinates and other arguments required for executing the opcode. The display file is made up of a series of these instructions. The Display file stores all this information forming a huge storage. In the new concepts, this problem is solved as the display file for each component is generated dynamically. Only domain specific component state, and behavior identity are stored in the object itself. This information is helpful in the generation of display file content required for rendering that object.

In the traditional model, the display file must be large enough to hold all the commands needed to create the image. One must assign meaning to the possible operation code before proceeding to interpret them. For example, in a building graphical information system, various geometrical elements such as point, line, circle, arc and polygon may be considered. The general attributes of any simple display file instruction are -- the type of the geometrical element and its color, required coordinates and other geomantic information specific to that element.

The instruction is interpreted by invoking the required vector generator. The vector generators of special geometrical elements may need more information than that available in the main display file. This information is also in the form of graphics commands, which are stored in a separate display file. For example, all the instructions for plotting a polygon are in the polygon display file. Each vector generator of this type has its own interpreter for the interpretation of these commands. The starting-address and size of these instructions also are the needed attributes which are stored in the main display file. Figure 1 shows model sample storage of this type.

The information of the display file is useful to model the object and create the required image. The reason behind this is two-fold: some measure of device-independence is achieved, and it is easy to perform image transformation by changing the position and orientation of the required image. The display file contains the information necessary to construct the required image. The information can be in the form of instructions such as “move the pen”, “draw a line”, and “plot the required polygon”.

Saving instructions such as these usually takes much less storage than saving the picture itself. Each instruction indicates an action for the display device. A display file interpreter is used to convert these instructions into actual images. The display file interpreter serves as an interface between graphics program and the display device. The display file instruction may be actually stored in a file either for a display layer or for transfer to another machine. Such files of imaging instruction are sometimes called ”metafiles”. Table 3 presents sample vector-generating algorithms.

The vector generation algorithms used for dynamic display file based graphic framework presented are supported as a set of function libraries developed using Microsoft MFC classes. These algorithms are portable to any development environment that supports basic graphic primitives.

These functions are used by the display file interpreter while converting the display file instructions into the required picture on the display device. This process of generating image makes our graphics software independent of the nature of the display device and upon its software.

Whatever may be the way of storing and plotting the required images we require some tools for interaction with the graphics system. Table 4 presents various sample user-routines for building-graphics information system. Figures 1 and 2 present display file models for graphic framework. [2-6]

III. DYNAMIC DISPLAY FILE CONCEPTS

Dynamic display files are modeled to enable computation of graphic geometry online as per the behavior of the components. Such models enable graphic components to mimic real time components.

For example a printed circuit board (PCB) contains several electrical and electronic components. Each component has its own behavior. Similar components repeat several times in the same PCB but with different name, location and connections. They are exhibiting behavior as per the connected components.

Consider simulation of a logic circuit with electronic lids and electronic switches on a PCB. The lids get on and off as per the state of switches and logic circuit output. The dynamic display files enable the geometry of such components to get computed dynamically for each event of the model.

The display file in this model will be filled with display file instructions dynamically. Same display file can be reused for several components decreasing the object and class number in an application. This will enable complex applications to run on light weight systems with low configuration.

The debug driver tool is an application used for testing faulty components of a PCB board. As per the board structure the geometry is automatically computed. The required information is name of the board, board

10.32377/cvrjst0602

type and structure of components on the board and their relations.

IV. GRPHIC APPLICATION USING DYNAMIC DISPLAY FILES

Implementation of graphic frameworks and applications using dynamic display files has several layers. Sample code segments and output screens along with advantages of such models are presented.

As a first step, display file and graphic primitive generation function libraries shown in table 2,3,4 need to be implemented. Function class frameworks are used for reuse and exporting such libraries as third party tools [1].

Graphic frameworks along with some graphic foundation classes are required in the next phase. Client component definition library and graphical user interface are required to build graphic applications.

Figure 3 presents a class diagram of the application developed in Visual studio Net 2010 that uses graphic framework developed using dynamic display file concepts. These concepts are more generic and they can be implemented in any application which supports basic graphic primitives. The class diagram presents MFC document view architecture used for providing GUI for the application. CView, CDocument, CDialog etc are MFC classes. Other classes are graphic framework classes that implement dynamic display file model.

V. CLIENT PROCEDURES FOR CONFIGURING GRAPHIC COMPONENTS

Some of the code segments that explain the implementation of dynamic display file are presented in this section. The domain client defines the component semantics for the generation of display file that compute the geometry required for display. Table 7 presents a sample code for component definition.

This procedure is not a member of CLogicCom class. It is a library function of a logic system for defining logic components. It is invoked from CLogicCom class Design method. This Design method is virtual function invoked whenever there is a need to re-compute the geometry of a component as per its behavior and status in the system. For example, in the LID device display the lights become on or off as per the data available in the system. The display file instructions are generated dynamically through invoking the procedure listed in table 9.

Figure 4 presents Logic Switch and Logic Display as per the user interface and logic circuit output. The green and white represent 1 and 0 (on and off). The red and black represent error and invalid data. The positions 4 and 7 represent error in red as the pins 4 and 8 are carrying invalid data. The first display is shown in green as the gate is AND gate and switches 2 and 3 are representing one. Depending on connections the component changes geometric behavior as per the requirement of simulation like components.

CONCLUSIONS

The traditional display file models and a model graphic framework based on traditional display files are presented. The dynamic display file concept and class diagrams along with sample code segments to demonstrate dynamic display file functioning concepts are presented. Output of a graphic application using dynamic display file concepts is presented. These concepts are applicable to any similar applications irrespective of application domain, object oriented development environment and development language with basic graphic support. The relation subsystem and other inherent concepts used are not covered in this paper.

REFERENCES

[1] Dr.Hari Ramakrishna, “A pattern language and traditional programming practices for exporting functionality” CVR Journal of Science & Technology, released in December 2013 ISSN 2277-3916

[2] Dr.Hari Ramakrishna, ”Pattern Approach to Build Traditional Graphic Frame works”, International 1 Journal of Computer Applications Volume 59– No.15, p35-42, December 2012. Published by Foundation of Computer Science ISSN :(0975 – 8887), New York, USA

[3] Dr. Hari Ramakrishna, “Design Pattern for Graphic/CAD Frameworks”, Ph.D thesis submitted to Faculty of Engineering Osmania University March 2003,

[4] Christopher Alexander, “An Introduction for Objectoriented Design”, A lecture Note at Alexander Personal web site www.patternlanguage.com

[5] Pattern Languages of Program Design. Edited by James O. Coplien and Douglas C. Schmidt. Addison-Wesley, 1995

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns: Elements of Reusable Software Architecture", Addison-Wesley, 1995

[7] Hari RamaKrishna “COM Applications for Real time Electrical Engineering Applications” IEEE sponsored International Conference at Bangalore - 2000.

[8] Hari RamaKrishna “Generation of flooring and wallpaper patterns using computer graphics” Proceedings of the First National Conference on Computer Aided Structural Analysis and Design, Jan 3-5,1996, Engineering Staff College of India and University College of Engineering, Osmania University, Hyderabad.

[9] Newman, W. S and Sproul, R.S (1981),Principles of interactive computer graphics McGraw-Hill International, second edition.

TABLE I
SEGMENT TABLE ATTRIBUTES

(1)	Segment name
(2)	Segment's starting address of the display file
(3)	Segment size i.e. number of instruction of segment
(4)	Segment visibility i.e. on or off
(5)	Segment transformation parameters i.e. scaling, translation, rotation around x,y,z axes
(6)	Segment reference point that is useful for transformations
(7)	Segment transparency (on or off) useful for hidden line and surface elimination Etc.

TABLE II
GRAPHIC SEGMENT TABLE INSTRUCTIONS

(1) Create-segment (n)
(2) close-segment (n)
(3) Append-segment (n)
(4) set-segment-visibility (n,I)
(5) Rotate-segment (n,ax,ay,az)
(6) translate-segment(n,tx,ty,tz)
(7) Set-segment-reference-point (n,x,y,z)
(8) scale-segment(n,sx,sy,sz)
(9) Show-segment (n)
(10) delete-segment (n)

TABLE III
DISPLAY ALGORITHMS

do-line3d (lc,bc,z,y,z),
do-point3d(lc,x,y,z),
do-circle3d(lc, cx,cy,cz,r,ax,ay,az),
doarc3d(lc,cx,cy,cz,r,sa,ea,ax,ay,az),
do-spehere)lc,cx,cy,cz,r) and
do-poly(lc,sadd,size:

where lc is the line foreground color, cz,cy,cz are the coordinates, sa, ea are the starting and the ending angles, ax, ay, az are the angles of inclination along x,y, and z axes respectively, and r is the radius.

TABLE IV
SAMPLE TRADITIONAL DISPLAY FILE USER ROUTINES FOR MANAGING GRAPHS

Move3d (x, y, z)
Line3d(x,y,z)
Line3d(lc,x,y,z)
Point3d(lc,x,y,z)
Arc3d(lc,x,y,z,r,sa,ea,ax,ay,az)
Circle3d(lc,x,y,z,r,ax,ay,az)

TABLE V
SAMPLE COMPONENT CLASS

```
class CLogicCmp :public Component
{
public:  -------
       BOOL   virtual IsLocated(CPoint);
       void virtual Design(void);
       ----------
};
```

TABLE VI
SAMPLE DISPLAY FILE INSTRUCTION ALGORITHM

```
void Component::LineTo(int x,int y)
{
         m_iNoOfInst++;
         DF[1][m_iNoOfInst] = 2;
         iPen_X = x;
         iPen_Y = y;
         DF[2][m_iNoOfInst] = iPen_X;
         DF[3][m_iNoOfInst] = iPen_Y;
}
```

TABLE VII
CLASSES OF DYNAMIC DISPLAY FILE FRAMEWORK

1) CDocument, CView, CLogicSystemView , CLogicsystemDoc are Microsoft based classes in Document view architecture.
2) CLogicCmp is inherited from Component and defines the behavior of domain specific components. In the above application components of a Degug driver tools which are defined. The function in table 6 will be invoked from design method of this class for loading behavior of a electronic LID component.
3) The Component class implement all the Display file procedures as per the definition of IDisplayFile Interface and other implicit procedures required.
4) The Graphic Element class implements all the procedures required for implementation of a graphic framework.
5) IGraphic Component is used to address all the graphic elements from Microsoft Document class. This can implement a generic persistence system which can be used to any graphic domain application using this framework.
6) The view class is inherited from Clogic Component for the purpose of creating a logic component. Even the GUI design can be reused for any domain similar to that of debugger driver tool.

TABLE VIII
COMPONENT CLASS MODEL

```
class Component : public CGraphElement
{
        < <Display file data definition >>
       << Display file implementation >>
<< virtual functions like >>
// For designing component at derived class
         void virtual Design(void);
// for rendering the component INTERPRETER
        void virtual Display(CDC* dc);
……
}
```


10.32377/cvrjst0602

TABLE IX
COMPONENT SEMANTIC DEFINITION FOR A ELECTRONIC DISPLAY LID COMPONENT

```
void VRLogicLID(Component* ge)
{// Component color
        ge->SetLineColor(ge->GetBkColor());
        ge->RectSolidAt(0,0,100,100);
//  Inside Area
        ge->SetLineColor(LIGHTGRAY1);
        ge->RectSolidAt(0,0,96,96);
        ge->SetLineColor(DARKGRAY1);
        ge->RectAt(0,0,94,94);
        ge->RectAt(0,0,96,96);
        ge->SetLineColor(DARKGRAY1);
        ge->RectAt(12,0,24,85);
        ge->RectAt(12,0,18,80);
// Designing light on/off status
int k=1;
for (int i=-35;i<=35;i+=10)
{
  if (ge->GetData(k)==1)
        ge->SetLineColor(RGB(0,255,0));
  else if (ge->GetData(k)==0)
    ge->SetLineColor(RGB(255,255,255));
        else if (ge->GetData(k)==2)
        ge->SetLineColor(RGB(255,0,0));
        else
        ge->SetLineColor(RGB(0,0,0));
// 255,255,255 is white(0 or OFF)
// all zeros black (junk data)
// 255 ,0,0 is red(error in output)
//0,255,0 is green(1 or ON)
  ge->RectSolidAt(12,i,15,8);
  ge->SetLineColor(RGB(0,0,0));
  ge->RectAt(12,i,-15,8);
  k=k+1;
}
ge->SetLineColor(DARKGRAY1);
for(int  i= -35; i<= 35; i+=10)
{
        ge->MoveTo(-45,i);
        ge->LineRel(-15,0);
}// displaying pins of the component
// displaying text of the components
 ge->TextBkColor(LIGHTGRAY1);
 ge->TextColor(RGB(255,0,0));
 ge->TextAt(-40,25);
 ge->Text11At(-40,-25);
} // end of the procedure
```

TABLE X
DESIGN PROCEDURE

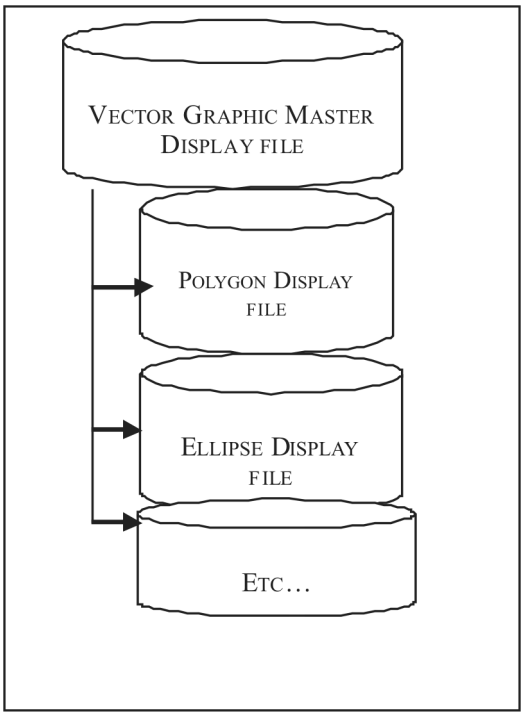
```
void CLogicCmp::Design(void)
{       switch(GetComponentType())
        { case 1: VRLogicLID (this);
                break;  …. }
}
```

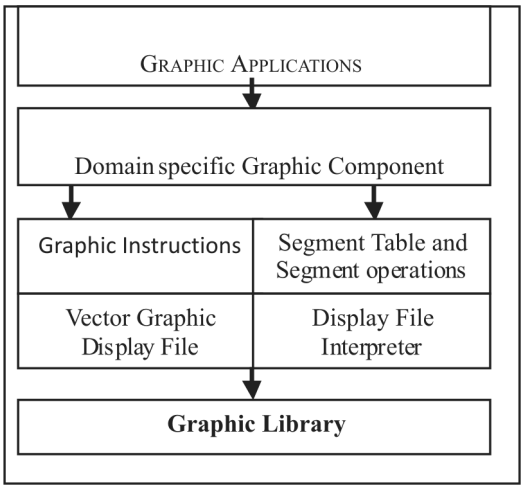
TABLE XI
INTERFACE IDISPLAYFILEINSTRUCTIONS

```
class IDisplayFileInstructions
{
public:
        // Display File Functions
        void virtual  MoveTo(int x,int y) = 0; // 1
        void virtual  LineTo(int x,int y) = 0; // 2
void virtual  TextAt(int x,int y) = 0; // 3  horizontal
void virtual  MoveRel(int x,int y) = 0;  // logical 1
void virtual  LineRel(int x,int y) = 0;  // logical 2
void virtual  TextRel(int x,int y) = 0;  // logical 3

void virtual  VerticalTextAt(int x,int y) = 0; // 4 Vertical
void virtual  VerticalTextRel(int x,int y) = 0; // logical 4
// 5 Rectangle
void virtual  RectAt(int x,int y,int a,int b) = 0;
// 6 used even for circles ellipses
void virtual  ArcAt(int x,int y,int sa,int ea,int r1, r2) = 0;
// 7 Filled Solid Ellise
void virtual  EllipseSolid(int x, y, a, b) = 0;
// 8 Solid Rectangle
void virtual  RectSolidAt(int x,int y,int a,int b) = 0;
// 9 Set color of line
void virtual  SetLineColor(COLORREF i) = 0;
// 10 Set FillColor
void virtual  SetFillColor(COLORREF i) = 0;
void virtual  Text1At(int x,int y) = 0; // 11
void virtual  Text2At(int x,int y) = 0; // 12
// 13 sets Text BkColor
void virtual  TextBkColor(COLORREF r) = 0;
// 14 Set TextColor
void virtual  TextColor(COLORREF col) = 0;
void virtual  Text11At( int x,int y) = 0; // 15 Text
void virtual  Text12At( int x,int y) = 0; // 16 Text
void virtual  Text21At( int x,int y) = 0; // 17 Text
void virtual  Text22At( int x,int y) = 0; // 18 Text

};
```

TABLE XII
GRAPHIC COMPONENT INTERFACE

```
class IGraphicComponent  :public
IDisplayFile, public IGraphicElement
{

 << Graphic Element interface contains
Graphic Framework user interface
Procedures >>
<< I Display file has
IDisplayFileInstructions and other related
interfaces >>

};
```





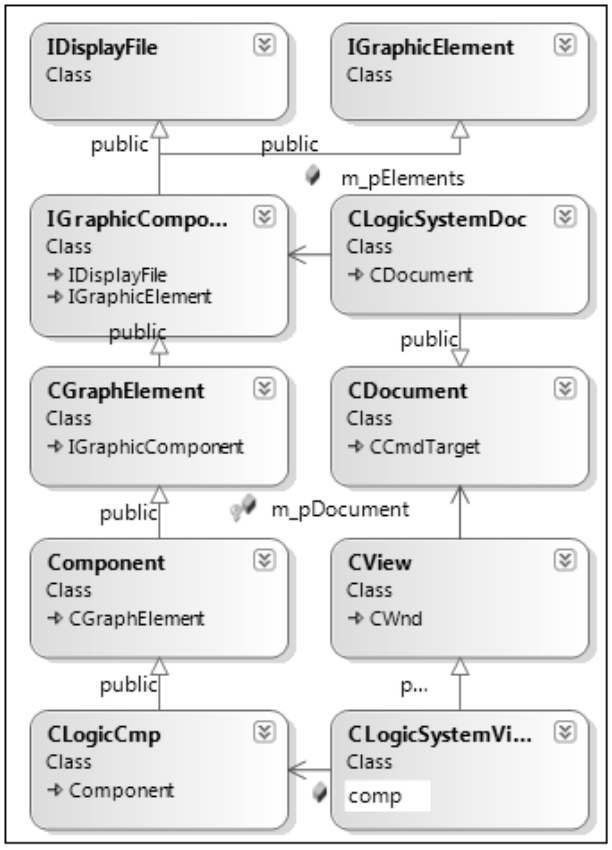


Figure: 3 Class diagram of a graphic application in Visual studio .NET 2010 using dynamic display files

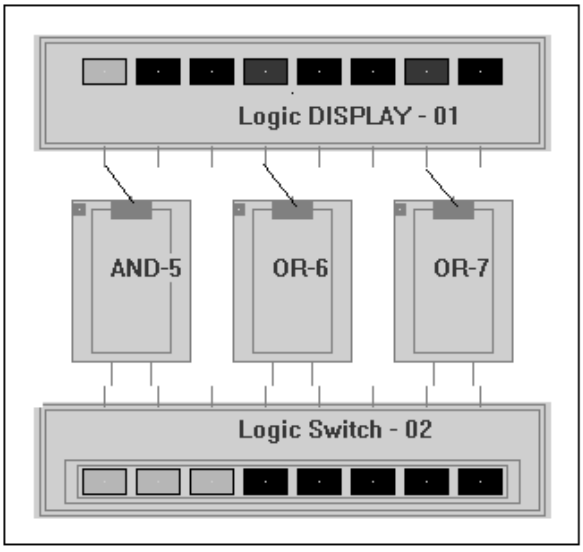


Figure: 4 A logic circuit designed with and OR gate, AND gate and logic switch components on a PCB.

10.32377/cvrjst0602