Component based Frameworks for exporting Graphic functionality through automation layer

Dr. Hari Ramakrishna,
Department of Computer Science and Engineering, C.B.I.T, Hyderabad, India
Email: dr.hariramakrishna@rediffmail.com

***Abstract*— The Component technology though complex to implement has several advanced features in development of graphic frameworks. Pattern Frames are evolved to solve such complexities. This paper uses Microsoft COM Technology for providing solution. These principles enable common developer to use advanced technology for providing solutions using complex technology in a simple way and providing patterns to common client to use these in simple processes. It presents procedure of building automation process. As Microsoft changes procedures reorienting structures, these processes work in all versions though they are demonstrated in Visual Studio 6.0. Results obtained and code segments are also presented. Abstract pattern frame, Wrapper and other pattern frames and Helper object are presented.**

***Index Terms*— Pattern- Frames, Automation Layer, IDL ODL script, Helper Object, Wrapper, Display files, Semantic graphic behavior, graphic components, graphic frameworks, debug driver tool, Microsoft graphic applications, object oriented models and frameworks. Dynamic display files. Component Object Model COM, Macros, Wizards.**

I. INTRODUCTION

The software Industry is adopting new procedures and technologies for rapid development of software. The requirements of the industry and client are also changing rapidly. Though industry started with business and information processing applications, mainly for railways and other industries, now the industry's concentration is on total simulation of real time environment, knowledge extraction, decision making, rapid application development using frameworks, making developer independent of development environments, using reuse techniques and patterns and frameworks.

In this connection several frameworks are developed in computer graphics, computer based design CAD, Geographical Information system GIS, Industrial Plant designed system and several military systems [8,9,10,11].The Intergraph Solid Edge, Imaginer, Smart Plant, Geological systems (GeoMedia) and several graphic frameworks have been developed in Microsoft Technology. They are exhibiting advanced technical features using Microsoft Technologies, namely COM Technology starting from Visual studio 2.0 from 1994 onwards.

COM technology has many advantages compared with Object Oriented Technology. But its implementation is very complex. Companies like Microsoft have provided several Wizards and frameworks stating from Visual Studio Versions 2.0, 4.0, 5.0, 6.0 and 7.0, and subsequently .NET versions incorporating more and more technical features. The COM Framework of Microsoft implements almost all Design Patterns.

II. OBJECT ORIENTED TECHNOLOGY VERSUS COMPONENT TECHNOLOGY

There are several unsolved problems in the present Object Oriented Technologies. A simple Object Oriented Technology cannot provide solutions to such problems. Typical problems one faces with the present object oriented technology are discussed below. [1-7,12]

1. In cases where new objects are formed inheriting from more than one base class, if two base classes have the same function, the inherited class cannot resolve to which base class it has to map that function.
2. The object hierarchy is too complex for the developer. For example, sometimes the MFC object hierarchy is too confusing to the developer. A typical complex object hierarchy is shown in Fig. 1 .
3. The encapsulation of objects is not perfect. The object details are not encapsulated. The inherited object should know everything about the base class along with its full hierarchy for using it. We are giving extra information to the user in this technology. Giving more information than necessary to the user is against abstraction and information hiding principles and is not advisable in any technology.
4. One more problem is that the objects do not have a common root and the C++ object hierarchies form a disjoint set of trees.
5. We have no solution to handle an unknown objet problem. That means, we cannot hold or perform even a minimum set of operations on an object for which information is unknown.
6. It is difficult to expose the behavior across process boundary

It is difficult to divide any task into independent modules. Tightly coupled modules will enforce restrictions on extendibility of the modules. If modules are loosely coupled, they can be designed and extended independently, without affecting other modules. There are several such problems, for which there is no solution in direct Object Oriented Technology. Today's Component Object Model is providing solutions to such problems. The COM technology is also an object-oriented technology. The COM object is also a C++ object, but the way it manages the data and behavior is entirely different. It adopts several

design patterns to solve above problems. Microsoft provides several frameworks to implement these solutions.

Though it provides solutions to many of the problems, it is too costly to manage. The COM object is not a single object; it is a set of objects. A set of interfaces it supports represents the behavior of a COM object. The COM object encapsulation is perfect. It hides entire object including the object source, object hierarchy and object name. Without knowing even the name of the object, one uses the COM object.

The modules in COM based applications are loosely coupled. They communicate through a set of interfaces, which the components of the module support. The components are assembled with a set of interfaces. COM does not support inheritance. The COM aggregation of components will do the job of inheritance in a more effective way. Aggregating existing components forms new components.

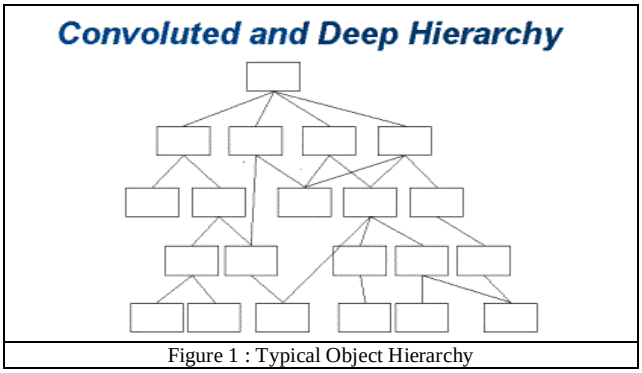


Figure 1 : Typical Object Hierarchy

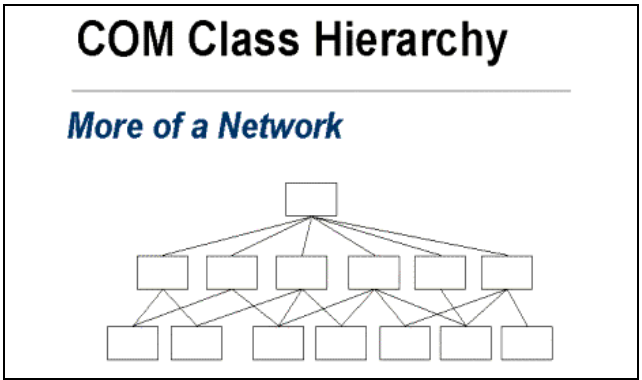


Figure 2 : 2 COM Hierarchy

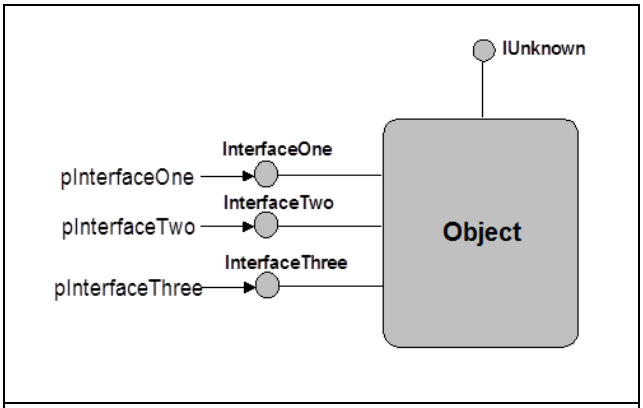


Figure 3 COM Object Model

The Hierarchy of Component Technology is simple. Fig. 2 presents a typical COM object hierarchy. Unlike C++ objects, all COM objects belong to the same family. The COM Hierarchy is a simple tree of depth two. All the COM objects and their interfaces are inherited from the common interface known as IUnknown interface. A Typical COM object is presented in the Fig. 3.

III. FEATURES OF COMPONENT TECHNOLOGY

The COM Concept of Ownership:The COM technology introduces the concept of the ownership subsystem. Each component has an owner. The owner of a component is also a component. A component may be owner of more than one component, but each component has only one owner. Any COM object without an owner is not treated as a full-fledged component. Such components cannot participate in aggregation and other activities of the software environment under consideration. A few special types of components with a specific purpose are without the owner. A class factory object of a COM object has no owner as it is given a special purpose of implementing a design pattern, namely 'Class Factory', which is a creational pattern, the intent of which is to create components in a uniform way through system registry without class name, server location and the type in which component it is implemented.

The owner of a component has full control over the component. A pointer to IUnknown interface of the owner component is stored along with the COM object. A component can become owner of itself but the owner IUnknown interface should not be null.

Fig. 4 presents two COM objects with owners. The first COM object holds its own IUnknown interface. In this case, it is owner of itself. The second COM object has an aggregated COM object in it. The owner of the aggregated COM object is the outer COM object. The outer object also should have an owner. In this case the outer object is the owner of itself. When we query an interface on any component, it will pass the call to the outer object, which is the owner of the object. The outer object also passes the call again to its owner. Finally, the call will reach the owner itself. When we query on the IUnknown interface of internal interfaces, the call will never pass the outer-object. If IUnknown is passed, the system will collapse as ownership gets collapsed, spoiling internal integrity and security.

The IUnknown interface alone can make the object unloaded, by making the interface count zero. That is why, the IUnknown is known as the controlling interface. These concepts will play an important role in the COM aggregation. The IUnknown interface is hidden within the outer-object. This is not given to the other components or procedures. This is as powerful as a pointer to the object.

COM object is a set of assembled components, but the user views it as a single object. Entire internal management is encapsulated through these concepts. Loading, managing

 DOI:10.32377/cvrjst0701

and unloading the inside components are the jobs of the outer object.

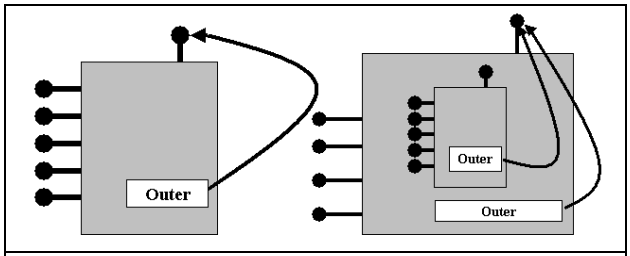


Figure :4 COM Aggregation with Coupling Components with IUnknown Interfaces

TABLE I
IUnknown Interface format

```
class IUnknown
{
  protected:
        ULONG      m_cRef;
  public:
    STDMETHODIMP QueryInterface(REFIID, LPVOID FAR*);
    STDMETHODIMP_(ULONG)        AddRef(void);
    STDMETHODIMP_(ULONG)        Release(void);
}
```

The definition of IUnknown interface is presented in Table 1.

i) The m_cRef means reference count to represent number of objects using the components. Once the count is zero, the object gets deleted automatically.
ii) The AddReff increases the reference count when the new client objects start using the object and decreases the count when it is released.
iii) When the object under the service leaves the component, automatically the count will be decreased; otherwise memory leak and runtime error crash the system. These are major problems in Component based systems.
iv) Query Interface of any Interface can ask other Interfaces from the Component for further use with the Interface ID. Depending on the permission, it releases in a systematic way unlike simple objects for which the number of clients is not known.

The COM Aggregation:

The Components cannot be inherited; as they are windows objects and they are registered objects. The name and location of the header file and server is not known to the client. The COM technology reuses the component using a special technique known as aggregation. The COM aggregation is managed in several ways depending up on the requirement. The way of managing the aggregated object will decides the type of aggregation.

i) Fig. 5 presents a typical COM aggregation. In this, the outer object delegates all the queries to all the aggregated objects in a specified order without seeing the queried interface. This is known as perfect delegation.
ii) Fig. 6 presents another type of COM aggregation. The Line object aggregates two point components. The Line query interface function will decide to which object it has to pass the query.
iii) Fig. 7 presents the COM technique of interface containment. The COM technology uses containment for overwriting the behavior. The outer object will re-implement the interface, which is available in the aggregated object.

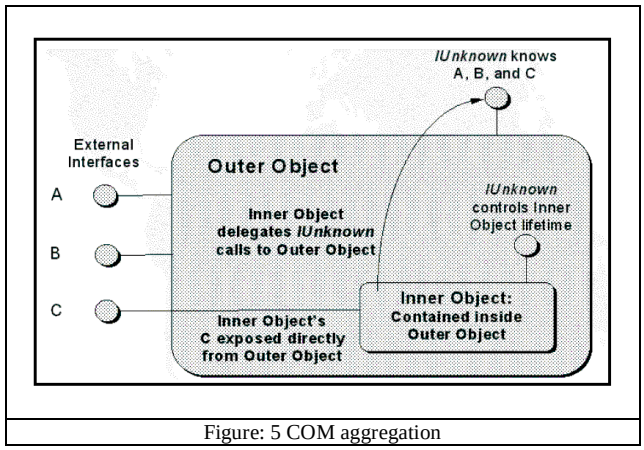


Figure: 5 COM aggregation

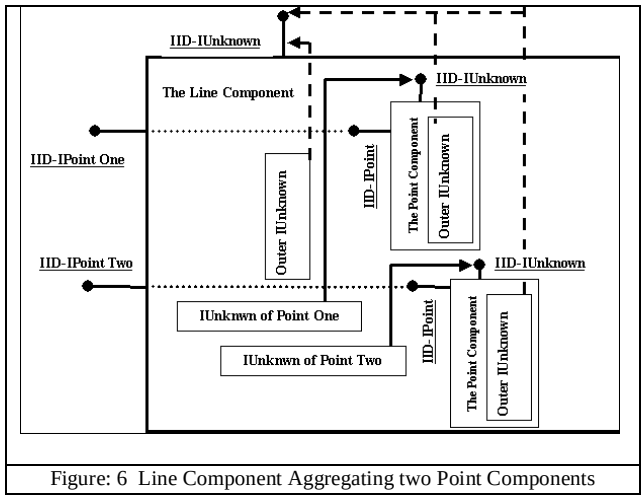


Figure: 6 Line Component Aggregating two Point Components

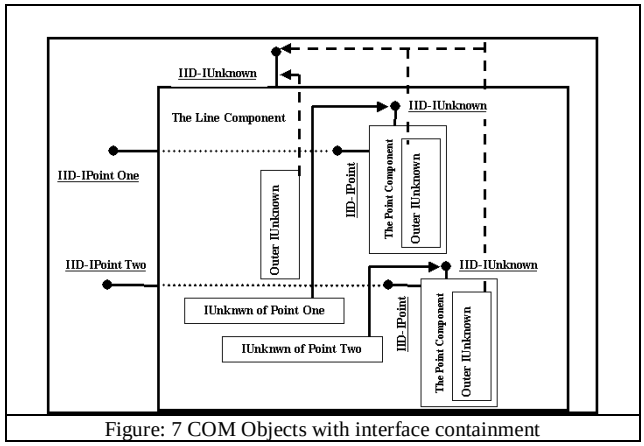


Figure: 7 COM Objects with interface containment

IV. FEATURES OF COMPONENT DEVELOPMENT PROCEDURES AND CONCETS

The component development process is complex compard with simple OOP based C++ application, but it has lots of features that solve several client requirements. The OLE features like cut and past, in place activation, drag and drop, container and server, and several modern requirements are possible only with this technology. As a consequence, often IT industry is forced to adopt these procedures. The steps required to build a simple Beeper object are as follows:

i) The beeper in a direct application needs a single class and a single application to give a beep sound for the given client input signal. In Component model several project and directories are required in a particular hierarchy; the files need to be shared across several projects. Sample files SRC (source files), INCLUDE (for header files), project files, Registration files for registry entry, INTERFACE files for interfaces, BIN for application's executable files (EXE) and DLL for server dynamic link libraries components.

i) The component needs a GUID globally unique ID generated by a special application of Microsoft, namely guidge.exe which generates a unique ID which will be unique across the world. An ID generated once will never get generated again so that Components and interfaces are referred with that ID. If two GUIDs match in any system totally, Windows under use will crash as shown in Table 2.

ii) IDL and ODL files with Interface Description language and Object Description language scripts are used to build automation layer so that functionality is exported to VB application. Compilation of this script gives header file to C++ and type library to VB layer. This exports C++ functionality to VB application.

iii) Microsoft provides various wizards for generating all these facilities automatically for unknown clients.

TABLE II
SAMPLE GUID FILE

```
#ifndef _H_GUID
        #define _H_GUID 1
        #include <windows.h>
        #include <objbase.h>

// {765BFF32-C207-11d0-BC7B-080036603003}
DEFINE_GUID(CLSID_CBeeper, 0x765bff32, 0xc207, 0x11d0,
0xbc, 0x7b, 0x8, 0x0, 0x36, 0x60, 0x30, 0x3);
#endif // _H_GUID
```

iv) The sample procedure to crate a beaper object throught system registry and class fatory presented in Table 3.

TABLE III
COM OBJECT CREATION CODE

```
STDMETHODIMP CBeeperClassFactory::CreateInstance
(LPUNKNOWN punkOuter,

    REFIID riid, LPVOID FAR *ppvObj)
    {
        CBeeper*        pObj;
        HRESULT                 hr;

        *ppvObj = NULL;
        hr=ResultFromScode(E_OUTOFMEMORY);

    if (NULL != punkOuter && !IsEqualIID(riid, IID_IUnknown))
        return ResultFromScode(E_NOINTERFACE);
        pObj = new CBeeper(punkOuter);
        if (NULL == pObj)   return hr;
        if (pObj->Initialize())
            hr = pObj->QueryInterface(riid, ppvObj);
        if (FAILED(hr))     delete pObj;
        else  ServerIncrementNumberOfObjects();
            return hr;
        }

STDMETHODIMP_(ULONG) CBeeper::AddRef(void)
{       return ++m_cRef;    }

STDMETHODIMP_(ULONG) CBeeper::Release(void)
ULONG   cRefT;
        cRefT = --m_cRef;
        if (!m_cRef) {
            delete this;
            ServerDecrementNumberOfObjects();
        }       return cRefT;
        }
```

VI. COMPONENT BASED GRAPHIC FRAMEWORKS

The Component Technology builds an application environment to provide techical services to client requirenents and providing resources without concern to either application domain or funtionality. It provides a communication environment that provides a session to enable client and server to do busness. The same procedures provided in this will be applicable in several requirements and applicaton domains.

The major components of any project environment are client and server. Different types of servers in use are 'In process servers'. They support and provide services through DDLs, local servers, and work in different processes but on same system. The local server works on the same network connected systems to distribute and share services to enable different systems of the same network to share services. An example of this is the DCOM of Microsoft.

New frame works provide several advanced technological features of communicating over Internet based protocol, for examples Simple Object Access Protocols (SOAP), Web services, mobile communications etc. All these Technologies can work like wrappers. The code and traditional technologies still work. Several Wizards are available to the developer for using all these techniques. These Technologies and wizards are little costly and complex. Developer will never get total control

 DOI:10.32377/cvrjst0701

over internal procedures though they decrease the development time and cost of development.

Several layers and code models of development of a Graphic framework and code segments are presented in this section without functional models. The functional design concepts are presented in [1]. Without using code, COM enables all these object oriented graphic frameworks to be exported to VB layer or other web based or mobile based technologies through wizards. But user will never get total control over the requirements for configuration other than using them.

i) The framework is a layered development. All the basic functions required should be developed as libraries, classes and generic object oriented frameworks [1,3]. For the domain considered, dynamic display files are used to simulate several graphic systems like Logic circuits, Printed circuit boards, Debugger driver tools using dynamic display files, shown in Table 4 show sample application oriented functional components [1].

TABLE IV(a)
DISPLAY ALGORITHMS LIBRARIES

```
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:

etc…
```

TABLE:IV(b)
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 IV(c) :
COMPONENT SEMANTIC DEFINITION FOR AN 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);
        ….
// 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 IV(d) :
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  RectAt(int x,int y,int a,int b) = 0;
// 14 Set TextColor
void virtual  TextColor(COLORREF col) = 0;
void virtual  Text11At( int x,int y) = 0; // 15 Text
        ---    -------  -------   ---------
};
```

ii) The COM needs to define interfaces and COM environment to export these components over application environment to the client through proper channel as per permissions and requirements, as shown in the following sample code segments in Table 5.

TABLE V:
TYPICAL GRAPHIC INTERFACE

```
#undef INTERFACE
#define INTERFACE IGPersist
 DECLARE_INTERFACE_(IGPersist, IUnknown)
 {
 STDMETHOD(QueryInterface) (THIS_  REFIID riid, LPVOID FAR
*ppvObj) PURE;
 STDMETHOD_(ULONG, AddRef)(THIS) PURE;
 STDMETHOD_(ULONG, Release)(THIS) PURE;
STDMETHOD(Serialize)(THIS_ CArchive &ar) PURE;
STDMETHOD_(CLSID,GetClsid) (THIS) PURE;
 };
 typedef IGPersist FAR* LPIGPersist;
```

iii) The class and Interface ID models are presented in table 6

TABLE VI
THE CLASS INTERFACE GUIDS AND REGISTRATION FILE MODEL

```
REGEDIT

HKEY_CLASSES_ROOT\CLine = CLine Object
```

```
HKEY_CLASSES_ROOT\CLine\Clsid = {0D8BAFE2-33C8-11d1-A0B3-0060974FF0B9}

HKEY_CLASSES_ROOT\Clsid\{0D8BAFE2-33C8-11d1-A0B3-0060974FF0B9} = CLine Object

HKEY_CLASSES_ROOT\Clsid\{0D8BAFE2-33C8-11d1-A0B3-0060974FF0B9}\INPROCSERVER32 = H:\COm_PHD\HCOM\COM-CODE\GPCOM1\bin\GCOM.dll

HKEY_CLASSES_ROOT\CComp = CComp Object

HKEY_CLASSES_ROOT\CComp\Clsid = {A445E8CA-216C-11d6-B98C-204C4F4F5020}

HKEY_CLASSES_ROOT\Clsid\{A445E8CA-216C-11d6-B98C-204C4F4F5020} = CComp Object

HKEY_CLASSES_ROOT\Clsid\{A445E8CA-216C-11d6-B98C-204C4F4F5020}\INPROCSERVER32 = H:\COm_PHD\HCOM\COM-CODE\GPCOM1\bin\GCOM.dll
```

vi) Pattern Frames to mke COM procedures simple

The development procedure of adding components from the frame work need to follow entire COM procedure. For this purpose, the following Abstract COM pattern framework is useful. [4]. Fig. 6 shows a block diagram of abstract component which enables the developer to use COM component like a simple C++ object without complex procedures and with Component features.

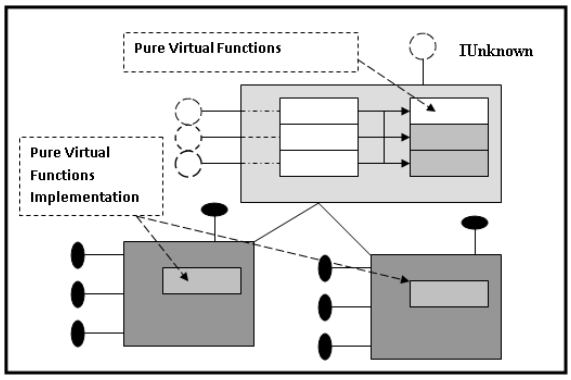


Figure 6 : a model block diagram of abstrat component

The VC client can use the framework like a simple object oriented framework using wrapper object as shown in Fig. 7.

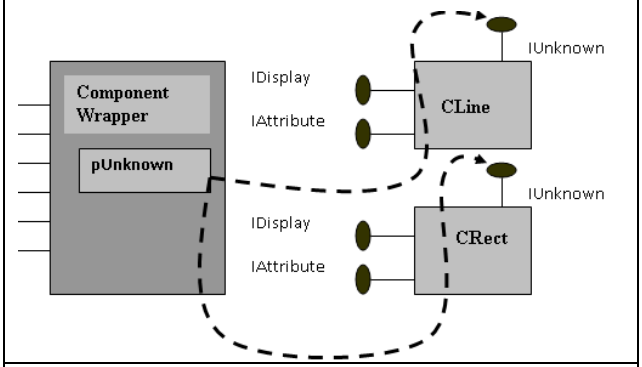


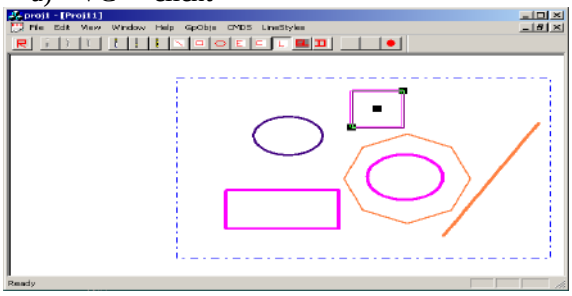
Figure : 7 Block diagram of Wrapper pattern frame

TABLE VII
MODEL WRAPPER DEFINITION

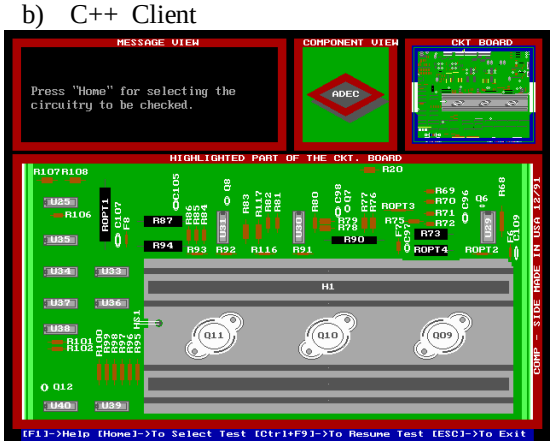
```
#ifndef HGraphicElement
#define HGraphicElement 0
#include <objbase.h>
#include "IGraph.h"
#include "Guid.h"
#ifdef HGraphicSERVER

  class __declspec( dllexport ) HGraphic
#else
   class __declspec( dllimport ) HGraphic
#endif
{

private:
      LPUNKNOWN    m_IUnknown;
      LPIGPersist        m_IPersist;
      LPIGAttributes    m_IAttributes;
      LPIGDisplay       m_IDisplay;
      LPIGEdit           m_IEdit;
      LPIGLocate        m_ILocate;

public:
         HGraphic(CLSID);
       ~HGraphic();

      HRESULT Serialize(CArchive &ar);
      CLSID   GetClsid(void);
      HRESULT SetPoints(ULONG,ULONG,ULONG,ULONG);
      HRESULT GetColor(void);
      HRESULT SetName(CString);
      CString GetName(void);
      HRESULT  GetName(CString* );
  ----  ------
      }
```

The Visual Basic can extract services from automation layer through wizards and ODL and IDL file outputs.

a) VC++ client

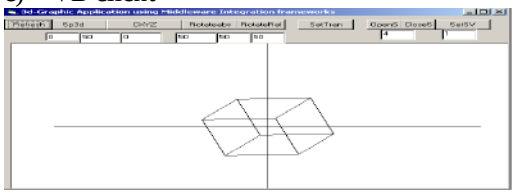


b) C++ Client



DOI:10.32377/cvrjst0701

c) VB client



d) Semantic logic application in VC++ in dot net 2010

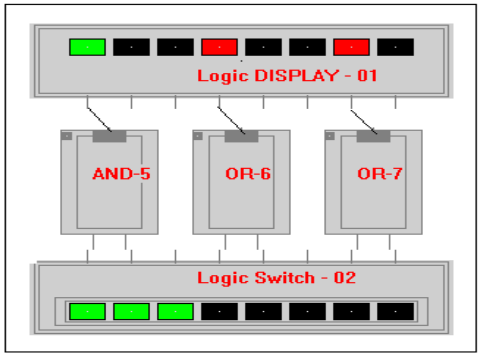


Figure 8 : Graphic application outputs of the framework

A simple visual base code to use server through wrapper objects and wizards is presented in Table 8.

TABLE VIII
SAMPLE VB CLIENT CODE SEGMENT

```
Private Sub Command1_Click ()
 HGP3D1.Sp3d Text1.Text, Text2.Text, Text3.Text, Text4.Text
End Sub

Private Sub Command10_Click ()
HGP3D1.S1
End Sub
4r
Private Sub Command11_Click()
HGP3D1.S2
End Sub

Private Sub Command4_Click()
HGP3D1.RoteteSegmentAbs Text7.Text, Text4.Text, Text5.Text, Text6.Text
 HGP3D1.ShowAll
End Sub
```

CONCLUSIONS

The component based frameworks, their processes, advantages, problems and features, various processes to make implementation simple, sample graphic application code segments, and outputs of several sample applications have been presented. Details of models are described in some of the papers in references, as indicated at various places in the paper. It is suggested that same procedures can be applied to other domains for the development of the framework for the other domain environment not concerned with domain or application requirements. The application presented can be treated as a model for demonstration of environment. Several wizards are available to pack the Object Oriented Frameworks. They can be used for exporting into new technical environments, but clients will not get total control over configuring the requirements and use totally Component features. The user can use only assigned or permitted services. So implementing core Component features as per layers is advised. Using and developing new pattern frames to make use of Component Technology is also advised.

REFERENCES

[1] Dr.Hari Ramakrishna, “Managing semantic of graphic components through remodeling traditional display files” , Journal of Science & Technology Journal, Volume VII, June 2014 ISSN 2277-3916

[2] 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

[3] 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

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

[5] Christopher Alexander, “An Introduction for Object-oriented Design”, A lecture Note at Alexander Personal web site www.patternlanguage.com

[6] Hari: Dec 2000 Hari RamaKrishna “COM as new Object Oriented Technology”, Procedings of CSI conference, December 2002 at Visakapatnam .

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

[8] Hari Ramaakrishna, “COM based CAD” Proceedings of International Conference on xxx, July 2000 at Jaipure, India

[9] Hari RamaKrishna “Application of computer graphics in interior design” Proceedings of Conference 1998 at Institutes of Engineers At Hyderabad.

[10] 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

[11] Hari RamaKrishna, “Object Oriented Graphic Frameworks”, International Journal of Engineering Research & Technology, Vol.2 - Issue 1 (January - 2013) e-ISSN: 2278-0181 This work is licensed under a Creative Commons Attribution 4.0 International License.

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