.NET & VBA Interoperability In MicroStation v8

by Stanislav Sumbera, Berit group, Czech Republic

Introduction

MicroStation v8 brings to developers among others two practical features:

- Integration of Microsoft's Visual Basic for Application.
- Complete possibility to compile MDL code into native DLL and use Visual Studio for debugging.

On Bentley newsgroup someone says that the letter "L" in MDL now means Library not Language. Third time lucky, Microsoft has recently released final version of .NET framework and its SDK which defines a new platform for software solutions.

This platform we my immediately use for MicroStation development as will be shown in the article. The key feature of each new technology is often backward compatibility or interoperability with the current code. We will focus in this contribution to these features – mutual interoperability among different platforms accessible in MicroStation.

Interoperability overview

Figure 1 shows that the central point of intercommunication is a Dynamic Link Library (DLL) which is capable to interact with different platforms. This could be one of the good reason why to compile MDL code into DLL – just to be in the "centre".

The upper right part of the picture was already discussed in article:"Java/Jmdl communication with MDL application" (MicroStation Manager 12/2001). We will give our attention to the VBA and .NET interoperability.

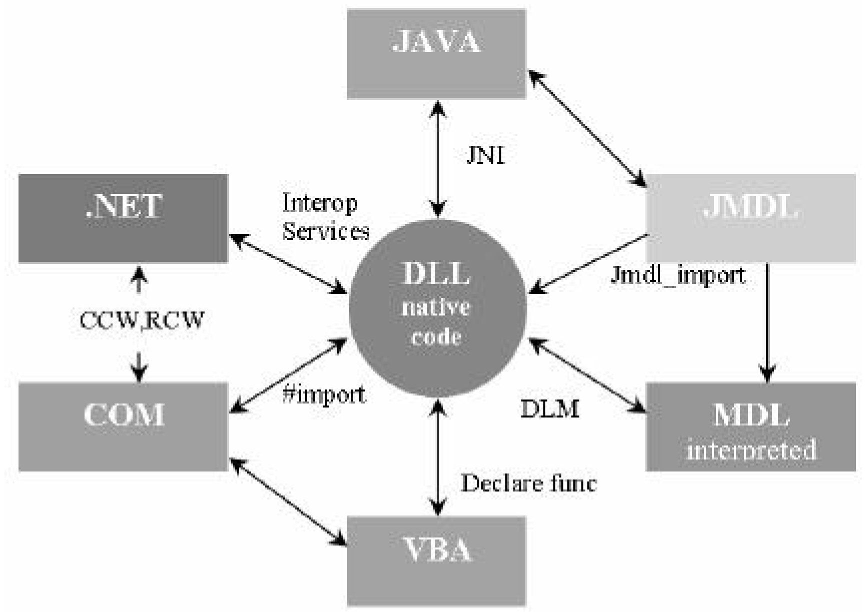

Figure 1 In-process interoperability schema in MicroStation

Designing DLL native code for interoperability

We will go through code which demonstrate interoperability to point out some interesting techniques. The native code handles up coming calls from VBA and .NET and invokes their procedures and methods.

The interoperability is designed through direct function pointers, thus make it more simpler for understanding. The dll library is called **nativeLib.dll** and is referred inVBA and .NET code below. The following steps to interact with the DLL are common to VBA and .NET:

1. Calling exported function in any DLL is almost trivial – you only need to declare function as exported and set proper calling convention to stdcall (VBA require this convention)

```
#define DLLEXPORT __declspec(dllexport)
#define STDCALL __stdcall
void DLLEXPORT STDCALL Dllfoo(int params){}
```

2. Calling procedure or method in VBA or .NET requires to declare function prototype with its parameters, obtain its pointer and then it is possible to call it :

```
// a) declare function with one parameter :
typedef int (STDCALL *METHODPTR)(int limit);

//global variable to hold pointer to the method
METHODPTR MethodQ= NULL;

// b) function to set method pointer (will be called from VBA or .NET)
int DLLEXPORT STDCALL dll_setMethodPointer (void (*MethodPtr) (void)){
  return MethodQ = (METHODPTR) MethodPtr;

// c) function to invoke method stored in MethodQ variable
int DLLEXPORT STDCALL dll_callMethod (short limit){
  return (MethodQ ? MethodQ(limit): FALSE);
}
```

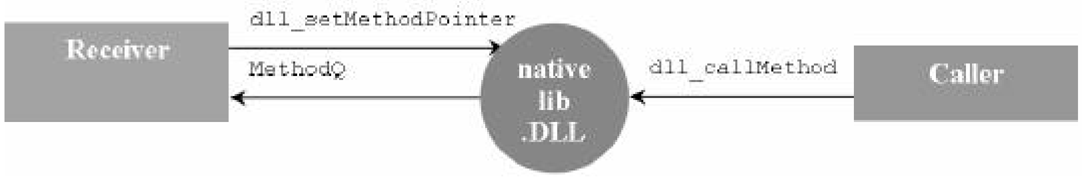

Figure 2 *Incomming and outgoing functions in DLL*

There are different functions for VBA, .NET, MDL and Java in nativelib.dll to enable calling between them.

Designing VBA to call DLL code

VBA can invoke any standard DLL function in very simple and straightforward way. This possibility allow to call Window's API or MDL built in functions (from version 8.00.02). For instance, to call MDL function for opening dialog box you need to declare at global section of VBA code imported function from **stdmdlbltin.dll,** then it is possible to call it:

```
Private Declare Function mdlDialog_openInfoBox_
Lib "stdmdlbltin" (ByVal prompt As String) As Long

Private Sub UserForm_Activate()
  mdlDialog_openInfoBox ("called from VBA")
End Sub
```

Designing .NET to call DLL code

.NET allows to call native (unmanaged) functions that are implemented in a DLL via Platform Invocation Services (PInvoke). The called function in DLL need not to be declared as standard :

```
public class DLLWrap{
  [DllImport("ustation.dll", CallingConvention=CallingConvention.Cdecl)]
  public static extern int mdlDialog_openInfoBox(String message);
}
private void Button_Click(object sender, System.EventArgs e){
  DLLWrap.mdlDialog_openInfoBox("message from .NET");
}
```

Designing VBA to accept calls from DLL

DLL is able to call VBA procedures through its pointer (see above). We need to set the proper procedure pointer first with help of **AddresOf** operator and than call DLL function to store the pointer for later invocation of VBA method :

```
'Define a function to be called from DLL in VBA module:
Public Sub OnVBACall(ByVal limit As Long)
  VBANonModal.Label1 = "VBA got " + Str$(limit)
End Sub

' Declare function from DLL which accept pointer to the VBA method:
Private Declare Function dll_setVBAMethodPointer_
  Lib "nativeLib.dll" (ByVal pMethod As Any) As Long

' Call the DLL function to set pointer on method OnVBACall
Private Sub UserForm_Activate()
  Call dll_setVBAMethodPointer(AddressOf OnVBACall)
End Sub
```

Designing .NET to accept calls from DLL

The .NET Framework defines a special type called Delegate that provides the functionality of a type-safe function pointer. We need to design method to be called from DLL, declare and call the native DLL function to set method pointer:

```
//declare signature of called method through a delegate
public delegate void OnNETCallDelegate(int limit);
public class DLLWrap{
   // declare member onNetCallRef to hold method reference
   public static OnNETCallDelegate onNetCallRef =
          new OnNETCallDelegate (DLLWrap.OnNETCall);

   // declare member to refer to Form (dialog box)
   public static NetForm netFormRef;
   [DllImport("nativeLib.dll")]
   public static extern int
          dll_setNETMethodCall(OnNETCallDelegate method);

   // receiving method of DLL call
   public static void OnNETCall(int limit){
            netFormRef.setLimitToNetForm(limit);
   }
}
// a class of Form
public class NetForm : System.Windows.Forms.Form{
  public NetForm(){
                  // set reference to this form
                  DLLWrap.netFormRef = this;

                  // set method to accept dll calls
                  DLLWrap.dll_setNETMethodCall(DLLWrap.onNetCallRef);
          }
}
```

Wrapping .NET into COM

.NET Framework provides COM callable Wrapper (CCW) enabling an arbitrary COM client to seamlessly call a method on a .NET object. .NET object appears to COM clients just as if it were a native COM object. This feature fits perfectly to the VBA concept which provide easy way to call COM methods and manage COM events. CCW makes COM object from .NET just as simple as possible:

```
// declare delegate for events
public delegate void onTransmitDelegateVba(int limit);

// define interfaces InterfaceType identifies
// how to expose an interface to COM
 [InterfaceType(ComInterfaceType.InterfaceIsDual)]
 public interface ITransmit{
             [DispId(1)]
             int setLimitToNetForm(int limit);
       }

// interface to sink events by COM clients
[InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)]
       public interface ITransmitEvent{
             [DispId(1)]
             void OnTransmitEventVba(int limit);
       }

// define class inherited from interface
[ComSourceInterfaces(typeof(ITransmitEvent))]
[ClassInterface(ClassInterfaceType.AutoDual)]
 public class NetForm : System.Windows.Forms.Form,ITransmit{
       // declare event
       public event onTransmitDelegateVba OnTransmitEventVba;
       // implement interface
       public int setLimitToNetForm (int limit){
       this.LabelRec.Text= ".NET Form got " + limit.ToString();
       return limit;
       }
       // raising event handled by the COM sink
       private void ButtonCallVba_Click
              (object sender, System.EventArgs e){
            OnTransmitEventVba(Convert.ToInt16(this.TextLimitVBA.Text));
   }
}
```

VBA client would use this object as ordinary COM:

```
'declare events and NetForm object
Dim WithEvents NetEvent As NetForm
Dim dotNetForm As New NETobject.NetForm

'show form, set event handle to COM object
Private Sub UserForm_Activate()
  dotNetForm.Show
  Set NetEvent = dotNetForm
End Sub

'call .NET method when button is pressed
Private Sub Button_OnNet_Click()
  dotNetForm.setLimitToNetForm (Val(NETLimit.Text))
End Sub

Public Sub NetEvent_OnTransmitEventVba(ByVal limit As Long)
   VBANonModal.Label1 = "VBA got from .NET " + Str$(limit)
End Sub
```

Accessing MicroStationDGN object from .NET

Calling COM object is possible via callable wrapper (RCW). RCW serves as proxy which exposes COM objects for .NET Framework, thus .NET clients may call any COM client. Thus we may deploy whole MicroStation DGN object into .NET application.

A .NET client (event sink) can receive events raised by an existing COM server (event source). COM interop generates the necessary delegates in metadata that you include in your managed client. An imported delegate signature comprises the sink event interface, an underscore, the event name, and the word EventHandler: *SinkEventInterface_EventNameEventHandler*.

```
MicroStationDGN.Application msApps = new MicroStationDGN.Application();

// set COM property
msApps.Caption = " Caption set by .NET";
MicroStationDGN.View msView;
msView = msApps.ActiveDesignFile.Views[1];

// do zoom on view 1
msView.Zoom(2);
msView.Redraw();

// wire event handler for OnDesignFileClosed event
MicroStationDGN.__ApplicationEvents_OnDesignFileClosedEventHandler
DesignEventClose = new
MicroStationDGN.__ApplicationEvents_OnDesignFileClosedEventHandler
(OnDesignClose);
msApps.OnDesignFileClosed += DesignEventClose;

// event handler
static void OnDesignClose(String designName){
MessageBox.Show(".NET recieved File Close event + designName");
}
```

Final words

It has been shown how to interoperate among DLL, .NET and VBA code. Moreover, you may download complete code for this article to test interoperability. In a code for this article there is actually more done. There are dialog boxes created in MDL, Java, VBA and .NET which serve as a test for application interaction, see Figure 3. In VBA code, some tricks are used to allow VBA and .NET Forms to be displayed behind ordinary MDL dialog boxes and behave as child window of MicroStation.

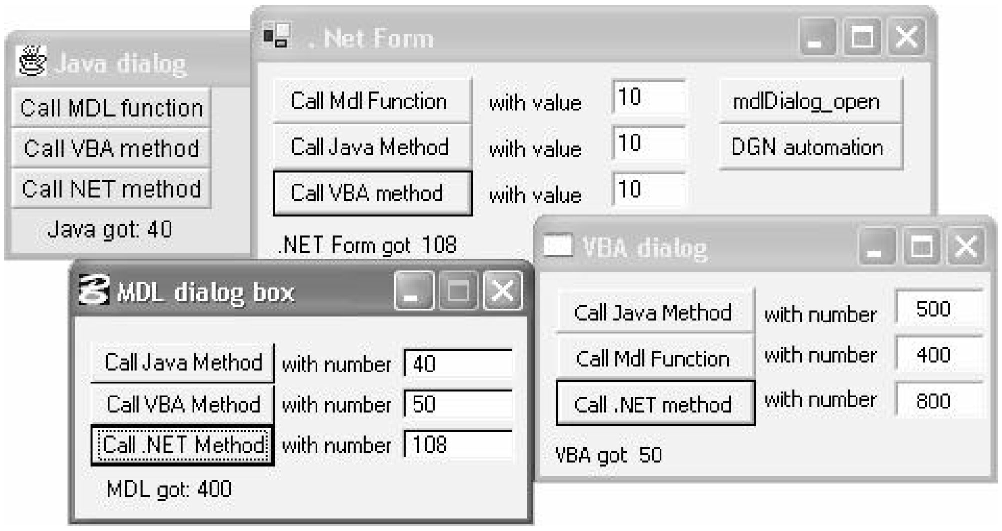

Figure 3 *Dialog boxes for interaction between MDL, Java, VBA and .NET*

We can use all good features of particular language for our solution. Most promising is to use .NET together with MDL compiled into native code. Moving MDL code into native DLL would save one step in interaction between different platforms and MDL.. NET platform is no doubt ready for MicroStation software solution. We may use it even instead of VBA with possibilities to write the application in C#, VB.NET, VC.NET or other. These examples were compiled using Java 2 SE, VBA v.6, VC++.NET 7 , C#.NET 7, and MicroStation V8 (08.00.02.20) compilers runing on Windows XP Pro.

About The Author

Stanislav Sumbera, Ph.D. is a MicroStation software developer for the Berit Group ***(www.berit.com)****. The programming code in this article is offered to the MicroStation community, as is“, without any responsibility for its use elsewhere. You may contact Stanislav Sumbera at* ***ssumbera@berit.cz****.*

CAD