TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics and Mechanics
53, Vol. III, 2010

Programming AutoCAD using Jawin from Java in JDeveloper

ANTAL Tiberiu Alexandru

Abstract: *The paper gives a free solution that can be used to program AutoCAD from Java based on the "Jawin - a Java/Win32 interoperability project" and using the JDeveloper programming environment. The main data types and methods used to create AutoCAD entities from Java are described using examples together with the steps to open such a project under JDeveloper.*
Key words: *Java, AutoCAD, Ole32, JDeveloper.*

1. INTRODUCTION

AutoCAD is an application that can be easily programmed using the VB or VBA programming languages based on the ActiveX technology. The idea of this article is to program AutoCAD from Java by exploiting the compatibilities of the ActiveX object model. At the Autodesk's Developer Center, the Questions and Answers section [3] is written that "Autodesk does not directly support application development using Delphi or Microsoft J++ language tools. However, the AutoCAD ActiveX® object model should be compatible, and if Java or Delphi is your development environment choice, you can use it". Jawin [4] is a free integration project, open source architecture for interoperation between Java and components exposed through Microsoft's Component Object Model (COM) or through Win32 Dynamic Link Libraries (DLLs). This project can be used with success to connect Java and AutoCAD with the help of JDeveloper, the Oracle's Java development tool. This is a Java application that contains code that will use code written in another language, which is called usually native code. To make possible the calling of the native code Java is linked to the system libraries. The usage of native code has however the price of portability loss. The Java Native Interface (JNI) was used from Java 1.0 as a standard way to access native code from the Java platform. JNI provides a set of low-level primitives that can be used to pass arguments from Java into native code, and vice versa. During the development process JNI requires the programmer to write custom native code each time new capabilities must be added. The main advantage of Jawin is that all the native code is in Jawin.dll and no additional native code needs to be written. This way the programmer can concentrate on his personal development problems instead of implementing custom native code. Also, Jawin is already structured and the programmer doesn't need advanced JNI and COM knowledge in order to use it in his applications. In short, we have higher productivity with less knowledge.

2. INSTALLATION OF Jawin AND JDeveloper CONFIGURATON

At [4] the "Binary and source releases" link will be used to download the Jawin files. The jawin-2.0-alpha1.zip compressed file is organized on directories; the latest build is 2005-03-23 and requires JDK 1.3 or newer. Decompressing the file will create the jawin-2.0-alpha1 directory that will contain most of the files needed to start the development. In JDeveloper create a new application, then a new project and then, from the *Project Properties*, at *Libraries and Classpath*, add

(using the *Add JAR/Directory…* button) the contents of the *lib* directory (*jawin.jar* and *jawin-stubs.jar*) to the project (see Figure 1).

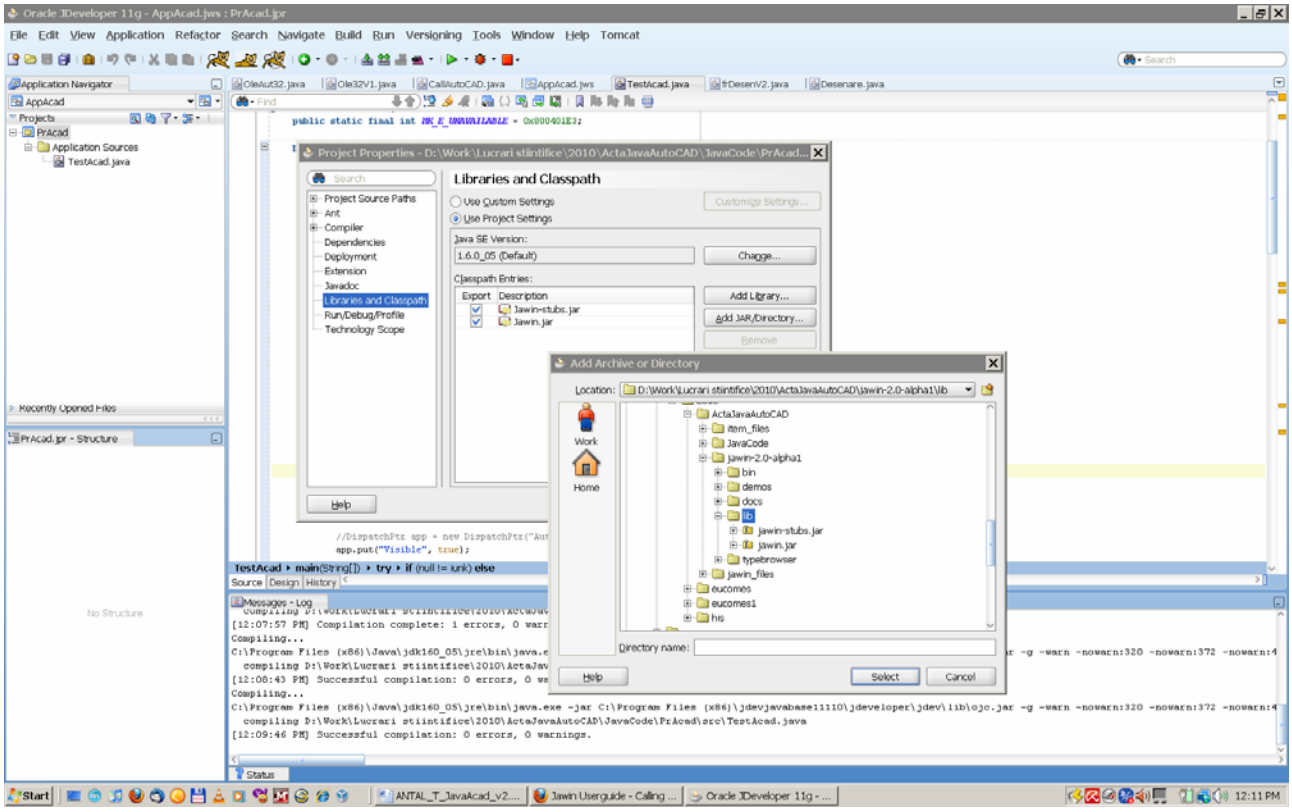

Fig. 1. Configuration of the JDeveloper project to work with Jawin.

3. CONNECTING FROM Java TO AutoCAD USING Jawin

The link with Java is based on the ActiveX (COM automation) interface implemented in the AutoCAD software. This link can be made by instantiating a new AutoCAD session or by connecting to an already opened AutoCAD instance session.

The following Java code will start a new AutoCAD instance, if no such instance running or it will link to an existing AutoCAD instance:

```
import org.jawin.COMException;
import org.jawin.DispatchPtr;
import org.jawin.IUnknown;
import org.jawin.Variant;
import org.jawin.win32.Ole32;

try {
  Ole32.CoInitialize();
  IUnknown iunk = null;
  try {
    iunk                                =
OleAut32.GetActiveObject("AutoCAD.Appl
ication.17");
      } catch (COMException ex) {
    if       (MK_E_UNAVAILABLE       !=
ex.hresult) {
          throw ex;
    } //end if
  } // end catch
  DispatchPtr app = null;

  // did we get a running instance?
  if (null != iunk) {
    // retrieve the dispatch interface
    app                                 =
(DispatchPtr)iunk.queryInterface
(DispatchPtr.class);
  } else {
     // create a new instance
     app              =              new
DispatchPtr("AutoCAD.Application.17");
    } //end else

 . . .

Ole32.CoUninitialize();
}     catch     (Exception     e)     {
  e.printStackTrace();
}
```

Because COM requires that all threads calling a COM object must initialize the COM library before making any COM calls, this is

done by calling `Ole32.CoInitialize()`. After a thread is finished with all COM calls, it should call `Ole32.CoUninitialize()`. The `DispatchPtr` is created directly, using the `ProgId` for the COM class you want to use (the name `DispatchPtr` comes from the fact that scriptable COM components implement the `IDispatch` interface). In order to gain access to AutoCAD from the thread we are in a `COMPtr` called `DispatchPtr` is used. This will also allow access to any subclass (methods and accessors) of the AutoCAD object model hierarchy. At the end of the application must call the `close()` method, which will release the underlying `IUnknown*`. If we want to find an opened AutoCAD instance and get a link to it things are a bit more complicated. I will only explain the process. First we need the source code `Ole32.java`, here we have to make a small adjustment and make the `CLSIDFromProgID()` called from `GetActiveObject()` a public method. I decided to keep the original `Ole32.java` and make a `Ole32V1.java` where I make the necessary adjustment. The code is not specific to AutoCAD and one implementation could be:

```
public static IUnknown GetActiveObject(String progID) throws COMException {
  try {
  // CLSIDFromProgID in orig. version of Ole32.java is private and must be public)
    GUID clsid = Ole32V1.CLSIDFromProgID(progID);
    Ole32V1.CoCreateInstance(clsid, Ole32V1.CLSCTX_ALL, WellKnownGUIDs.IID_IUnknown);

    NakedByteStream nbs = new NakedByteStream();
    LittleEndianOutputStream baos = new LittleEndianOutputStream(nbs);
    clsid.marshal(baos, null);
    byte[] result = funcGetActiveObject.invoke(instrGetActiveObject, stackSizeGetActiveObject, nbs, null, ReturnFlags.CHECK_HRESULT);

    return (IUnknown)IdentityManager.getCOMPtr(result, 0, WellKnownGUIDs.IID_IUnknown);
        } catch (IOException ioe) {
    throw new COMException(ioe);
    }
  }
```

Then, we can check if an AutoCAD instance is running (the `ProgID` is in this case `"AutoCAD.Application.17"`) by calling the `GetActiveObject()` method. For both cases the `app` variable of `DispatchPtr` type will hold a reference to the instance. This `app` variable will be used further to interact with AutoCAD.

4. PROGRAMMING AutoCAD FROM Java USING Jawin

In order see the effects of the Java code while programming the `put()` method is used to make AutoCAD window visible on the screen. `put()` will be used to set any property of AutoCAD and has the following syntax:

```
public void put(String prop, int val) throws COMException;
```

Then, the `get()` method is used to find the model space (`ms`) of the current document (`doc`) in order to start the programming of the drawing process.

```
app.put("Visible", true);
DispatchPtr doc = (DispatchPtr)app.get("ActiveDocument");
DispatchPtr ms = (DispatchPtr)doc.get("ModelSpace");
```

`get()` is a getter method that allows reading of the AutoCAD properties having the syntax:

```
public Object get(String prop) throws COMException;
```

The process of drawing an AutoCAD entity has two steps, first we must prepare the data, and then we must invoke the desired method. In the following example a line will be added to the model space. The `AddLine()` method is used from VBA to create a line in the model space. Two arrays must be used to specify the start point and the end point. The start point declaration array would look in VBA like `Dim sp(0 To 2) As Double` while in Java the equivalent code is: `double[] sp = new double[3];`

This however is not enough to pass the parameter to an AutoCAD method as the representation of data in Java and VBA are not the same. The Jawin `org.jawin.Variant.ByrefHolder` class (the name originates from the term passing by reference instead of the usual by value) has a `ByrefHolder` wrapping object that must be used, as in the following code, in order to succeed:

```
double[] sp = new double[3];
double[] ep = new double[3];

//start point
sp[0] = sp[1] = sp[2] = 0;

//end point
ep[0] = 1;
ep[1] = ep[2] = 3;

//create a 2D line
Variant.ByrefHolder     p1     =     new
Variant.ByrefHolder(sp);
Variant.ByrefHolder     p2     =     new
Variant.ByrefHolder(ep);
DispatchPtr           linie           =
(DispatchPtr)ms.invoke("AddLine",   p1,
p2);

//create a circle
double raza = 0.5;
Variant.ByrefHolder     r     =     new
Variant.ByrefHolder(raza);
DispatchPtr           cerc           =
(DispatchPtr)ms.invoke("AddCircle",
p2, r);
```

Two `ByrefHolder` objects, `p1` and `p2` are created based on the `sp` and `ep` Java array variables. The `ms` object is used to invoke DLL functions with the help of the `invoke()` methods. The use of `ByrefHolder` is compulsory even if the Java variable types are primitives. This case is shown for a circle, where the radius is a just a `double` primitive but still has to be passed as a `ByrefHolder` object.

One of the common programming tasks is the calling of AutoCAD commands directly from Java. We could solve the same problem with the help of the special API methods, still some things will be written faster this way. The following example will use `SendCommand` from the COM document object in order to call the *regen* and *zoom* AutoCAD commands.

```
//calling AutoCAD commands
doc.invoke("SendCommand", "_regen ");
doc.invoke("SendCommand", "_zoom e ");
```

The following example (see Figure 2) is going to create a `3DPolyline` in two ways. In the first case the points are known and stored into an array and in the second case the points are computed based on an arithmetic expression (spiral equations).

```
//3DPoly – first case
//4 points, 3 coordinates per point
double[] pl = new double[12];

//first point
pl[0] = 0; //x
pl[1] = 0; //y
pl[2] = 0; //z

//second point
pl[3] = 1;
pl[4] = 1;
pl[5] = 10;

//third point
pl[6] = 2;
pl[7] = 3;
pl[8] = 4;

//fourth point
pl[9] = 7;
pl[10] = 6;
pl[11] = -5;

//prepare the data
Variant.ByrefHolder    plp    =    new
Variant.ByrefHolder(pl);
//and draw it
DispatchPtr           pline           =
(DispatchPtr)ms.invoke("Add3DPoly",
plp);

//3DPoly – second case
//i - counts the point
//n – number of the points
//3*n because each point
// needs 3 coordinates

int i=0, n = (int)(2*Math.PI/0.1)+1;
double[] spiral = new double[3*n];

for(double t=0;t<2*Math.PI;t+=0.1) {
  spiral[i]=Math.cos(6*t);
  spiral[i+1]=Math.sin(6*t);
  spiral[i+2] =t;
  i+=3;
}
```

```
Variant.ByrefHolder pspiral = new
Variant.ByrefHolder(spiral);
DispatchPtr pline1 =
(DispatchPtr)ms.invoke("Add3DPoly",
pspiral);
```

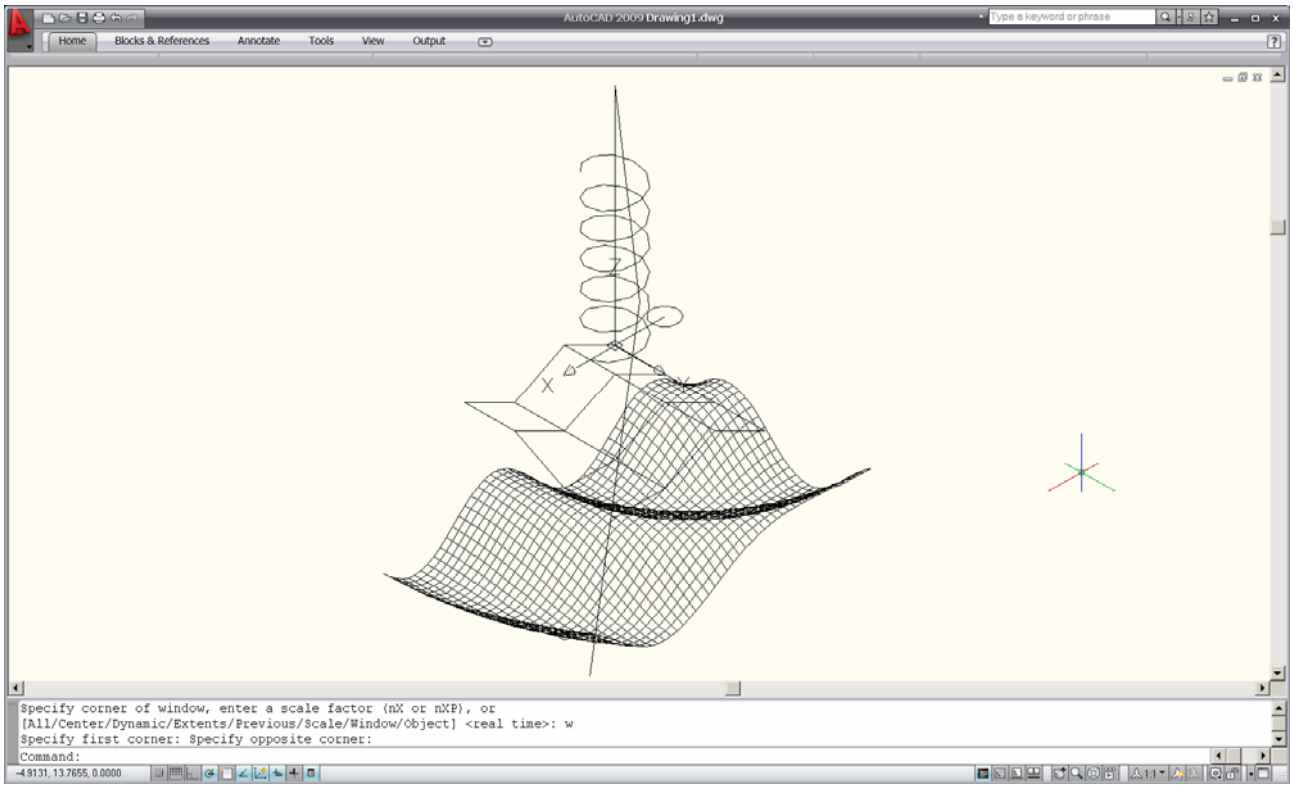

Fig. 2. The AutoCAD entities created from the Java based on Jawin.

The following example (see Figure 2) is going to create a `3DMesh` in two ways. In the first case the points are known and stored into an array, while in the second case the points are computed based on an arithmetic expression. The (x, y, z) coordinates for each point of the `3DMesh` are stored in three consecutive vector elements (`mesh[c]`, `mesh[c+1]`, `mesh[c+2]`). The `x0=0`, `y0=3` are just an offset that will "delay" in space the entities so they would not overlap. Two `for` cycles are used to compute each coordinate of the points from the `3DMesh` based on the $z = \sin(\sqrt{x^2 + y^2})$ expression.

```
//3Dmesh – first case
//16 points, 3 coordinates per point

double[] p3m = { 0, 0, 0, 2, 0, 1, 4,
0, 0, 6, 0, 1, 0, 2, 0, 2, 2, 1, 4, 2,
0, 6, 2, 1, 0, 4, 0, 2, 4, 1, 4, 4, 0,
6, 4, 0, 0, 6, 0, 2, 6, 1, 4, 6, 0, 6,
6, 0 };

Variant.ByrefHolder p3dm = new
Variant.ByrefHolder(p3m);
DispatchPtr d3mesh =
(DispatchPtr)ms.invoke("Add3DMesh", 4,
4, p3dm);

//3Dmesh – second case
//u*v points
//z=sin(sqrt(x^2+y^2)

int u=50, v=30, c=0;
double[] mesh = new double[3*u*v];
double x, y, x0=0, y0=3;
for (i=0; i < u; ++i) {
  x=0.25*i;
  for (int j=0;j<v;++j) {
    y=0.25*j;
    mesh[c]=x0+x;
    mesh[c+1]=y0+y;

mesh[c+2]=Math.sin(Math.sqrt(x*x+y*y))
;
    c+=3;
  }
}

Variant.ByrefHolder vecmesh = new
Variant.ByrefHolder(mesh);
```

```
DispatchPtr obvecmesh =
(DispatchPtr)ms.invoke("Add3DMesh", u,
v, vecmesh);
```

Another common programming task is reading the characteristics of some entities from AutoCAD. The following code is reading the start point (`lsp`) and the end point (`lep`) of the line drawn before. Then, the application is stopped and the user can switch to AutoCAD where it can modify the start point of the line. The new coordinates will be then printed along with the initial ones.

```
//read the line start (lsp) and end
(lep) points
double[] lep = (double[])
linie.get("EndPoint");
double[] lsp = (double[])
linie.get("StartPoint");

//print the values
System.out.println("Line Start
point:("+lsp[0] + ", " + lsp[1]+ ", "
+ lsp[2]+")");
System.out.println("Line End
point:("+lep[0] + ", " + lep[1]+ ", "
+ lep[2]+")");

//prepare to stop the application
int indata;
Scanner in = new Scanner(System.in);

// Reads an integer from the console
and stores into the indata variable
//to stop the application
indata = in.nextInt();
in.close();

//update the changes
linie.invoke("Update");

//read the changes
double[] newlsp =
(double[])linie.get("StartPoint");
System.out.println("Line Start
point:("+newlsp[0] + ", " + newlsp[1]+
", " + newlsp[2]+")");
```

5. CONCLUSIONS

The Jawin allows programming of AutoCAD from Java. The programmer must have decent knowledge about AutoCAD and the ActiveX technologies and good programming skills under Java.

6. REFERENCES

[1] Antal Tiberiu Alexandru, “Visual BASIC pentru ingineri”, RISOPRINT, Cluj-Napoca, 2003, p. 244, ISBN: 973-656-514-9.

[2] Tiuca, T., Precup, T., Antal, T. A., “Dezvoltarea aplicatiilor cu AutoCAD si AutoLISP”, Ed. PROMEDIA, 1995, p. 303, ISBN: 973-96862-2-2.

[3] http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=770204, Accessed at 4/9/2010 11:57 AM.

[4] http://jawinproject.sourceforge.net/, Accessed at 4/9/2010 12:02 PM.

[5] http://www.koders.com/java/fidA79D342745C88031CF506FC3CAA6550E8A9A787B.aspx?s=%22Stuart+Halloway%22#L22, Accessed at 4/10/2010 12:56 PM.

PROGRAMAREA AutoCAD-ului DIN Java FOLOSIND Jawin UTILIZAND MEDIUL JDeveloper

Lucrarea prezinta o solutie gratuita care se poate utiliza pentru programarea AutoCAD-ului, din limbajul Java, ce are la baza proiectul de interoperabilite “Jawin”. Prin exemple se arata structurile de date si metodele ce se utilizeaza pentru crearea entitatilor AutoCAD impreuna cu etapele pornirii unui astfel de proiect de sub mediul de programare JDeveloper.

ANTAL Tiberiu Alexandru, Associate Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanics and Computer Programming, antaljr@bavaria.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.