Under Construction:
Delphi 4 And Java, Part 2

by Bob Swart and Hubert Klein Ikkink

Two months ago we explored CORBA, and I claimed that CORBA was a good platform- and language-independent way of communication. Last time, we used CGI and sockets as means of communicating between Delphi 4 and Java (JBuilder 2) applications. This time, we will bring this quest to a conclusion by using CORBA as a communciation protocol between a JBuilder Java applet and a Delphi server-side application.

Recap

Last time, we implemented a 'to do list' application that we could use to manage a list of things to do. The user interface consisted of a Java applet, the server-side application was written in Delphi. First, we experimented with a plain CGI solution, then followed this up with a sockets approach.

We will now focus solely on CORBA as communication protocol, which means transforming the Delphi server into a CORBA server implementing CORBA methods, and the Java applet into a CORBA client, calling these methods. In theory, using JMIDAS for JBuilder 2, we can also decide to use DCOM (MIDAS) instead of CORBA, but we won't go into that here.

In all three cases, the Delphi server-side application would use the unit `IniMod` to store the to do list in a standard Windows .INI file (BobNotes.ini), as implemented in Listing 1.

Note that, compared with last month, we have changed the implementation of the `SetLines` function. We now pass the password, `Passw`, as the third argument to the `ReadString` method. As a consequence, the password will be correct if no such `User` exists, in which case that particular user (and password, and to do list) will be created. This simple change now enables us to create new users.

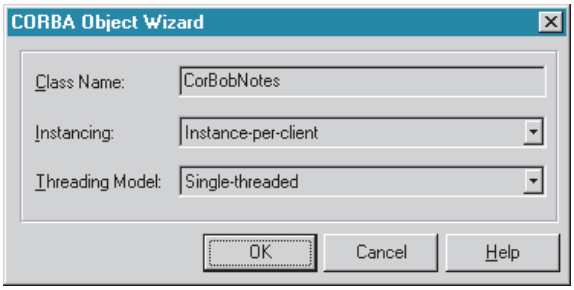
CORBA

The CORBA server implements two methods: one to get the to do list (with the `User` and `Password` as input arguments), and a second to set the entire to do list (`User`, `Password` and `ToDo` as input arguments). Although we now have the ability to create a new user (with a new password), we still have no option to change the password for a specific user. This can be done manually on the web server itself by the webmaster.

Start a new project, and add a CORBA object from the `Multi-Tier` tab of the `Object Repository`. Specify `CorBobNotes` as the name of the CORBA object, and make sure to use the defaults for instancing (`Instance-per-client`) and threading model (`Single-threaded`), as shown in Figure 1.

The instancing option specifies how the CORBA server application creates instances of the CORBA object. We can either specify instance-per-client, meaning a new

➤ *Figure 1: CORBA Object Wizard.*



➤ *Listing 1: IniMod 'storage' Unit.*

```
unit IniMod;
interface
uses
  SysUtils, Classes;
type
  ELoginFailed = class(Exception);
  procedure GetLines(const User, Passw: String;
    Lines: TStrings); // raises ELoginFailed
  procedure SetLines(const User,Passw: String;
    Lines: TStrings); // raises ELoginFailed
implementation
uses
  IniFiles;
var
  IniFile: TIniFile = nil;
procedure GetLines(const User,Passw: String; Lines:
  TStrings);
var
  i: Integer;
begin
  if (User <> '') and
    (IniFile.ReadString(User,'password','') = Passw)
    then begin
    Lines.Clear;
    for i:=1 to IniFile.ReadInteger(User,'lines',0) do
      Lines.Add(IniFile.ReadString(User,IntToStr(i),''))
  end else
    raise ELoginFailed.Create('Login failed.')
end {GetLines};
procedure SetLines(
  const User,Passw: String; Lines: TStrings);
var
  i: Integer;
begin
  if (User <> '') and (IniFile.ReadString(User, 'password',
    Passw) = Passw) then begin
    IniFile.EraseSection(User);
    { reset password }
    IniFile.WriteString(User,'password',Passw);
    IniFile.WriteInteger(User,'lines',Lines.Count);
    { linescount }
    for i:=1 to Lines.Count do
      IniFile.WriteString(User,IntToStr(i),Lines[Pred(i)])
  end else
    raise ELoginFailed.Create('Permission denied.')
end {SetLines};
initialization
  IniFile := TIniFile.Create('.\BobNotes.ini');
finalization
  IniFile.Free;
  IniFile := nil
end.
```

object is created for each client connected (until the connection is closed), or shared-instance, meaning one single instance of the CORBA object handles all client requests. One unique instance-per-client is our choice here.

Threading specifies how the client requests call our CORBA object interface. This can be set to either single-threaded, where each CORBA object gets only one client request at a time, or multi-threaded, where each client request gets its own thread. To avoid thread conflicts, I have specified single-threaded here (again the default), which means that the CORBA object instance data is safe, but I should still protect global memory from potential thread conflicts. With multi-threading we would also have to protect instance data itself.

Directly after we've created this CORBA object, we should save the entire project. Specify `Unit1` for the main form and `Unit2` for the CORBA object, with `BobNotes` as the project name. Now, go to `Unit2` and start the Type Library (using the `View | Type Library` menu option in Delphi) in order to add the two CORBA methods `GetLines` and `SetLines`.

Since we need to define two methods that will be callable by a Java application, we must be a little careful with the argument types. There is no Delphi `String` type in the dropdown list of argument types, but I do see `PChar`, which looks safe enough at the first glance. So, I've made all the arguments (`User`, `Password` and `Lines`) of type `PChar`. We'll handle conversion details later, let's first concern ourselves with the Delphi to CORBA to Java communication issues.

```
typelib BobNotes
  [ uuid '{883E4362-E5AC-11D2-92D0-0080C7C19BE0}',
    version 1.0,
    helpstring 'Project1 Library' ];
  uses STDOLE2.TLB, STDVCL40.DLL;
  ICorBobNotes = interface(IDispatch)
    [ uuid '{883E4363-E5AC-11D2-92D0-0080C7C19BE0}',
      version 1.0,
      helpstring 'Dispatch interface for CorBobNotes Object',
      dual,
      oleautomation ]
    procedure GetLines(none User, Password: PChar; out Lines: PChar)
      [dispid $00000001]; safecall;
    procedure SetLines(none User, Password, Lines: PChar)
      [dispid $00000002]; safecall;
  end;
  CorBobNotes = coclass(ICorBobNotes [default] )
    [ uuid '{883E4365-E5AC-11D2-92D0-0080C7C19BE0}',
      version 1.0,
      helpstring 'CorBobNotes Object' ];
end.
```

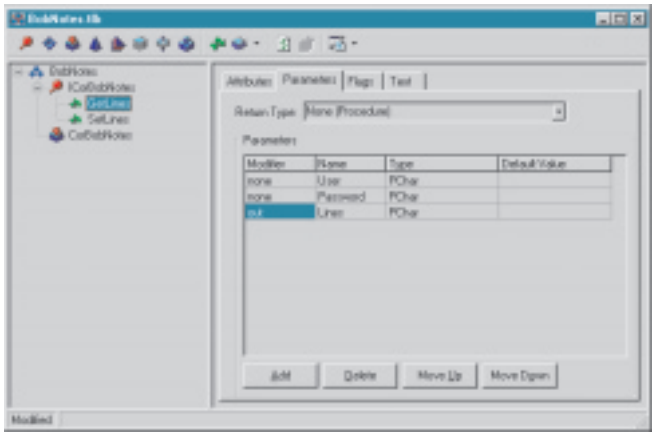
➤ *Listing 2: Delphi Pascal IDL for BobNotes.*

```
module BobNotes
{
  interface ICorBobNotes;
  interface ICorBobNotes
  {
    void GetLines(in string User, in string Password, out string Lines);
    void SetLines(in string User, in string Password, in string Lines);
  };
  interface CorBobNotesFactory
  {
    ICorBobNotes CreateInstance(in string InstanceName);
  };
};
```

➤ *Listing 3: CORBA IDL file for BobNotes (using PChar arguments).*

The Delphi 4 Pascal IDL file that gets generated for the `GetLines` and `SetLines` methods according to the type library definition shown in Figure 2 (using `PChars`) is shown in Listing 2.

Listing 2 doesn't contain a real IDL file, but the Delphi Pascal 'dialect' of an IDL file (that is, easier to read for Delphi users, but not readable by other CORBA development environments). In order to let JBuilder read the IDL for the `CorBobNotes` CORBA Server, we must use the type library again, but this time click on the upper right arrow button (`Export to CORBA IDL`) and export to a real CORBA IDL file that can be imported by JBuilder. This IDL file, which can be seen in Listing 3, translates the `PChar` type into a standard IDL `string` type, which is translated by JBuilder into a standard `String` type again.



➤ *Figure 2: Type Library using PChar arguments.*

When the JBuilder applet calls the `GetLines` method, the Java `String` arguments are marshalled into IDL `string` arguments, which are passed to the Delphi CORBA server, and then un-marshalled into Delphi `PChar` arguments. Unfortunately, the last step in this connection doesn't work correctly. It appears that the three `PChar` arguments point to the same location and, what's worse, when we return something (using `GetLines`), the JBuilder CORBA client isn't able to use the result: instead it produces an exception.

Of course, using a Delphi CORBA client that calls the `GetLines` method using `PChar` arguments works just fine, because marshalling from `PChar` to an IDL `string` and un-marshalling it back to a `PChar` again yields the same result. But a `PChar` can't be transformed into a Java `string`, or so it seems. The reason why I show this here is because CORBA claims to be independent of both platform and

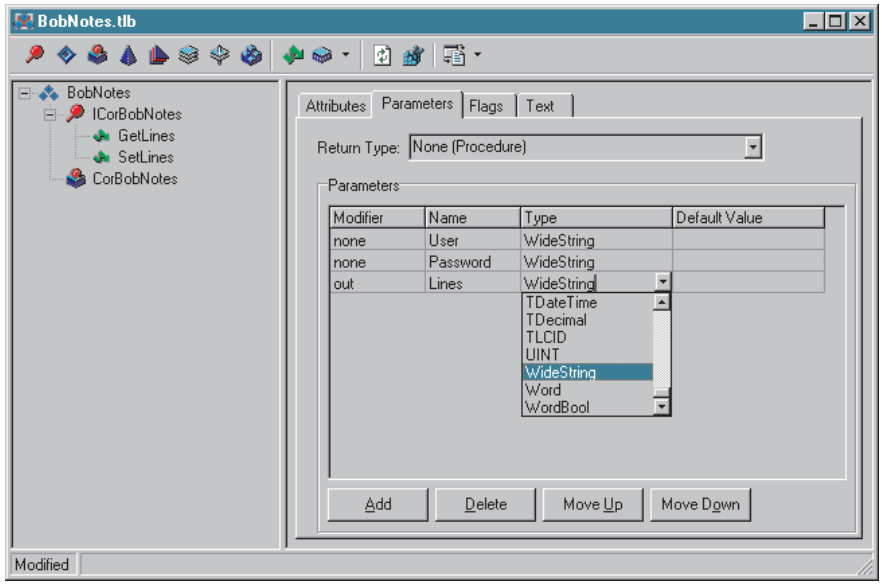
language, but the weakness in this scheme is the marshalling and un-marshalling algorithm, which must be able to perform the transformation between platforms and, in this case, languages without problems.

WideString

Back to the drawing board, or the type library in this case. Let's see what other types we might choose from for our arguments.

Like I said, there are no regular `string` types, but there is a `WideString` type (`wstring` IDL type) that we can use. So, just change the three argument types of `GetLines` and `SetLines` to `WideString`, and refresh the implementation (click the button with the two green arrows). This time, the IDL file will use `wstring` types, and I can tell you that it will work with JBuilder 2, as you'll see in a little while.

First, let's implement the `GetLines` and `SetLines` routines, using `WideString` type parameters. The CORBA server implementation of `TCorBobNotes` can be seen in Listing 4. Note that we're using the `Memo1` component on `Form1` (in `Unit1`) as a debug window for our CORBA server to see what the others are doing. Note also that this is generally not a very good idea, as each CORBA client could write to this (single) CORBA server and the same `Unit1.Form1.Memo1`, meaning potential multi-threaded issues. But since it's a mere debug version for this article only, you'll get the picture *(and you promise you won't do anything like this in real life, right?)*.



➤ *Figure 3: Type Library using WideString arguments.*

The Java applet would need to instantiate this CORBA server, and call the `GetLines` or the `SetLines` functions.

CORBA And JBuilder 2

In comparison with the implementations in the previous article, the CORBA implementation on the client side this time is very straightforward. JBuilder 2 Client/Server has a lot of the VisiBroker tools integrated in the IDE. This makes developing the Java applet with CORBA implementation easy.

The place to begin is with the IDL file generated by Delphi. This IDL file can be added to our JBuilder 2 project. Once we have got the IDL file in our project we are ready to process it. We will use the IDL to automatically generate all necessary Java classes for our Java applet.

Right click on the IDL file and select the `IDL properties...` option. This will open a dialog window with different options VisiBroker will use when the IDL file is processed. We can leave all options unchanged for our Java applet (after all, we must emphasise the Delphi side).

➤ *Listing 4: Implementation CORBA Server in Delphi.*

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  ComObj, StdVcl, CorbaObj, BobNotes_TLB;
type
  TCorBobNotes = class(TCorbaImplementation, ICorBobNotes)
  private
  public
  protected
    procedure GetLines(const User, Password: WideString;
      out Lines: WideString); safecall;
    procedure SetLines(const User, Password, Lines:
      WideString); safecall;
  end;
implementation
uses
  CorbInit, IniMod, Unit1;
procedure TCorBobNotes.GetLines(const User,
  Password: WideString; out Lines: WideString);
var
  Notes: TStringList;
begin
  Form1.Memo1.Lines.Add('---');
  Form1.Memo1.Lines.Add('Call to GetLines:');
  Form1.Memo1.Lines.Add(' User = ['+User+']');
  Form1.Memo1.Lines.Add(' Password = ['+Password+']');
  Notes := TStringList.Create;
  try
    IniMod.GetLines(User, Password, Notes);
    Lines := Notes.Text;
  finally
    Notes.Free
  end;
  Form1.Memo1.Lines.Add(' Lines = ['+Lines+']')
end;
procedure TCorBobNotes.SetLines(
  const User, Password, Lines: WideString);
var
  Notes: TStringList;
begin
  Form1.Memo1.Lines.Add('---');
  Form1.Memo1.Lines.Add('Call to SetLines:');
  Form1.Memo1.Lines.Add(' User = ['+User+']');
  Form1.Memo1.Lines.Add(' Password = ['+Password+']');
  Notes := TStringList.Create;
  try
    Notes.Text := Lines;
    IniMod.SetLines(User, Password, Notes);
  finally
    Notes.Free
  end;
  Form1.Memo1.Lines.Add(' Lines = ['+Lines+']')
end;
initialization
  TCorbaObjectFactory.Create('CorBobNotesFactory',
   'CorBobNotes', 'IDL:BobNotes/CorBobNotesFactory:1.0',
   ICorBobNotes, TCorBobNotes, iMultiInstance,
   tmSingleThread);
end.
```

When we close the dialog window, we can execute the Make command for our IDL file. Select the IDL file and right click with the mouse. Then select the Make command from the popup menu. Now the Visigenic IDL2JAVA compiler is started and all the necessary files, such as stubs and auxiliary files, are generated for us.

After the compile we can see a lot of new files in our project attached to the IDL file (simply click on the + symbol to see them). We can inspect these Java source files, but we *must not* change them. This is because when we do a rebuild of the IDL file all the files will be overwritten with new ones. Notice that the name of the package used for the Java source files is the same as the module name in the IDL file.

We now are able to use the generated files for our own applet. In order to be able to use the CORBA server, written in Delphi, we must take two steps. First of all, we must initiate an ORB object, so we are ready to use objects using the ORB. Next we must create an instance of the server in our applet. Then we can use the methods defined in the server class: GetLines and SetLines.

Initiating an ORB object is very simple. The following line creates an instance of the ORB named orb:

```
org.omg.CORBA.ORB orb =
  org.omg.CORBA.ORB.init(
  new String[] {}, null);
```

We use the static method init() defined in the ORB class to create a new instance. This method takes two arguments, which aren't important for our Java applet, so we leave them simple.

We have the ORB object and we can create instances of (server) objects available. And the Delphi server has one class to offer: CorBobNotesFactory. The CorBobNotesFactory is able to create instances of CorBobNotes objects. To create an instance of CorBobNotesFactory we must use the following line:

```
CorBobNotesFactory factory =
  CorBobNotesFactoryHelper.bind(
  orb);
```

For creating an instance we use the bind() method defined in the automatically generated Java class CorBobNotesFactoryHelper. The argument is the ORB object we created earlier.

CorBobNotesFactory has only one method: createInstance(). We use this method to create an instance of a CorBobNotes object:

```
IcorBobNotes bobNotes =
  factory.CreateInstance(
  "CorBobNotes");
```

And once we have a reference to the CORBA server object we can use it in our Java applet. The IcorBobNotes interface defines two methods: SetLines and GetLines. We can simply invoke these methods from within our applet:

```
BobNotes.SetLines(
  "Bob", "swart",
  "New appointment - May 1");
```

When using the GetLines() method we must pay special attention to the third argument, Lines, which is of class type org.omg.CORBA.StringHolder. In the IDL this argument is defined as an out argument, meaning that this argument will contain a return value. In order to be able to store this return value of type String (and for us to get the value) we must use the StringHolder class. Keep in mind that the argument cannot be of class type String, because the contents of a String object cannot be changed. The value property of this StringHolder class contains the return value as a String object.

Listing 5 shows a short example of an invocation of the GetLines() method.

➤ *Listing 5*

```
StringHolder holder = new StringHolder();
bobNotes.GetLines("Bob", "swart", holder);
System.out.println("holder.value? " + holder.value);
```

Well, folks, that wasn't too hard, was it? JBuilder 2 Client/Server supports CORBA in a straightforward way and because it is integrated into the IDE it is very easy to use. By the way, if you want to know about the forthcoming JBuilder 3 and how it compares with version 2, check the review that we will be doing for the June issue of *Developers Review*!

Conclusion

Combined with last month, we've seen at least three ways in which a Java applet can communicate with a Delphi server-side application. Last time, we explored CGI and sockets, and this time we used CORBA as communication protocol. Using JMidas, we can even use DCOM (see www.borland.com/midas for details).

Next time, we'll take a look at nested tables, another new feature introduced in Delphi 4. We'll see how they work, when to use them, and how they can especially benefit multi-tier applications written in Delphi. All this and more next time, *so stay tuned...*

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is a technical consultant and webmaster using Delphi, JBuilder and C++Builder for Bolesian and freelance technical author. In his spare time, Bob likes to watch video tapes of Star Trek Voyager and Deep Space Nine with his 5-year-old son Erik Mark Pascal and his 2.5-year-old daughter Natasha Louise Delphine.

Hubert A Klein Ikkink (aka Mr.Haki) has been using Java and JBuilder since they were invented and has wide commercial experience in the development and deployment of Java applications. Hubert is the webmaster and writes articles for Mr.Haki's JBuilder Machine at drbob42.com/JBuilder, trains and speaks on Java and is a freelance author too *[He probably doesn't have any spare time! Ed]*.