Under Construction: Delphi 4 And CORBA

by Bob Swart

One of the major enhancements in recent versions of Inprise tools is CORBA support. This is available in the Client/Server editions of Delphi 4 and JBuilder 2, plus C++Builder 4 Enterprise. In this article, we'll see how easy it is to build a CORBA server and client in Delphi 4 Client/Server.

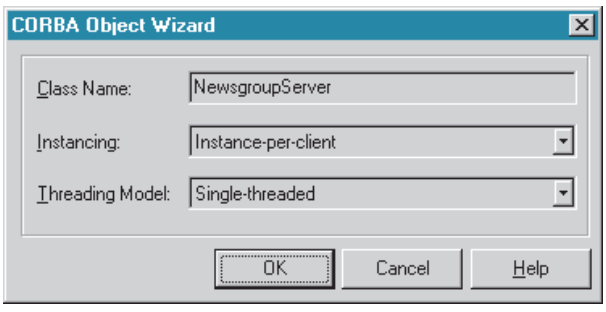
But first, let's look at what CORBA is all about. CORBA stands for Common Object Request Broker Architecture, and is an object oriented communication architecture between a client and a server. Communication is handled by an ORB (Object Request Broker) and IIOP (Internet InterORB Protocol). Using IDL (Interface Definition Language) we can specify objects with methods and properties. Methods are like functions that can be called by the client, and will be implemented (serviced) by the server. In order to do so, the IDL file must be compiled. This results in stub code for the client (so we can invoke the methods without worrying about the underlying communication) and skeleton code for the server (which is the basis for our communication on the server side). CORBA is independent of both platform and language. This can only work if the parameters and return types of the methods are transported over the network in a portable format. Conversion from a native type to a portable IDL type is called *marshalling*, while conversion back to a native platform or language type is called *un-marshalling*.

In using Delphi to put CORBA into practice I've decided to take the NNTP news reader component (from the last two issues) and create a newsgroup CORBA server and a CORBA Client. The CORBA News Server is connected to the 'real' internet (for example on our web server just in front of the company's firewall), while the CORBA News Client is running on my machine, which is not connected to the internet but uses the intranet to connect to the web server, behind the firewall (ie only 'inside' the office). This is a useful real world application, since the CORBA News Server is now able to obtain (and provide to the client) newsgroups and articles that otherwise I'd have no access to.

The CORBA News Server

The CORBA server application is just a regular application, using the `TBNNTP` component we created last time (but in general it can be any regular application). To turn it into a CORBA server we need to add a CORBA object to it. We use the CORBA object wizard from the Object Repository to create a server that can be accessed remotely by CORBA clients. From the `File | New - Multitier` tab, we chose CORBA Object, to get the CORBA Object Wizard (Figure 1).

First, we specify the classname of our CORBA object, derived from `TCorbaImplementation`. Note that you should not put a `T` in front of the classname, since it will be added automatically by Delphi (and so will the `I` for the interface class). The instancing option specifies how the CORBA server application creates instances of the CORBA object. We can either specify instance-per-client, so a new object is created for each client connected (until the connection is closed), or shared-instance, where one single instance of the CORBA object handles all client requests.



➤ *Figure 1*

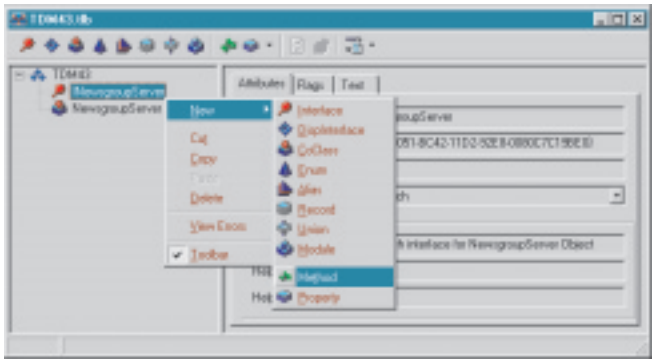
Although I'm the only client, I'm not sure if using the same CORBA object to handle all news requests is a good idea, so I've specified instance-per-client (the default).

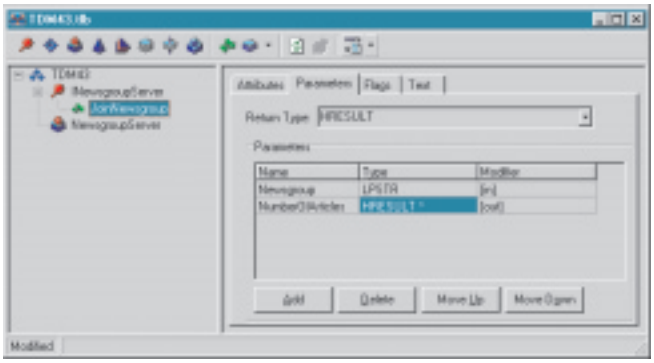
Threading specifies how the client requests call our CORBA object interface. This can be set to either single-threaded, where each CORBA object gets only one client request at a time, or multi-threaded, where each client request gets its own thread. To avoid thread conflicts, I've specified single-threaded (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).

This yields the source code shown in Listing 1.

Our CORBA object `TNewsgroupServer` is derived from `TCorbaImplementation`, and implements the `INewsgroupServer` interface (which we will specify soon). Note the `TDM43_TLB` unit, which is the (generated) ObjectPascal version of the type library that contains the compiled IDL specifications. Just like DCOM, for those with prior type library experience. To define new properties and methods for our CORBA object, we need to start the Type Library Editor from the `View | Type Library` menu.

Since our CORBA News Server will be passing information to our CORBA News Client, we must define some methods to pass the information along. Such as a method to connect to a particular newsgroup, a method to query the number of messages inside a newsgroup, and a method to get an article from that newsgroup (I won't bother with posting or replying at this time: see last month).





➤ *Left: Figure 2, Right: Figure 3*

To define a new method, we need to select the `INewsgroupServer` (ie the interface object) and either directly click on the green arrow button, or right click with the mouse and select `New | Method` (see Figure 2). We can change the name of the new method to `Join Newsgroup` and click on the `Parameters` tab to specify the parameters and return type. At this time, two things can happen. Either we see the default return type specified as `Integer` or as `HRESULT`. The former uses a special Pascal syntax for IDL, while the latter uses official IDL. Since I plan to use CORBA and IDL in many different development environments, I never use the Pascal-like IDL syntax, but always the real IDL. You can specify which IDL you want to see in the `Tools | Environment Options - Type Library` tab (the Language option is either Pascal or IDL). And even if you prefer to use the Pascal language yourself, for the remainder of this article we'll be using straight IDL.

Now, our method `JoinNewsgroup` should have one input argument (the `name` of the newsgroup, of type `LPSTR`, a `PChar` in Pascal terms, and modifier `[in]` so it's only used as a read-only input type), and a default return type (`HRESULT`, resulting in a procedure `safecall` in Pascal terms). Apart from the newsgroup input argument, we can click on the `Add` button to specify more arguments, like number of articles from the newsgroup in an output argument named `NumberOfArticles` of type `HRESULT*` (note that output parameters require a pointer type to be returned, so we cannot simply return an `integer` here). See also Figure 3.

After we've specified the arguments we need, we can click on the `Refresh Implementation` button on the toolbar (the one with the two little green arrows), to see a new server skeleton method defined in `unit2`:

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, ComObj, StdVcl,
  CorbaObj, TDM43_TLB;
type
  TNewsgroupServer = class(TCorbaImplementation, INewsgroupServer)
  private
  public
  end;
implementation
uses CorbInit;
initialization
  TCorbaObjectFactory.Create('NewsgroupServerFactory',
    'NewsgroupServer', 'IDL:TDM43/NewsgroupServerFactory:1.0', INewsgroupServer,
    TNewsgroupServer, iMultiInstance, tmSingleThread);
end.
```

➤ *Above: Listing 1*

➤ *Below: Listing 2*

```
procedure TNewsgroupServer.JoinNewsgroup(Newsgroup: PChar;
  out NumberOfArticles: HResult);
begin
  with Form1 do begin
    BNNTP1.JoinNewsgroup(Newsgroup);
    NumberOfArticles := BNNTP1.LastArticle - BNNTP1.FirstArticle
  end
end {JoinNewsgroup};
```

```
procedure JoinNewsgroup(
  Newsgroup: PChar;
  out NumberOfArticles:
  HResult); safecall;
```

This method should be called by a CORBA News Client in order to join a newsgroup and can be implemented as shown in Listing 2 (assuming that `unit1` is added to the `uses` clause of `unit2`, the `TBNNTP` component exists on the auto-created `MainForm`, and only one instance of both the CORBA object and form exist, as we specified in the CORBA Object Wizard earlier).

We can add more methods in this way, to obtain entire messages from the CORBA News Server, but the tricky part is getting the communication between the CORBA Client and Server working, so we'll focus on that now (we can add the other methods later).

Before we leave the CORBA News Server alone, however, I want you to take a look at the `Text` tab of the Type Library for the entire `TDM43` server, and take a look at what the IDL language looks like. If you plan to use CORBA more, then this should become second nature for you (and it isn't that hard to learn, especially since the CORBA IDL to ObjectPascal translation is done automatically when we update our implementation from within the Type Library).

The CORBA News Client

To start working with the CORBA News Client, we must make sure that we can find the CORBA News Server somewhere on our intranet. We must first run the VisiBroker Smart Agent (part of Delphi 4 Client/Server), or install Smart Agent as a service on at least one

machine on our intranet (in practice, the web server itself is a fine choice for this, but I could also use my own client machine). If the Smart Agent is running, we can start the CORBA News Server, which will just show the main form with the BNNTP component and do nothing (other than waiting for CORBA News Clients to send requests for joining newsgroups, since that's the only thing it can do at this time).

At this time, we can start building the CORBA News Client. Just start another regular application. Now, instead of adding a CORBA Object to this regular application (like we did for the server), we can just re-use the CORBA IDL that we created for the Server. Remember (from the introduction) that CORBA IDL compiles to both a Server Skeleton and a Client Stub. And we already implemented the Server Skeleton, so right now we only need a Client Stub, without implementation. In order to re-use the CORBA IDL, we just add the unit TDM43_TLB.PAS to our new application project using Project | Add to Project and select the TDM43_TLB.PAS file. Note that this file will be generated automatically whenever the IDL is refreshed (which we should generally only do at the Server side, since the Server side actually implements the IDL Skeleton methods, and the Client side only calls them). We must add the TDM43_TLB unit to the uses clause of our News Client unit1, in order to be able to call the CORBA NewsGroup object stub methods (that are implemented at the Server side, remember?).

We're almost ready. All we need to do now is make sure we can actually work with the CORBA NewsGroup object from within our client application. For this, we need to define a variable of type INewsgroupServer in the Form, and we can use the FormCreate and FormDestroy events to create and destroy this CORBA Object automatically for us (see Listing 3 for the syntax that we need to use).

The only thing that's left (for now) is a button and editbox on the CORBA News Client Form, with the OnClick event of the button calling the NewsgroupServer.JoinNewsgroup method, with the content of the editbox as an argument, placing the resulting number of articles value back in the editbox. This is implemented as shown in Listing 4.

A simple test run of the CORBA News Client (while the CORBA News Server and Smart Agent are running as well, of course), tells me there are 42 messages in my news.shoresoft.com/drbob newsgroup, all without a real internet connection on my machine!

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  TDM43_TLB;
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
  public
    NewsgroupServer: INewsgroupServer;
  end;
var Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
  NewsgroupServer :=
    TNewsgroupServerCorbaFactory.CreateInstance('NewsgroupServer');
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  NewsgroupServer := nil
end;
end.
```

➤ *Above: Listing 3*

➤ *Below: Listing 4*

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Newsgroup: PChar;
  Articles: HRESULT;
begin
  Newsgroup := PChar(Edit1.Text);
  NewsgroupServer.JoinNewsgroup(Newsgroup, Articles);
  Edit1.Text := Edit1.Text + ' ' + IntToStr(Articles)
end;
```

Advanced CORBA

What we've seen so far is basic CORBA. It gets better (and worse) if you want to extend the functionality with callbacks (from the client back to the server) or error recovery (say a newsgroup doesn't exist, and we want to raise an exception on the server).

Some of these more advanced CORBA techniques will be covered in a future article on callbacks, others may be found in *Delphi 4 Unleashed* or at the new CORBA section of my website at www.drbob42.com/CORBA/.

Next Time

In the introduction, I told you that CORBA is cross-platform and also cross-language. And that's also the introduction for next time, when my friend Hubert A Klein Ikkink (aka Mr.Haki) will join me to connect a Java applet to a Delphi web server application. The techniques we'll use include HTTP (CGI), sockets, CORBA and DCOM. We'll try to show you all the possible inter-language communication options, of course, 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 a freelance technical author. In his spare time, Bob likes to watch video tapes of Star Trek Voyager and Deep Space Nine with his 4.5-year-old son Erik Mark Pascal and his 2-year-old daughter Natasha Louise Delphine.