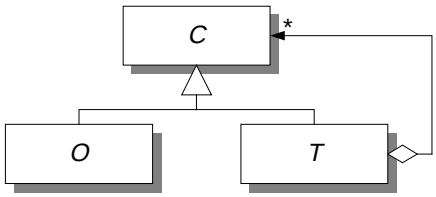
Development of the Pilot Application in Delphi

COT/3-10-V1.1



Centre for Object Technology

Revision history:	V0.1	18-08-98	First draft.
	V1.0	20-08-98	General revisions
	V1.1	08-09-98	First public version.

Author(s): Peter Petersen, Aarhus University
Kåre Kjelstrøm, Aarhus University

Status: Final

Publication: Public

Summary:

The report includes the investigation of Borland Delphi as a COM development environment using a Pilot Application.

The Centre for Object Technology (COT) is a three year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Danish National

Participants are:
Maersk Line, Maersk Training Centre, Bang & Olufsen, WM-data, Rambøll, Danfoss, Systematic Software Engineering, Odense Steel Shipyard, A.P. Møller, University of Aarhus, Odense University,

Content

Abstract

This document describes how Borland Delphi can be used for implementing the COM taxonomy pilot application. A description of the application can be found in COT-3-15: The COM Pilot Application.

This report is part of an evaluation of four development environments with respect to their support for COM based development. The main document of this evaluation is COT-3-4: Evaluation of COM Support in Development Environments.

1. Phase 1

1.1 Phase 1.1 – Create a component with an arithmetic interface

We have thus employed the IUnknown interface described in cot-3-15 (deep1_1.idl). Usually when designing a COM interface in Delphi, you will utilise the Type Library Editor, which is a graphical tool, to design functions, enumerations, properties and other COM related items. This is therefore the method we employed.

Delphi has a predefined application type for building in-process servers, the ActiveX library, which can be created from the file menu. This library automatically gives you an implementation of the 4 DLL functions DllGetClassObject, DllCanUnloadNow, DllRegisterServer and DllUnregisterServer, which are used by the Windows NT Service Control Manager (SCM) to handle COM servers. Step 2 in building the calculator component was to create a new ActiveX library application.

By selecting new from the file menu and then choosing the OleAutomation wizard from the ActiveX tab, you get a new interface and CoClass definition. Delphi automatically opens the typelibrary editor, and you can then add the desired method declarations to your interface.

When you push the refresh button, Delphi generates a wrapper file, which contains Pascal declarations of your interfaces. In addition, using the OleAutomation wizard automatically gives you a new source file with a skeleton for the CoClass.
When building a COM server, you also have to implement an object factory for your class. Delphi defines several such factories of increasing complexity, which you are free to use. When using the wizards, Delphi automatically adds the declaration

```
initialization
    TAutoObjectFactory.Create(ComServer, TCalc, Class_Calc,
ciMultiInstance);
```

to the end of the CoClass unit. An instance of the object factory will then be created by the SCM when the DLL is loaded.

Since the interface specification in this phase is bugged, we choose to let the implementation return a zero.

It is most convenient to create windows applications with a graphical interface in Delphi, and in this phase as well as the rest of the phases, we test the servers on such an application.

1.2 Phase 1.2 - Change the existing members of an interface

To correct the errors in the specification from phase 1.1, we opened the type library in the editor by selecting the tool from the menus. We then corrected the errors, pushed the refresh button and returned to the source code. All the function headers in both the wrapper file and the CoClass unit had been altered according to our changes, but the old implementations ie. the bodies of the functions were kept.

The type library editor has a hard time deciding whether to use one syntax or the other when auto-defining the method declarations. Sometimes an interface method will be defined as follows:

```
function Add(v1, v2: Integer; out Value: Integer): HResult;
stdcall;
```

At other times the editor will choose a more "Delphi like" syntax:

```
function Add(v1, v2: Integer): Integer; stdcall;
```

When editing a type library you may actually end up in a situation where a previous declaration gets changed from one syntax to the other, and this event can be quite tedious.

1.3 Phase 1.3 – Add a member to an existing interface

As before we opened the type library editor and added the new member to ICalc. Again CoClass code and Pascal wrapper were updated.

2. Phase 2

2.1 Phase 2.1 - Add an interface to Deep_Thought's Calc component

A new interface was added via the type library editor. In order to tell the editor that the interface belonged to the Calc CoClass, we selected the icon for the CoClass, clicked the members tab, right clicked the canvas and choose "insert interface" from the pop-up menu. Again the source files were nicely updated and as before all we had to do was to fill in the implementation of the new methods.

2.2 Phase 2.2 - Remove an interface from Deep_Thought's Calc component

To remove an interface, you will open the type library editor, right click the interface and select "delete" from the pop-up menu. This results in the wrapper file being updated, but the CoClass still contains both declarations and implementations of the member methods that the deleted interface defined. These must then be removed by hand.

3. Phase 3

3.1 Phase 3.1 – Add a component to Deep_Thought

Adding a new component is done through the wizard, precisely as described in phase 1.1. You get a unit for the CoClass and the existing wrapper file is updated.

3.2 Phase 3.2 - Remove a component from Deep_Thought

When removing a component through the type library editor, by deleting interface and CoClass definitions, the wrapper file is nicely cleaned up, but the CoClass unit is still a part of your project and must be removed by hand.

4. Phase 4

4.1 Phase 4.1 – Delegation

To implement a server that exposes the ICalc interface through containment, we started out creating a new ActiveX library and component as described in phase 1.1. The GUID

defined in the IDL file for this phase was pasted into the editor and the interface for ICalc2 was defined by adding it from the editor. In a manner similar to that described in phase 2.1, the interface was connected to the CoClass. Again the GUID for ICalc2 from the IDL file was pasted into the editor to replace the auto generated one.
Every Delphi defined COM object defines a virtual method Initialize, which the programmer is free to override and where he can specify code that should be executed when the object is launched. In order to delegate the method calls on add, subtract, multiply and divide to the old Calc server, we needed a pointer to its interface, and this was obtained in the initialize procedure:

```
procedure TCalcDel.Initialize;
begin
    Calc := CreateComObject(Class_Calc) as ICalc;
end;
```

Declarations for the variable Calc and the initialize procedure were added by hand. The 4 delegating methods were implemented by simply calling the corresponding functions in the old Calc object:

```
function TCalcDel.Add(v1, v2: Integer; out Value: Integer):
HResult;
begin
    Result := Calc.Add(v1, v2, Value);
end;
```

Finally the code for the new function "Power" was added to the auto generated skeleton.

4.2 Phase 4.2 – Aggregation

Every COM object defined using Delphi and the supplied default implementation of IUnknown is ready to be the inner object of an aggregate. Delphi handles this by keeping a pointer to the controlling IUnknown and by supplying 2 versions of QueryInterface, AddRef and Release. The first version is for the case when no aggregation takes place, and the second handles the aggregation case.
The outer object needs a little more work, since it is not possible to define a general QueryInterface in this case. Furthermore the outer object needs to initialise its inner object, when the server is first launched. The outer object was created in a manner similar to that described in phase 4.2. The main difference is that the ICalc interface was not connected to the CoClass, since its implementation is to be found in another DLL.
As with containment, we had to add the Initialize procedure:

```
procedure TCalc2.Initialize;
begin
    pUnkOuter := Self as IUnknown;

    // Get Iunknown of inner object
    OleCheck(CoCreateInstance(Class_Calc, pUnkOuter,
CLSCTX_INPROC_SERVER, IUnknown, pUnkInner));
end;
```

The implementation of QueryInterface first checks if the interface requested is implemented by the outer object, and if not, passes the parameters to the inner object's QueryInterface:

```
function TCalc2.QueryInterface(const IID: TGUID; out Obj):
Integer;
begin
    if GetInterface(IID, Obj) then
        Result := S_OK
    else
        Result := pUnkInner.QueryInterface(IID, Obj);
end;
```

5. Phase 5 - Create the Deep_Thought server

To change the in-proc server to an out-of-proc server, we created a new application, added the 2 units for the CoClass implementations along with the Pascal wrappers and specified that the project should use the type library from the in-proc server via the {$R} compiler directive.

<This did not work with IUnknown>

To move from out-of-proc to in-proc, you would create a new ActiveX library and follow the excact same procedure as in the reversed case.

6. Phase 6 – Create a client with a graphical interface

Delphi has a large array of graphical components that you can use at your convenience, and it also supports using ActiveX components. Native Delphi controls are linked into the target executable, whereas ActiveX controls, of course, are not. We used the native controls for defining the interface.

7. Phase 7 – Change Deep_Thought to implement the IDispatch interface

To change from IUnknown to IDispatch, we simply opened the type library editor and selected IDispatch as the parent. Furthermore we clicked the dual checkbox in order to get a dual interface for the server. After this, all we had to do was to register the server.

8. Phase 8 - Create installation disks

Delphi InProc servers support the register and unregister functions, and you will thus be able to install a new component by simply writing "regsvr32.exe Server.dll".
OutOfProc servers will automatically register themselves if you start them as you would an ordinary executable.
With this in mind, we see little need of an installation program.

9. Phase 9 - Exchange implementations

The result of this task task is described in cot-3-4.