Einführung in das Component Object Model unter Microsoft Windows

Das Component Object Model (COM)

Michael Puff
mail@michael-puff.de

2010-03-26

Die erste Version dieses Dokumentes wurde im Rahmen einer Evaluierung verschiedener Skinning-Bibliotheken für VCL und .Net erstellt.

Mit freundlicher Genehmigung der DATAWERK GmbH & Co. KG (www.datawerk.de)

Inhaltsverzeichnis

1. Einführung

Das Component Object Model ist eine von Microsoft entwickelte Technologie, um unter Windows Interprozesskommunikation und dynamische Objekterzeugung zu ermöglichen. COM-fähige Objekte sind sprachunabhängig und können in jeder Sprache implementiert werden, deren Compiler entsprechenden Code erzeugen kann. Der Zugriff auf die Funktionalität eines COM-Objektes erfolgt über ein Interface, welches nach Instanzierung Zugriff auf die angebotenen Funktionen des COM-Objektes ermöglicht.

2. Die COM–Architektur

COM basiert auf dem Client/Server Prinzip. Ein Client instanziert eine COM-Komponente in einem COM-Server und nutzt dessen Funktionalität über Interfaces.

2.1. COM–Server

Ein COM-Server ist eine DLL oder ausführbare Datei, die eine COM–Komponente beinhaltet und bereitstellt. Es gibt drei Arten von COM-Servern:

1. In-process Server
2. Local Server und
3. Remote Server

2.1.1. In–process–Server

Handelt es sich um einen in–process–Server, ist die COM-Komponente in einer DLL implementiert (*.dll, *.ocx). Diese DLL muss die Funktionen

- `DllGetClassObject()`
- `DllCanUnloadNow()`
- `DllRegisterServer()` und
- `DllUnregisterServer()`

exportieren. Wird eine COM–Komponente aus einem COM–Server instanziert, wird der zugehörige Server in den Prozess der Anwendung (COM-Client) geladen. Dies hat den Vorteil, dass in–process–Server sehr schnell sind, da der Zugriff auf die COM-Komponente ohne Umwege erfolgen kann. Allerdings hat es den Nachteil, dass jeder Prozess, der eine COM-Komponente nutzen will, die in einem in–process–Server liegt, diesen in seinen Adressraum laden muss, was nicht besonders speicherschonend ist.

2.1.2. Local Server

Local Server sind ausführbare Programme, die eine COM–Komponente implementieren. Instanziert ein COM–Client eine COM–Komponente, so wird das Programm, welches die COM–Komponente beinhaltet, gestartet. Die Kommunikation erfolgt über ein vereinfachtes

RPC-Protokoll (Remote Procedure Call). Local Server haben den Vorteil, dass sie nur einmal gestartet werden müssen, um mehrere Clients bedienen zu können. sie sind daher speicherschonender als die in–process Variante. Zudem lassen sich so mit Hilfe eines zentralen Servers leichter Zugriffe von den Clients auf gemeinsame Ressourcen synchronisieren. allerdings ist der Zugriff mittels RPC langsamer.

2.1.3. Remote Server

Befindet sich zwischen Server und Client ein Netzwerk, kommt DCOM (Distributed COM) zum Einsatz. Es unterscheidet sich von COM nur in sofern, als dass ein vollständiges RPC–Protokoll zum Einsatz kommt und ein Protokollstack vorgeschaltet wird. Aufgrund der Verwendung des vollständigen RPC–Protokolls werden die Aufrufe, auch bei geringer Netzwerklast, ziemlich verlangsamt.

2.2. Schnittstelle

Die Kommunikation zwischen COM–Server und COM–Client erfolgt über das COM–Interface. Jedes Interface wird über seinen Namen und eine GUID (Globally Unique Identifier) eindeutig identifiziert. Dadurch können mehrere Interfaces mit dem gleichen Namen existieren ohne dass es zu Verwechslungen kommt. Damit eine programmiersprachenübergreifende Kommunikation möglich ist, findet am Interface das so genannte Marshalling statt, welches die auszutauschenden Daten in eine vordefinierte Binärrepresentation umwandelt.

Ein Interface ist eine abstrakte Klasse, die nur virtuelle Methoden enthält, die wegen der Trennung von Definition und Implementation allesamt auf 0 im VTable gesetzt werden.

Wenn ein COM–Objekt ein Interface implementiert, muss es alle Methoden des Interfaces überschreiben. Dabei sind mindestens die drei Methoden von `IUnknown` zu implementieren, die für das Lifetimemanagement zuständig sind.

Ein Interface sieht in der für COM–Komponenten nutzbaren IDL (Interface Definition Language) unter Delphi wie folgt aus:

```
type
  IInterface = interface
    ['{00000000-0000-0000-C000-000000000046}']
    function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
  end;

  IUnknown = IInterface;
```

Jedes Interface muss über eine Interface–Vererbung die Funktionen des obigen Interfaces `IUnknown` definieren, da dies die grundlegenden Funktionen für COM implementiert.

Da Programmiersprachen wie Visual Basic Script keine Typen kennen, hat Microsoft eine weitere Möglichkeit entwickelt, Funktionen aus COM–Interfaces aufzurufen. Für diese Möglichkeit muss das Interface die Funktionen des Interfaces `IDispatch` definieren. Dies erlaubt es dann dem Programmierer, eine COM-Komponente über `IDispatch.Invoke()` anzusprechen, ohne dass der COM-Client die Typbibliothek des Servers kennen muss. Da der Zugriff über das Dispatch-Interface sehr viel langsamer als der Zugriff über ein typisiertes Interface ist, wird oft beides implementiert (Dual Interface), so dass sich der Programmierer bei Programmiersprachen, die Pointer beherrschen, den Weg des Zugriffs auf die COM-Komponente aussuchen kann.

2.3. COM–Komponente

Eine COM–Komponente bietet die aufrufbaren Funktionen des COM–Servers über ein Interface an. Die Instanzierung des Objektes erfolgt durch die Implementierung von `IClassFactory.CreateInstance()` im COM–Server. Zurückgeliefert wird dann eine Instanz der erzeugten Klasse. Und genau das ist der Punkt, den COM unter anderem auszeichnet: Das zur Verfügungstellen von Klassen in Programmbibliotheken (DLLs). DLLs können nämlich keine Objekte nach aussen zur Verfügung stellen, da eine DLL einen eigenen Speichermanager, gegenüber der Anwendung selber, welche die DLL nutzt, besitzt.

COM–Komponenten müssen, im Gegensatz zu gewöhnlichen Objekten, nicht wieder freigegeben werden. Dies übernimmt der COM–Server selbstständig. Wird ein Objekt instanziert, wird ein Referenzzähler hochgezählt, der bei einem Aufruf von `Release()` dekrementiert wird. So lange der Referenzzähler ungleich Null ist »lebt« das Objekt. Ein Aufruf von `Release()` erfolgt, wenn das Objekt im COM–Client auf einen Null–Zeiger (Delphi: `nil`, C++: `NULL`) gesetzt wird.

Eine COM–Komponente kann mehrere Interfaces zur Verfügung stellen. Dies ist insofern auch sinnvoll als das nur so eine Erweiterung der Funktionalität möglich ist ohne Gefahr zu laufen, bestehende Anwendungen, die diese Schnittstelle nutzen, neu kompilieren zu müssen. Da der Compiler die aus der VTable gelesenen Einsprungadressen der vom Client aufgerufenen Funktionen unter bestimmten Umständen fest kodiert. Wird nun das Interface einer Komponente geändert, kann sich die Einsprungsadresse ändern, was es den abhängigen Clients unmöglich machen würde, diese COM–Komponente zu nutzen ohne dass sie neu kompiliert werden müsste.

2.4. COM–Client

Als COM–Client bezeichnet man eine Anwendung, die die COM–Komponente eines COM–Servers letztendlich nutzt. Der Client kann die Funktionen der COM–Komponente nutzen, da diese in den entsprechenden COM–Interfaces deklariert sind. Die Veröffentlichung der

Interfaces erfolgt entweder über Typbibliotheken[1] oder Beschreibungen in der IDL (Interface Definition Language).

2.5. Apartments

COM-Objekte werden bei Instanzierung immer einem so genannten Apartment zugeordnet. Dabei handelt es sich um transparente Rahmen, welche zur Synchronisierung von Methodenaufrufen mehrerer Objekte dienen, die mit unterschiedlichen Anforderungen an die Threadsicherheit arbeiten. Wird COM nicht mitgeteilt, dass eine entsprechende Komponente threadsicher ist, wird COM nur einen Aufruf gleichzeitig an ein Objekt erlauben. Threadsichere Komponenten können auf jedem Objekt beliebig viele Aufrufe gleichzeitig ausführen.

Damit ein Thread COM benutzen kann, muss der Thread zuvor einem Apartment zugeordnet werden. Erst nach der Zuordnung ist eine Verwendung von COM möglich. Dabei kann ein Thread entweder einen schon bestehenden Apartment zugeordnet (MTA) werden oder es wird ein neues Apartment (STA) erstellt. Die Zuordnung geschieht mit der API-Funktion `CoInitialize()`. Programmiersprachen mit integrierter COM–Unterstützung führen diese Zuordnung meist automatisch. Erstellt man allerdings selber einen neuen Thread, so muss für diesen Thread die Zuordnung manuell mit einem Aufruf von `CoInitialize()` erfolgen und mit `CoUninitialize()` wieder aufgehoben werden.

Jede COM–Komponente wird beim Aufruf genauso einem Apartment zu geordnet. Falls die Apartment–Anforderungen der erzeugten Komponente zum Apartment Typ des erzeugenden Threads passen, wird das Objekt dem gleichen Apartment zugeordnet. Bei Aufrufen über Prozessgrenzen hinweg liegen die beiden Objekte immer in verschiedenen Apartments. Die Zuordnung zu einem Apartment kann während der Lebensdauer des Objektes nicht geändert werden.

Es gibt drei Arten von Apartments:

- Single Threaded Apartments (STA) besitzen genau einen Thread und beliebig viele Objekte. Aufrufe von verschiedenen Clients an das Objekt werden nacheinander abgearbeitet, wobei die Aufrufe der Clients in einer Warteschleife auf die Freigabe des Apartment–Threads warten. Dies ist vergleichbar mit dem Konzept der CriticalSections in der Windows API.
- Multi Threaded Apartments (MTA) besitzen beliebig viele Threads. In einem Multi Threaded Apartment können mehrere Clients gleichzeitig Aufrufe an das gleiche oder verschiedene Objekte machen. Dies erfordert allerdings auch eine entsprechend threadsichere Implementierung der Komponente.
- Neutral Thread Apartments (NTA) haben keine Threadaffinität. Jedes Objekt in einem NTA kann von einem STA/MTA Apartment ohne Threadübergang aufgerufen werden.

Näheres zu Apartments und deren Funktionalität auf der Seite http://www.codeguru.com in dem Beitrag „Understanding COM Apartments, Part I“[2].

[1] Eine Typbibliothek ist eine Binärdatei, die in einem MS spezifischen Format COM-Klassen und -Schnittstellen beschreibt.

[2] http://www.codeguru.com/Cpp/COM-Tech/activex/apts/article.php/c5529/

2.6. Typbibliotheken

Man muss für COM Routinen, die Out–of–process laufen – also einen eigenen Prozess besitzen – eine Schnittstellensprache verwenden, die IDL (Interface Definition Language). Sie wird in dem MS Format TLB gespeichert. In Delphi erstellt man dazu eine „Typbibliothek“ und fügt dort neue Schnittstellen ein. Diese wird in der IDL Sprache gespeichert.

Das muss so gemacht werden, weil jede andere Sprache diesen COM–Server ansteuern können soll – also nicht nur Delphi. Zudem müssen Parameter und Rückgabewert von einem Prozess in den COM-Prozess serialisiert (to marshall) werden. Es ist daher nicht möglich, beliebige Datentypen als Parameter zu verwenden. COM spezifiziert dazu seine eigenen Datentypen (BSTR) und andere müssen durch eine eigene Marshall–Implementation abgedeckt werden. Wenn man eine Typbibliothek erstellt hat, erstellt man darin ein COM–Objekt und implementiert es auch gleich. Also zuerst die abstrakten Methoden einfügen, dann das Interface ableiten und die Methoden implementieren.

3. Registration des COM–Servers

3.1. Einen COM–Server mit regsvr32 registrieren und de-registrieren

Aus Gründen, wie in Kapitel «4.3 Ortsunabhängigkeit» (Seite 14) erläutert, müssen COM–Server im System angemeldet oder anders ausgedrückt, im System bekannt gemacht werden, damit sie genutzt werden können. Dies geschieht mit dem Windows Kommandozeilenprogramm `regsvr32`. Um einen COM–Server zu registrieren wird das Programm wie folgt aufgerufen:

```
regsvr32 <Dateiname>
```

Wobei *Dateiname* der Dateiname der Datei ist, die den COM–Server enthält. Ein Beispielaufruf könnte demnach so aussehen:

```
regsvr32 COM_Test.dll
```

Um einen COM–Server wieder zu de–registrieren, wird das Programm mit dem zusätzlichen Schalter `\u` aufgerufen:

```
regsvr32 \u COM_Test.dll
```

Hinweis: Um einen COM–Server registrieren zu können müssen Schreibrechte im Registry–Zweig `HKEY_CLASSES_ROOT` vorhanden sein. In der Regel hat nur der Administrator diese Rechte. Eine Anwendung die einen COM–Server mitbringt, muss also von einem Administrator installiert oder eingerichtet werden. Im Idealfall, sollte entweder das Setup bei der Installation den COM–Server im System registrieren oder die Anwendung selber beim ersten Start, was aber wieder administrative Rechte erfordern würde.

3.2. COM–Server mittels API–Funktionen registrieren / de–registrieren

Jeder COM–Server implementiert und exportiert zwei Funktionen zum registrieren und de–registrieren seiner COM-Objekte: `DllRegisterServer` und `DllunregisterServer`. Will man nun einen Server im System registrieren und de–registrieren, muss man nur die entsprechende Funktion aus der COM–Server DLL aufrufen. Dazu müssen diese Funktionen jedoch dynamisch mit den API–Funktionen `LoadLibrary` und `GetProcAddress` geladen werden. Siehe dazu das folgendes Delphi–Beispiel:

```
type
  TDLLRegisterServer = function:DWORD;
  TDLLUnregisterServer = function:DWORD;

function RegisterServer(const Filename: String): Boolean;
var
  hLib: THandle;
  ProcAddress: TDLLRegisterServer;
begin
  Result := False;
  hLib := LoadLibrary(PChar(Filename));
  if hLib <> 0 then
  begin
    @ProcAddress := GetProcAddress(hLib, 'DllRegisterServer');
    if Assigned(ProcAddress) then
    begin
      Result := ProcAddress = S_OK;
    end
  end
end;

function UnregisterServer(const Filename: String): Boolean;
var
  hLib: THandle;
  ProcAddress: TDLLUnregisterServer;
begin
  Result := False;
  hLib := LoadLibrary(PChar(Filename));
  if hLib <> 0 then
  begin
    @ProcAddress := GetProcAddress(hLib, 'DllUnregisterServer');
    if Assigned(ProcAddress) then
    begin
      Result := ProcAddress = S_OK;
    end
  end
end;
```

3.3. Ist ein COM–Server / COM–Objekt im System bekannt?

Um zu testen, ob ein COM–Server im System registriert ist bzw. ob ein bestimmter COM-Objekt im System existiert, kann man entwder nach in der Registry im Zweig `HKEY\_CLASSES_ROOT\CLSID` nach der GUID suchen, wenn sie bekannt ist oder nach der `ProgID`, welche sich aus dem Dateinamen des COM–Servers ohne Dateiendung und der Interface-Bezeichnung des COM–OBjektes zusammen setzt, verbunden mit einem Punkt: `COMServer.InterfaceBezeichnung` (siehe Abbildung 3.1 auf Seite 12).

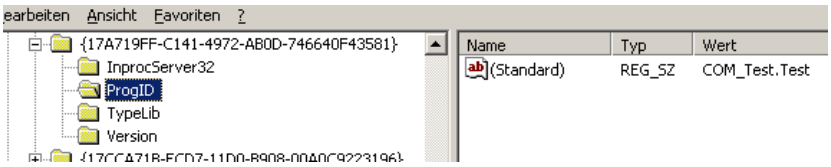

Abb. 3.1.: Registry–Editor mit geöffneten Eintrag für die Demo–Anwendung

Siehe dazu auch die Abbildung 4.1 auf Seite 14, wo der Dateiname des zugehörigen COM–Servers zu sehen ist.

3.3.1. Die API–Funktion CLSIDFromProgID

Will man innerhalb eines Programmes überprüfen, ob ein COM–Objekt zur Verfügung steht, kann man die API–Funktion `CLSIDFromProgID` benutzen:

```
HRESULT CLSIDFromProgID(
  LPCOLESTR lpszProgID,
  LPCLSID pclsid
);
```

Parameter	Bedeutung
lpszProgID	[in] Pointer to the ProgID whose CLSID is requested.
pclsid	[out] Pointer to the retrieved CLSID on return.

Tab. 3.1.: Parameter CLSIDFromProgID

Ein Beispiel für Delphi[1]:

```
function ProgIDExists(const ProgID:WideString):Boolean;
var
   tmp : TGUID;
begin
   Result := Succeeded(CLSIDFromProgID(PWideChar(ProgID), tmp));
end;
```

[1]Quelle: Delphipraxis (http://www.delphipraxis.net/post797655.html#797655), Autor: shmia.

4. Vorteile von COM

COM bietet viele Vorteile gegenüber herkömmlichen Techniken, zum Beispiel DLLs, Funktionen Anwendungen zur Verfügung zu stellen.

- sprachunabhängig
- versionsunabhängig
- plattformunabhängig
- objektorientiert
- ortsunabhängig
- automatisiert

Viele Windows Funktionen sind über COM zugänglich. Desweiteren ist COM die Basis für die OLE–Automation (Object Linking and Embedding) und ActiveX.

4.1. Sprachunabhängigkeit

Eine COM–Komponente kann in jeder beliebigen Programmiersprache implementiert werden. Ebenso kann der Zugriff über jede beliebige Programmiersprache erfolgen. Dies ist zwar bei DLLs auch der Fall, sollen allerdings DLLs Objekt–Instanzen zurückgeben, ist eine andere Technik nötig. Borland hat für Delphi, um dies zu ermöglichen, die BPL (Borland Package Library)–Technologie entwickelt. Dies ist natürlich nicht sprachunabhängig sondern eben nur mit Delphi bzw. dem Borland C++–Builder einsetzbar.

4.2. Versionsunabhängigkeit

Ein weiterer wichtiger Vorteil beim Einsatz von COM ist es, dass man die Verwaltung von neuen Softwarefeatures einfach in eine bestehende Anwendung integrieren kann. Oftmals kann es Probleme geben, wenn man herstellerneutrale oder herstellerübergreifende Softwarekomponenten mit weiteren Funktionen ausstattet. Dadurch kann zwar die eigene Software erweitert werden, jedoch besteht die Gefahr, dass andere Software, welche ebenfalls die herstellerübergreifenden Komponenten verwendet, nicht mehr funktionsfähig bleibt.

Mit COM hat man jetzt die Möglichkeit die Softwarekomponente mit weiteren Funktionen zu erweitern, in dem man weitere Interfaces hinzufügt. Dabei gehen die alten Interfaces nicht verloren. Interfaces werden also nicht erweitert oder verändert, sondern es werden weiter Interfaces hinzugefügt. Somit entsteht keine Inkonsistenz.

4.3. Plattformunabhängigkeit

x64-Applikationen können dank Marshalling auf 32bittige COM-Server zugreifen (und umgekehrt). Der COM-Server muss dann in einem eigenen Prozess laufen und seine Objekte können demnach nicht mit der INPROC_SERVER Variante instantiiert werden.

4.4. Ortsunabhängigkeit

COM ist ortsunabhängig, d. h. dass die einzelnen COM-Komponenten an einer zentralen Stelle (Registry) angemeldet werden und so der Zugriff auf die Komponenten unabhängig von ihrem eigentlichen Ort erfolgen kann. Dies bezeichnet man auch als Ortstransparenz[1]. Dies ist auch der Grund, warum COM–Server registriert werden müssen. Da man COM–Server über deren GUID anspricht, muss an einem zentralen Ort hinterlegt werden, welche GUID zu welchem COM–Objekt gehört und in welcher DLL sich das COM–Objekt befindet – deswegen muss ein COM–Client nicht wissen, wo sich die DLL mit dem COM–Server befindet. Dies erledigt alles Windows für ihn. Diese Informationen werden in der Registry hinterlegt. Zusätzlich werden dort Informationen zu dem COM–Server abgelegt, wie zum Beispiel der Einsprungspunkt der DLL, ob es sich um einen in–process Server oder Local Server handelt und es wird der Typmarshaller festgelegt, wenn es sich um einen Local Server handelt.

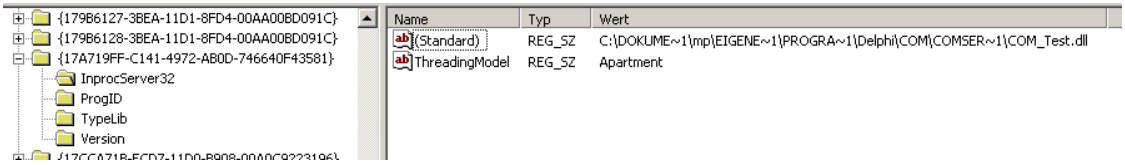

Abb. 4.1.: Registry–Editor mit geöffneten Eintrag für die Demo–Anwendung

4.5. Automatisierung

Das Steuern von Anwendungen über COM-Interfaces wird als Automatisierung bezeichnet.

[1] Ortstransparenz bedeutet in der EDV, dass der Benutzer einer verteilten Anwendung den tatsächlichen Ort des angefragten Objektes oder der angefragten Ressource nicht kennen muss.

5. Erstellen eines COM-Servers mit Delphi

Das folgende Beispiel bezieht sich auf die Vorgehensweise mit Borland Delphi 6. Bei neueren Versionen sollte es aber ähnlich gehen.

Mit Delphi lässt sich ein COM–Server relativ schnell und unkompliziert selber erstellen. Dazu sind letztendlich nur fünf Schritte nötig:

1. Erstellen einer ActiveX–Bibliothek.
2. Erstellen eines COM–OBjektes in der ActiveX–Bibliothek mit Deklaration des Interfaces.
3. Hinzufügen von Methoden zu dem Interface über den Typbibliothekseditor.
4. Implementation der Methoden
5. Registration des COM–Servers.

Die einzelnen Schritte sollen im Folgenden noch mal etwas ausführlicher beschrieben und erläutert werden.

1. Erstellen der ActiveX–Bibliothek

Um eine ActiveX–Bibliothek anzulegen, wählt man beim Erstellen eines neuen Projektes in der Objektgalerie unter dem Seitenreiter ActiveX «ActiveX–Bibliothek» aus (siehe Abbildung 5.1, Seite 15, Rahmen eins).

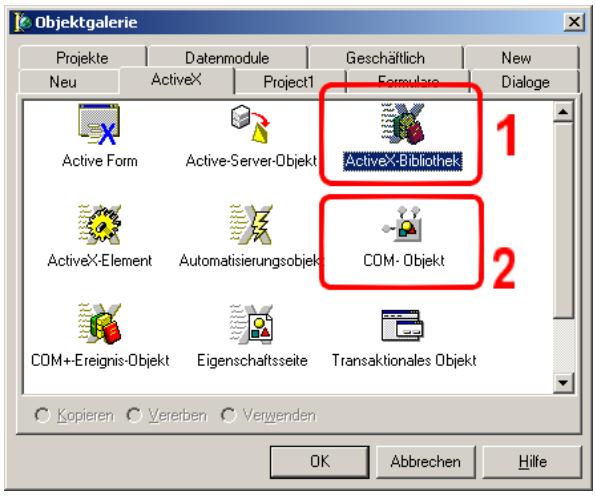

Abb. 5.1.: Objektgalerie

Delphi erzeugt daraufhin das Code–Grundgerüst einer DLL in der Projektdatei (dpr) – unsere COM–Server DLL.

5.1. Erstellen des COM–Objektes

Das Erstellen eines COM–Objektes erfolgt auch über die Objektgalerie, wie man in Abbildung 5.1 auf Seite 15 im Rahmen 2 sehen kann. Mit Hilfe des COM–Objekt–Experten (siehe Abbildung 5.2, Seite 16)

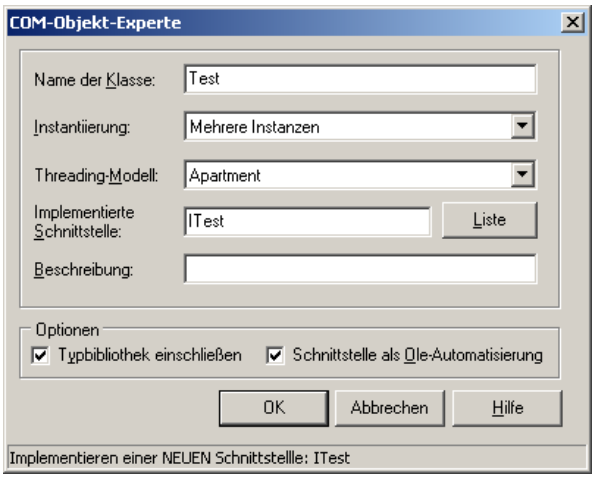

Abb. 5.2.: COM–Objekt–Erxperte

kann man dann das COM–Objekt konfigurieren. Unter anderem kann man

- die Art der Instantiierung festlegen,
- das Threading-Modell und
- die implementierte Schnittstelle.

Delphi erstellt daraufhin eine Unit zur Implementation des Interfaces und öffnet den Typbibliotheken–Editor (siehe Abbildung 5.3, Seite 17, um das Interface zu implementieren):

5.2. Hinzufügen und Implementation von Methoden

Über den Typbibliotheken–Editor kann man unter anderem neue Methoden, Eigenschaften (siehe Abbildung 5.3, Seite 17, Rahmen 1) anlegen, den COM–Server registrieren (siehe Abbildung 5.3, Seite 17, Rahmen 2) oder eine IDL-Datei erzeugen lassen. Die Implementation erfolgt dann direkt in der mit angelegten Unit.

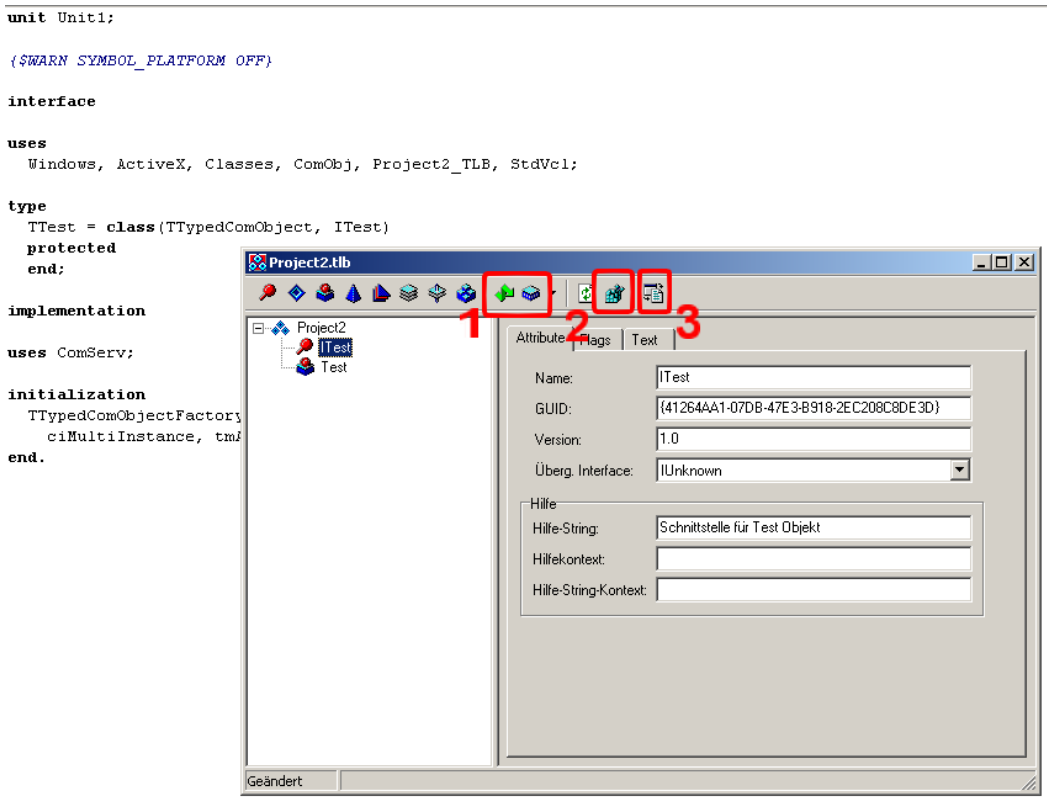

Abb. 5.3.: Typbibliotheken–Editor

Nach Hinzufügen einer Unit könnte dann der Quellcode von der Unit ungefähr so aussehen:

```
unit Unit1;

{\$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, ActiveX, Classes, ComObj, Project2_TLB, StdVcl;

type
  TTestCOM = class(TTypedComObject, ITestCOM)
  protected
    function Add(a, b: Integer): SYSINT; stdcall;
  end;

implementation

uses ComServ;

function TTestCOM.Add(a, b: Integer): SYSINT;
```

```
begin

end;

initialization
  TTypedComObjectFactory.Create(ComServer, TTestCOM, Class_TestCOM,
    ciMultiInstance, tmApartment);
end.
```

5. Registration des COM–Servers

Will man das Interface auch gleich nutzen, um es zum Beispiel zu testen kann man es auch über dem Typbibliotheken–Editor registrieren (siehe dazu Abbildung 5.3, Seite 17, Rahmen 2).

Mit Schaltfläche 3 kann man sich dann schliesslich noch eine IDL-Datei erzeugen lassen:

```
[
  uuid(F94B54FB-BF00-436D-89FC-14026CDE8A55),
  version(1.0),
  helpstring("Project2 Bibliothek")

]
library Project2
{

  importlib("STDOLE2.TLB");
  importlib("stdvcl40.dll");

  [
    uuid(EB80AEC8-5E7F-43B6-8B18-2D718ED653EA),
    version(1.0),
    helpstring("Schnittstelle für TestCOM Objekt"),
    oleautomation
  ]
   interface ITestCOM: IUnknown
  {
    [
    id(0x00000001)
    ]
    int _stdcall Add([in] long a, [in] long b );
  };

  [
    uuid(71027821-9B64-4D9C-9ED1-0D07B831C03F),
    version(1.0),
    helpstring("TestCOM")
  ]
  coclass TestCOM
  {
    [default] interface ITestCOM;
  };
```

```
};
```

A. Anhang

A.1. Weiterführende Literatur

Loos, Peter: Go to COM. Addison-Wesley, Auflage: 1. Aufl. (15. November 2000). ISBN 3827316782

Kosch, Andreas: COM/ DCOM/ COM+ mit Delphi. Software & Support, Auflage: 2. Aufl. (Dez. 2000). ISBN 3935042019