Komponenten-orientierte Programmierung

Component Object Model

Problem: Benutzung von Programmteilen durch andere Programme.

(1) Prozeduren bzw. Module im Quellcode
Nachteile:
- Nur für jeweils eine Programmiersprache geeignet
- Quellcode wird offen gelegt

(2) Module im Bindecode
Nachteil: Systemabhängig und z.T. sogar Versionsabhängig (z.B. TPU Module in Turbo-Pascal)

(3) DLL Konzept von Windows bzw. Dynamisch gebundene Bibliotheken bei UNIX
Vorteile:
- Binäre Schnittstelle
- DLL kann zur Laufzeit geladen werden. Ein System kann nachträglich um zusätzliche DLLs erweitert werden.

Nachteile:
- Systemabhängig
- Programm und DLL müssen im gleichen Prozess laufen. Keine Client-Server Architektur möglich
- Die DLL stellt nur Prozeduren zur Verfügung, keine Objekte.
- Keine Überprüfung der Parameterverträglichkeit zwischen Deklaration in der DLL und Benutzung.

"DLL" ist die Abkürzung für Dynamic Link Library, also eine Programmbibliothek, die erst zur Laufzeit in eine Anwendung eingebunden wird. Eine DLL enthält Prozeduren und Funktionen. Sie kann aus mehreren Units bestehen, und es ist sogar möglich, Formulare ("Fenster") zu verwenden.

Und wozu braucht man das nun? Eine DLL hat folgende Vorteile:

- Prozeduren, die von mehreren Anwendungen genutzt werden, können in DLLs ausgelagert werden. Die einzelnen Anwendungen werden dadurch kleiner, und der gemeinsam genutzte Code muss nur noch an einer Stelle gepflegt werden.

- Ein Programmteil, der sich des öfteren ändert, kann in eine DLL ausgelagert werden, damit bei Updates nur noch eine kleine DLL weitergegeben muss.
- Prozeduren und Funktionen in einer DLL können auch von anderen Programmiersprachen aus genutzt werden. So kann man aus einer C++-Anwendung auf eine Delphi-DLL zugreifen und umgekehrt.

```
library Project1;

uses
  SysUtils,
  Classes;

{$R *.RES}

function addiere(zahl1, zahl2: integer): integer;
stdcall;
begin
  result:=zahl1+zahl2;
end;

exports
  addiere;

begin

end.
```

Wenn es möglich sein soll, dass Funktionen oder Prozeduren einer DLL auch von anderen Programmiersprachen genutzt werden können, muss das reservierte Wort **stdcall** hinter den Funktions- bzw. Prozedurkopf geschrieben werden. In unserm Fall also:
function addiere(zahl1, zahl2: integer): integer; **stdcall**;

Bei stdcall handelt es sich um eine sog. "Aufrufkonvention". Weitere sind register, cdecl und safecall. Sie unterscheiden sich darin, wie Parameter übergeben werden und ob dabei CPU-Register verwendet werden. Für Aufrufe der Windows-API wird stdcall verwendet; register ist das effizienteste. Auf jeden Fall muss darauf geachtet werden, dass sowohl in der DLL auch beim Aufruf aus einer Anwendung die gleiche Aufrufkonvention

verwendet wird.

Möglichkeiten der Einbindung

Die Einbindung zum Startzeitpunkt (Load-Time Dynamic Linking) ist einfach zu handhaben, hat jedoch den Nachteil, dass gleich zu Programmstart alle in das Programm auf diese Art eingebundenen DLLs mitgeladen werden, auch wenn sie vielleicht nicht benötigt werden. Dies braucht Speicherplatz und verlängert den Startvorgang der Anwendung. Ist eine DLL nicht vorhanden, tritt gleich beim Start ein Fehler auf.

Zur Laufzeit engebundene DLLs (Run-Time Dynamic Linking) werden dagegen nur dann in den Arbeitsspeicher geladen, wenn sie auch wirklich gebraucht werden. Man kann also noch bis kurz vor den Aufruf den Namen der DLL festlegen und prüfen, ob diese auf dem System auch vorhanden ist. Diese Einbindungsart hat den Nachteil, dass sie komplizierter zu programmieren ist, da sich der Entwickler selber um den Ladevorgang und das anschließende Entfernen der DLL aus dem Arbeitsspeicher kümmern muss.

Einbinden zum Ladezeitpunkt

```
unit Unit1;

interface
  function addiere(zahl1, zahl2: integer):
integer; stdcall;

implementation

function addiere(zahl1, zahl2: integer): integer;
stdcall;
external 'rechnen.dll';

end.
```

Im Implementationsteil schreiben man statt der eigentlichen Funktion aber nur noch "external" gefolgt von dem Namen der DLL. Die DLL sollte sich dazu im gleichen Verzeichnis wie die Anwendung befinden. Sollen mehrere Anwendungen aus

verschiedenen Verzeichnissen auf eine DLL zugreifen können, ist sie am besten im System-Verzeichnis (z.B. C:\Windows\System) aufgehoben.

Verwendet wird die Funktion anschließend ganz normal. Überall, wo obige Unit "Unit1" über "uses" eingebunden ist, kann geschrieben werden:

```
  x:=addiere(12, 3);
```

Einbinden zur Laufzeit

```
unit Unit1;

interface

uses windows;

type
  TSummenFunktion = function(zahl1, zahl2:
integer): integer; stdcall;
  function addieren(zahl1, zahl2: integer):
integer;

implementation

function addieren(zahl1, zahl2: integer):
integer;
var SummenFunktion: TSummenFunktion;
    Handle: THandle;
begin
  Handle:=LoadLibrary(PChar(ExtractFilePath
  (ParamStr(0))+'rechnen.dll'));
  if Handle <> 0 then begin
    @SummenFunktion :=
    GetProcAddress(Handle, 'addiere');
    if @SummenFunktion <> nil then begin
      result:=SummenFunktion(zahl1, zahl2);
    end;
    FreeLibrary(Handle);
  end;
end;

end.
```

Die Einbindung zur Laufzeit etwas aufwändiger ist als die zum Programmstart. Allerdings hat sie den Vorteil, dass man den Namen der Bibliothek oder der Funktion erst zur Laufzeit festlegen muss.

COM Modell

Das COM Modell erweitert die DLL um Objekte und erlaubt COM Objekte in anderen Prozessen und auch auf anderen Rechnern (DCOM)
Ähnliche Konzepte:

- CORBA
- Remote Procedure Calls (RPC)
- JAVA Enterprise Beans

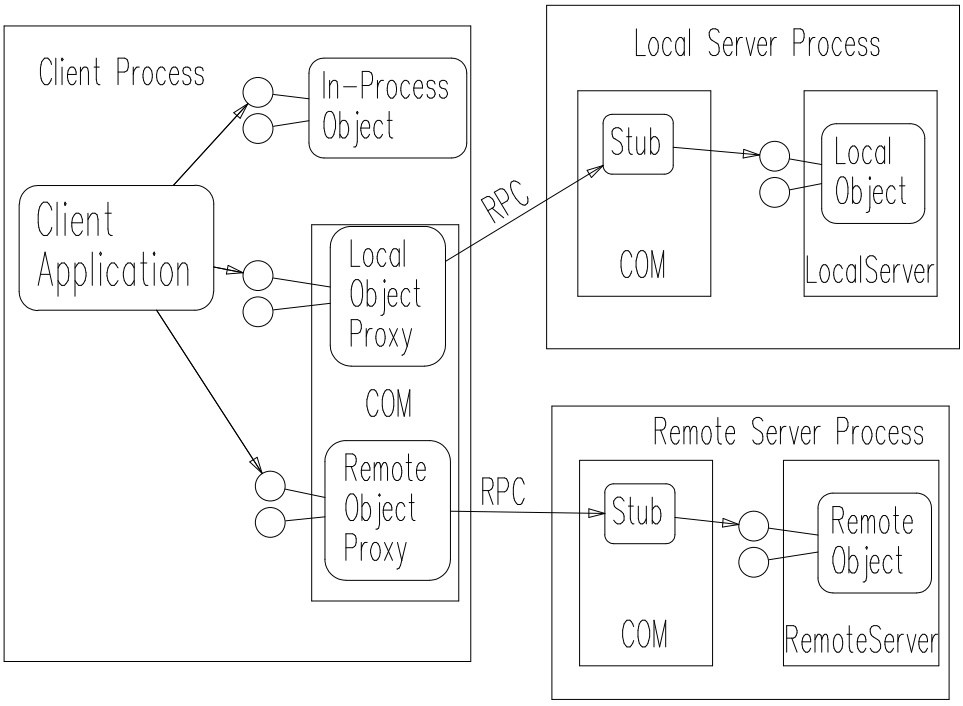
COM Objekte bilden Server, die von anderen Programmen, Clients, genutzt werden können.

Servertypen:

InProcessServer:
Das COM Objekt wird in den Prozeß eingebunden.
Realisiert durch eine DLL

LocalServer:
Das COM Objekt läuft in einem anderen Prozeß, jedoch auf dem gleichen Rechner ab.
Realisiert durch eine EXE Datei auf gleichem Rechner

RemoteServer:
Das COM Objekt läuft in einem anderen Prozeß auf einem anderen Rechner ab.
Realisiert durch eine EXE Datei auf einem anderen Rechner.



Ziel: Für die Programmierung des Client ist es unerheblich, von welcher Art das COM Objekt ist.

Probleme:

Versioning: Da der Client und der Server unabhängige Programme sind, ist eine Änderung der Schnittstelle problematisch.
Lösung: Eine Schnittstelle kann nicht geändert werden, in einem Objekt können jedoch neue Schnittstellen hinzugefügt werden.

Eindeutige Namensvergabe: Da verschiedene Objekte von verschiedenen Leuten geschrieben werden, können unbeabsichtigt Namenskonflikte auftreten.
Lösung: Kein lesbarer Name sondern eine automatisch erzeugte 128 Bit Zahl.

Der Zugriff auf die Objekte erfolgt über **Interfaces**

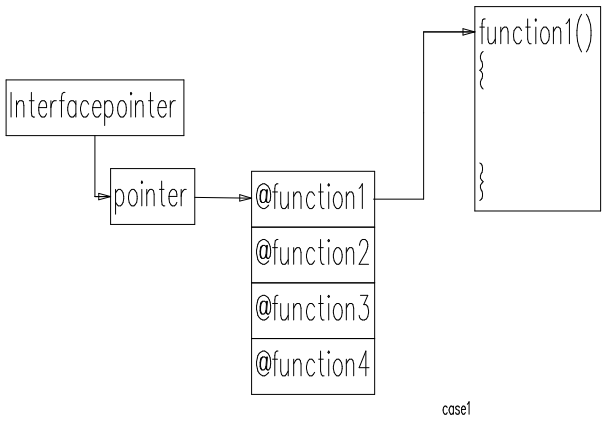
Ein Interface ist ein Zeiger auf eine Methodentabelle.
Ein Interface hat einige Eigenschaften einer Klasse:

- Methoden

jedoch nicht:

- Konstruktor, Destruktor, Daten

Aufbau eines Interface:



case1

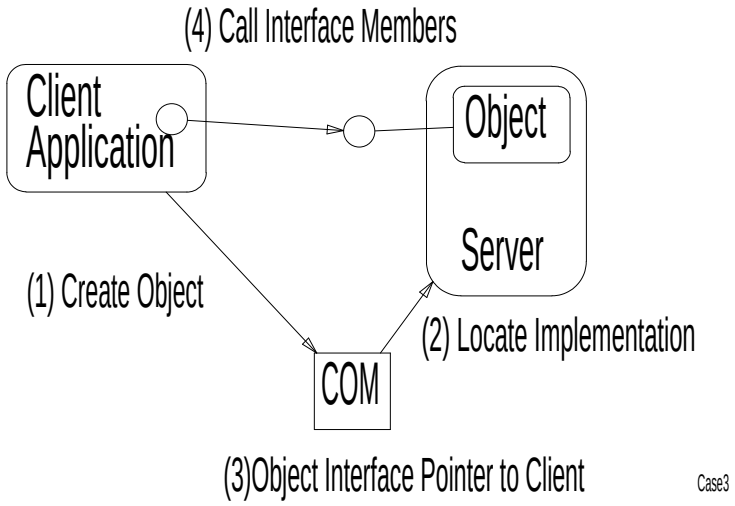
Bei InProcessServer erfolgt der Zugriff auf die COM Objekte wie auf C++ Objekte mit virtuellen Methoden
In den anderen Fällen stellt die COM Bibliothek automatisch die Proxys und Stubs zur Verfügung und realisiert die RPC Zugriffe.

Die Speicherverwaltung von Client und Server sind getrennt.
Funktion: CoGetMalloc

Parameter: In, Out, In-Out
- In: Parameter Allokation durch den Aufrufer
- Out: Parameter Allokation durch die Funktion, Freigabe durch Aufrufer
- In-Out: Am Anfang allokiert durch den Aufrufer, während des Aufrufs evtl. Freigegeben und neu allokiert durch die Funktion.

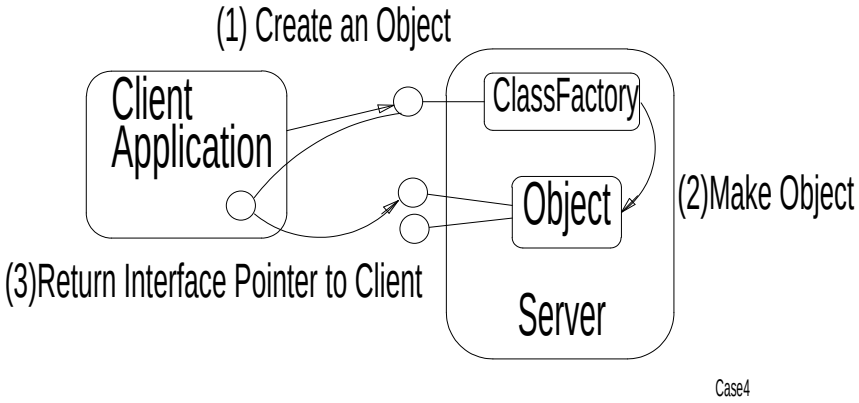
COM Client

Erzeugung und Zugriff auf COM Objekte



Die Suche der Implementierung des Servers erfolgt über die Registry.
[HKEY_CLASSES_ROOT\CLSID\{BAB686E2-9E33-11D1-A3C9-0020AFF35E36}]
[HKEY_CLASSES_ROOT\CLSID\{BAB686E2-9E33-11D1-A3C9-0020AFF35E36}\LocalServer32]
@="C:\\COM\\MY\\MYSERVER.EXE"

Die Erzeugung der COM Objekte erfolgt häufig über das Interface **`Iclassfactory`**



hr:=CoGetClassObject(clsid, grfContext, PServerInfo, iid, ppv)

clsid: Identifier der Class Factory
grfContext: (inprocserver, localserver, remoteserver)
Pserverinfo: nil bzw. Zeiger auf den Server bei RemoteServer
iid: Identifier des gewünschten Interfaces
ppv: Zeiger auf das FactoryInterface

Der clsid ist der Bezeichner der Klasse und wird in der Registry gesucht. Dort ist der entsprechende Server angegeben, der dann von der COM Bibliothek gestartet oder aufgerufen wird. Der Server erzeugt entweder ein neues Objekt oder stellt die Adresse eines vorhandenen zur Verfügung.

Wenn man ppv hat kann man damit Objekte erzeugen

hr:=ppv.CreateInstance(pUnkOuter, iid, ppvObject)
pUnkOuter: nil oder Zeiger auf ein Umgebendes Objekt in dem das neue Objekt erzeugt werden soll.
iid: Identifier des gewünschten Interfaces
ppvObject: Zeiger auf das Interface

Der Identifier des Interface ist dem Objekt bzw. dem Factory Interface bekannt.

Beide Aufrufe können auch zusammen gefaßt werden:

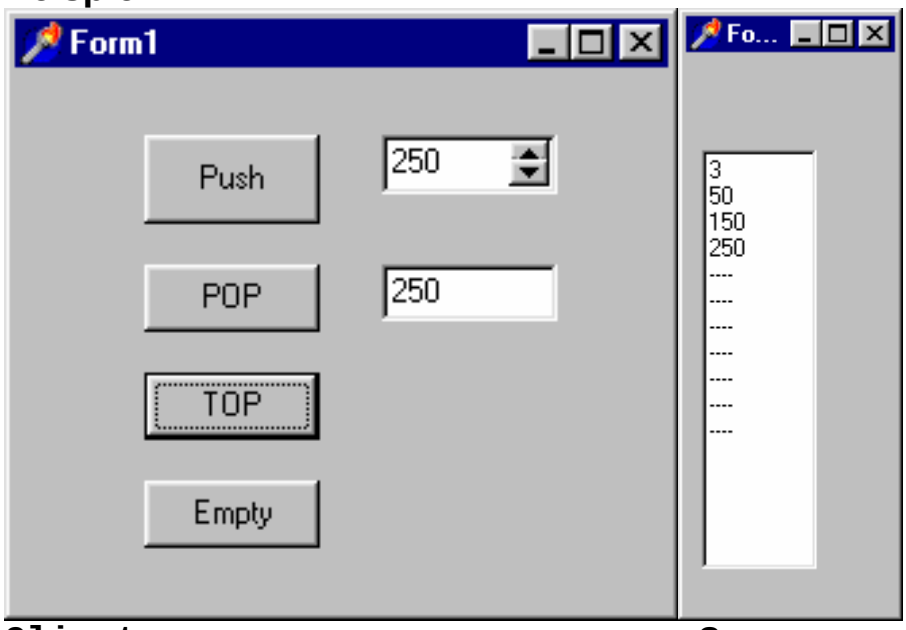
hr:=CoCreateInstance(clsid, pUnkOuter, grfContext, iid, ppvObject)

Es wird mit der Factory clsid ein Interface iid erzeugt und auf ppvObject übergeben

Wenn man ein Interface eines Objekts hat, kann man eine Methode dieses Interface aufrufen.

PpvObject.push(myvalue);

Beispiel:



Client **Server**

Fragen:
1. Wie werden bei einem nicht InProcessServer die Parameter transferiert zwischen Client und Server (Marshalling)?
2. Wie kommt man an die Namen der Methoden?

```
{Typbibliothek}
const
  LIBID_stackserver: TGUID = '{BAB686E0-9E33-
11D1-A3C9-0020AFF35E36}';
const
{ Komponentenklassen-GUIDs }
  Class_stackserver: TGUID = '{BAB686E2-9E33-
11D1-A3C9-0020AFF35E36}';
type
Istackserver = interface(IUnknown)
  ['{BAB686E1-9E33-11D1-A3C9-0020AFF35E36}']
  procedure push(x: Integer); safecall;
  function pop: Integer; safecall;
  function top: Integer; safecall;
  function empty: Integer; safecall;
end;

var mystack:Istackserver;

hr:=CoCreateInstance(Class_stackserver, nil,
CLSCTX_INPROC_SERVER or CLSCTX_LOCAL_SERVER,
Istackserver, mystack);

mystack.push(eingabe.value);
```

Falls der Stackserver ein InProcessServer ist entfällt Problem (1). Push wird wie eine virtuelle Methode über die Methodentabelle aufgerufen. Die Typbibliothek wird nicht benötigt.

Sonst wird (1) über die OLE-Automation (OLEAUT32.DLL) gelöst.

[HKEY_CLASSES_ROOT\Interface\{BAB686E1-9E33-11D1-A3C9-0020AFF35E36}]
@="Istackserver"
[HKEY_CLASSES_ROOT\Interface\{BAB686E1-9E33-11D1-A3C9-0020AFF35E36}\ProxyStubClsid]
@="{00020424-0000-0000-C000-000000000046}"
[HKEY_CLASSES_ROOT\Interface\{BAB686E1-9E33-11D1-A3C9-0020AFF35E36}\ProxyStubClsid32]
@="{00020424-0000-0000-C000-000000000046}"
[HKEY_CLASSES_ROOT\Interface\{BAB686E1-9E33-11D1-A3C9-0020AFF35E36}\TypeLib]
@="{BAB686E0-9E33-11D1-A3C9-0020AFF35E36}"
"Version"="1.0"

Bei ProxyStubClsid ist eingetragen:

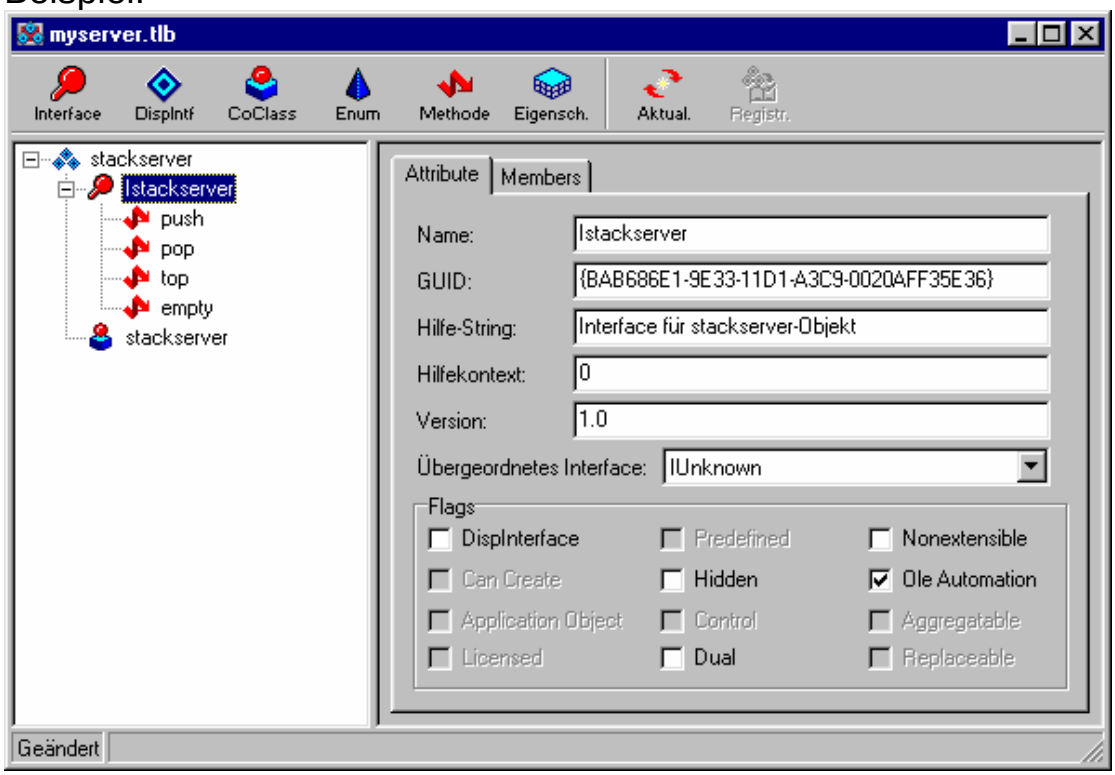
[HKEY_CLASSES_ROOT\CLSID\{00020424-0000-0000-C000-000000000046}]
@="PSAutomation"
[HKEY_CLASSES_ROOT\CLSID\{00020424-0000-0000-C000-000000000046}\InprocServer32]
@="oleaut32.dll"
"ThreadingModel"="Both"

Bei der TypeLib ist eingetragen:

[HKEY_CLASSES_ROOT\TypeLib\{BAB686E0-9E33-11D1-A3C9-0020AFF35E36}]
[HKEY_CLASSES_ROOT\TypeLib\{BAB686E0-9E33-11D1-A3C9-0020AFF35E36}\1.0\0\win32]
@="C:\\COM\\MY\\MYSERVER.EXE"

Ruft der Client eine Methode auf geht sie an den eingebundenen Proxy. Dieser sucht in der Registry das Programm mit der Typbibliothek, i.A. der Server, und ruft dort die Typbibliothek (COM Objekt) auf. Über die Typbibliothek erfährt er die Parametertypen der Methode und kann die Daten an den im Server eingebundenen Stub übermitteln. Dieser ruft dann das Interface mit geeigneten Parametern als virtuelle Methode auf.
D.h. der Benutzer kommt mit Proxy und Stub nicht in Berührung. Die Softwareentwicklungsumgebung muß jedoch ein Werkzeug besitzen, daß das Typbibliotheksobjekt erzeugt.

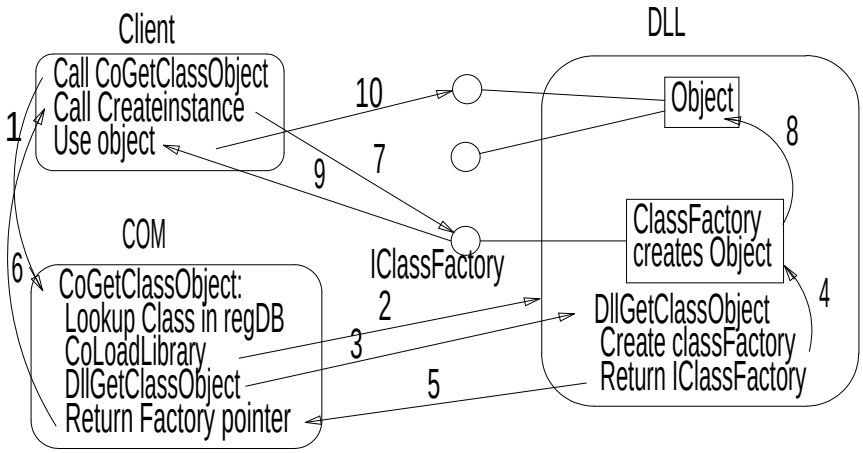
Ein solches Werkzeug ist z.B. in Delphi enthalten, es erzeugt auch die Pascal- Source für die Interface Definition der COM- Klassen. Von Microsoft gibt es Werkzeuge, die aus einer textuellen Beschreibung die Typbibliothek erzeugen.
Beispiel:



COM Server

Noch offen:
Wie werden im Server die COM Klassen bekannt gemacht.

InProcessServer:




```
var factory:tmystackfactory;

function DllGetClassObject(const CLSID, IID:
TGUID; out Obj): HResult;stdcall;
var ppv:pointer absolute obj;
begin
 factory:=tmystackfactory.create;
 ppv:=factory;
 result:=0;
end;

exports DllGetClassObject;
```

Die Definition der Klassen und Interface erfolgt in der DLL:

```
type
imystackfactory=interface(iclassfactory)
 ['{12345678-0000-0000-C000-000000000046}']

 function QueryInterface(const iid: TIID; out
obj): HResult; stdcall;
```

Mit QueryInterface kann man alle Interface des Objects bekommen. Iid ist der Identifier, obj liefert das Interface.

```
 function _AddRef: Longint;stdcall;
 function _Release: Longint;stdcall;
```

AddRef und Release dienen zur Verwaltung der Lebensdauer. Beim Erzeugen wird der Refrenzzähler auf 1 gesetzt. Kopiert man den Zeiger ruft man AddRef auf um den Referenzzähler zu erhöhen.
Mit Release gibt man den Zeiger frei,
der Refrenzzähler wird erniedrigt und bei 0 wird das Interface freigegeben.

```
 function CreateInstance(const unkOuter:Iunknown;
const iid: TIID;out obj): HResult; stdcall;
```

Liefert den Interfacepointer zum Interface iid. Falls noch nicht geschehen wird erst das Objekt erzeugt. Über unkOuter können auch hierarchische Objekte erzeugt werden.

```
end;
```

Im Server muß nicht nur das Interface sondern auch die Klassendefinition sein.

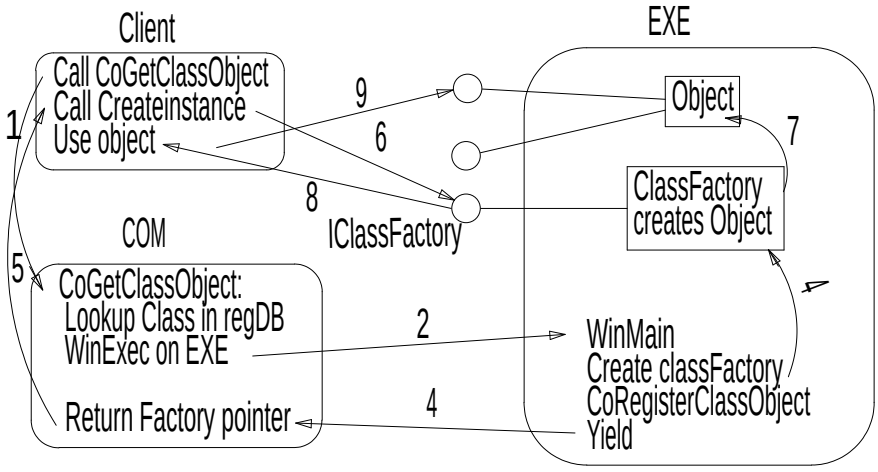
```
tmystackfactory=class(tobject, iunknown,
iclassfactory, imystackfactory)
 { IUnknown }
 function QueryInterface(const IID: TGUID; out
Obj): Integer; virtual;stdcall;
 function _AddRef: Integer; virtual;stdcall;
 function _Release: Integer; virtual;stdcall;
 { IClassFactory }
 function CreateInstance(const UnkOuter:
IUnknown; const IID: TGUID;
   out Obj): HResult; virtual;stdcall;
```

```
 function LockServer(fLock: BOOL): HResult;
virtual;stdcall;
 constructor create;
end;
```

Die Definition von CreateInstance kann wie folgt aussehen:

```
function tmystackfactory.CreateInstance(const
unkOuter: IUnknown; const iid: TIID;out obj):
HResult;
var ppv:pointer absolute obj;
begin
ppv:=nil;
if uidequal(iid,imystack)then begin
 ppv:=tmystack.create;result:=0;
end else if uidequal(iid,imytree)then begin
 ppv:=tmytree.create;result:=0;
end else result:=$99;
end;
```

LocalServer:




```
doregister( LIBID_stackserver,
Class_stackserver);
mystackfactory:=tmystackfactory.Create;
mystackfactory.initialize;
mystackserver:=tstackserver.create;
mystackserver.initialize;
CoInitialize(nil);
hr:=CoRegisterClassObject(Class_stackserver,
mystackfactory, CLSCTX_LOCAL_SERVER,
REGCLS_multipleUSE, FRegister);
if hr<>s_ok then showmessage('Fehler bei
CoRegisterClass') else Application.Run;
```

Class_stackserver ist der Identifier der Factory(Klasse).
Der Zeiger mystackfactory ist jetzt der COM Bibliothek bekannt und kann vom ServerProcess zum ClientProcess übermittelt werden. Die COM Bibliothek im Client Process reicht dann den Zeiger auf das Interface im Proxy an den Client. Die Verbindung von Proxy und Stub erfolgt von nun an automatisch durch die COM Bibliothek.

ActiveX

What is ActiveX? (nach Charlie Kindel, Microsoft):
A marketing name for a set of technologies and services, all based on the Component Object Model (COM)
It's just COM!

Also: Ein ActiveX Control (Steuerelement) ist eine COM Komponente mit folgenden Eigenschaften:

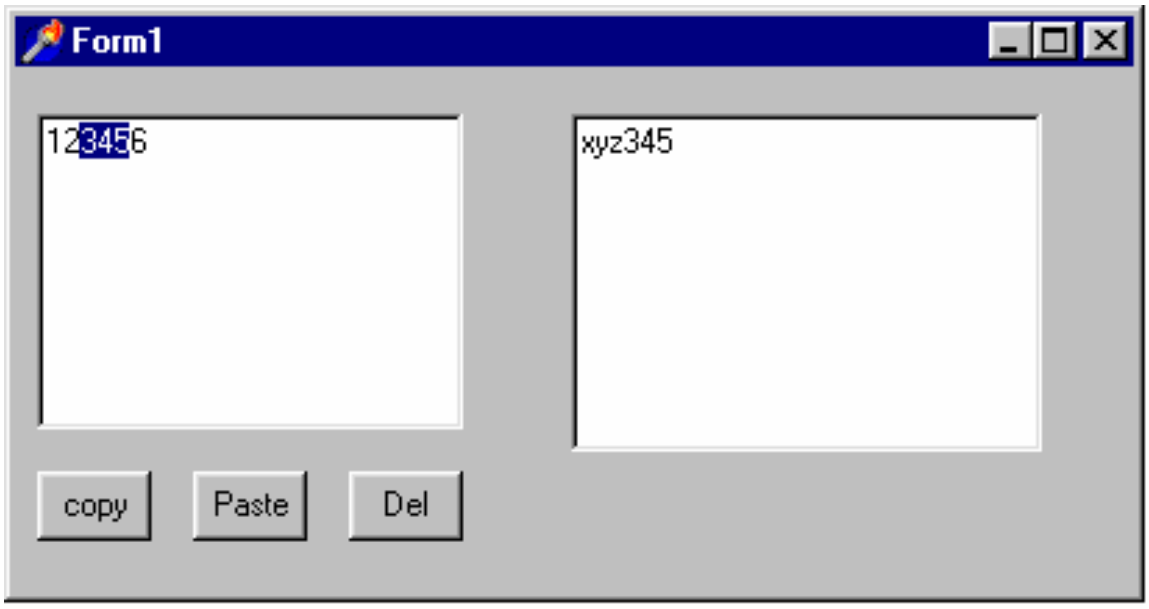
- Sie läuft als InProcessServer im Clienten z.B. im Web-Browser
- Sie hat ein „Design-Time" User-Interface.

Erstellen eines ActiveX Control:
VCL (Visual Component Library) Element erstellen. Z.B. ein Editor (TMemo).

```
Type TMemoX = class(TActiveXControl, IMemoX)
private
 { Private-Deklarationen }
 FMemo: TMemo;
protected
 { Protected-Deklarationen }
 ……
 procedure write(const x: WideString); safecall;
 function read: WideString; safecall;
 procedure delete; safecall;
end;

procedure TMemoX.write(const x: WideString);
begin
with fmemo do begin
 text:=text+x;
 hideselection:=false;
 repaint;
end;
end;
function TMemoX.read: WideString;
begin
result:=fmemo.seltext;
fmemo.hideselection:=false;
end;
procedure TMemoX.delete;
```

```
begin
fmemo.seltext:='';
end;
```

Beispiel für einen Client, der TMemoX benutzt:

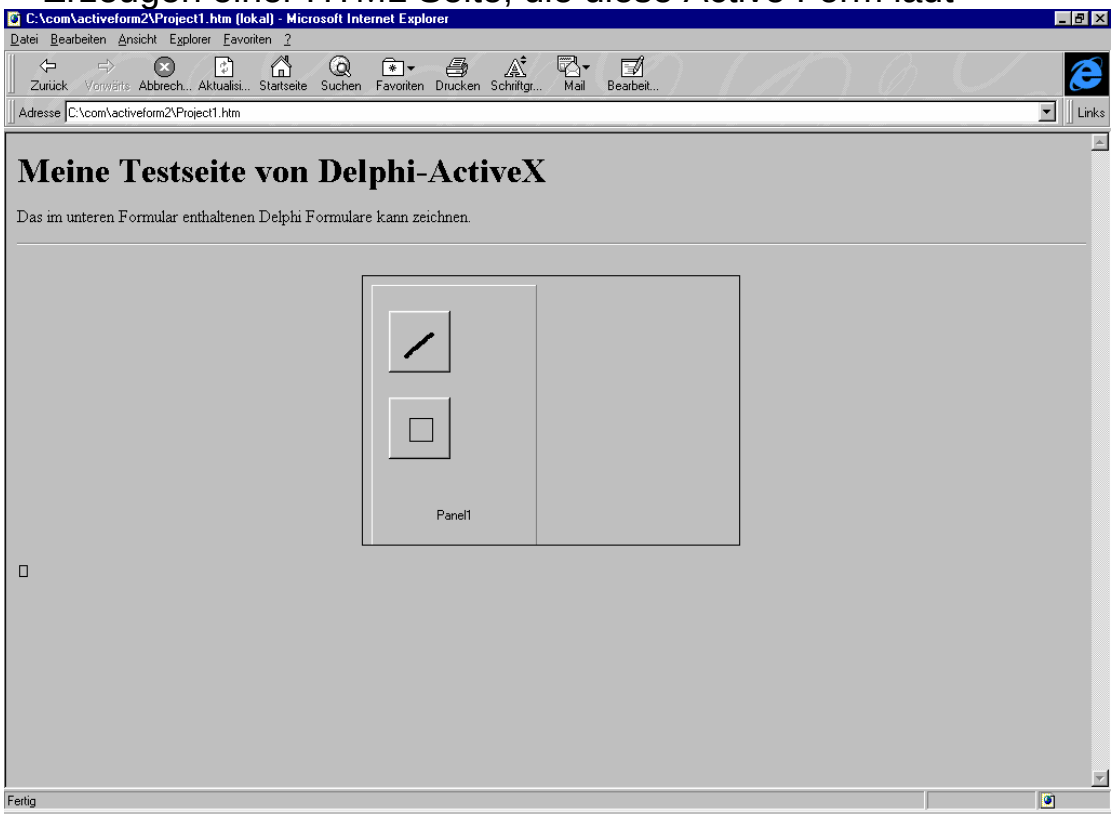



```
type TForm1 = class(TForm)
 Memo1: TMemo;
 copy: TButton;
 Paste: TButton;
 Del: TButton;
 MemoX1: TMemoX;
 procedure docopy(Sender: TObject);
 procedure dopaste(Sender: TObject);
 procedure dodel(Sender: TObject);
end;
var Form1: TForm1;

procedure TForm1.docopy(Sender: TObject);
begin
with memox1 do begin
 write(memo1.seltext);
end;
memo1.hideselection:=false;
end;
```

```
procedure TForm1.dopaste(Sender: TObject);
var s:string;i,l:integer;
begin
with memo1 do begin
 hideselection:=false;
 i:=selstart;l:=sellength;
 sellength:=0;selstart:=i+l;
 s:=memox1.read;
 seltext:=s;
 selstart:=i+l;
 sellength:=length(s);
 hideselection:=false;
 rePaint;
end;
end;

procedure TForm1.dodel(Sender: TObject);
begin
memox1.delete;
end;
```

ActiveX Steuerelemente im Web bereitstellen.

- Erzeugen einer ActiveX Form
- Erzeugen einer HTML Seite, die diese Active Form lädt




```
<HTML>
<H1> Meine Testseite von Delphi-ActiveX </H1><p>
Das im unteren Formular enthaltenen Delphi
Formulare kann zeichnen.
<HR><center><P>
<OBJECT
      classid="clsid:F2286703-BF09-11D1-873C-
00009296A6DE"

codebase="c:\com\activeform2\Project1.ocx#version
=1,0,0,0"
      width=350
      height=250
      align=center
      hspace=0
      vspace=0
>
</OBJECT>
</HTML>
```

Die ActiveXForm kann wie folgt deklariert werden:

```
type TActiveFormX = class(TActiveForm,
IActiveFormX)
 Panel1: TPanel;
 SpeedButton1: TSpeedButton;
 SpeedButton2: TSpeedButton;
 procedure speedbutoon1click(Sender: TObject);
 procedure speedbutton2click(Sender: TObject);
 procedure formmousemove(Sender: TObject; Shift:
TShiftState; X,Y: Integer);
 procedure formmousedown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
 procedure formmouseup(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
 procedure formcreate(Sender: TObject);

type ttool=(none,line,rectangle);
var tool:ttool;
 enable:boolean;
 startx,starty,previousx,previousy:integer;

procedure TActiveFormX.speedbutoon1click(Sender:
TObject);
begin
tool:=line;
end;

procedure TActiveFormX.speedbutton2click(Sender:
TObject);
begin
tool:=rectangle;
canvas.brush.style:=bsclear;
end;
```

```
procedure TActiveFormX.formmousemove(Sender:
TObject; Shift: TShiftState;
  X, Y: Integer);
begin
if enable then begin
 canvas.pen.mode:=pmnotxor;
 case tool of
  rectangle: begin
   canvas.rectangle( startx, starty, previousx,
previousy);
   canvas.Rectangle(startx, starty, x, y);
  end;
  line:begin
   canvas.moveto( startx, starty);
   canvas.lineto( previousx, previousy);
   canvas.moveto( startx, starty);
   canvas.LineTo( x, y);
  end;
 end;
 previousx:=x;previousy:=y;
end;
canvas.Pen.mode:=pmcopy;
end;

procedure TActiveFormX.formmousedown(Sender:
TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
startx:=x;starty:=y;previousx:=x;previousy:=y;
canvas.MoveTo(x,y);
enable:=true;
end;

procedure TActiveFormX.formmouseup(Sender:
TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
enable:=false;
case tool of
 line:canvas.lineto(x,y);
 rectangle:canvas.Rectangle(startx,starty,x,y);
end;
end;
```

```
procedure TActiveFormX.formcreate( Sender:
TObject);
begin
tool:=none;enable:=false;
end;
```

Das ganze wird in folgende DLL eingebettet:

```
library Project1;
uses
 ComServ,
 Project1_TLB in 'Project1_TLB.pas',
 ActiveFormImpl1 in 'ActiveFormImpl1.pas'
 {ActiveFormX: TActiveForm} {ActiveFormX:
CoClass};

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.TLB}
{$R *.RES}
{$E ocx}
begin
end.
```