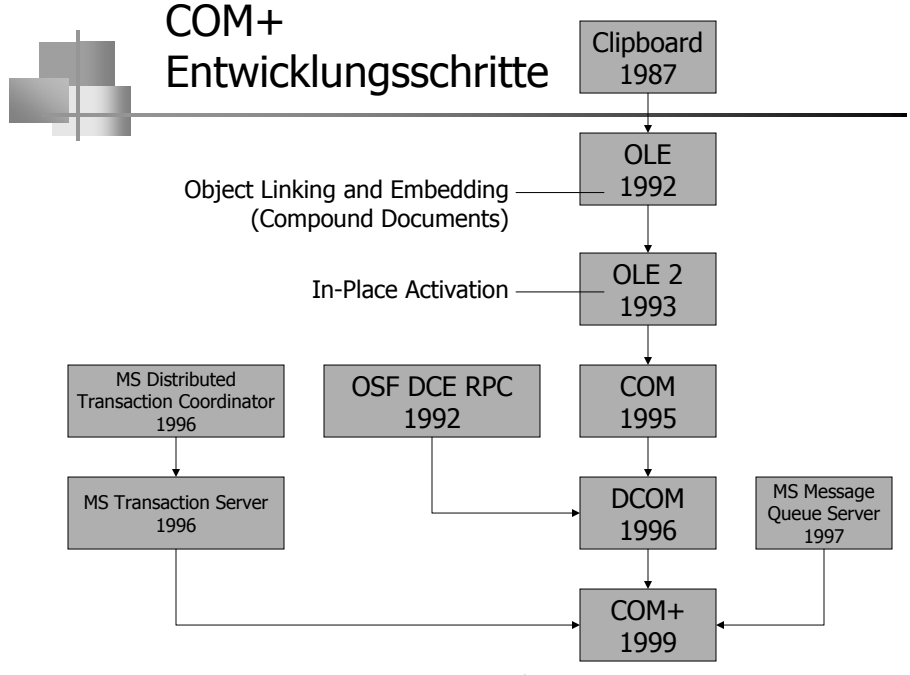
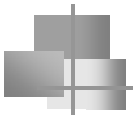
OLE, DCOM und COM

Die frühen Komponentenmodelle
der Firma Microsoft

COM+ Entwicklungsschritte

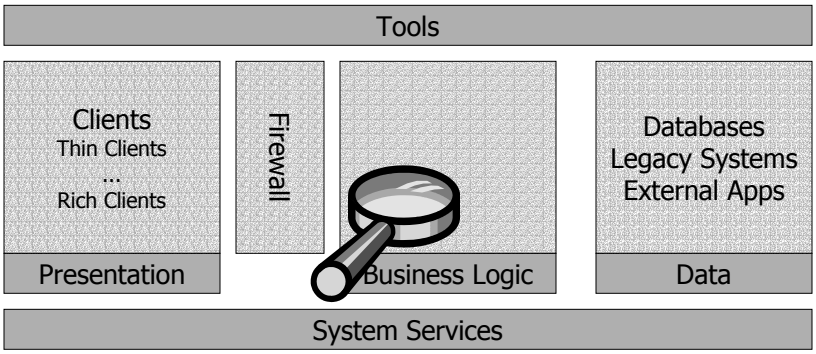


OLE, DCOM und COM+ 2

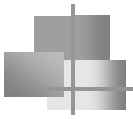


Windows DNA

- DNA = Distributed interNet Application Architecture
- 3-Tier Ansatz



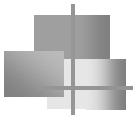
OLE, DCOM und COM+ 3



Component Services

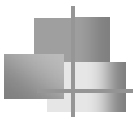
- COM
  - Interface-based Programming
  - Basic Component Facilities
- DCOM
  - Remoting Architecture
  - Distributed Component Services
- COM+
  - Load Balancing
  - In-memory Database
  - Object Pooling
  - Queued Components
  - Event Model
- MTS
  - Transaction Services
  - Resource Pooling
  - Role-based Security
  - Administration
  - Just-in-time Activation

OLE, DCOM und COM+ 4



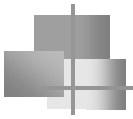
Basics

- Binärer Komponentenstandard
  - Implementierungssprache unbestimmt
- Komponente realisieren Interfaces
- Interface = Schnittstellenbeschreibung
  - Abstrakte C++-Klasse mit virtuellen Funktionen
  - vtbl definiert Schnittstelle
- Minimal
  - **`IUnknown`**: Elementarmethoden jeder Komponente
  - **`IClassFactory`**: Komponentenerzeugung



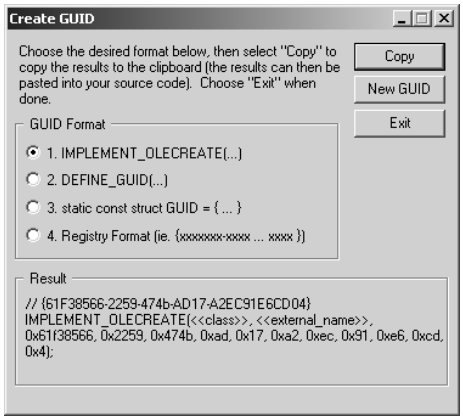
Identifikation von Komponenten

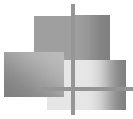
- Interfaces besitzen eine GUID
- 128 Bit „Zufallszahl”
  - z.B. IUnknown {00000000-0000-0000-C000-000000000046}
  - Erzeugung über API: **`CoCreateGuid()`**
  - Erzeugungstool: guidgen.exe
- Registry = Namensdienst



guidgen.exe

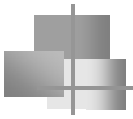
- Erzeugung eigener GUIDs
  - Hinreichend zufällig
- Bestandteile
  - Aktuelles Datum
  - Aktuelle Uhrzeit
  - Fortlaufende „Clock Sequence" (persistent)
  - Inkrementeller Zähler (hochfrequente Abfragen)
  - MAC-Adresse der Netzkarte (geht aber auch ohne)





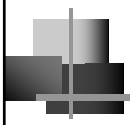
IUnknown

```
interface IUnknown
{
    typedef [unique] IUnknown *LPUNKOWN;

    HRESULT QueryInterface (
        [in] REFIID riid,
        [out, iid_is(riid)] void **ppvObject );

    ULONG AddRef ();

    ULONG Release ();
}
```

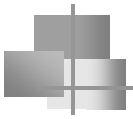


IClassFactory

```
interface IClassFactory: IUnknown
{
    HRESULT CreateInstance (
        [in, unique] IUnknown *pUnkOuter,
        [in] REFIID riid,
        [out, iid_is(riid)] void **ppvObject);

    HRESULT LockServer ( [in] BOOL flock );
}
```

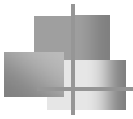



Erste COM-Schritte



Beispiel

- Aufgabe
  - Übertragung von n ganzen Zahlen
  - „Komponente" bildet die Summe
- Schritte
  1. Interface definieren
  2. Header-Dateien generieren
  3. Komponente implementieren
  4. Client programmieren
  5. Registry-Eintrag



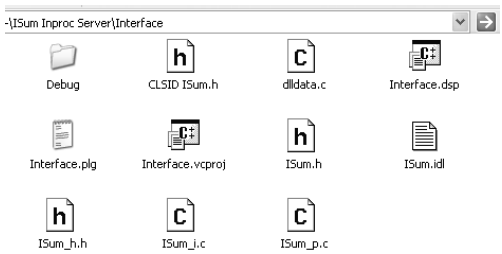
1. Interface

- Utility-Projekt in Visual Studio
- Interface **`Isum.idl`**:

```
import "unknwn.idl";

[ object,
uuid(1E10C200-E306-11D3-A557-B6EEE489CA00)
]

interface ISum : IUnknown {

    HRESULT AddElements (
           [in] int nElem,
           [in, size_is(nElem)] int *elements,
           [out] int *sum
    );

};
```

2. MIDL ausführen

- Mehrere Dateien werden generiert
- Bedeutung:
  - C++-Version des Interfaces
  - GUID-Definitionen
  - Proxy-Stubs für entfernte Server



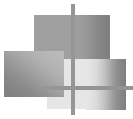
OLE, DCOM und COM+ 13

3. Komponente implementieren

- Jedes Interface der Komponente muß implementiert werden
  - Ableitungen von den abstrakten Basisklassen
  - „Pure virtual functions" implementieren
- Virtuelle Funktionen müssen immer wieder implementiert werden (Binärstandard)

Sum.cpp

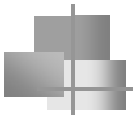
OLE, DCOM und COM+ 14



4. Client programmieren

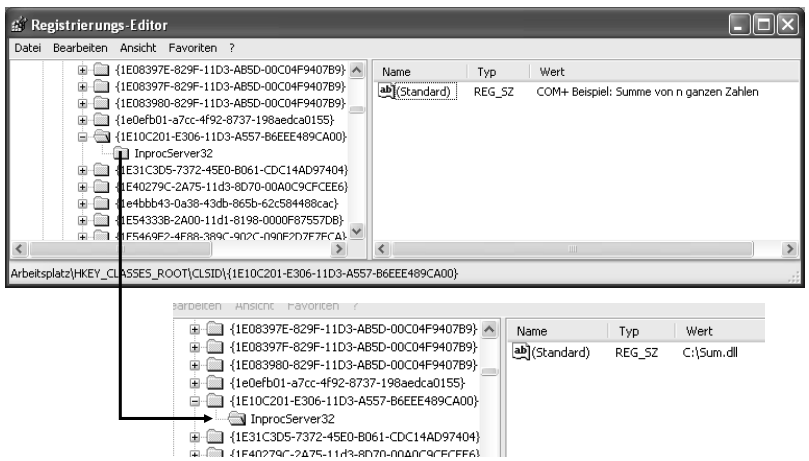
- Initialisierung des Komponentenzugriffs
- **`Iunknown`**-Interface über GUID erfragen
- Downcast zum gewünschten Interface
- Komponente benutzen
- Explizites Reference-Counting beachten

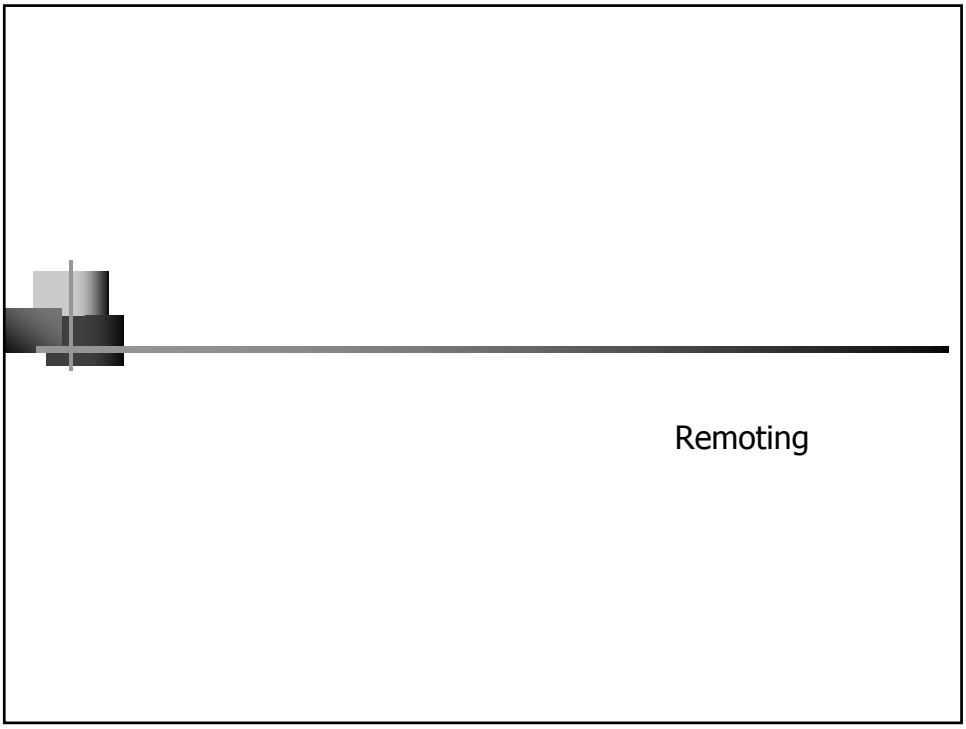
Client.cpp

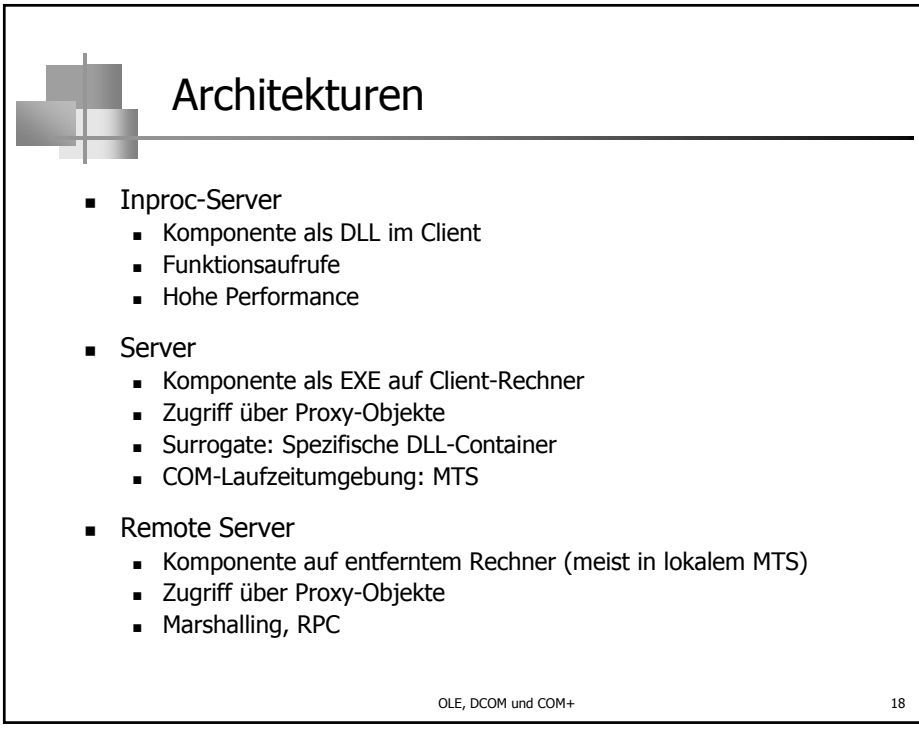


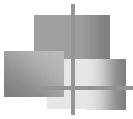
5. Registry-Eintrag

- **`HKCR/CLSID/{GUID}`**




Remoting


Architekturen
Inproc-Server
Komponente als DLL im Client
Funktionsaufrufe
Hohe Performance
Server
Komponente als EXE auf Client-Rechner
Zugriff über Proxy-Objekte
Surrogate: Spezifische DLL-Container
COM-Laufzeitumgebung: MTS
Remote Server
Komponente auf entferntem Rechner (meist in lokalem MTS)
Zugriff über Proxy-Objekte
Marshalling, RPC
OLE, DCOM und COM+
18

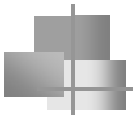


ISum: Local Server

- Expliziter Aufruf eines lokalen Servers:
  - `hr = CoCreateInstance(CLSID_ISum, NULL, CLSCTX_LOCAL_SERVER, IID_IUnknown, (void **) &pUnknown);`

  statt
  - `hr = CoCreateInstance(CLSID_ISum, NULL, CLSCTX_INPROC_SERVER, IID_IUnknown, (void **) &pUnknown);`
- Es geht aber auch transparent
  - Zuerst nach Inproc-Server schauen
  - Dann nach lokalem Server
  - Dann ggf. entfernter Server

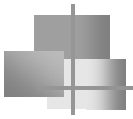
OLE, DCOM und COM+ 19



Wir verlassen die Prozeßgrenzen

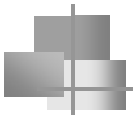
- RPC = Remote Procedure Call
  - Argumente in Nachricht verpacken
  - Nachrichtenübertragung an den Server
  - Auf Antwortnachricht warten
  - Ergebnis an den Aufrufer zurückgeben
- Marshalling und Unmarshalling
  - MIDL generiert die notwendigen Routinen automatisch
  - Proxy = Client-Seite
  - Stub = Server-Seite
- Verankerung der Proxy- und Stubroutinen in Registry

OLE, DCOM und COM+ 20



Im Beispiel

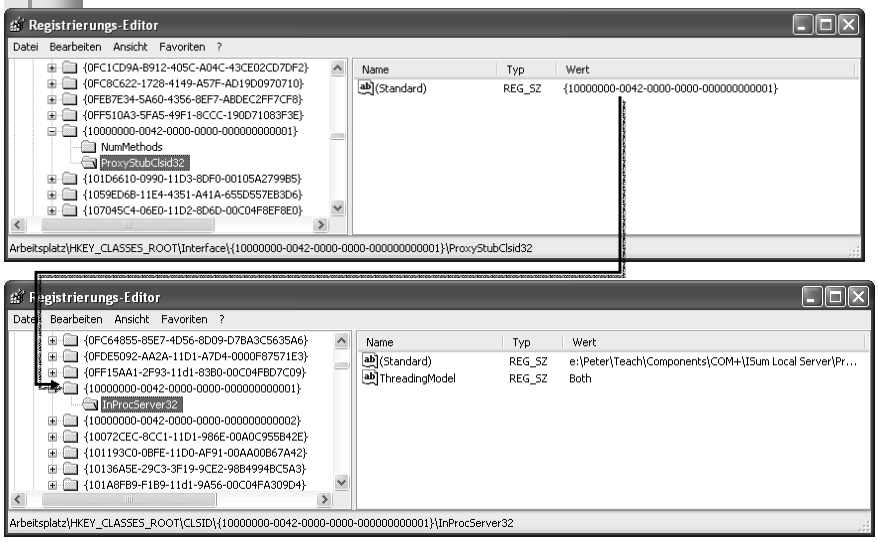
- Relevante GUIDs für das Beispiel:
- Interface ISum
  10000000-0042-0000-0000-000000000001
- Komponente CoSum (CLSID)
  10000000-0042-0000-0000-000000000002



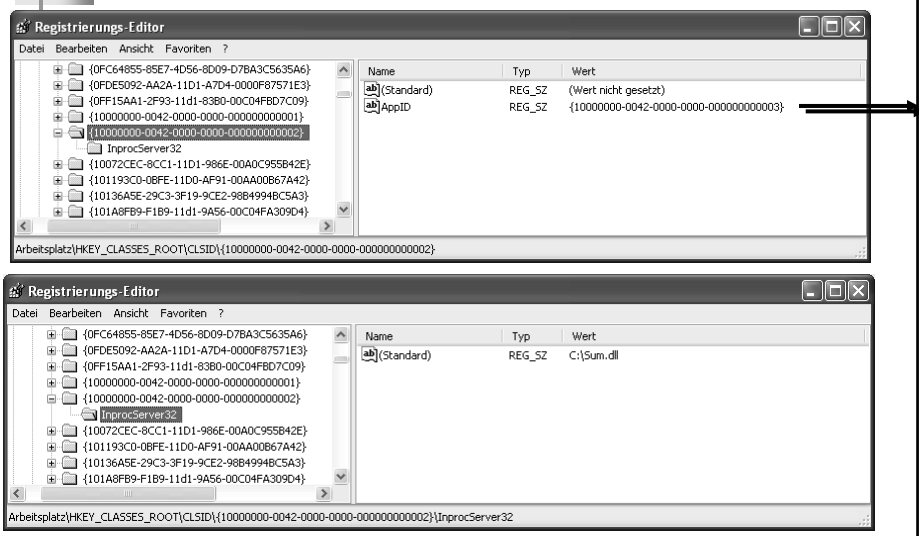
Zusätzlicher Schritt

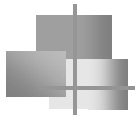
- ProxyStub-DLL erzeugen
  - Benötigt werden die von MIDL erzeugten Dateien:
    - dlldata.c
    - ISum_i.c
    - ISum_p.c

ProxyStub-DLL registrieren

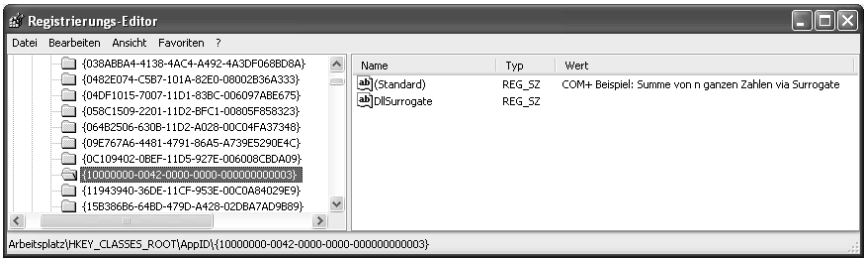


Registryeinträge für lokalen Server





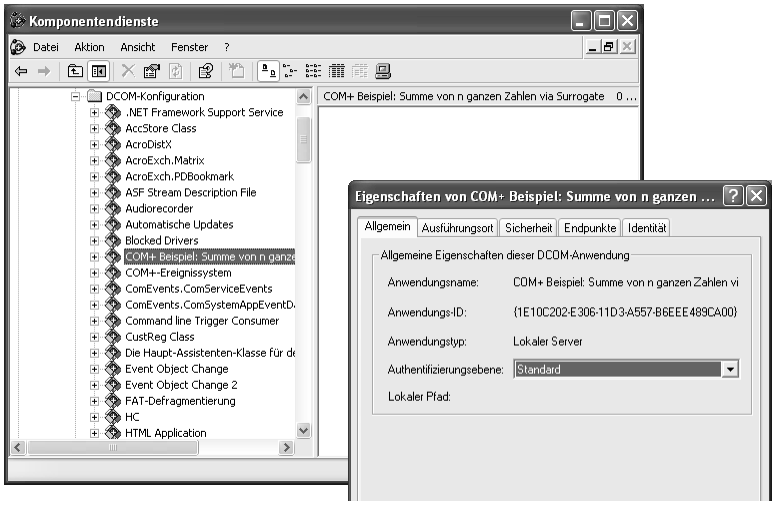
… und AppID



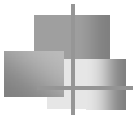
Mögliche AppID-Schlüssel

- AccessPermission
- ActivateAtStorage
- AuthenticationLevel
- DllSurrogate
- LaunchPermission
- LocalService
- RemoteServerName
- RunAs
- ServiceParameter (bei gesetztem LocalService)

dcomcnfg.exe

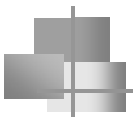


Automation



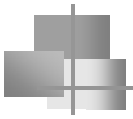
Interfacevarianten

- Statischer Zugriff
  - Interface der gewünschten Komponente bekannt
  - Zugriff über generierte Schnittstellenbeschreibung
  - Überprüfungen zur Übersetzungszeit
- Dynamischer Zugriff
  - Interfacebeschreibung liegt in Form einer Type Library vor
  - Erfragen der Schnittstelle
  - Dynamischer Methodenaufruf
  - Keine Überprüfungen zur Übersetzungszeit
  - Vgl. Introspection (Java), Dynamic Interfaces (Corba)
  - Beim Zugriff über VB, VBA oder Java notwendig



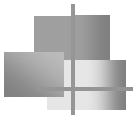
Dynamische Interfaces

- Automation
- Schnittstelle **`IDispatch`**
- Kombination aus statischem und dynamischem Interface möglich



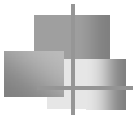
IDispatch

```
[
    object,
    uuid(00020400-0000-0000-C000-000000000046),
    pointer_default(unique)
]

interface IDispatch : IUnknown
{
    typedef [unique] IDispatch * LPDISPATCH;

    HRESULT GetTypeInfoCount(
        [out] UINT * pctinfo );

    HRESULT GetTypeInfo(
        [in] UINT iTInfo,
        [in] LCID lcid,
        [out] ITypeInfo ** ppTInfo);

    ...
```

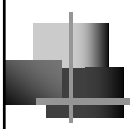
IDispatch (2)

```
HRESULT GetIDsOfNames(
    [in] REFIID riid,
    [in, size_is(cNames)] LPOLESTR * rgszNames,
    [in] UINT cNames,
    [in] LCID lcid,
    [out, size_is(cNames)] DISPID * rgDispId );

[local]
HRESULT Invoke(
    [in] DISPID dispIdMember,
    [in] REFIID riid,
    [in] LCID lcid,
    [in] WORD wFlags,
    [in, out] DISPPARAMS * pDispParams,
    [out] VARIANT * pVarResult,
    [out] EXCEPINFO * pExcepInfo,
    [out] UINT * puArgErr );
```



IDispatch (3)

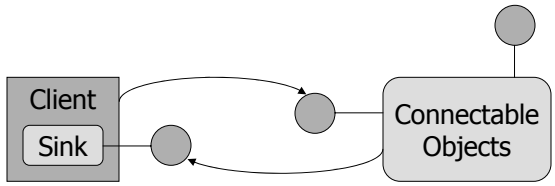
```
[call_as(Invoke)]
    HRESULT RemoteInvoke(
        [in] DISPID dispIdMember,
        [in] REFIID riid,
        [in] LCID lcid,
        [in] DWORD dwFlags,
        [in] DISPPARAMS * pDispParams,
        [out] VARIANT * pVarResult,
        [out] EXCEPINFO * pExcepInfo,
        [out] UINT * pArgErr,
        [in] UINT cVarRef,
        [in, size_is(cVarRef)] UINT * rgVarRefIdx,
        [in, out, size_is(cVarRef)]
            VARIANTARG * rgVarRef
    );
}
```

Extras

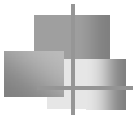
Connection Points

- Rückrufbare Client-Objekte
- Connectable Objects



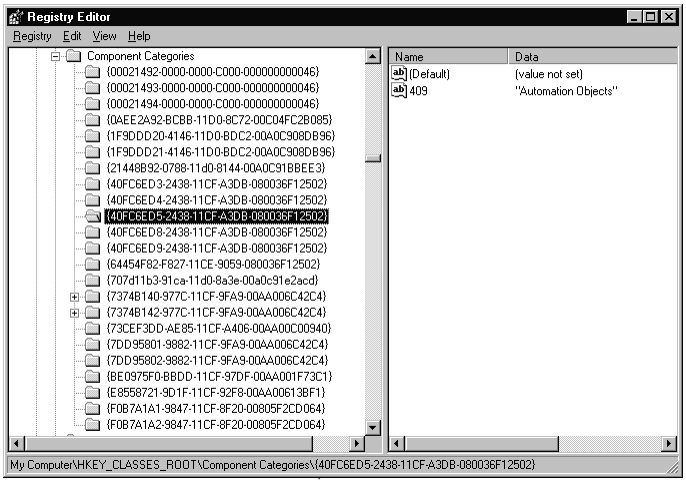
Moniker

- Moniker benennen andere Objekte
- Varianten
  - FileMoniker - Wrapper für Pfadname einer Datei
  - ItemMoniker - Identifiziert Objekt innerhalb eines Objekts
  - PointerMoniker - Verweis auf ein aktives Objekt
  - AntiMoniker
  - CompositeMoniker
  - ClassMoniker - Wrapper für CLSID einer COM-Klasse
  - URL Moniker
  - OBJREF Moniker - Kapselt marshaled IUnknown Interface Pointer auf ein bestimmtes Objekt

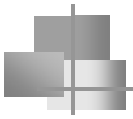


Component Categories

- Category Identifier (CATID)

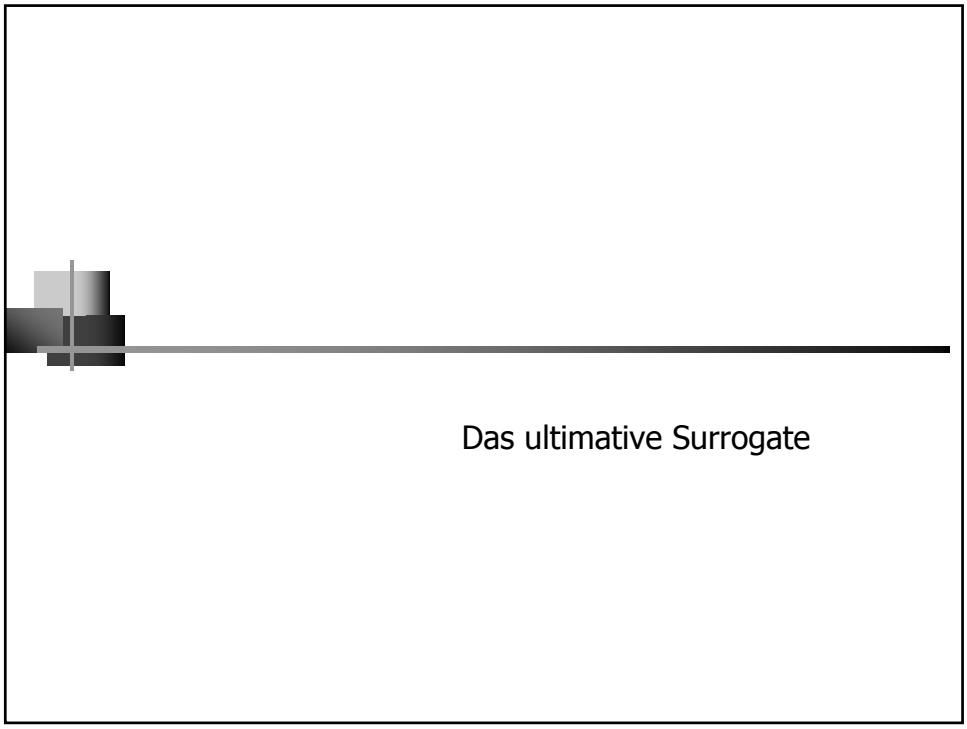


37



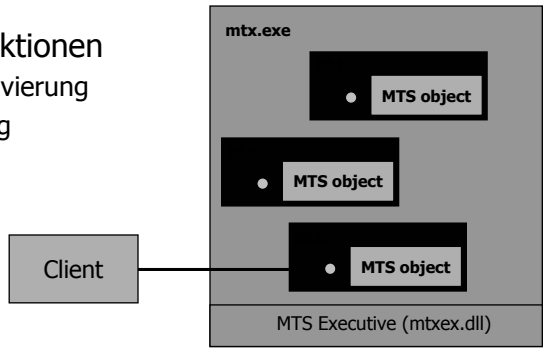
Security

- ACL innerhalb des Systems (vgl. Windows NT)
- Im Verteilungsfall in Anlehnung an OSF DCE
- Authentizitätsprüfung, wenn
  - 0: Default (Systemdefinierter Standard 1-6)
  - 1: None
  - 2: Connect
  - 3: Call
  - 4: Packet
  - 5: Packet Integrity: Test auf Paketveränderung
  - 6: Packet Privacy: Inkl. Verschlüsselung

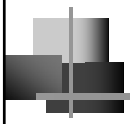


MTS

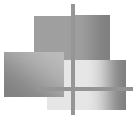
- COM+-Laufzeitumgebung (vgl. EJB Server)
- MTS = „Ultimate Surrogate"
- Wesentliche Funktionen
  - Just-in-time Aktivierung
  - Resource Pooling
  - Transaktionen
  - Persistenz
  - Administration
  - Security



OLE, DCOM und COM+ 40

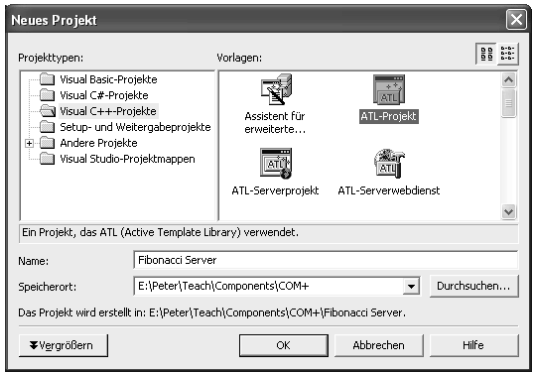


Active Template Library

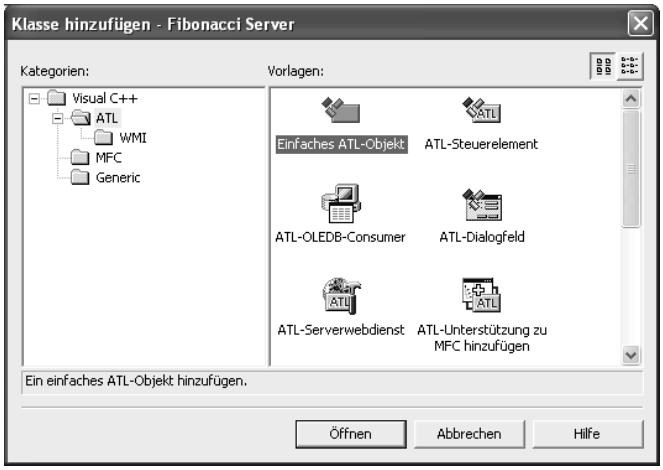


ATL

- Active Template Library
  - Teil der MFC
  - Basiert auf STL
- Grundfunktionen
  - Implementierungen für
    - **`IUnknown`**
    - **`IClassFactory`**
    - **`IDispatch`**
  - Komponenten-registrierung
- DLL oder EXE möglich

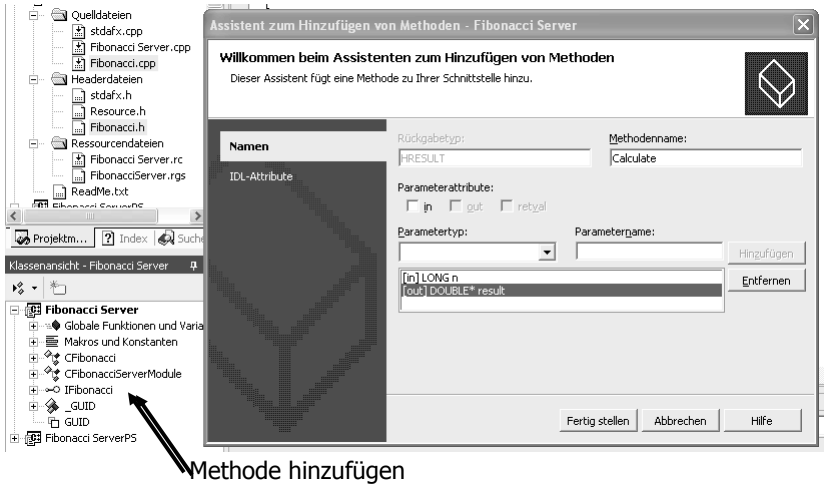


Die Basis um ein ATL-Objekt ergänzen



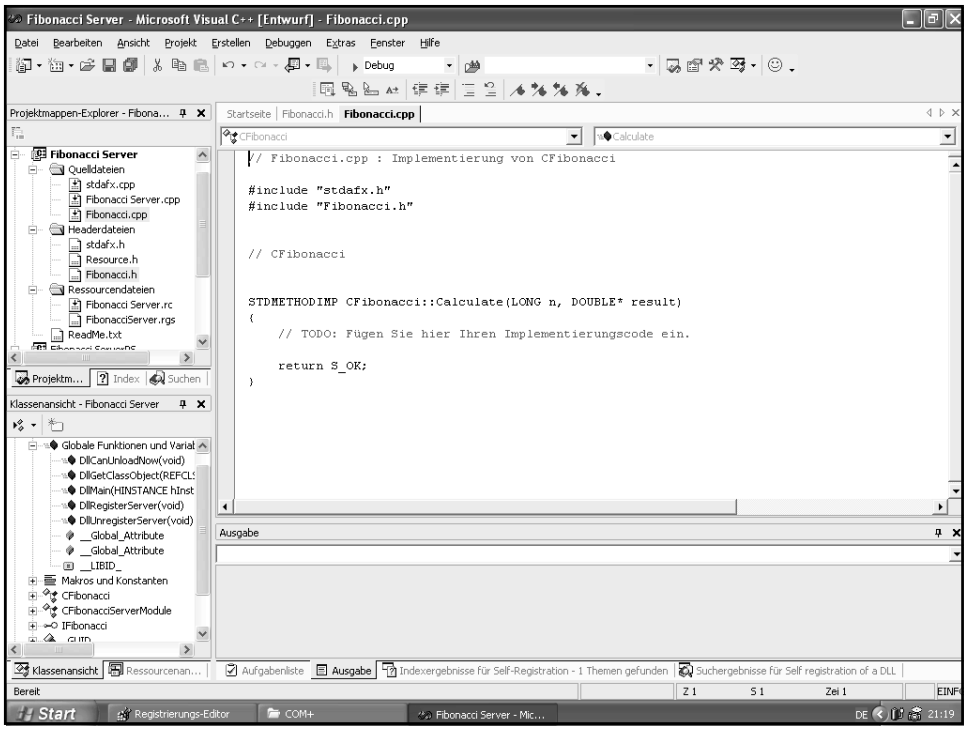
Wie heißt das Kind?

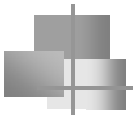
Und was kann die Klasse?



Methode hinzufügen

OLE, DCOM und COM+ 45





Literatur

- Guy Eddon, Henry Eddon
  Inside COM+ Base Services
  Microsoft Press 1999

  geplanter 2. Band „Inside COM+ Component Services"
  ist immer noch nicht erschienen ☺