

Middleware für verteilte industrielle Umgebungen

Remote Procedure Call
(Client-Server-Modell)

Komponentenbasierte Software
(DCOM)



RPC – Remote Procedure Call

- ❑ Erweiterung des Prozeduraufrufs zum Fernaufruf
- ❑ Ziel: Syntaktische und semantische Uniformität
 - Aufrufmechanismus
 - Sprachumfang
 - Fehlerfälle
- ❑ Definition (nach Nelson)
 - Synchrone Übergabe des Kontrollflusses
 - Ebene der Programmiersprache
 - Getrennte Adressräume
 - Kopplung über relativ schmalen Kanal
 - Datenaustausch: Aufrufparameter und Ergebnisse



Eigenschaften

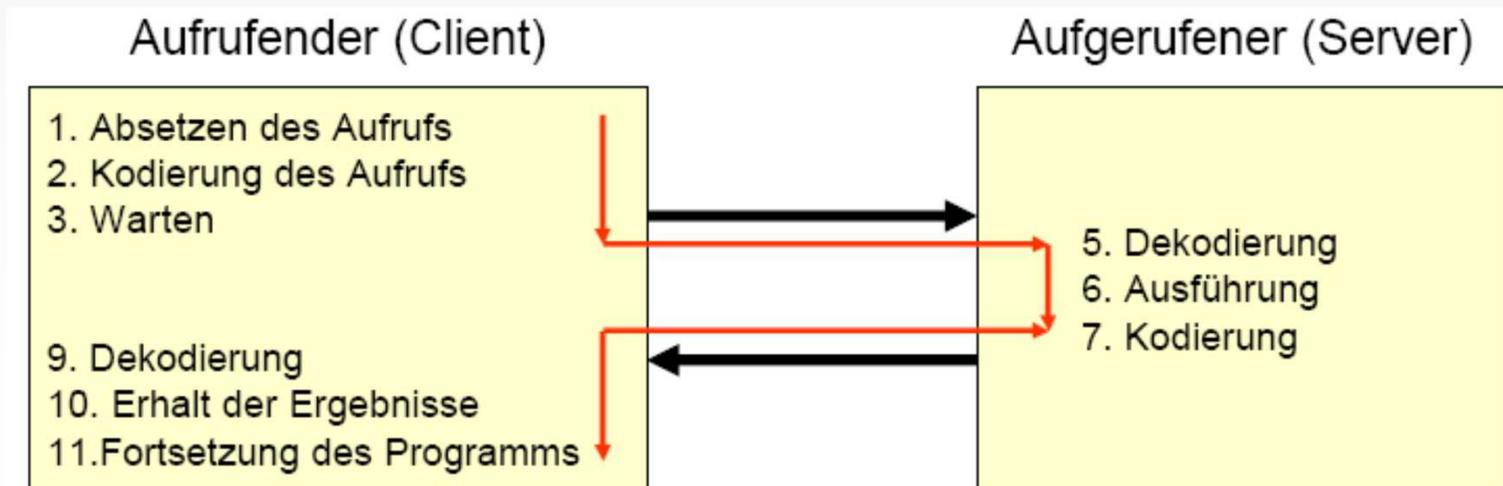
- ❑ Semantik konventioneller Prozeduraufrufe aufrechterhalten . . .
 - . . . obgleich die Ausführungsumgebung radikal anders ist
- ❑ die Semantik lokaler Aufrufe ist nur teilweise erreichbar
 - aufrufende und aufgerufene Ebene sind örtlich getrennt
 - sie besitzen keinen gemeinsamen Adressraum
 - Code und Daten liegen nicht im selben Arbeitsspeicher
 - beide Ebenen arbeiten weitestgehend autonom
 - sie sind verschiedenen Rechnern zugeordnet
 - sie werden von verschiedenen Prozessen ausgeführt
 - beide Ebenen können unabhängig voneinander ausfallen
 - der Client, während der Server seinen Aufruf bearbeitet
 - der Server, während der Client eine Antwort erwartet
 - die Sicht "alles-oder-nichts" ist nicht gegeben
- ❑ spezielle Fernaufrufprotokolle sind erforderlich



RPC – Remote Procedure Call

□ Ablauf

- Aufrufer im Wartezustand
- Parameter- und Aufrufübertragung ins Zielsystem
- Prozedurausführung
- Rückmeldung
- Fortsetzung der Programmausführung



Semantikaspekte

□ Parameter

■ Eingabeparameter

- client-seitig Werte in die request-Nachricht eintragen
- server-seitig Werte der request-Nachricht entnehmen

■ Ausgabeparameter

- server-seitig Werte in die reply-Nachricht eintragen
- client-seitig Werte der reply-Nachricht entnehmen

■ Ein-/Ausgabeparameter

- in *request*- und *reply*-Nachricht transportieren

□ Parameterübergabe

■ call-by-value ist "trivial"

■ call-by-reference erfordert zusätzliche Maßnahmen

- "Eingabe" und/oder "Ausgabe" ist explizit zu spezifizieren



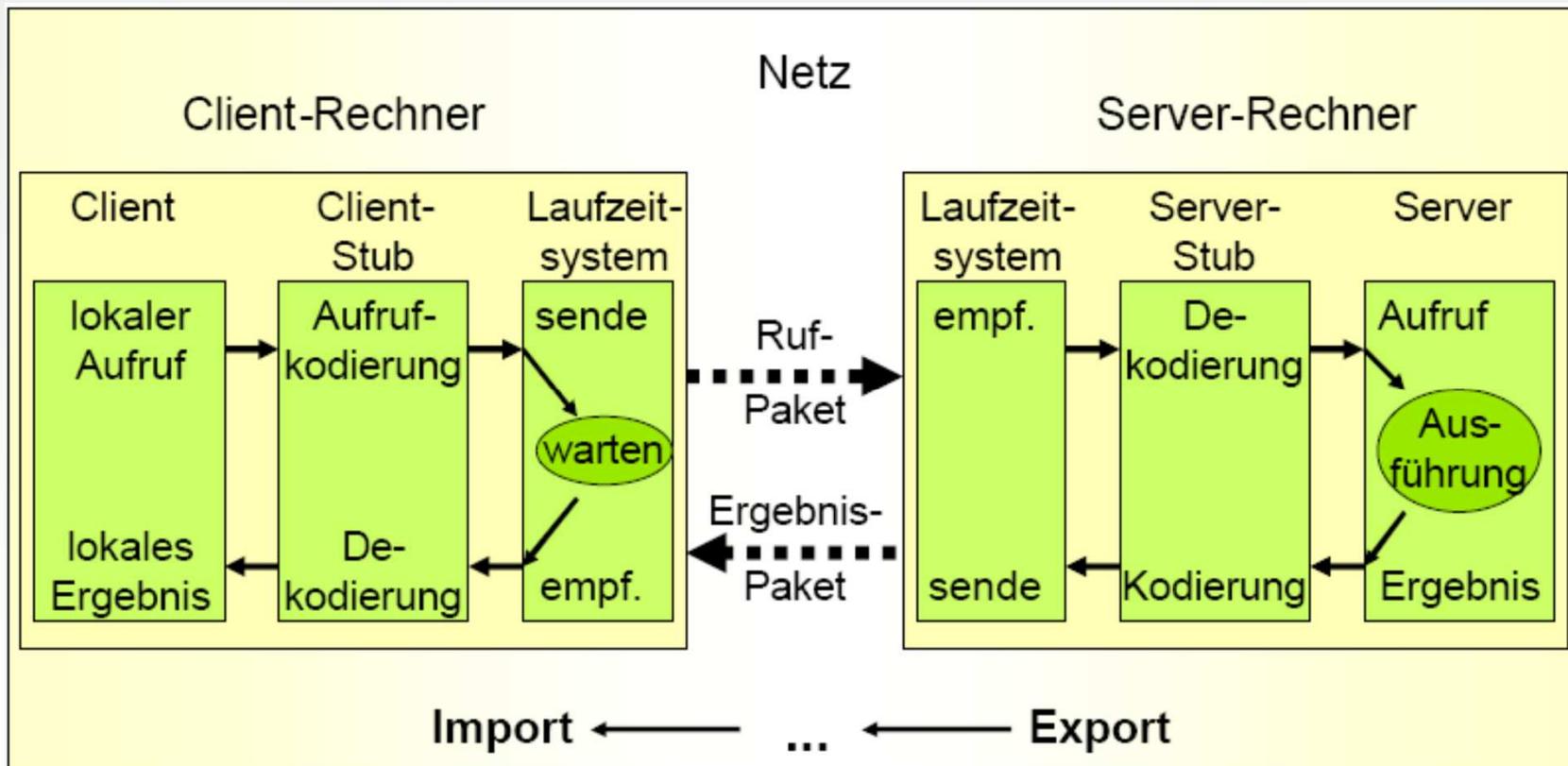
Semantikaspekte (ff)

- ❑ interface definition language (IDL)
- ❑ Gültigkeitsbereiche
 - Variablen der aufrufenden Umgebung sind ungültig
 - örtlich voneinander getrennte Ausführungsumgebungen
- ❑ Speicheradressen
 - sind systemweit uneindeutig
 - Zeiger in Nachrichten zu übertragen, ist (nahezu) sinnlos

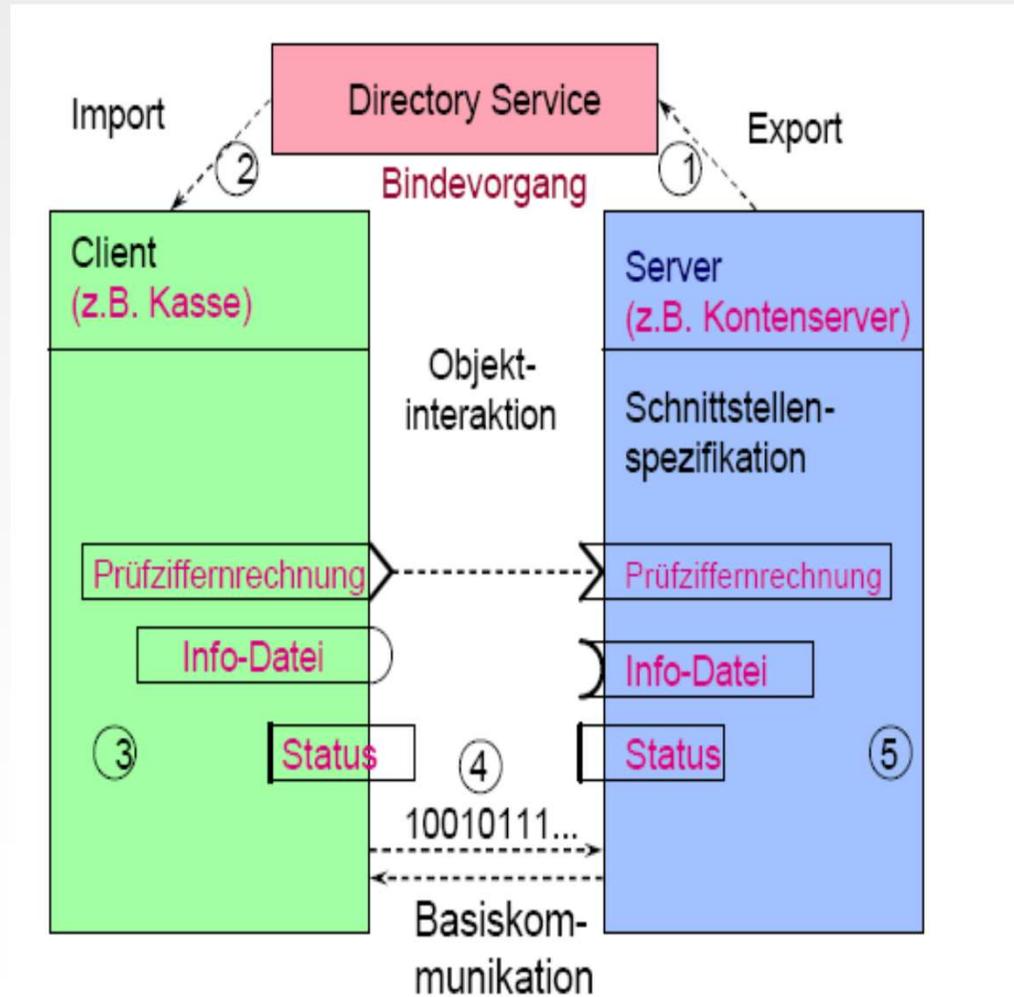


Systemarchitektur

□ RPC-DEC (Distributed Computing Environment)



Kommunikation



Schnittstellenbeschreibung: IDL

```
[uuid(765c3b10-100a-135d-1568-040034e67831),version(1.0)]
```

```
interface DocumentServer // Schnittstelle für Dokumenten-Server
{
    import "globaldef.idl"; // Import allg. Definitionen
    const long maxDoc=10; // Maximale Produktanzahl
    typedef [string] char *String; // Datentyp für Character-Strings
    typedef struct {
        String documentName; // Dokumentname
        String documentDescription; // Textuelle Beschreibung
        long size; // Speicherumfang
    } DocumentDescription; // Dokumentbeschreibung
    typedef struct {
        DocumentDescription desc; // Dokumentbeschreibung
        String header; // Dokumentkopf
        char *data; // Dokumentdaten
    } Document; // Dokument
```



Schnittstellenbeschreibung: IDL

```
[idempotent] long documentQuery ( // Anfrage nach Dokumenten
  [in] String documentName[maxDoc], // Dokumentnamen
  [out] DocumentDescription *dd[maxDoc], // Beschreibungen
  [out] long *status); // Statuswert der Operation

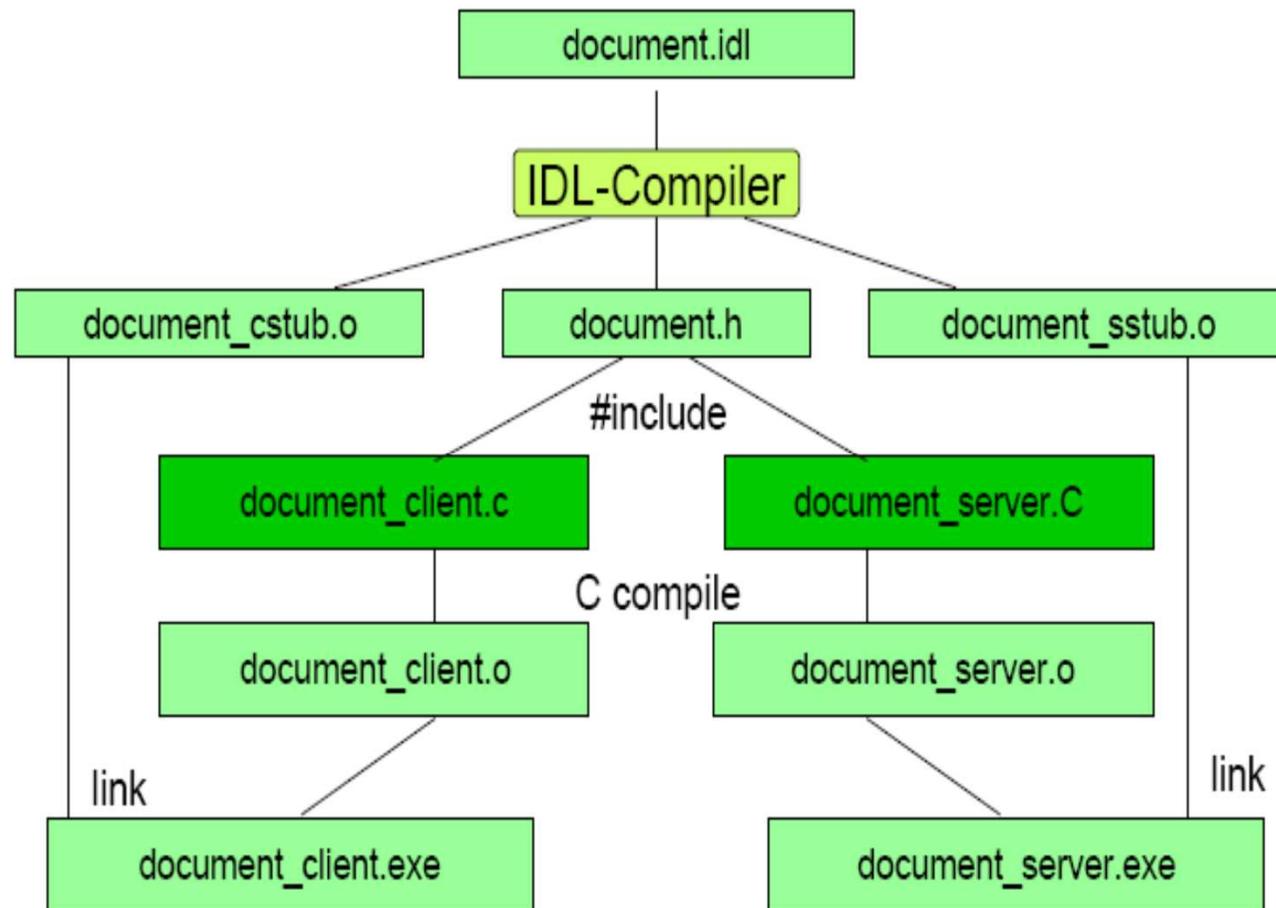
long insertDocument ( // Dokument einfügen
  [in] Document *d, // Neues Dokument
  [out] long *status); // Statuswert der Operation

long removeDocument ( // Dokument löschen
  [in] String name, // Dokumentname
  [out] long *status); // Statuswert der Operation

long fetchDocument ( // Beschaffen eines Dokuments
  [in] DocumentDescription *dd, // Dokumentbeschreibung
  [out] Document *d); // Gesuchtes Dokument
}
```



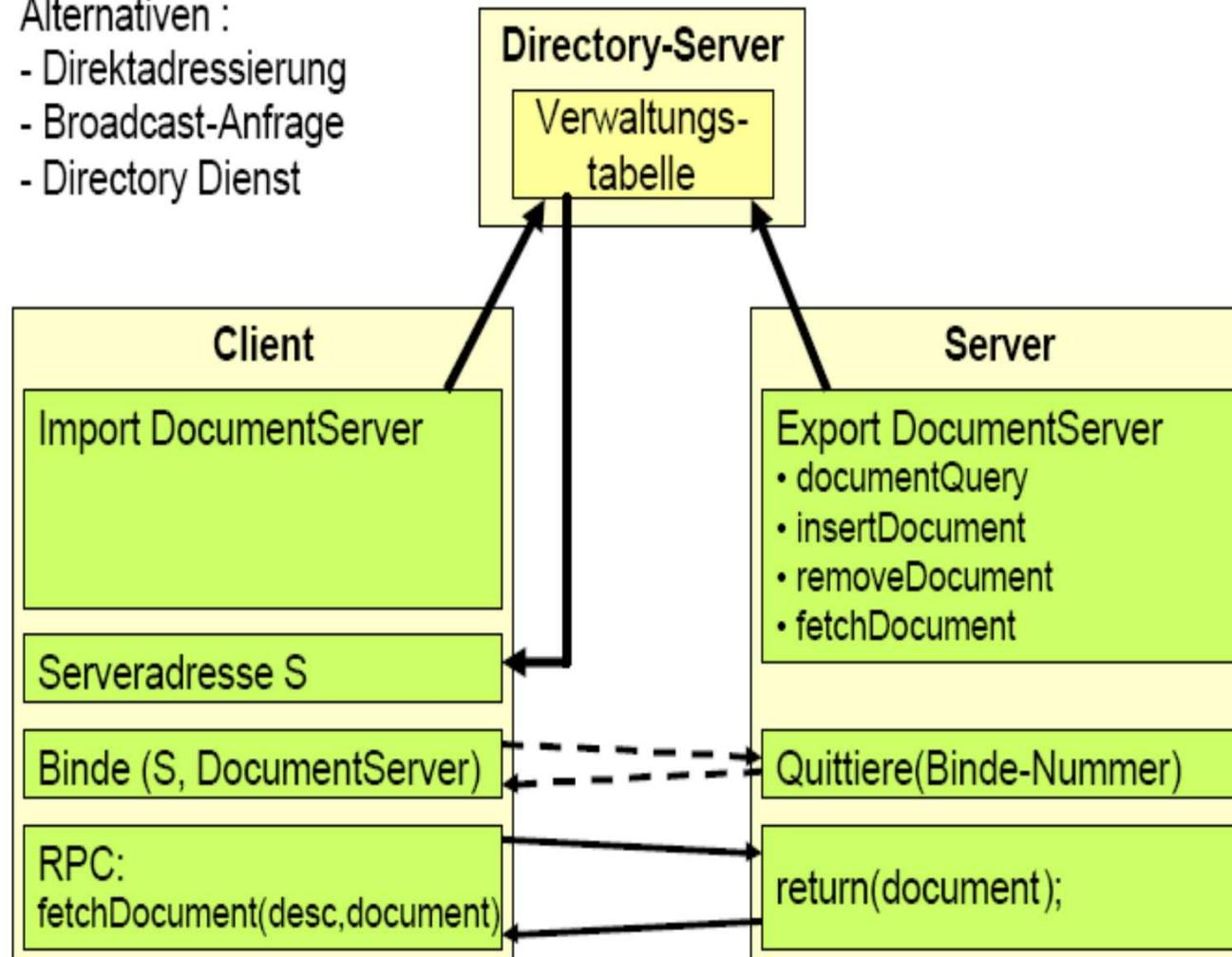
Stub-Generierung



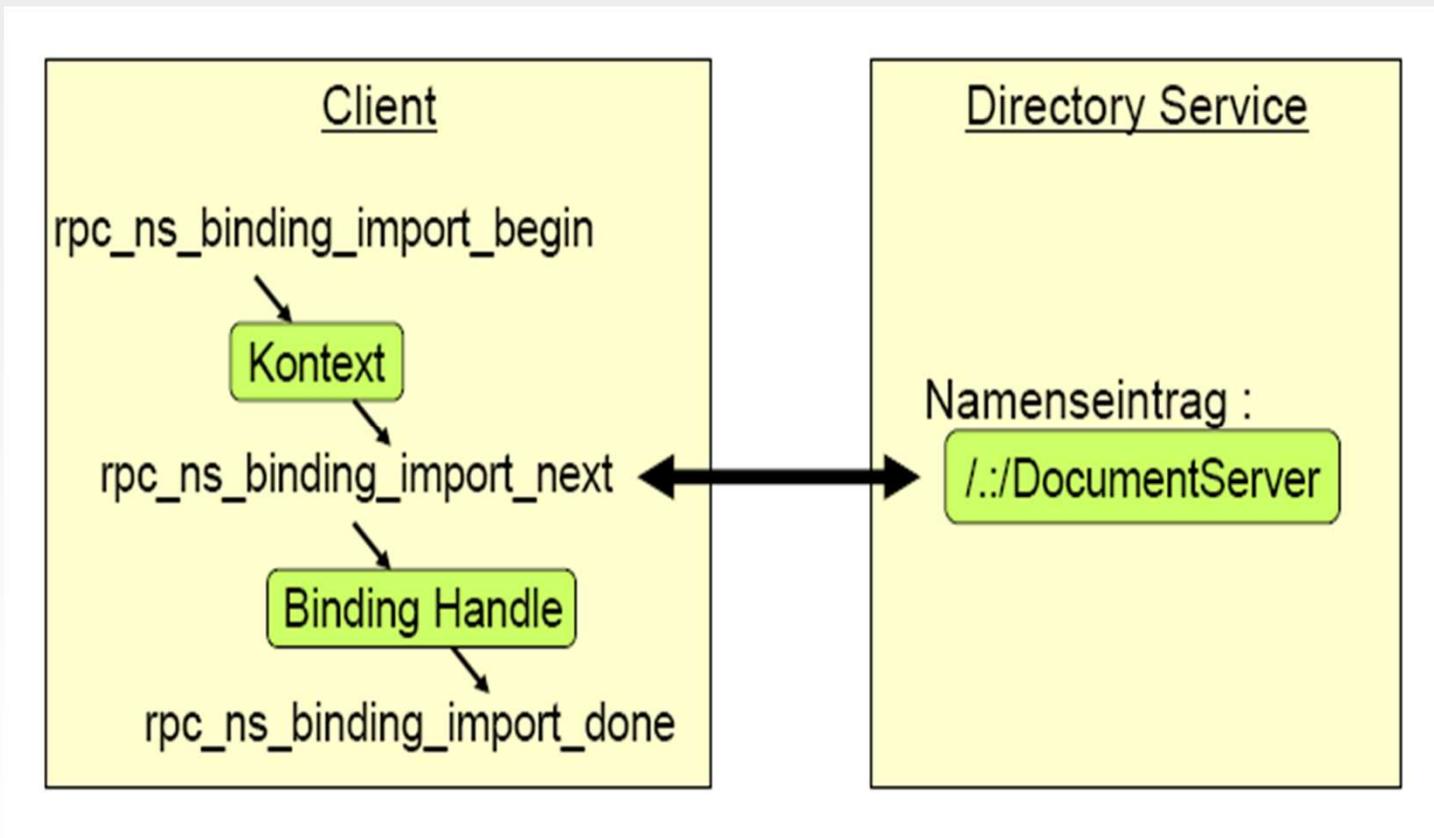
Bindevorgang

Alternativen :

- Direktadressierung
- Broadcast-Anfrage
- Directory Dienst



Explizites Binden durch Client



Details beim Binden

❑ Caching von Binde-Information

- Information auf Client-Seite global für alle Prozesse
- Erkennung veralteter Information (z.B. Timeout)
- Begrenzte Informationshaltung auf Serverseite (Skalierbarkeit, Wiederanlauf)

❑ Zeitpunkt des Bindens

- Übersetzungszeitpunkt
- Linkzeitpunkt
- Dynamisch
- Gemischte Verfahren
 - Logische Namen
 - Erste Lokalisierung beim Binden zur Initialisierungszeit
 - Erneutes Lokalisieren bei Fehlern



Flexibilität

Aufwand



Transportprotokolle bei DCE

- ❑ TCP / IP
- ❑ UDP / IP
- ❑ Herstellerspezifisch
- ❑ Beispiel:

```
rpc_server_use_protseq(„ncacn_ip_tcp“,  
                        rpc_c_protseq_max_reqs_default,  
                        &status);  
  
rpc_server_use_all_protseq(rpc_c_protseq_max_reqs_default,  
                           &status);
```



Laufzeitunterstützung: Prozesse

□ Prozessverwaltung

- Zuordnung von Prozessen zu Prozedurausführungen, Lösung der Verklemmungsproblematik
- „Lightweight“ Prozesse (Threads) :
 - Gemeinsamer Adressraum
 - Schnelle Erzeugung und Prozessumschaltung
 - Große Anzahl von Prozessen möglich→ Einsatz bei RPC - Server - Implementierungen
→ Einsatz auch bei Client → asynchron
- Prozesszuordnung
 - Prozesserzeugung pro Aufruf oder
 - Prozess - Pool
- Pufferweitergabe über Referenzen über weitere Protokollschichten
→ **Effizienz**



Prozesse auf Client-Seite

□ Prozesserzeugung :

- explizit durch `pthread_create()`
- Übergabe von Startroutine und Parametern
- Felder von Prozessen möglich

□ RPC :

- separat in Threads eingebettet
- Rückgabewert über `pthread_exit()`

□ Synchronisation

- blockierendes `pthread_join()`
- separat für alle Threads



Fehlerbehandlung beim RPC

❑ Fehlerfälle :

1. Fehler während der Prozedurabarbeitung
2. Übertragungsfehler

❑ Fehlerursachen

- Ausfall eines beteiligten Rechners
 - Server-Seite → endloses Warten des Clients → Timeout
 - Client-Seite → Weitere Bearbeitung als „Orphan“
- Unerreichbarkeit des Zielknotens
 - dynamisches Binden
- Veraltete Prozedur - Export - Information
 - dynamisches Binden



Fehlerbehandlung beim RPC (ff)

❑ Fehlersemantik (Spector) :

■ **Maybe**

- Höchstens einmalige Ausführung ohne Benachrichtigung
- im Fehlerfall → nur für „unwichtige“ Operationen

■ **At - least - once**

- Mindestens einmalige Ausführung
- nur bei **idempotenten** Operationen

■ **At - most - once**

- Erkennung und Löschen von Duplikaten
- Ausführung nur, wenn keine Rechnerausfälle vorliegen

■ **Exactly - once**

- Ausführung genau einmal
- auch bei Rechnerausfällen
- → Transaktionskonzepte mit Wiederanlauf von Komponenten



Fehlersemantik

Fehlerarten Fehlerklassen	Fehlerfreier Ablauf	Nachrichten- verluste	Zusätzlich Ausfall des Servers	Zusätzlich Ausfall des Clients
Maybe	Ausführg.: 1 Ergebnis : 1	Ausführg.: 0/1 Ergebnis : 0/1	Ausführg.: 0/1 Ergebnis : 0/1	Ausführg.: 0/1 Ergebnis : 0/1
At-Least-Once	Ausführg.: 1 Ergebnis : 1	Ausführg.: >=1 Ergebnis : >=1	Ausführg.: >=0 Ergebnis : >=0	Ausführg.: >=0 Ergebnis : 0
At-Most_Once Only-Once-Type-1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 0/1 Ergebnis : 0/1	Ausführg.: 0/1 Ergebnis : 0
Exactly-Once Only-Once-Type-2	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1



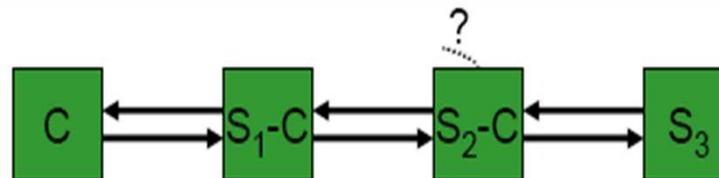
Fehlersemantik beim DCE RPC

Attribut	Semantik
default: at-most-once	Max. einmalige Ausführung mit Rückgabewert und ggf. expliziter Fehlermeldung
idempotent	Ggf. mehrmalige Ausführung mit Rückgabewert und ggf. expliziter Fehlermeldung
maybe	Ggf. mehrmalige Ausführung ohne Rückgabewert und ohne Fehlermeldung
broadcast	Ggf. mehrmalige Ausführung bei allen passenden Servern mit Rückgabewert des ersten abgeschlossenen Aufrufs



Probleme des RPC

- ❑ Übertragung großer Datenmengen
 - Synchroner Mechanismus
 - → kleine Übertragungseinheit
 - Keine verbindungsorientierte Übertragung
 - Keine Flusskontrolle und Pufferung
 - Antwortzeit- statt Durchsatzoptimierung
- ❑ Verkettung von Bearbeitungseinheiten
 - Strenge Client- / Server Semantik
 - Keine vorzeitige Datenweitergabe
 - Kein direkter Kontrolltransfer über mehr als 2 Partner

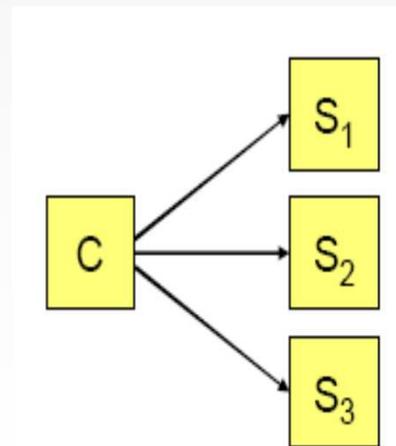


M. Riedl - Middleware - 05_RPC_DCOM



Probleme des RPC (ff)

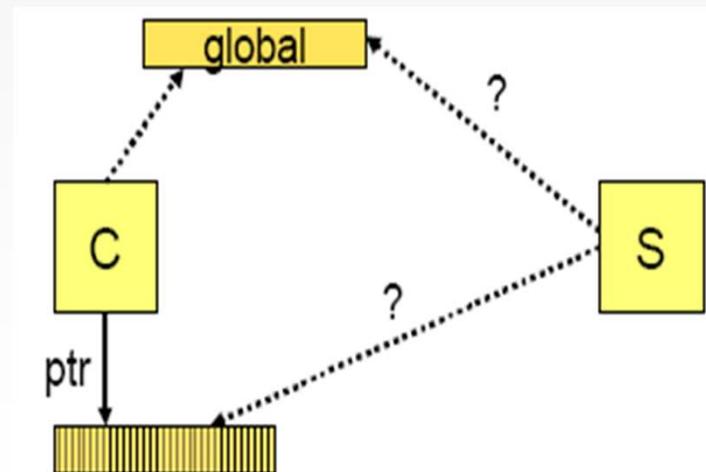
- ❑ Vertauschung von Client- / Server-Rollen
 - Keine gleichberechtigten Kommunikationspartner
 - Keine zwischenzeitlichen Ergebnismeldungen
 - Keine „Rückfragen“
- ❑ • Multicast / Broadcast nicht unterstützt



Probleme des RPC (ff)

□ Transparenzverletzungen

- Variable Parameter und Typenanzahl (z.B. `printf("%s%d", x1, x2);`)
- Zeigerparameter (z.B. `char *x, ...`)
- Globale Variable
- Fehlersemantik



Komponentenbasierte Software



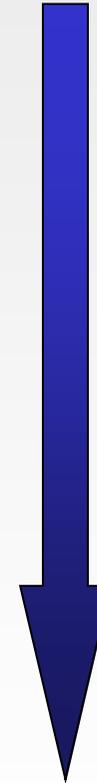
Entwicklungsschritte der Software-Entwicklung

- ❑ Traditionelle Software-Entwicklung
 - Strukturierter Entwurf, ...
 - C, Pascal, VB

- ❑ Objektorientierte Software-Entwicklung
 - OOD, Implementierung
 - C++, Modula, Java, C#

- ❑ Komponentenorientierte Software-Entwicklung
 - (D)COM
 - JavaBeans
 - .Net

- ❑ Serviceorientierte Architektur ???



Kapselung,
Wiederverwendbarkeit



Warum Komponenten?

❑ Traditionelle Software

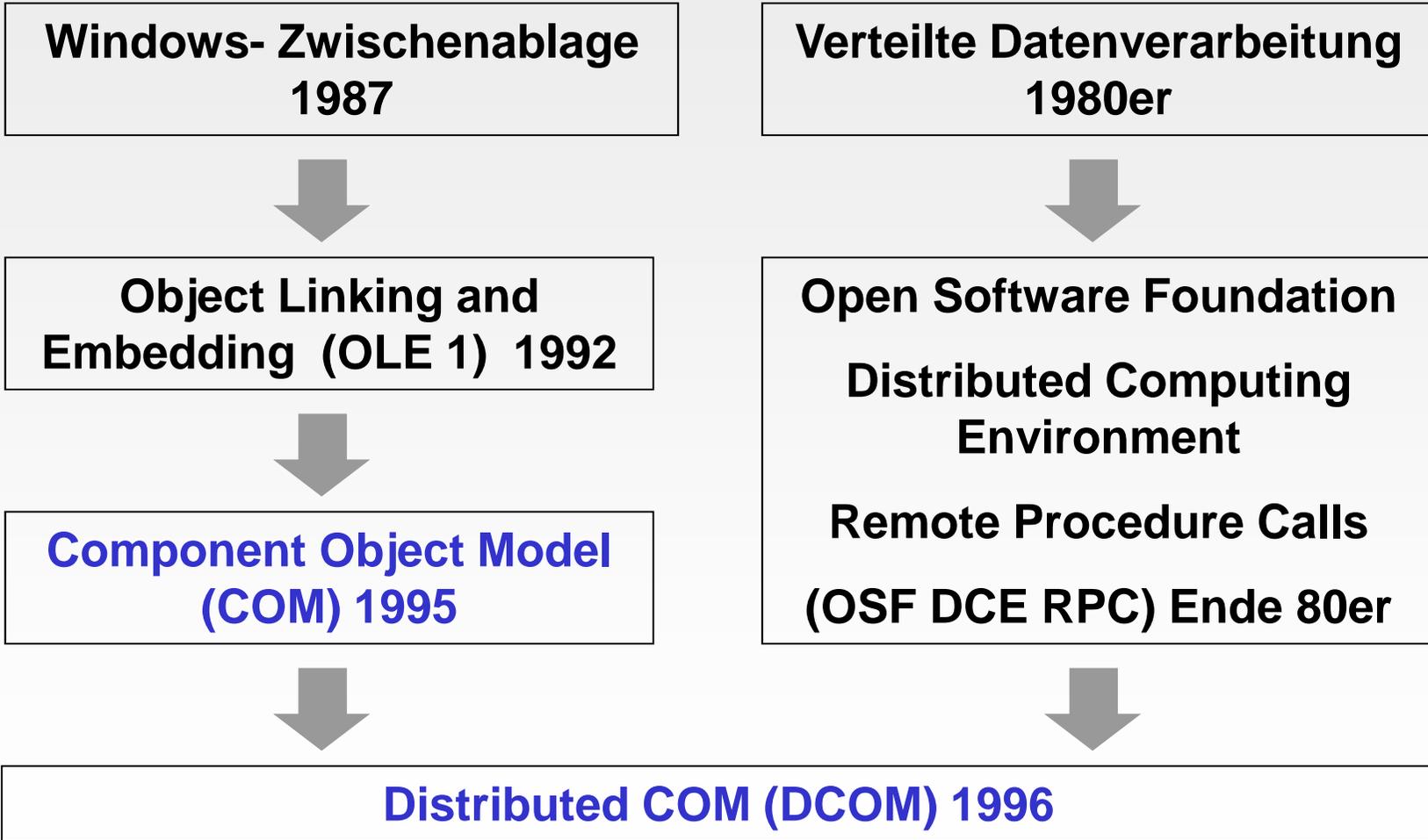
- unübersichtliche, monolithische Struktur (teilweise)
- aufwendig zu erstellen
- schwierig zu warten bzw. an Umweltveränderungen anzupassen

❑ Componentware (Komponentensoftware)

- besteht aus wiederverwendbaren, kombinierbaren Funktionseinheiten (Komponenten)
- Komponenten sind sprachenunabhängig
- fertige Komponenten stehen in Softwarebibliotheken zur Verfügung
- Schnelle, flexible Entwicklung und Aktualisierung von Softwaresystemen wird ermöglicht



Geschichte von COM/DCOM



Nutzung von COM

- ❑ COM ist Bestandteil der Windows-Betriebssysteme
- ❑ Im Gegensatz zu JavaBeans definiert COM einen **binären** Standard
 - Keine Vorgabe, wie die Bindung einer konkreten Programmiersprache an COM erfolgen soll
- ❑ Unabhängig von Programmiersprachen
- ❑ Erzeugung von Objekten
 - Clients können von einer COM-Komponente Objekte erzeugen und diese COM-Objekte benutzen, indem sie einen Zeiger auf eine Schnittstelle der Komponente anfordern
 - Über diesen Zeiger kann der Client dann Operationen der Komponente aufrufen.



Nutzung von COM (ff)

□ COM-Objekt vs. COM-Komponente

- Literatur : COM-Objekt (COMobject) wird irreführenderweise synonym für die Komponenten und die von ihnen erzeugten Objekte benutzt
- Nachfolgend wird explizit zwischen einer **COM-Komponente** und einem **COM-Objekt** unterschieden
- Ein **COM-Objekt** ist ein von einer **COM-Komponente** erzeugtes Objekt.



Begriffe

- ❑ Die COM-basierte Komponente
 - Kann mehrere COM-Klassen enthalten
 - kann sprachenunabhängig vom Client instantiiert werden

- ❑ Eine COM-Klasse implementiert mindestens eine **Schnittstelle**.
- ❑ Eine **Schnittstelle** in COM besteht aus einer Menge von **Methoden**, die in einer semantischen Beziehung zueinander stehen.

- ❑ Die Instanz einer COM-Klasse heißt **COM-Objekt**.
- ❑ Das COM-Objekt
 - bietet seine Funktionen als **Methoden** für Clients über eine oder mehrere **Schnittstellen** an
 - kann von Clients über einen **Schnittstellenzeiger** angesprochen werden

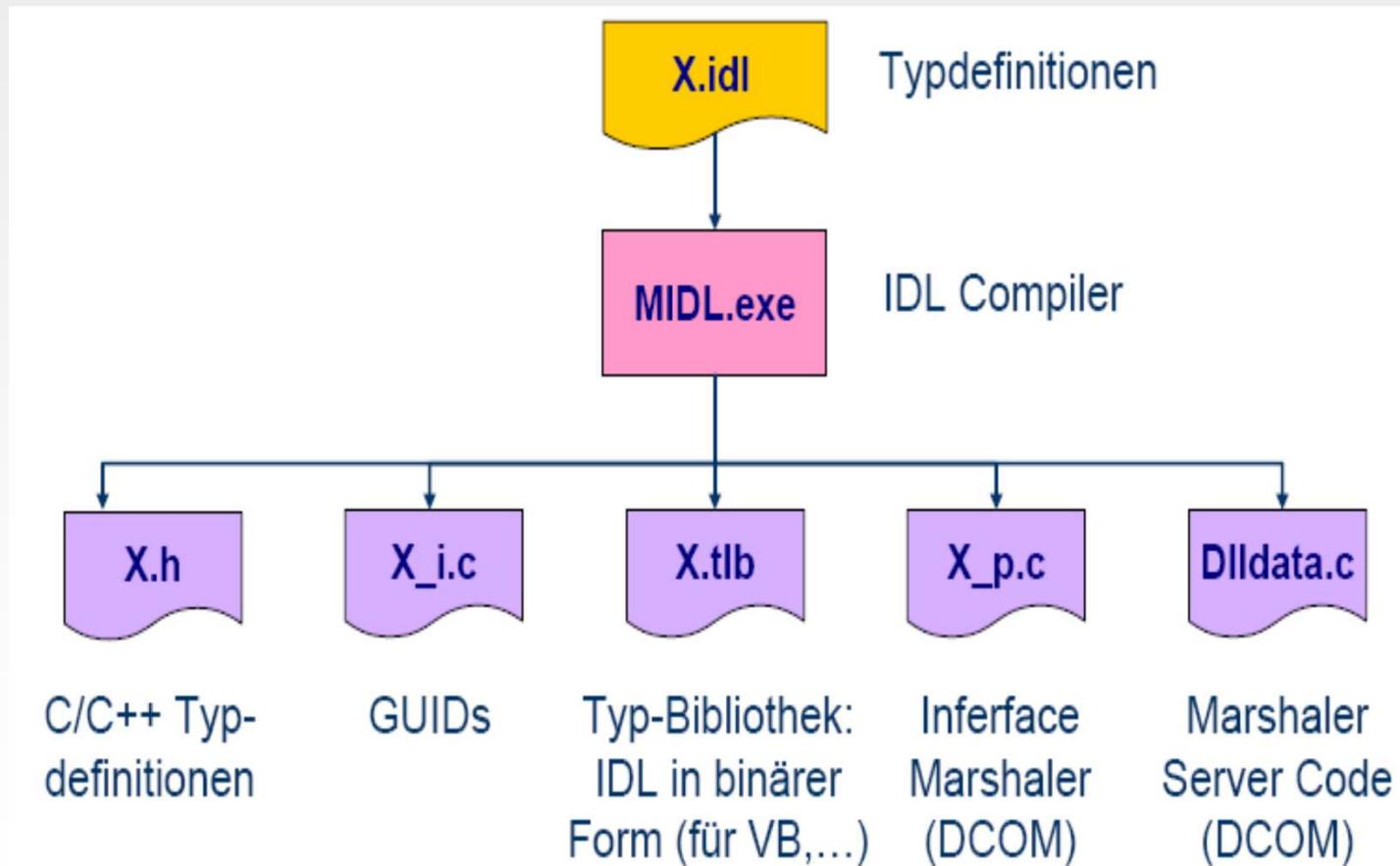


Interface Definition Language

- ❑ Basiert auf IDL von OFS DCE RPC (wie CORBA).
- ❑ Beschreibung von Interfaces und Komponenten in C-ähnlicher Syntax.
- ❑ Erweiterung um objektorientierte Aspekte (Vererbung, Polymorphismus).
- ❑ Gewährleistet Sprach- und Ortsunabhängigkeit.
- ❑ Transportprotokoll von DCOM ist Microsofts Implementierung von **DCE RPC**.



Microsoft IDL Compiler (MIDL)



Beispiel IDL File

```
import "unknwn.idl";
```

```
[  
  uuid(2A09015D-EB91-452d-B594-AD39665BB52E),  
  object,  
  helpstring("String Component")  
]
```

```
interface IString : IUnknown {  
  HRESULT Length([out, retval] int* len);  
}
```

```
[  
  uuid(9ACD001E-023D-40a0-A8C9-FD64C2FB8DC2),  
  object,  
  helpstring("Serialization Interface")  
]
```

```
interface ISerializable : IUnknown {  
  HRESULT Load([in] BSTR fileName);  
  HRESULT Save([in] BSTR fileName);  
}
```



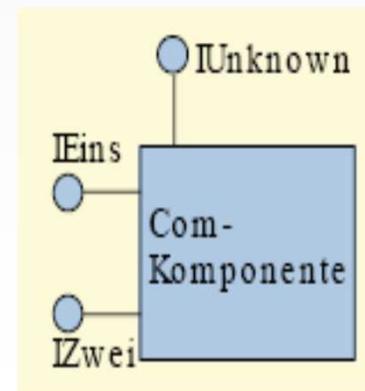
UUIDs und GUIDs

- ❑ Weltweit eindeutiger Schlüssel
 - 128 bit: C0F6EE7E-9F71-4252-9A2E-9878A15B2D83
 - Abhängig von Netzwerkadresse und Zeitpunkt der Generierung.
 - UUID= Universally Unique Identifier
 - Bezeichnung in DCE RPC
 - GUID= Globally Unique Identifier
 - Bezeichnung in COM
- ❑ Eindeutige Identifizierung von
 - Interfaces (IID)
 - Komponenten (ClassFactories) (CLSID)
 - Typbibliotheken (LIBID)



IDL Schnittstellen

- ❑ COM-Komponenten implementieren i. Allg. nicht nur eine Schnittstelle
- ❑ 2 Schnittstellen-Namen
 - Ein textueller Name, der nicht eindeutig ist und per Konvention mit dem Buchstaben »I« beginnt
 - GUID (Global Unique Identifier)
 - Global eindeutige Nummer, die nach einem speziellen Algorithmus gebildet wird und in Zeit und Raum eindeutig ist
 - Im Zusammenhang mit Schnittstellen werden GUIDs auch als Interfacelidentifier (IIDs) bezeichnet



IDL Methoden

```
HRESULT Methodname ([in]shortarg1,  
                    [out, retval]int* ret);
```

□ HRESULT

- 32-Bit Fehler-Code: allgemeiner + Methoden-spezifischer Teil.

- Verwendung in C++:

```
HRESULT hr= pIf->Methodname (i, &r);  
if (SUCCEEDED(hr)) { ... }
```

- Kann z.B. auf Exceptions (oder ähnliches) abgebildet werden.

□ Directional Attributes

- **[in]**: Client → Server
- **[out]**: Client ← Server
- **[in,out]**: Client ↔ Server
- **[retval]**: Kann auf Rückgabewert abgebildet werden

z.B. **VB**: res = obj.**Methodname**(i)



IDL Datentypen

- Standardtypen:

`boolean, byte, small, short, long, int, char, wchar_t, float, double`

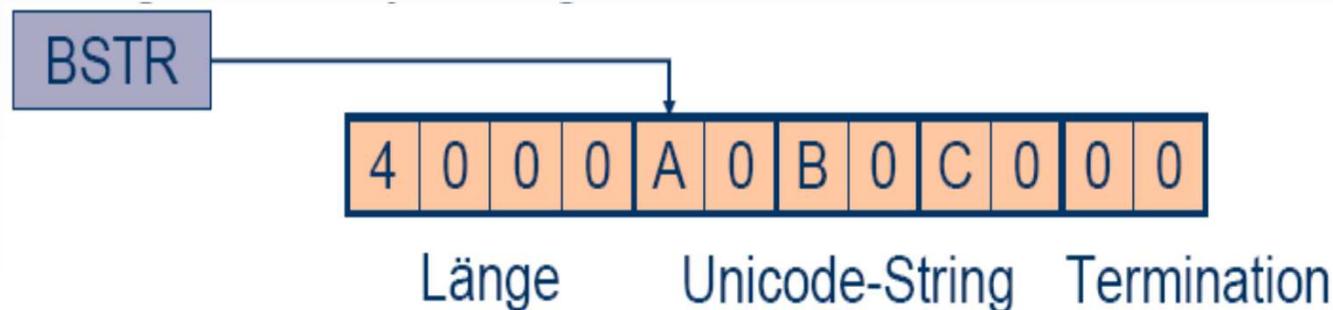
- Enumerationen: C-Syntax

- Strukturen: C-Syntax

- Pointer: C-Syntax

- Zusätzliche Attribute für effizienteren Marshaling-Code

- Strings: "BinaryStrings"



Regeln

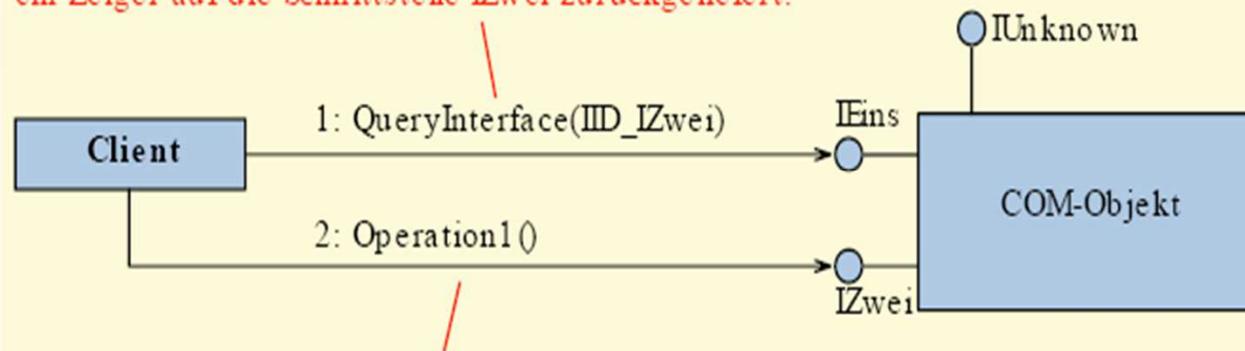
- ❑ Vererbung von Schnittstellen
 - COM unterstützt die Schnittstellenvererbung
- ❑ Schnittstelle **IUnknown**
 - Es gibt eine von COM vorgegebene Schnittstelle, die jede COM-Komponente implementieren muss
 - Schnittstelle **IUnknown**
 - Alle benutzerdefinierten Schnittstellen müssen von **IUnknown** erben
 - **IUnknown** enthält die Operationen `QueryInterface()`, `AddRef()` und `Release()`
- ❑ Methode **QueryInterface**
 - Erlaubt es, einen Zeiger auf eine der Schnittstellen einer Komponente anzufordern. Dies geschieht, indem **QueryInterface** die IID einer Schnittstelle als Eingabe übergeben bekommt
 - Alle Standard-Schnittstellen sind von Microsoft dokumentiert



Regeln (ff)

- Die IID der gewünschten Schnittstelle entnimmt der Entwickler vor dem Aufruf von **QueryInterface** der zugehörigen Dokumentation
- Als Ergebnis liefert **QueryInterface** im Erfolgsfall einen gültigen Zeiger auf die angeforderte Schnittstelle
- Im Fehlerfall wird ein Fehlercode zurückgegeben.

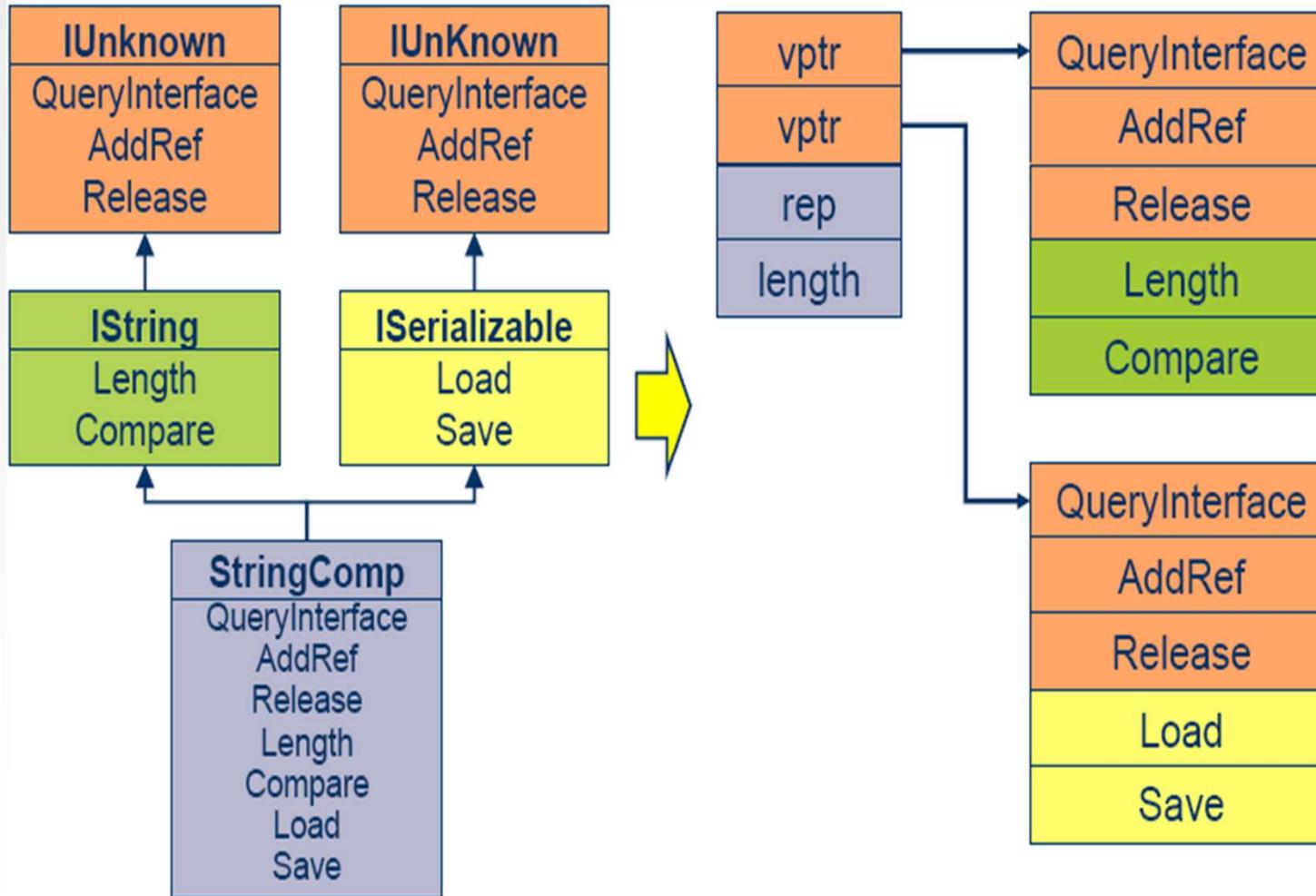
Client benutzt den Zeiger auf die Schnittstelle IEins, um mit Hilfe von QueryInterface(IID_IZwei) einen Zeiger auf die Schnittstelle IZwei anzufordern. Als Ergebnis wird ein Zeiger auf die Schnittstelle IZwei zurückgeliefert.



Der Client besitzt nun einen Zeiger auf die Schnittstelle IZwei. Er kann nun die Operation Operation1 von IZwei aufrufen.



Implementierung der Schnittstellen



Implementierung von QueryInterface

```
HRESULT __stdcall QueryInterface(REFIID riid, void** ppi) {
    if (IID_IUnknown == riid)
        *ppi = static_cast<IString*>(this);
    else if (IID_IString == riid)
        *ppi = static_cast<IString*>(this);
    else if (IID_ISerializable == riid)
        *ppi = static_cast<ISerializable*>(this);
    else {
        *ppi = NULL;
        return E_NOINTERFACE;
    }
    (reinterpret_cast<IUnknown*>(*ppi))->AddRef();
    return S_OK;
}
```



IUnknown: Reference Counting

```
class xyz: public Iabc{
    ULONG refCnt;
public:
    xyz() { refCnt= 0; }
    ULONG __stdcall AddRef() { return ++refCnt; }
    ULONG __stdcall Release() {
        if (--refCnt== 0) {
            delete this;
            return 0;
        }
        return refCnt;
    }
}
```

- ❑ Client muss mit `AddRef()` und `Release()` die Anzahl der Referenzen auf Komponente verwalten.
- ❑ Bei MTA müssen diese Funktionen **threadsafe** gemacht werden → Verwendung von `InterlockedIncrement` und `InterlockedDecrement`.

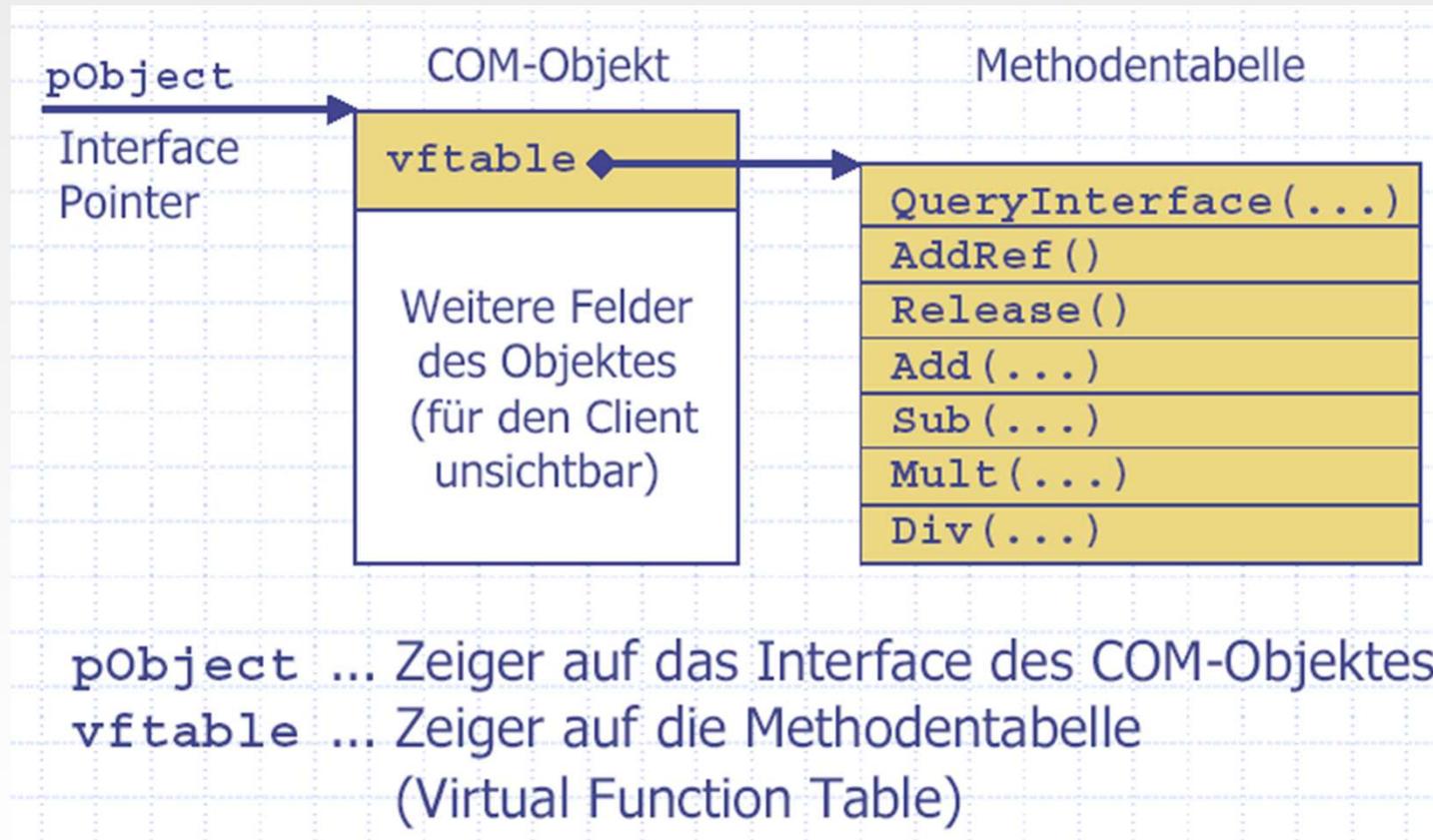


Implementierung der benutzerdefinierten Schnittstellen

```
class xyz: public Iabc{
    ULONG refCnt;
    int length;
public:
    xyz() { refCnt= 0; length = 0; }
    HRESULT __stdcall Length(int* len) {
        *len = length;
        return S_OK;
    }
    ...
}
```



Zugriff auf Methoden - Die Tabelle virtueller Funktionen



Die vtbl gibt es **pro Klasse nur einmal**, unabhängig von der Anzahl der Instanzen.



Deployment (Modultypen)

□ Module

- Auslieferung als DLLs oder Anwendungen (.exe)
- Jedes Modul: ein oder mehrere COM-Komponenten

□ Komponente in einer DLL

- Die DLL wird in den Prozessraum des aufrufenden Clients geladen
 - Deshalb auch prozessinterner Server genannt
 - Vorteil: Schnelle Kommunikation

□ Komponente in einer Anwendung

- Als lokaler Server (local server) bezeichnet
 - Das Modul ist eine Anwendung (.exe-Datei)
 - Nachteil: langsamer Zugriff
 - Vorteil: Laufen als eigenständige Anwendung.



Merkmale der Module

- ❑ Stück Binärcode
- ❑ Kann eine oder mehrere Schnittstellen implementieren
- ❑ Können nicht voneinander erben (in OOP-Terminologie)
 - Nur Schnittstellen-Vererbung
- ❑ Es kann nur über ihre Schnittstellen kommuniziert werden
- ❑ Ein Client kann einen Zeiger auf alle Schnittstellen, die eine Komponente implementiert, mit Hilfe der Operation `QueryInterface` und der entsprechenden IID erhalten



COM-Objekte erstellen: ClassFactory

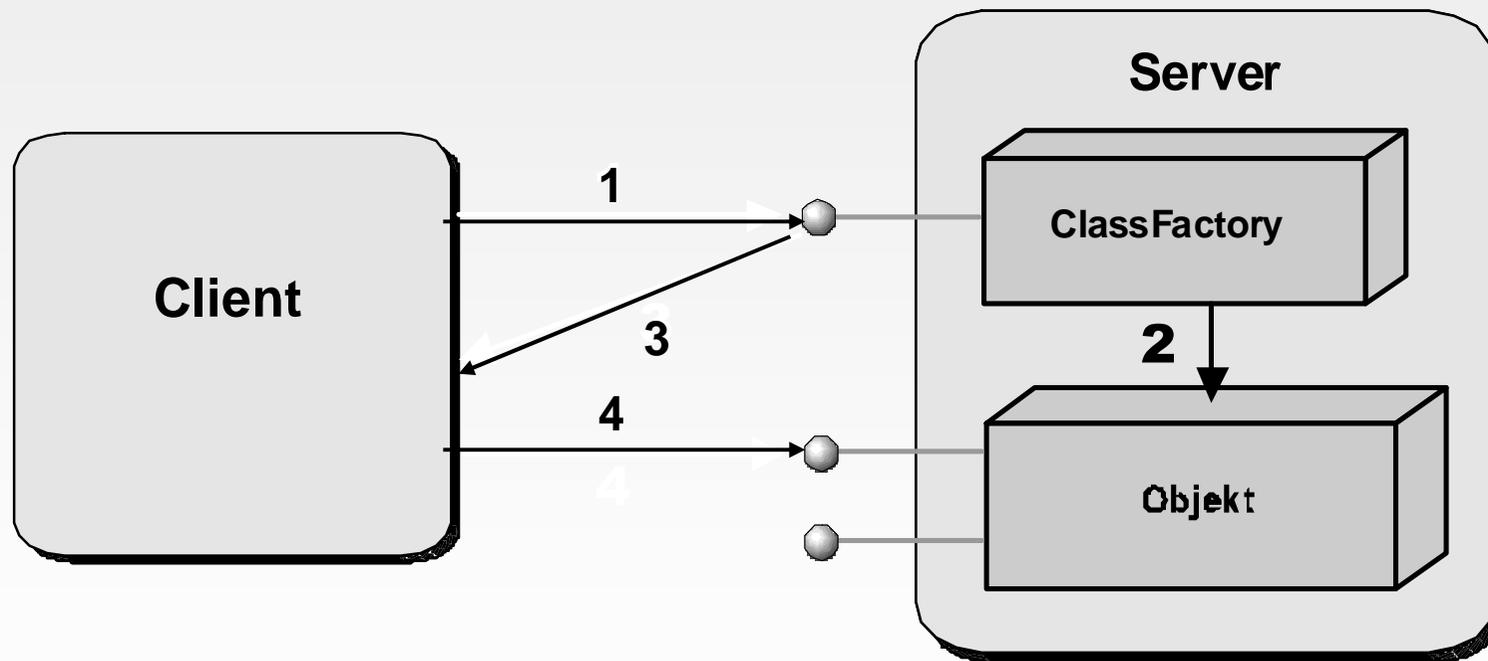
- ❑ ClassFactory ist eine Komponente, mit der COM-Objekte erzeugt werden können.
- ❑ Muss für jede Komponente implementiert werden.
- ❑ Interface:

```
interface IClassFactory: IUnknown {  
    HRESULT CreateInstance([in, unique] IUnknown* pUnkOuter,  
                            [in] REFIID rrid,  
                            [out, iid_is(rrid)] void** ppObj);  
    HRESULT LockServer([in] BOOL lock);  
}
```

- CreateInstance: Erzeugen eines COM-Objekts.
- LockServer: Verhindert frühzeitige Freigabe des COM-Objekts.



COM-Objekte erstellen: ClassFactory (ff)



Bedeutung der Registry

- Wie erzeugt man COM-Objekte und wie gelangt man an den ersten Schnittstellenzeiger?
 - Die CLSID einer Komponente wird zusammen mit dem Aufenthaltsort des Moduls, das sie implementiert, in der [Windows Registry](#) abgelegt
 - Eine COM-Komponente muss registriert werden
 - Im Zweig HKEY_CLASSES_ROOT/CLSID hinterlegt jede Komponente als Schlüssel ihre CLSID
 - Dieser Schlüssel enthält als Wert den Namen der Komponente
 - Ein Unterschlüssel enthält den Dateinamen des Moduls, welches die Komponente implementiert
 - Der Name des Schlüssels ist [InprocServer32](#) bei DLL-Modulen, [LocalServer32](#) bei Exe-Modulen.

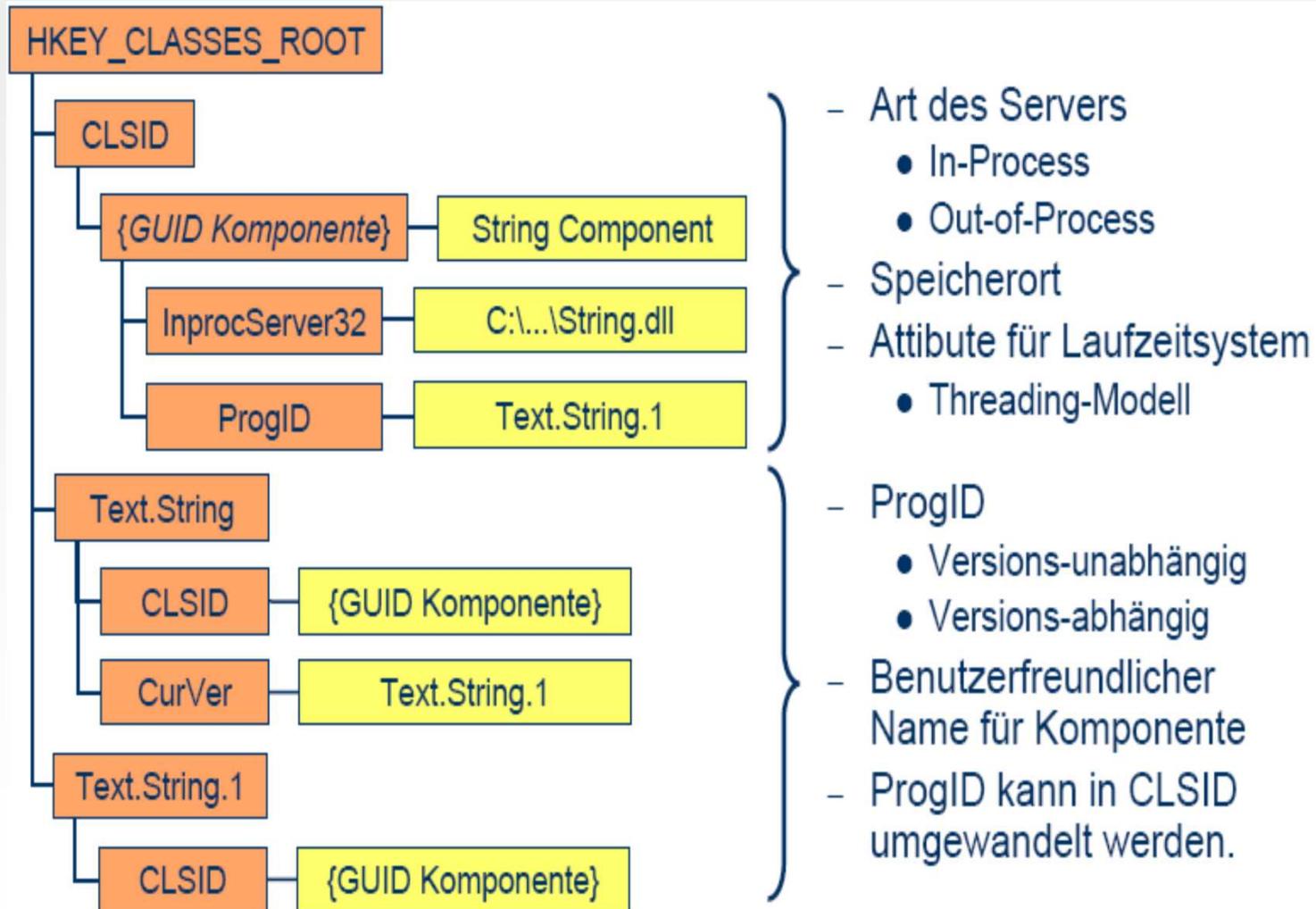


Bedeutung der Registry (ff)

- Zweig `HKEY_CLASSES_ROOT`
 - Jede Komponente trägt zusätzlich als Schlüssel ihren Namen ein. Dies kann zu Konflikten führen:
 - Haben 2 Komponenten den gleichen Namen, dann überschreibt die zuletzt registrierte Komponente den Schlüssel der anderen Komponente
 - Beispiel: `HKEY_CLASSES_ROOT/Excel.Application`
 - Ein Unterschlüssel namens `CLSID` enthält als Wert die `CLSID` der Komponente
- Mit den Informationen aus der [Registry](#) können die COM-Bibliotheken Komponenten zur Verfügung stellen.



Einträge in der Windows Registry



Schnittstelle einer COM-DLL

- ❑ Jede COM-DLL kann mehrere Komponenten enthalten.
- ❑ COM-DLL muss folgendes Interface aufweisen:

```
LIBRARY "xyz"
```

```
EXPORTS
```

```
DllGetClassObject@1 PRIVATE
```

```
DllCanUnloadNow@2 PRIVATE
```

- DllGetClassObject: Instanziert eine **ClassFactory** und gibt eine Referenz darauf zurück.
- DllCanUnloadNow: Gibt zurück, ob noch Referenzen auf COM-Objekte oder Locks auf Komponente existieren.



Selbstregistrierung einer COM-DLL

- ❑ Schnittstelle der COM-DLL kann um zwei Funktionen für Selbstregistrierung erweitert werden:

```
LIBRARY „xyz“
```

```
EXPORTS
```

```
DllGetClassObject @1 PRIVATE
```

```
DllCanUnloadNow @2 PRIVATE
```

```
DllRegisterServer @3 PRIVATE
```

```
DllUnregisterServer @4 PRIVATE
```

- ❑ Registrierung der Komponente

```
regsvr32 <Pfad>\xyz.dll
```

- ❑ Deregistrierung der Komponente

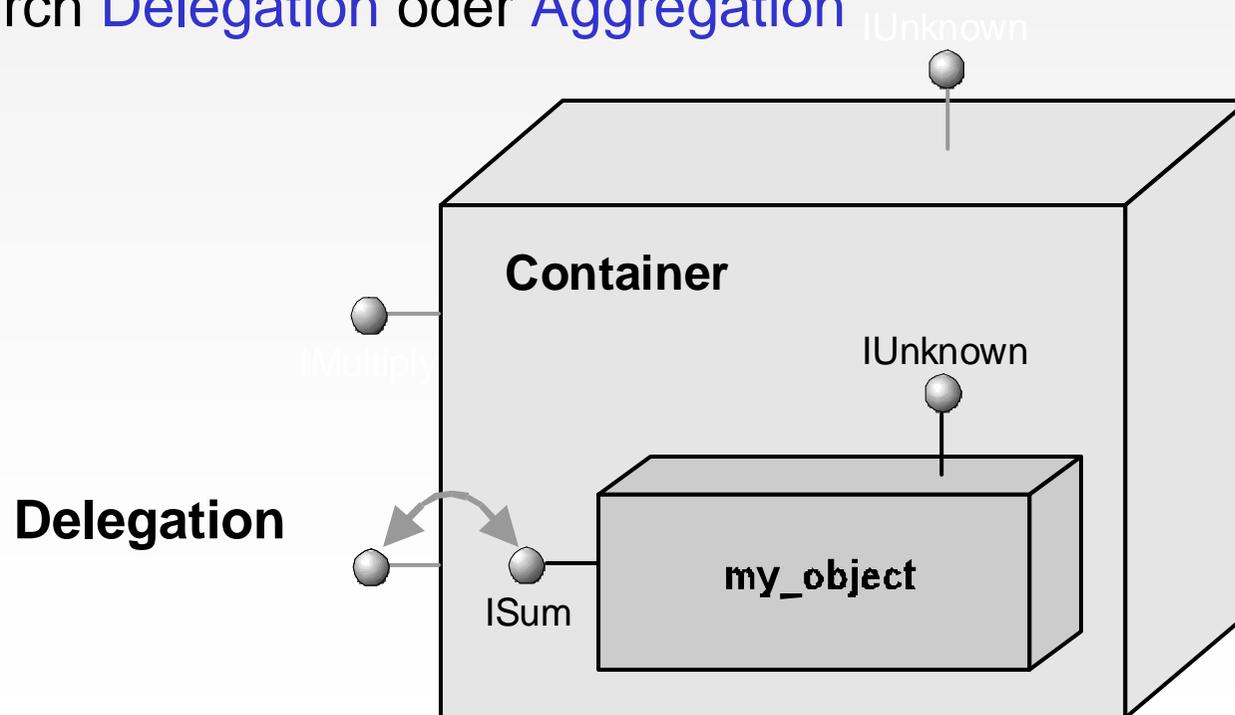
```
regsvr32 /u <Pfad>\xyz.dll
```



Wiederverwendungsmechanismen

Delegation

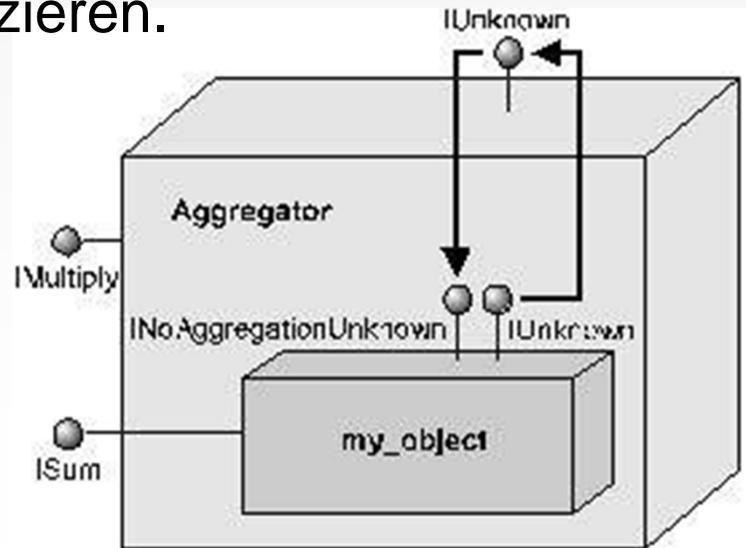
- ❑ **Schnittstellenvererbung** statt Quellcodevererbung
- ❑ Quellcodevererbung ist nur innerhalb von Komponenten möglich
- ❑ Wiederverwendung von COM-Objekten erfolgt zur Laufzeit durch **Delegation** oder **Aggregation**



Wiederverwendungsmechanismen

Aggregation

- ❑ Mechanismus kann nur zwischen prozessinternen Komponenten eingesetzt werden
- ❑ Schnittstellen des internen Objekts werden tatsächlich als Schnittstellen des äußeren **Aggregators** offengelegt
- ❑ Im Gegensatz zur **Delegation** kann der Client also direkt mit dem internen Objekt kommunizieren.

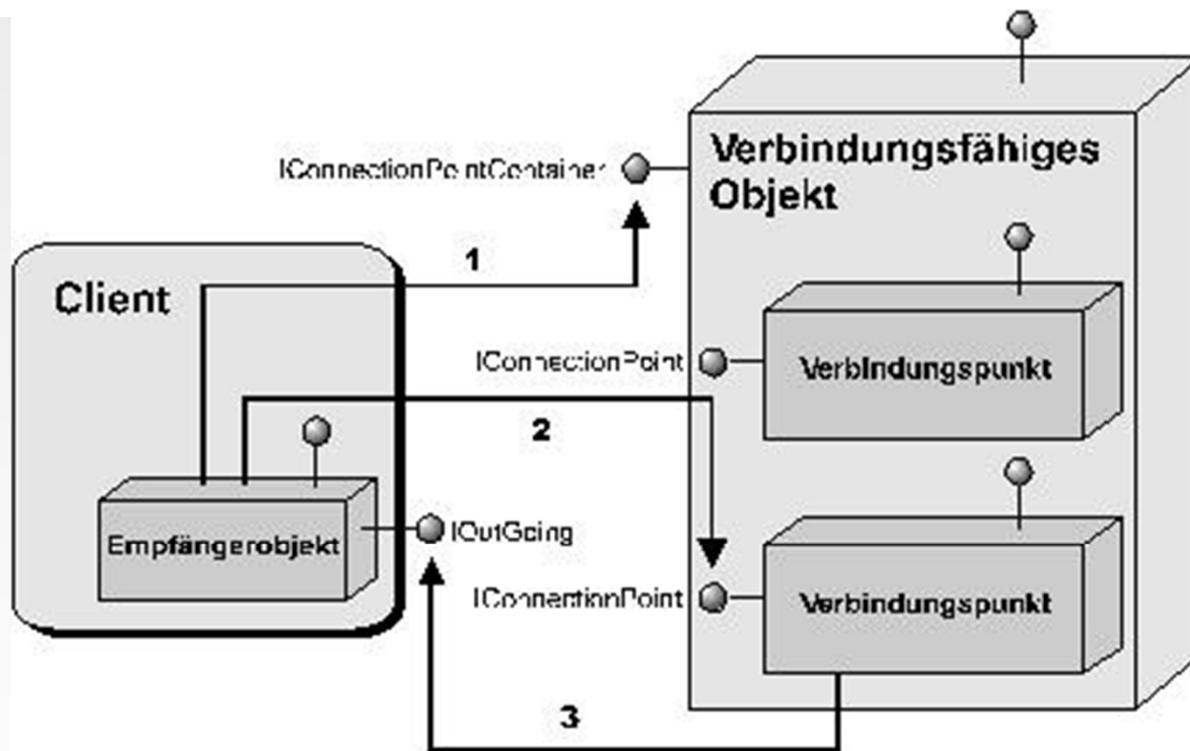


Verbindungsfähige Objekte

- ❑ **Verbindungsfähige Objekte** sind in der Lage, den **Client** über einen vom Objekt selbst initiierten Aufruf anzusprechen. Hierzu ist ein **empfangendes Objekt** innerhalb des Client nötig, das die Aufrufe des verbindungsfähigen Objekts entgegennimmt.
- ❑ Dadurch sind Komponenten in der Lage, Funktionen im Client unabhängig von ihm zu steuern.
- ❑ Notwendige Schnittstellen: **IConnectionPointContainer**, **IConnectionPoint**
- ❑ Der bekannteste Typ verbindungsfähiger Objekte ist das **ActiveX-Steuerelement**



Verbindungsfähige Objekte (ff)



DCOM

- ❑ Netzwerkprotokoll höherer Ebene, das es COM-basierten Komponenten ermöglicht, über ein Netzwerk zusammenzuarbeiten
- ❑ unterstützt neben TCP/IP und HTTP viele weitere Protokolle
- ❑ Stellung von **DCOM** in einem Protokollstapel im Vergleich zum **OSI** -Schichtenmodell:

Anwendung	DCOM / RPC
Darstellung	
Sitzung	Winsock-Treiber
Transport	UDP
Netzwerk	IP
Sicherung	Ethernet-Treiber
Physisch	Ethernet-Karte

- ❑ DCOM ist kein eigenständiges Protokoll, das in einer höheren Schicht über RPC steht
- ❑ benutzt RPC und verändert die PDU vom RPC-Protokoll durch eigene Daten



DCOM (ff)

- ❑ ist verbindungsorientiert
 - ❑ garantiert Datenreihenfolge
 - ❑ sorgt nach einer Remote-Instantiierung für die Übermittlung einer Objektexporteurkennung (OXID) an den Client

 - ❑ **OXID-Resolver**
 - Dienst von DCOM, ist auf jedem DCOM-Rechner installiert
 - speichert und übersetzt OXIDs (bezeichnen RPC-Verbindungszeichenfolgen) und stellt sie lokalen Clients zur Verfügung
 - sendet für jeden Client in regelmäßigen Abständen Ping-Nachrichten an alle Server im Netz, damit Komponenten einen Ausfall bemerken und aus dem Speicher entfernt werden können
- verteilter Garbage-Collector !



COM+

- ❑ Weiterentwicklung von COM
- ❑ wird zusätzliche Dienste bieten (z.B. Implementierungsvererbung oder Abfangen von Methodenaufrufen)
- ❑ das COM+-Laufzeitmodul wird Standardimplementierungen der Standard-COM-Schnittstellen anbieten (sind in J++ und Visual Basic bereits integriert, deshalb größerer Nutzen für C++-Programmierer)
- ❑ definiert Standarddatentypen mit Speicherdarstellung in der COM+-Spezifikation als weitere Unterstützung der Interoperabilität (wichtig bei Erstellung von Komponenten in verschiedenen Programmiersprachen)

❑ → .Net

