

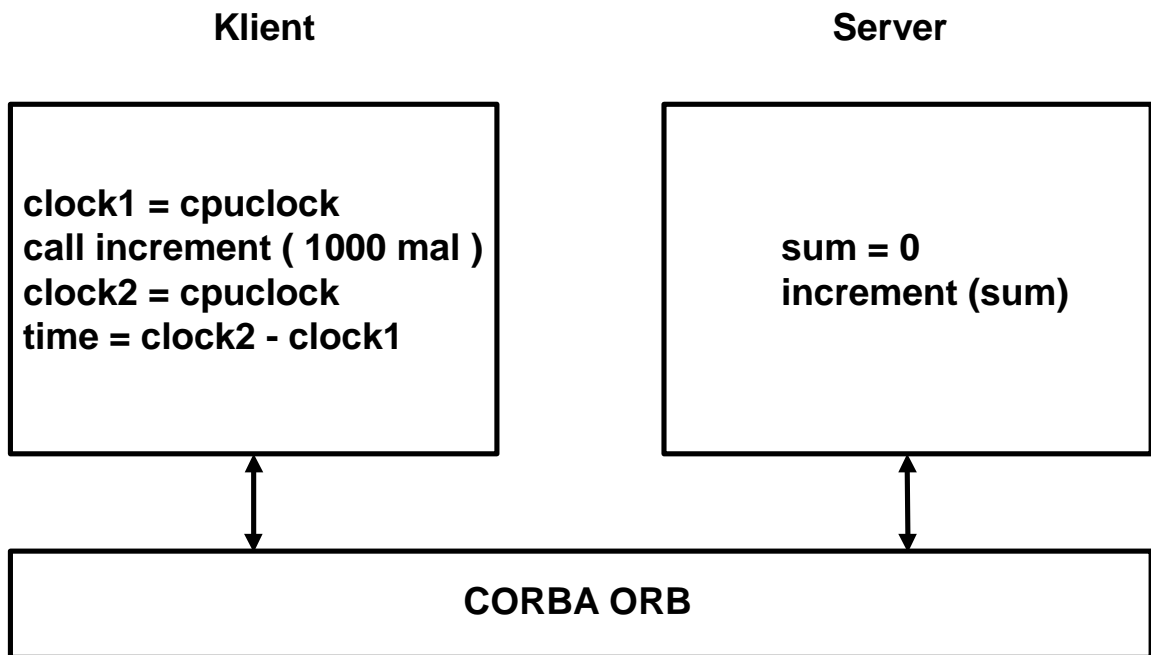
Internet Anwendungen unter OS/390

**Dr.rer.nat Paul Herrmann
Prof. Dr.-Ing. Udo Kebschull
Prof. Dr.-Ing. Wilhelm G. Spruth**

WS 2000/01

Teil

ES/390 Java Einrichtungen

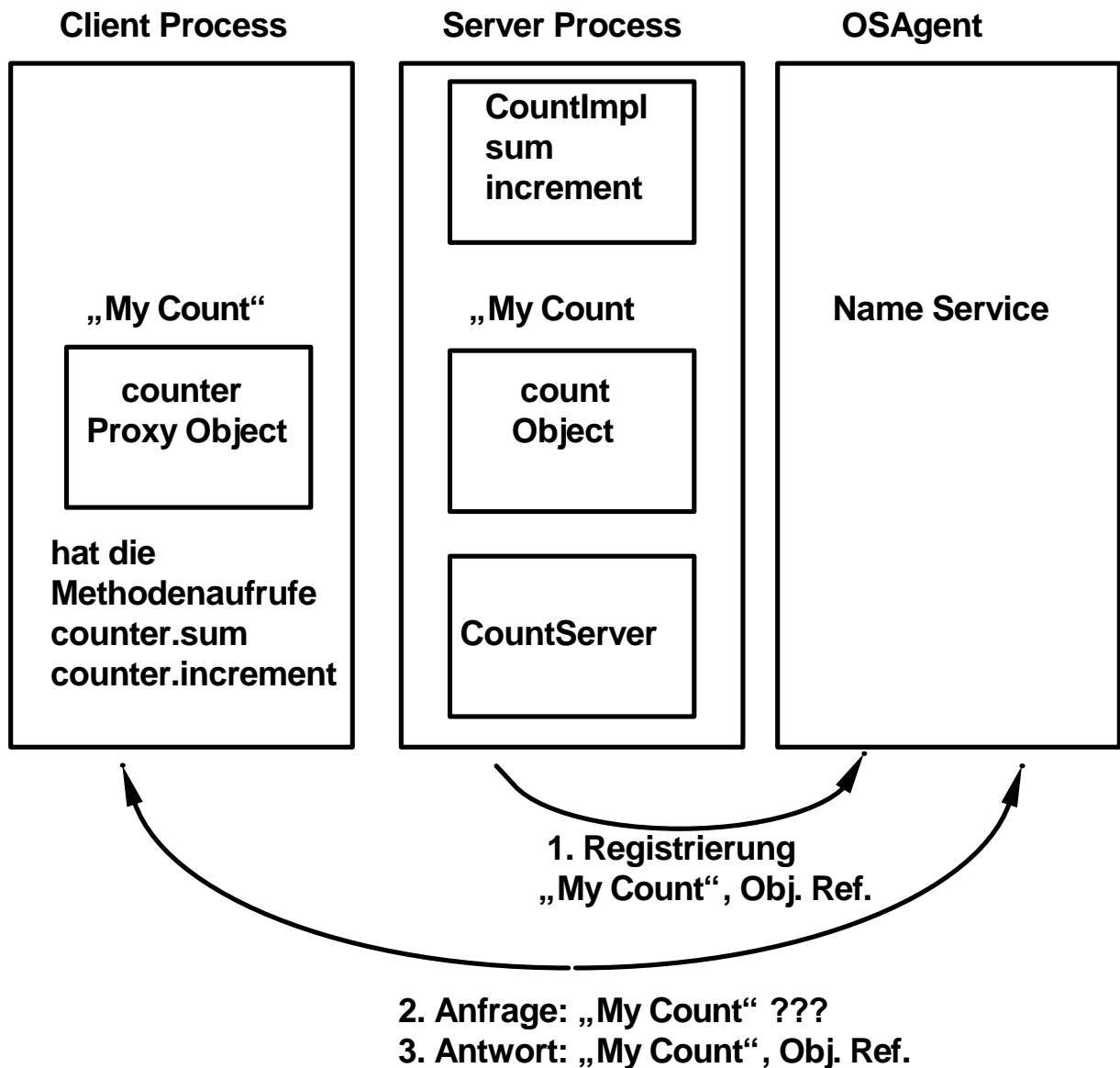


CORBA Beispiel (1)

Implementierung des Beispiels in Java (Klient und Server)

Klient und Server Code werden in einem CORBA Module (einer Java Package) mit dem Namen „Counter“ zusammengefaßt.

Die Schnittstelle zwischen Klient und Server erhält den Namen „Count“. Dies ist gleichzeitig die Bezeichnung der Klienten und Server Klassen der CORBA Module Counter.



Der interne „state“ des Objektes „My Count“ wird durch 1 Variable (int sum) dargestellt.

Counter Beispiel

```
// Count.idl  
  
module Counter  
{  
  interface Count  
  { attribute long sum;  
    long increment();  
  };  
};
```

IDL Schnittstellen Definition der Count Schnittstelle

Der IDL Precompiler erzeugt aus der Schnittstellen Definition Java Programme für

**Client Stub
Server Skeleton,
weitere Hilfsprogramme**

und lädt die Schnittstellen Beschreibung in das Interface Repository.

```

// CountClient.java Static Client, VisiBroker for Java
class CountClient
{ public static void main(String args [ ] )
  { try
    { // Initialize the ORB
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
null);

      // Bind to the Count Object
      Counter.Count counter = Counter.CountHelper.bind(orb, "My
      Count");

      // Set sum to initial value of 0
      counter.sum((int)0);

      // Calculate Start time
      long startTime = System.currentTimeMillis( );

      // Increment 1000 times
      for (int i = 0 ; i < 1000 ; i++ )
      { counter.increment();
      }

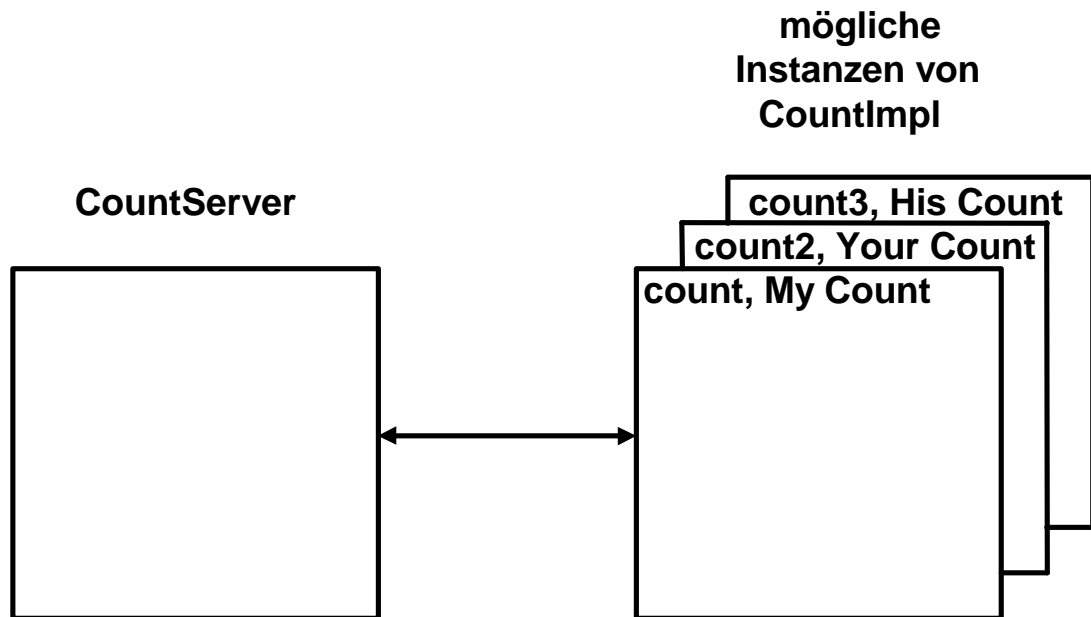
      // Calculate stop time
      long stopTime = System.currentTimeMillis( );

      //print statistics
      System.out.println("Avg Ping = "
        + ((stopTime - startTime)/1000f) + " msecs");
      System.out.println("Sum = " + counter.sum( ));
    }

    // handle exceptions
    catch(org.omg.CORBA.SystemException e)
    { System.err.println("System Exception");
      System.err.println(e);
    }
  }
}

```

Count Client Programm



Count Server Implementierung

Die Server Seite von Count besteht aus 2 Programmen.

CountServer initialisiert ORB, BOA und erzeugt die **Count** Objekte (die Instanzen der Klasse **Count**).

Im vorliegenden Beispiel ist dies nur 1 Objekt mit den beiden Bezeichnungen „count“ (klein geschrieben) bzw. „My Count“. In der Regel werden mehrere Instanzen der Klasse **Count** erstellt.

CountImpl enthält den Code für das Server Objekt, welches vom Klienten aufgerufen wird.

// CountServer.java: The Count Server main program

```
class CountServer  
{ static public void main(String[] args)  
  { try  
    { // Initialize the ORB  
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,  
      null);  
  
      // Initialize the BOA  
      org.omg.CORBA.BOA boa = orb.BOA_init();  
  
      // Create the Count object  
      CountImpl count = new CountImpl("My Count");  
  
      // Export to the ORB the newly created object  
      boa.obj_is_ready(count);  
  
      // Ready to service requests  
      boa.impl_is_ready();  
    }  
    catch(org.omg.CORBA.SystemException e)  
    { System.err.println(e);  
    }  
  }  
}
```

CountServer

Count Main Server Programm

Initialisierung des ORB, BOA und des Count Objektes.

Die beiden Befehle CountImpl ... und boa.obj erstellen und exportieren eine einzige Objekt Instanz („count“ , „My Count“).

// CountImpl.java: The Count Implementation

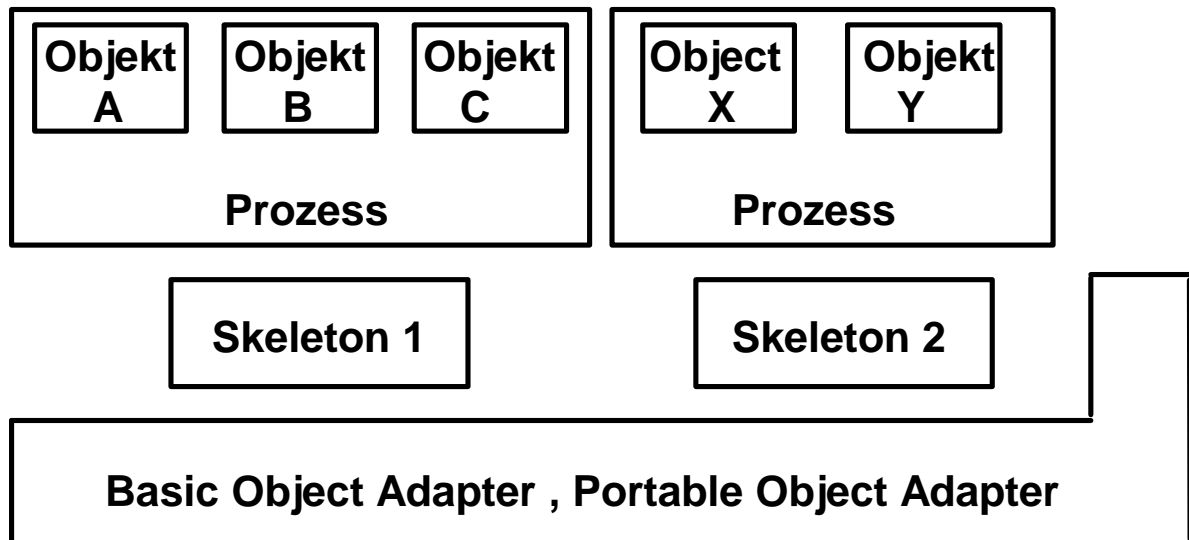
```
class CountImpl extends Counter._CountImplBase  
// „extends“ establishes inheritance from Counter._CountImplBase
```

```
{  
private int sum;  
  
// Constructor, calls ist super, the CORBA skeleton class  
CountImpl(String name)  
{ super(name);  
  System.out.println("Count Object Created");  
  sum = 0;  
}  
  
// get sum  
public int sum()  
{ return sum;  
}  
  
// set sum  
public void sum(int val)  
{ sum = val;  
}  
  
// increment method  
public int increment()  
{ sum++;  
  return sum;  
}  
}
```

CountImpl

Count Server Implementierung

CORBA Server



CORBA Unterscheidet zwischen Server und Objekt

Objekt implementiert Schnittstelle

Server beinhaltet typischerweise mehrfache Objekte (shared server), z.B. alle von der gleichen Klasse

Der Server wird erstmalig aktiviert, wenn ein Request für eines seiner Objekte eintrifft. Weitere Requests werden auf einer „first come, first serve“ Basis abgearbeitet. Typischerweise 1 Thread pro Objekt.

(Ein „Persistent Server“ ist ein shared Server, der die ganze Zeit aktiv ist, z.B. für Transaktionsverarbeitung.)

Schritte zur Programmausführung

- 1. Erstellung der Dateien**
Count.idl
CountClient.java
CountServer.java
CountImpl.java
Programmieren der CORBA Anwendung
- 2. idl2java Count.Idl**
Schnittstellen
Aufruf des IDL Compilers
Es werden die Beschreibungen in Java erzeugt
- 3. javac CountClient.java**
javac CountServer.java
javac CountImpl.java
*compiliert die *.java Programme, erzeugt *.class Programme*
- 4. start osagent**
startet den Name Server
- 5. start java CountServer**
startet den Server
- 6. java CountClient**
Programm
ruft das Klienten auf.

Java Client/Server-Modell

Auf dem Klienten ist nur eine Java Laufzeitumgebung vorhanden. Eine Klientenanwendung wird bei Bedarf dynamisch vom Anwendungsserver heruntergeladen.

Anwendungen können bei Bedarf in einem Cache-Bereich des Hauptspeichers oder des lokalen Plattenspeichers zwischengespeichert werden.

Alle Datenzugriffe erfolgen ebenfalls über das Netz.

Im Gegensatz zu X-Windows-Terminals oder 3270-Terminals läuft die Anwendung auf dem Klienten.

Laut Gardner Group beträgt die „Cost of Ownership“ eines Fat Client 11 900 \$ / Jahr und Arbeitsplatz. Die Cost of Ownership eines Thin Client wird von Sun auf 2 500 \$ / Jahr und Arbeitsplatz geschätzt.

Ein Benutzer kann sich von einem beliebigen Intranet-Zugang aus anmelden und hat einen Zugriff auf seine vollständige Benutzerumgebung. Reduzierung administrativer Kosten; Einwählmöglichkeit von unterwegs oder von zu Hause.

Anwendungsprogramme sind sicherer als Systeme zur Ausführung von Maschinencode. Java Laufzeitsystem stellt sicher, daß geladene Anwendungen, die auf den Klienten geladen werden, die Integrität der Umgebung nicht verletzen.

Leichte Portierbarkeit der Anwendungen auf unterschiedliche Systemplattformen.

```
<h1>Count Client Applet</h1>
<hr>
<center>
<APPLET CODE=CountClientApplet.class WIDTH=300 HEIGHT=60
  CODEBASE=classes>
  <param name=org.omg.CORBA.ORBClass value=com.visigenic.vbroker.orb.ORB>
</APPLET>
</center>
<hr>
```

HTML Code für den Count Client Applet Aufruf

```

// CountClientApplet.java Applet Client, VisiBroker for Java
import java.awt.*;
public class CountClientApplet extends java.applet.Applet
{ private TextField countField, pingTimeField;
  private Button runCount;
  private Counter.Count counter;
  public void init()
  { // Create a 2 by 2 grid of widgets.
    setLayout(new GridLayout(2, 2, 10, 10));
// Add the four widgets, initialize where necessary
    add(new Label("Count"));
    add(countField = new TextField());
    countField.setText("1000");
    add(runCount = new Button("Run"));
    add(pingTimeField = new TextField());
    pingTimeField.setEditable(false);
try
  { // Initialize the ORB.
    showStatus("Initializing the ORB");
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this, null);
// Bind to the Count Object
    showStatus("Binding to Count Object");
    counter = Counter.CountHelper.bind(orb, "My Count");
  } catch(org.omg.CORBA.SystemException e)
  {
    showStatus("Applet Exception" + e);
    e.printStackTrace(System.out);
  }
}
public boolean action(Event ev, Object arg)
{ if(ev.target == runCount)
  { try
    { // Set Sum to initial value of 0
      showStatus("Setting Sum to 0");
      counter.sum((int)0);
// get data from and set value of applet fields
      showStatus("Incrementing");
      int stopCount = Integer.parseInt(countField.getText());
      pingTimeField.setText(" ");
// Calculate Start time
      long startTime = System.currentTimeMillis();
// Increment stopCount times
      for (int i = 0 ; i < stopCount ; i++ )
        { counter.increment();
        }
// Calculate stop time; show statistics
      long stopTime = System.currentTimeMillis();
      pingTimeField.setText("Avg Ping = "
        + Float.toString((float)(stopTime- startTime)/stopCount)
        + " msecs");
      showStatus("Sum = " + counter.sum());
    } catch(org.omg.CORBA.SystemException e)
    { showStatus("System Exception" + e);
      e.printStackTrace();
    }
    return true;
  }
  return false;
}
}

```

Count Client Applet

Realisierung des ORB:

Der CORBA-Standard gestattet viele Wege, einen ORB zu realisieren:

- Werden durch ein lokales Betriebssystem und das Netzwerk geeignete Kommunikationsmechanismen bereitgestellt, kann der ORB als Sammlung von Routinen zur Umsetzung der ORB-Schnittstellen in Systemaufrufe (sowohl auf der Client- als auch auf der Server-Seite) realisiert werden. Zum Transport von Requests und Responses wird ein vorhandenes Protokoll mit dem jeweiligen Adressierungsstil genutzt.
- Die gesamte ORB-Funktionalität kann auf einem zentralen Server konzentriert sein, der alle Client-Requests entgegennimmt und an jeweilige Objekt-Implementation weiterleitet.
- Der ORB kann als Bestandteil des lokalen Betriebssystems realisiert werden (insbes. wenn dieses für verteilte Anwendungen konzipiert ist, z.B. *Spring* von Sun Microsystems).

CORBA Services

CORBA Services sind modulare zusätzliche System-Dienstleistungen, welche die Funktionalität des ORB ergänzen. Sie stellen Bausteine für die Anwendungsentwicklung dar. Es existieren Spezifikationen der OMG für:

- **Naming Service**
- **Persistence Service**
Speicherung von Objekten in relationalen oder Objekt Datenbanken oder einfachen Dateien
- **Concurrency**
Lock Manager Dienste für Transaktionen oder Threads
- **Transaction Service**
Two-Phase Commit Koordination für flache oder nested Transactions

sowie weitere Dienste wie Security, Message, Time, Event, Trader,

OMG CORBA Namensdienst

Mechanismus, mit dem Objekte auf dem ORB andere Objekte lokalisieren

Ein „Name-Binding“ ist eine Zuordnung Name - Objekt.

Verwendet hierarchische Struktur. Basiert auf transparenter Kapselung existierender Namensdienste wie LDAP, X.500, DNS oder Sun NIS.

Bildet den Namen eines Objektes auf eine eindeutige, maschinenlesbare Objekt Referenz ab.

Es kann mit absoluten Namen oder mit Namen innerhalb ihres Context gearbeitet werden. spruth, informatik, uni-tuebingen und de könnten Komponenten eines CORBA Namens sein.

Eine Komponente besteht aus 2 Attributen: identifier, kind. Das Attribute kind kann eine Beschreibung der Komponente enthalten, z.B. file-typ.

Einige ORB Hersteller, z.B. Inprise, Orbix, bieten zusätzlich eigene, proprietäre Namendienste an, die Broadcasts verwenden und nur innerhalb des gleichen IP Subnetzes arbeiteten.

Persistenz

Persistenz beschreibt etwas, das über seine erwartete Lebensdauer hinaus existiert oder auch nach der Programmausführung noch vorhanden ist (beispielsweise die Zuordnung eines Geräte-Laufwerksbuchstabens).

In einer objektorientierten Sprache beschreibt Persistenz Objekte, die außerhalb des Gültigkeitsbereichs des Programms, das sie erzeugt hat, existieren, und zwar hinsichtlich Zeit und Ort. Ein persistentes Objekt kann in einer Datei oder einer Datenbank abgelegt und später wieder benutzt werden, oder es kann auf eine andere Maschine übertragen werden.

Um ein Objekt persistent zu machen ist ein Mechanismus notwendig, der das Objekt in eine Form umwandelt, in der es in eine Datei oder in eine Datenbank abgelegt werden und später aus dieser Form das Objekt wieder erzeugen kann.

Persistenz wird in der Regel implementiert, indem der Status (die Attribute) eines Objekts zwischen den einzelnen Programmausführungen gespeichert wird. Wenn das Objekt erneut benötigt wird, wird es aus seiner gespeicherten Form wieder hergestellt. Der Herstellungsprozeß erzeugt ein neues Objekt, das mit dem ursprünglichen identisch ist. Das wiederhergestellte Objekt ist zwar nicht das selbe Objekt, aber sein Status und sein Verhalten sind identisch.

Bei der Persistenz werden den gespeicherten Daten alle Objektattribute (etwa Klassenname, Feldname und Zugriffs-Modifier) zugeordnet, so daß verhindert wird, daß die Daten versehentlich mit einem falschen Objekttyp abgelegt werden.

Vergleich DCE - CORBA

DCE arbeitet mit Prozeduren, CORBA arbeitet mit Objekten

IDL hat ähnliche Funktion. DCE IDL basiert auf C

Die OMA plant Object Services, die den DCE Naming Services und DCE Security Services vergleichbar sind

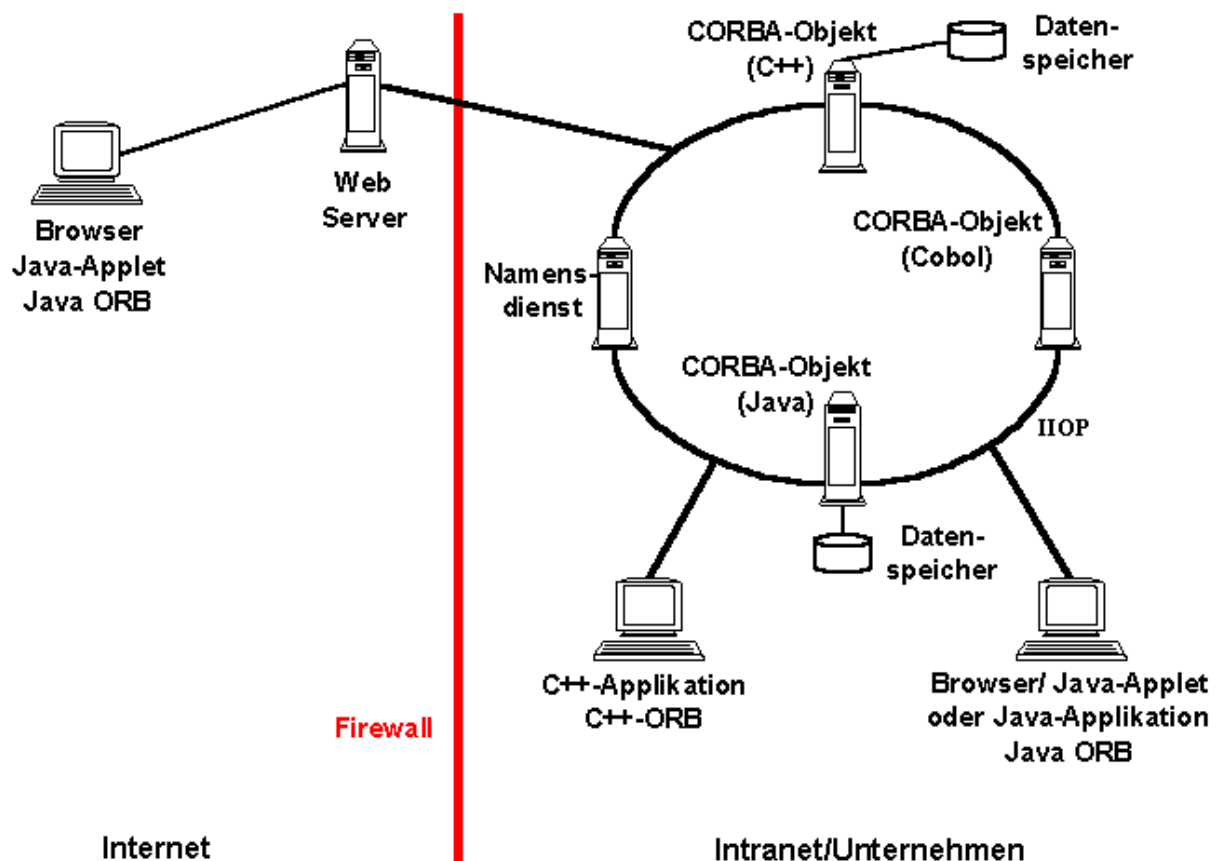
CORBA mit Visibroker

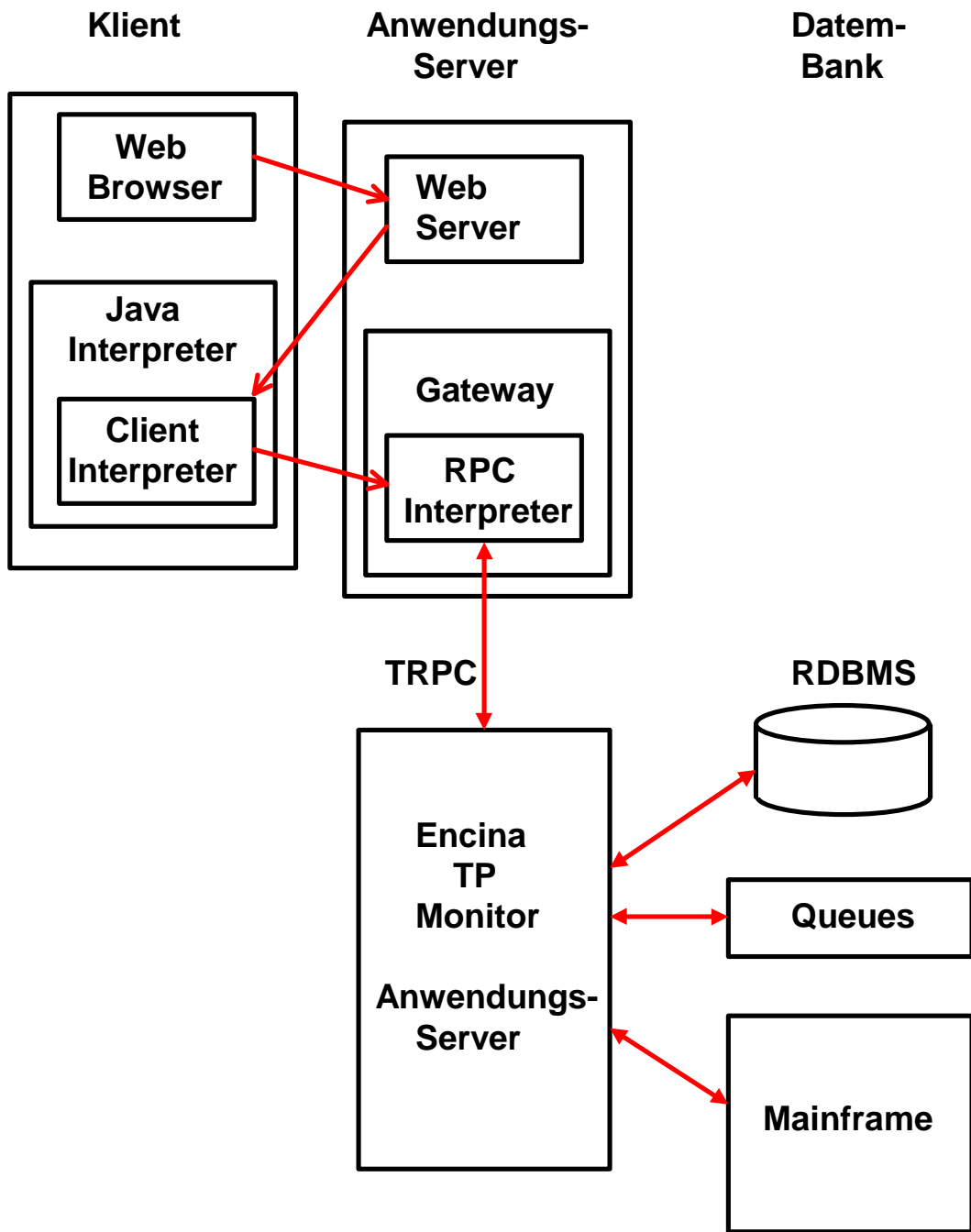
Unmittelbare Verfügbarkeit von CORBA durch die Sprachanbindung an Java.

ORB in Java implementiert – wird mit Applet heruntergeladen oder ist (Netscape Communicator 4.xx) im Browser bereits integriert

Serverobjekte in beliebigen Sprachen implementierbar. (Language Mapping zu IDL muß existieren)

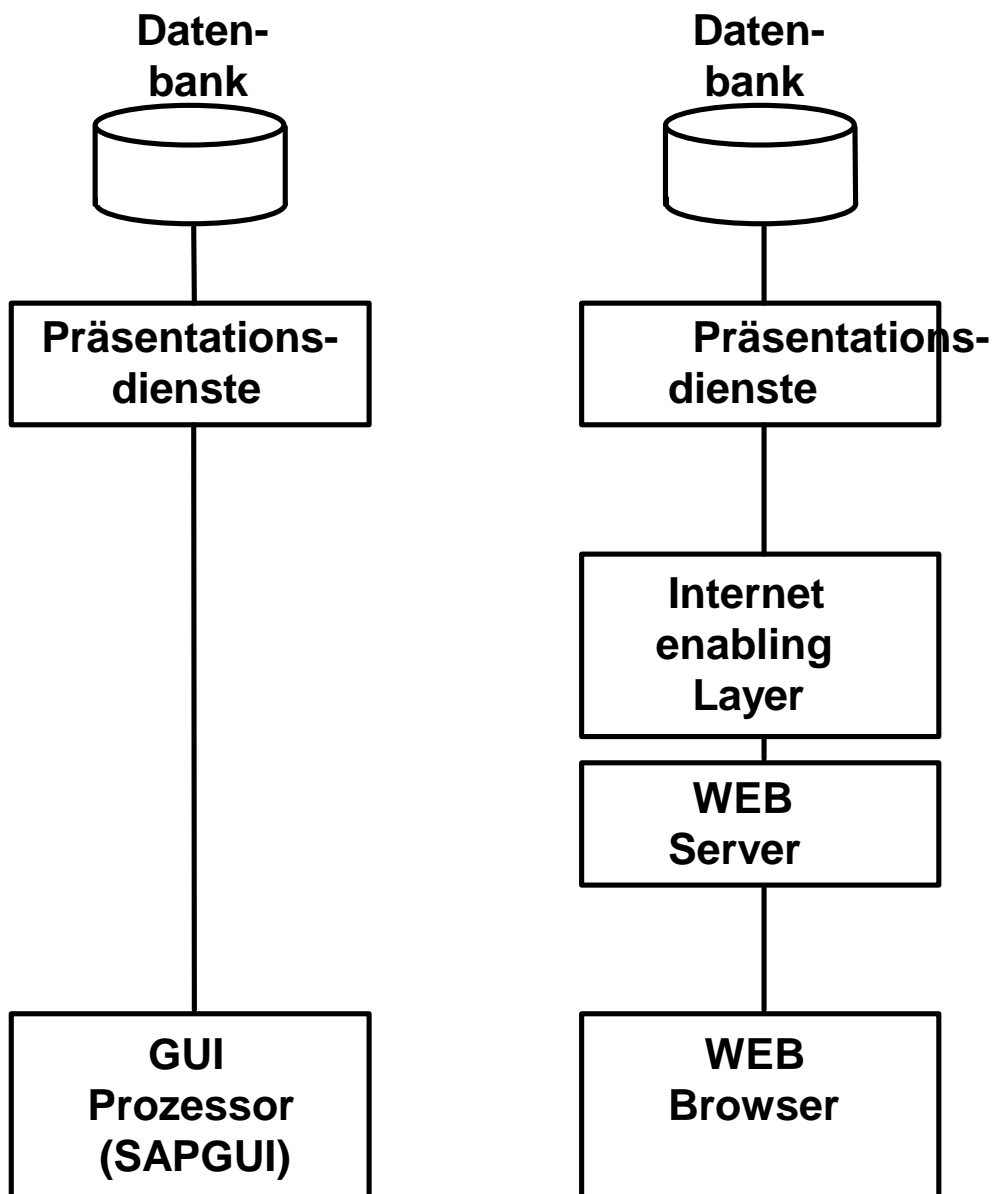
Internetzugriffe über einen integrierten Webserver – dadurch zentrale Haltung der Anwendungen, zentrale Konfiguration und Wartung der öffentlichen Programme.





Java Thin Client für den Encina TP Monitor

Client enthält keine Stubs, sendet Nachricht an RPC Interpreter.
Dieser übersetzt Nachricht in PRPC oder TRPC.



GUI Implementierung mit Hilfe eines WEB Browsers

Microsoft DCOM

Anderer Name: Objekt RPC (ORPC)

Microsoft's proprietäre Alternative zu CORBA.

Verwendet (leicht modifizierten) DCE RPC (MRPC).

integriert wie CORBA Komponenten unterschiedlicher Hersteller in binärer Form, die in unterschiedlichen Sprachen geschrieben wurden. Verwendet hierzu eine IDL.

Keine Vererbung, statt dessen „Aggregation“.

Verfügbar (in naher Zukunft) unter OS/390 (MVS), OpenVMS und den meisten Unix Dialekten (incl. LINUX).

Implementierte CORBA Systeme

März 1999

Bank of America	Customer Service System	Visigenic, Hitachi Open TP Broker Object Transaction Manager
Credit Suisse		IONA for MVS
Deutsche Morgan Grenfell	Information for remote Trading locations	IONA
Dresdner Bank	Strategic Arbitrage and Risc Mgt. Syst.	
Capital One Financial Corp.	Credit Card. Transact.	IONA, MQ Series, Oracle Persistence Mapping
	Mehrere Mill. Tx/Tag	SW, Encina, Unix
First Union Corp.	Zugriff auf Kunden Daten (Bank und Versicherung) Component Reuse	Visigenic, OS/390, Object Bridge (CORBA-DCOM)
Ericson		IONA
American Airlines	Middleware between HTTP WEB Server und SABRE, dazu CARGO Mgr.	IONA
www.bahn.hafas.de	1000 Anfragen/h	IONA Orbix WEB

Microsoft DCOM

Microsoft's proprietäre Alternative zu CORBA

Verwendet DCE RPC

Keine Vererbung

Microsoft ActiveX

Integration der Internet-Technologie mit der MS Objekt-Architektur

Beinhaltet DCOM, OLE/OCX, Scripting

ActiveX Controls enthalten Binärcode ausführbarer Programme. Laufen schnell im Gegensatz zu Java Byte Code. Allerdings große Dateien, verringerte Sicherheit.

MS Internet Explorer ist ein ActiveX-fähiger Web Browser. Produkte wie VisiBridge der Fa. Visigenic ermöglichen es ActiveX-Klienten, auf CORBA-Server zuzugreifen

MINFU's

Microsoft Nomenclature Foul Up's

COM (Component Object Model)

Ursprünglich ein Transport Mechanismus für lokale Objekte. Der Transportmechanismus für verteilte Objekte hieß ursprünglich DCOM (Distributed COM). Heute wird DCOM als COM bezeichnet.

COM+

Windows 2000 Version von COM (mit Erweiterungen)

OLE (Object Linking and Embedding)

1991-1994 High Level Interface Protokoll für zusammengesetzte Dokumente, die COM als Transport Mechanismus verwenden

1994-1996 Schnittstelle plus COM

1996-2000 Schnittstelle unabhängig von COM

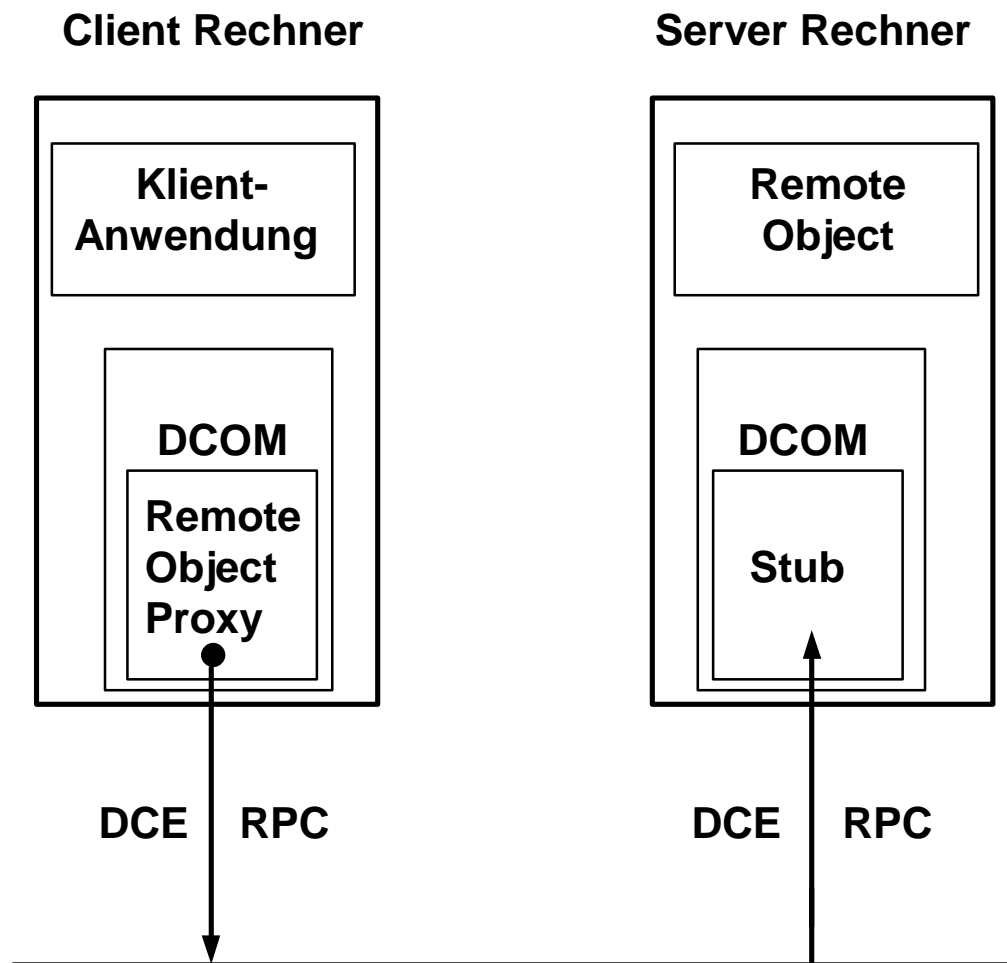
OLE for Life Insurance (OLifE) ist ein COM Industriestandard der Lebensversicherer, etabliert zwischen 1994 - 1996.

DNA (Distributed Network Architecture)

???????

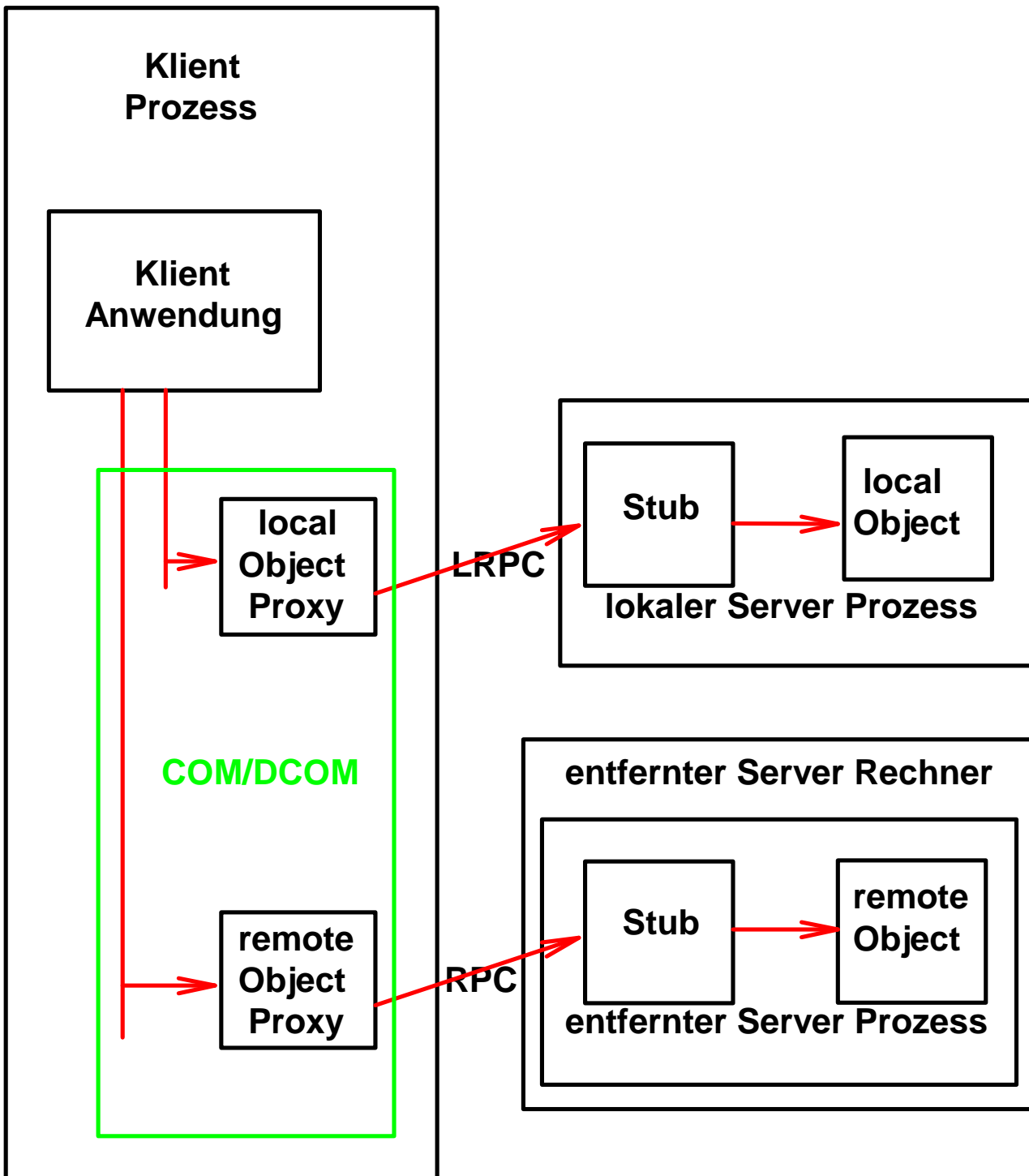
ActiveX

Eingetragener Microsoft Markenname mit stark wechselnder Semantik.



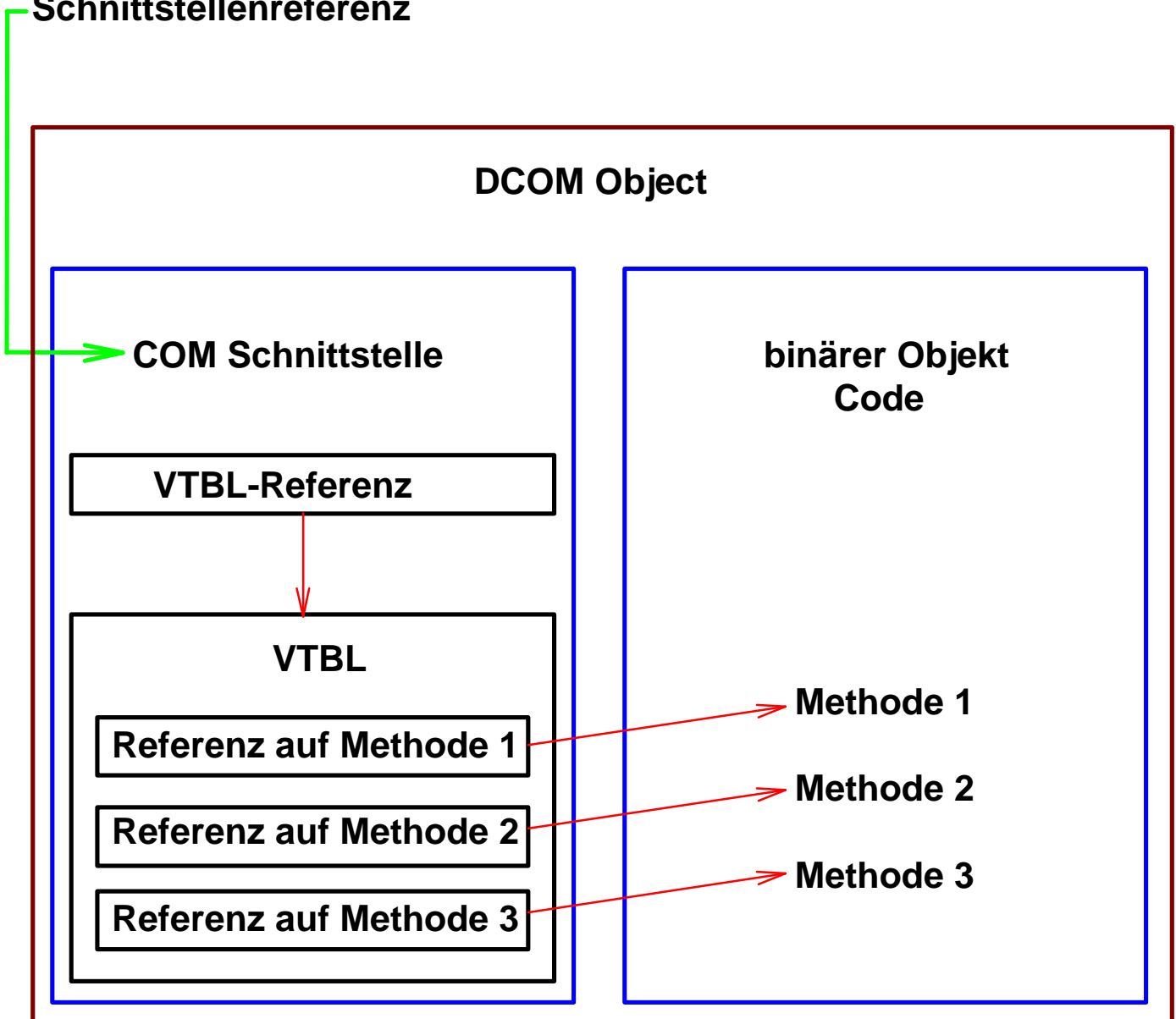
DCOM Kommunikation über Remote Procedure Calls

Arbeitet der Server auf dem gleichen Rechner wie der Klient, werden „Lightweight RPC's“ eingesetzt (LPC, Local Procedure Call, des NT Überwachers).



COM/DCOM kommunizieren über den (DCE) RPC. Befinden sich Klient und Server auf der gleichen Maschine, werden Lightweight RPC's (LRPC) eingesetzt.

Schnittstellenreferenz



**Eine DCOM Schnittstelle ist eine Referenz auf eine Tabelle
Diese enthält Referenzen auf Methoden**

DCOM Schnittstelle

Der Klient benutzt die Schnittstelle nur durch den v-table, nie direkt durch einen Methodenaufruf.

Die ersten drei Elemente der v-table sind immer:

- QueryInterface
- AddRef
- Release

Die „IUnknown“ Schnittstelle kennt nur diese drei Methoden. Alle anderen Schnittstellen „erben“ von IUnknown.

AddRef und Release implementieren die Lebenszeitverwaltung eines Objektes. Addref wird bei jedem Zugriff durch einen Klienten automatisch aufgerufen, und inkrementiert einen Referenzzähler. Klient muß Release selbst aufrufen (dekrementiert den Referenzzähler).

Literatur: D.N.Gray et al. „Modern Languages“. Communications of the ACM, May 1998, Vol. 41, No.5, S. 55.

DCOM Vererbung

Aggregation

A Component encapsulates Services of other Components

Vererbung wird durch ein Web von Zeigern erreicht, welche unterschiedliche Schnittstellen miteinander verbinden (aggregate).

Beispiel: Komponente A bietet Methoden an, die in Komponente B verfügbar sind. Hierzu bietet A eine Schnittstelle für den Methodenaufruf in B an. Ein Aufruf dieser Methode wird von A an B zur Verarbeitung weitergereicht.

DCOM Leistungsverhalten

**Beispiel: Pentium 120 Mhz, NT 4.0 Betriebssystem,
50 Bytes an Parametern zwischen 2 Prozessen
transportieren**

- 1. Transport zwischen 2 Prozessen auf dem gleichen
Rechner ohne DCOM
3 Millionen Aufrufe/s**
- 2. Transport zwischen 2 Prozessen auf dem gleichen
Rechner unter Verwendung von DCOM und dem MT LPC
2000 Aufrufe/s**
- 3. Transport mit DCOM zwischen 2 Prozessen auf
unterschiedlichen Rechnern
500 Aufrufe/s**

Microsoft Object Request Broker (ORB)

DCOM-Gegenstück zu CORBA ist der Microsoft ORB

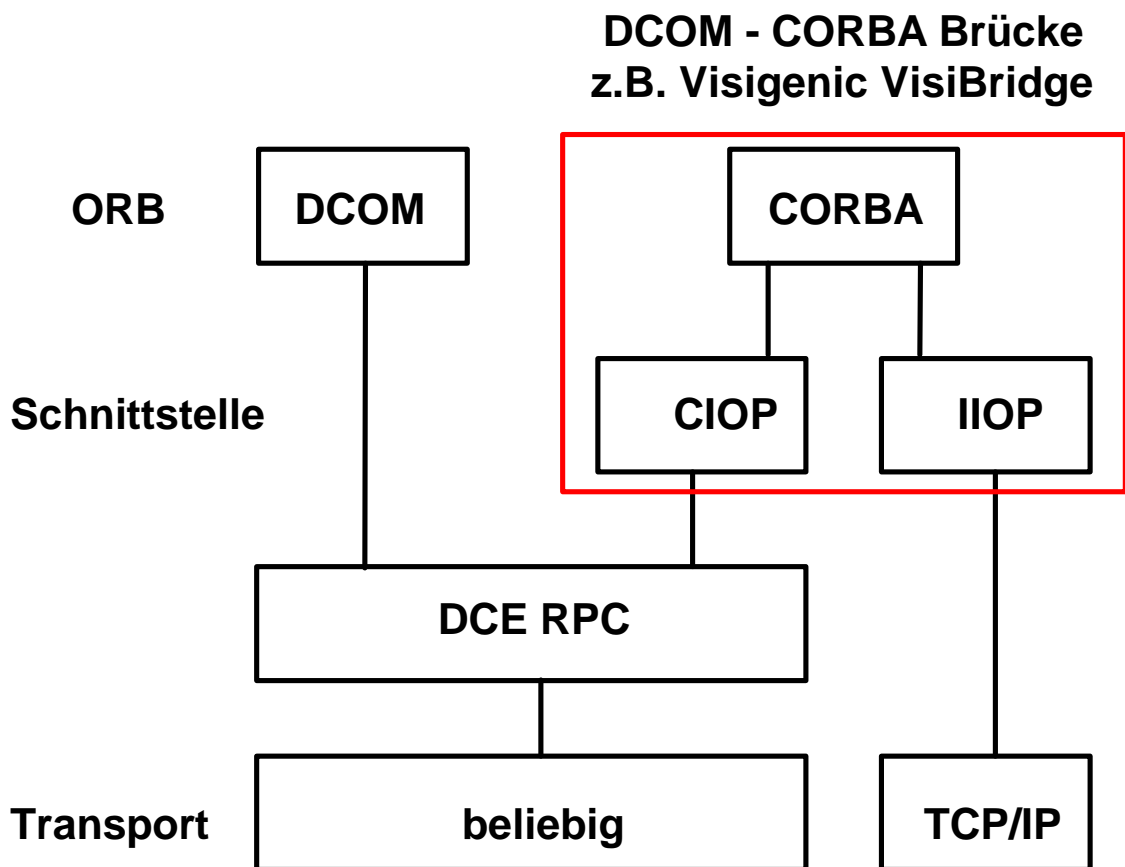
CORBA-inkompatibel

Vermittlung von Methodenaufrufen zwischen binären Objekten

DCOM = Objektmodell

OLE = Komponentenarchitektur

Basiert auf DCE



Vergleich CORBA - DCOM

Beide Technologien werden vermutlich für längere Zeit in Wettbewerb miteinander stehen.

In reinrassigen Microsoft Umgebungen (z.B. mittelständische Unternehmen) dürfte DCOM überlegen sein.

In heterogenen Umgebungen ist DCOM bisher wenig vertreten.

CORBA ist besser geeignet, bestehende Altanwendungen einzubinden.

Es ist denkbar, auf weitverbreitete Echtzeitbetriebssysteme wie VxWorks oder Psos eine CORBA basierte Anwendung aufzusetzen.

Java Browser Sicherheit (Sandbox) ist besser als DCOM (vertrauensbasierte Verfahren).

Gartner: DCOM wird 2002 den CORBA Stand von 1998 erreichen