-Microsofts Welt-

Alex Koch
Michael Pietz
Stefan Seichter

Inhaltsverzeichnis

Seite

1 Abbildungsverzeichnis 2
Microsofts DCOM Modell 3
2 Einleitung 3
3 Objekte, Schnittstellen und Methoden 3
4 Verteilte Kommunikation 4
5 Wo bist Du 6
6 IUnknow 6
7 Version 9999 8
Microsofts .Net Strategie 9
8 Einleitung 9
9 XML und Webdienste 9
10 Komponenten 9
10.1 Dienstplattform 9
10.2 .Net-Framework 10
10.3 .Net Bausteindienste 10
10.4 .Net-Orchestration 10
10.5 .Net-Unternehmensserverfamilie 10
11 Interessante Neuerungen – Das .Net Framework 10
11.1 .Net Programmiersprachen 10
11.2 Metasprache 11
11.3 Gemeinsame Regel 11
11.4 Common Language Runtime (CLR) 12
12 .Net und COM, DCOM, COM+ 12
13 Das .Net Remoting Framework 12
13.1 Verteilter Kontakt 12
13.2 Das Proxy Konzept 12
13.3 Kommunikationskanäle 13
13.3 Aktivierung 13
14 Fazit 13
Vergleich der Programmierung einer DCOM und RMI-Anwendung 15
15 Einführung 15
16 Vergleich der Programmierung 15
16.1 Programmierung einer RMI- Anwendung 15
16.2 Programmierung einer DCOM - Anwendung 19
17 Fazit 26
18 Literaturverzeichnis 28

1 Abbildungsverzeichnis

Seite

Abbildung 1: Vtable 3
Abbildung 2: DCOM Objekt 4
Abbildung 3: In Process Server.....4
Abbildung 4: Local Server.....5
Abbildung 5: Remote Server..... 5
Abbildung 6: Marshalling..... 6
Abbildung 7: Versionierung.....8
Abbildung 8: .Net Architektur..... 11

Microsofts DCOM Modell

Michael Pietz, Juni 2001

2 Einleitung

Aufgrund der stetig wachsenden Größe von Softwareprogrammen und der damit zunehmenden Komplexität ist der Softwareentwicklungsbereich kontinuierlich auf der Suche nach effizienten Lösungen zur Verbesserung des Programmieraufwandes. Ein wichtiges Ziel, das sich die Programmierer gesetzt haben, ist, einen Programmcode, der in verschiedenen Programmiersprachen wie z.B.: C; C+; Java; Visual Basic usw. geschriebenen wurde, miteinander kommunizieren zu lassen. Die Möglichkeit, auch von entfernten Rechnern auf vorhandenen Code zuzugreifen und dessen Methoden bzw. Funktionen zu nutzen, bieten Platz für neue Visionen bei der Softwareentwicklung.

Ein erster Lösungsansatz ist die Komponententechnologie. Die Firma Microsoft trägt mit ihrem Distributed Component Object Modell (DCOM) -Kommunikationsmodell mit dazu bei, diese Visionen umzusetzen/durchzusetzen.

Diese neuen Entwicklungen ziehen jedoch neue Probleme mit sich, die berücksichtigt und eingeplant werden müssen beim Einsatz dieser sogenannten Middleware:

- Die Skalierbarkeit (wenn z.B. eine große Anzahl Clients mit ihrem Server kommunizieren und diesen dabei längere Zeit beanspruchen und nicht wieder freigeben)
- Verteilung der Prozessauslastung auf mehrere Rechner
- Die Verwaltung von Transaktionen
- Das erforderliche Sicherheitsmanagement bei einem Zugriff auf verteilte Rechnersysteme

DCOM wurde durch Erweiterungen aus dem Vorgänger COM entwickelt und somit um die Möglichkeit der Verteilung der Komponenten auf verschiedene Knoten in einem Netzwerk ausgebaut. COM ist das Komponentenmodell der Windows-Betriebssystemfamilie und wurde für die Verwaltung von Verbunddokumenten und die komponentenorientierter Anwendungsprogrammierung entwickelt.

3 Objekte, Schnittstellen und Methoden

DCOM ist ein binärer Standard. Dieser bietet eine virtuelle Funktionstabelle ("Vtable", siehe Abb. 1), über die programmiersprachenunabhängig, mittels Zeiger, die gewünschte Funktion der Komponenten angesprochen werden kann.

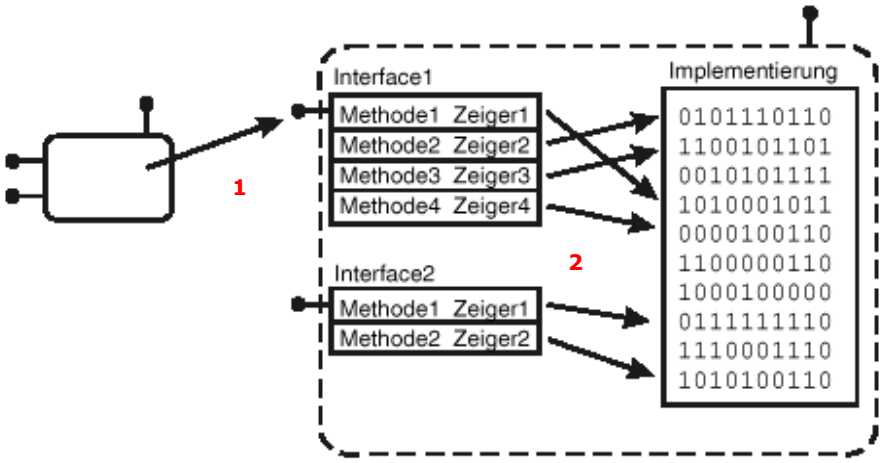


Abbildung 1: Vtable

Quelle: http://www.informatik.uni-bremen.de/grp/flowtec/papers/doc/dcom/html/

Zeiger 1: Der Interfacezeiger (Schnittstellenzeiger) verweist auf die virtuelle Tabelle, die weitere Zeiger beinhaltet (einen Zeiger für jede Methode dieser Schnittstelle).

Zeiger 2: Der Methodenzeiger verweist auf den Implementierungscode, der den Dienst tatsächlich ausführt.

Ein DCOM Objekt wird nicht über den Namen identifiziert, sondern wird über eine eindeutige, 128 Bit große Nummer angesprochen. Diese Nummer (abhängig von der Computerprozessornummer und einem Zeitstempel) wird von einem Softwaredienstprogramm generiert. In der Praxis können somit keine zwei Objekte mit der gleichen Identifikationsnummer existieren.

Ein Objekt kann mehrere Schnittstellen (Interfaces) bereitstellen und eine Schnittstelle kann wiederum mehrere Methoden besitzen. Auch die Schnittstelle eines Objektes wird über eine eindeutige Nummer angesprochen, die auf die selbe Art erzeugt wird wie die des Objektes.

- Die Identifikationsnummer eines Objektes wird als "Globally Unique Identifier" (GUID) bezeichnet
- Die Klassenidentifikation als "Class ID" (CLSID)
- Die einer Schnittstelle als "Interface Identificator" (IID)

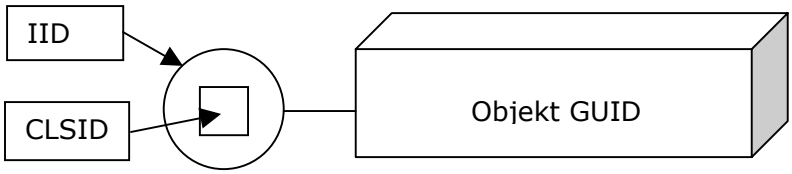


Abbildung 2: DCOM Objekt

4 Verteilte Kommunikation

DCOM ist eine Erweiterung des Component Object Model (COM). COM definiert, auf welche Weise Komponenten mit einem Client ohne zwischengeschalteten Vermittler kommunizieren können.

Es können innerhalb von COM/DCOM drei verschiedene Serverarten unterschieden werden:

1) "In Process Servers" → werden im selben Speicheradreßraum wie ihre Clients ausgeführt. Der Client kommuniziert direkt mit der Komponente und ruft Methoden dieser auf.

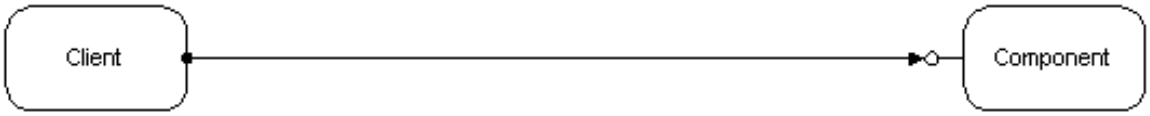


Abbildung 3: In Process Server

Quelle: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

2) "Local Server" → werden in einem separaten Prozeß auf derselben Maschine wie der Client ausgeführt, die Kommunikation erfolgt über "Light Remote Procedure Calls" (LRPCs), eine vereinfachte Form der "Remote Procedure Calls". Möchte ein Client mit einer Komponente eines anderen/fremden Prozesses kommunizieren, so

kann er diese Komponente nicht direkt ansprechen, sondern muß den Weg des Kommunikationsmodells einhalten.

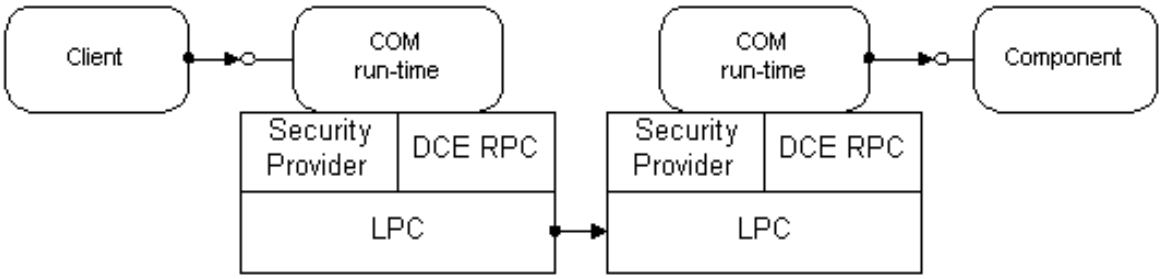


Abbildung 4: Local Server

Quelle: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

3) "Remote Servers" → werden in einem separaten Prozeß auf einer anderen Maschine ausgeführt, dies ist die weitestgehende Verteilung von Objekten. Die Kommunikation erfolgt über "Remote Procedure Calls" (RPC), das dem Protokoll der "Distributed Computing Environment (DCE) der "Open Software Foundation" (OSF) weitgehend entspricht. DCOM ersetzt in diesem Fall die LRPCs und nutzt Netzwerkprotokolle wie z.B: TCP/IP; HTTP; IPX/SPX; NetBIOS; UDP oder Apple Talk für die Kommunikation über die "etwas längere" Leitung.

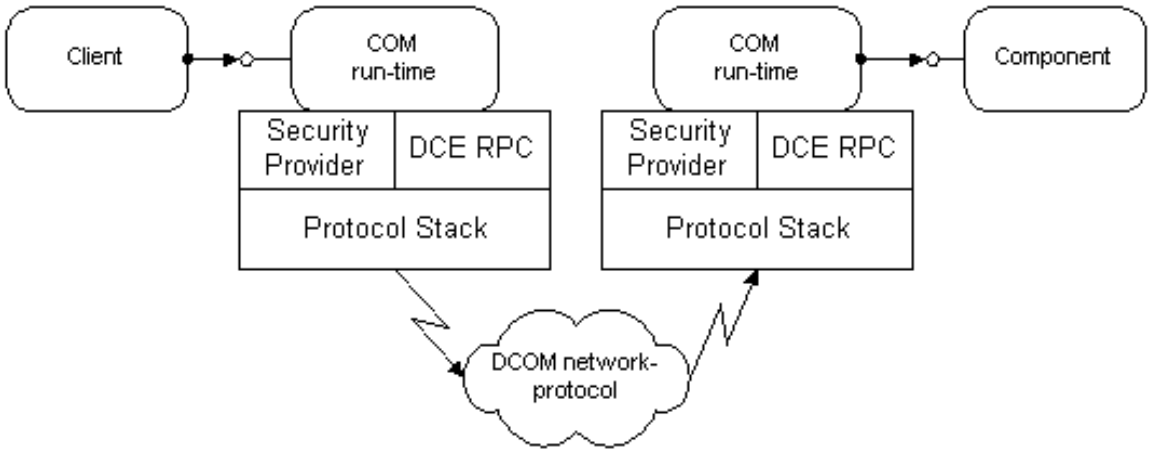


Abbildung 5: Remote Server

Quelle: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

Ein Methodenaufruf ist für das aufrufende Objekt (Client-Object) transparent und damit auch die Lokalität des aufgerufenen Objekts (Server-Object) unerheblich.

Der Client ruft die Methoden eines Stellvertreter (Proxy) auf, dieser besitzt die gleichen Methoden und Schnittstellen wie das entfernt lokalisierte Server-Object. Das Proxy-Object verpackt nun die übergebenen Parameter für die Übertragung ("Marshalling"). Die Übertragung erfolgt dann per RPC. Ein Stub-Object packt die Parameter auf dem entfernten Rechner aus ("Unmarshaling") und ruft die Methode des Server-Objects auf. Die Antworten/Ergebnisse werden dann mit demselben Verfahren wieder zurück zum Client transportiert.

Da das Client-Object nicht zwischen einem Server-Object und einem Proxy-Object unterscheiden kann und das Server Object nicht ein Client-Object von einem Stub unterscheiden kann, ist die verteilte Komponentenkommunikation für DCOM Objecte vollkommen transparent.

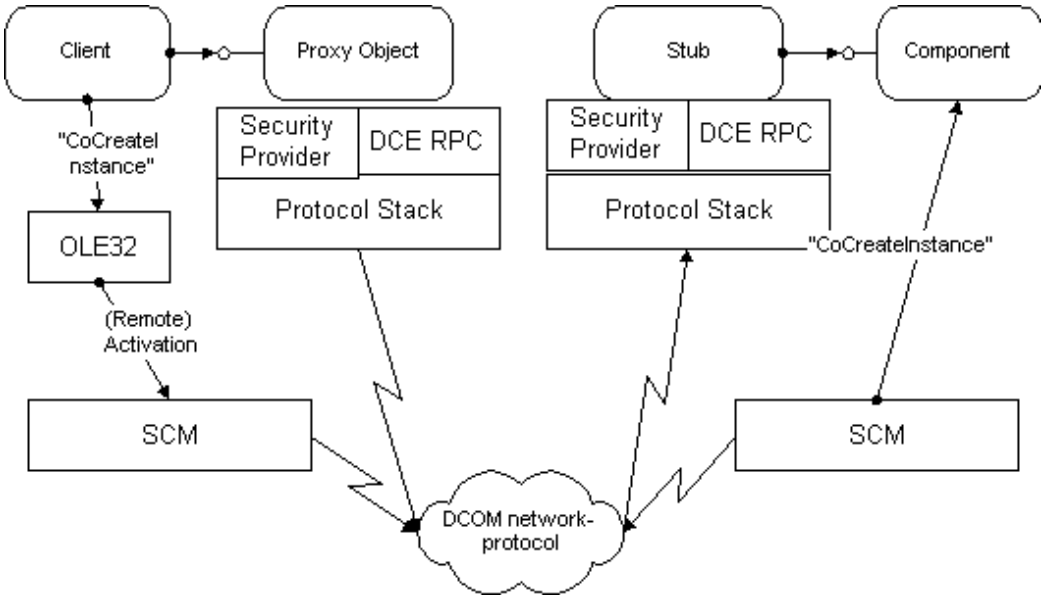


Abbildung 6: Marshalling

Quelle: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp

Anmerkung:
Ein Remote Procedure Call (RPC) ist ein synchroner Kontrollfluß, bei dem der aufrufende Client wartet, bis der Server die Prozedur beendet hat und eine Antwort bzw. das Ergebnis zurückgibt. Die Datenübergabe in Form von Prozeduraufrufen und von aktuellen Parametern zwischen Programmen in unterschiedlichen Adreßräumen ist langsamer als die Kommunikation über den Interaktionspfad des eigenen Rechners. Der RPC Mechanismus läßt sich im OSI Schichtenmodell in Schicht 7 (Anwendungsschicht) einordnen.

5 Wo bist Du

Um in einem Netzwerk einen bestimmten Rechner auszuwählen, auf den bzw. dessen Komponenten zugegriffen werden soll, bietet DCOM verschiedene alternative Verfahren. Die Identifizierung von entfernten Rechnern kann über die in TCP/IP verwendeten Domänennamen, die IP Adressen, NetBIOS Namen oder die IPX/SPX Namen der Firma Novell erfolgen.

6 IUnknow

Jedes DCOM Objekt muß mindestens die Schnittstelle "IUnknow" besitzen, wobei der Buchstabe "I" diese als Schnitstelle (Interface) kennzeichnet. Die IUnknow Schnittstelle bestizt die folgenden drei Methoden:

1) *QueryInterface*, mit dieser Methode kann ein Client-Object zur Laufzeit bestimmen, ob das angesprochene Server-Object eine weitere Schnittstelle anbietet und bekommt, wenn es so sein sollte, den Schnittstellenzeiger zurück.
2) *AddRef*, ist zur Erhöhung des Referenzzeigers (der Schnittstellenzeiger wurde vom Server-Object an das Client-Object weitergegeben) um den Wert eins.

3) *Release*, sobald die Schnittstelle vom Client-Object nicht mehr benötigt wird, wird die Methode Release angefordert, die daraufhin das Objekt freigibt.

Jede Schnittstelle muß diese drei Methoden von IUnknow erben.

Wird ein Objekt von keinem anderen Objekt mehr benötigt, was daran zu erkennen ist, dass der Referenzzeiger von der AddRef Methode auf null erniedrigt wurde, so wird das Objekt zerstört. Um zu überprüfen, dass Objekte durch z.B. Systemabstürze, Netzwerk- oder Hardwarefehler nicht mehr zu erreichen sind bzw. zerstört wurden, wird von DCOM das sogenannte "pinging" angewendet. Bei diesem Verfahren sendet der Client alle zwei Minuten (periodisch) eine Nachricht. Bleiben mehr als drei Nachrichten unbeantwortet, d.h. der Sender bekommt keine Antwort, dann wird das Objekt als "nicht mehr existent" angesehen. DCOM erniedrigt den Referenzzeiger und gibt das Objekt frei. Wird das Objekt später noch einmal benötigt, so wird automatisch ein neues Objekt instanziert.

Möchte ein System mit DCOM Objekten arbeiten, so muß es Basisdienste anbieten. Diese Basisdienste sind in der DCOM Bibliothek enthalten und können über Funktionsaufrufe, die mit der Vorsilbe "Co" beginnent, aktiviert werden z.B.:

a) CoCreateInstance(Ex) (…)
   Dieser Basisdienst übergibt die CLSID an die Bibliothek, die dann das entsprechende DCOM Objekt in ihrer Systemregistrierung ausfindig macht und mit Hilfe dieser Registrierung der Datei zuordnet, die den zugehörigen Binärcode enthält.
b) CoGetInstanceFromFile
c) CoGetInstanceFromStorage
d) CoGetClassObject (…)
e) CoGetClassObjectFromURL

Eine Objektinstanz mitsamt den dazugehörigen Daten wird als "Moniker" bezeichnet.

Die Moniker (DCOM-Objekte) sorgen auf unterschiedlichen Wegen dafür, dass bestimmte DCOM Objekte gefunden oder auch aktiviert werden können. Da unter DCOM jeder Zugriff über eine Schnittstelle läuft, wird die spezielle Schnittstelle "IMoniker" für diesen Zweck definiert. Der Moniker erzeugt gleich nach seiner Aktivierung ein eigenes Moniker-Object und sorgt dafür, dass dies in der "Running Object Table" (ROT) –eine Tabelle für alle aktiven Objekte- eingetragen wird. Nun kann jeder interessierte Client in den Besitz eines Zeigers auf den Moniker kommen, sodass der Client jederzeit wieder eine Verbindung zum Server aufbauen kann.

7 Version 9999

Um Fehlerkorrekturen, Leistungsverbesserungen oder Erweiterungen in ein bestehendes System integrieren zu können, ist die Versionierung von Komponenten notwendig. Durch das Einbringen "besserer" Versionen kann die Wartung und Weiterentwicklung von Software gewährleistet werden. Bestehende Schnittstellen dürfen bei einer Versionsänderung nicht geändert werden!

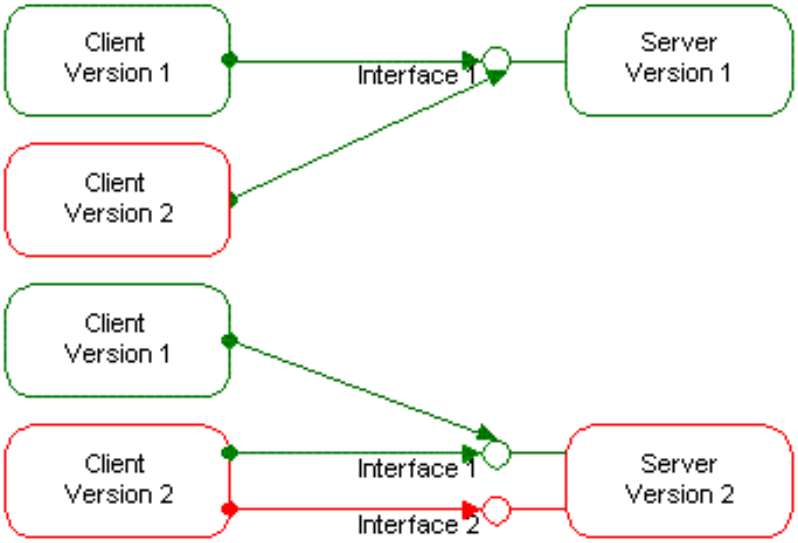


Abbildung 7: Versionierung

Quelle: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomarch.asp

Fall 1:
In der Abbildung soll der Wechsel von Version 1 nach Version 2 eine Leistungsverbesserung durch eine neuere Softwarekomponente darstellen. Beide Versionen greifen trotz neuer Funktionalität der Version 2 auf die gleichen Schnittstellen des Servers zu.

Fall 2:
Hier wurde der Server aktualisiert (neue Methoden hinzugefügt) und dabei um eine zusätzliche Schnittstelle erweitert. Der Client der mit Version 1 arbeitet, kommuniziert problemlos über die alte Schnittstelle, nach dem Softwareupdate des Clients kann dieser zusätzlich auch über die neuere Schnittstelle (Interface 2) mit dem Server kommunizieren.

Für neue Versionen von DCOM Objekten gilt die Forderung, daß keine neuen Methoden zu bereits existierenden Schnittstellen hinzugefügt werden dürfen, sondern dass für neue Methoden zusätzliche Schnittstellen definiert werden müssen.

Durch diese Abbildung wird deutlich, dass es möglich ist, "neue" Clients zu entwickeln, die mit "alten" Servern arbeiten und dass "alte" Server die "neueren" Clients bedienen können.

Microsofts .Net Strategie

Stefan Seichter, Juni 2001

8 Einleitung

Im Gegensatz zu COM, DCOM und COM+, beschreibt Microsofts .Net Strategie nicht nur eine Programmierschnittstelle, die die Entwicklung von verteilten Software ermöglichen soll, sondern vielmehr eine weitreichende Neuerung in der Windows-Architektur und der Unternehmensphilosophie.

Hinter dem werbeträchtigen Namen .Net verbirgt sich Microsofts Strategie einer vollständigen betriebsystem- und hardwareunabhängigen Lösung zur Entwicklung und dem Betrieb verteilter Softwarearchitekturlösungen. Noch konsequenter als bisher sollen, alle Dienste unabhängig von der eingesetzten Hardware und Software verfügbar sein.

Insbesondere zielt .Net auf die Entwicklung verteilter Webdienste. Microsoft versucht die Interoperabilität zwischen den einzelnen .Net Komponenten durch die Nutzung von offenen Standards wie XML und SOAP zu realisieren.

Hinter .Net steht also nicht nur eine neue Funktion in neuen Windowsversionen, sondern eine grundlegende Änderung der Marketing- und Unternehmensstrategie von Microsoft.

9 XML und Webdienste

Als Basis des Datenaustauschs und der Datenhaltung in der .Net-Strategie sieht Microsoft XML, das sich als Quasi-Standard im Internet schon durchgesetzt hat. Der Vorteil in der Verwendung von XML ergibt sich aus der Trennung der Datenhaltung und der Darstellung. So können Informationen unabhängig von der Plattform ausgetauscht und weiterverarbeitet werden.

Webdienste im Sinne von .Net sind auf XML basierende Websites, die Dienstleistungen unterschiedlicher Art anbieten. Durch die Verwendung von XML bleiben Informationen zwischen verschiedenen Diensten austauschbar und können so universal genutzt werden.

Webdienste können öffentlich und unentgeltlich nutzbar sein, kommerziell angeboten oder selbst entwickelt werden. Ein Beispiel für einen Webdienst ist der Passwort-Bausteindienst, der es einem Benutzer erlaubt mit demselben Benutzernamen und Passwort mehrere Websites zu nutzen.

10 Komponenten

Die .Net-Plattform enthält fünf Komponenten:

- Microsoft Windows-Betriebsystem als Dienstplattform
- Net-Framework
- Net-Bausteindienste
- Net-Orchestration
- Net-Unternehmensserverfamilie

10.1 Dienstplattform

Als Grundlage für die Entwicklung von .Net Anwendungen und Webdiensten sollen die Betriebsysteme Windows 2000, Windows ME, Windows CE und zukünftige Windowsversionen dienen. Diese Betriebsysteme bieten als Basis für .Net Applikationen Betriebsystemdienste an, die eine schnelle und einfache Entwicklung möglich machen sollen. Windows 2000 Server als .Net-Server bietet als Beispiel diese Dienste an:

- Sicherheitsverwaltung
- Ein-/Ausgabe zwischen Speicher, Datenträger und TCP/IP,
- Standardbasiertes XML-Subsystem

Durch den Einsatz des Internet Explorers sollen auch Windows-Einzelplatzversionen .Net-tauglich werden.

10.2 .Net-Framework

Das .Net-Framework fasst alle Komponenten zusammen, die der Entwicklung von .Net-Anwendungen und –Webdiensten dienen. Hier sind die weitreichendsten und interessantesten Neuerungen speziell für Anwendungsentwickler zu erwarten. Der Abschnitt „Interessante Neuerungen – Das .Net Framework" geht intensiver auf die Änderungen ein.

10.3 .Net Bausteindienste

Bausteindienste im .Net Umfeld sind kommerziell angebotene vorgefertigte Webdienste, die als Grundlage für eigenen Entwicklungen dienen sollen. Microsoft bietet Dienste wie Identitätsdienste, Benachrichtigung und Messaging oder Speicherung an. Es ist denkbar, dass andere kommerzielle Anbieter ebenfalls eigene Webdienste anbieten werden.

10.4 .Net-Orchestration

Die Zusammenarbeit von Web-Speicherlösungen und Workflow Lösungen nennt Microsoft Orchestration. Hauptaugenmerk wird auf eine gute Skalierbarkeit gelegt. Geschäftsprozesse sollen mit Kundenanforderungen wachsen können, ohne dass sich Entwickler Gedanken über deren technischen Realisierung machen müssen. So soll es erleichtert werden Workflow-Lösungen anzubieten, unabhängig ob diese auf verteilten Systemen oder über mehrere Standorte verteilt eingesetzt werden sollen.

Microsoft bietet hierfür den BizTalkServer 2000 an, der als Entwurfswerkzeug und Laufzeitumgebung für Geschäftsprozesse und Webdienste dient.

10.5 .Net-Unternehmensserverfamilie

Microsoft bietet zahlreiche Serversysteme für den schellen Aufbau einer Webinfrastruktur an. SQL Server 2000, Internet Security and Acceleration Server 2000, Application Center 2000 und BizTalkServer 2000 sind einige davon.

Der Exchange Server 2000 bietet beispielsweise die Möglichkeit Front-End-Dienste physikalisch getrennt von Back-End-Diensten zu betreiben.

11 Interessante Neuerungen – Das .Net Framework

Völlig neue Wege geht Microsoft mit der Programmierschnittstelle für .Net. Zukünftige Anwendungen werden nicht wie früher auf Basis der Win32-API entwickelt, sondern bauen auf Funktionen auf, die die .Net Framework Basisklassen bieten.

11.1 .Net Programmiersprachen

Die neue .Net Plattform soll insgesamt 27 verschiedene Programmiersprachen unterstützen, die zur Entwicklung der .Net-Anwendungen genutzt werden können. Neben Sprachen wie C++ und Visual Basic .Net (VB 7.0) wird von Microsoft eine neue vollständig objektorientierte Sprache namens C# entwickelt.

Alle Sprachen sollen auf gemeinsame .Net-Bibliotheken zugreifen können. Außerdem übernehmen alle .Net geeigneten Sprachen alle Funktionen von Visual-Studio .Net wie Dokumentation, Debugging und Objektmodelle.

Drittanbieter werden weitere Sprachen wie Smalltalk oder Perl anbieten. Rational plant einen Java Compiler für .Net.

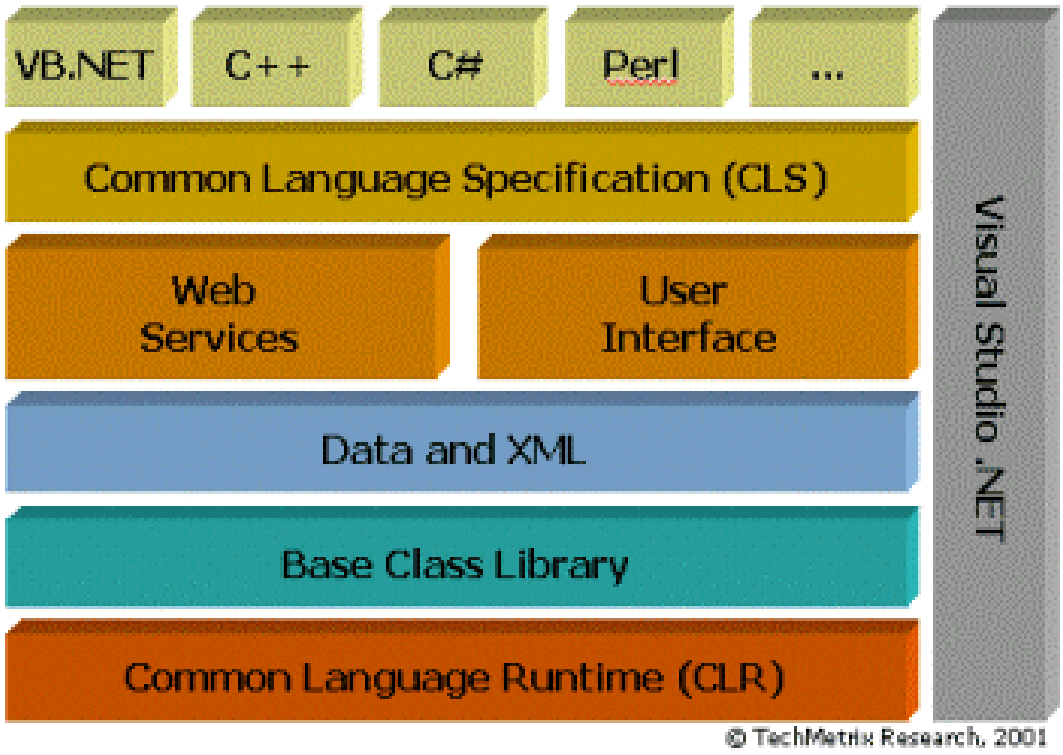


Abbildung 8: .Net Architektur; Quelle: Lauer C**.**

11.2 Metasprache

Alle .Net Programme werden zuerst in eine Metasprache kompiliert, die Microsoft Intermediate Language (**MSIL**). MSIL ist ein binärer Bytecode, der hardware- und betriebsystemunabhängig ist.

Programme, die in MSIL vorliegen, heißen zukünftig Portable Executabels (**PEs**)

11.3 Gemeinsame Regel

Alle Programmiersprachen, die .Net unterstützen, müssen Eigenschaften erfüllen, die in einem Regelwerk namens Common Language Specification (**CLS**) festgelegt sind. Dazu gehört zum Beispiel die Unterstützung der .Net Framework Basisklassen und die Entwicklung eines MSIL-Compilers für diese Sprache.

Die Einhaltung der CLS wird allerdings dazu führen, dass sich der Charakter einzelner Programmiersprachen ändern wird. Ein Java, dass das .Net Framework unterstützt, darf Sun-Bibliotheken nicht mehr benutzen. Somit ist nicht mehr gewährleistet, dass Enterprise Beans und ähnliche Konzepte in einem „.Net-Java" weiterverwendet werden können.

Man könnte daher sagen, dass es nur noch eine einzige Programmiersprache in .Net geben wird: MSIL.

11.4 Common Language Runtime (CLR)

Zur Ausführung der Programme – also zur Laufzeit - wird MISL von einem Just In Time Compiler (**JIT**) in Maschinencode kompiliert. Common Language Runtime wird diese JIT-Umgebung genannt, die ähnlich wie die Java Runtime Environment arbeitet.

CLR überwacht die Ausführung von .Net Programmen. Wie in Java verwaltet CLR die Speicherallokation und den Garbage Collector. Läuft ein .Net Programm unter der Kontrolle von CLR spricht man von „managed Code". Alternativ können Programme im nativen Code, ohne die Verwaltung der CLR ablaufen. Man spricht dann von „unmanaged Code".

Bei dem Einsatz von CLR ist durch den erhöhten Verwaltungs- und Kontrollaufwand mit einem Performance-Verlust von mindestens 10% zu rechnen.

Obwohl prinzipiell eine Betriebsystemunabhängigkeit erreicht werden könnte, spricht Microsoft bei .Net lediglich von der Unterstützung unterschiedlicher Hardware. Ob andere Betriebsysteme .Net-Unterstützung erhalten werden, ist nicht bekannt.

12 .Net und COM, DCOM, COM+

.Net löst die bisherige COM, DCOM, COM+ Architektur ab. Alle Funktionen zur Erstellung von COM- und DCOM-ähnlichen Anwendungen sind in das neue .Net Framework integriert worden, so dass der Entwickler keine andere Programmierschnittstelle ansprechen muss als die .Net Framework Basisklassen.

Die neue Technik zur Erstellung von dynamischen Webseiten, ASP.Net, kann .Net Applikationen kapseln. Beim Aufruf der ASP.Net Seiten werden dann von CLR die .Net Programme ausgeführt.

Ähnlich funktioniert auch der Aufruf von „alten" COM-Objekten. Sie werden in ASP.Net Programmen gekapselt, laufen dann aber ohne CLR-Kontrolle ab.

13 Das .Net Remoting Framework

.Net bietet die Möglichkeit über das Remoting Framework verteilte Anwendungen zu entwickeln, die dann unter der Kontrolle der CLR ablaufen. Von Remoting spricht man immer dann, wenn .Net-Objekte über die Grenzen einer Anwendung mit andern .Net-Objekten in Kontakt treten. Dabei spielt es keine Rolle, ab sich die andere .Net-Anwendung auf dem gleichen oder einem anderen Rechner befinden.

13.1 Verteilter Kontakt

Verteilte .Net Anwendungen lassen sich in 2 Kategorien einteilen: *Client-activated Objects* und *Server-activated Objects*, die die Dienstanbieter- und Dienstebenutzerseite darstellen. .Net Remoting unterstützt zahlreiche Lifetime-Modelle, die dafür sorgen, dass nicht mehr benötigte Remote-Objekte aus dem Speicher entfernt werden.

Der Austausch von Informationen zwischen den Remote-Objekten wird durch *Formater* gewährleistet, die einen Informationsaustausch sowohl über XML und SOAP als auch im Binärformat bei performancekritischen Anwendungen realisieren können.

Die Werteübergabe bei entfernten Funktionsaufrufen erfolgt „by Value" und sollte als *[seralizable]* markiert sein oder das *ISerializable* Interface implementieren.

13.2 Das Proxy Konzept

Wenn ein Client-Objekt versucht eine Remote-Objekt zu aktivieren erstellt das .Net Framework eine Instanz der Klasse *TransparentProxy*, die eine Schnittstellenbeschreibung der aufgerufenen Klasse ist und alle Classes und Interfaces des Remote-Objekts enthält.

Bei der Erstellung des TransparentProxy-Objekts wird dieses bei der CLR registriert. Alle Methodenaufrufe des Client-Objekt an das Proxy-Objekt werden von der Runtime interpretiert und an das Remote-Objekt weitergeleitet.

Befindet sich das Remote-Objekt in derselben *Application Domain* (Anwendungsgrenze) wird der Methodenaufruf des Client-Objekts direkt an das aktuelle Remote-Objekt weitergeleitet. Befindet sich das Remote-Objekt außerhalb der Anwendungsgrenzen, werden die Aufrufparameter in ein *IMessage*-Objekt verpackt, das an eine *RealProxy Klasse* weitergeleitet wird durch Aufruf der *Invoke-Methode*. Diese Klasse ist verantwortlich für eine Weiterleitung der Informationen an das Remote-Objekt.

TransparentProxy und *RealProxy* werden unter der Aufsicht der Runtime bei der Erstellung des Remote-Objekts gebildet.

13.3 Kommunikationskanäle

Channels sind für den Transport von Nachrichten zu und von Remote-Objekten verantwortlich. Ein Client-Objekt kann alle Channels verwenden, die an einem Server-Objekt registriert sind. Folgende Regeln für die Benutzung von Kanälen müssen erfüllt sein:

- Mindestens ein Channel muss am Remoting Framework registriert sein, bevor Objekte an ihm registriert werden können und bevor auf Remote-Objekte zugegriffen werden kann.
- Channels werden für Application Domains registriert; stirbt ein Prozess, werden all seine registrierten Channels zerstört.
- Es können nicht mehrere Channels auf demselben Port registriert werden.
- Das Remoting Framework stellt sicher, dass das Remote-Objekt auf dem richtigen Kanal angesprochen wird; der Client ist verantwortlich, dass die Methode *RegisterChannel* im *ChannelService* aufgerufen wird.

Es gibt verschiedene Arten von Channels, zum Beispiel den HTTP Channel, der Nachrichten mittels des SOAP-Protokolls verschickt, oder den TCP Channel, der einen Binär Formater benutzt um Nachrichten zu versenden.

13.3 Aktivierung

Bevor Remote-Objekte genutzt werden können, müssen sie am *RemotingService* des Remoting Frameworks registriert werden. Normalerweise registriert eine Remote-Anwendung Objekte indem sie zuerst einen oder mehrere Channels registriert, dann die Objekte dem RemotingService bekannt macht und anschließend wartet bis der Prozess beendet ist.

Ein Objekt muss mit folgenden Informationen am RemotingService registriert werden:

- Assemblername in der die Klasse enthalten ist
- Typ und Name des Remote-Objekts
- *URL* unter der ein Client das Remote-Objekt lokalisieren wird
- Objekt Modus, der für die Serveraktivierung benötigt wird (*SingleCall* oder *Singleton*)

Remote-Objekte sind nur solange verfügbar, wie der Prozess läuft, der sie registriert hat. Anschließen werden sie aus den RemotingService entfernt.

14 Fazit

Mit .Net geht eine der weitreichensten Neuerungen in der Microsoft Firmenstrategie einher. War es früher üblich die Benutzer durch proprietäre Lösungen zu einheitlicher Microsoft Infrastruktur zu zwingen, so soll mit .Net ein globaler universeller Netzdienst mit austauschbaren, wiederverwertbaren Modulen entstehen.

Für die Entwicklung verteilter Anwendungen bedeutet .Net eine Vereinfachung durch .Net Framework Basis Klassen, die eine verteilte Programmierung erleichtern sollen.

Der Erfolg der frei zugänglichen Java-Programmierschnittstelle von Sun mag Auslöser für den Paradigmenwechsel Microsofts in .Net gewesen sein. .Net weist starke Parallelen zu der Java Architektur auf. Abzuwarten bleibt, ob auch Microsoft seine .Net-Schnittstelle

kostenlos zur Verfügung stellt und wie konsequent der Gedanke der Plattformunabhänigkeit umgesetzt werden wird. Diese beiden Faktoren werden den Erfolg von .Net beeinflussen.

.Net ist zur Zeit lediglich eine Vision, die durch einige Serverprodukte und Beta-Versionen gestützt wird. Ziel ist ein globaler Internetdienst mit austauschbaren Modulen und grenzenloser Interaktion zwischen den Teildiensten. Inwieweit Microsoft Hardware- und insbesondere Plattformunabhängigkeit umsetzten wird, bleibt noch unklar. Wichtigster Bestandteil ist sicherlich das .Net Framework, das eine plattformunabhängige Entwicklung gewährleisten soll.

Vergleich der Programmierung einer DCOM und RMI-Anwendung

Alex Koch, Juni 2001

15 Einführung

Distributed Computing hat sich als allgemeines Paradigma inzwischen im IT- Bereich etabliert und im realen Praxiseinsatz vielfach bewährt. Die modernen Middleware- Architekturen wie RMI und DCOM ersetzen Client/Server- Computing mit Peer-to-Peer Kommunikation und liefern Vorteile bei der konkreten Anwendungserstellung und Softwareverteilung. In der vorliegenden Arbeit werden diese beiden Middleware-Technologien (RMI und DCOM) gegenüber gestellt. Da RMI sprachabhängig (Java) ist, werde ich beide Vergleichsbeispiele in Java programmieren. Neben den Vorteilen wie Plattformunabhängigkeit und herunterladbaren Programmen in Form von sog. Applets, sind auch Mechanismen für die Kommunikation zwischen im Netzwerk verteilten Anwendungen in Java bereits vorhanden.

Für die Programmierung von verteilten Anwendungen im Netz (Intra-/Internet) bietet Java verschiedene Technologien. Java-Sockets sind innerhalb der Sprache selbst schon vorhanden und ermöglichen eine einfache Kommunikation zwischen Client und Server. Auf Objekt-Ebene schließlich ist mit Hilfe von RMI eine Kommunikation zwischen verteilen Java-Anwendungen möglich. Weiterhin gibt es Java-Implementierungen der CORBA und der DCOM Middelware-Architekturen. Durch Verwendung eines Broker-Mechanismus, wie ihn z.B. CORBA/OMG oder DCOM/Microsoft spezifiziert, ist es möglich, Client und Server in verschiedenen Programmiersprachen auf verschiedenen Plattformen zu realisieren und sie dennoch netzweit transparent miteinander zu verbinden.

Innerhalb der Sprache Java sind im Packet java.net schon verschiedene Klassen für Sockets, URLs, IP-Adressen und HTTP-Verbindungen vorhanden. Dies ermöglicht eine einfache Socket-Kommunikation zwischen Client und Server. Die Programmierung auf Socket-Ebene bietet Vorteile im Performance-Bereich, ist aber recht low-level und vor allem für große Anwendungen nur schwer wartbar. Programmierer wollen sich normalerweise um die Anwendungslogik kümmern und nicht um Netzwerkkommunikation. Da aber bei der direkten Verwendung von Sockets keine Schnittstellenvereinbarungen zwischen den Objekten der Anwendung getroffen werden, sind Anwendungen auf dieser Basis nur schwer wartbar.

RMI schafft dieser Tatsache Abhilfe.

Java ist aber in unserem Fall nicht gleich Java, wie es normalerweise sein soll. Microsofts Java enthält keine RMI Unterstützung. Auf einer reinen Microsoft Plattform mit Microsoft Java wird RMI Anwendung oder Applet nicht funktionieren. Und umgekehrt, mit Suns JDK lässt sich keine DCOM Anwendung schreiben und starten. Um eine DCOM Anwendung in Java zu schreiben muss Microsofts Java her, z.B. MS Visual J++ oder mindestens MS Java SDK.

16 Vergleich der Programmierung

16.1 Programmierung einer RMI- Anwendung

Neben der Möglichkeit mit Sockets eine Verbindung auf relativ niedrigen Abstraktionsniveau aufzubauen, sind mit RMI und Objekt Serialization Möglichkeiten vorhanden, eine Java- zu- Java Kommunikation im Netzwerk auf Objekt-Ebene zu betreiben. Innerhalb von Java ist also bereits eine sog. Middleware- Technologie vorhanden. Für die verteilbaren Objekte spezifiziert der Programmierer die Schnittstellen mit Hilfe von Java. Lediglich eine

zusätzliche Vererbung von java.rmi.Remote zeigt an, dass dieses Objekt transparent über das Netz angesprochen werden kann.

Als Beispiel werde ich eine Eurorechnerapplikation programmieren. Da der Kurs des Euro sich jeden Tag und jede Stunde ändert, ist es sinnvoll die Umrechnung an einer zentralen Stelle durchzuführen, an der auch alle Kurse bekannt sind. Also benötigt man Distributed Computing.

Das RMI Beispiel wurde mit dem Sun JDK1.3 programmiert.

Es ist üblich, dass man bei der Distributed Computing Programmierung mit der Definition der Schnittstelle des Serverobjektes anfängt. Also definiere ich die Schnittstelle für unseren Eurorechner:

```
package eurocalculatorrmi;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Hashtable;

public interface IEuroCalculator extends Remote
{
   //Liefert eine Liste mit bekannten Währungen zurück
   public Hashtable getCurrencyList() throws RemoteException;

   public float calculateCurrencyToEuro( float currencyAmount, String currency )
      throws RemoteException, CurrencyNotFoundException;

   public float calculateEuroToCurrency( float euroAmount, String currency )
      throws RemoteException, CurrencyNotFoundException;
}
```

Das Java-Interface wird von einer Klasse implementiert. Objekte dieser Klasse sind die sogenannte verteilten Objekte.

```
package eurocalculatorrmi;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Hashtable;

public class EuroCalculator extends UnicastRemoteObject implements IEuroCalculator
{
   private Hashtable currencies = new Hashtable();

   public EuroCalculator() throws RemoteException
   {
      super();

      currencies.put( "DEM", new Float( 1.987 ));
      currencies.put( "USD", new Float( 0.987 ));
      currencies.put( "FRF", new Float( 8.7 ));
   }

   public Hashtable getCurrencyList() throws RemoteException
   {
```

```
      return currencies;
   }

   public float calculateCurrencyToEuro( float currencyAmount, String currency )
      throws RemoteException, CurrencyNotFoundException
   {
      if( !currencies.containsKey( currency ) )
         throw new CurrencyNotFoundException();
      return (currencyAmount / ((Float)currencies.get( currency )).floatValue());
   }

   public float calculateEuroToCurrency( float euroAmount, String currency )
      throws RemoteException, CurrencyNotFoundException
   {
      if( !currencies.containsKey( currency ) )
         throw new CurrencyNotFoundException();
      return (euroAmount * ((Float)currencies.get( currency )).floatValue());
   }
}
```

Aus dieser Klasse werden mit Hilfe eines RMI-Compilers Stub und Skeleton erzeugt. Der Client spricht niemals das Objekt dieser Klasse direkt an. Alle Methodenaufrufe auf dem Implementierungsobjekt werden über seine Schnittstelle aufgerufen. Auf Anforderung bekommt der Client von dem NamingService eine Referenz auf das Implementierungsobjekt. Hinter dieser Referenz verbirgt sich der Stub. Stub hat die Aufgabe, bei einem Methodenaufruf (Request), die Parameter in ein sog. Request-Packet einzufügen. Anschließend wird der Request an den Server gesendet, dort von dem Skeleton ausgepackt und an die oben erwähnte Implementierung der Methode übergeben. Nach Abarbeitung der Methode werden entspr. Rückgabewerte an den Client geschickt. Dies geschieht wieder unter Verwendung von Stub und Skeleton. Der große Vorteil ist, dass dieser Vorgang für Programmierer transparent ist. Das Objekt wird so angesprochen, als ob es lokal im Client vorhanden wäre. Lediglich der Erhalt der ersten Objektreferenz (gewissermaßen ein Zeiger über das Netzwerk auf das entfernte Objekt) geschieht über den NamingService. Im NamingService registrieren sich Server-Objekte unter Angabe ihres Namens und der eigenen Objektreferenz.

Die Registrierung des Implementierungsobjektes führt für uns die unten aufgeführte Klasse durch. Sie erzeugt eine Instanz des Implementierungsobjektes und registriert diese im NamingService mit Angabe des Hostnamens des Rechners auf dem das Objekt erzeugt wurde und des Namens unter dem das Objekt angesprochen werden soll.

```
import eurocalculatorrmi.*;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class Server
{
   public static void main(String args[])
         {
      // Create and install a security manager
      if (System.getSecurityManager() == null)
      {
         System.setSecurityManager(new RMISecurityManager());
      }
```

```
      try
      {
         EuroCalculator obj = new EuroCalculator();

         // Bind this object instance to the name "EuroCalculator"
         if( args.length != 0 )
         {
            Naming.rebind("//" + args[0] +"/EuroCalculator", obj);
         }
         else
            Naming.rebind("//localhost/EuroCalculator", obj);

         System.out.println("EuroCalculator bound in registry");
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

Clients können zu einem Namen dann die gesuchte Objektreferenz erhalten. Der Naming-Service ist Bestandteil von RMI, und wird dort als sog. RMI- Registry, ähnlich der Windows-Registry, bezeichnet.

```
import eurocalculatorrmi.*;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.util.Enumeration;
import java.util.Hashtable;

public class Client
{
   public static void main(String args[])
   {
      // Create and install a security manager
      if (System.getSecurityManager() == null)
      {
         System.setSecurityManager(new RMISecurityManager());
      }

      IEuroCalculator serverRef = null;

      try
      {
         if( args.length != 0 )
         {
            serverRef = (IEuroCalculator)Naming.lookup("//" + args[0] +
"/EuroCalculator");
         }
         else
            serverRef =(IEuroCalculator)Naming.lookup("//localhost/EuroCalculator");
```

```
            Hashtable currencies = serverRef.getCurrencyList();
            if( !currencies.isEmpty() )
               System.out.println("Vollgende Waehrungen koennen umgerechnet werden: ");
            Enumeration currencyNames = currencies.keys();
            String currencyName = null;
            while( currencyNames.hasMoreElements() )
            {
               currencyName = (String)currencyNames.nextElement();
              System.out.println( currencyName + " Kurs: " +
              ((Float)currencies.get( currencyName )).floatValue() );
            }

            System.out.println("\n");
            System.out.println("Rechne 54.56 DM in Euro um:");
            System.out.println("Betrag in Euro: " +
          serverRef.calculateCurrencyToEuro( (float)54.56, "DEM"));

            System.out.println("\n");
            System.out.println("Rechne 54.56 Euro in US Dollar um:");
            System.out.println("Betrag in USD : " + serverRef.calculateEuroToCurrency(
(float)54.56, "USD"));
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }
}
```

16.2 Programmierung einer DCOM - Anwendung

DCOM ist seit WinNT 4.0 standardmäßig im Betriebssystem enthalten, für Win95 existiert momentan eine Beta Version von Microsoft. DCOM wurde durch Erweiterungen aus dem Vorgänger COM entwickelt. COM ist seit Windows 3.1 Bestanteil des Betriebssystem und wurde zur komponentenorientierte Anwendungsprogrammierung (OCX) verwendet. Microsoft forcierte in den letzten Jahren seine Anstrengungen im Bereich der Enterprise-Anwendungen und erweiterte deshalb COM um die Möglichkeiten der Verteilung der Komponenten auf verschiedene Knoten in einem Netzwerk. Im Gegensatz zu CORBA handelt es sich bei COM/DCOM nicht um einen hersteller-unabhängigen Standard. Microsoft will aber durch Gründung der OpenGroup DCOM standardisieren lassen. Zusätzliche Services sind nicht spezifiziert sondern werden in Form mehrere neuer Produkte für Messaging und Transaktionsmanagement sofort auf den Markt gebracht (Viper, Falcon). Für die Spezifikation der Schnittstellen verwendet Microsoft seine eigene IDL (Interface Definition Language), die oft auch als MIDL bezeichnet wird. Ein Compiler bildet auf Implementierungssprachen wie C, C++, Java und viele anderen ab. Zusätzlich können Sprachen wie Visual Basic sog. TypeLibs importieren und damit auf DCOM Objekte zugreifen. Von den Plattformen ist momentan Windows NT 4.0 und Win2000 offiziell unterstützt, die SAG portiert im Auftrag von Microsoft DCOM auf UNIX Plattformen. Interessant ist die DCOM Anbindung in Microsofts eigener Java-Entwicklungsumgebung. Hier ist (auf Windows NT) eine sehr enge Integration zwischen Java (J++) und DCOM/COM vorhanden. Zahlreiche Wizzards (z.B. ATL-Wizzard für C++) generieren die notwendigen Implementierungsrahmen; die Aufgabe des Entwicklers ist die notwendigen Methoden zu Implementierung und hier und da auch noch Eintragungen per Makros zu machen. Werden diese vergessen und der

Code geht dennoch durch den Compiler, sind Fehler per Laufzeit nur sehr schwer zu finden. (in Java wie in C++)

Auf Windows NT ist die Unterstützung von DCOM durch Microsoft recht stark. Die Technologie ist eng mit den MS- eigenen Tools wie DevStudio (VC++, VB, ATL) integriert und stellt dort recht gute Abstraktionsebene zur Verfügung. Bei der Entwicklung auf Nicht-Windows- Plattformen ist man auf die Hilfe der Software- AG angewiesen. Das Darmstädter Unternehmen bietet inzwischen Beta Versionen von DCOM für Solaris und Linux an. Digital Unix und HP-UX sowie MVS sollen demnächst folgen. Allerdings beschränkt sich die Unterstützung momentan noch auf das Basis COM-API und ist daher relativ low-level. Auch von den Entwicklungstools und der Codegenerierung ist man ausschließlich auf die Windows-Plattformen angewiesen.

Um eine Java- DCOM Anwendung zu programmieren benötigt man folgende Werkzeuge:

- GUIDGEN.EXE (Bestandteil jedes Windows System)
- Midl Compiler (Bestandteil von Microsoft Visual C++)
- Microsoft Visual J++ oder Microsoft Java SDK

Ich werde kostenfreie Microsoft Java SDK 4.0 (steht bei Microsoft auf der http://www.microsoft.com/java zum Download bereit) einsetzen.

Die Programmierung wird wie bei RMI mit der Definition der Schnittstelle des Serverobjekts angefangen. DCOM Objekte sind Programmiersprachen unabhängig und werden in einer IDL (Interface Definition Language) definiert. Wie schon oben erwähnt, benutzt Microsoft seine eigene Interface Definition Language -MIDL dafür. Die Schnittstelle des Eurorechners in MIDL sieht wie folgt aus:

```
[uuid(B15BB801-EB78-4844-80C9-A78BE3473696),
   helpstring("EuroCalculator"), version(1.0)]
library EuroCalculator
{
       importlib("stdole32.tlb");

       [object, uuid(B15BB802-EB78-4844-80C9-A78BE3473696), dual,
        oleautomation, pointer_default(unique), helpstring("IEuroCalculator Interface")]

      interface IEuroCalculator : IDispatch
      {
          [id(1), helpstring("getCurrencyList Method")]
          HRESULT getCurrencyList( [out,retval] SAFEARRAY( BSTR ) *ar );

          [id(2), helpstring("calculateCurrencyToEuro Method")]
          HRESULT calculateCurrencyToEuro([in] float currencyAmount,
                                          [in] BSTR currency, [out,retval] float * rtn );

          [id(3), helpstring("calculateEuroToCurrency Method")]
          HRESULT calculateEuroToCurrency([in] float euroAmount,
                                          [in] BSTR currency, [out,retval] float * rtn );
      };

      [uuid(B15BB803-EB78-4844-80C9-A78BE3473696),
       helpstring("EuroCalculator Object")]
      coclass EuroCalculator
      {
         [default] interface IEuroCalculator;
```

```
    };
};
```

Die GUID's(globally unique identifier), die in der Form **B15BB801-EB78-4844-80C9-A78BE3473696** auftreten, wurden mit Hilfe GUIDGEN.EXE Tools generiert und sind weltweit eindeutig. Diese von der OSF (Open Software Foundation) definierten Identifikatoren haben eine Länge von 128 Bit. Bei ihrer Generierung findet neben der Ethernet-Adresse der installierten Netzwerkkarte zusätzlich Zeitinformation Berücksichtigung. Folglich kann es zwischen unterschiedlichen Quellen von COM-Objekten zu keinen Kollisionen kommen.

Diese Schnittstellendefinition wird in der Datei mit dem Namen **EuroCalculator.idl** abgespeichert, und mit **midl EuroCalculator.idl** kompiliert. Midl Compiler erzeugt eine Type Library Datei mit Namen **EuroCalculator.tlb**. Sinnvoller Platz für diese Datei wäre z.B. das Verzeichnis %WINDOWS%\System32.

Danach muss die neu Erzeugte Type Library in der Windows Registry registriert werden und die notwendigen Java Interfaces müssen generiert werden. Aus einer Type Library kann man für fast alle gängigen Sprachen mit Hilfe eines entsprechenden Compilers Interfaces generieren.

Mit Hilfe von Visual J++ soll es möglich sein, den umgekehrten weg zu gehen. Das heißt, man kann ein Java Interface schreiben und dann daraus eine Type Library erzeugen. Dieses Werkzeug hatte ich nicht zur Hand und kann dies deswegen nicht bestätigen.

Das Registrieren der Type Library und Erzeugen von Java Schnittstellen führt man mit dem Tool **JactiveX,** das in dem Java SDK und Visual J++ enthalten ist. In den früheren Versionen von Visual J++ bzw. Java SDK war dafür das Tool javatlb zuständig.

Folgendes Kommando führt die Aktion durch:
JactiveX /javatlb /r c:\WINDOWS\system32\EuroCalculator.tlb

Die Optionen */javatlb* und */r* sind dabei wichtig und haben folgende Bedeutung:

- /javatlb – javatlb Modus
- /r – Type Library registrieren

Die Ergebnisse werden in drei Dateien *IEuroCalculator.java, EuroCalculator.java* und *IEuroCalculatorDefault.java* im Verzeichnis %WINDOWS%\Java\TrustLib abgespeichert. Das Interface und Type Library werden auch in der Registry unter HKCR\Interface und HKCR\TypeLib eingetragen. Diese Einträge kann man mit Regedit.exe überprüfen.

Für uns ist nur Interface von Interesse. In Java sieht es wie folgt aus:

```
//
// Auto-Erstellung verwendet JActiveX.EXE 5.00.3601
//   (JactiveX /javatlb /r c:\WINNT\System32\EuroCalculator.tlb)
//
// WARNUNG: Entfernen Sie keine Kommentare, die "@com"-Anweisungen enthalten.
// Diese Quelldatei muss kompiliert werden von einem Compiler, der @com verarbeiten
kann.
// Falls Sie den Microsoft Visual J++-Compiler verwenden, m\u00fcssen Sie
// Version 1.02.3920 oder h\u00f6her verwenden. \u00c4ltere Versionen werden zwar
keinen Fehler melden,
// aber Sie werden keine COM-aktivierten Klassendateien erzeugen.
//

package eurocalculator;
```

```
import com.ms.com.*;
import com.ms.com.IUnknown;
import com.ms.com.Variant;

// Dual interface IEuroCalculator
/** @com.interface(iid=B15BB802-EB78-4844-80C9-A78BE3473696, thread=AUTO,
type=DUAL) */
public interface IEuroCalculator extends com.ms.com.IUnknown
{
  /** @com.method(vtoffset=4, dispid=1, type=METHOD, name="getCurrencyList")
      @com.parameters([vt=8,type=SAFEARRAY] return) */
  public com.ms.com.SafeArray getCurrencyList();

  /** @com.method(vtoffset=5, dispid=2, type=METHOD,
name="calculateCurrencyToEuro")
      @com.parameters([in,type=R4] currencyAmount, [in,type=STRING] currency,
[type=R4] return) */
  public float calculateCurrencyToEuro(float currencyAmount, java.lang.String cur-
rency);

  /** @com.method(vtoffset=6, dispid=3, type=METHOD,
name="calculateEuroToCurrency")
      @com.parameters([in,type=R4] euroAmount, [in,type=STRING] currency, [type=R4]
return) */
  public float calculateEuroToCurrency(float euroAmount, java.lang.String currency);


  public static final com.ms.com._Guid iid = new com.ms.com._Guid((int)0xb15bb802,
(short)0xeb78, (short)0x4844, (byte)0x80, (byte)0xc9, (byte)0xa7, (byte)0x8b,
(byte)0xe3, (byte)0x47, (byte)0x36, (byte)0x96);
}
```

Wie man den Kommentaren entnehmen kann, dürfen diese Java Quellcodedateien nur mit einem geeigneten Compiler kompiliert werden, der auch die COM spezifische Kommentare versteht und übersetzt. Dafür eignet sich eigentlich nur Microsoft Visual J++ Compiler ab Version 1.02.3920. Außerdem muss dem Compiler noch eine zusätzliche Option mitgegeben werden (/x-) damit er auch die COM- spezifische Kommentare übersetzt.
Alle drei Klassen müssen aber übersetz werden, z.B. mit:

jvc /x- C:\WINDOWS\Java\TrustLib\eurocalculator.java*

Jetzt kann das Serverobjekt implementiert werden.

```
import com.ms.com.*;
import eurocalculator.*;
import java.util.Hashtable;
import java.util.Enumeration;

public class EuroCalculator implements IEuroCalculator
{
   private Hashtable currencies = new Hashtable();
```

```
   public EuroCalculator() throws ComException
   {
      super();

      currencies.put( "DEM", new Float( 1.987 ));
      currencies.put( "USD", new Float( 0.878 ));
      currencies.put( "FRF", new Float( 8.7 ));
   }

   public com.ms.com.SafeArray getCurrencyList() throws ComException
   {
      Enumeration keys = currencies.keys();

      SafeArray sa = new SafeArray(Variant.VariantString, currencies.size());
      String stra[] = new String[currencies.size()];
      int index = 0;
      while( keys.hasMoreElements() )
      {
         stra[index] = (String)keys.nextElement();
         index++;
      }
      sa.fromStringArray(stra);
      return sa;
   }

   public float calculateCurrencyToEuro( float currencyAmount, String currency )
      throws ComException
   {
      if( currencies.containsKey( currency ) )
         return (currencyAmount / ((Float)currencies.get( currency )).floatValue());
      return 0;
   }

   public float calculateEuroToCurrency( float euroAmount, String currency )
      throws ComException
   {
      if( currencies.containsKey( currency ) )
         return (euroAmount * ((Float)currencies.get( currency )).floatValue());
      return 0;
   }

}
```

Die Objektklasse muss mit jvc kompiliert und in das Verzeichnis %WINDOWS%\Java\Lib kopiert werden.

Danach muss diese Klasse als CoClasse des Interfaces bekannt gemacht werden und zwar mit folgendem Kommando:

Javareg /register /class:EuroCalculator.class /clsid:{B15BB803-EB78-4844-80C9-A78BE3473696} /surrogate

Die CLSID muss die gleiche wie bei der CoClasse in der EuroCalculator.idl sein.

Man kann diesen Eintrag mit regedit unter HKCR\CLSID überprüfen.

Um dem Client den Zugriff auf den Server zu erlauben, müssen einige DCOM Einstellungen im Betriebssystem auf der Servermaschine angepasst werden. Dafür startet man ein Tool namens DCOMCNFG (Man braucht Administratorrechte).

- Im Bereich Standardsicherheit \ Standardzugriffsberechtigungen geht man auf Standard bearbeiten und fugt das SYSTEM Account und den User, der DCOM Anwendung von der Clientmaschine nutzen möchte, hinzu.
- Im Bereich Standardsicherheit \ Standardstartberechtigungen geht man auf Standard bearbeiten und fugt das SYSTEM und INTERACTIVE Accounts und den User, der DCOM Anwendung von der Clientmaschine nutzen möchte, hinzu.

Jetzt ist unser Serverobjekt fertig und steht zum zugriff aus dem Netz bereit.

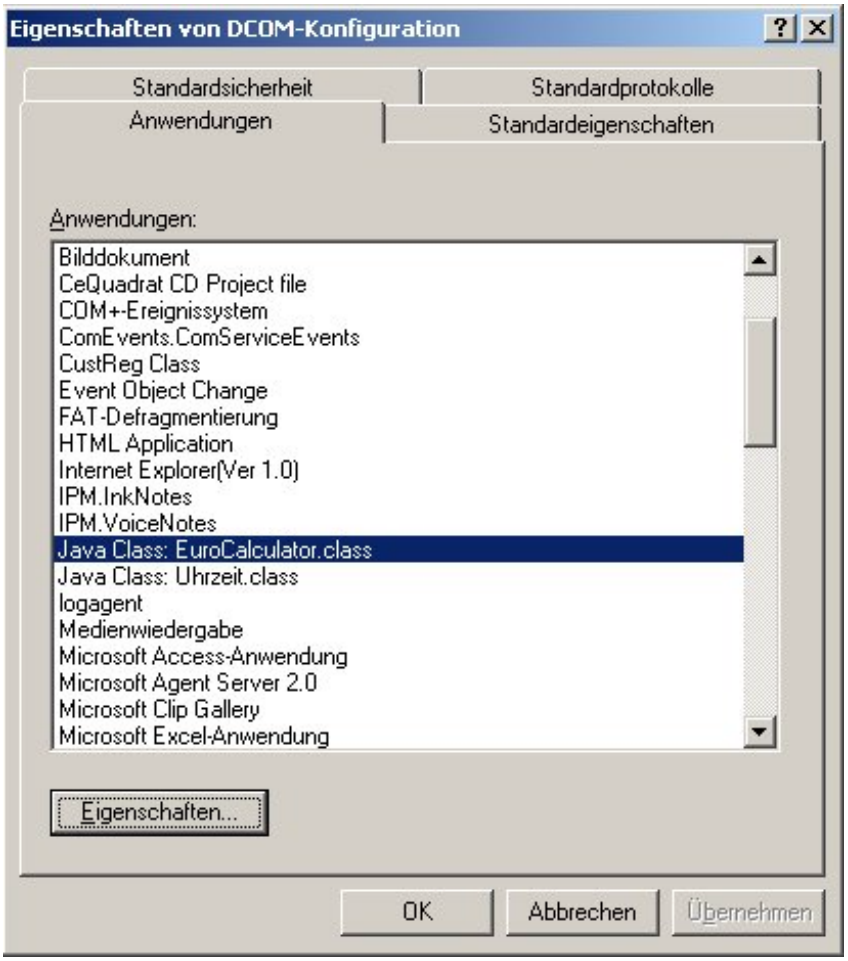
Betrachten wir jetzt die Clientmaschine und alles was da noch eingerichtet werden muss.

Als erstes wird die EuroCalculator.tlb auf die Clientmaschine kopiert und wie auf dem Server registriert. (*JactiveX /javatlb /r c:\WINDOWS\system32\EuroCalculator.tlb)*

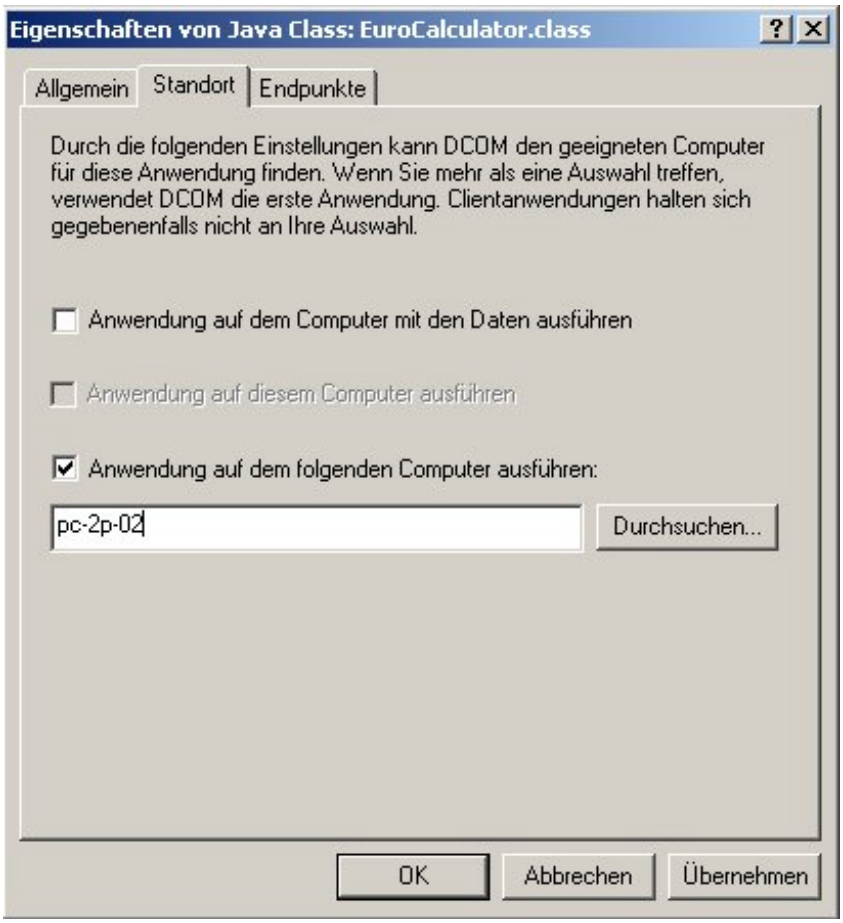
Danach wird die EuroCalculator.class Klasse registriert mit:

Javareg /register /class:EuroCalculator.class /clsid:{B15BB803-EB78-4844-80C9-A78BE3473696}

Danach wird die Location von dem Server bestimmt. Das macht man mit dem schon bekannten Tool DCOMCNFG. Man selektiert die JavaClass: EuroCalculator.class und betätigt den „Eigenschaften..“ Button.



Im Bereich „Standort \ Anwendung auf dem folgenden Computer ausführen" gibt man den Hostnamen der Servermaschine. Die Anderen beide Checkboxen müssen leer bleiben.



Als nächstes checkt man ob alle Registry einträge auch wirklich korrekt sind. Dafür startet man regedit und sucht nach EuroCalculator. Als erstes sollte in HKCR \ AppID ein Eintrag gefunden werden. Als Server Hostname sollte da der Hostname eingetragen sein, den man gerade eingegeben hat. Wenn man die suche fortsetzt, wird noch ein Eintrag in HKCR \CLSID gefunden. Man muss an dieser Stelle die InProcServer32 und LocalServer32 Einträge löschen wenn die existieren. Wenn man die Suche noch mal fortsetzt sollte noch der Eintrag für das Interface in HKCR \ Interfaces gefunden werden.

Als nächstes erzeugen wir einen Client der unseren EuroCalculator nutzen wird. Die Implementierung ist sehr einfach.

```
import eurocalculator.*;

public class EuroCalcClient
{

   public static void main(String[] args)
   {
      try
      {
         IEuroCalculator calc = (IEuroCalculator)new EuroCalculator();
```

```
            String[] currencies = calc.getCurrencyList().toStringArray() ;
            if( currencies.length > 0 )
               System.out.println("Vollgende Waehrungen koennen umgerechnet werden: ");
            for (int i = 0; i < currencies.length; i++)
            {
               System.out.println( currencies[i] );
            }

            System.out.println("\n");
            System.out.println("Rechne 54.56 DM in Euro um:");
            System.out.println("Betrag in Euro: " + calc.calculateCurrencyToEuro( (float)54.56, "DEM"));

            System.out.println("\n");
            System.out.println("Rechne 54.56 Euro in US Dollar um:");
            System.out.println("Betrag in USD : " + calc.calculateEuroToCurrency( (float)54.56, "USD"));
         }
         catch(Exception e)
         {
            e.printStackTrace();
         }
      }
}
```

Man sieht, dass Serverobjekt so angesprochen wird, als ob er Lokal ablaufen würde.

Nach der Kompilierung mit jvc soll der Client in Microsoft VM (jview) laufen und auf den Serverobjekt zugreifen können.

17 Fazit

Jedes von diesen zwei Paradigmen hat Vor- und Nachteile.

RMI-Paradigma

Vorteile:

- RMI ist sehr einfach und braucht keine zusätzliche Interface Definition Language
- RMI ist plattformunabhängig

Nachteile:

- RMI ist sprachenabhängig

DCOM-Paradigma

Vorteile:

- Mit geeigneten Werkzeugen einfach zu programmieren.
- DCOM ist sprachenunabhängig.

Nachteile:

- Nur sehr eingeschränkte Platformunabhängigkeit

- Administration ist sehr komplex.
- Administrator-Rechte notwendig.
- Im Bezug auf Java Programmierung, sehr spezifisch und Herstellerabhängig

18 Literaturverzeichnis

Hawkins J., Obermeyer P.: Microsoft .Net Remoting: A Technical Overview, http://msdn.microsoft.com/library/techart/hawkremoting.html, Redmond 2000

Hermann M.: Entwicklung von .Net Zusammenarbeitslösungen, http://www.microsoft.com/net, Redmond 2000

Imgrund B.: DCOM – Komponententechnologie, http://www.informatik.uni-bremen.de/grp/flowtec/papers/doc/dcom/html, Bremen 1999

Lauer C.: Introducing Microsoft .Net, http://www.dotnet101.com/articles/art014_dotnet.asp, Juni 2001

Stahl M.: Microsoft DCOM, http://www.stal.de/Downloads/DCOM/DCOM.html, Mai 2001

Willers M.: Weite Reise, c´t 06/2001, Seite 252-259, Hannover 2001

N.N.: The .Net Framework ans COM, http://www.microsoft.com/net/developer/frameworl.com.asp, Juni 2001

N.N.: Microsoft Corporation, DCOM Technical Overview, http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomtec.htm#dcomtec_arch

Horstmann M.; Kirtland M.: DCOM Architecture, http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm, Juli 1997

Buyya R.: High Performance Cluster Computing: Programming and Applications, Vol. – 2, Prentice Hall 1999

Merkle B.: "Von Client/Server-Anwendungen zu Java wechseln", Interactive Objects Software GmbH

N.N.:Microsoft DCOM http://www.microsoft.com/oledev