Dokument Management Objekt
och
N-skikt programmering

Examensarbete i Datavetenskaplig linje
Vid Umeå universitet
Stockholm februari 2004

Examinator vid Umeå universitet: Per Lindström
Handledare på Linq företaget: Christer Gustafsson

Författare: Shahram Shahinzadeh

Sammanfattning

Detta är en rapport som beskriver ett examensarbete på 20 poäng. Arbetet har utformats av Företaget Linq och Umeå universitet.
Produktutvecklingsföretaget Linq har utvecklat en företagsportal och vill komplettera sin produkt med ett dokumenthanteringsobjekt (Document Management Object). Företaget kräver en applikation för att kunna hantera dokument av olika typer. Avhandlingen består av två delar:

- Att studera möjligheter för att implementera ett program med avseende på de produkter som är anpassande för detta mål.
- Andra delen av uppgiften är att implementera en applikation och det sak hantera olika dokument av olika typer på bästa sätt.

Denna rapport ger en beskrivning av de tekniker och teknologier som har studerats för att utveckla ett dokument hanteringsobjekt. I början av denna avhandling undersöks Windows DNA arkitektur, komponent teknologi, Microsoft komponent teknologi COM (Component Object Model), DCOM (Distribuerade COM), MTS (Microsoft Transaction Server) och SQL språk. I andra delen av rapporten beskrivs lösningen på uppgiften och fördelar med den.

Abstract

This is a report that describes a thesis (20 credits). Linq Company and Umeå University form the thesis. The developer's Company Linq has developed a portal and wants to complete the product with a Document Management Object. The Company requires an application to handle document of different kinds. This application has two parts:

- To study possibilities for implementing a program that fits to old products of the company.
- The other part of the work is to implement an application that has to handle different documents of different kinds in the best way.

This report gives a description of those different techniques and technologies that have been studied of the writer to perform the Document Management Object. In the start of this thesis Win DNA architecture, Component Technology, Microsoft Component Technologies COM, DCOM, MTS and the SQL language are investigated.
In part two of this report the solution of the problem is described and the benefit of using the proposal approach to solve the problem is discussed.

1 Inledning

1.1 Bakgrund

Till följd av ökad användning av datorer och ökad näthastighet har behovet av att kunna hanterar olika typer av dokument synts mycket tydligare. Nästan alla företag strävar efter att datorisera sina arbeten i stora mängder så att de kan utnyttja denna teknik på bästa möjliga sätt. Internet har blivit ett viktigt medium och varje företag har skapat en webbadress på nätet. Detta grundar sig på att företaget vill både minska kostnaden och försnabba arbetsprocedur. Genom en uppkoppling till Internet och en webbläsare som presentationsverktyg kan många användare ta del av den information som erbjuds var de än befinner sig.
På samma sätt har kravet för hantering av dokument i stora skalor ökats markant. Det krävs att programmerare både konstruera ett användarvänligt och avancerat system. Dessutom dyker det upp en del nya problem t ex säkerhet, uppdatering, standardisering osv.

1.2 Mål och syfte

Uppdragsgivare och problemformulerare är produktutvecklingsföretaget Linq. Linqs vision är att öka kommunikationen och integrera företagets alla system, med andra ord centralisera eller samla alla tjänster i en plats. En företagsportal har utvecklats för att uppnå detta. I denna portal kan företaget lägga in sina tjänster så att varje datoranvändare inom företaget kan nå dem. En sådan tjänst kan t.ex. vara ett DMO (Document Mangement Object) som hanterar dokumentet inom en grupp av användare.
Syftet med detta arbete är att studera de olika teknikerna för hantering av olika dokument och till slut implementera en applikation för detta ändamål, d.v.s. ett DMO. DMO:et ska köras både på intranät och Internet. Denna applikation ska ta hand om olika uppgifter bland annat versionshantering, rättigheter och identifikation.

1.3 Metod

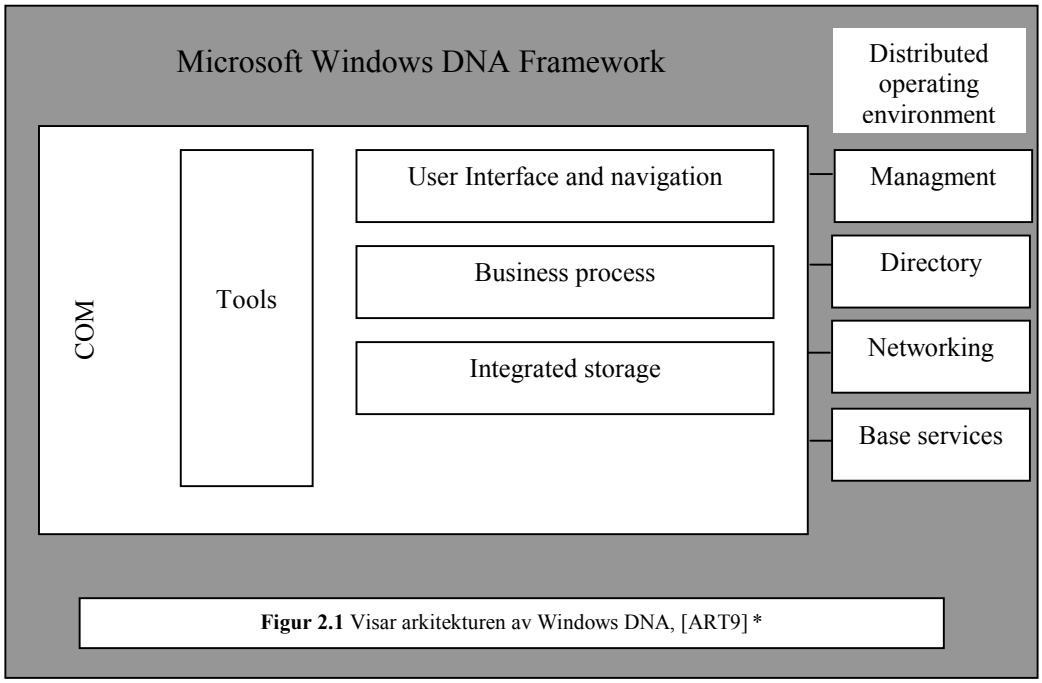
Arbetet är uppdelat i flera faser, studie av komponent teknologi, jämföra komponent teknologi med traditionell teknologi, design och utveckla DMO. Analysfasen gav en mycket grov bild av problemet eftersom kunskapen inte var tillräcklig i början och därför krävdes en omgång av denna fas efter en del studier.
I studiefasen studerades aktuella och även framtida teknologier för att konstruera en bred och komplett applikation.
I implementeringsfasen koncentrerats så mycket som möjligt kring plattformsoberoende, säkerhet och återanvändbarhet.
Den största delen av arbete har ägnats åt att studera olika teknologier och tekniker bland annat MSCOM (Component Object Model), DCOM (Distibuted MSCOM), MTS (Microsoft Trasaction Server), N-tier, .NET osv.

Utvecklings miljö för detta arbete är NT4/Windows2000 Beta både Workstation och server. MSVB (Microsoft Visual Basic), MSVBA (Microsoft Visual Basic Application), MTS (Microsoft Transaction Server), MSADO (Microsoft Active Data Object) och MSSQL (Microsoft Structural Query Language) är en del av de verktyg och tekniker som också har använts i detta arbete.
I nästa del av rapporten kommer vissa av nämnda teknologierna att beskrivas i omfattande grad.

2 Teknologier och tekniker

2.1 Windows Distribuerade Internet applikationer (DNA)

Microsoft har kallat sin version av distribuerade applikationer för Windows Distributed interNet Applications (DNA) och avser distribuerade applikationer i främst Windows miljö. DNA bygger på att applikationer utvecklas som mindre komponenter som sedan sammanfogas för att skapa en helhet. Man kan byta en komponent mot en annan utan att påverka hela applikationen. Den logiska strukturen för Windows DNA, figur 2.1[*], bygger på tre lager eller skikt:



Figur 2.1 Visar arkitekturen av Windows DNA, [ART9] *

- Presentationslagret: detta lager används som ett gränssnitt mot användaren. Presentationslagret implementeras oftast som ett vanligt exekverbart program eller med hjälp av en webbserver och ett skriptspråk
- Affärslogiklagret: detta lager utför den mesta delen av exekveringen. Här kontrolleras och efterföljs organisationens affärs regler. Genom att centralisera affärslogiklagret underlättar man uppdateringar av applikationen. Lagret implementeras som komponenter som kan exekvera

[*] Bilden avbildad från referens [ART9].

- antingen på en server (Web baserad applikationer) eller på klienten (Klient/Server applikationer).
- Dataåtkomstlagret: detta lager har till uppgift att hantera kommunikationen med datakällor och förse lagret affärslogik med data. Datakällor behöver inte vara traditionella databaser utan kan vara filer, kataloger, e-post osv. Lagret implementeras som komponenter som oftast exekverar på servern med datakällan.

Dessa tre lager samarbetar för att skapa en helhetsapplikation. Beroende på applikationens storlek eller komplexitet så varierar storleken på lagren eller gränserna mellan lagren. Detta har gjorts med hjälp av olika teknologier som tillsammans skapar en miljö som förenklar att utveckla distribuerade applikationer. Till exempel i Windows NT 4 används Component Object Model (MSCOM) och ibland Microsoft Transaction Server (MTS) för att skapa en miljö för komponenter. När det gäller distribuerade komponenter används Distributed MSCOM d.v.s. DCOM som är ett kommunikationsprotokoll mellan komponenter på olika processer eller datorer.

2.2 Lager och Arkitektur

I den traditionell modell (1-lager), applikationen är inte delad i olika delar och allt kod är i en och samma modul. Begreppet lager arkitektur har utvecklas från 1-lager till N-lager. Oavsett i vilken arkitektur en applikation är konstruerad, kommer applikationen att hantera presentation, affärslogik och dataåtkomst lagerna i sin funktionalitet.
I en *1-lager* utförs hela logiken på ett ställe och delar av applikationen är beroende av varandra. Om något i en applikation ändras kommer det att påverkas hela applikationen.
Med utveckling av Internet har behovet att ha olika lager i en applikation ökats och därför används klient/server modellen (C/S) som beskriver en *2-lager* arkitektur. Applikationen delas i presentationlager som utförs i klientdelen, data och affärsregler i serverdelen.
I *3-lagers* arkitektur finns tre olika lager för att utföra olika moment i arkitekturen. Denna arkitektur innehåller presentationlager, affärslager och dataåtkomst lager.
Komponenter i alla tre lager kan placeras på en och samma dator eller delas upp på tre typer av datorer klientdator, applikationsserver och databasserver. Vanligtvis läggs databaskomponent i MTS Server och presentationkomponent i klient.
N-tier är en utveckling av 3-tier som erbjuder obegränsad lager enligt det behov som krävs i en applikation. Med andra ord delas de tre huvud delar i mindre delar (lager) för att öka flexibilitet och skalbarhet i en applikation.

2.3 Komponentteknologier

2.3.1 Bakgrund

Eftersom exekverande filer (EXE-filer) är stora och tar mycket plats både på hårddisken och på arbetsminne (RAM) så används av Dynamic Link Library filer (DLL-filer). Sådana filer ligger till grund för komponentteknologin i Microsoft miljö. DLL-filer innehåller kod som används av flera program samtidigt, på så sätt sparar man plats på minnet jämfört med vanliga EXE-filer. Koden i DLL-filer laddas vid behov. Koden kommer att anropas via ett s.k. Application Programming Interface (API) och sökvägen i filsystemet hård kodas oftast i EXE-filen. Nackdelen med sådana DLL-filer är att om dem ändras och kompileras om, så kommer den binära representationen av DLL-filen också att ändras vilket leder till att även applikationen som använder DLL-filen måste kompileras om. Detta resulterar i att uppdateringar blir tidskrävande. Komponentteknologin löser detta problem genom att använda gränssnitt (Interface), eftersom gränssnitt är fristående från implementationen, d.v.s. om komponentens implementation ändras internt så kommer inte gränssnittet att påverkas. Med andra ord kommer applikationerna som använder sig av komponenten inte påverkas av detta. Men däremot en ändring av interfacet kommer att orsaka ändringar i applikationen och en kompilering av applikationen krävs därför.

2.3.2 Komponent

Begreppet komponent har många definitioner och betydelser men kortfattat kan man säga att en komponent registrerar sig i ett system och talar om hur den kan anropas. En komponent avser mjukvara som ersätts med en annan mjukvara med samma gränssnitt. Ett av huvudmålen med komponenter är återanvändning. Gränssnitt eller interface är den viktigaste delen i en komponent och grunden för allt komponenttänkande. Målet är att frigöra hur saker sköts från vad som kan göras. Gränssnittet är alltså oberoende. I fortsättning av rapport kommer jag att presentera Microsoft COM teknologi.

2.4 Komponent Objekt Modell (MSCOM)

2.4.1 Bakgrund

MSCOM är en utveckling av Microsoft gamla teknologi OLE (Object Linking and Embedding) och ActiveX.
Termen OLE används nuförtiden för att beskriva den teknologi som används för att skapa sammansatta objekt. ActiveX återanvänds som i sitt ursprungliga ändamål d.v.s. visuella komponenter och komponenter som används i Internet. COM publicerades av Microsoft i 1993. Component Object Model är både en standard och en implementation. D.v.s. COM specificerar hur kommunikationen med komponenter sker och implementationen av COM ger tjänster som att hitta och skapa instanser av komponenter.

COM är en språkoberoende teknologi, komponenter kan skrivas med vilket språk som helst som är COM-kompatibelt, komponenter kan sedan anropas av ett annat språk. Till exempel en komponent kan skrivas i C++ för att skapa ett effektiv binärkod och anropas av Visual Basic eller Delphi som är bättre för grafiska gränssnitt mot användaren.
COM är också en platsoberoende teknologi, på vilken dator komponenten exekverar ska inte spela någon roll. Programmet som använder sig av komponenten kan vara en arbetsstation och komponenten kan vara placerad på samma dator eller en server. Men anropet av komponentens metod är detsamma ur programmerarens synvinkel.
Som sagt COM är en specifikation och jag kommer att gå igenom MSCOM, Microsoft implemenation av COM och hur de jobbar.
Språkoberoendet och platsoberoende löses genom att separera gränssnittet från implementation av komponenten och använda Virtual Function Table(VTLB). Platsoberoende skapas av DCOM (Distributed COM). Distribuerad COM (DCOM) är en förlängning av COM som gör att utvecklare av en distribuerad applikation inte behöver bry sig om var komponent befinner sig. Anropet av komponenter sker på samma sätt, oavsett om komponenterna finns lokalt eller på en annan dator. DCOM protokoll tar hand om datatransport mellan komponenter och datorer. Programmeraren behöver inte skriva någon kod för att göra en komponent distribuerad.

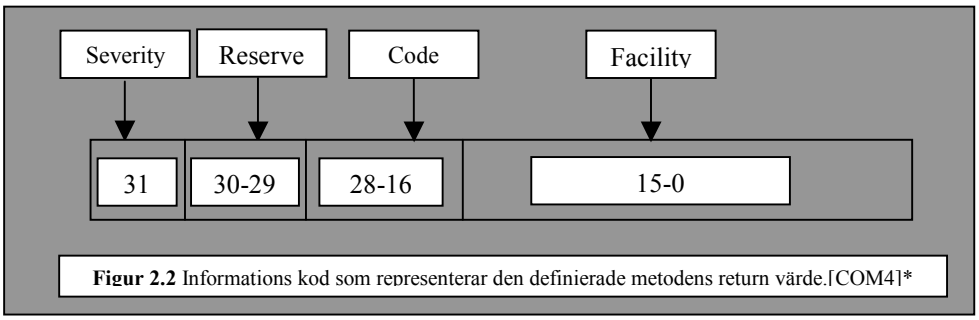
2.4.2 Gränssnitt

COM är baserad på objekt men objekten i COM skiljer sig från objekt i vanliga programmeringsspråk. För att undvika ändringar i en applikation använder COM gränssnitt (interface) för att hantera olika objekt.
All kommunikation med komponenter sker genom komponenters gränssnitt (interface). Interface definieras som en samling av semantiskt relaterade metoder, d.v.s. en definition av metoderna och dess parametrar.
Gränssnitt representerar ett kontrakt mellan anropade program och komponenter. Detta leder till att publicerade komponenters gränssnitt inte bör ändras, d.v.s. antalet metoder och metoders identitet eller signatur. Men detta förhindrar inte att implementationen av koden inte kan ändras, t.ex. när det gäller underhållning av programmet. Om en DLL-file i traditionell programmering skulle korrigeras och kompileras om så behöver även klientprogram som använder sig av DLL-filen kompileras om. När det gäller komponenter så behöver man inte kompilera om klientprogram om interfacet inte ändras. Men det händer ibland om komponenten ska utökas med ny funktionalitet och därmed fler metoder. I sådana situationer kan man behålla det gamla interfacet och ge komponenten ett nytt interface för den nya funktionaliteten, med andra ord skapa en ny version av gränssnittet.
Liksom klasser i objektorienterade språk så kan gränssnitt ärva från andra gränssnitt.
Alla interface måste ärva från IUnknown. IUnknown är ett fördefinierat interface och huvudinterface i COM. Alla interface måste ärva från IUnknown. Komponenter enligt COM kan ha flera gränssnitt. COM definierar också en del standardinterface inklusive deras associerade IID (Interface Identitet) för att underlätta kommunikation mellan komponenter.

Gränssnittet IUnknown ger komponenter några grundläggande egenskaper och består av tre metoder:
1: QueryInterface () används för att fråga efter en pekare till ett visst gränssnitt och som parameter till metoden sänds ett IID och returnerar en pekare till objektet. Med andra ord söks efter en implementaion av ett interface.
2: Addref () ökar en räknare som håller reda på antal referenser till objektet. Räknaren uppdateras när ett objekt skapas eller en referens/pekare sätts till att peka på ett objekt.
3: Release () minskar en räknare som håller reda på antal referenser till objektet och när räknaren med referenser blir noll så kan komponenten förstöra sig.
Ett annat exempel av interface är IDispatch. Detta interface används för att hantera komponenter som ska användas i skriptspråk.

2.4.3 Returtyp i COM

Värdet som returneras från ett funktionsanrop på ett interface är HRESULT. I COM måste man specificerar ett returvärde för varje metod. HRESULT är en 32-bit nummer som ger information om returtypen. Figur 2.2* visar olika delar av HRESULT.



Figur 2.2 Informations kod som representerar den definierade metodens return värde.[COM4]*

Här nedan visas de konstanter som beskriver de vanliga svaren från HRESULT.
S_OK
S_FALSE
E_FAIL
E_NOTIMPL menas att medlem av metoden inte är implementerad
E_OUTOFMEMORY menas att ett fel uppstått eftersom en resurs inte kan dynamiskt skapas.
E_INVALIDARG Fel parameter har skickats
E_UNEXPECTED
E_POINTER menas att pekaren pekar på NULL

2.4.4 Identifierare i COM

På grund av att det inte räcker med strängar för att unikt identifiera komponenter och gränssnitt använder Microsoft ett 128-bitars nummer kallat

* Bilden avbildad från referens [COM4].

Globally Unique Identifier (GUID). Dessa GUID genereras ofta automatisk av programmeringsmiljön. Eftersom det är svårt att minnas ett svårt tal, används ett läsbart namn (ProgramID, ProgID).

2.4.5 IDL språket

MSCOM definierar IDL för att hantera olika delar i en komponent, bland annat attribut av interface, metoder och metoders parameter. IDL var ursprungligen ett DCE RPC (Distributed Computing Environment Remote Procedure) språk och liknar mycket C++. DCE IDL är ett oberoende språk så att olika utvecklare kan kompilera koden med hjälp av en kompilator (MIDL). Figur 2.3 visar ett exempel skriven i IDL.

```
import  “unknwn.idl”;
//interface IX
[
        object,
        uuid(00000000-0000-0000-00000-00000000)
        helpstring(“IX interface”)
]
interface IFoo: Iunknown
{
        HRESULT  MethodX([in, string] wchar_t* szIn);
        HRESULT  MethodY([out, string] wchar_t** szOut);
}
```

Figur 2.3 Ett exempel av interface i COM.

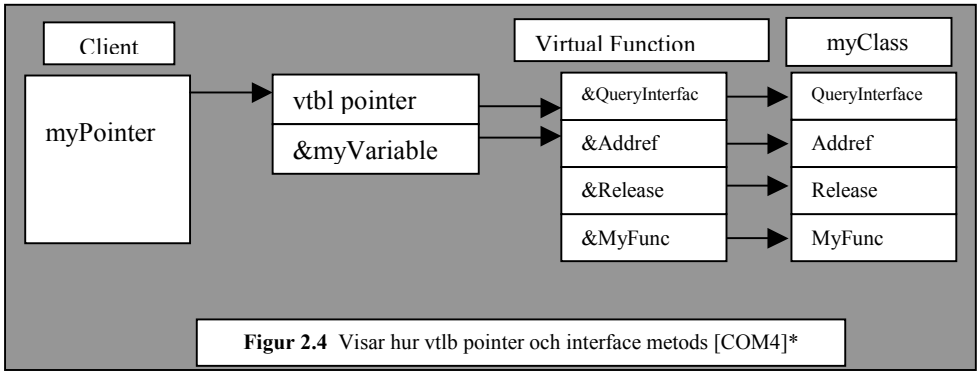
IDL-filen måste inkludera Unknown.idl som innehåller IDL-definitionen för gränssnittet IUnknown (QueryInterface(), AddRef(), Release()). Microsofts kompilator ger automatisk C/C++ filer för COM interface. Som framgår av figur (2.4), ett IDL-fil följande delar:
- object, det är ett nyckelord som definierar ett COM interface.
- uuid, det specificerar IDD för interfacet.
- helpstring, används för att sätta en hjälpsträng i ett typ bibliotek

IDL-filen importerar en del systemfiler som beskriver definierade interface. Win32 SDK (Software Developer Kit) inkluderar en kompilator som kallas, MIDL.EXE som kompilerar IDL-filen och genererar 5 olika filer för olika ändamål.

1: Foo.H innehåller C/C++ interface definitioner och associerade GUID
2: Foo_I.C innehåller GUID-konstanter i C
3: Foo.TLB kallas type library (Typ bibliotek).
4: Foo_p. c innehåller marshalling kod.
5: DLLDATA.C innehåller kod för proxy-stub (beskrivs senare)

2.4.6 Virtuell Funktions Tabell (VTBL)

När det definieras en pure abstrakt klass i C++ så skapas en array med hjälp av en kompilator. Denna array innehåller en pekare till en pekare som i sin tur pekar på adressen av funktionerna i objektet. Denna array kallas ofta som Virtuell Funktion Tabell eller VTBL. COM använder denna teknik för att komma åt olika funktioner i ett objekt. Interface är egentligen en abstrakt bas klass.
Så strukturen på en COM VTBL liknar mycket en basklass i C++ med några skillnader. I en VTBL av en COM komponent ser man alltid de tre funktionerna som varje interface har ärvt från IUnknown (QueryInterface, Addref och Release)



Figur 2.4 Visar hur vtlb pointer och interface metods [COM4]*

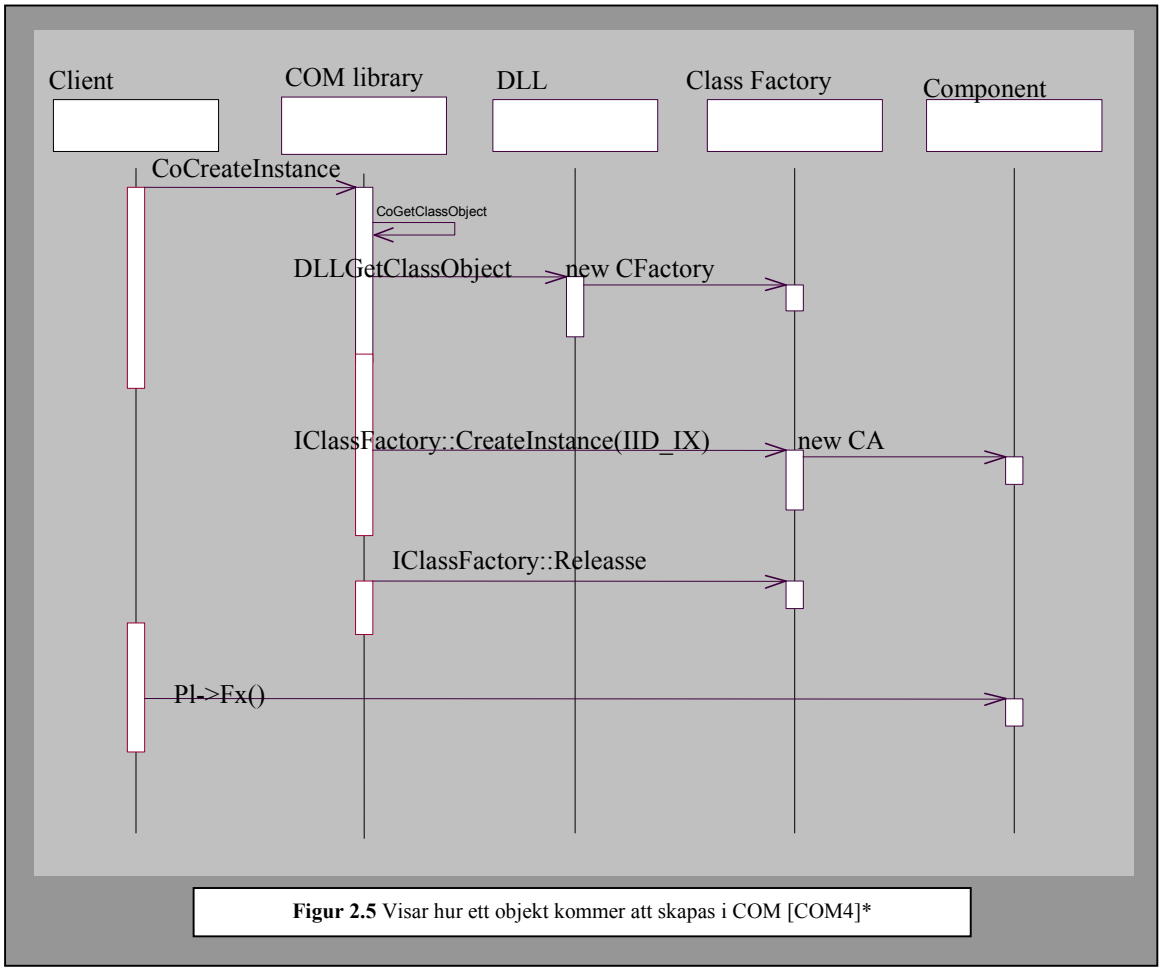
I figur 2.4* visas fyra metoder, Myfunc och de tre metoderna från interfacet IUnknown.
C++ klasser kan hantera och använda klassens data genom instanser men COM komponenten har aldrig tillgång till data.

2.4.7 Att skapa objekt i COM

En COM-klass implementerar ett eller flera interface (gränssnitt) och utifrån klassen skapas objekt för att anropa metoder i gränssnittet. Efter att en klass har skapat ett objekt behövs ingen referens eller pekare till klassen. Det vill säga klassen används bara för att skapa objekt (klassen är bara en mall) och all kommunikation med objektet hanteras genom gränssnittet. Klasser identifieras med Class ID (CLSID) som är ett GUID. Men för att underlätta för programmeraren används ofta det logiska namnet, program ID (ProgID) när man skriver kod.
För att skapa ett objekt behövs en pekare till COM-klassen för det objekt man vill skapa. För utföra denna operation använder COM den fördefinierade funktionen CoGetClassobjeckt(). Denna funktion har bland annat klassens ProgID som parameter och returnerar en pekare till klassens interface IclassFactory(). Därefter anropas metoden CoCreateInstance (som ligger i interfacet IclassFactory()) och denna ger tillbaka en interfacepekare till

* Bilden avbildad från referens [COM4].

objektet av klassen. Figuren (2.6) visar de olika steg som utförs för att skapa ett objekt i COM.
Först anropar klienten CoCreateInstance och i sin tur anropar CoGetClassObject som söker efter komponentens sökväg i registret. Om den hittar sökvägens laddas DLL-filen. Efter att DLL-filen har laddats så anropar CoGetClassObject DLLGetClassObject. DLLGetClassObject är implementerade i DLL-server och dess jobb är att skapa ClassFactory. DLLGetClassObject frågar ClassFactory efter interfacet IClassFactory som returnera till CoCreateInstance. Sedan använder CoCreateInstance IClassFactory interfacet för att anropa dess CreateInstance funktion. IClassFactory: CreateInstance anropar operator new för att skapa komponenten. Den frågar komponenten efter aktuella interfacet. När den har fått interfacet tas Classfactory bort och interfacet returneras till klient. Nu kan klienten använda interface-pekaren för att anropa metoder på komponenten. Se Figur 2.5*



Figur 2.5 Visar hur ett objekt kommer att skapas i COM [COM4]*

* Bilden avbildad från referens [COM4].

CoCreateInstance tar fem parametrar och returnerar som de flesta metoder i COM HRESULT enligt följande:

```
HRESULT __stdcall CoCreateInstance (
                    const CLSID& clsid,
                    IUnknown *pIUnknowOuter,
                    DWORD dwClsContext,
                    const IID& iid,
                    void ** ppv ) ;
```

1. CONST CLSID& clsid, GUID för Klass_ID
2. IUnknown *pIUnknownOuter, en pekare till ett objekt som detta objekt är aggregerade av.
3. DWORD dwCLSContext, var komponenten ska exekveras, samma dator, annan dator eller samma process.
4. Const IID& iid, GUID för interface_ID
5. void** ppv, nn pekarvariabel till önskade object

2.4.8 Registrering av komponenter

Ett anrop till CreateInstance() väcker upp COM:s service Control Manager(SCM) för att hitta aktuell klass och skapa objektet. Innan objektet skapas utför SCM en säkerhetskontroll för att se om den anropade klienten tillåter att skapa objektet över nätverket. Efteråt kan SCM kontrollera om varje anrop till en metod är tillåten. En komponent måste vara registrerad i registret för att SCM ska kunna hitta och instansiera komponenten. När en komponent registreras skapas nycklar i registret som bland annat innehåller sökvägen till komponenten. Om komponenten används på samma dator som den utvecklades brukar vissa IDE:er (t.ex. VisualBasic) registrera komponenten automatiskt. Eller så kan man använda sig av programmet REGSVR32.EXE (t.ex. Visual C++). Om komponenten ska exekveras på en annan dator, så måste datornamn anges i registret.

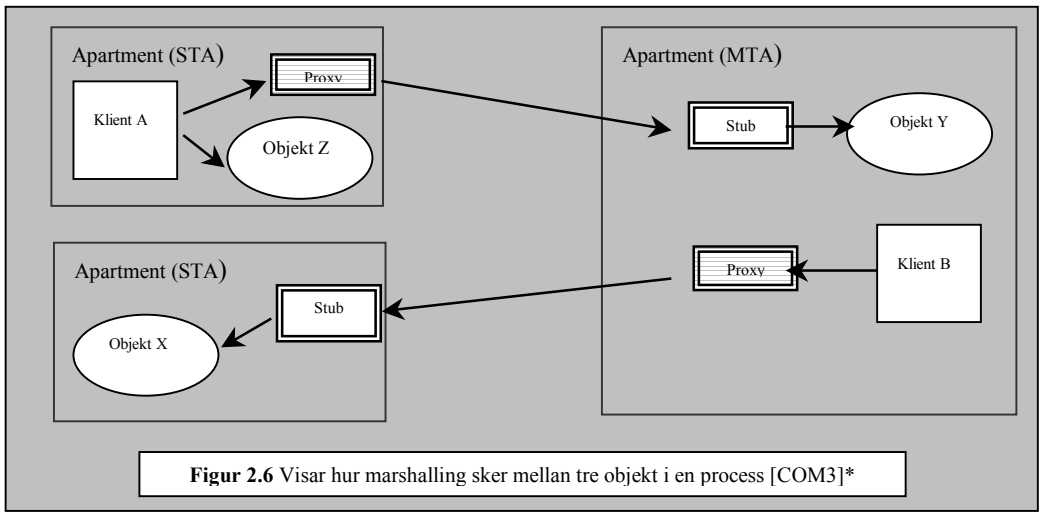
2.4.9 Apartment

När en programkod ska exekveras sker detta i en process och processen tilldelas en del resurser av operativsystem bland annat CPU-tiden och minne. Exekveringen inom processen utförs av något som kallas tråd (thread). De resurser som tilldelades av processen delas mellan olika trådar. Varje process kan innehålla en eller flera trådar och varje tråd associeras med en apartment för att kunna hantera synkronisering av MSCOM objekts exekvering. Komponenter i en apartment är inte medveten om komponenter i andra apartment därför kan man inte dela på globala variabler eller hålla en referens direkt till en komponent i andra apartment. Vilken typ av apartment komponenten ska exekveras i bestäms med hjälp av trådmodellen, när man utvecklar eller registreras komponenten. För applikationer anges typen av apartment när COM-miljön initieras med hjälp av API-funktionerna CoInitialize() och CoInitializeEx().

I MSCOM finns tre olika apartment, SingleThread Apartment (STA), Multi Thread Apartment (MTA) och Neutral Thread Apartment (NTA).
I en STA kan endast en komponents metod exekveras åt gången och resten av metoder måste stå i en meddelandekö vilket leder till långa svarstider. Synkronisering löses med hjälp av Windows inbyggda meddelandekö för fönster som hanterar meddelande i den ordning de kommer fram. Varje process kan ha flera STA och den första STA som skapas kallas för Main STA. MTA hanterar inte synkronisering av meddelandekö och det kan finnas flera anrop av samma eller flera komponenter samtidigt. I detta fall är det upp till utvecklaren själv att ta hand om synkronisering. MTA kräver en mer komplex programmering jämför med STA men däremot vinns i skalbarhet och olika metoder kan exekveras samtidigt. I denna typ av apartment kan varje process endast ha en MTA.
NA har kommit tillsammans med Windows2000 och COM+ och varje process av denna typ kan bara ha en NA. När en tråd exekveras i NA miljö kan objektens kontext användas utan trådbytning.

2.4.10 Marshalling

Marshalling i MSCOM innebär att skapa en indirekt referens till komponenter i andra apartment eftersom COM inte tillåter att en klient har en referens till en komponent i en annan apartment. Detta är på grund av att COM ska kunna hantera de olika typer av synkronisering som finns i STA och MTA.
Vid marshalling skapas två små objekt, en proxy på klientsidan och en stub på serversidan som hanterar kommunikationen mellan två objekt indirekt. När en klient anropar en metod i komponenten kommer den att göra anropet mot proxy-objektet. Proxyn paketerar eventuella parametrar och skickar anropet till stub-objektet som paketerar upp parametrarna. Stuben anropar sedan metoden i komponenten och tar hand om resultatet. Resultatet paketeras för att sedan skickas tillbaka till proxy som i sin tur paketerar upp resultatet och skickar till klienten, figur 2.6*



Figur 2.6 Visar hur marshalling sker mellan tre objekt i en process [COM3]*

* Bilden avbildad från referens [COM3].

Marshalling leder till mer arbete och exekveringstid bli längre därför bör undvikas komponenter som ska exekveras på olika apartment.

2.4.11 Trådmodeller

MSCOM erbjuder fem olika trådmodeller och de används för att ange i vilken typ av apartment som komponenten ska exekveras.

Single Tråd

Denna modell är ett arv från tiden då bara 16-bitars Windows fanns. I 16-bitars Windows kunde bara en process exekveras åt gången.

Apartment tråd

De komponenter som är avsedd för denna modell kan endast exekveras i en STA. Om anropade applikationen exekveras i en STA kommer komponenten att exekveras i samma STA. Däremot om applikationen exekveras i en MTA kommer en ny STA skapas för komponenten att exekveras i. Det senare fallet kräver att operativsystemet utför marshalling.

Free tråd

Denna modell används för komponenter som endast exekveras i en MTA. Om anropande applikationen exekverar i en MTA kommer komponenten exekvera i samma MTA och anropen kan ske direkt mot komponenten. Men om anropande applikationen exekverar i en STA kommer komponenten att skapas i processen i en MTA och marshalling behövs.

Both

I denna modell exekveras komponenten i samma apartment som anropande applikation. Med andra ord om anropade applikationen befinner sig i en STA kommer komponenten att exekveras i en STA och samma sak gäller för MTA.

Neutral

Komponenter med denna modell använder sig av en annan typ av proxy-objekt eller light weight proxy.

Det kommer att beskrivas mer djupare om trådning modell i samband med MTS i nästa avsnitt.

2.4.12 Datatyper i COM

COM erbjuder en del av datatyper som används i IDL-språket. Figur 2.7* visar de olikatyper i COM.

Language	IDL	Microsoft C++	Visual Basic	Microsoft java
Base Types	Boolean	Unsigned char	Unsupported	Char
	Byte	Unsigned char	Unsupported	Char
	Small	Char	Unsupported	Char
	Short	Short	Integer	Short
	Long	Long	Long	Int
	Hyper	Int64	Unsupported	Long
	Float	Float	Single	Float
	Double	Double	Double	Double
	Char	Unsigned char	Unsupported	Char
	wchar_t	wchar_t	Integer	Short
	Enum	Enum	Enum	Int
	Interface pointer	Interface pointer	Intrface ref.	Interface Ref.
Extended Types	VARIANT	VARIANT	Variant	ms.com.Variant
	BSTR	BSTR	String	java.lang.string
	VARIANT_BOOL	Short [-1/0]	boolean [True/false]	boolean[true/false]

Figur 2.7 Tabellen visar de typerna som IDL stödjer [COM1]*

2.5 Exekvering av komponenter

2.5.1 Exekvering i COM

När komponenter kompileras kan detta göras till en vanlig exekverbar fil (EXE-fil) eller till länkbar fil (DLL-fil). En
DLL-fil innehåller också exekverbar kod, men DLL-filen behöver en process som anropar den exekverbara koden. En sådan process kan vara en EXE-fil eller en surrogatprocess (bl.a. MTS).
En EXE-fil kan exekvera som ett fristående program men det kan också exekvera som en tjänst i Windows NT/2000. En tjänst kan exekvera utan att någon användare är inloggad i motsats till ett fristående program som kräver att en inloggad användare startar programmet. DLL-filer måste laddas med hjälp av ett anrop från en EXE-fil eller en surrogatprocess.
En applikation kan exekvera på tre olika sätt i COM:

In-process
: Samma process som anropande applikation/komponent som DLL-file

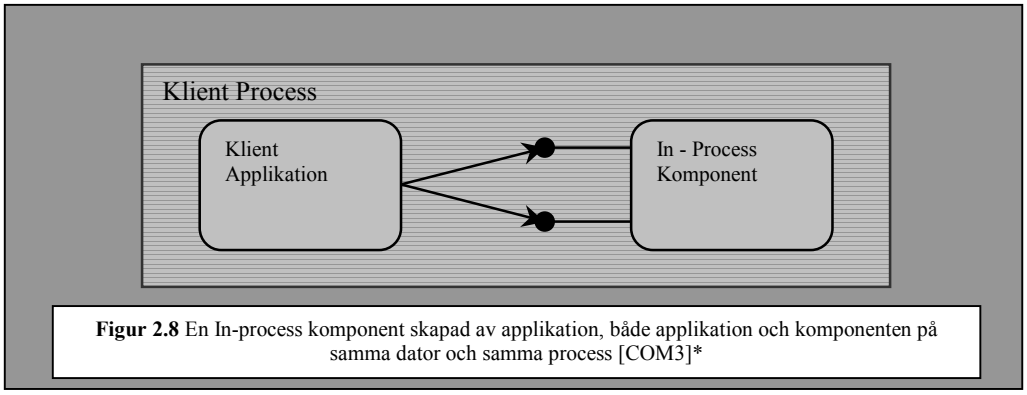
Out-process
: Som en fristående programvara som EXE-file

Out-process (remote server)
: På en annan dator som DLL-fil eller EXE-file.

* Bilden avbildad från referens [COM1].

2.5.2 In-process

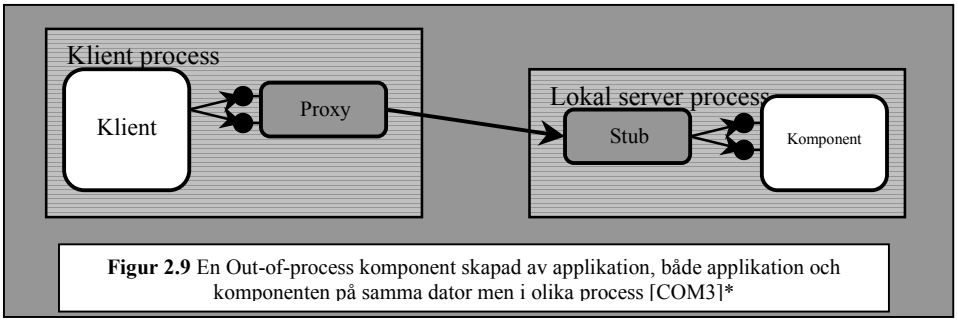
När en komponent exekverar In-process exekverar komponenten på samma dator och i samma process som anropande applikation. Komponenten delar på anropande applikationens resurser. Eftersom anropen av komponents metoder kan ske direkt mot komponenten så blir svarstiderna relativa snabba. En komponent som ska exekvera In-process måste skapas som en DLL-file. Nackdelen med In-process exekvering är att om komponenten utför en förbjuden åtgärd kan den krascha hela exekveringen i processen, figur 2.8*



Figur 2.8 En In-process komponent skapad av applikation, både applikation och komponenten på samma dator och samma process [COM3]*

2.5.3 Out-of-process

När komponenter exekverar på samma datorer men utanför anropande applikationsprocess kommer komponenterna att ha tillgång till egna resurser, figur 2.9*



Figur 2.9 En Out-of-process komponent skapad av applikation, både applikation och komponenten på samma dator men i olika process [COM3]*

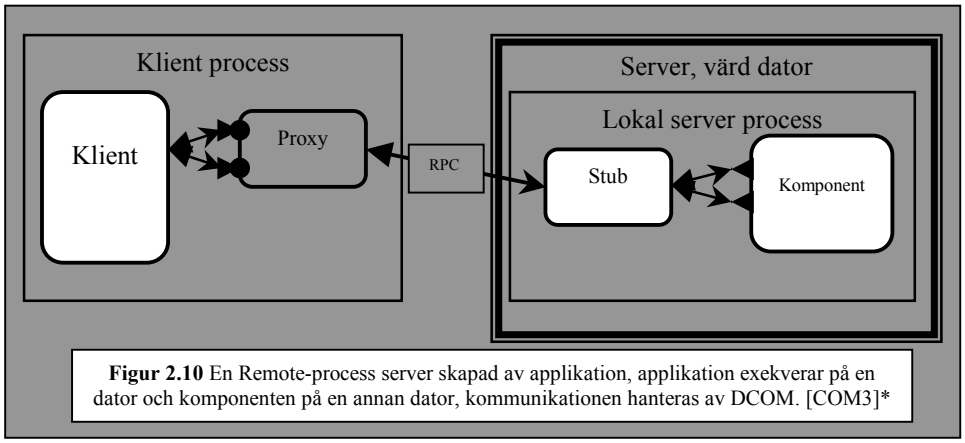
Nackdelen är att anrop av komponentens metoder måste ske genom så kallad marshalling vilket leder till mer arbete och ger därmed längre svarstider. Anropet av metoder är lika som vid In-process. MSCOM hanterar denna marshalling transparent. De komponenter som ska exekvera Out-of-process kan skapas som EXE-filer eller DLL-filer som använder till exempel MTS som surrogatprocess. Fördelen med Out-of-process är att om komponenten

* Bilderna avbildad från referens [COM3].

utför en förbjuden åtgärd kan påverkan av applikationens exekvering förhindras genom felhantering i anropande kod.

2.5.4 Remote-server

Fast komponenten här exekverar på en annan dator är principen detsamma som Out-of-process. I detta fall kommer marshalling att ske på samma sätt men mer omfattande, figur 2.10*



Figur 2.10 En Remote-process server skapad av applikation, applikation exekverar på en dator och komponenten på en annan dator, kommunikationen hanteras av DCOM. [COM3]*

Svarstider kan bli längre eftersom anrop av metoder måste skickas över ett nätverk. Komponenter som ska exekvera på annan dator kan skapas som EXE-filer eller DLL-filer som använder MTS som surrogatprocess.

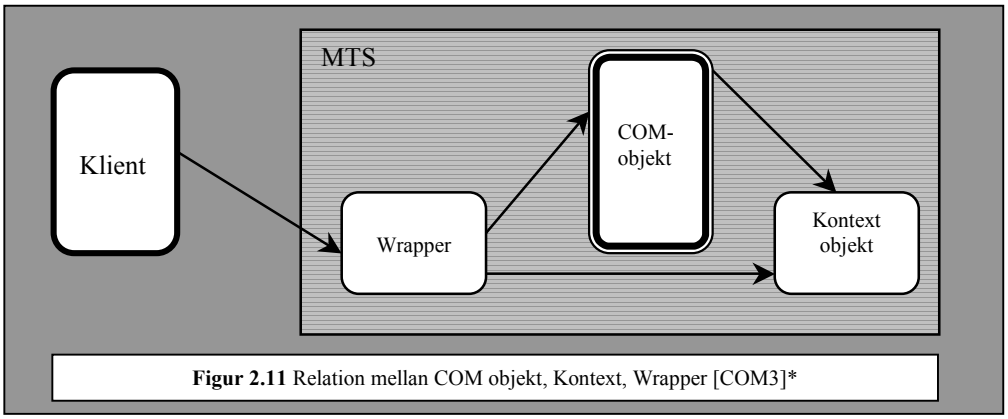
2.6 Microsoft Transaktion Server (MTS)

Microsoft Transaction Server är en utökning av MSCOM. MTS erbjuder en del tjänster som underlättar att programmera distribuerade system och programmerare kan koncentrera sig på att skriva affärslogiken. Utvecklingen av komponenter som ska exekveras i MTS skiljer sig lite från lokala komponenter. MTS erbjuder olika tjänster bland annat transaktionshantering och surrogatprocess åt In-process komponenter som exekverar på en server. MTS är bäst användbart i multiklient/server applikation och Webapplikationer.
I klient/server modell har MTS tre följande uppgifter:

- Komponents transaktion, MTS koordinerar och övervakar transaktionstillstånd på komponenten när den körs.
- Objekt brokering, MTS är ansvarig för att skapa objekt från fjärklienter
- Resurs pooling, MTS använder optimala resurser för att skapa en pool av resurser för att komponenter kan används vid olika tillfället.
- Säkerhet, MTS ger en flexibel och säkertmodell för att kontrollera access på komponenten under exekvering (runtime).
- Administration, MTS tillåter att ändra konfigurationen av klient/server system både under och efter utvecklingen utan att behöva ändra i koden.

* Bilden avbildad från referens [COM3]

När ett COM-objekt skapas i MTS skapas också två objekt till, ett wrapperobjekt och ett kontextobjekt. Wrapperobjektet har samma gränssnitt som COM-objekt och placeras mellan klienten och själva COM-objektet. Det vill säga klienten får en gränssnittpekare till wrapper-objektet som i sin tur refererar till COM-objektet. Men alla anrop görs via wrapper-objektet och MTS kan stödja Just-In-Time (JIT) aktivering och säkerhetskontrollerar anrop mot objekt med hjälp av Kontextobjekt. Kontextobjektet innehåller all information om objektet så som transaktionsstatus och säkerhetsinställningar, figur 2.11*.



Figur 2.11 Relation mellan COM objekt, Kontext, Wrapper [COM3]*

JIT-aktivering av komponenter i MTS används för att spara resurser på applikationsservern. Detta gör MTS genom att inte skapa själva COM-objektet innan klienten anropar en metod i objektet. När metoden avslutas tar MTS bort objektet för att återta de resurser som objektet använde. Wrapper-objektet kommer att vara kvar så länge klienten har en referens till COM-objektet (wrapperobjektet). JIT-aktivering och objektpooling ger en bättre prestanda och servern kan tjäna flera klienter.

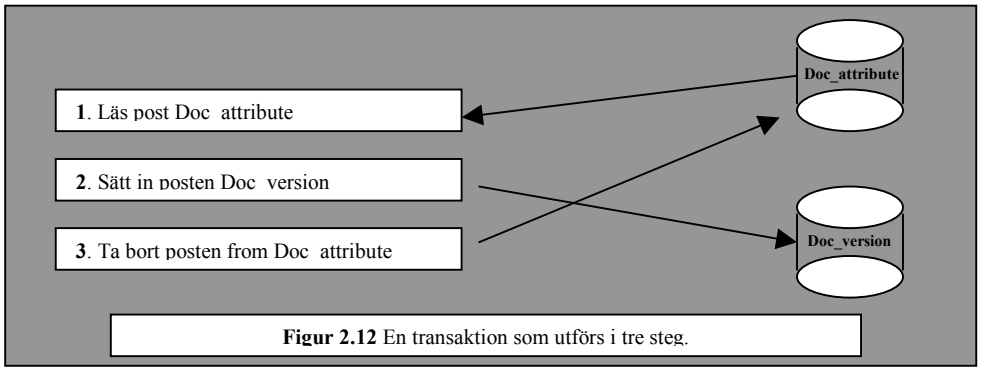
2.6.1 Transaktion

En transaktion är en samling av jobb som består av flera steg eller en operation som antigen körs i sin helhet eller inte alls. Många steg eller jobb tar plats i en transaktions räckvidd men i slutet av proceduren måste transaktionen antigen göra commit eller abort. Commit görs när alla steg av transaktionen har utförs och abort utförs när transaktionen misslyckas. En transaktion måste vara ACID, det vill säga bör följa fyra regler:

- Atomära (Atomicity): en transaktion måste utföras i sin helhet eller inte alls.
- Konsekventa (Consistency): resultatet av två eller flera transaktioner ska vara samma oavsett i vilken ordning de avslutas.
- Isolerade (Isolation): en transaktion ska inte påverka en annan.
- Beständiga (Durability): förändringar gjorda i transaktioner ska bestå, (ej försvinna efter datorhaveri).

* Bilden avbildad från referens [COM3].

Komponenttransaktion liknar mycket databastransaktionskoncept. MTS stödjer komponenttransaktion genom att bevaka dem när de ska komma åt databasen.
En viktig sak med komponenttransaktion är möjligheten att ta hand om transaktion under multiple databas. Anta en enkel uppgift att flytta en post från en databas till annan, figur 2.12.



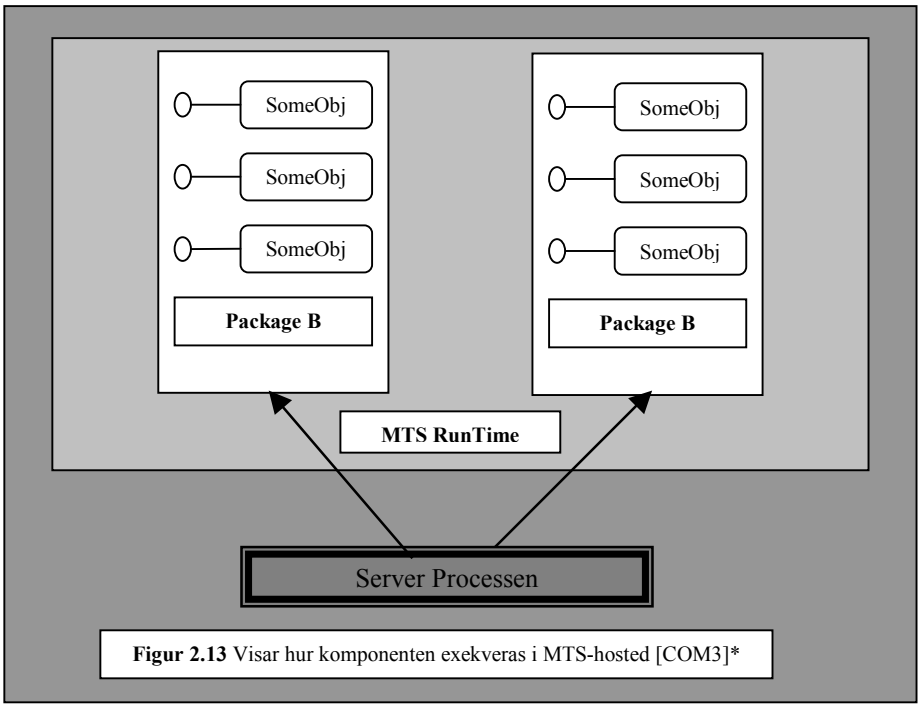
Figur 2.12 En transaktion som utförs i tre steg.

Flytta en post från en databas till annan databas kräver tre steg:

1. Läsa posten från en källa databas.
2. Sätta in den nya posten destination databaser med det data som lästes från första steget.
3. Ta bort original posten från källa databasen.

Alla steg ska exekveras oberoende av varandra. Om ett fel uppstår under steg 2 och inte i steg 3, är resultatet inte det som önskas eftersom missas originalposten för alltid.
Ett annat möjligt problem som kan inträffa är om systemet kan få fel under steg 3, resultatet blir att finnas två data av samma post, som inte är önskvärt.
Som nämnts tidigare kan ett COMobjekt bygga på tre olika typer, In-process, Lokal–process och Remote–process. Om ett objekt ska vara MTS-anpassande måste det byggas som en In-Process (dynamic link libraries). Detta val hjälper MTS att kontrollera var kod och data för varje instans av en komponent ska placeras. Vanligtvis exekveras komponenten i en MTS-hosted serverprocess, figur 2.13*.

* Bilden avbildad från referens [COM3].



Figur 2.13 Visar hur komponenten exekveras i MTS-hosted [COM3]*

För att kunna utveckla komponentbaserade transaktioner, måste MTS hantera all tillståndsinformation för varje instansierad komponent av alla komponenter som är installerade under dessa kontroller. Detta görs via mottagning av instansierad komponent och skapande av ett syskon (sibling) objekt under skapandet av originalobjekt. Det andra objektet är känd som kontextobjekt och dess jobb är underhållning av information om kontext av nya objekt. En del information som är lagrade i ett kontextobjekt är transaktionsdeltagande, tråd av exekveringsinformation och säkerhetsinformation.

2.6.2 Kommunikation med MTS via kontext objekt

Varje kontextobjekt har COM-interfacet IObjectContext, som definierar alla metoder som aktuell komponent kan anropa. IObjectContext representerar en primitiv API-funktion för MTS komponent.
IObjectContexts metoder innehåller tre huvudfunktioner, komplettera eller avbryta komponents transaktion, möjliggöra eller avbryta en transaktion, kontrollera säkerhet av den som anropar komponenten.
De två viktiga metoderna (SetComplete och SetAbort) i IObjectContext informerar MTS när en komponent har lyckats eller misslyckats.
Komponenterna anropar i sin tur IobjectContect::SetComplete() i vanliga fall men om det inträffar något fel används IObjectContect::setAbort().

2.6.3 Skaffa en referens till IObjectContext

För att anropa IObjectContext:s metoder måste komponenten skaffa en referens för interfacet (IObjectContext) och sedan måste anropas globala API funktionen GetObjectContext().
Komponenten anropar GetObjectContext() som returnerar IObjectContext interface pekare som implementerar via komponentens syskonkontextobjekt. På det interfacet (IObjectContext) sitter SetComplete och Setabort() metoder.

2.6.4 Möjliggöra transaktion i MTS

Efter installation av komponenter måste konfigureras dem för olika transaktionsnivåer. Figur 2.14* visar en tabell som beskriver fyra möjliga nivåer.

Nivå	Beskrivning
1. Behöver en transaktion	Indikera att en komponent alltid körs under en transaktion. Om komponentens skapare är under en transaktion, då används den transaktion för den nya komponent, annars skapas en ny transaction.
2. Behöver en ny transaktion	MTS skapar en ny transaktion objektet oavsett transaktionstillstånd av komponentens skapare.
3. Support transaktion	Kör objekten under transaktion om skaparen kör under en, annars kör inte under en transaktion
4. Inte support transaktion	Objekt körs inte under kontext av transaktion. En syskon kontext skapas fortfarande for komponent, även om komponenten inte kör transaktion.

Figur 2.14 En beskrivning av de olika transaktionsnivåerna i MTS [COM3]*

2.6.5 Ärva en förälders transaktionskontext

Tre av de fyra transaktions nivåer använder transaktion kontext av komponents skapare för att bestämma kontexten av den nya komponenten. Den situation som förekommer mest är när en komponent skapar och underhåller en referens till en annan komponent. Om man väljer en transaktionssupport nivå förutom fjärde nivå (Inte support transaktion) kontrollerar MTS först den transaktionens kontext av förälder komponenten för att bestämma kontexten för barnet.

* Bilden avbildad från referens [COM3].

Val av andra nivå (behöver en ny transaktion) orsakar att MTS skapar en ny transaktionkontext för varje instans av komponenten. Alltså, MTS skapar en nästlad transaktion om objektensskapare kör under sin egen transaktion. Det måste väljas ny transaktion om man inte vill att barnets komponent påverkas av förälderns transaktion.

2.6.6 Implementation av IObjectControl interface

Som nämnts förut är ett interface en kollektion av metoder som en komponent är tvungen att implementera (i C++ sammanhang kallas Abstrakt bas klass). Interface definieras oberoende av någon komponent. För bättre skalbarhet och prestanda implementeras IObjectControl. Detta interface innehåller tre metoder Active, Deactive och CanbePooled.
När MTS skapar en instans av en COM-komponent kommer att följande steg inträffa:

1. MTS skapar en instans av komponenten och frågar efter IObjectControl interface.
2. Om komponenten stödjer IobjectControl anropar MTS IobjectControl.Active() som en C++ konstruktör fast för COM objekt. Nödvändiga saker kan läggas i Active metod. Om komponenten inte stödjer IObjectControl då kommer MTS inte att anropa metoderna.
3. Den normala processningen kommer att hända tills den sista klienten befriar den sista referensen till komponenten.
4. MTS kommer att anropa IObjectControl.CanBePooled() för att veta om komponenten stödjer objekt-pooling eller ej.
5. Innan objektets instans blir förstörd kommer MTS att köra IObjectControl.Dactive() metod. Denna metod är lämplig plats för att releasa resurser som man inte behöver längre.

På grund av att databas-kopplingar och initieringar är dyra brukar dessa placeras i IObjectControl.Active().

2.6.7 Komponent säkerhet

Säkerheten i MTS kan uppnås via IObjectContext interface. IObjectContext interface definierar två metoder som används för att kontrollera säkerhet och roll, (IsSecurityEnable() och IsCallerInRole()).
IsSecurityEnable returnerar boolska värde som indikerar om säkerheten är möjlig för komponenten. IsCallerRole tar emot en singel sträng parameter som identifierar roll och det är den returnerade boolskt värde som indikerar om rollen.

2.6.8 Gemensam data mellan instanser av komponenter

En invecklad klient/server applikation behöver ofta omplacera data mellan objekten. Ibland behövs det skicka data mellan oberoende instanser av en

komponent, även om livslängden av dem inte är likvärda. Eller om det behövs veta antal instanser som har skapats och förstörts för ett objekt (den typ av information kan användas för en webbsida för att analysera saker) då möjliggör MTS den via Shared Property Manager.
MTS stödjer Shared Property Manager med ett litet objekt modell som består av tre COM-komponenter som SharedPropertyGroupManager, SharedPropertyGroup och SharedProperty objekt.
MTS allokerar en singel SharedPropertyGroupManager objekt instans för varje server process som den hanterar. MTS hanterar en separat process för varje paket som innehåller installerade egenskaper på den.
SharedPropertyGroupManager hanterar en kollektion av SharedPropertyGroup. En SharedPropertyGroup innehåller en kollektion av SharedProperty objekten. SharedProperty objekt är det data element som sparar en singel värde, (Integer, String, Array eller Double).

2.6.9 MSCOM trådningmodell och MTS

Utvecklare använder multipla trådar per process för att uppnå den högsta grad av skalbarhet. Denna metod möjliggör att en singel server kan svara samtidigt på flera klientanrop.
Utan multiple trådning måste varje klient stå i kö för att bli besvarad. MSCOM erbjuder tre dominerade och en rekommenderad trådningsmodell för komponent baserade server under Windows singel trådning, apartment trådning, fri trådning.

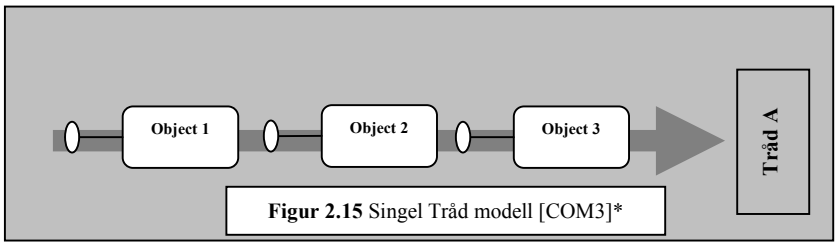
Singeltrådning
Singeltrådning ger den sämsta skalbarheten för att en singeltråd exekverar service för alla komponenters instanser i den servern.

Apartmenttrådning
När det gäller Apartmenttrådning, använder servern mer än en tråd för varje individuell komponents instans. Denna typ av trådning är mer skalbar än singeltrådningsmodell. Däremot måste fortfarande en singel tråd svara för ett objekt.

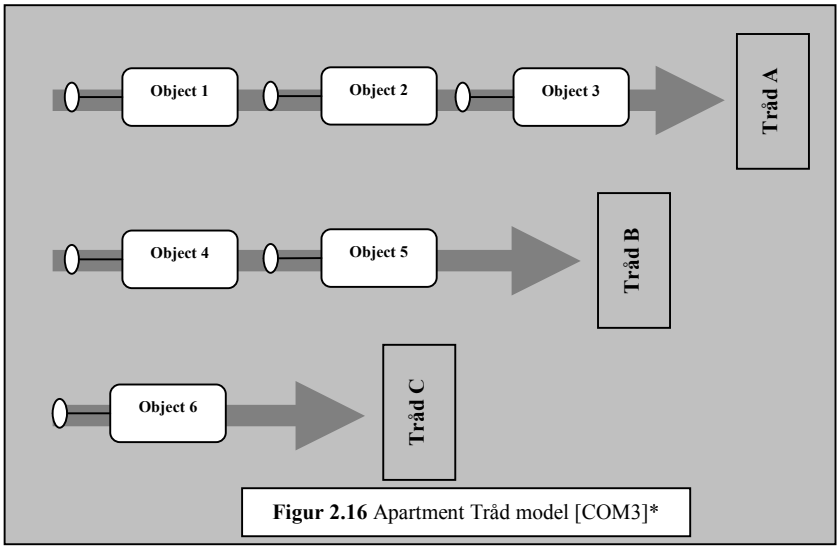
Freetrådning
Free trådningsmodell är mest skalbar eftersom går att ha många trådar i komponent och de kan exekveras när som helst. Multiple trådar kan exekverar i samma metod och samma tid.

Vi ser att det finns olika trådningsmodeller och för varje av dem finns fördelar och nackdelar t.ex. singeltrådning är enkel men det ger dålig prestanda, eller fri trådningsmodell är svår att implementera men det ger en skalbar komponent. Man behöver inte ha djup förståelse om trådningsmodeller när det gäller att skriva MTS baserade komponent. Ett av målen med MTS är att enkelt bygga system med hög skalbarhet. Man behöver inte skapa många trådar i en komponent för skalbarhet däremot kan man skapa sina komponenter som singeltrådadkomponent och låta MTS skapa många trådar för olika instanser.

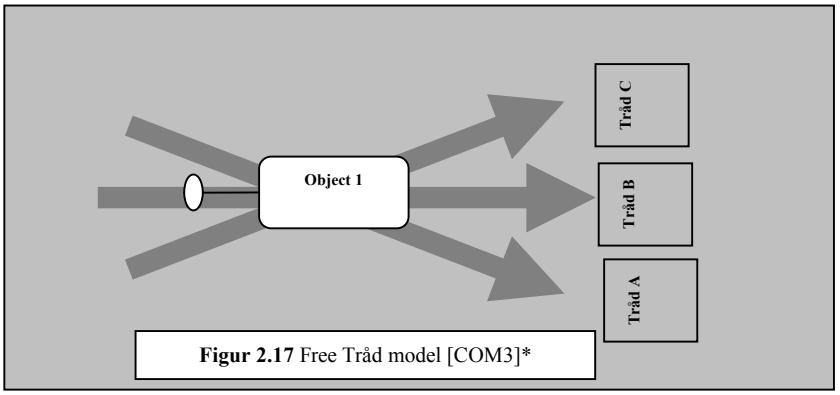
Figur. 2.15-2.17*



Figur 2.15 Singel Tråd modell [COM3]*



Figur 2.16 Apartment Tråd model [COM3]*



Figur 2.17 Free Tråd model [COM3]*

* Bilderna avbildad från referens [COM3].

2.7 Frågespråket SQL

SQL står för Structured Query Language som är ett standardiserat programspråk eller frågespråk för relationsdatabaser. Formellt står SQL för Structured Query Language, men eftersom språket bygger på IBM's äldre frågespråk Sequel uttalas det fortfarande så.
I en relationsdatabas finns det bara en typ av datastruktur, tabellen. Denna likformighet har givit upphov till en ny sorts språk för databasen, SQL. Genom att ge en uppgift till databashanteraren kan hämtas ett antal rader information från tabeller. En annan viktig del i detta språk är att specificeras vad som skall göras istället att ange hur det ska göras. Det är detta arbetssättet som möjliggör att en vanlig användare kan komma åt data. I SQL utförs sökning, filtrering, sortering av relation databashanteringssystemet (RDBMS). Man kan använda SQL interaktivt mot databasen eller inbäddat i 3GL språk som C, Cobol, Fortran m fl. Största fördelen med SQL är att det går att skapa dynamiska webbsidor med hjälp av bakomliggande databaser. Grunderna i SQL är likadana för olika databaser, men sedan skiljer det sig åt, kan sägs att det finns olika dialekter. Några exempel på databaser är Access, SQL Server, MySQL och Oracle.
SQL-statements (satser) kommer att delas i fyra kategorier:

- Queries, beskriver de satser för att söka fram data med valfritt sökvillkor och ordning. Dessa satser kommer inte att ändra något i data bara tar fram data. Dessa satser börjar i vanliga fall med SELECT och satsvillkor.
- DML (Data Manipulation Statements), denna kategori beskriver de satser som används för att ändra data och utförs av följande tre sätt:
  - INSERT, sätter in nya rader eller poster med data i tabeller.
  - UPDATE, uppdaterar kolumnvärden på befintliga rader.
  - DELETE, tar bort rader från tabeller.
- DDL (Data Definition Statements), beskriver de satser som används för att skapa ,underhålla och ta bort dataobjekt, t.ex. CREATE TABLE och DROP VIEW.
- DCL (Data Control Statements), beskriver de satser som används för att kontrollera access till databas, t.ex. GRANT CONNECT och GRANT SELECT.

3 Problemlösning och implementering

3.1 Bakgrund

Efter en del studier av olika teknologier visade det sig att uppgiften kan lösas genom att använda Windows DNA, COM, DCOM och MTS.
Komponenter är skapade enlig COM/DCOM teknik. MTS har använts för att hantera transaktioner och kommunikationer. Lösningen har valts på grund av de fördelar som finns i 3-tier arkitekturen bland annat skalbarhet, flexibiliteten, återanvändbarhet, säkerhet osv.
Som nämnts tidigare innehåller DNA presentationslager, affärslager och dataåtkomstlager och detsamma gäller lösningen på problemet som beskrivs nedan.

3.2 Presentationslager

Presentationslager är ett interface mot slutanvändare och brukar implementeras grafiskt. Detta lager har skrivits i två olika versioner, fristående applikation som har implementerats i Microsoft Visual C++, Visual Basic och inbäddad applikation som har implementerats i Microsoft Word, Microsoft Excel och Web-applikation med hjälp av ASP.

3.3 Dataåtkomst lager

Dataåtkomstlager har bestått av en databas i SQL. Denna databas innehåller fem olika tabeller.

- Tabell 1 (Doc_Attribute): I denna tabell sparas attributen för ett dokument. Varje dokument har fem attribut.
  1. Doc_id, varje dokument innehåller en Doc_id som identifierar dokumentet. Detta attribut genereras av SQL programmet.
  2. Doc_Name, dokumentens namn.
  3. Doc_Type, beskriver typ av dokumentet om det är Word, Excel, egen definierad osv.
  4. Number_Version, anger vilken version dokumentet är 1, 2, 3, 4 eller 5.
  5. Check_in_Out, visar om det går att uppdatera dokumentet eller inte (in-check och out-check). Boolskt värde som indikerar om dokument går att uppdatera eller inte.

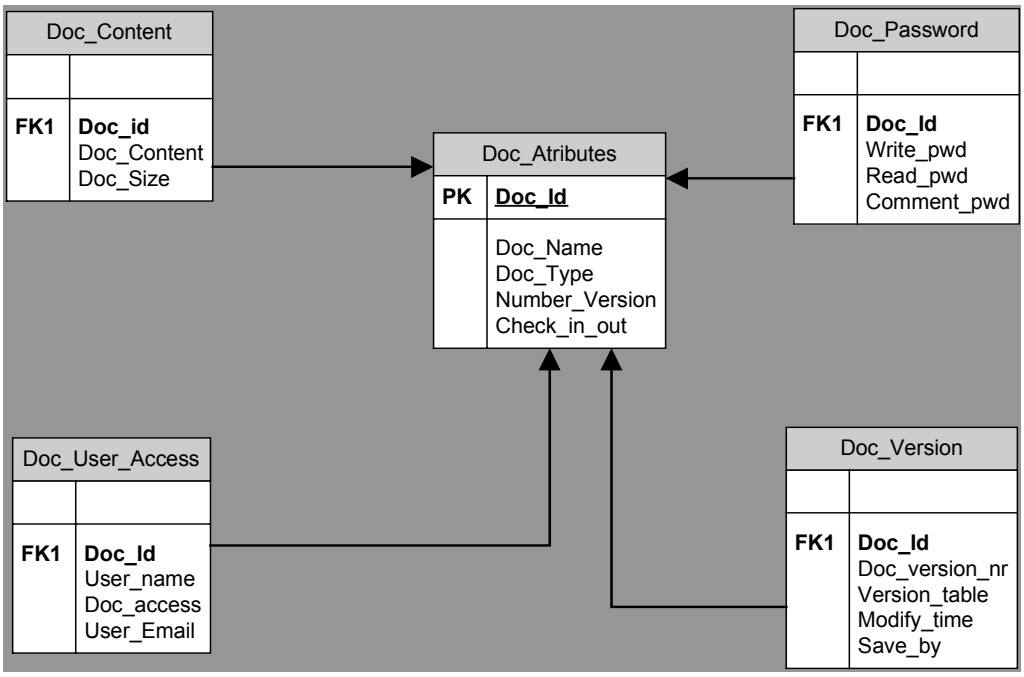
- Tabell 2 (Doc_Content): här sparas innehållet av dokument och det finns fem sådana tabeller, en för varje version. Denna tabell innehåller tre attribut.
  1. Doc_id, identifierar dokumentet.
  2. Doc_Content, beskriver själva innehållet i dokumentet.
  3. Doc_Size, beskriver storleken på dokumentet.

- Tabell 3 (Doc_Password): här sparas dokumentets lösenord (password) och det finns också fem såna tabeller, en för varje version. Denna tabell innehåller fyra attribut.
  1. Doc_id, identifiera password.
  2. Write_pwd, innehåller password för att öppna dokument för skrivning.

3. Read_pwd, innehåller password för att öppna dokument för läsning.
4. Comment_pwd, innehåller password för att kommentera dokument (Word).

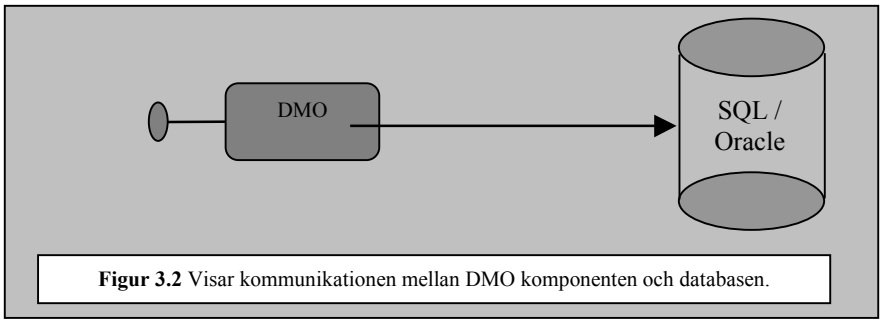
- Tabell 4 (Doc_User_Access): beskriver hur accessen ska utföras och innehåller fyra attribut.
  1. Doc_id, identifierar användaren.
  2. User_name, innehåller användarnamn.
  3. Doc_access, beskriver vilken access en användare har R, W, C eller ingenting.
  4. User_Email, beskriver ägaren av dokumenten (e-mail adress).

- Tabell 5 (Doc_Version): denna tabell innehåller information om version.
  1. Doc_id, identifierar en version.
  2. Doc_version_nr, beskriver versionens nummer av ett dokument.
  3. Version_table, referera till Doc_Content tabell.
  4. Modify_time, anger i vilken tid dokumenten har skapats.
  5. Save_by, anger vem som har sparat sist.



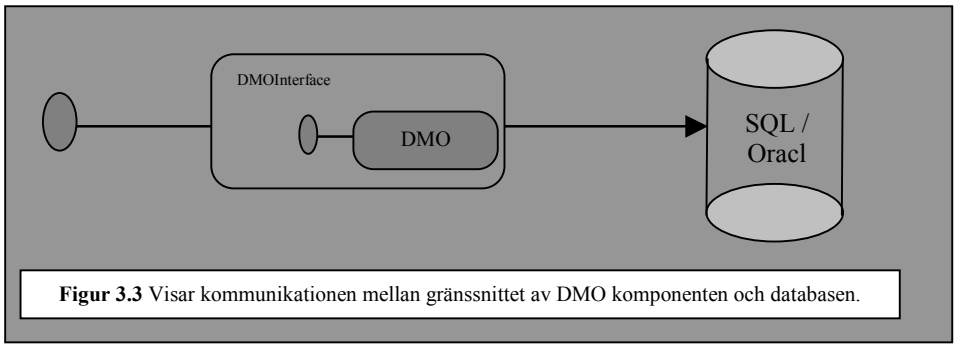
Figur 3.1 DMO Databas diagram (relation mellan tabeller)

I Back-End delen finns en generell modul för att kommunicera med databasen och den innehåller fyra grundläggande operationer Insert, Delete, Read, Update.
Komponenten fungerar oberoende av databasen, tabellnamn, antal kolumn eller placering av databasen. Alla operationer kommer att utföras enligt principen för transaktioner (ACID regler). Applikationen kan hantera både SQL och Oracle, figur 3.2.



Figur 3.2 Visar kommunikationen mellan DMO komponenten och databasen.

DMO har implementerats i Visual Basic med hjälp av Microsoft Transaction Server (MTS) och innehåller tio objekt.
För att kommunicera med DMO finns det en annan komponent som heter DMOInterface, figur 3.3.

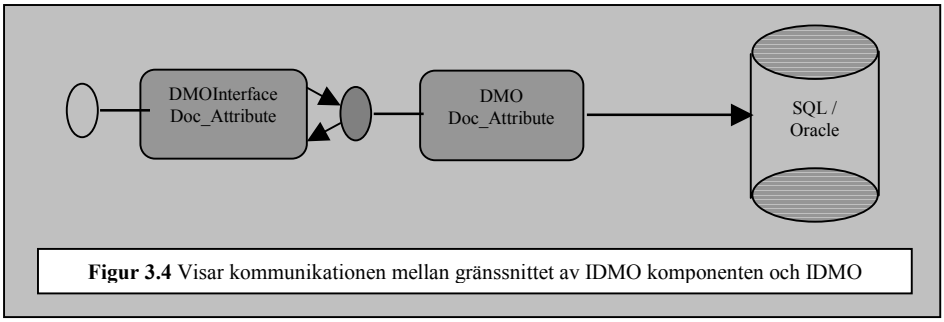


Figur 3.3 Visar kommunikationen mellan gränssnittet av DMO komponenten och databasen.

DMOInterface har implementerats för att underlätta ändringar i applikationen. Detta gör programmet mer återanvändbar och flexibelt. Att det finns två lager eller komponenter för att transportera data har flera fördelar. Om i framtiden vill bytas datakällan från databasen till filsystem då räcker det med att skapas en DMO som läser från filsystem istället från databasen. Eller om ändras några attribut eller egenskaper i ett dokument då räcker bara att ändras i DMOInterfacet. Det går också att skriva olika delar av programmet i anpassande språk med avseende på prestanda, t.ex. om datakällan är filsystem är det mycket bättre att DMO skrivs i C++ jämför med Visual Basic. Detta möjliggör också en bättre fel hantering av programmet.
DMOobjekt innehåller fem komponenter som tar hand om varje tabell Doc_Attribute, Doc_Content, Doc_password, Doc_User_Access och Doc_Version. I varje komponent finns det funktioner som tar hand operationer som utförs på komponenten:

– Read, läser från källdata. Källdata är specificerade med in-parameter.
– Delete tabort saker från källdata.
– Update, uppdatera gamla värden med nya.
– Insert, skriver till källdata.
– Query, söker i källdata.

Varje komponent utför sina operationer på källdata via motsvarande komponent i DMO objekt, figur 3.4.



Figur 3.4 Visar kommunikationen mellan gränssnittet av IDMO komponenten och IDMO

3.4 Affärslogiklager

Detta lager hanterar kommunikationen mellan dataåtkomst och applikationlager. Detta lager uppdelas i sex delar Access Hantering, Version Hantering, Lastbalansering, CheckIn/CheckOut, Offline/OnLine och UnikIdentifering. Här nedan beskrivs de viktiga delarna.

3.4.1 Accesshantering

Just nu hanterar detta lager tre nivåer av access:
1. Read, möjliggör användare att se/läsa dokument.
2. Write, möjliggör användare att skriva eller modifiera dokument.
3. Comment, möjliggör användare att lägga till kommentarer på dokumenten.

För att hantera accessnivåer på dokument används en tabell i databasen som innehåller olika egenskaper d.v.s. Doc_id, User_name, Doc_access, User_Email.
Varje användare som är inloggad på nätverk eller maskiner har ett användarnamn (username). Med hjälp av API-funktioner läses användarnamnet och sedan letas efter användarnamnet i databasen för att kontrollera dess access nivå (D.v.s. en del av säkerheten är integrerad med Windows OS). Om första gången skapas ett dokument sparas användarnamnet och tilldelas rättigheten Write. Administrator har rättighet att ändra alla access nivåer. Om vill läggas till en annan access nivå till nuvarande access nivåer då är businesslogiken det enda ställe som behövs ändra i Dokument management applikation (fördel med DNA arkitekt).

3.4.2 VersionHantering

En annan sak som tas hand om i detta lager är versionshantering. Det finns en tabell Doc_Version som innehåller olika kolumner (egenskaper), d.v.s. Doc_id, Doc_version_nr, Version_table, Modify_time och Save_by. Doc_version_nr anger versionsnummer av dokumenten. På motsvarande sätt finns det också i Version_table ett Doc_version_nr som säger i vilken tabell den rätta versionen ligger. T.ex., om Doc_version_nr är 2 och Version_table är 3 det tolkas att version 2 ligger i Doc_content_3. Om någon har en access nivå på ett dokument då gäller den på alla versioner.
Applikationen tar hand om fem versioner av dokumenten. Detta innebär att om man sparar för sjätte gången då sparas det på första versionen och den gamla versionen kommer att raderas bort.

3.4.3 Lastbalansering

Applikationen använder en slumpmässig lastbalansering. Detta innebär att om begärs en version av dokument så är inte självklart att version 1 finns i Doc_Content_1. De olika Doc_content_x tabellerna sparas i olika servrar som sedan hämtas oberoende av varandra. Dessutom undviker man att belasta bara en server för att hämta dokumenten.
Den mesta fördröjnings tiden i dokumenthanteringssystem är sparande eller läsning av ett dokument d.v.s. flaskhalsen är Read och Write operation. Andra fördelen med att sprida dokumentena över flera servrar är tillgänglighet. När begärs en version av ett dokument om den servern är ur funktion då kan hämtas näst senast versioner av dokumentet från en annan server. Här måste påpekas att det inte är ett krav att sprida alla Doc_content_x tabeller i olika servrar.
Doc_Version komponenten är ansvarig för att hitta rätt version i rätt server och spara nya versioner slumpmässigt på annan server. Om lastbalanseringsmekanism ska ändras, räcker att byta Doc_Version komponent med annan komponent. T.ex. från slumpmässig lastbalansering ändra till normal fördelning (när ett anrop av dokumenten begärs för första gången innehåller Version_1 Doc_content_1 och Version_2 Doc_content_2 o.s.v.) Figur 3.5 visar denna situation. Så fort ny version sparas över gamla versioner av dokument då flyttas sista version till näst sista version o.s.v.

Version_1	Doc_content_1
Version_2	Doc_content_2
Version_3	Doc_content_3
Version_4	Doc_content_4
Version_5	Doc_content_5

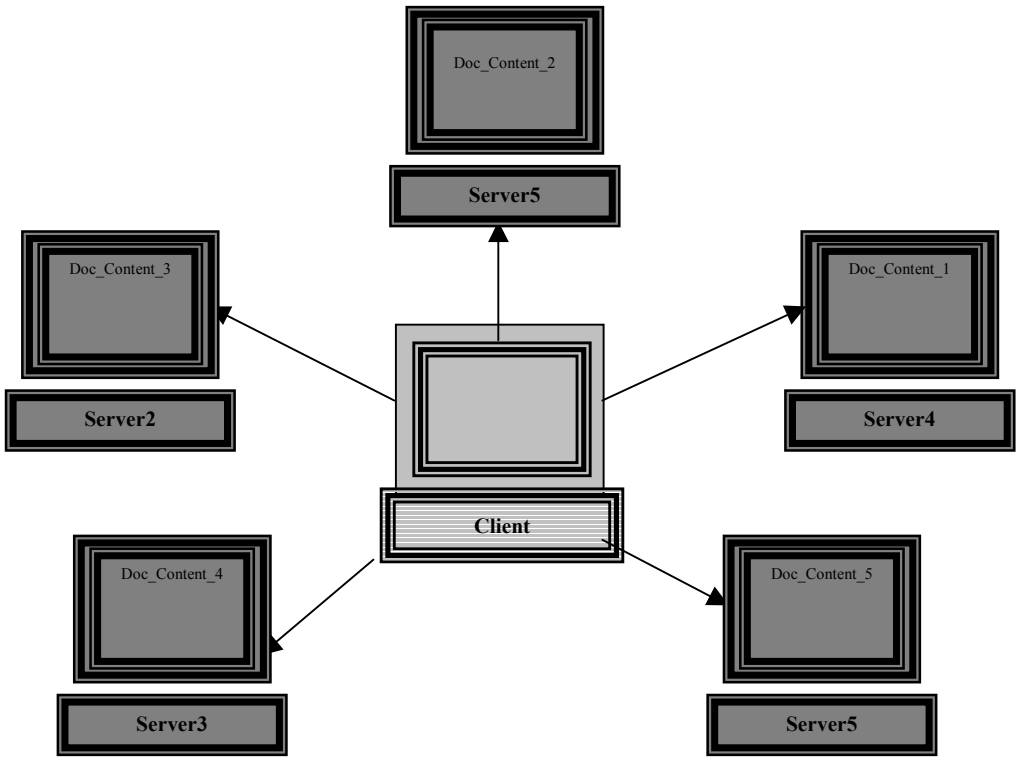
Figur 3.5 Visar olika versioner av dokumenten.

I figur 3.6 visas de dokumenten efter att ny version sparas. Observera att första versionen raderas bort eftersom det bara finns fem versioner av ett dokument.

Version_5	Doc_content_1
Version_1	Doc_content_2
Version_2	Doc_content_3
Version_3	Doc_content_4
Version_4	Doc_content_5

Figur 3.6 Visar olika versioner av dokumenten sparas i olika tabeller.

Med denna metod kan en klient kopplas till olika server och varje server kan innehålla ett dokument. Om någon server kraschar kan andra servrar användas.



Figur 3.7 Visar koppling mellan en klient och fem server med olika versioner av samma

3.4.4 Check In/Out

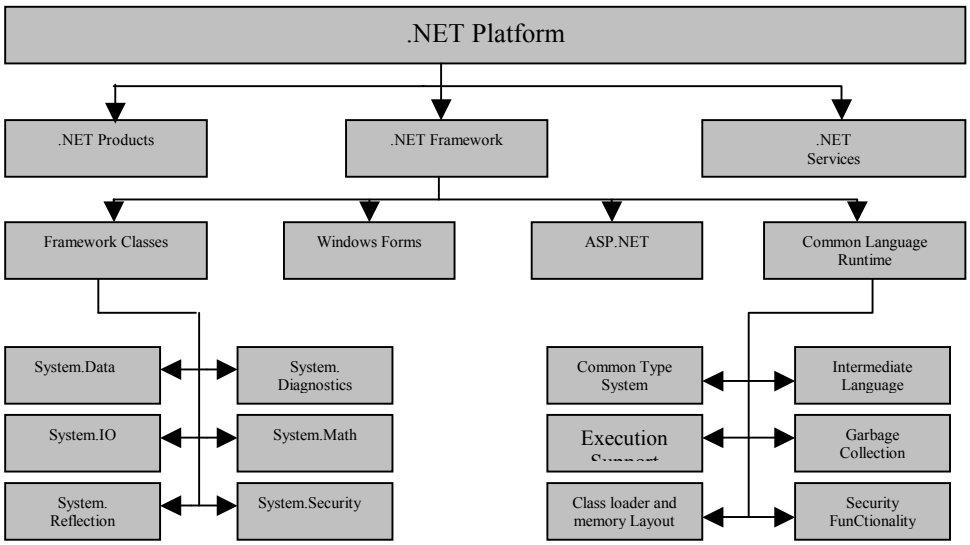
För att hantera Check In/Out används ett attribut för varje dokument som indikerar om dokumenten är uppdaterat (CheckaOut) eller inte i Doc_attribute tabellerna. Om ett dokument är uppdaterat får alla som har rättigheter att se dokumentet, se en read version förstås.
Affärslogikapplikationen innehåller olika klasser och komponenter för att utföra de sakerna som nämndes ovan.

4. Slutsatser

Komponenttänkande är ett nytt sätt att designa dagens applikationer. Fördelar med MSCOM är många, bland annat fås bättre moduler, kan implementeras i olika program språk för olika komponenter o.s.v. De viktigaste nackdelarna med MSCOM är livslängden av komponenten, trådningsmodell och minneshantering. Applikationer som använder komponenten måste vara försiktig när det gäller aktivering och inaktivering av komponenten och den svåra trådningsmodellen gör inte saker bättre, fel trådmodell leder till sämre prestanda. Det finns förstås regler för minneshanteringar i MSCOM. Ett annat problem som jag stötte på under mitt arbete var registrering av komponenter. MSCOM har misslyckats när det gäller installation (deployment) av applikationer som använder detta med andra ord det var verkligen en DLL mardröm (när versioner av komponenter inte matchar varandra). I nästa version av COM (COM+ eller .NET) fås inte dessa problem på grund av det finns ett ramverk som tar hand om exekvering av komponenter och varje komponent har sin egen XML-konfigurationsfil som används för registrering.
Att hantera dokument kan göras på många olika sätt, men problemet är i vilken omfattning det ska göras och vad syftet är av hanteringen. Därför har jag lagt stor vikt på återanvändbarhet, skalbarhet och flexibilitet. Lösningen av uppgiften resulterade i hög grad med de krav som företaget ställde i början och ibland mycket mer. Att studera olika teknologier gav mig en väldig bra syn på komponenthantering och olika teknologier. Denna uppgift var mycket lärorik och nyttig på flera sätt. En av de viktiga saker som jag lär mig efter en sådan uppgift är att jobba självständigt.
Utvecklingsfasen av problemet var väldig tidskrävande. Problemet som inträffade under arbetet var att jag var mycket slarvig att skriva rapport under arbetets gång. Detta orsakade att hela rapporten kanske inte kommer att bli sammanhängande. Under tiden har Microsoft släppt efterföljare till COM teknologin kallad .NET. Därför har lagts en undersökning av en ny komponentteknologi som appendix i slutet av rapporten. I detta appendix ges en bild av .NET teknologi.

5. Appendix A: Dot Net Teknologi

Dot Net (.NET) är Microsoft nästa komponentteknologi som baseras på XML och SOAP. Man kan dela .NET i tre stora delar .NET services, .NET products och .NET Framework, Figur 5.1*.



Figur 5.1 Visar olika delar I .NET teknologi. [DOTNET1]*

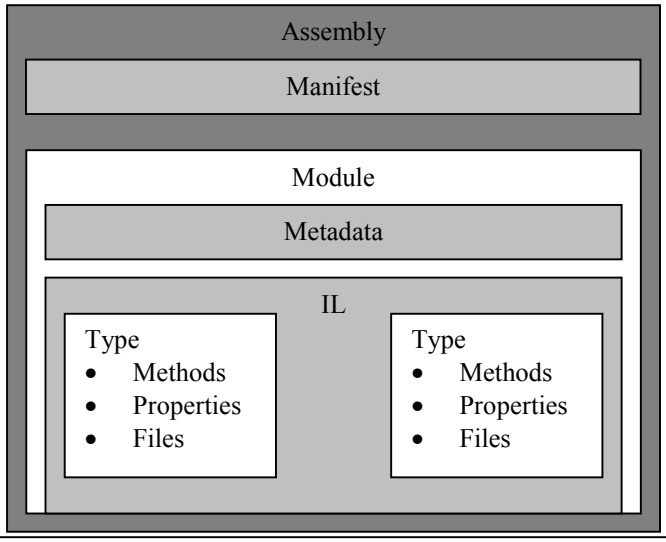
DOTNET Products beskriver alla Microsofts produkter som är tillgängliga och avsedda för att utveckla program med hjälp av XML och SOAP.
DOTNET Services möjliggör för utvecklare att skapa sina egna tjänster genom att använda .NET plattformen.
Den delen som hanterar komponenter i denna teknologi är .NET Framework som består av fyra delar. I denna del finns .NET Active Server Page (ASP) som är en utökning av det gamla ASP med många fördelar. .NET's ASP största fördelen över det gamla är möjligheten till språkoberoende programmering och dynamik webbapplikationer som kan köras på vilken plattform som helst. Detta mål erbjuder Microsoft i sin teknologi genom att använda XML-språket och SOAP-protokollet. Det vill säga det är inte nödvändigt längre att skriva kod i ett skriptspråk eftersom .NET ASP hanterar även vanliga språk. Med Windows forms som också ligger i Framework hanteras de grafiska delar av ett program. Framework Classes är en samling av fördefinierade klasser, interface, strukturer och typer som samarbetar med CLR (Commn Language Runtime) för att hantera olika datatyper. CLR är kärnan i .NET Framework och exekveringsmaskin för .NET applikationer. Nyckeln i .NET är den som kallad managed kod (koden som har skrivits enligt .NET typ definition). CLR hanterar en del saker bland annat:

- Minneshantering, CLR erbjuder Garbage Collector(GC) som hanterar allokering av minne i en applikation.

* Bilden avbildad från referens [DOTNET1]

- Kompilering och exekvering av kod sker i två steg som hanteras av CLR.
- Säkerhet och felhantering, CLR erbjuder en del funktioner för hantering av exception och säkerhet.
- Access till metadata, det skapas en del kod av CLR som specificerar olika parameter, funktioner och komponenter i ett program.

En .NET applikation skiljer sig lite från COM applikationen. I .NET en applikation innehåller en eller flera assembly, figur 5.2*.



Figur 5.2 Visar hur en applikation kan se ut I .NET, [DOTNET2]*

En assembly är en självbeskrivande och återanvändbar enhet som består av:

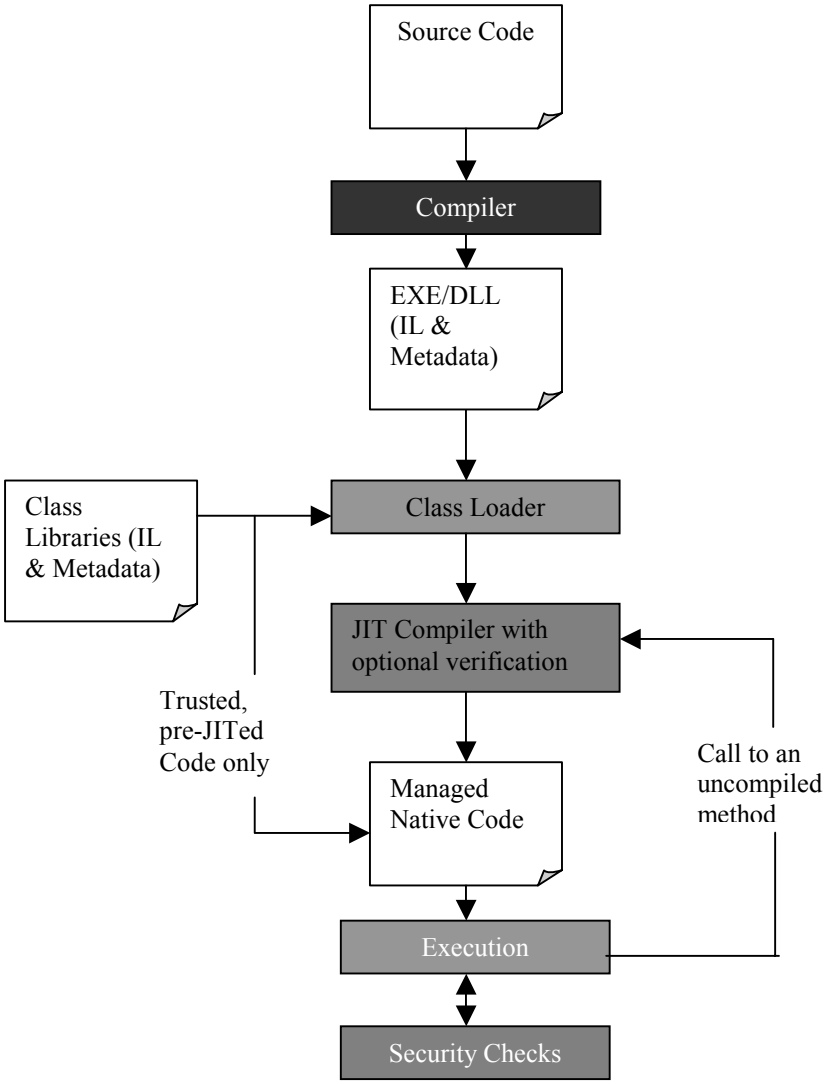
- Manifest som beskriver assemblys identitet.
- Module, är DLL eller EXE filer resurs filer t.ex. bilder.
- Metadata, ger information om olika delar i assembly och gör det självbeskrivande.
- Intermediate Language (IL), det är ett CPU-oberoende språk mellan källkod och maskinkod avsedd för .NET applikationer.

För att hantera språkoberoende och snabba exekvering av programmet kompileringen sker i två steg, först kompileras koden från källkod till IL och alla kompilatorer måste följa .NET specifikation för att kunna kompilera en källkod till IL. Andra steget utförs av CLR med hjälp av JIT-kompilatorer som är konstruerad i CLR och kompilerar från IL till maskinkod. Figuren 5.3 visar denna proceduren.
Microsoft tillsammans med denna teknologi presenterar ett nytt språk för sin teknologi som kallas C# (C sharp). Alla klasser i .Net är skriven som ett helt objektorienteradespråk. C# syntax liknar C++ men det samlar de mesta fördelar som finns i JAVA, Visual Basic och C++.

De saker som har blivit bättre i .NET jämför med MSCOM är bland annat minneshantering och komponentregistrering. .NET ramverk tar hand om referenser till objekten, med andra ord en klient behöver inte ta hand om referensuppräkning. Komponentregistrering har förbättras på grund av att .NET ramverk använder XML baserade konfigurationsfil som beskriver hur och var komponent är registrerad.

* Bilden avbildad från referens [DOTNET2]

Figur. 5.3*



Figur5.3. Visar hur ett program kommer att exekveras I .NET, [ART5] *

* Bilden avbildad från referens [ART5].

6. Appendix B: Ord lista

Active Server Pages (ASP)
: Web sidor innehållande skriptkod, utvecklat med start av Microsoft Internet Information Server 3.0 (IIS). Denna server-baserade körningen av miljö kör ett skript eller komponenter på en server. De Web sidor som exekveras på webbserver.

ActiveX
: En teknologi som möjliggör att komponenter kommunicerar med varandra via COM.

ActiveX Control
: Komponenter som har användaregränssnitt baserad på ActiveX teknologi

ActiveX Data Objects (ADO)
: Microsofts dataaccess programinterface, ADO är COM objekt, se även COM.

ANSI
: American National Standard Institute

Application Programming Interface (API)
: En mängd of funktioner ändvändas för att kommunicerar med komponent, applikation och OS.

ASCII
: American Standard Code for Information Interchange

Client/Server
: En arbetsmodell är klienten (arbetsstation) begär information från servern, och servern lämnar ut den information som begärts. Ett lokalt nätverk (LAN) eller ett globalt nätverk (WAN) ansluter både klienten och server eller den som begär tjänst är klient och den som svarar är server.

Common Object Request Broker Architecture (CORBA)
: En objektgrupp för att kommunicera mellan distribuerade objekt. Med CORBA kan program skrivna i alla olika språk köras överallt från ett nätverk och från valfri plattform.

Compiler
: Översätter programkoden till en körbar fil

Data Access Objects (DAO)
: Ett av Microsofts Program interface som används för dataaccess.

Database Managment System (DBMS)

Ett mjukvara interface som tar hand om användarens förfrågningar för databas funktioner, kontroller, säkerhet, integritet, SQL, editering, uppdatering och oberoende av data i databasen.

Debug

Känner av, lokaliserar och rättar logiska fel i programmets kod.

Distributed Component Object Model (DCOM)

Microsofts motsvarighet till CORBA, DCOM egenskaper som arbetar över nätverk.

Distributed processing

Reducerar antalet steg över ett nätverk. Varje hopp från klienten till servern gör att applikationen tar mer tid på sig för att utföra operationen. Genom att utföra mer arbete på klientsidan, kommer applikationen att svara snabbare mot användarens förfrågningar.

Dynamic Data Exchange (DDE)

En mekanism som supportas av Windows som tillåter två applikationer att utbyta data kontinuerligt och automatiskt

DHTML

Dynamic HTML. Med hjälp av DHTML kan ha flexibla HTML sidor.

Dynamic Link Library (DLL)

Tillåter dig att binda kod till en eller flera exekverbara program under drift i stället för att länka vid kompilering, genom att skapa en miljö som kan ha samma kod i basen.

Metadata

Data som beskriver annan data.

Object Model

Den strukturella formen för ett objektorienterat programmeringsspråk, design och applikation. Innehållande ett objekts arkitektur samt detaljer om hur ett objekt ser ut och arbetar mot andra objekt som finns.

Object-Oriented Database Managment System (OODBMS)

Ett underhållande databassystem som behandlar objekt (delar av kod eller data). Användbart för data med komplexa relationer som DBMS skulle ha svårt att klara av. OODBMS kan arbeta med multimedia data typer.

Object-Oriented Programming (OOP)

En programmeringsteknik som tillåter dig att arbeta med grupp av datastrukturer och funktioner som kan arbeta tillsammans med andra objekt. Inkapsling av data och ärv är huvud nyckelord i detta teknik.

OLE Automation
: Skapar programmerbara objekt och applikationer som andra program kan kalla på. Detta är första steget för att skapa återanvändbara procedurer och funktioner.

ODBC
: Open Database Connectivity

Property
: En egenskap hos ett visst objekt

Query
: Frågor som skickas till databas.

Relational Database
: En databas eller databassystem (RDBMS) som organiserar data och relationer mellan dem. Relationsdatabaser tillåter definitioner, lagring och frågor. Vissa fält kan vara designade som nycklar, vilket gör att olika tabeller kan sättas i relation till varandra.

Remote Data Objects (RDO)
: Microsofts programinterface för dataaccess och används av VB för att komma åt remote -ODBC data.

Run-Time
: Förklarar tillståndet av händelser som körs när ett program arbetar. Motsatsen till detta är design-time.

Server
: En dator i ett nätverk som innehåller begärd information (såsom webbsidor, filer och andra tjänster) till klienten.

SQL
: Structured Query Language

Table
: En logisk gruppering av relaterad information som är arrangerad i rader och kolumner, på liknande sätt som ett kalkylblad .

UML
: Unified Modeling Language. Ett objektorienterat designspråk.

UI
: User Interface. En del av ett program som visas för användaren, innehållande menyer, former, knappar mm. Windows har ett grafiskt user interface (GUI), jämfört med DOS som har command-line interface.

Variant

En speciell datatyp som kan innehålla nummer, strängar, eller datum samt Null eller Empty. Variant är det normala datatypen för VB. Det är så alla variabler är om deras typer inte deklareras.

7. Referenser

Books

[COM1] Dale Rogerson, 1996, *Inside COM (Programming Series)*

[COM2] Don Box, 1998, *Essential COM*

[COM3] Steven Gray, Rick Lievano, Roger Jennings, 1999, *Microsoft Transaction Server 2.0*

[COM4] Dr Richard Grimes, Alex Stockton,1998, *Beginning ATL COM Programming*

[DOTNET1] David S. Platt, 2001, *Introducing Microsoft® .NET*

[DOTNET2] James Conard, 2001, *Introducing .NET*

[CSHARP] Tom Archer, 2001, *Inside C#*

Internet

[ART1] Burton Harvey, M.S., MCSD,
Sharpen Up on C#,
<http://www.perfectxml.com/Conf/Wrox/Files/burt_csharp.pdf>,
2004-03-08

[ART2] Jeffrey Richter,
Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework,
<http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/1100/gci/toc.asp>,
2004-03-08

[ART3] Microsoft msdn,
.NET Framework Developer's Guid,
<http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcomwrappers.asp>,
2004-03-08

[ART4] Bob Muglia,
Q&A: For Developers, Microsoft Group VP Muglia Says Microsoft is Delivering on .NET Now,
<http://www.microsoft.com/Presspass/features/2001/oct01/10-23mugliaqa.asp>,
2004-03-08

[ART5] Jeffrey Richter,
Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web,
<http://msdn.microsoft.com/msdnmag/issues/0900/framework/default.aspx>,
2004-03-08

[ART6] Aravind Corera,
Understanding Classic COM Interoperability With .NET Applications,
<http://www.codeproject.com/dotnet/cominterop.asp>,
2004-03-08

[ART7] Devigus Engineering AG,
Visual Studio.NET XML nTtier Application Framework,
http://www.devigus.com/files/DotnetTechnologyBackgrounder.pdf
2004-03-08

[ART8] N-Tier.com,
N-tier Background Articles Index,what is n-tier computing,
<http://n-tier.com/articles.html>,
2004-03-08

[ART9] Mark T. Hoske,
DNA for Manufacturing Microsoft Creates Framework for Software Tools,
<http://www.manufacturing.net/ctl/article/CA194151>,
2004-03-08