Datavetenskap

Erik Nordström och Ola Noreklint

Nya möjligheter med .NET

– en utredning av skillnaderna mellan DCOM- och webbtjänsttekniken

---

Examensarbete, C-nivå

Första utgåvan

Nya möjligheter med .NET

– en utredning av skillnaderna mellan DCOM- och webbtjänsttekniken

Erik Nordström och Ola Noreklint

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Erik Nordström

Ola Noreklint

Godkänd 2002-06-13

Handledare: Tim Heyer

Examinator: Tim Heyer

Sammanfattning

Det här är ett examensarbete på C-nivå i datavetenskap som jämför de tekniska aspekterna av *Distributed Component Object Model (DCOM)* och webbtjänster. Detta görs på uppdrag av Pronyx Sweden AB i Karlstad, för att undersöka möjligheten att börja använda webbtjänstteknik för att ersätta eller komplettera DCOM-tekniken, som används idag. För att få ut så mycket som möjligt av rapporten krävs grundläggande kunskaper om datakommunikation och Internet. Rapporten täcker några av de grundläggande begreppen kring de olika teknikerna för att möjliggöra en jämförelse med avseende på vilka för- och nackdelar användningen av respektive teknik medför. Detta innebär att områden som skalbarhet, anropssätt och egenskaper hos serverapplikationer diskuteras.

I rapporten presenteras även en exempelimplementation för att möjliggöra prestandamätningar såsom aktiveringstid, anropstid och minnesåtgång. Resultatet av jämförelsen blev att webbtjänsttekniken i viss grad rekommenderades för Pronyx Sweden AB. För att begränsa utredningen har vi inte betraktat den viktiga säkerhetsaspekten som man ofta gör när man behandlar fallet med Internet.

New possibilities with .NET

– an inquiry of the differences between DCOM- and Web Service technology

Abstract

This is a Bachelors project in computer science, comparing the technical aspects of the *Distributed Component Object Model (DCOM)* and Web Services. This project is commissioned by Pronyx Sweden AB in Karlstad, to investigate the possibilities to start using the Web Service technology for replacing or complementing the DCOM technology, witch is used today. To understand as much as possible of this report the reader has to have basic knowledge of computer communication and the Internet. The report at hands covers some of the basics concepts of the different technologies to enable a comparison with respect to the advantages and disadvantages usage of the techniques will bring about respectively. This means discussing areas as scalability, ways to call a server application, and server application properties.

In the report a presentation of an implementation is given to enable measurements of performance as activation time, calling time and memory consumption. As a result of the comparison, the Web Service technology was to a certain extent recommended for Pronyx Sweden AB. To limit the inquiry we did not look at the important aspect of security, which is often done when dealing with the Internet.

Tack till

Vi vill tacka vår handledare på Pronyx Sweden AB i Karlstad, Michael Olsson, för att han inspirerat och hjälpt oss i vårt arbete och också för att han lärt oss se ljuset i den mörka tunneln. Han har även varit en stor informationskälla och lagt grunden för vårat arbete.

Vi vill också tacka vår handledare på Karlstads universitet, Tim Heyer, för en mycket noggrann och proffsig handledning och rättning av denna rapport.

Till sist vill vi tacka alla på Pronyx Sweden AB i Karlstad för ett vänligt bemötande och trivsamma stunder i fikarummet.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Idag är DCOM-tekniken ett av de vanligare sätten för olika applikationer i *Windows* miljö att kommunicera med varandra i distribuerade miljöer, något som kan ske även över Internet. Tekniken används först och främst av *Microsofts* operativsystem *Windows* som dominerar marknaden för persondatorer. *Microsoft* har också utvecklat DCOM-tekniken.

Det finns dock brister med denna teknik. För att applikationer på en plattform ska kunna kommunicera med applikationer på en annan plattform måste båda plattformarna ha stöd för DCOM, vilket inte gäller alla plattformar idag. Detta får som följd att DCOM-applikationer inte alltid kan nå varandra för kommunikation.

Dessa brister gör att webbtjänsttekniken blir intressant. En webbtjänst kan ses som en applikation tillgänglig över Internet. Med webbtjänster försvinner kravet om smarta plattformar eftersom kommunikationen mellan applikationer kan ske över det standardiserade Internetprotokollet *Hyper Text Transfer Protocol (HTTP)* istället. Enda kravet är att plattformen har kontakt med Internet och stöd för HTTP.

En annan stor fördel med webbtjänster är det enkla sättet på vilket kommunikationen sker. För att anropa en webbtjänst krävs endast ett textbaserat meddelande som är läsbart även för programutvecklaren. Detta innebär att man i princip kan skapa ett anrop manuellt och sända det till webbtjänsten utan att ha tillgång till någon speciell utvecklingsmiljö.

För att ta del av webbtjänsternas fördelar, och därmed gå ifrån DCOM-tekniken, finns det dock vissa saker som bör tänkas på. Webbtjänstens egenskaper skiljer sig en hel del från DCOM-applikationens. Skillnaden mellan sättet att anropa en webbtjänst och en DCOM-applikation är många och det är inte helt trivialt att byta teknik.

För att bilda en uppfattning om vad ett byte skulle innebära och vilka för- och nackdelar som finns har vi i denna rapport gjort en utredning av skillnaderna mellan DCOM- och webbtjänsttekniken.

2 Bakgrund

Microsoft har släppt en ny utvecklingsplattform *Visual Studio .NET* som innebär nya möjligheter när det gäller programvaruutveckling i *Windows-miljö*. Pronyx Sweden AB i Karlstad är ett företag som tillverkar datorsystem för att planera, optimera och styra industriell produktion och har ett intresse av att undersöka denna utvecklingsplattform. Företaget använder sig idag av *Microsoft Visual Studio 6.0* som utvecklingsplattform men har planer på att använda *Microsoft Visual Studio .NET* i framtiden.

En nyhet med *Microsoft Visual Studio .NET* är att kommunikationen mellan applikationer kan ske med protokollet HTTP. Detta innebär att en applikation gjord i *Microsoft Visual Studio .NET* kan använda HTTP-protokollet när den anropar en annan applikationen, vilket på ett enkelt sätt möjliggör att kommunikationen kan ske över Internet. I *Microsoft Visual Studio 6.0* sköts kommunikationen med DCOM vilken antagligen är den mest utbredda tekniken för kommunikation mellan distribuerade applikationer idag. När det gäller kommunikation mellan applikationer kallas den applikation som tillhandahåller en tjänst för serverapplikation och den som utnyttjar tjänsten för klientapplikation. Med komponent menas i denna rapport något exekverbart som kan sättas ihop med andra komponenter för att utföra en uppgift. En serverapplikation är därför en komponent. Figur 1 och 2 visar skillnaden mellan *Microsoft Visual Studio 6.0* och *Microsoft Visual Studio .NET*.

Figur 1: Microsoft Visual Studio 6.0: Kommunikationen mellan klientapplikationen och serverapplikationen sker med DCOM.

Figur 2: Microsoft Visual Studio .NET: Klientapplikationen kan använda HTTP och Internet för att kommunicera med serverapplikationen.

Kommunikationen med HTTP på detta sätt innebär alltså att applikationer kan anropa varandra över Internet. En serverapplikation tillgänglig över Internet kallas för en webbtjänst. I *Microsoft Visual Studio .NET* finns ett väl utvecklat stöd för att konstruera webbtjänster och klientapplikationer för att anropa webbtjänsterna, något som är helt nytt gentemot *Microsoft Visual Studio 6.0*.

Det som Pronyx Sweden AB i dagsläget anser vara mest intressant med *Microsoft Visual Studio .NET* är möjligheten att på ett enkelt sätt konstruera och använda webbtjänster. Att implementera serverapplikationer som webbtjänster skulle i framtiden helt, eller i alla fall delvis, kunna ersätta utvecklingen av de DCOM-applikationer som företaget använder sig av idag. Bortsett från de säkerhetsproblem som finns med Internet idag är detta en mycket attraktiv lösning eftersom det skulle innebära stora rationaliseringar när det gäller leveransen av system. Stora delar av systemen skulle kunna ligga på företagets egna servrar och kommas åt av små installerade klientapplikationer hos kunden via Internet. En stor del av de ändringar och uppgraderingar som idag görs ute hos kunden skulle då kunna göras på hemmaplan istället. Bl.a. skulle detta troligtvis innebära att den tid personalen spenderar ute hos kund minskade. Att på detta sätt samla funktionaliteten till samma plats skulle också innebära en bättre översikt av de komponenter som finns vilket i sin tur ökar möjligheten till återanvändning av dessa, något som idag görs i liten skala. Ett problem med att introducera webbtjänster idag är att alla kunder inte har tillgång till Internet. Dock finns anledning att tro på en lösning inom en snar framtid, speciellt om kunder informeras om de möjligheterna som finns.

I denna rapport kommer vi att presentera tekniken runt webbtjänster i jämförelse med DCOM-tekniken. Denna jämförelse kommer att bestå av två delar: dels en teknisk jämförelse där de grundläggande begreppen tas upp (se kapitel 3) och dels en implementation där både DCOM- och webbtjänsttekniken illustreras (se kapitel 4 och 5).

Målet är att kunna dra slutsatser om konsekvenserna kring ett införande av webbtjänsttekniken för Pronyx Sweden AB, vilka för- och nackdelar som finns och, om ett införande anses aktuellt, när detta skulle kunna ske.

3 Teknisk jämförelse

I detta kapitel, som är den första delen av den tekniska jämförelsen, kommer vi att ge en förklaring av de grundläggande begreppen kring DCOM-tekniken (se kapitel 3.1) och webbtjänsttekniken (se kapitel 3.2) för att sedan kunna visa hur ett motsvarande anrop kan fungera. Kapitlet avslutas med en teknisk jämförelse mellan de båda teknikerna.

3.1 DCOM-teknik

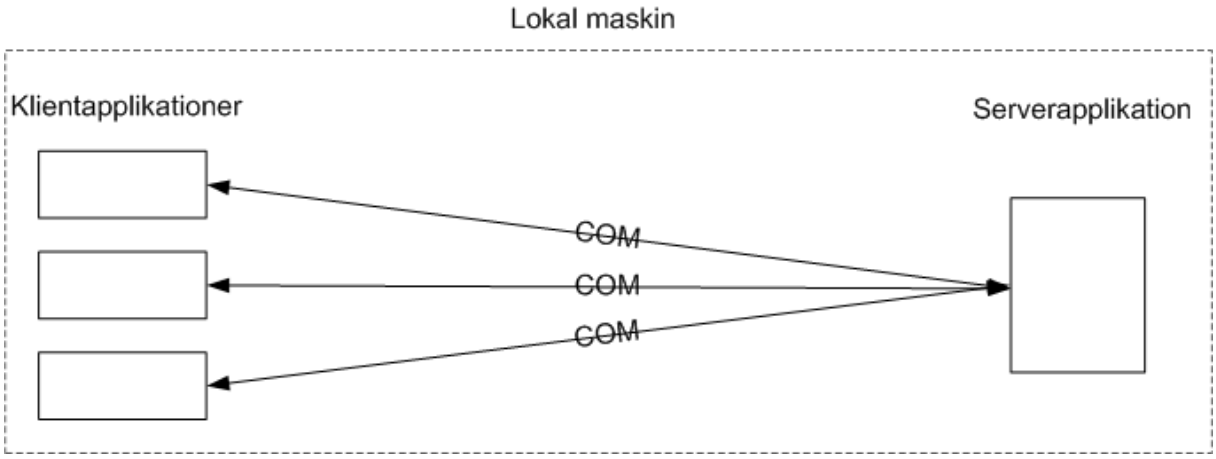
Distributed Component Object Model (DCOM) är ett protokoll som tillåter applikationer att kommunicera direkt över ett nätverk på ett pålitligt, säkert och effektivt sätt. DCOM bygger på *Component Object Model (COM)* som definierar hur olika applikationer interagerar med varandra. Skillnaden mellan COM och DCOM är att DCOM sköter kommunikationen när applikationerna finns på olika datorer i ett nätverk och COM används när applikationerna finns på samma dator. Notabelt är att detta är transparent för applikationer vilket innebär att en applikation inte behöver skilja på om COM eller DCOM används. Eftersom DCOM endast utökar COM kan det förklaras utifrån en förklaring av COM. Kapitlet utgår därför ifrån en förklaring av COM. Mer om COM och DCOM finns i [1], [2] och [3].

3.1.1 Bakgrund och förklaring

COM växte fram ur O*bject Linking and Embedding (OLE)* som släpptes av *Microsoft* för första gången 1992. Första versionen av OLE hade som uppgift att tillhandahålla en förbättrad mekanism för sammansatta dokument, d.v.s. dokument med innehåll från olika applikationer, i Windows. Detta innebar t.ex. att ett inklistrat objekt i Word från Excel uppdaterades automatiskt om objektet ändrades i Excel. Det blev också möjligt att redigera Excel-objektet direkt från Word. När andra versionen av OLE kom 1993 var tekniken förbättrad och möjligheten fanns för applikationer att anta samma skepnad som andra applikationer och därmed tillåta redigering av ett sammansatt dokument från en och samma applikation. Denna teknik var komponentbaserad och första steget mot COM. I och med *Windows 95* var COM-tekniken implementerad och komponentmodellen ett vedertaget begrepp. DCOM utvecklades

senare från COM och *Remote Procedure Call (RPC)*[1] och fanns tillgängligt för första gången 1996 i och med *Windows NT 4.0*. Mer om detta finns i [4].

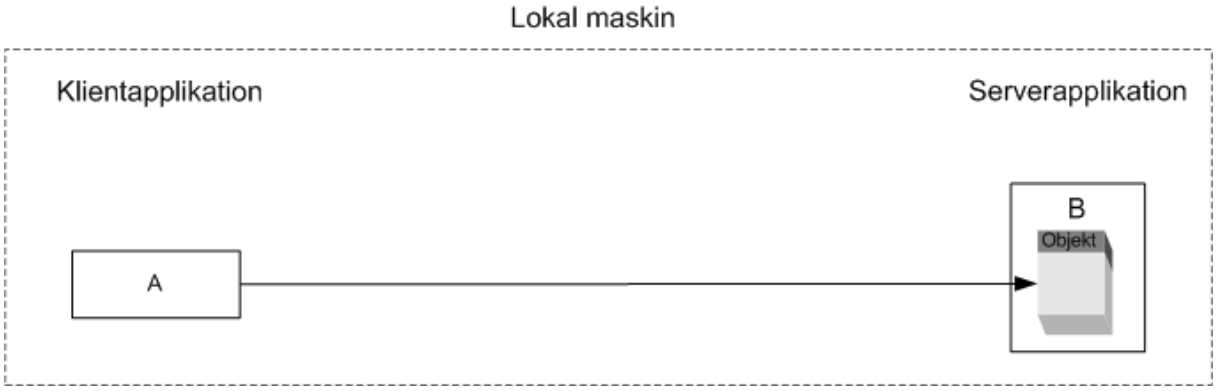
Ett av de vanligare sätten att undvika redundant information och samla funktionalitet idag är att använda sig av COM. Följande figur beskriver idén bakom detta:



Figur 3: En serverapplikation innehåller den samlade funktionaliteten och tillhandhåller denna funktionalitet som en tjänst för klientapplikationerna. Kommunikationen mellan klientapplikation och serverapplikation sköts av COM.

COM är ett allmänt och standardiserat sätt för applikationer att kommunicera med varandra och, som namnet antyder, används komponenter i denna kommunikation. En serverapplikation, som den i figur 3 ovan, är en komponent eftersom den kan sättas ihop med, d.v.s. användas av, andra applikationer för att utföra någon uppgift. Serverapplikationen, som tillhandhåller en tjänst, kan med COM göra detta genom objekt. Ett objekt är en komponent som kan skapas av en annan komponent, där den skapande komponenten är ett klassobjekt. Ett objekt är också en instans från en klass, som är mall för objektet. En klass har metoder och egenskaper. Detta innebär att det på serverapplikationen finns klasser från vilka klassobjekt kan skapa objekt. Dessa objekt kan sedan användas av klientapplikationen. Notabelt är att applikationer kan fungera både som server- och klientapplikationer genom att samtidigt tillhandahålla tjänster och använda sig av andra applikationers tjänster. Anta att vi har två applikationer A och B, där B har ett objekt som är tillgängligt för A enligt följande figur:

[1] RPC är en tjänst från *Microsoft* för att utföra distribuerade metodanrop.



Figur 4: Klientapplikation A kan använda tjänsten som serverapplikationen B tillhandahåller genom ett objekt.

Detta upplägg medför bl.a. följande positiva egenskaper:

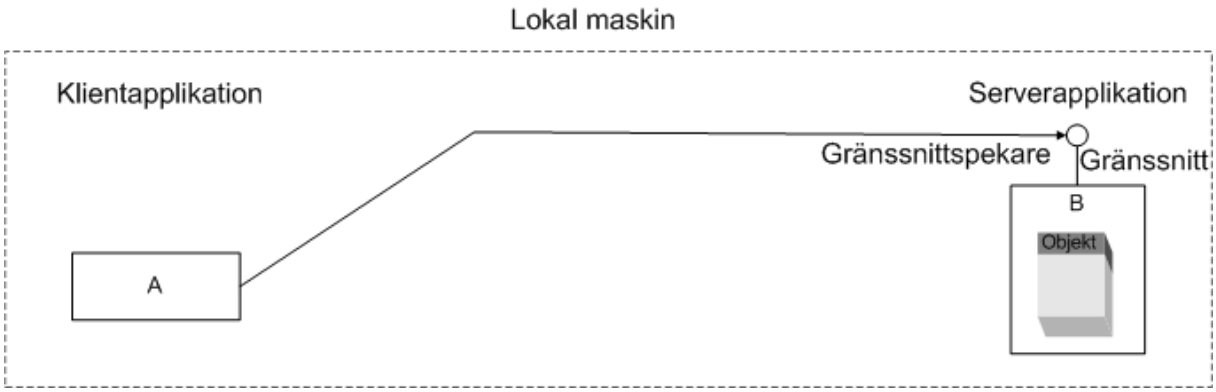
- En mycket enkel abstrakt förklaring på vad som händer: Applikation A kontaktar applikation B för att komma åt det aktuella objektet. B gör objektet tillgängligt för A så att A kan använda det, d.v.s. anropa metoderna eller använda egenskaperna.
- Objektet i applikation B kan enkelt återanvändas i andra sammanhang. Om applikationerna skulle köras på två olika maskiner skulle samma upplägg vara möjligt. (Protokollet skulle då benämnas DCOM).
- Att arbeta med objekt är naturligt och enkelt och stöds av de flesta programspråk.

3.1.2 Teknisk redogörelse

Nedan följer en förklaring av viktiga begrepp när det gäller COM-tekniken.

3.1.2.1 Gränssnittspekare

Kommunikationen mellan klientapplikation A och serverapplikation B enligt figur 4 ovan möjliggörs i COM genom en pekare (referens) som A har. Denna pekare kallas gränssnittspekare och kan användas på samma sätt som en vanlig pekare till ett objekt. Klientapplikationen använder gränssnittspekaren för att komma åt metoderna i objektet i B. Genom COM tas den underliggande kommunikationen om hand. Följande figur visar gränssnittspekaren i förhållande till applikationerna:



Figur 5: Utvecklar föregående bild med gränssnittspekare och gränssnitt. Klientapplikationen har en gränssnittspekare som är en referens till ett gränssnitt (se nedan) på objektet som finns i serverapplikationen.

3.1.2.2 Gränssnitt

En gränssnittspekare är en referens till ett gränssnitt. Ett gränssnitt definierar hur klientapplikationen ser objektinstansen skapad från någon klass i serverapplikationen. I det här fallet vilka metoder klientapplikation A kan anropa via det givna objektet i serverapplikation B. Rent konkret är detta mängden av objektets metoder som visas för klienten. Detta innebär att klientapplikationen vet metodernas namn, eventuella parametrar och ett eventuellt returvärde.

Ett problem är dock objektets tillförlitlighet, för även om metoderna och parametrarna har namn som verkar ha en logisk innebörd är detta ingen garanti för att så är fallet. Metoden kan i princip returnera vilket värde som helst bara det är av den typen som gränssnittet anger.

För att lösa detta problem används ett kontrakt mellan klientapplikationen och objektet. Ett kontrakt definierar vad som händer med objektet när metoderna anropas och kan inte implementeras i kod utan måste dokumenteras utanför källkoden. Kontrakt är viktigt vid utbyte och återanvändning av objekt för att veta den exakta funktionaliteten man kan förvänta sig av objektet. Utan kontrakt vore återanvändning av serverapplikationer i praktiken omöjlig eftersom klientapplikationer inte skulle kunna vara säkra på vad objektens metoder vara avsedda att göra. Detta skulle i så fall innebära att hela idén med COM föll. Kontrakt som finns måste hållas eftersom det är enda sättet för klientapplikationen att förstå hur objektets metoder är menade att fungera. Klientapplikationen varken får eller behöver se hur objektet är implementerat. För objektet innebär detta att metoderna kan implementeras på vilket sätt som helst så länge kontraktet upprätthålls.

Ett gränssnitt är namngivet på två olika sätt: dels ett 128-bitars nummer, oftast angivet hexadecimalt, s.k. *Globally Unique Identifiers (GUID)* och dels med en textsträng alltid med prefixet ”I” för ”Interface” som är det engelska ordet för gränssnitt. Ett gränssnitt, *Iexempel*, skulle kunna ha GUID: *ecabb0bd-7f19-11d2-978e-0000f8757e2a*. Anledningen till att ha två olika namn är att gränssnittet måste kunna unikt identifieras på något sätt men ändå ha ett namn med en logisk innebörd. GUID garanterar unik identifiering och textsträngen sätts lämpligen till ett namn som på något sätt förklarar funktionaliteten hos de metoder som implementerar gränssnittet.

3.1.2.3 Referensräknare

Referensräknare är ett sätt för objektet att hålla reda på när det används av klientapplikationer. När klientapplikationen A kontaktar serverapplikationen B för att få tillgång till ett objekt försöker B skapa objektet. Om det lyckas kan B ge A tillgång till det genom COM och gränssnittspekare. Om det misslyckas kommer COM meddela detta till A genom att returnera *NULL*.

Om objektet kan skapas är det klientapplikationens uppgift att öka referensräknaren som finns i objektet. Detta görs i samma anrop som när objektet skapas genom en metod *Addref()* som alla COM-objekt implementerar och som klientapplikationen kan komma åt genom ett standardgränssnitt vilket beskrivs nedan. Nu finns information om att objektet används. Om en annan applikation (A*) skulle kontakta B och få tillgång till samma objekt skulle återigen en ökning behöva göras av referensräknaren. Detta skulle då innebära att referensräknarens värde skulle vara två. Efter att klientapplikationerna är klara med objektet åligger det dem också att minska referensräknaren. Detta görs genom *Release()* som liksom *Addref()* är en metod vilken implementeras av standardgränssnittet. När referensräknarens värde är noll vet objektet att det kan förstöra sig själv, vilket det gör, och B meddelas om detta och kan avsluta.

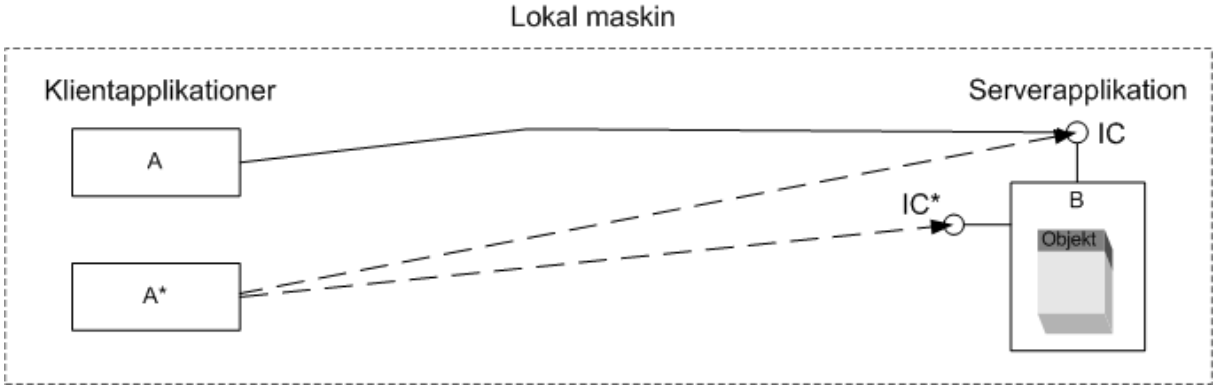
Viktigt är att referensräkningen sker på antal referenser till gränssnitt för objektet och inte för antal klientapplikationer som har tillgång till det. Detta innebär att om en klientapplikation har gränssnittspekare till två gränssnitt på ett objekt samtidigt så kommer referensräknaren vara två. Om ytterligare en klientapplikation får referens till ett gränssnitt på objektet kommer referensräknaren vara tre. *Release()* ska göras när klientapplikationen är klar med ett gränssnitt.

Ett problem med referensräkning är att hanteringen sköts av klientapplikationen. Om klientapplikationen av någon anledning inte använder referensräknaren eller glömmer *Addref()* eller *Release()* vid något tillfälle kan felaktigheter uppstå. Exempel på sådana

felaktigheter är att objektet förstörs innan alla klientapplikationer har avslutat eller att objektet finns kvar utan att det används. Det senare skulle innebära minnesläckage.

3.1.2.4 Förfrågan om gränssnitt (”Interface Querying”)

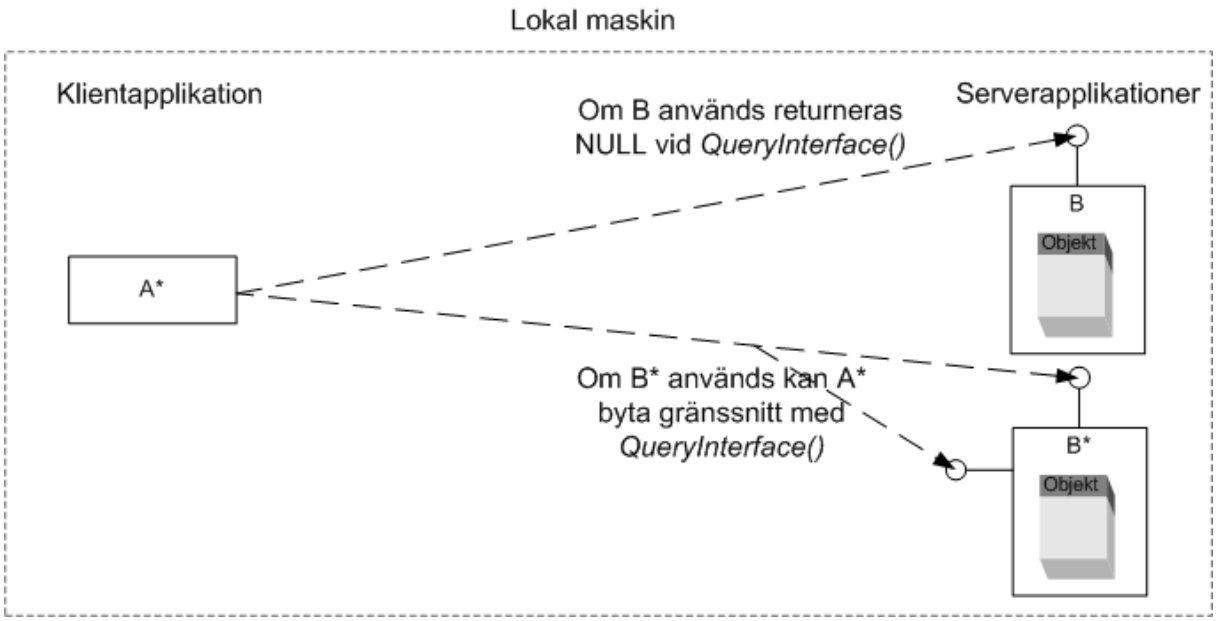
Eftersom ett objekt kan exponera flera gränssnitt måste det också ha möjlighet att ge klientapplikationen just det eller de gränssnitt som önskas. Detta görs genom ”Interface querying” som liksom *Addref()* och *Release()* är en funktionalitet tillhandahållen genom standardgränssnittet beskrivet nedan. Genom denna funktionaliteten blir gränssnittsbytet dynamiskt, d.v.s. möjligt under körning. En fördel med detta upplägg exemplifieras enligt följande: Anta till en början en klientapplikation A och en serverapplikation B. A använder sig av det objekt som B tillhandahåller med hjälp av COM. Objektet implementerar ett gränssnitt IC som A har referens till genom en gränssnittspekare. Anta nu att objektet i B behöver utökas med fler metoder för att kunna användas av en nyare klientapplikation A*, som utnyttjar både de gamla och de nya metoderna. Att utöka gränssnittet IC för att A* ska kunna använda detta är inte möjligt eftersom gränssnittet mot A då skulle förstöras. Istället skapas ett nytt gränssnitt IC* som innehåller de nya metoderna. A kan nu fortsätta använda objektet som vanligt och A* byter mellan gränssnitten med hjälp av *QueryInterface()*, metoden som implementerar ”Interface querying”. Tack vare byte av gränssnitt under körning kan serverapplikationen användas av både A och A* och även utökas efter framtida behov. Detta exempel illustreras i följande figur:



Figur 6: Exempel 1 med ”Interface querying” illustrerar dynamiskt gränssnittsbyte.

En annan fördel med dynamiskt gränssnittsbyte är att klientapplikationen kan uppnå ett visst versionsoberoende från serverapplikationen enligt följande: Anta två versioner av serverapplikationen B. Den gamla, B, och den nya, B*, där B* utökar B med nya metoder och

ett nytt gränssnitt. Betrakta nu klientapplikation A* enligt exemplet ovan. A* gör ett anrop till B, får en referens till standardgränssnittet och har nu möjligheten att anropa *QueryInterface()* för att byta gränssnitt. A* kan nu vara implementerad för att vara kompatibel mot båda versionerna av B, d.v.s. både B och B*, beroende på vilken av versionerna objektet är en instans ifrån. Detta eftersom *QueryInterface()* returnerar *NULL* om det efterfrågade gränssnittet inte finns, vilket kan tas om hand, och man slipper då fel under körningen. Detta innebär att A* kan anpassa sig till en äldre version av serverapplikationen om så behövs. Detta exempel illustreras i följande figur:



Figur 7: Exempel 2 med ”Interface querying” illustrerar versionsoberoende.

QueryInterface() ansvarar också för att *addRef()* görs. Detta innebär en ökning av referensräknaren varje gång *QueryInterface()* anropas. *Release()* måste dock göras separat av klientapplikationen när den är klar med gränssnittet.

3.1.2.5 IUnknown

Att kunna räkna referenser och ha möjlighet att göra förfrågan om gränssnitt under körning är nödvändigt för att få en effektiv kommunikation mellan klient– och serverapplikationer. Utan referensräkning skulle inte objektet veta när det är dags att förstöra sig själv och utan gränssnittsförfrågan skulle klientapplikationer inte alltid kunna ta del av all funktionalitet från serverapplikationen. Detta innebär att de tre standardmetoderna *Addref(), Release()* och *QueryInterface()* alltid måste finnas tillgängliga för klientapplikationen i det gränssnittet som

den för tillfället refererar till. Detta ger upphov till standardgränssnittet *IUnknown* som exponerar de tre metoderna ovan. *IUnknown* är vanligtvis (se kapitel 3.1.3.2) det första gränssnitt klientapplikationen får en referens till när ett objekt efterfrågas. Från detta gränssnitt används *QueryInterface()* för att komma åt de andra gränssnitten. Viktigt är dock att alla gränssnitt tillåter användandet av *Addref()*, *Release()* och *QueryInterface()*. För att lösa detta ärver alla gränssnitt från *IUnknown* vilket innebär att alla gränssnitt inkluderar standardmetoderna först av sina metoder.

3.1.2.6 IDL och Type Library

För att beskriva gränssnitt används *Interface Definition Language (IDL)*. En IDL-fil kan beskriva flera olika gränssnitt och kompileras till en binär fil, ett *Type Library*, som kan utgöra mall för hur serverapplikationer konstrueras. Detta innebär att de serverapplikationer som inkluderar ett *Type Library* och implementerar ett gränssnitt som finns beskrivet där måste göra det fullständigt. Om en serverapplikation endast implementerar några av ett gränssnitts metoder eller har ej överensstämmande namn ges kompileringsfel. En serverapplikation som inkluderat ett *Type Library* behöver dock inte implementera alla gränssnitt som finns där.

När en serverapplikation implementerat de aktuella gränssnitten från IDL-filen och kompilerats kommer samtidigt ett nytt *Type Library* skapas. Detta *Type Library* beskriver då de gränssnitt som finns implementerade i serverapplikationen och kan utnyttjas av klientapplikationen genom COM, vilket kallas tidig bindning och beskrivs i kapitel 3.1.2.7. På detta sätt kan klientapplikationen få vetskap om hur serverapplikationens gränssnitt ser ut utan att ha tillgång till själva applikationen. I IDL-filerna knyts ett GUID till varje gränssnitt vilket innebär att dessa GUID alltid kommer vara de samma för dessa gränssnitt.

3.1.2.7 Bindning

Som nämnts tidigare använder klientapplikationen GUID för att identifiera gränssnitt i serverapplikationen. Detta gäller även för klasser, som också är knutna till ett GUID. Ett GUID gör på detta sätt att COM vet vilken metod som ska anropas. Det finns två sätt för klientapplikationen att få reda på dessa GUID, genom tidig eller sen bindning.

- Tidig bindning: Klientapplikationen har en referens till serverapplikationens *Type Library* och läser av de GUID som är satta där. Detta innebär att klientapplikationen alltid använder sig av samma GUID för att anropa serverapplikationen. Det innebär också att klientapplikationen tappar bort serverapplikationen om de GUID som finns där byts ut, vilket exempelvis sker vid

kompilering. Detta är ett problem med tidig bindning som kan lösas genom att göra serverapplikationen binär kompatibel. Binär kompabilitet innebär att serverapplikationen behåller sina ursprungliga GUID trots att den kompileras om.

- Sen bindning: Varje gång klientapplikationen ska anropa serverapplikationen söks registret igenom på den lokala maskinen för att få tag på de GUID som är aktuella. Detta kan göras eftersom det i registret finns poster där de textliga namnen på klassen och gränssnittet binds till GUID. Klientapplikationen vet alltså de textliga namnen och COM översätter dessa till GUID. Fördelen med detta är att klientapplikationen är oberoende av om serverapplikationen kompileras om och får sina GUID utbytta eftersom COM då uppdaterar registret och klientapplikationen hittar rätt ändå. En nackdel med sen bindning är det ineffektiva tillvägagångssättet med att leta i registret varje gång. Detta gör att metodanropen tar längre tid. En annan nackdel med detta dynamiska tillvägagångssätt är att vissa fel inte kan upptäckas under kompileringen utan inträffar under körningen vilket kan leda till att applikationen kraschar.

Mer om bindning finns i [6].

3.1.2.8 Marshaling

Tekniskt innebär fjärranrop en del problem eftersom det inte är meningsfullt att skicka pekare mellan olika maskiner. Detta beror på att pekare är minnesadresser och eftersom olika minnesutrymmen används kommer en adress i ett visst minnesutrymme inte ha någon relevans i ett annat. Följden blir att när en klientapplikation tillhandahåller en gränssnittspekare från en serverapplikation så är detta egentligen inte fallet. Istället finns ett objekt i klientapplikationen (proxy-objekt) som simulerar det riktiga objektet och en klient i serverapplikationen (stub) som simulerar den riktiga klienten. Proxy-objektet representerar det riktiga objektet för klientapplikationen genom att exponera exakt samma gränssnitt och genom att veta hur kommunikationen med serverapplikationens stub går till. Stubben representerar å andra sidan klienten genom att använda sig av samma gränssnitt och veta hur kommunikationen med proxy-objektet går till. Exakt hur kommunikationen mellan stub och proxy-objektet går till beskrivs inte av COM utan kan implementeras fritt. På detta sätt uppnås protokolloberoende vilket är en mycket viktigt aspekt av COM. Rekommenderad implementation från *Microsoft* finns dock och benämns *standard marshaling*. När s*tandard marshaling* används kommer både klientapplikation och serverapplikationen ladda in ett bibliotek (komponent), *proxy*/*stub DLL*, som tar hand all *marshaling*. *Marshaling* för

klientapplikationen blir enligt detta resonemang processen att paketera metodanrop och parametrar och sända paketet till komponenten. Det finns tre olika typer av *marshaling*:

1. *Inter-thread marshaling*. Anrop till ett objekt som är *in-process* men på en annan tråd (*thread)*.
2. *Inter-process marshaling.* Anrop till ett objekt i en lokal exekverbar serverapplikation (EXE). Denna typ av anrop kallas *Lightweight Remote Procedure Call (LRPC)*.
3. *Inter-machine marshaling.*. Anrop till ett objekt på en annan maskin. Denna typ av anrop kallas *Remote Procedure Call (RPC)*.

Som man kan se i punkt 3 så är *inter-machine marshaling* den typ som används vid DCOM. I kapitel 3.1.3.1 beskrivs övergången från COM till DCOM.
Mer om *marshaling* finns i [4] och [5].

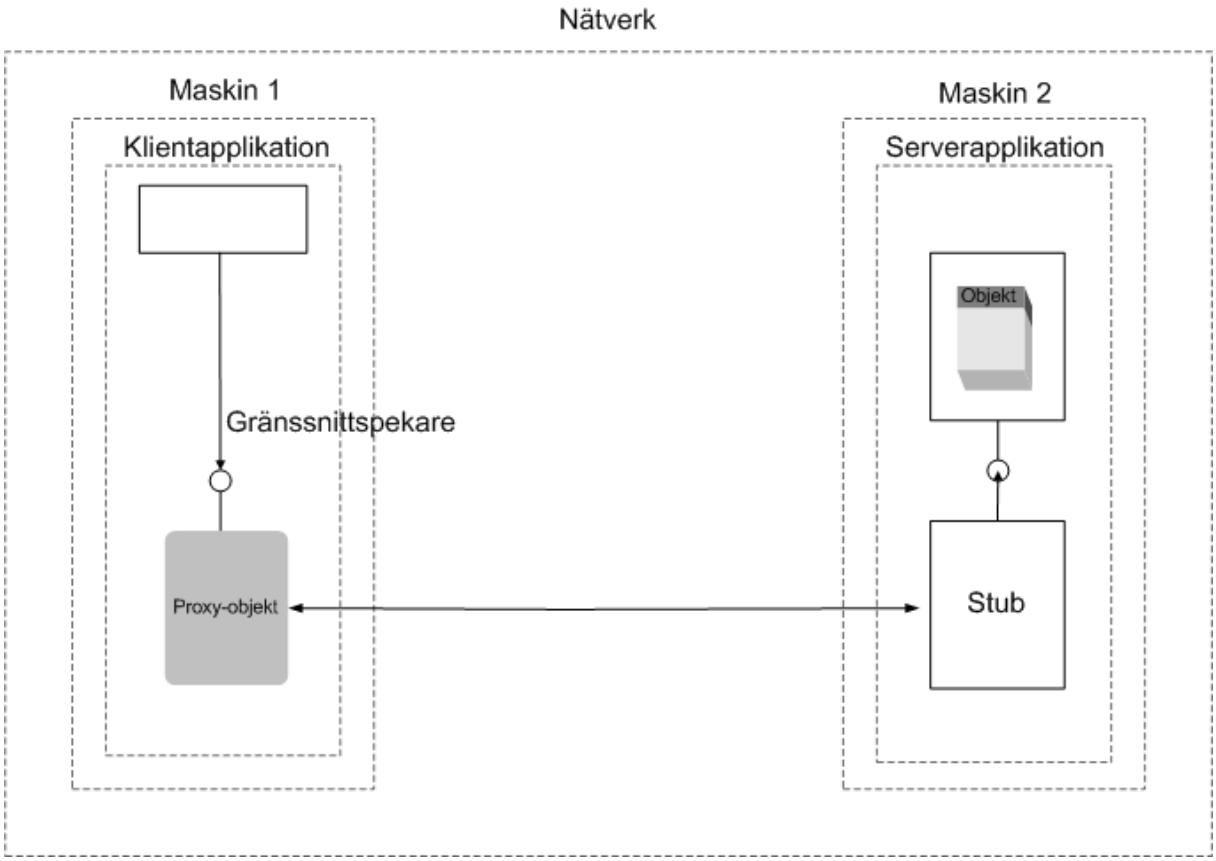
3.1.3 Hur går ett DCOM-anrop till?

I detta kapitel beskrivs vad som skiljer DCOM från COM och ett exempel på ett DCOM-anrop.

3.1.3.1 DCOM istället för COM

Som nämnts tidigare så är DCOM den distribuerade COM-tekniken, d.v.s. DCOM utökar COM till att fungera mellan datorer på någon typ av nätverk.

Enligt kapitel 3.1.2.8 (*Marshaling*) kommer det alltid att finnas ett proxy-objekt på klientapplikationen och en stub på serverapplikationen när det handlar om DCOM. DCOM ser till att proxy-objektet ser ut på exakt samma sätt som objektet på servermaskinen och stubben ser ut som klientapplikationen för serverapplikationen. Detta kan illustreras som i följande figur:



Figur 8: Gränssnittspekaren pekar till ett proxy-objekt som ser ut som det riktiga objektet men finns i klientapplikationen. Proxy-objektet kommunicerar med stubben på serverapplikationen som i sin tur kommunicerar med objektet.

Det viktigaste med det som figuren ovan beskriver är att klientapplikationen kommer att uppfatta det riktiga objektet som lokalt. Tack vare detta uppnås transparens vilket innebär att klientapplikationen inte vet om COM eller DCOM används. Objektet framstår som lokalt i båda fallen.

3.1.3.2 Exempel på anrop med DCOM

Anta samma uppställning som i figur 8 ovan. Klientapplikation A på maskin 1 ska nu anropa metoden *Summa()* som finns i serverapplikation B på maskin 2. *Summa()* har två heltalsparametrar som adderas och svaret returneras. För att A ska kunna hitta B i sitt anrop har A tillgång till den aktuella klassens GUID. Detta GUID fås genom tidig- eller sen bindning beroende på vad som används (se kapitel 3.1.2.7). För att komma åt *Summa()* behöver A också veta vad det gränssnitt som exponerar metoden heter, i det här fallet *Isum*.

Det första som händer är att A kontaktar B med en förfrågan om ett objekt av den aktuella klassen. I sin förfrågan anger A klassen och gränssnitt som den vill ha referens till, i det här

fallet *IUnknown* (det vanliga sättet är att använda standardgränssnittet *IUnknown*, att använda *Isum* direkt hade också fungerat). Klassobjektet i B skapar nu det riktiga objektet samtidigt som COM skapar ett proxy-objekt på klientapplikationen och en stub på serverapplikationen genom att båda applikationerna laddar in *proxy*/*stub DLL* (*standard marshaling* används). Detta görs för att ta hand om den *marshaling* (se kapitel 3.1.2.8) som kommer att ske.

A får tillbaka en gränssnittspekare som refererar till *IUnknown*. För att komma åt metoden *Summa()* använder A *QueryInterface()* och byter gränssnitt, d.v.s. omdirigerar gränssnittspekaren, till *Isum*. Gränssnittspekaren till *IUnknown* behövs inte längre och *Release()* kan anropas för att ta bort denna referens. Metoden *Summa()* kan nu enkelt anropas genom gränssnittspekaren, som kan användas likt vanliga pekare, och svaret returneras och tas om hand. Efter att objektet är färdiganvänt anropar klientapplikationen *Release()* för referensen till *Isum*. Eftersom referensräknaren nu är noll kan objektet förstöra sig själv.

3.2 Webbtjänstteknik

En webbtjänst (engelska: *Web Service*) är en serverapplikation vilken kan anropas via standardwebbprotokoll. I detta kapitel kan man läsa om hur webbtjänster fungerar och vad de består av.

3.2.1 Bakgrund och förklaring

En webbtjänst tillhandahåller tjänster som kan nås över Internet, man anropar en metod på en webbtjänst och får ett resultat i retur. Med webbtjänster kan man enkelt ansluta sig till tjänster över världens största nätverk, Internet. DCOM använder sig inte av Internet p.g.a. säkerhetsskäl då det är svårt att hålla ihop *Windows NT*-domäner över Internet. Men programmeraren behöver inte bry sig om de krångliga detaljerna med att arbeta över Internet eftersom detta sköts med hjälp av standardwebbprotokoll som till exempel *Hyper Text Transfer Protocol (HTTP)* och *Simple Object Access Protocol (SOAP)*. SOAP beskrivs i senare kapitel.

Webbtjänster bygger på komponentmodellen, på samma sätt som COM-komponenter, och använder sig av standardwebbprotokollen enligt ovan för åtkomst av tjänsten.

För kommunikationen med en webbtjänst gäller att informationen överförs i XML-format och meddelandet är skickat som ett SOAP-meddelande. Det finns två andra viktiga delar i webbtjänster. Den ena är *Web Service Description Language (WSDL)* och den andra är *Universal Description Discovery och Integration (UDDI)*. Man kan enkelt förklara några av webbtjänstens delar genom att säga att en webbtjänst är beskriven av WSDL, publicerad och

hittad av UDDI och är bunden och blir aktiverad genom SOAP. Mer om dessa kan läsas om i senare kapitel. *Web Service Flow Language (WSFL)/XLANG* är ett språk som används för att sätta samman flera webbtjänster till en ny tjänst. Detta förklaras inte ytterligare i kapitlet.

3.2.2 Teknisk redogörelse

Nedan följer några av de viktiga begreppen när man talar om webbtjänster.

3.2.2.1 XML

Extensible Markup Language (XML) är ett språk som idag kan användas bl.a. till att på ett strukturerat och standardiserat sätt överföra data mellan två parter. XML, liksom HTML, är utvecklat av *World Wide Web Consortium (W3C)*[2] och är en öppen industristandard, vilken representerar data på ett plattformsoberoende sätt. XML är en framtagen delmängd av *Standard Generalized Markup Language (SGML)*, det vill säga en förenklad specifikation på hur man definierar dokumenttyper. Ordet "extensible" kommer av att man till skillnad från HTML kan använda sina egna elementnamn. Man kan därför i XML egendefiniera element och undviker därmed den begränsningen med HTML.

I början av 90-talet framställdes HTML för att, som ett universellt format, överföra dokument över Internet. Men med tiden var den statiska hanteringen av webbsidor inte tillräcklig. Behovet av dynamiska sidor ökade och 1996 började *XML Working Group* (ursprungligen känd som the *SGML Editorial Review Board*), som bildades under överinseende av W3C, utveckla XML. Den 10 februari 1998 blev versionen 1.0 en W3C rekommendation vilket kan anses vara en standardisering som följd av W3C:s inflytande.

För att enkelt skildra hur ett XML-dokument ser ut och hur det kan användas har vi valt att ge ett exempel på hur det kan se ut när man ska hämta information om vilka verktyg som finns på en sida.

Strukturen för ett XML-dokument är väldigt likt ett HTML- eller ett SGML-dokument något som illustreras i följande figur:

[2] W3C är ett standardiseringsorgan för Internet teknologier.

```
<?xml version="1.0" ?>
<tools>
<tool>
        <title>Softingtool</title>
        <authors>
                <author_name>Ola Noreklint</author_name>
                <author_name>Erik Nordstroem</author_name>
        </authors>
        <price SEK="X kr"/>
</tool>
</tools>
```

Figur 9: Ett exempel på ett XML-dokument. Dokumentet innehåller information om vad det finns för redskap.

Koden är skriven i ett textdokument och måste alltid sparas som en XML-fil, d.v.s. som t.ex. filnamn.xml. Vi går igenom koden för att se vad som egentligen händer:

```
<?xml version="1.0" ?>
```

Detta talar om för den mottagande applikationen att den får ett XML-dokument som är kompatibelt med version 1.0 av XML-specifikationen. Frågetecknet står för att det är en definition och inte ett element.

Varje XML-dokument måste innehålla ett rotelement, vilket innebär att dokumentet har en unik start- och sluttagg. I exemplet heter rotelementet *tools*, vilket associerar till att vi har en fil innehållande verktyg:

```
<tools>
…
</tools>
```

Starttaggen börjar alltid med ’<’ och slutar med ’>’ innehållande elementets namn och ett antal attribut. Sluttaggen börjar alltid med ’</’ och slutar med ’>’. Innehållet mellan dessa två får endast vara namnet på det element taggen avslutar.

Varje element definieras av en start- och en sluttagg samt det innehåll som dessa två omsluter. Innehållet kan vara en text, ytterligare element eller båda delarna. I vårt exempel innefattar det elementen:

```
<tool>
        <title>Softingtool</title>
        <authors>
                <author_name>Ola Noreklint</author_name>
                <author_name>Erik Nordstroem</author_name>
        </authors>
        <price SEK="X kr"/>
</tool>
```

Vi ser här hur *tool* innehåller elementen *title, authors och price*, där *title* innehåller en text och *authors* innehåller element som i sin tur innehåller text. *price* är en tom elementtagg som avslutas med ’/’. Valutan och beloppet finns inuti SEK-attributet.

Vi har nu berättat om hur ett XML-dokument kan se ut. Men hur används det? Vi belyser här två områden:

- Ett XML-dokument kan användas för att presentera information. För att dokumentet ska kunna läsas behöver vi en webbläsare**.** Den fullständiga koden kommer då att visas. För en mer attraktiv visualisering kan man använda sig av ett så kallat ***style sheet***. Detta består av en separat **css**-fil där vi använder oss av ett helt annat språk för att förklara hur vi vill att dokumentet ska presenteras.
- Ett annat sätt är att använda den som ren informationsfil. Vid t.ex. kommunikation med en webbtjänst används SOAP, som i sin tur består av XML-kod. Detta behandlas i nästa kapitel.

Man kan se att XML är ett mycket enkelt och lättförståligt språk med en liten mängd syntaxregler. Den enda egentliga utbyggnaden av XML-syntaxen är **namnrymder** (engelska: *namespaces*). **Namnrymder** gör det möjligt att identifiera och särskilja varje innebörd av ett element så att risken för semantiska konflikter elimineras. Ett exempel på en konflikt som kan uppstå är det vanliga ordet ”titel”, som kan betyda såväl boktitel som yrkestitel. Utanför ursprungsdokumentet riskerar <titel> att blandas ihop med andra element med samma namn, men med en annan innebörd. Genom ett prefix (till exempel <person:titel>) ger man

elementet en universiell betydelse. Principen för **namnrymder** är inte helt okontroversiell, men trots det används den av i snart sagt alla viktiga applikationer inom XML-familjen. Läs mer om namnrymder i [11].

När man använder sig av XML-dokument finns möjligheten att kontrollera dessa genom att använda sig av *XML Schema Definition (XSD)*. **XSD** tillåter definiering av strukturen och datatyper för XML-dokumentet så att det motsvarar de riktlinjer som satts upp av W3C. Detta innebär att man kan jämföra ett XML-dokument mot en **XSD** och få svar på om dokumentet har rätt struktur och innehåller rätt datatyper. Vanliga datatyper (sträng, heltal o.s.v.) finns inbyggda i **XSD**. Egenskapade datatyper kan definieras med hjälp av *simpleType* och *complexType*. Dessa tillåter skapandet av komplexa datatyper, d.v.s. datatyper som består av andra datatyper. Ett schema måste inkludera följande namnrymd:

http://www.w3.org/2001/XMLSchema

Denna namnrymd anges för att referera till W3C och för att referera till de definitioner som finns angivna där.

Läs mer om XML i [7], [8], [9] och [10].

3.2.2.2 SOAP

Simple Object Access Protocol (SOAP) är ett XML-baserat protokoll som är designat för utbyte av information i en decentraliserad distribuerad miljö. SOAP använder sig av existerande Internetprotokoll och format som inkluderar t.ex. HTTP, SMTP och MIME.

Till skillnad från andra RPC-protokoll stöder inte SOAP avancerade funktioner som t.ex. *distributed garbage collection*[3]. Detta var ett av målen med designen av SOAP, att det skulle vara enkelt. Det andra var att det skulle vara utbyggbart.

SOAP utvecklades först av Microsoft som en möjlighet att anropa externa funktioner via HTTP. Till vidareutvecklingen av SOAP anslöt sig flera företag. Deras sammanlagda ansträngningar ledde fram till SOAP 1.1 specifikationen, som lämnades över till W3C den 8 maj 2000. W3C har sedan dess gjort ändringar som lett fram till SOAP 1.2 specifikationen, 9 juli 2001, som ännu inte antagits som en rekommendation, men väntas göra det inom kort.

[3] *Distributed garbage collection* är en tjänst som rensar upp ej borttagna objektinstanser ur minnet.

Designen av SOAP är väldigt enkel.

SOAP består av fyra delar:

1. **SOAP envelope**. Konstruktionen definierar ett generellt ramverk för att förklara vad som finns inne i meddelandet och vem som ska ta hand om det.
2. **SOAP encoding rules**. Definierar en mekanism som kan byta ut applikationsdefinierade datatyper, d.v.s. representera typerna i XML-kod istället.
3. **SOAP RPC representation**. Definierar en konvention som kan användas till att representera fjärranrop och svar till procedurer, d.v.s. hur metodanrop och svar anges med XML-kod i ett SOAP-meddelande.
4. **SOAP binding framework**. Definierar ett abstrakt ramverk för utbyte av SOAP-meddelande mellan noder som använder underliggande transportprotokoll, d.v.s. definierar hur ett SOAP-meddelande kan vara buret av ett HTTP-meddelande.

Ett SOAP-meddelande är ett XML-dokument som består av ett obligatoriskt *Envelope*, en valfri *Header* och en obligatorisk *Body*.

- *SOAP envelope* är toppelementet i XML-dokumentet som representerar SOAP-meddelandet.
- *SOAP header* är ett element som kan användas till att utöka funktionaliteten för kommunikationen. Här kan SOAPAction läggas till för att brandväggar ska kunna filtrera oönskade anrop.
- *SOAP body* är den del av meddelandet där informationen som ska skickas till mottagaren ligger.

I figur 10 går det att se en symbolisk bild på ett SOAP-meddelande:

Figur 10: Envelope som omsluter både header-elementet och det obligatoriska body-elementet.

SOAP har stor betydelse när det gäller webbtjänster eftersom det är via SOAP-meddelanden som kommunikationen med en webbtjänst sker. När en klientapplikation gör ett anrop till en webbtjänst görs detta genom ett SOAP-meddelande. När webbtjänsten returnerar görs detta också i ett SOAP-meddelande.

Läs mer om SOAP i [7], [12], [13] och [14].

3.2.2.3 WSDL

Web Service Description Language (WSDL) ger en XML-baserad presentation av de metoder som en viss webbtjänst erbjuder, d.v.s. webbtjänstens gränssnitt.

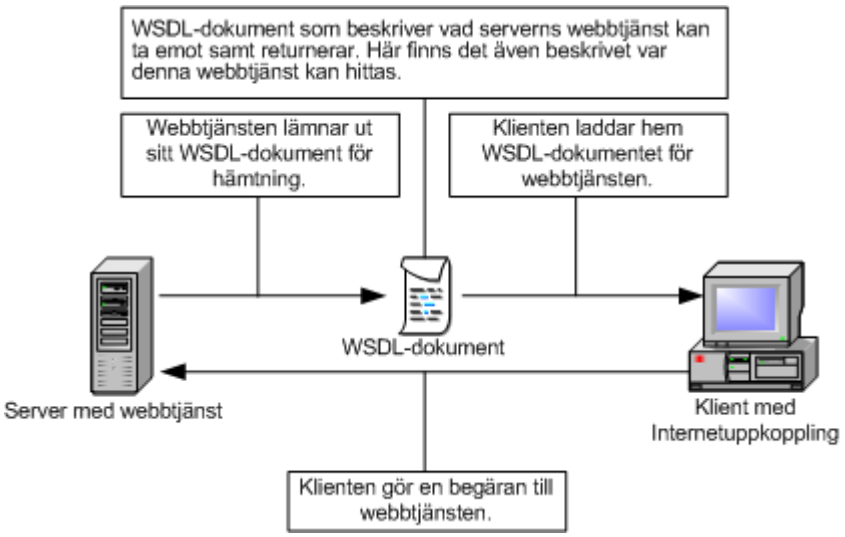
Ett WSDL-dokument ger information om namnen på metoderna, typ av parametrar och typ av returdata. Dokumentet gör det även möjligt att använda såväl statisk bindning (namn och parametrar kontrolleras vid kompileringen av klientapplikationen som anropar webbtjänsten) som dynamisk bindning (kontroller sker först vid anrop under exekvering).

Genom att ladda hem ett WSDL-dokument kan den som vill använda sig av tjänsten förstå hur de ska kommunicera med den. Ett WSDL-dokument beskriver all information som behövs för att kunna sätta upp kommunikationen mot den webbtjänst som dokumentet beskriver. Hur ett anrop utförs mot tjänsten, vilka parametrar anropet ska innehålla och vad som returneras är det mest intressanta. Det första som definieras i dokumentet är de typer som används till parametrar och returvärde för metoder.

Ett smidigt sätt att använda sig av ett WSDL-dokument är att direkt referera till den. Tack vare speciellt utvecklade verktyg, som t.ex. *Intellisence*[4] i *Microsoft Visual Basic .NET*, kan man sedan se metoderna som webbtjänsten tillhandahåller direkt under implementeringen av klientapplikationen.

WSDL-dokument skapas automatisk av *Microsoft Visual Studio .NET* vid kompilering av webbtjänster men kan även skapas manuellt av den som utvecklar webbtjänsten.

I figur 11 ser vi hur ett WSDL-dokument skickas och används mellan en klientmaskin och en servermaskin.



Figur 11: Visar hur ett WSDL-dokument kan hämtas.

För närvarande används version 1.1 av WSDL och finns hos W3C som ett förslag på hur tjänster kan beskrivas och är öppen för diskussion. Den är således ingen standard och det framgår tydligt att detta är något som är under utveckling. Många företag har anslutit sig till vidareutvecklingen av version 1.1, bland andra IBM, vilka även var med och utfärdade den första versionen, vilket var version 1.0.

Det finns sex olika definitioner (nyckelord) som bygger upp ett WSDL-dokument.

1. **types -** Definierar egna datatyper som beskriver utbyte av meddelanden.
2. **message -** Definierar dataformatet för kommunikationen.

[4] *Intellisence* är ett automatiserat verktyg i *Microsoft Visual Studio 6.0* och *Microsoft Visual Studio .NET* för användning av definierade typer och funktioner.

3. **portType -** En mängd abstrakta operationer. Varje operation refererar till ett inkommande meddelande och ett utgående meddelande. Det är med hjälp av denna som det anges hur kommunikationen mot tjänsten utförs.

4. **binding -** Definierar vilket kommunikationsprotokoll som ska användas vid överförandet av ett anrop, dvs. vilket protokoll som ska bindas till ett specifikt *portType* element.

5. **port -** Specificerar en adress för en bindning, d.v.s. definierar vart meddelandet ska skickas med kommunikationsprotokollet.

6. **service -** Används för att samla ihop en mäng relaterade portar och därigenom definierar den fysiska positionen för en webbtjänst.

Definitionerna **types**, **message** och **portType** tillhör de abstrakta och används för att beskriva anrop på ett plattforms- och språkoberoende sätt. **Binding**, **port** och **service** innehåller mer konkret information om webbtjänsten. Ett exempel på ett WSDL-dokument finns i bilaga D.

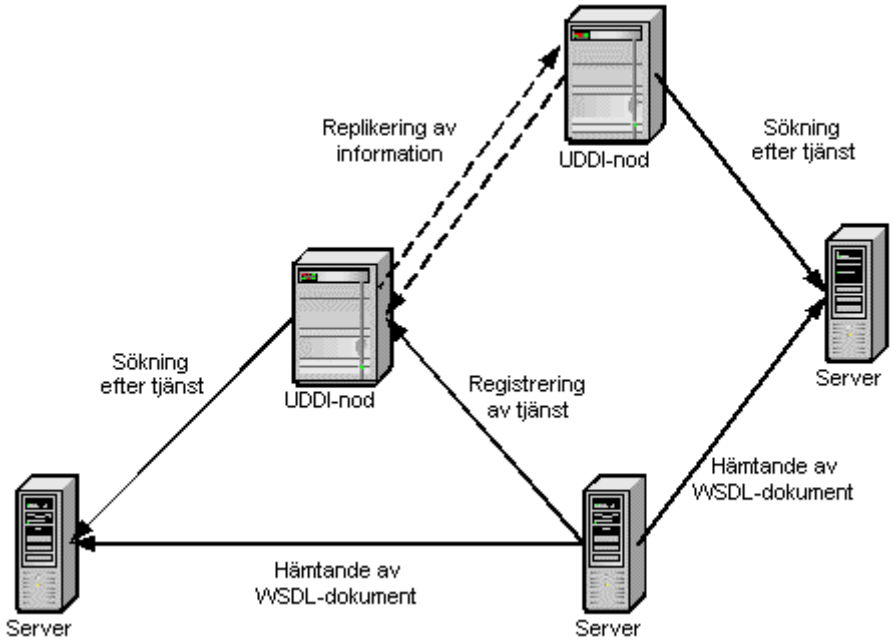
Läs mer om WSDL i [7] och [15].

3.2.2.4 UDDI

Universal Description, Discovery och Integration (UDDI) är namnet på en grupp webbaserade register som tillhandahåller information om webbtjänster och dess tekniska gränssnitt. Registren körs genom en webbsida och kan användas såväl av de som vill göra webbtjänster tillgängliga som de som söker dem.

Genom att ha tillgång till och använda offentliga söksidor för UDDI-register kan användaren söka information om webbtjänsterna som är registrerade. De grundläggande tjänsterna hos söksidorna är kostnadsfria.

Logiskt sett ska UDDI-register te sig som en enda enhet, vilken sköter all administrering av informationen. Fysiskt består UDDI-register av ett flertal noder utspridda över världen. Varje nod i nätverket har en unik UUID-nyckel (en typ av GUID, se kapitel 3.1.2.2) för att identifieras, men kan trots det vara uppbyggt av ett flertal datorer. Mening är dock att varje nod i nätverket ska kunna upptäcka ändringar som utförs i en enskild nod så att en fråga om tillgängliga webbtjänster medför samma resultat oavsett på vilken nod som anropet utförs. Hur UDDI-noderna och servrarna kan förhålla sig till varandra ser vi ett exempel på i figur 12.



Figur 12: En överskådning hur server och UDDI-noder förhåller sig till varandra. Som användare kan man utnyttja sök- och registreringstjänsten som webbsidan på servern tillhandahåller. Sökningen sker på UDDI-noderna.

Version 1.0 av UDDI publicerades den 6 september år 2000 av *Ariba Inc.*, *IBM* och *Microsoft*. Vid detta tillfälle låg 36 företag bakom UDDI. När version 2.0 kom ut den 8 juni 2001 var antalet företag uppe i fler än 175. Efter det att den andra versionen publicerades är det planerat att en version till ska skrivas, innan specifikationen ska lämnas över till W3C. UDDI är tänkt att lösa samma problem som *Yahoo!* löste för Internet 1994 i form av sökning efter information. Med hjälp av UDDI ska webbtjänster kunna lokaliseras genom ett register, där befintliga webbtjänster kan registreras.

UDDI-projektet använder sig av W3C- och IETF-standarder som XML, HTTP och DNS-protokoll och tidigare versioner av SOAP. IETF står för *Internet Engineering Task Force* och DNS för *Domain Name System.*

Det finns tre saker som UDDI bidrar med till webbtjänster: det är ett standardiserat sätt att beskriva webbtjänster på, en enkel mekanism för att hitta tjänsterna och ett centralt register som samlar ihop alla registrerade webbtjänster. Informationen som en tjänst kan registrera inkluderar svar på frågorna: *”Vem har registrerat tjänsten?”*, *”Vad gör tjänsten?”*, *”Var är tjänsten upplagd?”*. Enkel information som namn, tjänstidentifierare (*D&B DUNS Number®*[5]

[5] *Dun & Bradstreet Data Universal Numbering System*. En unik niosiffrig kod som hjälper till att identifiera över 53 miljoner företag över hela världen.

mm.) och kontaktinformation svarar på vem som registrerat tjänsten. Vad tjänsten gör besvaras av en beskrivning som finns med på webbsidan. Svaret på var tjänsten är upplagd inkluderar tjänstens *Uniform Resource Locator* (*URL*). Dessa referenser kallas för *tModel* på denna webbsida. All information presenteras tydligt i en webbläsare där du kan läsa och referera till WSDL-dokumentet från den länkande URL:en. Man kan även referera till dokumentet direkt om man redan vet webbtjänstens adress.

Datastrukturen beskriver hur UDDI-registret är uppbyggt för att lagra information om webbtjänsten. Strukturen är XML-baserad och kan manipuleras med olika SOAP-meddelanden. Själva informationen som finns i XML-strukturen är tänkt att innehålla fakta om en verksamhet, både allmän och mer teknisk information. Basstrukturen består av fem stycken datatyper/kärnelement för att underlätta förståelsen samt sökningen av information. Varje datatyp innehåller i sin tur datafält, som innehåller information om en verksamhet och mer teknisk information.

Läs mer om UDDI i [7], [16] och [17].

3.2.3 Hur går ett webbtjänstanrop till?

För att få en tydlig bild av hur ett webbtjänstanrop går till beskriver vi detta genom ett exempel. Exemplet bygger på uppsökandet av webbtjänsten *Service1* med metoden *Summa()* och vi kommer att ta upp hur man upprättar en förbindelse till den och hur man använder den med hjälp av det vi tidigare redogjort i dokumentationen.

3.2.3.1 Att hitta en webbtjänst

För att använda sig av den önskade webbtjänsten måste man först och främst veta var någonstans på webben den ligger. Om man inte vet det från början kan man använda UDDI (Se kapitel 3.2.2.4) och söka efter den. Detta gör man t.ex. på webbsidan http://uddi.microsoft.com/. Hittar man den här kan man enkelt referera till den. Det är i stort sett det samma som att referera till en vanlig serverapplikation, men i det här fallet använder man sig av tjänstens WSDL-dokument. Dokumentet kan användas genom att skriva in sökvägen till webbtjänsten och fråga efter dess WSDL enligt följande:

http://localhost/SummaWebService/Service1.asmx?WSDL

WSDL-dokumentet gör det möjligt för programmeraren att se vilka funktioner som tjänsten tillhandahåller och hur de används (Se kapitel 3.2.2.3).

3.2.3.2 Exempel på anrop till webbtjänst

I bilaga B och D ser vi webbtjänsten *Service1* med metoden *Summa()* och dess kod respektive WSDL-dokument. För att använda funktionen måste två parametrar skickas med. Dessa kommer att fungera som termer för summeringen. Anta en klientapplikation A som ska anropa metoden S*umma()* som finns på webbtjänsten B. Anropet och aktiveringen av tjänsten görs båda i ett och samma SOAP-meddelande som A skickar till B. I bilaga C.1 kan vi se hur meddelandet ser ut till webbtjänsten. Överst anges var tjänsten finns och hur meddelandet skickas. *SOAPAction* är till för att brandväggen ska kunna skilja på vilka som har rättigheter till porten och inte. Den blå texten kommer i anropet att ersättas med riktiga värden, så att *long* kommer i elementet *term1* att ersättas med **3** och elementet *term2* med **4**. *length* ersätts med ett värde som representerar innehållets längd. B tar emot SOAP-meddelandet och ser till att *Summa()* anropas med de givna parametrarna. I bilaga C.2 ser vi hur B returnerar svaret i ett nytt SOAP-meddelande. A har nu gjort en förfrågan till webbtjänsten som i sin tur returnerat svaret. Svaret kan efter detta användas av A.

Att anropa en metod på en webbtjänst kan gå till på flera sätt. *Microsoft Visual Studio .NET* är en plattform som specialiserat sig på webbtjänster. Där finns funktioner som hjälper till med händelseförloppen och döljer SOAP-meddelandet för användaren. Att anropa webbtjänsters metoder ser därför likadant ut som metodanrop till vanliga objekt. Ett annat sätt är att anropa tjänsten genom att ange adressen till den och de parametrar som krävs direkt i webbläsaren, ett s.k. *HTTP Get*:

http://localhost/SummaWebService/Service1.asmx/Summa?term1=3&term2=4

Svaret redovisas då direkt i webbläsaren:

```
<?xml version="1.0" encoding="utf-8" ?
<long xmlns="http://tempuri.org/">7</long>
```

Den URL som finns i svaret är ett exempel på en namnrymd.
Webbtjänsten är alltså ganska flexibel när det gäller anrop till den.

3.3 Jämförelse: webbtjänst och DCOM-tekniken

I detta kapitel jämförs webbtjänsttekniken med DCOM-tekniken.

3.3.1 Gränssnitt till en serverapplikation

För att kunna anropa en serverapplikation krävs ett gränssnitt mot klientapplikationen. I kapitel 3.1.2.6 beskrivs hur man med IDL kan definiera ett gränssnitt i fallet med DCOM-tekniken och sedan länka serverapplikationen till den kompilerade IDL-filen (*Type Library*) och använda den kompilerade IDL-filen som mall, d.v.s. specificera gränssnitt som man sedan använder i serverapplikationen. Motsvarigheten till detta finns inte för webbtjänster. Istället genereras gränssnittet utifrån den färdiga webbtjänsten i ett WSDL-dokument (se kapitel 3.2.2.3). En IDL-fil och ett WSDL-dokument kan ändå sägas vara jämförbara eftersom de i text beskriver gränssnittet för serverapplikationen. Vi kan inte se någon speciell fördel för DCOM-tekniken med specificering av gränssnitt i IDL.

Ett WSDL-dokument kan också anses vara jämförbar med ett *Type Library* eftersom det är möjligt för serverapplikationer att referera till dessa. Här kan noteras att WSDL-filer är XML-baserade och därför enkla att förstå för olika plattformar, t.ex. miljöer med *Microsoft Windows* eller *UNIX* som operativsystem. Ett *Type Library* kan vara svårt att förstå mellan olika plattformar eftersom det är kompilerad kod och binärt. Detta är viktigt eftersom webbtjänsttekniken på detta sätt uppnår plattformsoberoende. DCOM-tekniken kan ses som att vara mer lämpade till system gjorda av *Microsoft*.

Slutligen kan man säga att hanteringen av gränssnitt underlättas i fallet med webbtjänsttekniken då man slipper använda registret och GUID, som i fallet med DCOM-tekniken. Detta ligger till grund för den stora fördelen med webbtjänsttekniken gentemot DCOM-tekniken: att man inte behöver uppdatera registret på varje klientmaskin när man uppdaterar serverapplikationen vilket diskuteras ytterligare i kapitel 3.3.2.

3.3.2 Bindning

För bindning mellan applikationerna i DCOM- och webbtjänsttekniken, d.v.s. sättet klientapplikationen hittar serverapplikationen på, skiljer sig tillvägagångssättet betydligt. För DCOM-tekniken gäller enligt kapitel 3.1.2.7 att de GUID som är aktuella måste registreras på maskinen klientapplikationen finns på. Genom att registret används kan sedan klientapplikationen hitta serverapplikationen. Detta innebär att en installation måste göras på den maskin som klientapplikationen körs på för att skriva in nya GUID i registret. För webbtjänsttekniken är bindningen baserad på den URL som webbtjänsten har. En URL till en webbtjänst kan hittas genom att manuellt söka med hjälp av UDDI (se kapitel 3.2.2.4).

3.3.3 Att anropa en serverapplikation

För ett anrop till en serverapplikation, via DCOM- eller webbtjänstanrop, finns stora skillnader som åskådliggjorts i tidigare kapitel (se kapitel 3.1.3.2 och 3.2.3.2). Att anropa en serverapplikation med webbtjänsttekniken är t.ex. betydligt mindre komplicerat än att göra det med DCOM-tekniken.

I fallet med DCOM-tekniken behövs minst en, men oftast två turer över nätverket för att objektet ska skapas och klientapplikationen ska få rätt gränssnittspekare innan själva metodanropet kan utföras. En eller två turer beror på om man först får en gränssnittspekare till *IUnknown* och sen byter gränssnitt eller om man får en gränssnittspekare till det gränssnitt som ska användas direkt. Det vanliga är att man går via *IUnknown* (se även kapitel 3.1.3.2). Till detta tillkommer också att klientapplikationen måste hålla reda på referensräknaren, något som gör att risken finns för klientapplikationen att åstadkomma felaktigheter, som t.ex. minnesläckage på den maskin serverapplikationen exekverar på. För anropet med webbtjänsttekniken gäller att aktivering av objektet och anropet till metoden sker i samma anrop (SOAP-meddelande). Webbtjänsttekniken innebär alltså färre meddelanden över nätverket men också att det inte finns något som knyter klientapplikationen till serverapplikationen eftersom kommunikationen mellan applikationerna endast baseras på SOAP-meddelanden. Klientapplikationen kan heller inte åstadkomma felaktigheter i serverapplikationens minne eftersom objektet försvinner så fort det anropats. Med DCOM-tekniken är klientapplikationen hårt knuten till serverapplikationen genom *marshaling* (se kapitel 3.1.2.8). Att ha applikationerna hårt knutna till varandra behövs för att möjliggöra tillstånd. I kapitel 3.3.4 tas diskussionen kring tillstånd upp mer ingående.

3.3.4 Tillstånd

En viktig skillnad mellan DCOM-tekniken och webbtjänsttekniken är att serverapplikationer i fallet med DCOM-tekniken har tillstånd vilket motsvarande serverapplikationer i webbtjänsttekniken inte har. Med tillstånd menas att ett objekt finns kvar mellan två anrop, d.v.s. objektet har ett tillstånd och är bestående. Detta får bl.a. följande konsekvenser:

För DCOM-tekniken innebär detta att objekt kan tillåtas ha egenskaper. Eftersom det är samma objekt som anropas varje gång får egenskaper en mening. Då detta inte gäller webbtjänsttekniken är det inte meningsfullt med egenskaper för objekten här. Enligt objektorienteringens definition av klasser finns kravet att en klass ska innehålla metoder och egenskaper. Ett objekt skapad från webbtjänst kan enligt ovanstående resonemang inte ha

egenskaper på ett meningsfullt sätt, vilket innebär att webbtjänsttekniken inte riktigt uppfyller detta krav, något som DCOM-tekniken gör.

Att använda objektens egenskaper är dock inte alltid att föredra. Att sätta data till egenskaper (kallas *Chatty*) istället för att skicka den som parameter till en metod (kallas *Chunky*) leder till fler turer över nätverket. Att webbtjänster saknar egenskaper behöver alltså inte vara negativt med detta resonemang.

3.3.5 Skalbarhet

Det finns två typer av skalbarhet (engelska: *scalability*): *Scale out* och *Scale up*. *Scale out* innebär att man delar upp ett problem och löser delproblemen distribuerat, d.v.s. på olika maskiner. *Scale up* innebär att man i samma takt som problemets storlek ökar tillsätter mer och mer resurser på den lokala maskinen för att lösa problemet. Med ett problem här menas användandet av en serverapplikation från flera olika klientapplikationer.

I princip kan man inte använda *Scale out* i DCOM-tekniken eftersom serverapplikationerna har tillstånd och det därför är samma objektinstans som klientapplikationen kommunicerar med vid efterföljande anrop. Möjligheten till att distribuera ett problem försvinner därför delvis eftersom andra objekt inte får användas istället om inte bindningen mellan klientapplikationen och objektet bryts först. En annan sak som talar emot *Scale out* för DCOM-tekniken är att eventuella alias för en serverapplikation måste finnas i registret och därifrån behandlas av en DNS, som dynamiskt kan omdirigera anropen. Detta är betydligt mer komplicerat än för webbtjänsttekniken där *Scale out* fungerar ledigt, helst med hjälp av en DNS-server. Denna typ av skalbarhet är en stor fördel för webbtjänsttekniken. För att effektivisera en hårt trafikerad webbtjänst skulle man kunna tänka sig flera serverapplikationer som utför samma sak och en DNS-server som distribuerar ut klientapplikationernas anrop med någon lämplig schemaläggningsalgoritm.

S*cale up* i DCOM-tekniken kan exemplifieras enligt följande: anta en serverapplikation som används av 10000 klientapplikationer. Detta innebär att 10000 objektinstanser existerar samtidigt, eller i alla fall att det finns 10000 kopplingar (proxy-objekt och stubbar) mellan serverapplikationen och klientapplikationerna att hålla reda på. Dessa 10000 kopplingar finns oberoende av om klientapplikationen för tillfället anropar serverapplikationen eller inte och kräver därmed minne. Om antalet kopplingar skulle öka och öka skulle det till slut resultera i att servermaskinens resurser tar slut även om man utökar resurserna så mycket man kan. Med *scale up* kommer man till slut till en nivå när resurserna tar slut.

Om man nu istället antar en serverapplikation med motsvarande funktionalitet i webbtjänsttekniken kommer det inte finnas några kopplingar att hålla reda på här. Istället finns objektet eller objekten endast när de används. När antalet klientapplikationer som använder serverapplikationen ökar kommer ändå inte resurser krävas på servermaskinen för att hålla reda på några kopplingar.

3.3.6 Transaktioner

Med en transaktion menas att operationer som utförs, utförs antingen alla eller inga alls. Möjligheten finns alltså att ångra de operationer som utförts under en transaktion. Idag finns ingen hantering av transaktioner för webbtjänsttekniken utvecklad, vilket innebär att serverapplikationer implementerade som webbtjänster inte kan delta i distribuerade transaktioner. För DCOM-tekniken finns detta vilket innebär en klar fördel gentemot webbtjänsttekniken. Transaktionshantering för webbtjänsttekniken borde dock kunna finnas inom en snar framtid eftersom inga principiellt tekniska begränsningar finns. Problemet idag är bara avsaknaden av en vedertagen standard.

3.3.7 Uppgradering

För ett företag kan det vara viktigt att uppgradera serverapplikationer som tidigare implementerats. Enligt vad som redan diskuterats i 3.3.1 och 3.3.2 har webbtjänsttekniken en klar fördel här eftersom ingen uppdatering behöver göras av registret på klientmaskinerna som i fallet med DCOM-tekniken.

4 Exempelimplementation

I detta kapitel beskrivs exempelimplementationen som ligger till grund för den andra delen av jämförelsen, vilken beskrivs i kapitel 5, där också mätningar görs utifrån denna implementation.

4.1 Inledning

För att belysa skillnaderna mellan ett anrop med DCOM-tekniken och webbtjänsttekniken har vi gjort en exempelimplementation. Exempelimplementationen är ett system för att skicka e-post utan att ha tillgång till ett e-postprogram och konto som är det vanliga. Istället ska klientapplikationen man kör anropa en serverapplikation som i sin tur levererar e-brevet. Funktionen att kunna skicka e-post gjordes för att exempelimplementationen skulle kunna utföra något konkret och för att vi skulle lära oss om utvecklingsmiljön. Den används inte i de mätningar som görs. Serverapplikationen kan nås antingen med DCOM-tekniken eller med webbtjänsttekniken. Klientapplikationen ska också anropa serverapplikationen för att mäta ping-tiden, d.v.s. anropstiden. Hur dessa anrop ser ut kommer att skilja sig en del i de olika teknikerna. För att kunna göra en så bra jämförelse som möjligt har vi använt samma miljö att anropa från i de båda fallen. Denna miljö är *Microsoft Visual Basic 6.0*. För att kunna anropa en webbtjänst från *Microsoft Visual Basic 6.0* används *Microsoft SOAP Toolkit 2.0*, som består av fördefinierade klasser för att ta hand om kommunikation mot webbtjänster med hjälp av SOAP-protokollet. *Microsoft SOAP Toolkit 2.0* kan laddas hem gratis från *Microsofts* hemsida.

4.2 Kravspecifikation

Klientapplikationen och serverapplikationen har separata kravspecifikationer. Nedan beskrivs de krav vi, i samspråk med vår handledare på Pronyx Sweden AB, bestämt för exempelimplementationen. De viktigaste kraven är de som involverar ping-funktionaliteten eftersom funktionen att skicka e-post inte används vid mätningarna.

4.2.1 Klientapplikationen

- Klientapplikationen ska kunna skicka e-post.

- Klientapplikationen ska kunna göra test på ping-tid.
- Man ska kunna få information om programmet.
- Möjligheten ska finnas att antingen använda DCOM- eller webbtjänstteknik.

4.2.1.1 E-postfunktionen

- Klientapplikationen ska ta in tre textsträngar: <mottagare>, <ämne> och <innehåll>.
- Om inget <ämne> anges ska denna textsträng sättas till ”inget”.
- Man ska inte kunna bifoga filer.
- Man ska inte kunna skicka e-brev med blank adress.
- Om man försöker skicka ett e-brev med en felaktig struktur på mottagareadressen, d.v.s. att om ”@” och ”.” inte finns med eller inte står i efterföljande ordning, så ska felmeddelande visas.
- Ingen automatisk avsändaresignatur ska finnas.
- Man ska kunna få hjälp om funktionen.

4.2.1.2 Ping-funktionen

- Ping-tiden ska visas i en textruta.
- Man ska kunna få hjälp om funktionen.

4.2.2 Serverapplikationen

- Serverapplikationen ska ta emot all information om e-brevet och leverera det till rätt mottagare.
- Serverapplikationen ska tillåta ett ping-anrop från klientapplikationen.

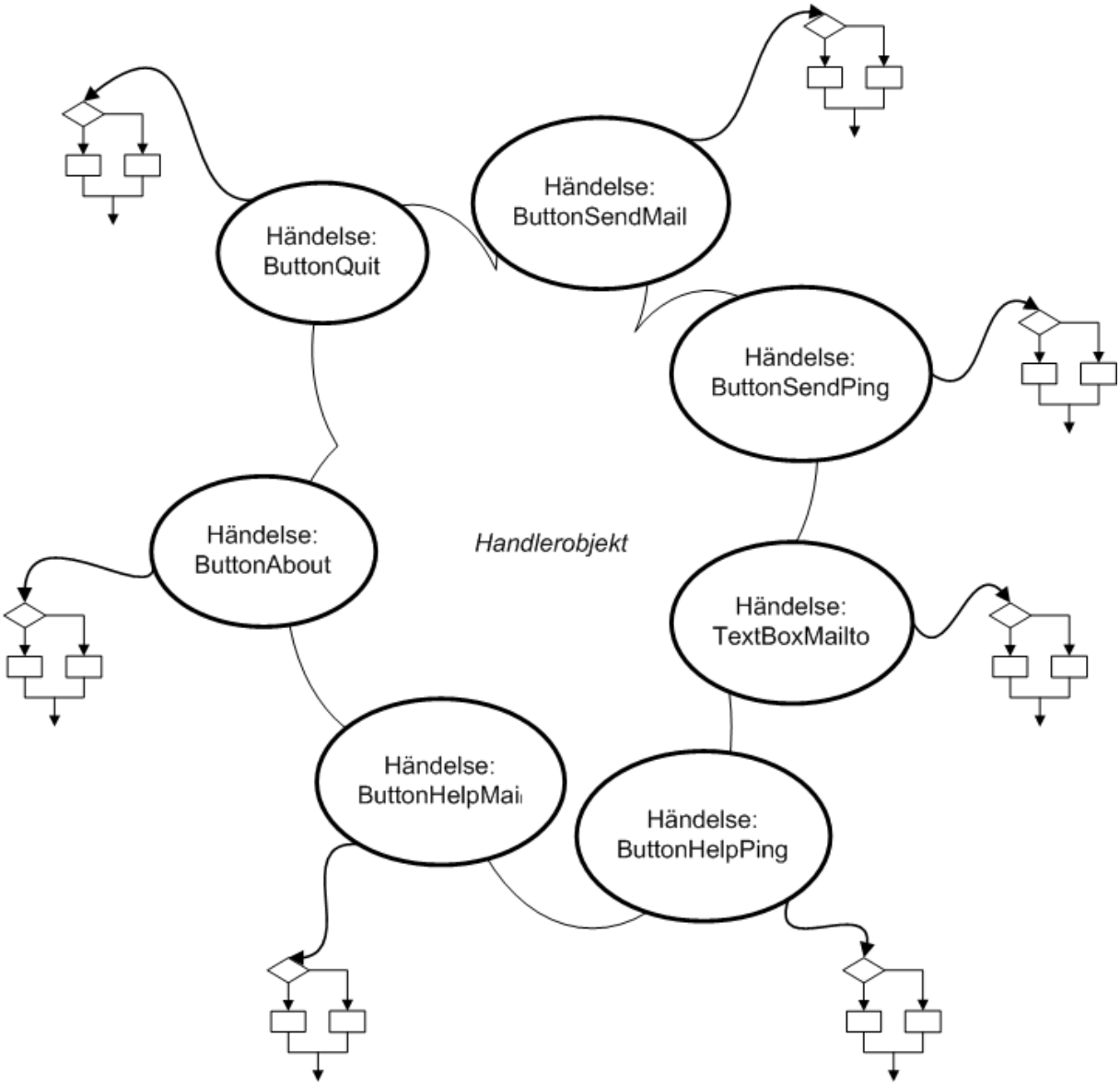
4.3 Klientapplikationen

I detta kapitel beskrivs klientapplikationen.

4.3.1 Design

Eftersom utvecklingsmiljön i *Microsoft Visual Basic 6.0* förenklar implementeringen med hjälp av ett grafiskt gränssnitt mot användaren kommer designen av klientapplikation ha en viss struktur som den i figur 13. En stor fördel med utvecklingsmiljön är att man i princip kan rita upp användargränssnittet. Programspråket är i grund och botten imperativt (trots att objekt finns) och designen beskrivs därför med en metod för detta. Till den grafiska delen av språket hör också att signaler från användaren tas om hand automatiskt. Man kan identifiera ett *Handlerobjekt* som tar hand om händelser och ser till att den motsvarande proceduren körs

när en händelse inträffar. En händelse är t.ex. ett klick på musen eller en ändring i en textruta och en procedur är en sekvens kod att exekvera. Upplägget med *Handlerobjekt*, händelser och flödesscheman kan beskriva enligt följande figur:



Figur 13: I Visual Basic finns ett Handlerobjekt som fångar upp händelser och aktiverar motsvarande procedur. En procedur beskrivs av ett flödesschema.

Följande procedurer finns implementerade i klientapplikationen:

Procedur:	**Aktiverande händelse:**	**Beskrivning:**
ButtonSendMail_Click()	Klicka på knapp ButtonSendMail	Skicka e-post
ButtonSendPing_Click()	Klicka på knapp ButtonSendPing	Skicka ping
TextBoxMailto_Change()	Ändra i textruta TextBoxMailto	Aktivera/Inaktivera knapp ButtonSendMail
ButtonHPing_Click()	Klicka på knapp ButtonHPing	Hjälp om Ping-funktionen
ButtonHMail_Click()	Klicka på knapp ButtonHMail	Hjälp om e-post-funktionen
ButtonAbout_Click()	Klicka på knapp ButtonAbout	Information om programmet
ButtonQuit_Click()	Klicka på knapp ButtonQuit	Avsluta programmet

I bilaga A finns flödesscheman över procedurerna.

4.3.1.1 Funktionsbeskrivning

Följande funktioner kan matchas mot aktiviteterna i de flödesscheman som finns i bilaga A eller används integrerade med några av de andra funktionerna som anges.

FillXmlMessage():

Returvärde: Sträng *XmlDoc.xml* som är kodad i XML och är det ifyllda meddelandet som ska skickas till serverapplikationen.

Parametrar: Sträng *strXML* som är kodad i XML och är det tomma meddelandet som ska skickas till serverapplikationen.

Beskrivning: Tar emot det tomma meddelandet som ska skickas till serverapplikationen och fyller det med information. Använder *CheckMailSubject()* för att avgöra om *ämne* angivits eller ej. Om *ämne* angivits skrivs detta in i meddelandet, annars skrivs ”inget” in i meddelandet.

MailSendSoap():

Returvärde: Inget.

Parametrar: Inga.

Beskrivning: Fyller i och skickar det ifyllda XML-meddelandet till serverapplikationen med hjälp av *FillXmlMessage()* (webbtjänstteknik).

MailSendDcom():

Returvärde: Inget.

Parametrar: Inga.

Beskrivning: Fyller i och skickar det ifyllda XML-meddelandet till serverapplikationen med hjälp av *FillXmlMessage()* (DCOM-teknik).

CheckMailAddr():

Returvärde: Boolesk variabel *blnMailOk* som är det booleska värdet på om den angivna adressen är korrekt eller ej.

Parametrar: Inga.

Beskrivning: Kollar om den angivna adressen är korrekt eller ej.

CheckMailSubject():

Returvärde: Booleskt värde på om ämne är angivet eller ej.

Parametrar: Inga.

Beskrivning: Kollar om ämne är angivet eller ej.

PingSendSoap():

Returvärde: Inget.

Parametrar: Inga.

Beskrivning: Anropar ping-funktionen på serverapplikationen (webbtjänstteknik) och mäter tider för anropet. Tiden som mäts är dels för att läsa in WSDL-dokumentet och dels för att exekvera ping-metoden på serverapplikationen. Tiderna skrivs ut.

PingSendDcom():

Returvärde: Inget.

Parametrar: Inga.

Beskrivning: Anropar ping-funktionen på serverapplikationen (DCOM-teknik) och mäter tiden för anropet. Tiden skrivs ut.

MyErrorHandler():

Returvärde: Inget.

Parametrar: Sträng *description* som är beskrivningen av uppkommet fel. Sträng *source* som är felets källa. Heltal (*Long*) *number* som är felets nummer.

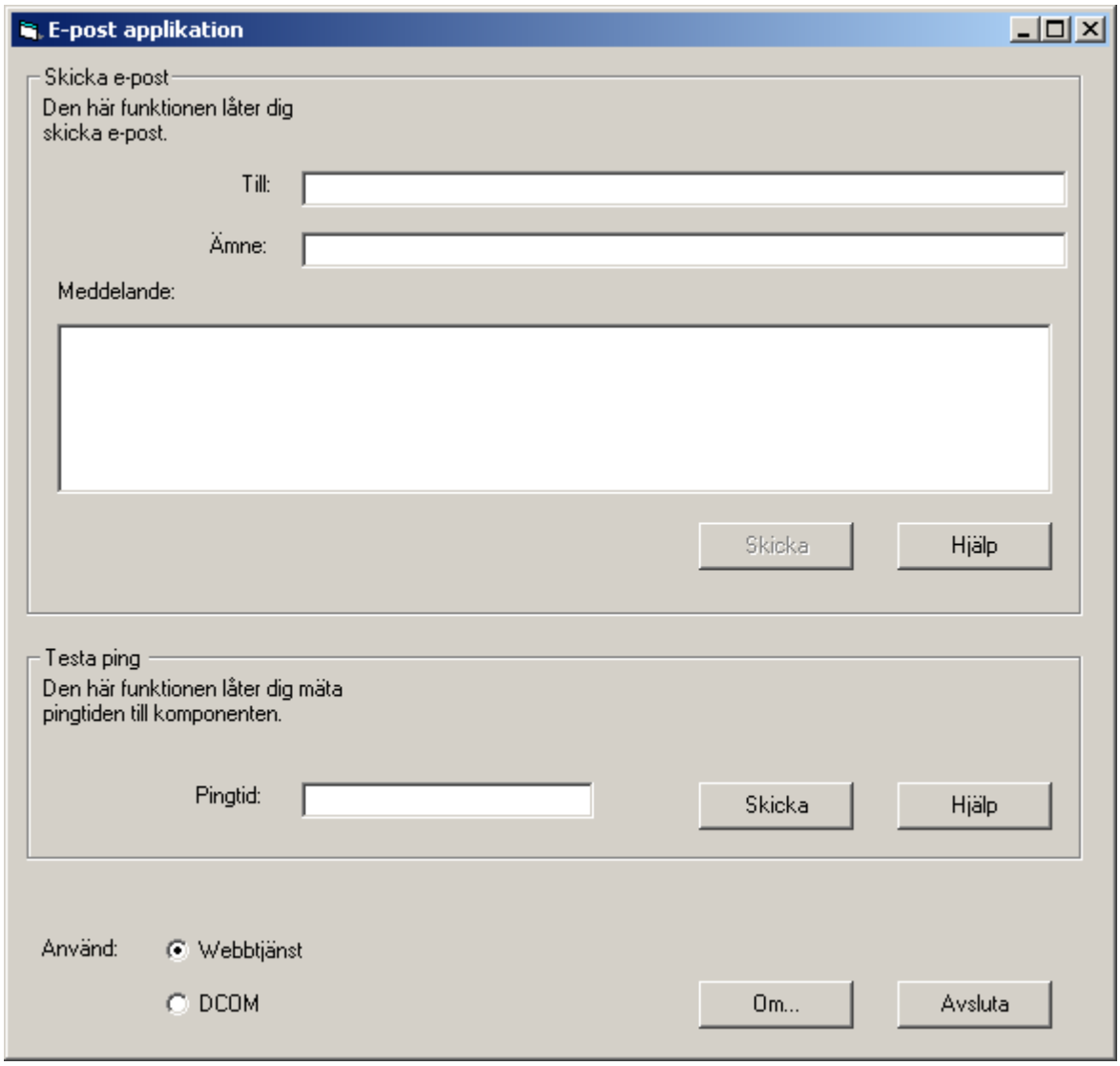
Beskrivning: Skriver, i en textruta, ut information om felet som uppstått.

4.3.2 Implementation

Klientapplikationen är en exekverbar fil, kompilerad från ett STANDARD EXE–projekt i *Microsoft Visual Basic 6.0*.

4.3.3 Grafiskt gränssnitt

Figur 14 visar det grafiska gränssnittet för klientapplikationen.



Figur 14: Klientapplikationens grafiska gränssnitt.

4.4 Serverapplikationen

Serverapplikationen är implementerad dels med DCOM-teknik (se kapitel 4.4.2.1) och dels webbtjänstteknik (se kapitel 4.4.2.2). För att få till ett DCOM-anrop med denna COM-komponent används *COM+*. *COM+* gör att serverapplikationen körs i ett eget minnesutrymme.

4.4.1 Design

Designen för serverapplikationen innefattar metodbeskrivningarna nedan.

4.4.1.1 Metodbeskrivning

Följande metoder är de som kan anropas från klientapplikationen på serverapplikationen, antingen via DCOM- eller webbtjänstteknik. Dessa metoder kan tydligt kopplas till sitt sammanhang. Serverapplikationen är skapad i *Microsoft Visual Basic 6.0* och i *Microsoft Visual Basic .NET*.

SendMail():

Returvärde: Inget

Parametrar: Sträng *strXML*. *strXML* är kodad i XML och innehåller argumenten till e-brevet: Mottagare, ämne och meddelande.

Beskrivning: Tar emot informationen om e-brevet från klientapplikationen och skickar det till mottagaren.

GetEmptyxml():

Returvärde: Sträng *strXML*. *strXML* är kodad i XML och innehåller mallen för hur den parameter ser ut som *SendMail()* kan ta emot.

Parametrar: Inga

Beskrivning: Returnerar en korrekt mall för den XML-sträng som *SendMail()* tar emot som parameter.

Ping():

Returvärde: Inget

Parametrar: Inga

Beskrivning: Metod som inte exekverar något. Används för att mäta tiden det tar att anropa/aktivera serverapplikationen.

4.4.2 Implementation

I detta kapitel beskrivs implementeringen av serverapplikationen.

4.4.2.1 Microsoft Visual Studio 6.0

Serverapplikationen är en COM-komponent (dll) kompilerad från ett *ActiveX DLL–projekt* i *Microsoft Visual Basic 6.0*.

4.4.2.2 Microsoft Visual Studio .NET

Serverapplikationen är kompilerad från ett *Web Service –projekt* (webbtjänst) i *Microsoft Visual Basic .NET*. För att rationalisera implementeringen är webbtjänsten endast en omslutning kring COM-komponenten gjord i *Microsoft Visual Basic 6.0*. Detta innebär att klientapplikationen anropar webbtjänsten som i sin tur anropar COM-komponenten. Detta gäller dock inte ping-funktionen eftersom denna funktion körs direkt på webbtjänsten vilket innebär att upplägget med att webbtjänsten omsluter COM-komponenten inte har något inflytande på de mätningar som redovisas i kapitel 5. Inga mätningar görs på funktionen att skicka e-post.

Anropet från webbtjänsten till serverapplikationen gjord i *Microsoft Visual Studio 6.0* sker med COM i det här fallet eftersom dessa finns på samma maskin.

5 Prestandamätning

I detta kapitel redovisas den andra delen av jämförelsen, vilken grundar sig på exempelimplementationen som beskrivits i föregående kapitel.

För att mäta prestanda använder vi implementerade ping-funktioner. En ping-funktion har ingen uppgift förutom att låta klientapplikationen aktivera serverapplikationen, alternativt anropa den om aktiveringen redan är gjord, och mäta tiden som aktiveringen/anropet tar. Mätningen gick till så att vi distribuerade ut de två serverapplikationerna, en implementerad i DCOM-tekniken och en i webbtjänsttekniken, på en maskin som fick fungera som servermaskin, och klientapplikationen på en annan maskin för att fungera som klientmaskin. Nätet som sammanband dessa maskiner var av typen *Local Area Network (LAN)*. Vi gjorde separata mätningar för första anropet, d.v.s. aktiveringen av serverapplikationerna och efterföljande anrop. När vi skulle mäta aktiveringstid startade vi om servermaskinen mellan mätningarna för att undvika eventuellt sparad information om serverapplikationen som testades. En omstart garanterade då att aktiveringen blev ny. Vi gjorde också mätningar på minnesåtgången för det aktiverade objektet på servermaskinen, d.v.s. hur mycket minne det aktiverade objektet krävde. Minnesåtgången såg vi som en differens före och efter aktiveringen i *Task Manager* i *Windows*. På servermaskinen stängde vi ner så många processer som vi kunde för att mätningen skulle bli så bra som möjligt. Eftersom nätverket som användes är switchat, d.v.s. varje maskin ligger på en egen switch, antar vi att fördröjningar p.g.a. belastning av nätet är minimalt.

Om inget annat anges är mätningarna utförda med tidtagningsfunktioner tillhandahållna av utvecklingsmiljön som används. I dessa fall redovisas resultatet av mätningen i klientapplikationen vilket framgår av det grafiska gränssnittet av klientapplikationen i kapitel 4.3.3.

5.1 DCOM-anrop

I tabell 1 och 2 finns resultatet av mätningarna.

Tabell 1: DCOM-teknik: Första anrop/aktivering av serverapplikationen.

Mätning	Tid (ms)	Minnesåtgång (kilobytes)
1	2173	3960
2	1592	3864
3	2604	5520
4	1993	5496
5	1993	5484
Medelvärde	2071	4864

Tabell 2: DCOM-teknik: Efterföljande anrop. Objektet är aktiverat och finns både före och efter anropet. Att mäta minnesåtgången har därför ingen relevans.

Mätning	Tid (ms)
1	30
2	20
3	20
4	20
5	10
Medelvärde	20

5.2 Webbtjänstanrop

För att anropa webbtjänsten används *Microsoft SOAP Toolkit 2.0* vars anropssätt är lite speciellt. Innan själva anropet till ping-metoden hämtas WSDL-dokumentet. Vi mäter här två olika tider, dels tiden för att hämta detta WSDL-dokument (WSDL-tid) och dels tiden för att utföra metodanropet (pingtid). Det blir här svårt att separera dessa två när det gäller aktiveringen. Vi approximerar aktiveringstiden att vara summan av dessa två tider. I tabell 3, 4 och 5 finns mätningarna redovisade.

Tabell 3: Webbtjänstteknik: Första anrop/aktivering av serverapplikationen via klientapplikationen

Mätning	**WSDL-tid (ms)**	**Pingtid (ms)**	**Tid (ms)**	**Minnesåtgång (kilobytes)**
1	9294	1522	10816	13880
2	9373	1532	10905	13834
3	9284	1532	10816	13880
4	9314	1532	10846	13908
5	9283	1622	10905	13840
Medelvärde	9310	1548	10858	13868

Tabell 4: Webbtjänstteknik: Efterföljande anrop med klientapplikationen. Objektet är aktiverat och finns både före och efter anropet. Att mäta minnesåtgången har därför ingen relevans.

Mätning	WSDL-tid (ms)	Pingtid (ms)	Tid (ms)
1	60	181	241
2	70	170	240
3	70	181	251
4	70	211	281
5	70	201	271
Medelvärde	68	189	257

Tabell 5: Webbtjänstteknik: Första anrop/aktivering av serverapplikationen med HTTP Get

Mätning	Pingtid(s)	Minnesåtgång (kilobytes)
1	12	13476
2	10	13580
3	10	11900
4	10	13448
5	10	13540
Medelvärde	10	13189

Denna mätning är gjord med stoppur när ett HTTP Get görs direkt mot webbtjänsten via en webbläsare. Tiden som mäts är den mellan att URL:en anges och resultatet presenteras i webbläsaren. Att utföra mätningar av efterföljande anrop med stoppur är inte möjligt här. Pingtiden anges här i sekunder eftersom noggrannheten är mindre med stoppur.

5.3 Resultat

Vi sammanfattar värdena i en tabell. Alla värden är medelvärden.

Tabell 6: Sammanfattning över prestandatest

Anropssätt	**Första anrop/aktivering (ms)**	**Efterföljande anrop (ms)**	**Minnesåtgång (kilobytes)**
DCOM	2071	20	4864
Webbtjänst via applikation	10858	257	13868
Webbtjänst via webbläsare	10000	-	13189

Detta indikerar att aktivering av serverapplikationen i webbtjänsttekniken tar betydligt längre tid än i DCOM-tekniken. En skillnad på mellan 5-10 sekunder måste anses som lång tid vid ett metodanrop. För efterföljande anrop gäller samma sak, att DCOM-tekniken är snabbare. Relativt sett fördubblas tidsskillnaden mellan dem från 5 till 10 gånger mer för anropet med webbtjänsttekniken. Minnesåtgången ser vi också skiljer dem åt. DCOM-tekniken kräver nästan en tredjedel av vad webbtjänstanropet gör. Att anropet via webbläsaren är snabbare än via klientapplikationen kan förklaras genom att inget WSDL-dokument behöver läsas.

Sammanfattningsvis kan man säga att prestandamätningen klart indikerar att prestandan för DCOM-tekniken är bättre än för webbtjänsttekniken.

6 Slutsatser

Att byta från DCOM-tekniken till webbtjänsttekniken kan inte anses som trivialt men kan ändå vara nödvändigt inom en snar framtid. Genom denna utredning har vi berikat våra kunskaper kring om ett byte skulle vara aktuellt för Pronyx Sweden AB.

En fördel som webbtjänsttekniken medför är enkelheten att distribuera och anropa en metod via Internet. Detta är i många avseenden bra men i dagsläget är det inte alla kunder som har tillgång till Internet vilket gör att vi inte tycker att en snabb övergång till webbtjänsttekniken är aktuell.

Bland andra fördelar med webbtjänsttekniken kan nämnas textbaserade kommunikationsprotokoll och gränssnitt vilket resulterar i plattformsoberoende och skalbarheten.

Några av nackdelarna med webbtjänsttekniken är indikationen på sämre prestanda och möjligtvis att transaktionshanteringen är begränsad. Man bör tänka över vikten av dessa aspekter, speciellt om försämrad prestanda kommer att ha stor påverkan för systemen som tillverkas.

Fördelen med att kunna uppgradera webbtjänster på företagets egna servrar från kontoret och ändå ha kompatibla klientapplikationer ute hos kund kan däremot vara en avgörande faktor som gör att företaget i framtiden bör byta till webbtjänsttekniken.

Efter att Pronyx Sweden AB tagit del av resultatet från prestandamätningarna, som tyder på att DCOM-tekniken innehar fördelen med att vara både snabbare och mindre minneskrävande, måste man ta ställning till om skillnaden mellan tiderna och minnesåtgången är för stora eller om värdena för webbtjänsttekniken är acceptabla.

Om Pronyx Sweden AB kommer fram till att prestandaförsämringen kan accepteras anser vi att företaget gradvis skulle kunna gå över till webbtjänsttekniken genom att börja med att ha webbtjänster på lokala nätverk. Då skulle man kunna ta del av den positiva aspekten att slippa göra installationer på varje maskin när serverapplikationen ändras men ändå slippa säkerhetsproblemen med Internet. När säkerheten över Internet också har beaktats och kan försäkras skulle dessa serverapplikationer, utan att behöva ändras, kunna göras tillgängliga via nätet. Övergången till webbtjänsttekniken på lokala nätverket skulle kunna påbörjas redan idag.

7 Referenser

[1] *DCOM*. Microsoft Corporation. http://www.microsoft.com/com/tech/dcom.asp. 1998-03-30 (senaste uppdatering)

[2] Binh, Ly. Friendship *Building & COM*, http://www.techvanguards.com/com/concepts/friendliness.asp. 2001-06-30 (senaste uppdatering)

[3] Dr. GUI on Components, COM and ATL. Microsoft Corporation. http://www.microsoft.com/com/news/drgui.asp. 1998

[4] Eddon, Guy och Eddon, Henry. *Inside Distributed COM*. Microsoft Press. 1998

[5] Chmielewski, Jan. *COM+ Programming With C++ and ATL: Hands-On (Course 406)*. LEARNING TREE INTERNATIONAL. 1999

[6] Balena, Francesco. *Binary compatibility in VB*, http://www.comdeveloper.com/articles/binarycomp.asp. 1999-08-21

[7] Cornes, Ollie, Goode, Chris, Llibre, Juan T. Ullman, Chris, Birdwell, Rob, Kauffman, John, Krishnamoorthy, Ajoy, Miller, Christopher L, Raybould, Niel och Sussman, Dave. *Beginning ASP.NET using VB.NET*. Wrox Press Ltd. 2001

[8] Bray, Tim, Paoli, Jean och Sperberg-McQueen, C. M. *Extensible Markup Language (XML) 1.0*. (Översättningsarbete: Erik Mjöberg, Bearbetning: Gustaf Liljegren och Jan Östberg.) http://www.xml.se/xml/REC-xml-19980210-sv.html. 1998-02-10

[9] Liljegren, Gustaf. *Vad är XML?* http://www.xml.se/xml/vad.html. 2001-07-30

[10] Liljegren, Gustaf. *Varför XML?* http://www.xml.se/xml/varfor.html. 2002

[11] Bray, Tim, Hollander, Dave och Layman, Andrew. *Namespaces in XML*. W3C. http://www.w3.org/TR/REC-xml-names/, 1999-0114

[12] *SOAP*. Microsoft Corporation. http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000523, 2002

[13] Gudgin, Martin, Hadley, Marc, Moreau, Jean-Jacques och Frystyk Nielsen, Henrik, *SOAP Version 1.2*. W3C. http://www.w3.org/TR/2001/WD-soap12-20010709, 2001-07-09

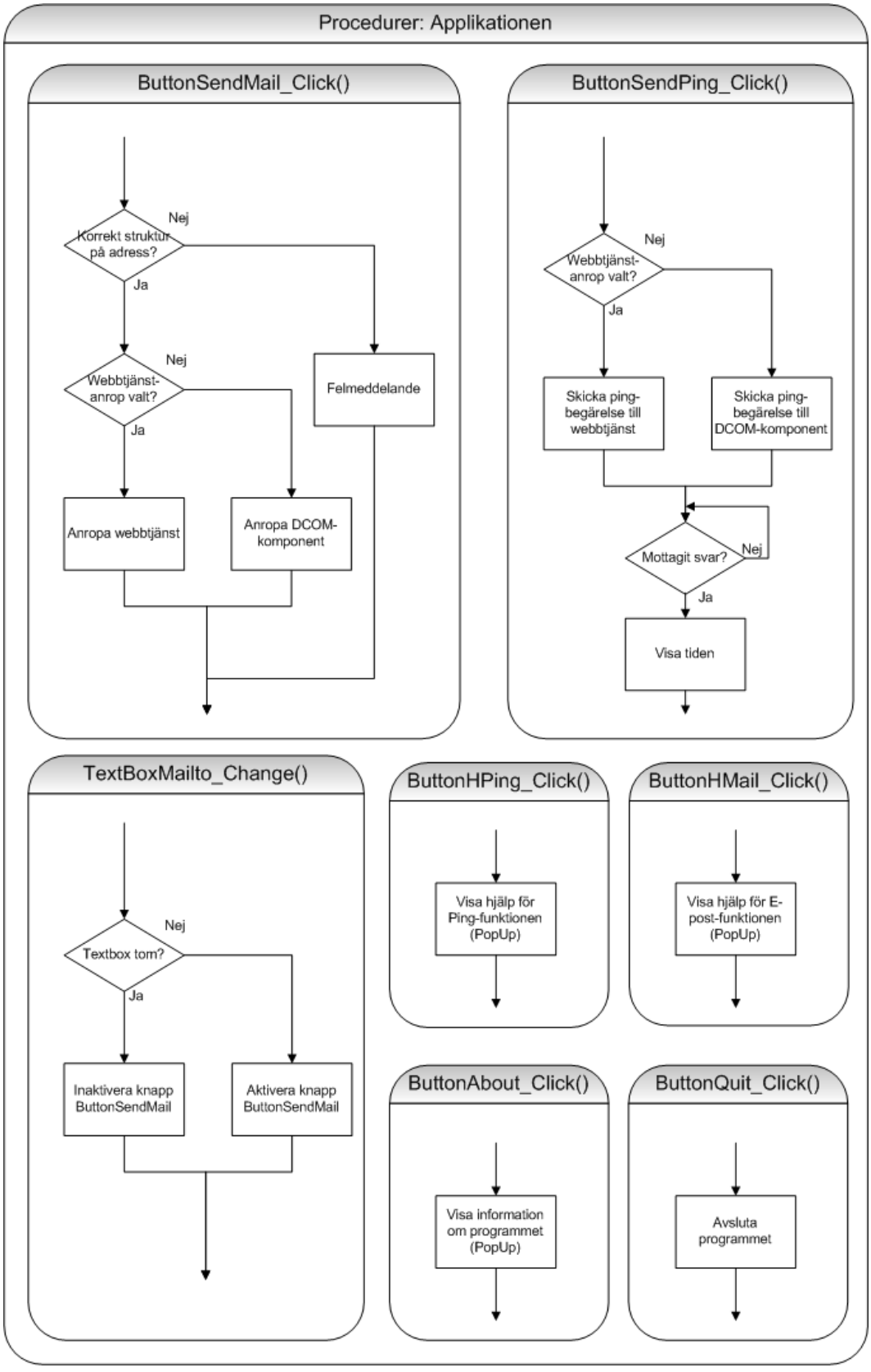
[14] Gudgin, Martin, Hadley, Marc, Moreau, Jean-Jacques och Frystyk Nielsen, Henrik, *SOAP Version 1.2 Part 2: Adjuncts*. W3C. http://www.w3.org/TR/2001/WD-soap12-part2-20011002, 2001-10-02

[15] Christensen, Erik, Curbera, Francisco, Meredith, Greg, Weerawaranam, Sanjiva. *Web Services Description Language (WSDL) 1.1*. W3C http://www.w3.org/TR/2001/NOTE-wsdl-20010315. 2001-03-05

[16] McKee, Barbara, Ehnebuske, Dave och Rogers, Dan. *UDDI Version 2.0 API Specification*. http://uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.doc. 2001-06-08

[17] *UDDI Technical White Paper*. Interational Business Machines Corporation och Microsoft Corporation. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf. 2000-0906.

A Bilaga: Flödesscheman för applikationen



B Bilaga: Kod för webbtjänstexemplet *Summa.*

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    Web Services Designer Generated Code

    <WebMethod()> Public Function Summa(ByVal term1 As Long, ByVal term2 As Long) As Long
        Summa = term1 + term2
    End Function

End Class
```

C Bilaga: SOAP-protokoll

C.1 SOAP request

POST /SummaWebService/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: **length**
SOAPAction: "http://tempuri.org/Summa"

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Summa xmlns="http://tempuri.org/">
      <term1>long</term1>
      <term2>long</term2>
    </Summa>
  </soap:Body>
</soap:Envelope>
```

C.2 SOAP response

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: **length**

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SummaResponse xmlns="http://tempuri.org/"><SummaResult>long</SummaResult>
    </SummaResponse>
  </soap:Body>
</soap:Envelope>
```

D Bilaga: WSDL-dokument

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://tempuri.org/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://tempuri.org/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified"
        targetNamespace="http://tempuri.org/">
      - <s:element name="Summa">
        + <s:complexType>
            <s:element minOccurs="1" maxOccurs="1"
              name="term1" type="s:long" />
            <s:element minOccurs="1" maxOccurs="1"
              name="term2" type="s:long" />
          </s:sequence>
        </s:element>
      - <s:element name="SummaResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="1" maxOccurs="1"
                name="SummaResult" type="s:long" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:element name="long" type="s:long" />
      </s:schema>
    </types>
  - <message name="SummaSoapIn">
      <part name="parameters" element="s0:Summa" />
```

```
  </message>
- <message name="SummaSoapOut">
    <part name="parameters" element="s0:SummaResponse" />
  </message>
- <message name="SummaHttpGetIn">
    <part name="term1" type="s:string" />
    <part name="term2" type="s:string" />
  </message>
- <message name="SummaHttpGetOut">
    <part name="Body" element="s0:long" />
  </message>
- <message name="SummaHttpPostIn">
    <part name="term1" type="s:string" />
    <part name="term2" type="s:string" />
  </message>
- <message name="SummaHttpPostOut">
    <part name="Body" element="s0:long" />
  </message>
- <portType name="Service1Soap">
  - <operation name="Summa">
      <input message="s0:SummaSoapIn" />
      <output message="s0:SummaSoapOut" />
    </operation>
  </portType>
- <portType name="Service1HttpGet">
  - <operation name="Summa">
      <input message="s0:SummaHttpGetIn" />
      <output message="s0:SummaHttpGetOut" />
    </operation>
  </portType>
- <portType name="Service1HttpPost">
  - <operation name="Summa">
      <input message="s0:SummaHttpPostIn" />
      <output message="s0:SummaHttpPostOut" />
    </operation>
  </portType>
```

```
- <binding name="Service1Soap" type="s0:Service1Soap">
    <soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
  - <operation name="Summa">
      <soap:operation soapAction="http://tempuri.org/Summa"
        style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
- <binding name="Service1HttpGet" type="s0:Service1HttpGet">
    <http:binding verb="GET" />
  - <operation name="Summa">
      <http:operation location="/Summa" />
    - <input>
        <http:urlEncoded />
      </input>
    - <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
- <binding name="Service1HttpPost" type="s0:Service1HttpPost">
    <http:binding verb="POST" />
  - <operation name="Summa">
      <http:operation location="/Summa" />
    - <input>
        <mime:content type="application/x-www-form-
          urlencoded" />
      </input>
    - <output>
        <mime:mimeXml part="Body" />
```

```
        </output>
      </operation>
    </binding>
- <service name="Service1">
  - <port name="Service1Soap" binding="s0:Service1Soap">
      <soap:address
        location="http://localhost/SummaWebService/Service1.a
        smx" />
    </port>
  - <port name="Service1HttpGet" binding="s0:Service1HttpGet">
      <http:address
        location="http://localhost/SummaWebService/Service1.a
        smx" />
    </port>
  - <port name="Service1HttpPost" binding="s0:Service1HttpPost">
      <http:address
        location="http://localhost/SummaWebService/Service1.a
        smx" />
    </port>
  </service>
</definitions>
```