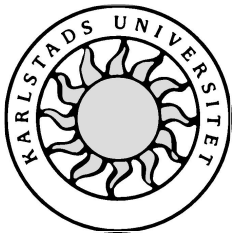


Datavetenskap

Patrik Engström

Ronny Hägerman

Tillämpning av COM och MVC för Realisering av Distribuerade Grafiska Gränssnitt

---

Examensarbete, C-nivå

2003:24

Tillämpning av COM och MVC för Realisering av Distribuerade Grafiska Gränssnitt

Patrik Engström

Ronny Hägerman

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.

Patrik Engström

Ronny Hägerman

Godkänd, 2003-06-04

Handledare: Hannes Persson

Examinator: Stefan Alfredsson

Sammanfattning

Denna uppsats redogör för skapandet av en laboration inom området grafiska användargränssnitt. Då laborationen bygger på COM-teknologi ges en översikt över denna teknologi och även de närbesläktade begreppen DCOM, ActiveX och COM+. Vidare beskrivs utvecklingsmiljöerna Visual Basic och Visual C++ med fokus på de delar som inverkat på implementationen. En sammanfattning över konstruktionsmönstret Model-View-Controller och hur detta koncept genomsyrar implementationen ges. Principer för konstruktionslösningen redovisas och då olika alternativ varit aktuella ges en motivering till det aktuella valet. En beskrivning över implementationen ges med fokus på de punkter som är centrala för förståelsen av denna. De båda utvecklingsmiljöerna Visual Basic och Visual C++ jämförs utifrån de erfarenheter som erhållits under utvecklandet av den aktuella laborationen. Slutligen innehåller uppsatsen även en handledning för de studenter som är tänkta att utföra laborationen.

Realizing distributed graphical interfaces by applying COM and MVC

Abstract

This paper describes the creation of a lab assignment within the area of graphical user interfaces. Since this laboratory work is based on COM-technology an overview of this technology is given, including the related technologies DCOM, ActiveX and COM+. Further the development environments Visual Basic and Visual C++ is described, focusing on those parts influencing the implementation. A summary of the design pattern Model-View-Controller is given with an explanation of it's significance for the implementation. The design principles are described and when several alternatives have been considered, a motivation of the final choice is given. A description of the implementation is given, focusing on those parts central for the understanding of the implementation. The two development environments Visual Basic and Visual C++ are compared based upon the experiences obtained during the development of the laboratory work. Finally this work contains instructions for the students intended to perform the laboratory work.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Introduktion

Inom ämnet datavetenskap vid Karlstads universitet ges en kurs i grafiska användargränssnitt. I kursen ingår ett antal laborationer som obligatoriskt moment. Examensarbetets mål är att utveckla ett underlag till en ny laboration för denna kurs. Laborationen skall delvis baseras på samma princip som en redan existerande laboration, men med den skillnaden att en annan teknik och en annan utvecklingsmiljö skall användas.

1.1 Problemformulering

Den befintliga laborationen syftar till att studenterna skall skriva ett grafiskt användargränssnitt till en operatörspanel i en rymdfärja. Själva logiken behöver studenterna inte skriva, däremot skall de utveckla det grafiska gränssnittet som återspeglar logiken.

Som nämndes ovan skall den nya laborationen baseras på en redan existerande laboration men bygga på annan teknik. Den befintliga laborationen är skriven i C++ med hjälp av utvecklingsmiljön Qt från det norska företaget TrollTech. För ytterligare information om detta företag se [27]. Den tänkta lösningen skall utvecklas i Microsoft Visual Basic (VB) och använda Component Object Model (COM) som beskrivs på [17]. I mån av tid skall även en lösning i Microsoft Visual C++ (VC++) med Microsoft Foundation Classes (MFC) utvecklas. Se [18] för en närmare beskrivning av MFC. Såväl VB som VC++ ingår i Microsoft Visual Studio. För mer information om Visual Studio, se [22].

Uppgiften är alltså att utveckla själva modellen, i detta fall en simulerad rymdfärja, med hjälp av ovanstående tekniker och utvecklingsverktyg. Dessutom skall ett grafiskt användargränssnitt som använder modellen konstrueras. Skapandet av detta användargränssnitt kan sägas motsvara den uppgift som skall utföras av studenterna. Modellen och tillhörande grafiska användargränssnitt skall baseras på konstruktionsmönstret Model-View-Controller (MVC).

1.2 Syfte

Syftet med uppsatsen är att utveckla en laboration i kursen grafiska användargränssnitt vid Karlstads universitet. Uppsatsen kan även komma att utgöra ett underlag för utveckling av laborationer i andra kurser inom datavetenskap. Vidare skall uppsatsen redogöra för

teknikerna COM, Distributed COM (DCOM), ActiveX och COM+ samt hur COM och DCOM används i Visual Basic och Visual C++.

1.3 Avgränsning

Uppsatsens syfte är inte att gå in på djupet i de olika COM-teknikerna; ActiveX och COM+ kommer endast att beröras ytligt. Det är inte heller ett mål att redogöra ingående för något av språken Visual Basic eller Visual C++.

1.4 Mål

Examensarbetet skall leda fram till följande:

- Väldokumenterad och välfungerande programkod, utformad så att den kan användas i de datasalar som disponeras av institutionen för informationsteknologi. Programkoden skall omfatta följande:
    - Modellen av rymdfärjan, skriven med hjälp av någon COM-teknik.
    - Grafiskt användargränssnitt, utgörande ett exempel på hur modellen kan användas.
- En användarmanual som beskriver hur ovanstående program fungerar.
- En laborationsspecifikation för studenter som kan komma att utföra laborationen.
- En uppsats som redogör för följande:
    - Teknikerna COM, DCOM , ActiveX samt COM+.
    - Användning av COM och DCOM i Visual Basic och Visual C++.
    - Händelsehantering i ovanstående språk.
    - Beskrivning av konstruktionsmönstret MVC.
    - Beskrivning av konstruktionslösningar i Visual Basic och Visual C++.
    - Jämförelse mellan konstruktionslösningen i Visual Basic och Visual C++.

1.5 Förutsättningar och krav

För att kunna genomföra ovanstående krävs följande programvara:

- Microsoft Visual Studio (Professional Edition) 6.0.
- Microsoft Windows NT 4 (eller senare).

1.6 Disposition

Kapitel 2 inleds med en kort introduktion till COM. Här beskrivs varför det finns ett behov för en teknik som COM, följt av en redogörelse av några tekniker som är baserade på COM, närmare bestämt DCOM, ActiveX och COM+. Kapitlet avslutas med en beskrivning av några för- och nackdelar med COM. Visual Basic och Visual C++ beskrivs i kapitel 3. Här beskrivs de delar av utvecklingsverktygen som är väsentliga för konstruktionslösningen, främst hur COM och händelsehantering implementeras. I början av kapitel 4 ges en introduktion till konstruktionsmönstret Model-View-Controller följt av en beskrivning av uppgiften som skall lösas i detta examensarbete. Kapitlet avslutas därefter med en beskrivning av den principiella konstruktionslösningen.

Kapitel 5 redogör för hur implementationen och testningen av konstruktionslösningen utförts. Såväl server- som klientdelen beskrivs. Vid beskrivningen av klientdelen ges dessutom en beskrivning av det grafiska användargränssnittet. Några av de aspekter som skiljer VB från VC++ beskrivs i kapitel 6. De slutsatser som uppsatsförfattarna kommit fram till under faktainsamlande, implementering och dokumentering beskrivs slutligen i kapitel 7.

2 COM

I detta kapitel beskrivs de olika COM-teknikerna. Kapitlet börjar med en allmän översikt som förklarar vad COM egentligen är, varför det skapades och en beskrivning av dess arkitektur. Därefter ges en beskrivning av DCOM följt av en kort genomgång av ActiveX och COM+. Avslutningsvis beskrivs några av teknikernas för- och nackdelar.

Syftet med kapitlet är att skapa förståelse för COM-tekniken som i senare kapitel kommer att användas i konstruktionslösningen.

Kapitlet innehåller inga kompletta kodexempel, sådana återfinns i kapitel 3 (*Visual Basic och Visual C++*) samt i bilaga B. Anledningen till detta är att skapandet och användandet av en COM-komponent ser mycket olika ut i dessa språk.

2.1 Översikt

COM står för Component Object Model och kan definieras på tre olika sätt enligt [9]:

- COM är en specifikation. Specifikationen finns tillgänglig på [17].
- COM är en filosofi för programvaruutveckling, som bygger på användande och återanvändande av mjukvarukomponenter.
- COM är en binär standard.

Med hjälp av COM är det möjligt att skapa komponenter som kan kommunicera med varandra oavsett vilket språk och utvecklingsmiljö dessa konstruerats i. Att COM är en binär standard innebär att komponenter som skapats med COM är kompatibla på binär nivå. Det är binärkompatibiliteten som möjliggör språkoberoendet.

COM är avsett att vara en plattformsoberoende standard men är i praktiken en teknologi som främst används på 32-bitars versioner av Windows. Microsoft har dock släppt en version för MacOS och andra företag har skapat versioner för Solaris, Unix och Linux, se [6].

2.2 Bakgrund

I detta delkapitel förklaras varför det finns ett behov av komponentteknologier och under vilka omständigheter COM utvecklades.

2.2.1 Komponenter

Innan det fanns komponentteknologier som COM att tillgå, kompilerades stora program genom att källkoden skickades till kompilatorn för kompilering. Denna process var tidsödande och skapade stora exekveringsfiler. En lösning på detta problem är komponentbaserad programmering, där binära komponenter produceras och används. I och med användandet av små binära komponenter förenklas programuppdateringar, då de binära komponenterna kan uppdateras och bytas ut oberoende av varandra. Mer om detta finns att läsa på [23].

2.2.2 Object Linking and Embedding (OLE)

OLE är en teknik för att länka och lägga in dokument skapade i ett annat program. Syftet med OLE var från början att bättre kunna integrera Microsofts Office-program med varandra. Exempelvis skulle en ordbehandlare kunna beordra ett kalkylprogram att öppna och skriva ut ett specifikt dokument. Första versionen av OLE (OLE1) var baserad på Dynamic Data Exchange (DDE), som tillåter olika tillämpningar att utbyta information med varandra. Dessvärre var DDE långsamt (se till exempel [23]) och kräver att de båda program som skall utbyta information redan har startats. Eftersom DDE inte var tillräckligt bra beslutade Microsoft att utveckla en ny infrastruktur för OLE version 2 (OLE2). Den nya infrastrukturen döptes till COM.

2.2.3 Syftet med COM

Inför utvecklandet av COM satte Microsoft upp fyra mål som skulle uppfyllas, detta enligt [23]:

- Binär återanvändbarhet. System uppbyggda av binära komponenter är enklare att konstruera, underhålla samt bygga ut.
- Objektorientering. Även tidigare återanvändes binär kod, men då i form av DLL (Dynamic Link Library) som saknade stöd för koncept som objektorientering.
- Språkoberoende. Det skall vara möjligt att skriva komponenter i valfritt språk. Komponenter skall kunna samverka med varandra även om de är skriva i olika språk.
- Kommunikation mellan processer. En viktig aspekt var att dölja komplexiteten vid denna typ av kommunikation. Ursprungligen kunde kommunikationen endast ske mellan processer på samma dator, men i kapitel 2.4 förklaras hur detta i och med DCOM utvecklades till att inkludera processer på olika datorer.

OLE2 och första versionen av COM lanserades 1993. Senare lanserades utökningar av COM som DCOM, ActiveX och COM+, dessa behandlas i kapitel 2.4, 2.5 respektive 2.6.

2.3 COM-arkitekturen

Detta delkapitel redogör för COM-arkitekturen, det vill säga hur COM är uppbyggt och fungerar. Inledningsvis ges en översikt över några vanliga begrepp, därefter beskrivs några olika sätt som COM-servrar kan exekvera på. Vidare förklaras det centrala begreppet interface samt hur ett interface kan definieras. Avsnittet beskriver även hur COM-objekt instansieras. Avslutningsvis ges en översikt av de problem som kan uppstå i samband med användandet av trådar tillsammans med COM. Detta kommer inte att ha någon större betydelse för den aktuella implementationen av rymdfärjemodellen, dock är det viktigt att ha vissa grundkunskaper inom detta för den som programmerar COM-komponenter.

2.3.1 Översikt

Generellt finns tre typer av objekt inom COM, se [25]:

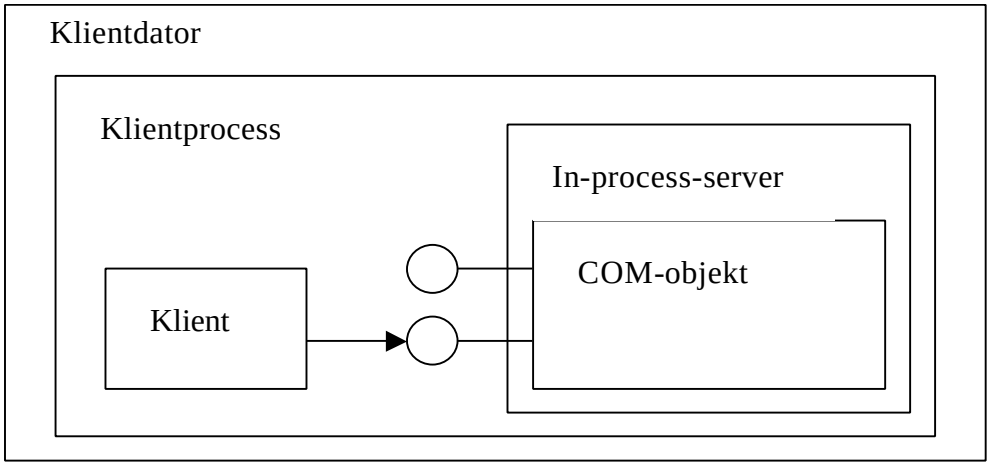
- Klient. En COM-klient är inget egentligt COM-objekt utan benämningen på en tillämpning som använder ett eller flera COM-objekt.
- Server. Varje COM-objekt är implementerat i en server, se [6]. En COM-server kan enligt [25] sägas bestå av tre element (dessa förklaras längre fram i kapitlet):
    - Interface
    - CoClass
    - Typbibliotek (type library)
- ActiveX-kontroll. En ActiveX-kontroll kan lite förenklat sägas vara en in-process COM-server med ett grafiskt användargränssnitt. Med in-process avses att servern exekverar i samma minnesutrymme som klienten, begreppet förklaras utförligare i nästa delkapitel. ActiveX beskrivs i delkapitel 2.5.

2.3.2 Servertyper

En COM-server kan exekvera på tre olika sätt: in-process, out-of-process och remote. Dessa beskrivs i tur och ordning nedan.

2.3.2.1 In-process

Med in-process avses en server som exekverar i samma minnesutrymme som den klient som använder servern. In-process är den enklaste typen av COM-server. Eftersom server och klient delar minnesutrymme kan kommunikationen mellan de båda ske på ett enkelt sätt (pekarparametrar medför till exempel inga problem). En COM-server som är av typen in-process implementeras i en DLL-fil. Figur 2.1 illustrerar hur en klient använder ett COM-objekt genom ett av dess två interface. Begreppet interface beskrivs i delkapitel 2.3.3.



Figur 2.1: In-process COM-server

2.3.2.2 Out-of-process

En server som är av typen out-of-process körs till skillnad från en in-process-server inte i samma minnesutrymme som klienten. Komponenten är implementerad som en exekverbar fil som körs i form av en helt egen process. Eftersom server och klient inte delar minnesutrymme kan kommunikationen mellan dem inte ske direkt på samma sätt som vid en in-process-server. Denna typ av kommunikation påminner om remote-varianten som beskrivs nedan.

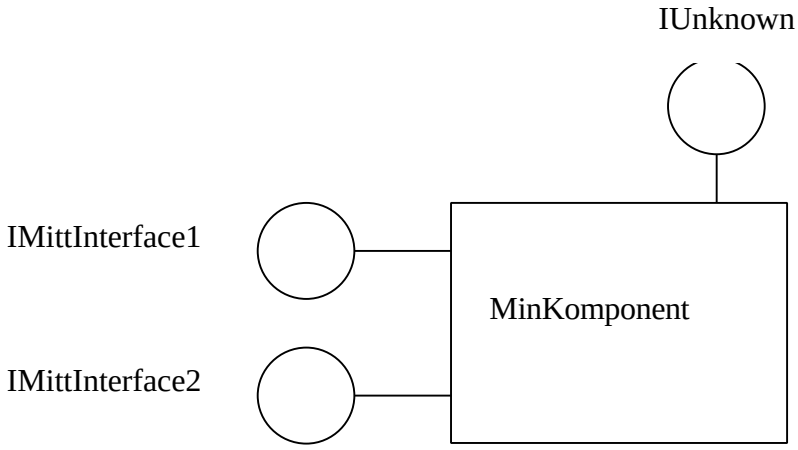
2.3.2.3 Remote

DCOM möjliggör en tredje servertyp: remote. En COM-server av denna typ är en out-of-process-server som dessutom lagras och exekveras på en annan dator än klienten. Mer om denna typ av kommunikation återfinns tillsammans med DCOM i kapitel 2.4.

2.3.3 Interface

Begreppet interface är centralt då det gäller COM. Ett interface kan liknas vid en headerfil i C++, all information som omvärlden skall få om en komponent skall finnas här och den

information som omvärlden inte behöver känna till skall hållas utanför. Namngivningskonventionen är sådan att ett interfacenamn alltid bör starta på ”I” följt av ett beskrivande namn. En komponent och dess interface brukar illustreras som i figur 2.2.



Figur 2.2: COM-komponenten MinKomponent med tillhörande interface

En komponents interface får aldrig ändras efter det att den har lanserats. Detta av en enda anledning - bakåtkompabilitet. Föreligger ett behov av att ändra eller lägga till något måste ett nytt interface skapas. Därigenom kan de klienter som redan använder objektet använda det gamla interfacet medan nya klienter kan använda det nya uppdaterade interfacet. [26] anger följande ändringar som exempel på tillfällen då ett nytt interface kan behöva skapas:

- Ändrat antal funktioner.
- Ändrad ordning av funktionerna.
- Antalet parametrar i en funktion.
- Ändrad ordning av en funktions parametrar.
- Ändrad parametertyp.
- Möjliga returvärden från en funktion.
- Ändrad typ av returvärde.
- Ändrat syfte med en funktion.
- Ändrat syfte med en parameter.

En komponent kan implementera flera interface, i praktiken alltid minst det fördefinierade interfacet IUnknown och ett interface som komponentens skapare definierar. Ett annat mycket

vanligt interface är IDispatch. De båda interfacen IUnknown och IDispatch beskrivs i de nästkommande två avsnitten.

De funktioner som deklareras i ett interface måste vara uppbyggda på ett särskilt sätt eftersom de är avsedda att anropas utifrån, av objekt som kan vara skrivna i ett helt annat programmeringsspråk, i avsnittet ges därför en kort beskrivning av hur funktioner bör vara utformade avseende överföring av parametrar och returvärden. Därefter ges en beskrivning av hur en COM-komponent kan anropa funktioner hos sina klienter genom att använda ett utgående interface. Avsnittet avslutas med en beskrivning av Interface Definition Language (IDL) som är ett språk för att definiera interface, samt hur filer i detta språk är uppbyggda.

2.3.3.1 Interfacet IUnknown

Alla COM-komponenter måste alltid implementera interfacet IUnknown. IUnknowns uppgift är att ge klienterna uppgifter om vilka övriga interface som finns och att hålla reda på hur många klienter som använder komponenten. IUnknown innehåller tre metoder:

- QueryInterface
- AddRef
- Release

QueryInterface lämnar information om vilka interface som finns tillgängliga. AddRef och Release används när en klient vill använda respektive inte längre behöver använda ett specifikt interface. Alla övriga interface ärver (direkt eller indirekt) IUnknown.

2.3.3.2 Interfacet IDispatch

IDispatch är ett annat viktigt interface. Det finns till för att tillåta scriptspråk (t.ex. Visual Basic) att använda en komponent. IUnknown är nämligen utvecklat främst för Visual C++. IDispatch behöver inte implementeras om komponenten inte skall användas tillsammans med scriptspråk. IDispatch innehåller följande metoder:

- GetTypeInfoCount
- GetTypeInfo
- GetIDsOfNames
- Invoke

Invoke ger tillgång till de metoder och egenskaper som definierats i ett interface medan de tre övriga ger tillgång till information om objektet/interfacet.

2.3.3.3 Returvärden

Alla metoder som definieras i ett interface bör returnera ett värde av typen HRESULT (IUnknown::AddRef och IUnknown::Release är undantag från denna regel). Ett HRESULT innehåller ett antal datafält och kan sägas lagra information om huruvida något gick snett och i så fall vad. I tabell 2.1 listas de vanligaste returkoderna (enligt [26]). Eftersom denna returtyp endast kan användas för att förmedla värden av typen sant/falskt måste utparametrar användas istället för returvärden då andra typer önskas, mer om detta i nästa avsnitt.

Returkod	Innebörd
S_OK	Funktionen lyckades och returnerade sant. S_OK är definierat som 1.
NOERROR	Samma som S_OK
S_FALSE	Funktionen lyckades och returnerade falskt. S_FALSE är definierat som 0.
E_UNEXPECTED	Oväntat fel.
E_NOTIMPL	Funktionen har inte implementerats.
E_NOINTERFACE	Komponenten stödjer inte det efterfrågade interfacet. Kan returneras av QueryInterface.
E_OUTOFMEMORY	Komponenten kunde inte allokera tillräckligt med minne.
E_FAIL	Ospecifierat fel

Tabell 2.1: Returkoder

2.3.3.4 Parametrar

Parametrar till en funktion kan vara av tre olika slag:

- In
- Ut
- In-ut

Det går även att deklarera en parameter som ”[out, retval]”. Anledningen till detta är att utparametern då kan användas som ett traditionellt returvärde i enklare språk som VB.

Vid val av returtyper bör en viss försiktighet iakttas. Om komponenten kommer att användas av tillämpningar skrivna i andra språk är det inte säkert att dessa språk är överens om datatypens uppbyggnad. Om detta är fallet bör endast vissa typer som garanterar kompatibilitet användas.

För en out-of-process-server bör det även beaktas att referensparametrar måste kopieras vilket kan innebära försämrad prestanda och problem med att överföra komplicerade pekarstrukturer.

2.3.3.5 Utgående interface - förbindelsepunkter

Det normala är att en klient/server-modell använder sig av synkrona funktionsanrop, så är också fallet när det gäller COM. Med synkrona anrop menas att klienten anropar en serverfunktion och därefter avvaktar tills funktionen utförts, först när svaret kommit fortsätter klienten sin exekvering. Ibland är detta inte önskvärt, vid en krävande beräkning kan det t.ex. vara att föredra att låta klienten fortsätta sin exekvering direkt för att istället bli underrättad då beräkningen slutförts. Vid sådan asynkron kommunikation brukar benämningen callback användas.

Att implementera callbacks inom COM är tyvärr inte helt enkelt, servern har normalt sett inget sätt att meddela klienten, utan all kommunikation sker på klientens initiativ. Ett sätt att lösa detta vore att skapa även klienten som en COM-komponent, men det vanliga är att connection points – förbindelsepunkter används.

En förbindelsepunkt upprättas genom att COM-komponenten deklarerar ett särskilt interface för detta. Det speciella med detta interface är att det är ett utgående interface, det implementeras inte av komponenten själv utan av de klienter som önskar bli underrättade då händelser inträffar i komponenten. En klient som vill bli underrättad då en händelse inträffar i COM-komponenten måste alltså själv implementera de metoder som deklarerats i interfacet. Förutom att implementera interfacet måste klienten anropa en funktion hos servern för att registrera att den önskar bli underrättad om dessa händelser (samt avregistrera sig på motsvarande sätt). Servern i sin tur håller reda på alla klienter som registrerat sig för att bli underrättade om eventuella händelser och loopar igenom alla dessa då en händelse inträffar. Detta förfarande innebär alltså att servern anropar funktioner hos sina klienter vilket i sig kan

innebära en viss risk: om en klient fastnar i en evighetsloop under exekveringen av funktionen kommer servern att fortsätta vänta förgäves.

Mekanismen med förbindelsepunkter kommer att ha stor betydelse för implementationen av laborationen eftersom det föreligger ett behov av att låta servern ta initiativ till kommunikation med sina klienter.

2.3.3.6 Interface Definition Language

För att definiera ett interface används IDL. I en IDL-fil definieras de tre nyckeldelarna hos en COM-komponent:

- Interface
- Typbibliotek. Ett typbibliotek är en kompilerad IDL-fil som innehåller information om en komponents interface. Genom att använda ett typbibliotek kan tillämpningar få besked om vilka interface en komponent stödjer och namnen på medlemmarna (funktionsnamn etc.) i dessa interface.
- CoClass. En CoClass (Component Class, komponentklass) är en klass där implementationen av de metoder som definierats i interfacen sker. En COM-komponent kan ha en eller flera komponentklasser.

Ofta föreligger inget behov att modifiera IDL-filer manuellt eftersom utvecklingsmiljön kan generera och underhålla dessa automatiskt. För att kunna programmera med COM bör ändå programmeraren besitta en viss kunskap om dessa filer och hur de används.

För att kompilera IDL-filer kan kompilatorn Microsoft IDL (MIDL) användas. Kompilering kan göras manuellt med kommandot *midl filnamn.idl* men görs normalt automatiskt då tillämpningen som använder filen kompileras. Det bör nämnas att Microsofts version av IDL inte följer den IDL-standard som utvecklats av Open Software Foundation (OSF) utan är en utökad version.

2.3.3.7 Uppbyggnad av IDL-filer

I bilaga B.1 återges ett exempel på innehållet i en IDL-fil. Exemplet är delvis byggt på exempel från [25]. Kommentarer i en IDL-fil görs på traditionellt C-sätt (*/*kommentar*/*). Attribut står före det element de refererar till och hålls samman med hakparanteser.

I exemplet definieras IMittObj som ärver från IDispatch (och därmed också IUnknown). Egenskapen "object" talar om för kompilatorn att IMittObj är ett COM-interface och inte ett

interface för Remote Procedure Call (RPC). Strängen efter ”uuid” är ett GUID-nummer (mer om GUID i kapitel 2.3.5), i exemplet finns tre sådana, ett för interfacet IMittObj, ett för komponentklassen MinCoClass och ett för typbiblioteket. ”Dual” anger att det finns två sätt att komma åt de egenskaper och metoder interfacet definierar (i praktiken att interfacet implementerar både IUnknown och IDispatch). ”Helpstring” används enbart för att lagra ett namn som är lättare för en mänsklig hjärna att komma ihåg. ”Import” används för att hämta in innehållet från en annan fil (jämför #include i C++). Exemplets enda interface (IMittObj) har en metod (utöver de som ärvts från IDispatch), MinMetod. Metoden tar emot fyra parametrar och returnerar ett värde av typen HRESULT. Typbiblioteket i exemplet har också en egenskap för att lagra ett versionsnummer.

Då exemplet i bilaga B.1 sparas som exempel.idl och kompileras kommer följande filer att skapas:

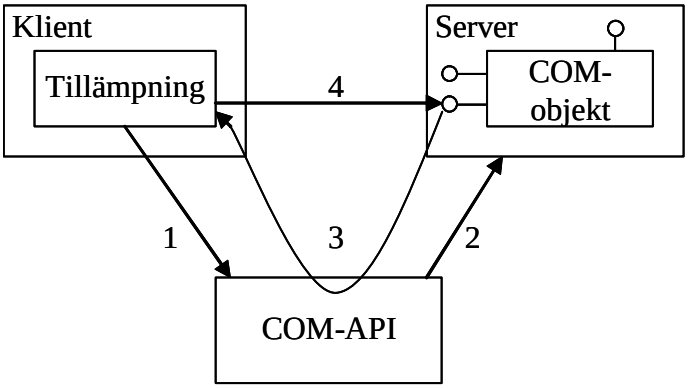
- dlldata.c (implementerar DLL:en som innehåller proxy/stub)
- exempel.h (innehåller deklarationer för komponentens interface)
- exempel.tlb (en binär version av IDL-filen)
- exempel_i.c (lagrar GUID-definitioner)
- exempel_p.c (implementerar proxy/stub)

Vilka av dessa filer som senare kommer till användning kan skilja sig åt beroende på utvecklingsmiljön.

2.3.4 Instansiering av COM-objekt

När en klient behöver använda ett COM-objekt går det i korthet till på följande sätt (enligt [6]), vilket även illustreras i figur 2.3:

1. Klienten anropar COM Application Programming Interface (API).
2. COM lokaliserar objektet med hjälp av systemregistret och startar en serverprocess. Hur lokaliseringen går till beskrivs nedan i avsnittet *Globally Unique Identifier*.
3. Serverprocessen skapar det efterfrågade objektet och returnerar en interface-pekare. Konceptet med interface-pekare och funktionstabeller beskrivs i avsnittet *Virtuell funktionstabell*.
4. Klienten och servern kan kommunicera med varandra genom interface-pekaren.



Figur 2.3: Instansiering av COM-objekt

Avsnittet redogör även kort för de två begreppen Class factories och Monikers som inte kommer att ha någon betydelse för implementationen av rymdfärjemodellen men som är centrala begrepp då det gäller instansiering av COM-objekt.

2.3.4.1 Globally Unique Identifier

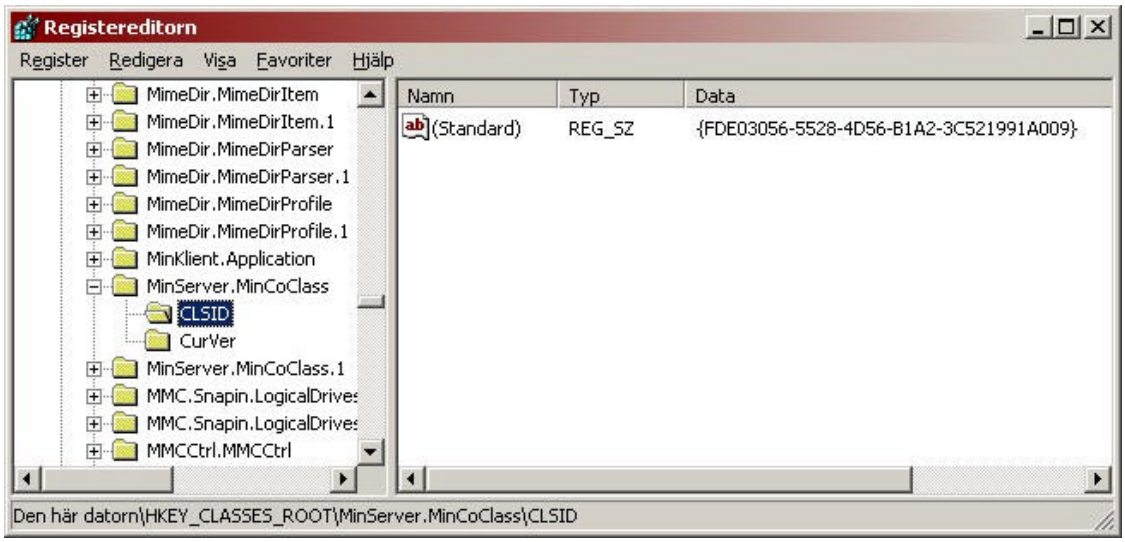
För att identifiera en COM-komponent finns Globally Unique Identifier (GUID). Detta är en 128-bitars datatyp som lagrar ett unikt värde med vars hjälp en komponent kan identifieras. För att Windows skall kunna hitta en komponent måste dess GUID vara registrerat i systemregistret. Figur 2.4 illustrerar hur en registrerad komponent representeras i systemregistret.

GUID-värden genereras utifrån nätverkskortets MAC-adress och tiden då värdet skapades. (Om den aktuella datorn saknar nätverkskort slumpas istället en adress fram, risken att två komponenter skulle få samma GUID är ändå försumbar.)

I systemregistret lagras värden som strängar på den hexadecimala formen 336C8C70-A62B-11D0-AD5F-00AA00A219AA.

Förkortningarna kan för den oinvigde tyckas vara något förvillande, ett annat namn för GUID är Universally Unique Identifier (UUID). Dessutom finns Interface Identifier (IID), CoClass Identifier (CLSID) och Library Identifier (LIBID) för att definiera interface, komponentklasser respektive typbibliotek. Alla dessa är olika typer av GUID.

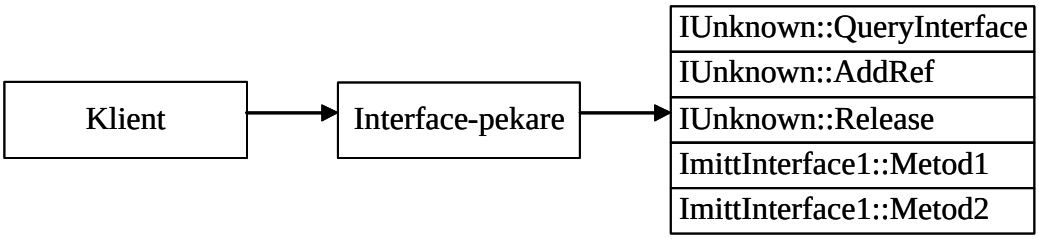
För att manuellt generera ett GUID kan något av Microsofts verktyg uuidgen.exe eller guidgen.exe användas.



Figur 2.4: Lagring av GUID i Windows systemregister

2.3.4.2 Virtuell funktionstabell

Alla metoder och egenskaper hos en komponent nås via den virtuella funktionstabellen, vtable (eller VTBL). Denna tabell innehåller en mängd pekare till komponentens funktioner. Andra program har dock inte fri åtkomst till denna tabell, utan denna tabell nås via en s.k. interface-pekare. Funktionstabellen måste alltid inledas med pekare till funktionerna som deklarerats i interfacet IUnknown.



Figur 2.5: Interface-pekare och virtuell funktionstabell

När en klient skapar en instans av ett objekt får klienten en pekare till funktionstabellen. Genom att anropa IUnknown::QueryInterface får klienten information om vilka övriga funktioner som implementerats och kan därefter utnyttja övriga funktionspekare i tabellen för att använda dessa funktioner.

2.3.4.3 Class factories

Instanser av COM-objekt kan skapas på två olika sätt, genom anrop till funktionen CoCreateInstance eller genom att använda en class factory - klassfabrik. (CoCreateInstance

använder dock också en klassfabrik men detta sker i det fördolda, se [26]). Klassfabriker är COM-objekt som skapats med syfte att instansiera andra COM-klasser. Benämningen klassfabrik är dock något missvisande eftersom det inte är klasser som skapas utan instanser av dessa. För att en klassfabrik ska kunna användas krävs att komponenten implementerar interfacet IClassFactory. IClassFactory har två metoder, *CreateInstance* som skapar instanser av objektet och *LockServer* som kan användas för att låsa servern i minnet. Normalt avslutar en server sin exekvering då den inte längre har några klienter, detta kan inverka negativt på prestandan om en klient behöver skapa objekt med korta intervall. Genom att anropa LockServer kan detta problem undvikas. En annan fördel med att använda klassfabriker är att det ger en större kontroll över hur objekt ska instansieras, till exempel är det möjligt att ge flera klienter en referens till samma objekt istället för att skapa ett nytt objekt för varje klient.

2.3.4.4 Monikers

En moniker är ett COM-objekt med förmågan att skapa en instans av ett COM-objekt och initiera dess innehåll. På så sätt blir det möjligt för en klient att använda ett objekt som har exakt samma innehåll som ett tidigare använt objekt. Mer om detta finns att läsa på [6].

2.3.5 COM och trådar

För att åskådliggöra ett enkelt problem på att trådning kan innebära problem kan tillämpning för lagerhantering användas som exempel. Tillämpningen är försedd med en funktion för att registrera att en enhet flyttats från lager A till lager B enligt följande pseudokod:

```
1. Flytta(int antal)
2. {
3.     int nyttA = A – antal;
4.     int nyttB = B + antal;
5.     A = nyttA;
6.     B = nyttB;
7. }
```

Antag att både A och B innehåller 10 enheter till en början. Antag vidare att två trådar samtidigt anropar funktionen, det är då möjligt att trådarna exekverar på följande sätt:

1. Tråd 1 exekverar rad 1-3 och lagrar värdet nio i den lokala variabeln nyttA.
2. Tråd 2 exekverar samtliga rader, lagrar värdet nio i den lokala variabeln nyttA och elva i nyttB för att sedan överföra dessa värden till A och B.

3. Tråd 1 fortsätter exekveringen med rad 4-7, lagrar värdet tolv i nyttB, därefter lagras värden från nyttA och nyttB i A respektive B, det vill säga A innehåller nu värdet nio och B innehåller värdet tolv.

Resultatet av detta blir alltså att det plötsligt finns totalt tjugoen enheter registrerade vilket givetvis inte är acceptabelt. I COM finns det två olika trådningsmodeller för att hantera sådana trådningsrelaterade problem:

- Fria trådar
- Apartment

Det är upp till skaparen av en COM-komponent att välja en av dessa modeller. Då modellen med fria trådar används är det helt upp till konstruktören att göra modellen trådsäker (t.ex. genom att använda semaforer). Om apartmentmodellen (apartment = våning, lägenhet) däremot används gör COM automatiskt komponenten trådsäker. Då flera trådar samtidigt anropar funktioner i komponenten placeras anropen i en kö och betas av i turordning med hjälp av en meddelandeloop.

2.3.5.1 Callbacks

När det gäller callbacks så kan inte en COM-komponent utan vidare signalera till klienten att en händelse inträffat från vilken tråd som helst. Ett exempel på detta är en funktion som startar en ny tråd (till exempel en timer) och omedelbart returnerar medan den nya tråden fortsätter exekvera för att i ett senare skede signalera en händelse. I en komponent som använder sig av modellen med apartment-trådar är detta inte möjligt. Då fria trådar används är det visserligen möjligt, men kan i vissa fall leda till programkrascher (se [19]) och bör därför undvikas. Det finns två tillvägagångssätt för att lösa detta problem enligt [19]:

- Att skicka ett meddelande till den första tråden och låta händelsen inträffa där.
- Att skicka interface-pekaren som refererar till callbackinterfacet till den andra tråden genom marshalling (se 2.4.3) och därigenom göra det möjligt för denna att signalera direkt.

Microsoft rekommenderar det första av dessa alternativ, se [16].

2.4 DCOM

I detta delkapitel redogörs för en utökning av COM som kallas Distributed COM (DCOM), som kan användas vid utvecklande av distribuerade tillämpningar. Inledningsvis förklaras vad DCOM är och hur det fungerar. Därefter följer en beskrivning av multitier-system och hur DCOM kan användas för att skapa dessa. Avslutningsvis ges en kort inblick i vilka prestandaproblem som kan uppstå vid användandet av DCOM, samt en introduktion till säkerhet med DCOM.

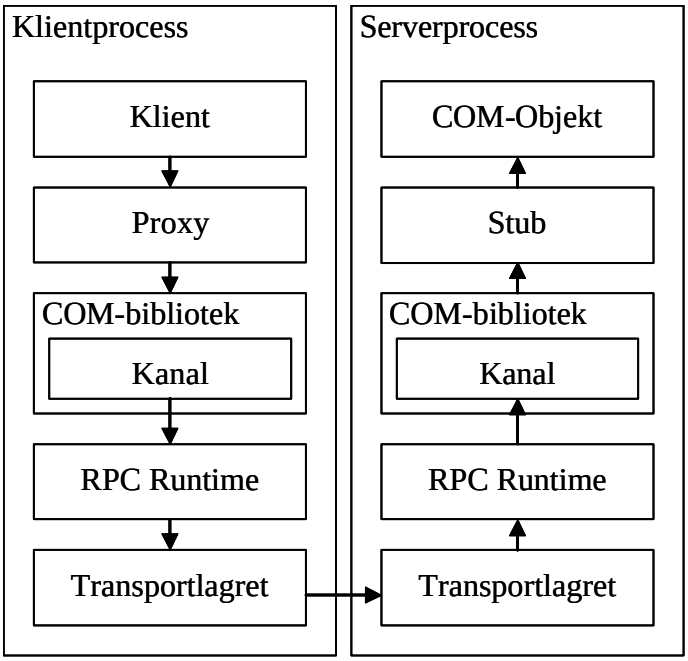
2.4.1 Bakgrund

DCOM lanserades tillsammans med Windows NT 1996. COM designades förvisso för kommunikation mellan olika processer, med detta gällde endast för processer på en och samma dator. I och med införandet av DCOM blev det möjligt att låta processer på olika datorer kommunicera med varandra.

2.4.2 Kommunikation

Kommunikationsprotokollet som används vid kommunikation mellan två processer är en variant av RPC som har stöd för objektorientering och kallas därför ibland Object RPC (ORPC). DCOM gör RPC osynligt för programmeraren, vilket ger fördelen att programmeraren inte behöver vara insatt i hur RPC fungerar för att använda DCOM (se [23]). För att en klient skall kunna kommunicera med en DCOM-komponent måste komponenten registreras i Windows systemregister på klientsidan.

Figur 2.6 visar hur kommunikationen fungerar vid användning av COM-objekt utanför den aktuella processen. Observera att figuren är giltig för en serverprocess på såväl samma dator som klienten som för serverprocess på en annan dator. Ligger serverprocessen på samma dator som klienten sker anropen med Local RPC (LRPC), istället för vanlig RPC.



Figur 2.6: Kommunikation mellan klient och COM-objekt

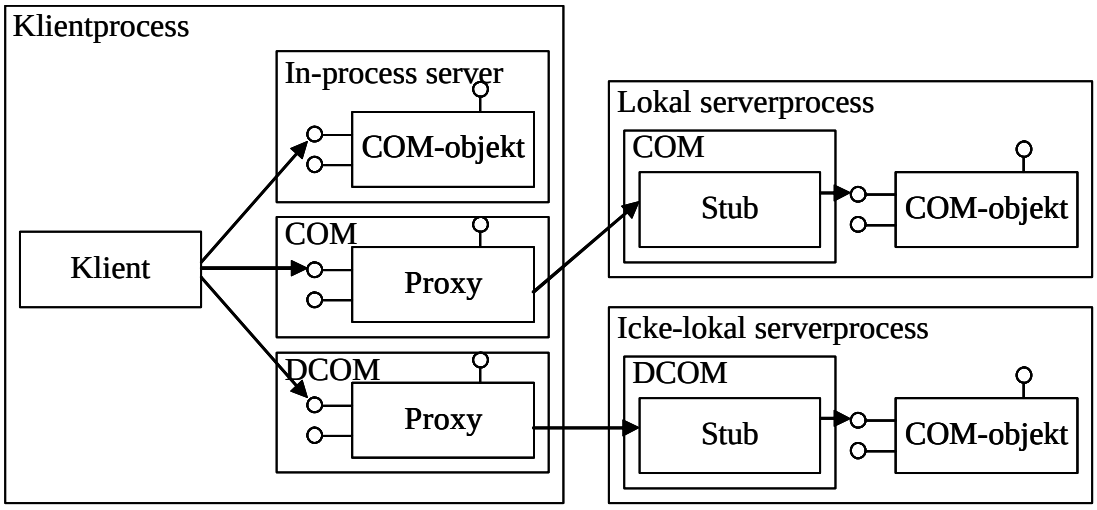
2.4.3 Proxy & Stub

Funktionspekarna i den virtuella funktionstabellen är inte aktuella vid kommunikation mellan processer. För att nå ett objekt i en annan process används istället något som kallas proxy och stub. Mer om detta finns att läsa på [23].

Kommunikationen mellan en klient och ett COM-objekt i en annan process (antingen på samma dator, eller på en annan dator) fungerar genom att en proxy körs i klientprocessen, och en stub körs i objektets process. En kommunikationskanal upprättas mellan klientens proxy och objektets stub med hjälp av RPC. En klient anropar en metod hos objektet via sin proxy som vidarebefordrar anropet till objektets stub via RPC. När objektets stub tar emot anropet så skickas det vidare till objektet som klienten vill kommunicera med. Eventuella returvärden skickas sedan tillbaka till klientens proxy via objektets stub. För att kunna skicka parametrar och metodanrop (mellan processer) måste dessa serialiseras, alltså omvandlas till en form som gör det möjligt att skicka dem. Processen att packa ihop metodanrop och parametrar och skicka dem som serialiserad data kallas marshalling.

Hanteringen av proxy och stub sköts oftast automatiskt av programmeringsmiljön, exempelvis döljer VB hanteringen av dessa för programmeraren.

Figur 2.7 visar hur kommunikationen mellan klient och COM-objekt sker i tre olika fall; när COM-objektet ligger i samma process som klienten, när COM-objektet ligger i en annan process på samma dator som klienten samt när COM-objektet ligger på en annan dator.



Figur 2.7: Kommunikation mellan klient och objekt i tre olika fall

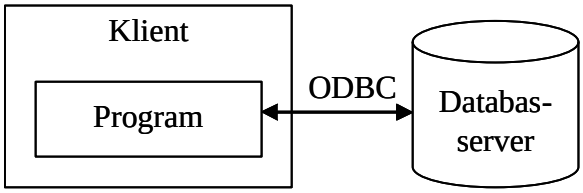
2.4.4 Multitier

Multitier betyder ungefär flerskikt, och innebär i detta sammanhang att ett mjukvarusystem delas in i flera delar. En vanlig uppdelning är att ha flera klienter och en server som klienterna arbetar mot (detta kallas även för 2-tier). För lite större system kan det vara praktiskt att införa ytterligare ett skikt, nämligen ett mellanskikt mellan klienterna och servern (vilket ger ett 3-tier system). I mellanskitet läggs vanligen delar av den logik som annars skulle ha placerats på klientsidan enligt [28]. DCOM utvecklades bland annat för att det skulle vara enkelt att bygga komponenter i mellanskiktet. För utförligare beskrivningar hänvisas läsaren till [23].

2.4.5 Multitier-exempel med DCOM

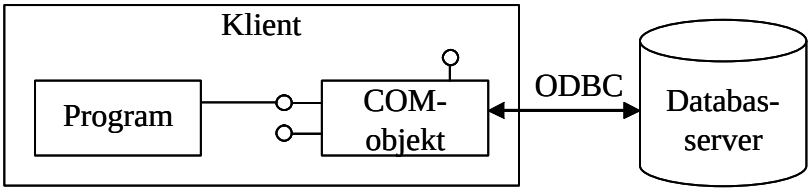
Fördelarna med DCOM och mellanskikt illustreras med ett exempel, där flera klienter arbetar mot en databasserver. Det tänkta scenariot är att en ändring av databasservern utförs, som kräver att databasanropen i klientprogramvaran måste modifieras, för att se hur dessa ändringar avspeglas i hela mjukvarusystemet. Anropen mot databasservern i exemplen nedan antas ske med Open Database Connectivity (ODBC), men det kunde lika gärna ha varit ett annat databasprotokoll eftersom dessa inte har någonting med COM-teknikerna att göra.

Figur 2.8 visar ett 2-tier system utan COM och DCOM. För att klienternas databasanrop skall ändras måste hela klientprogramvaran uppdateras på alla klienter.



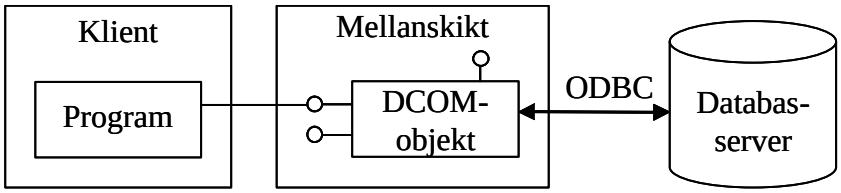
Figur 2.8: 2-tier utan COM/DCOM

Figur 2.9 visar ett annat 2-tier system där klientprogramvaran använder en COM-komponent för att nå databasservern. I detta fall är det endast COM-komponenten som behöver modifieras och läggas ut på klientdatorerna, övriga delar av programvaran påverkas ej.



Figur 2.9: 2-tier med COM

Figur 2.10 illustrerar ett multitier-system (3-tier med klient, mellanskikt och server). I detta fall är en DCOM-komponent, som ansvarar för alla databasanrop, placerad i mellanskiktet. Klienterna använder komponenten som antas innehålla all databasrelaterad logik. Detta leder till att ändringar av databasanropen endast påverkar DCOM-komponenten. I detta läge kan DCOM-komponenten uppdateras utan att klienterna berörs. Detta illustrerar fördelen med mellanskikt och DCOM. Observera att Figur 2.10 för enkelhetens skull inte är helt korrekt i det avseendet att konceptet med proxy och stub har utelämnats.



Figur 2.10: 3-tier med DCOM

2.4.6 Prestanda

Anropen till objekt som ligger på en annan dator tar minst 1000ggr längre tid än vanliga COM-anrop via den virtuella funktionstabellen enligt [23]. Vidare kan det vara bra att känna till att VB gör så kallade synkroniserade metodanrop till objekt på andra datorer. Detta innebär att VB-programmet är blockerat medan svar från metodanropet inväntas.

2.4.7 Säkerhet

Vid utvecklande av COM-komponenter som endast skall används lokalt behöver ingen extra hänsyn till säkerhet tas, exekveringsrättigheter kan styras på sedvanligt sätt genom konfigurering av rättigheter knutna till olika användarkonton. Skall däremot COM-komponenter användas av klienter på andra datorer tillkommer en rad säkerhetsaspekter, att utan vidare tillåta all sådan exekvering kan innebära en stor säkerhetsrisk enligt [24].

Behörigheten att använda COM-komponenter på en annan dator (här kallad server) kan delas in i två delar, startbehörighet och åtkomstbehörighet. Startbehörigheten beskriver vem som har rättighet att starta COM-processen på servern, medan åtkomstbehörigheten beskriver vem som har rättighet att använda COM-processen på servern. Dessa behörigheter kan ställas in för varje enskild COM-komponent, om ingen sådan inställning görs gäller den globala inställning som ställts in för COM-komponenter generellt.

En klient som skapar ett COM-objekt som ligger på en server måste ha ett användarkonto på servern. Antingen måste klienten och servern tillhöra samma domän, eller så måste samma användarnamn och lösenord vara giltiga för en lokal användare på servern.

På servern kan COM-komponenten startas upp med tre typer av användaridentiteter:

- Den interaktiva användaren. COM-komponenten kommer att startas med den inloggade användaren som ägare till COM-processen.
- Användaren som startar tillämpningen (på klientdatorn). Om användaren TestUser är inloggad på klientdatorn, kommer COM-komponenten startas med användaridentiteten TestUser på serverdatorn.

- Specificerad användare. Detta alternativ ger möjligheten att manuellt specificera vilket användarnamn COM-komponenten kommer att exekvera under.

Det går även att konfigurera olika så kallade autentiseringsnivåer. Exempelvis kan denna sättas till *None* och användargruppen *Everyone* (eller *Alla* som den heter i svenska versioner av Windows) ges fullständiga rättigheter, detta för att ge alla åtkomst till servern.

Alla COM-relaterade säkerhetsinställningar lagras i Windows systemregister. De kan ändras direkt därifrån eller med hjälp av verktyget dcomcnfg (vilket kräver administratörs-rättigheter). Säkerhetsinställningar kan även konfigureras direkt i programkod med hjälp av kommandot *CoInitializeSecurity*, dessa inställningar gäller före de som finns lagrade i systemregistret. Det är dock inte möjligt att kodmässigt ange en lägre säkerhetsnivå än vad som finns angivet i systemregistret utan endast att höja säkerhetsnivån.

2.5 ActiveX

Detta delkapitel ger en kort beskrivning av ActiveX, och förklarar vad ActiveX har för koppling till COM. Delkapitlet inleds med en bakgrundsbeskrivning och avslutas med en översiktlig beskrivning av ActiveX.

2.5.1 Bakgrund

Förr kunde 16-bits VB-program använda kontroller som kallades Visual Basic Extensions (VBX) för att få utökad funktionalitet. Observera att det endast var i C och C++ som själva VBX-kontrollerna kunde utvecklas. I samband med att Windows fick 32-bits stöd (Windows 95/NT) lanserades en ny 32-bits variant av VBX som nu kunde användas i andra språk än Visual Basic. De nya kontrollerna kallades OLE Custom control (OCX), men fick byta namn till ActiveX-kontroller i samband med introduktionen av ActiveX. Om detta kan läsas på [2].

När termen ActiveX introducerades 1996 användes den för att referera till COM-teknologi som på något sätt hade samband med internet. Därefter har begreppet utvidgats till att mer och mer bli ett samlingsnamn på COM-baserad teknologi i allmänhet. Faktum är att ActiveX lika gärna kan ses som ett slags marknadsföringsbegrepp som en avgränsad teknologi enligt [5]

2.5.2 Översikt

Den klassiska ActiveX-kontrollen är en COM-komponent med grafiskt användargränssnitt (i vanliga fall) som kan utföra någon form av datamanipulering, kommunicera med sin omgivning och kan integreras i skiftande miljöer såsom webbsidor, program skrivna i

VB/VC++, MS-Access-formulär och så vidare. En sådan kontroll kan vara allt ifrån en färgglad knapp vars enda uppgift är att signalera att den blivit intryckt, till ett komplett program, färdigt att exekvera så snart det integrerats i till exempel en webbsida. I likhet med övriga COM-komponenter så är inte ActiveX-kontroller bundna till att utvecklas i något särskilt språk utan kan till exempel skrivas i VB och användas i VC++.

ActiveX är dock mer än ActiveX-kontroller. Med hjälp av ActiveX-scripting kan kontroller ges instruktioner med hjälp av ett enkelt scriptspråk. Detta utnyttjas främst på webbsidor för att ge en ActiveX-kontroll instruktioner med hjälp av VB-script eller JavaScript. Till exempel skulle en kontroll kunna instrueras att anta samma bakgrundsfärg som resten av webbsidan istället för att behöva skrivas om och kompileras på nytt (under förutsättning att kontrollens skapare givit den en sådan egenskap).

ActiveX-dokument är en teknologi för att öppna dokument tillhörande ett program i ett annat program. Exempelvis kan en länk till ett Word-dokument i Internet Explorer klickas på, för att öppna dokumentet inuti Internet Explorer.

ActiveX-automatisering innebär att ett program kan styras av ett annat program. Detta sker genom att det styrande programmet exekverar ett script som utför olika uppgifter i programmet som skall styras.

2.6 COM+

I detta delkapitel ges en översiktlig beskrivning av en nyare variant av COM vid namn COM+. Först ges en bakgrundsbeskrivning som förklarar varför det fanns ett behov av COM+. Därefter ges en kort beskrivning av de utökningar som COM+ innehåller.

2.6.1 Bakgrund

Grunden till COM+ lades då Microsoft utvecklade Microsoft Transaction Server (MTS) för Windows NT Server. MTS var en utvecklad variant av COM som hade en del extra funktioner. Bland annat fanns stöd för transaktioner, utökade säkerhetsfunktioner samt enklare administration, se [7].

Problemet var att det nu fanns två komponentteknologier, MTS och COM. Vid utvecklande av komponenter måste en av dessa teknologier väljas, då det inte går att använda båda samtidigt. MTS var ingen del av COM, eftersom de bygger på olika programmeringsmodeller och olika ”runtime” lager. Detta ledde även till att utveckling av MTS-komponenter erfordrade kunskaper inom båda programmeringsmodellerna, just eftersom MTS i grunden var baserat på COM.

Inför lanseringen av Windows 2000 utvecklades COM+ som skulle lösa problemen med MTS och COM. COM+ för samman de bästa egenskaperna hos COM och MTS (se [23]), samt en del nya koncept i ett enda format. Dessutom är det enklare att utveckla komponenter med COM+ än med MTS enligt [23].

2.6.2 Utökningar

COM+ har samma förbättringar som MTS, samt följande:

- Lastbalansering. När en klient skall kommunicera med ett objekt på en server sker en lastbalansering, där klientens förfrågan omdirigeras till den server med lägst belastning. Givetvis måste objektet som klienten kommunicerar med finnas på alla servrar.
- COM+ Catalog. Såväl COM som MTS använder systemregistret för att lagra konfigurationsinställningar. COM+ lagrar däremot inställningarna i en databas som kallas COM+ Catalog.
- In-Memory Database (IMDB). För de applikationer som kräver snabb databasåtkomst finns IMDB, som är en databas där tabellerna ligger lagrade i internminnet.

Vidare finns stöd för exempelvis Object Pooling (en teknik för att hålla objektinstanser i minnet för att möjliggöra snabbare åtkomst till dessa), bättre händelsehantering samt utökade administrationsmöjligheter av komponenter. För mer information om nyheterna i COM+, se [7].

2.7 För- och nackdelar med COM

I detta avsnitt ges en kort översikt av de viktigaste för- och nackdelarna med COM.

2.7.1 Fördelar med COM

COM har bland annat följande fördelar enligt [25]:

- COM-komponenter är lätt utbytbara:
  I de flesta fall så uppstår förr eller senare ett behov av att uppgradera mjukvara. Med COM är det enkelt att uppgradera delar av en applikation – den aktuella COM-komponenten kan bytas ut utan att applikationens övriga delar påverkas.

- COM-komponenter är ideala vid ändrade krav:

COM-konceptet medför en naturlig uppdelning av applikationen och delar är lätt utbytbara (se ovan). Detta medför att det med lite planering och kännedom om vilka delar i programmet som kan komma att ändras är lätt att isolera dessa i egna komponenter. Dessa komponenter kan sedan ändras utan risk för att ändringarna påverkar andra delar av applikationen. [25] tar som exempel upp ett program för skattedeklaration. Beskattningsregler kommer aldrig att vara hundraprocentigt lika från ett år till ett annat, med kännedom om detta kan berörd kod samlas i en eller flera egna komponenter som kan bytas ut vid behov. På så sätt undviks upprepade ändringar i övriga programdelar som lättare skulle kunna leda till fel i programvaran och försämra läsbarheten hos källkoden.

- Förbättrad återanvändbarhet:
  Ökad uppdelning medför större chans till återanvändning. Med COM-komponenter uppstår ökad möjlighet till återanvändning och integrering av applikationer. Uppdateringar av en komponent påverkar inte de program som redan använder komponenten.

- Parallell utveckling underlättas:
  Vanligtvis när en COM-komponent skall konstrueras designas dess interface först. Så fort interfacen är klara kan komponenten distribueras till de programmerare som behöver använda komponenten. Utvecklingen av komponenten och de programdelar som använder den kan därefter ske parallellt.

Dessa fördelar är inte unika för COM, de skulle kunna beskriva objektorienterad programmering i allmänhet. Detta är naturligt eftersom idéerna med COM bygger på objektorientering men försöker gå ett steg längre.

2.7.2 Nackdelar med COM

COM har bland annat följande nackdelar enligt [25]:

- Komponentversioner kan orsaka problem:
  Varje komponent identifieras med ett unikt id (GUID) som lagras i windows systemregister. Varje gång komponenten uppdateras identifieras den med ett nytt sådant id, detta kan lätt leda till problem under utvecklingen. Exempelvis kan ett

svårupptäckt fel bero på att ett program inte använde den komponent som utvecklaren trodde, utan en äldre version.

- Bevarandet av gamla interface:
  När en COM-komponent väl skapats så kommer dess interface aldrig att ändras (med anledning av att komponenten skall vara bakåtkompatibel). Efter att antal uppdateringar kan komponenten ha fått ett stort antal interface där många är nästan identiska. Detta leder till att det blir svårare att förstå hur komponenten skall användas och osäkerhet om vilket interface som bör användas. Dessutom blir komponenten svårare att underhålla. Till slut kan utvecklingen nå den punkt då det blir nödvändigt att helt enkelt skapa en ny komponent och flytta all funktionalitet till den. I ett sådant läge faller naturligtvis också tanken med automatisk bakåtkompabilitet.

- COM-komponenter kräver noggrann planering:
  En COM-komponent blir aldrig bättre än dess interface. Dåligt planerade interface kommer att medföra fler uppdateringar, fler uppdateringar kommer att öka riskerna för problem med komponentversioner och gamla interface (se tidigare redovisade nackdelar).

3 Visual Basic och Visual C++

Vid implementationen av konstruktionslösningen kommer de båda utvecklingsverktygen Visual Basic och Visual C++ att användas. Syftet med detta kapitel är att översiktligt beskriva viktiga delar av dessa utvecklingsmiljöer och språk för att därigenom underlätta förståelsen av den implementationsbeskrivning som följer i kapitel 4 och 5. Dessutom kommer hanteringen av COM i de båda språken att beskrivas, vilket också har stor betydelse för implementationen. Kapitlet avslutas med en beskrivning av hur COM-komponenter skapade i något av de båda programspråken kan distribueras.

Som klargjordes i kapitlet *Avgränsningar* är avsikten inte att ge någon komplett beskrivning av vare sig Visual Basic eller Visual C++.

3.1 Introduktion till Visual Basic

Detta avsnitt inleds med en kort beskrivning av vad Visual Basic är och hur det utvecklades. Därefter förklaras hur händelsehanteringen i språket fungerar. Anledningen till att händelsehanteringen beskrivs är att den har en central roll i konstruktionslösningen och implementationen, inte minst eftersom händelsehanteringen även används i samband med nyttjandet av COM.

3.1.1 Vad är Visual Basic?

Microsoft lanserade VB 1991. VB är baserat på språket BASIC, men är anpassat för att utveckla fönsterbaserade program för Microsoft Windows. I andra programspråk (till exempel C) krävdes hundratals rader kod för att producera mycket enkla fönsterbaserade program i Windows-miljö (se [12]), men med hjälp av VB kunde grafiska användargränssnitt skapas utan någon nämnvärd mängd kod. Själva grafiska gränssnittet ritas upp med hjälp av utvecklingsverktyget och utvecklaren behöver endast skriva den programspecifika programkoden i stället för att skriva hundratals rader kod för att få ett fungerade fönsterbaserat program. Benämningen VB kan referera till både språket och utvecklingsmiljön.

VB blev en succé, enligt [12] mycket tack vare dess enkelhet. Dessutom utvecklade många tredjepartstillverkade så kallade VBX-filer, som gav VB utökad funktionalitet.

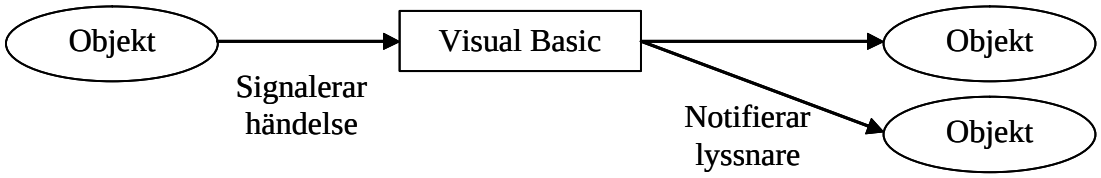
Den versionen av VB som är aktuell för implementationen är version 6, eftersom denna version finns tillgänglig på datorerna vid Karlstads universitet. Den senaste versionen av VB är VB.NET, som är anpassad för Microsofts .NET plattform. Tabell 3.1 ger en översikt av de olika versioner av VB som lanserats genom åren.

Version	Några nyheter	Lanseringsår
VB 1.0	(Första versionen)	1991
VB 2.0	Snabbare. Databasåtkomst via ODBC.	1992
VB 3.0	Stöd för OLE.	1993
VB 4.0	32-bits stöd och OCX i stället för VBX	1995
VB 5.0	ActiveX. Stöd för riktig kompilering av exekverbara filer.	1997
VB 6.0	Utökade Internet-funktioner.	1998
VB.NET	Anpassad för .NET-plattformen.	2002

Tabell 3.1: Visual Basic-versioner

3.1.2 Visual Basic och händelsehantering

Windows och likaså VB är händelsebaserade, vilket betyder att något utförs som ett resultat av att en händelse har inträffat. En händelse kan exempelvis inträffa när användaren av ett program klickar på en knapp eller stänger ett fönster. En händelse kan även framkallas med programkod. Det kan ske antingen explicit genom att exekvera ett kommando som triggar en händelse eller implicit genom att exempelvis ändra någon egenskap hos ett objekt som i sin tur triggar en händelse. För att fånga upp en händelse måste ett objekt lyssna på en specificerad händelsetyp. När en händelse inträffar skickar objektet som genererade denna en signal till VB som i sin tur notifierar de objekt som valt att lyssna på händelsen. Detta illustreras i figur 3.1.



Figur 3.1: Händelsehantering i Visual Basic

3.1.2.1 Exempel med händelser

Programkoden nedan deklarerar händelsen med namnet *TestEvent*:

```
Public Event TestEvent(ByVal TestVar As String)
```

När en klass har deklarerat en händelse kan händelsen triggas från ett annat ställe i programkoden (i detta exempel skickas strängen ”Hello” med som argument), se nedan:

```
RaiseEvent TestEvent(“Hello”)
```

För att lyssna på ett objekts händelser används nyckelordet *WithEvents* när objektet deklareras. Om de två ovanstående kodraderna tillhörde klassen *EventDemo* kan ett objekt deklareras så att det blir möjligt att lyssna på dess händelser med följande programkod:

```
Public WithEvents EventDemoInstance As EventDemo
```

Detta innebär att när händelsen *TestEvent* i *EventDemo* triggas, kommer en funktion med namnet *EventDemoInstance_TestEvent* att anropas, alltså objektets namn, ett understreck samt namnet på händelsen. Funktionen kan se ut på följande sätt:

```
Private Sub EventDemoInstance_TestEvent(ByVal TestVar As String)
...
End Sub
```

Eftersom programkoden skickade strängen ”Hello” när händelsen skapades, kommer *TestVar* i detta exempel att innehålla strängen ”Hello”.

3.2 Visual Basic och COM

I detta avsnitt redogörs för hur COM kan användas i VB. Läsaren kommer att lägga märke till att det är enkelt att skapa och använda en COM-komponent i VB, eftersom VB gör mycket av arbetet automatiskt utan att programmeraren lägger märke till det.

När referenser till menyer och inställningar i detta delkapitel görs, avses utvecklingsmiljön VB.

3.2.1 Skapa COM-komponenter i Visual Basic

För att skapa en COM-komponent i VB skapas ett nytt projekt av typen *ActiveX EXE* eller *ActiveX DLL* (för en out-of-process respektive en in-process server).

3.2.1.1 Komponentens interface

I VB är det inte aktuellt att använda sig av IDL för att definiera ett interface, faktum är att det enda programmeraren behöver göra för att möjliggöra för en klient att komma åt objektets egenskaper och metoder är att deklarera dessa som publika. På samma sätt skapas enkelt ett utgående interface genom att deklarera publika händelser.

Principen med in- och ut-parametrar till funktioner märks inte i VB utan funktioner skrivs på formen med traditionella returvärden. För att få en uppfattning om hur komponentens interface framstår för klienter skrivna i ett annat språk än VB kan verktyget *OLE View* som ingår i Visual Studio användas för att söka upp komponenten under *TypeLibraries.* Antag att en komponent innehåller följande kod:

```
Event EventTest()

Public Function FuncTest(x As Integer) As Integer
...
End Sub
```

När OLE View används för att undersöka komponentens interface kommer följande kod att visas:

```
...
HRESULT FuncTest([in, out] short* x, [out, retval] short* );
...
void EventTest();
...
```

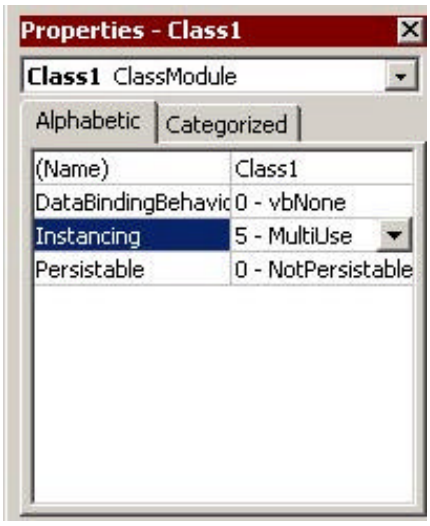
Funktionen returnerar alltså på sedvanligt sätt ett värde av typen HRESULT och använder en ut-parameter istället för returvärdet som deklarerades i VB. Värt att notera är att parametern x deklarerats som [in, out] eftersom referensparametrar är standard i VB och att den VB-specifika datatypen *Integer* ändrats till *short*.

3.2.1.2 Identifiering

När det gäller GUIDs så skapas och registreras dessa automatiskt i systemregistret då projektet kompileras. Eftersom det inte är önskvärt att skapa nya GUIDs vid varje omkompilering krävs det ett litet manuellt ingrepp genom att välja *Project – <Projektnamn> Properties* i menyn och därefter välja *Binary Compability* under fliken *Component*. Projektet måste dock kompileras en gång innan denna inställning görs. Det namn som kommer att registreras för komponenten blir detsamma som projektets namn.

3.2.1.3 Alternativ för instansiering av komponentklasser

För att styra hur en komponentklass skall instansieras, till exempel om alla klienter skall använda samma objekt istället för att tilldelas varsin instans av klassen, används i COM klassfabriker. Inte heller detta märker VB-programmeraren något av utan detta styrs genom att ange olika alternativ för egenskapen *Instancing* (figur 3.2).



Figur 3.2: Egenskapsdialog

De olika alternativen för instansiering av en komponentklass är följande:

1. Private
2. PublicNotCreatable
3. SingleUse
4. GlobalSingleUse
5. MultiUse
6. GlobalMultiUse

Alternativ 1 (*Private*) innebär att klienter inte har direkt tillgång till klassen, detta alternativ måste väljas för att en klass inte automatiskt skall bli en komponentklass. Alternativ 2 (*PublicNotCreatable*) används då det är önskvärt att klienter skall kunna använda en klass men inte kunna skapa instanser av den, instansieringen måste istället skötas av en annan komponentklass. Detta kan till exempel användas för att ge flera klienter tillgång till ett och samma objekt (denna princip kommer senare att tillämpas i implementationslösningen). Skillnaden mellan *MultiUse* och *SingleUse* är att då SingleUse används så kommer en ny instans av komponenten att laddas i datorns minne för varje gång ett objekt av klassen skapas. *Global* har den betydelsen att en klient kan använda klassens metoder utan att först instansiera ett objekt.

Standardinställningen för klasser i ett VB-projekt av typen ActiveX EXE eller ActiveX DLL är *MultiUse* vilket innebär att endast en serverprocess kommer att exekvera och att varje klient som använder en komponentklass kommer att tilldelas en egen instans av klassen.

Alternativ 3 och 4 är inte tillgängliga i ett projekt av typen ActiveX DLL eftersom en in-process server inte exekverar som en självständig process.

3.2.1.4 Remote server

Att skapa en remote server är i stort sett identiskt med att skapa en komponent för lokal exekvering. Genom att välja *Project - <projektnamn> Properties* i menyn och därefter bocka i rutan *Remote Server Files* under fliken *Component* blir det möjligt för servern att exekvera som ett DCOM-objekt. När detta alternativ är valt kommer en Type Library-fil (TLB-fil) att skapas vid kompilering. Denna fil kan klienten använda för att exempelvis få reda på vilka metoder som COM-objektet på servern har.

3.2.2 Använda COM-komponenter i Visual Basic

För att kunna använda en COM-komponent i en VB-klient måste en referens till komponenten läggas till i det aktuella klientprojektet. Genom att välja *Project – References* i menyn öppnas en dialogruta där alla COM-komponenter som finns registrerade i systemregistret listas, genom att bocka i rutan vid den aktuella komponenten är det därefter möjligt att använda den i koden. (För information om hur en komponent kan läggas till i systemregistret hänvisas till kapitel 3.5). För att deklarera en variabel av typen *MinKomponentklass* hörande till COM-komponenten *MinServer* används koden:

```
Dim MittObjekt As MinServer.MinKomponentklass
```

Om komponenten har ett utgående interface som ska användas i klienten:

```
Dim WithEvents MittObjekt As MinServer.MinKomponentklass
```

För att genomföra själva instansieringen av objektet (och starta servern om den inte redan exekverar) blir koden:

```
Set MittObjekt = CreateObject("MinServer.MinKomponentklass")
```

Som synes är anropet till CreateObject det enda som avslöjar att MittObjekt är en COM-komponent, i övrigt är koden densamma som den skulle varit för ett lokalt objekt.

När objektet väl har initierats med ovanstående kod kan det tillsammans med sina egenskaper, funktioner och händelser användas på samma sätt som ett lokalt objekt. Då objektet inte längre skall användas bör kopplingen till servern avslutas genom koden:

```
Set MittObjekt = Nothing
```

En klient som använder sig av en COM-komponent kompileras och exekveras på sedvanligt sätt.

För att använda en COM-komponent som ligger på en annan dator måste objektets TLB-fil finnas tillgänglig. TLB-filen läggs till som en referens i klientprojektet genom att välja *Project – References*, *Browse* och därefter väljs den aktuella TLB-filen. Nu kan COM-komponenten användas på samma som om den hade varit placerad på samma dator, med ett undantag. Själva initieringen av komponenten ser ut såhär:

```
Set MittObjekt = CreateObject("MinServer.MinKomponentklass", "srv")
```

Där ”srv” är namnet på den dator där servern finns, den enda skillnaden mot en lokal server är att namnet på serverdatorn måste specificeras. Namnet kan vara i form av ett datornamn eller en IP-adress.

3.2.3 Ett enkelt exempel

I bilaga B2 beskrivs steg för steg ett exempel på hur en server och en klient kan skrivas i VB med hjälp av grundläggande COM-funktionalitet. Servern har en funktion *add* som lägger samman två heltal och en egenskap *Color* som är läs- och skrivbar. Dessutom finns en händelse *ServerCallback*. Servern i exemplet exekveras som en lokal out-of-process-server.

3.3 Introduktion till Visual C++

Detta avsnitt inleds med en kort beskrivning av vad Visual C++ och MFC är och utvecklingen av dessa. Därefter förklaras hur händelsehanteringen med MFC i Visual C++ fungerar. Anledningen att detta beskrivs är, precis som med beskrivningen av Visual Basic och dess händelsehantering, att dessa bitar har en central roll i implementationen som beskrivs i senare kapitel.

3.3.1 Vad är Visual C++?

Enligt [13] lanserade Microsoft VC++ 1993. VC++ är en av de vanligaste utvecklingsmiljöerna för att utveckla objektorienterade C++ program enligt [10]. Det går att utveckla ett flertal typer av program baserade på olika tekniker genom att välja mellan olika projekttyper i den inbyggda projektguiden.

En central del av VC++ är klassbiblioteket MFC. MFC togs fram för att underlätta utvecklandet av objektorienterade Windowsprogram i C++. MFC var en vidareutveckling av Application Framework X (AFX), vars syfte också var att underlätta utvecklandet av Windowsprogram. MFC kan ses som en abstraktion av Windows API och döljer en del komplexa detaljer för programmeraren. MFC består dessutom av en stor uppsättning klasser, exempelvis för hantering av olika datastrukturer samt grafik och fönsterhantering, se [8].

3.3.2 Visual C++ och händelsehantering

I VC++ används MFC-bibliotekets så kallade ”Message Maps” för att kontrollera händelsehanteringen. De händelser som Windows tar emot, exempelvis en musklickning, mappas (det vill säga översätts) till en funktion som exekveras då händelsen tas emot. Antag att en dialogruta, med klassnamnet *CDemo*, har en verkställ-knapp som identifieras med identifieraren *IDC_BUTTON_APPLY*. För att utföra något när användaren klickar på knappen behövs följande programkod:

```
BEGIN_MESSAGE_MAP(CDemo, CDialog)
          //{{AFX_MSG_MAP(CDemo)
          ON_BN_CLICKED(IDC_BUTTON_APPLY, OnButtonApply)
          //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

ON_BN_CLICKED är namnet på händelsen, som inträffar då användaren har klickat på en knapp. Som nämndes tidigare är det konstanten *IDC_BUTTON_APPLY* som identifierar verkställ-knappen. Den sista parametern *OnButtonApply* är namnet på den metod (i klassen *CDemo*) som kommer anropas då händelsen inträffar. *OnButtonApply* är en helt vanlig metod med följande utseende:

```
void CDemo::OnButtonApply ()
{
          // Kod som utförs då användaren klickat på 'Verkställ'
}
```

3.4 Visual C++ och COM

Detta delkapitel beskriver hur COM-komponenter kan skapas och användas i VC++. Tonvikten ligger på användandet av komponenter medan skapandet kommer att beröras mycket kort. Att använda COM i VC++ kräver en hel del kod som kan framstå som komplicerad för den oinvigde, med anledning av detta kan delkapitlet upplevas som en aning tungläst då det innehåller en del kodexempel. Syftet är dock att visa hur koden principiellt kan se ut, inte att förklara alla detaljer. Läsaren behöver därför inte oroa sig över att en del detaljer i koden kan framstå som en aning kryptiska.

När referenser till menyer och inställningar i detta delkapitel görs, avses utvecklingsmiljön VC++.

3.4.1 Skapa COM-komponenter i Visual C++

För att skapa en COM-komponent i VC++ kan det vara lämpligt att använda sig av *ATL COM AppWizard* och välja antingen *Dynamic Link Library* eller *Executable* för en in-process respektive en out-of-process server. Det tredje alternativet, *Service*, innebär som namnet

antyder att servern kan exekvera som en service, det vill säga att ingen användare behöver vara inloggad på den aktuella datorn.

Skapandet av en komponent beskrivs ytterst kort i detta avsnitt, läsaren uppmanas att läsa igenom exemplet i bilaga B.3 då där ges en steg-för-steg beskrivning avsedd att på ett enklare sätt visa principen. En genomläsning av [3] och [11] kan rekommenderas för den läsare som därefter fortfarande känner att något är oklart. [1] ger ett exempel på hur en COM-komponent kan skapas utan att använda MFC eller Active Template Library (ATL).

3.4.1.1 Skapa komponentklass

En komponentklass kan enkelt skapas genom att i menyn välja *Insert – New ATL Object*. I *ATL Object Wizard* som då startas kan olika egenskaper såsom trådningsmodell och stöd för förbindelsepunkter anges.

3.4.1.2 Specificera in- och utgående interface

Att lägga till funktioner till komponentens interface är en relativt enkel process som i huvudsak går ut på att då fliken *FileView* visas i utvecklingsmiljön högerklicka på aktuellt objekt och välja något av nedanstående alternativ:

- *Add Method* lägger till en metod (antingen i det ingående eller i det utgående interfacet beroende på vilket objekt utvecklaren högerklickat på).
- *Add Property* lägger till en egenskap i interfacet.
- *Implement Connection Points* för att upprätta ett utgående interface. Observera att det kan vara nödvändigt att först kompilera projektets IDL-fil (görs enklast genom att högerklicka på filen direkt i utvecklingsmiljön och välja *Compile*) för att detta alternativ skall vara tillgängligt.

3.4.2 Använda COM-komponenter i Visual C++

I detta avsnitt beskrivs hur COM-komponenter kan användas i VC++. Inledningsvis beskrivs vilka inställningar som krävs för att ett projekt skall kunna använda COM. Därefter förklaras hur COM-objekt och tillhörande händelser initieras och används, följt av en beskrivning av hur dessa avallokeras.

Det bör nämnas att det finns ett antal olika alternativ till metodiken som beskrivs i avsnittet. Syftet är alltså inte att ge någon fullständig bild av hur COM-komponenter kan användas utan bara visa på en möjlighet.

3.4.2.1 Konfigurera projekt för COM

När projektet som skall använda en eller flera COM-komponenter skapas bör automatiseringsstöd inkluderas. Automation behövs för att de händelser som COM-komponenten sänder ut skall kunna hanteras, detta stöd är således inte nödvändigt om enbart synkron kommunikation är aktuell. Själva projektet skapas förslagsvis med hjälp av *MFC AppWizard* som är ett av flera valbara alternativ då ett nytt projekt skall skapas. När projektet är skapat är nästa steg att inkludera COM-komponenten i projektet. Detta görs med:

```
#import "MinServer.tlb" no_namespace named_guids raw_interfaces_only
```

I exemplet ovan specificerades COM-komponentens typbibliotek-fil (tlb-fil), vilket är aktuellt då COM-komponenten ligger på en annan dator. Alternativt kan exempelvis ”MinServer.dll” (i fallet då komponenten är av typen in-process) eller ”MinServer.exe” (då komponenten är av typen out-of-process) anges, under förutsättning att någon av dessa finns tillgängliga.

De övriga parametrarna ovan måste finnas där, men deras innebörd är mindre viktig. Attributet *no_namespace* anger att inget så kallat namespace kommer användas. *raw_interfaces_only* gör att prefixet *raw_* inte kommer att läggas till före metodnamn. *named_guids* ger möjligheten att använda namn som exempelvis *CLSID_MinKomponentklass* kan användas för att representera GUID-nummer. För ytterligare information om de olika import-attributen hänvisas läsaren till [20].

För att använda COM och tillhörande händelser måste h-filerna *afxctl.h* och *atlbase.h* inkluderas, detta sker med vanliga *#include*-satser. Slutligen måste en mindre modifikation göras i cpp-filen som har hand om automation. Om klassen som implementeras i denna fil heter *CMyEvents* återfinns följande rad, där *IID___MinKomponentklass* måste ändras till *DIID___MinKomponentklass* (observera att detta är enda förekomsten av *IID___MinKomponentklass* som skall ändras, anledningen till detta är att det är det utgående interfacet hos serverns komponentklass som avses.)

```
INTERFACE_PART(CMyEvents, IID___MinKomponentklass, Dispatch)
```

3.4.2.2 Instansiering av COM-objekt

När ovanstående ändringar har utförts kan själva COM-biblioteket initieras med följande kodrader:

```
HRESULT hres = CoInitialize(NULL);
if (FAILED(hres))
          //(hantera COM-initieringsfel)
```

Nästa steg är att initiera COM-objektet. Först skapas en pekar-variabel till objektet, i detta fall är den av typen *_MinKomponentklass*. Därefter sker själva instansieringen, varefter pekarvariabeln *pMinKlass* kan användas för att komma åt COM-objektet. I kodexemplet nedan används *CLSCTX_LOCAL_SERVER* som tredje parameter, vilket anger att COM-objektet finns på den lokala datorn och att det körs i en separat process. Den enda modifikationen som behövs för att instansiera ett COM-objekt som ligger på en annan dator är att ändra denna parameter till *CLSCTX_REMOTE_SERVER*. I de flesta fall kan det vara lämpligt att ange båda alternativen, för att det skall fungera oavsett var COM-objektet ligger. Detta görs med *CLSCTX_LOCAL_SERVER* | *CLSCTX_REMOTE_SERVER*, det vill säga operationen OR med de båda konstanterna. För en komplett beskrivning av alla tillgängliga alternativ, se [14]. Det viktiga här är visa hur CoCreateInstance används för att skapa en referens till en instans av en komponentklass, inte att förstå alla dess parametrar i detalj.

```
_MinKomponentklass *pMinKlass = NULL;
hr=CoCreateInstance(CLSID_MinKomponentklass, NULL,
          CLSCTX_LOCAL_SERVER, IID__MinKomponentklass,
          reinterpret_cast<void**> (&pMinKlass));
if (FAILED(hres))
          //(hantera fel vid skapande av COM-objekt)
```

3.4.2.3 Händelsehantering med COM-objekt

För att kunna hantera händelser som COM-objektet sänder ut, måste en hel del initieringar göras. Detta fungerar på samma sätt med COM som med DCOM.

Förslagsvis skapas en ny klass där alla metoder som händelser mappas till placeras. I exemplet nedan antas den klassen heta *CEventHandler*. Till en början måste en del variabler deklareras i den klassen som initierar COM-objektet. Dessa variabler placeras förslagsvis under private-delen i klassdeklarationen enligt nedan:

```
private:
          CEventHandler *pEventHandler;
          IUnknown* pUnkSink;
          IUnknown* pUnkSrc;
          DWORD pdwCookie;
```

Variabeln *pdwCookie* används av servern för att identifiera den skapade förbindelsen, då händelsehanteringen senare skall kopplas ifrån måste denna skickas med som ett argument för att servern på ett korrekt sätt skall kunna göra frånkopplingen. Variabeln *pUnkSink* är en interface-pekare till en instans av klassen *CEventHandler* medan *pUnkSrc* är en interface-pekare till den instans av COM-objektet objektet vars händelser är av intresse. (Benämningarna härrör sig från engelskans *source* och *sink* – den som orsakar händelsen och den som tar hand om den.) Dessa variabler initieras nedan, varefter kommandot *AfxConnectionAdvise* skapar själva kopplingen mellan COM-objektets händelser och *CEventHandler*-objektet som skall hantera dessa händelser.

```
pEventHandler = new CEventHandler;
pUnkSink = pEventHandler->GetIDispatch(FALSE);
pdwCookie = 0;
pUnkSrc = NULL;

CComPtr<IUnknown> spUnk = (pMinKlass);
pUnkSrc = spUnk.p;
BOOL bResult = AfxConnectionAdvise(pUnkSrc,
          DIID___MinKomponentklass, pUnkSink, TRUE, &pdwCookie);
if(!bResult)
          //(hantera fel vid initiering av händelsehantering)
```

Nu är själva kopplingen mellan COM-objektet och *CEventHandler*-objektet utförd. Nästa steg är att utforma *CEventHandler* på ett sätt att den mappar händelser till funktionsanrop. Denna utformning påminner en del om den som används vid icke-COM-relaterad händelsehantering som beskrevs tidigare. Följande bör placeras i cpp-filen tillhörandes klassen *CEventHandler*:

```
BEGIN_DISPATCH_MAP(CEventHandler, CCmdTarget)
          //{{AFX_DISPATCH_MAP(CEventHandler)
          DISP_FUNCTION(CEventHandler, "TestEvent",
                    OnTestEvent, VT_EMPTY, VTS_NONE)
          //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

Programkoden ovan kopplar COM-objektets händelse *TestEvent* till metoden *OnTestEvent*. Det är mycket viktigt att *OnTestEvent* har exakt samma antal och typ av parametrar som *TestEvent* för att kopplingen skall utföras korrekt. Argumentet *VT_EMPTY* anger att metoden inte returnerar något och *VTS_NONE* anger att metoden inte har några parametrar. *OnTestEvent* är en helt vanlig metod som kan se enligt följande:

```
void CEventHandler::OnTestEvent ()
{
          // Anropas då COM-objektet skickar händelsen 'TestEvent'
}
```

3.4.2.4 Avinitiering av COM-objekt

För att avsluta kopplingen mellan COM-objektets händelser och det objekt som händelserna kopplats till måste *AfxConnectionUnadvise* användas. För avallokering av ett COM-objekt används pekaren till objektet, i detta fall *pMinKlass*, för att anropa metoden *Release*. Som tidigare nämndes används även *pdwCookie* (som tilldelades ett värde då kopplingen skapades) för att avsluta kopplingen korrekt.

När COM-biblioteket inte längre behövs skall även detta avallokeras, vilket sker med kommandot *CoUninitialize*, se nedan:

```
HRESULT hres = AfxConnectionUnadvise(pUnkSrc,
          DIID___MinKomponentklass, pUnkSink, TRUE, pdwCookie);
if (FAILED(hres))
          //(hantera fel vid avstängning av händelsehantering)
pMinKlass ->Release();
CoUninitialize();
```

3.4.3 Ett enkelt exempel

I bilaga B.3 beskrivs steg för steg ett exempel på hur en server respektive en klient kan skrivas i VC++ med hjälp av grundläggande COM-funktionalitet. Servern har en funktion ”add” som lägger samman två heltal och en egenskap ”Color” som är läs- och skrivbar. Dessutom finns en händelse ”ServerCallback”. Servern i exemplet exekveras som en out-of-process-server.

Observera att det finns många sätt att göra saker i VC++, exemplet gör inte anspråk på att vara bättre än någon annan metod. Exemplet är inte avsett att vara någon komplett handledning för VC++ och COM utan bara att fungera som en enkel fungerande klient-server-lösning.

Exemplet utför samma sak som VB-exemplet i bilaga B.2. för att läsaren ska kunna bilda sig en uppfattning om likheter och skillnader. De båda COM-komponenterna är dock inte kompatibla med varandra då deras interface skiljer sig en aning.

I exemplet visas även hur VC++ inbyggda *ClassWizard* kan användas för att koppla funktioner till modellens händelser.

3.5 Distribuering av komponenter

För att en klient skall kunna använda en COM-komponent, skapad i antingen VB eller VC++ måste komponenten vara installerad på klientdatorn och registrerad i dess systemregister. Detta kan i princip gå till på tre olika sätt:

- Komponenten har skapats och kompilerats på klientdatorn.
- Komponenten kopieras manuellt till klientdatorn och registreras på denna (med kommandot *regsvr32 filnamn.dll* för en in-process respektive *filnamn /RegServer* för en out-of-process server). Detta är ett enkelt sätt att distribuera komponenten om den inte är beroende av andra komponenter.
- Genom att skapa ett installationsprogram med hjälp av programmet *Package and Deployment Wizard* som ingår i Visual Studio. Detta kan vara ett lämpligt sätt att få med alla erforderliga filer då komponenten är beroende av andra objekt. I bilaga B.4 ges ett konkret exempel på hur det går till att skapa installationsprogram med detta program. Exemplet tar upp distribuering av den VB-server och den VB-klient som skapades i bilaga B.2.

4 Konstruktionslösning

I detta kapitel redogörs för konstruktionslösningen av rymdfärjemodellen med tillhörande klienter. Kapitlet inleds dock med en beskrivning av konceptet Model-View-Controller (MVC). Anledningen till detta är att hela implementationen är uppbyggd enligt denna filosofi och en förståelse för MVC-konceptet är därför central för att förstå varför implementationen är uppbyggd som den är. Kapitlet är inriktat på den principiella konstruktionslösningen och går inte närmare in på programkod och implementationsdetaljer etcetera, då detta kommer i stället att redogöras mera ingående i kapitel 5.

4.1 Model-View-Controller

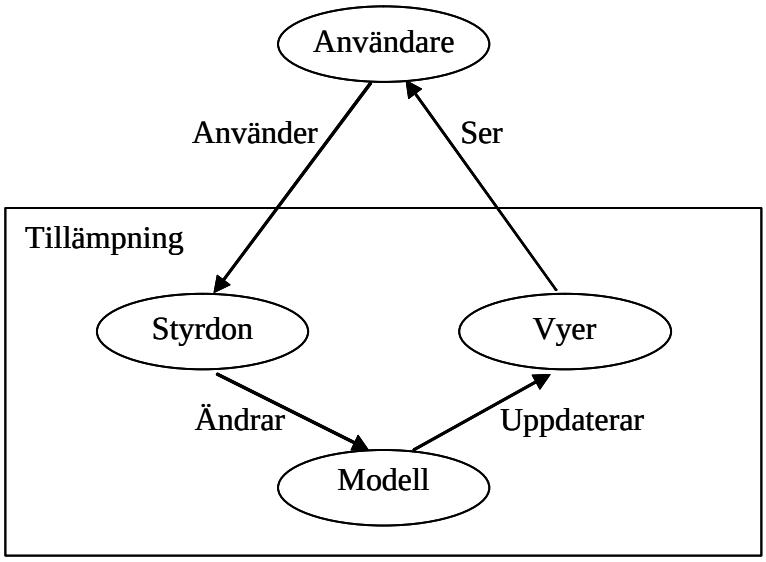
MVC är ett centralt begrepp i kursen Grafiska användargränssnitt och laborationen som skall konstrueras skall därför sträva efter att återspegla och ge större förståelse för detta konstruktionsmönster och dess fördelar. I detta avsnitt beskrivs hur MVC fungerar samt några av dess fördelar.

4.1.1 Vad är MVC?

Ett grafiskt användargränssnitt kan sägas bestå av två huvuddelar som interagerar med användaren, styrdon och vyer. Styrdonen används för att ge användaren erforderlig kontroll över händelseförloppet, till exempel trycka på en knapp för att utföra en beräkning eller dra i en rullningslist för att ändra ett värde. Vyernas uppgift är att informera användaren om det faktiska händelseförloppet, till exempel en indikator som visar återstående tid för en operation eller en textruta med information. Syftet med MVC är att isolera själva programlogiken från dessa vyer och styrdon och i stället låta denna logik utgöra en egen modell. Detta ger flera fördelar, dessa redogörs för i kapitel 4.1.2. Konstruktionsmönstret Model-View-Controller är alltså, som namnet antyder, uppbyggt omkring dessa tre delar:

- Model. Modellen innehåller all programlogik.
- View. Vyerna visar information om modellen.
- Controller. Styrdonen används för att påverka modellen.

Principen för detta koncept illustreras i figur 4.1.



Figur 4.1: Princip för MVC-konceptet

Det bör tilläggas att en komponent i ett grafiskt användargränssnitt kan utgöra både ett styrdon och en vy, till exempel ett skjutreglage som kan påverkas både av användaren och av modellen. Detta innebär inte att komponenten är någon slags hybrid utan den har två distinkta roller, då användaren påverkar den utgör den ett styrdon och då modellen påverkar den utgör den en vy.

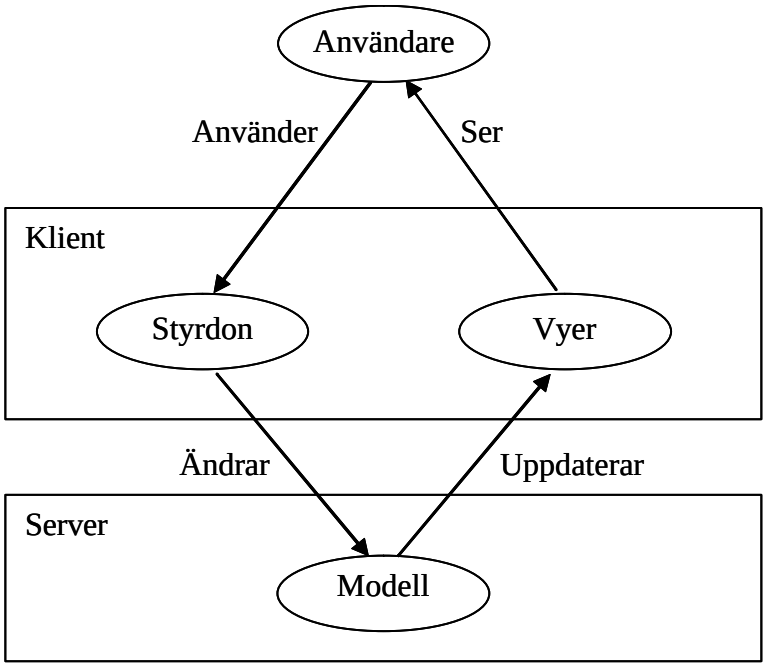
I praktiken brukar emellertid ofta en viss grad av direktkoppling kvarstå mellan styrdon och vyer. Det brukar då röra sig om logik som inte har så mycket med själva modellen att göra utan sådant som är relaterat till användargränssnittet, knappar som tonas ned etcetera.

4.1.2 Fördelar

MVC-konceptet har en rad fördelar som beskrivs närmare i detta avsnitt. En del fördelar beskrivs även på [4].

4.1.2.1 Modularitet

MVC underlättar utvecklandet av klient/server-program. Eftersom programlogiken är helt separerad från det grafiska användargränssnittet med dess styrdon och vyer kan den relativt enkelt placeras på en annan dator. En sådan klient/serverlösning illustreras i figur 4.2.



Figur 4.2: Separering av GUI och modell vid klient/server-lösning

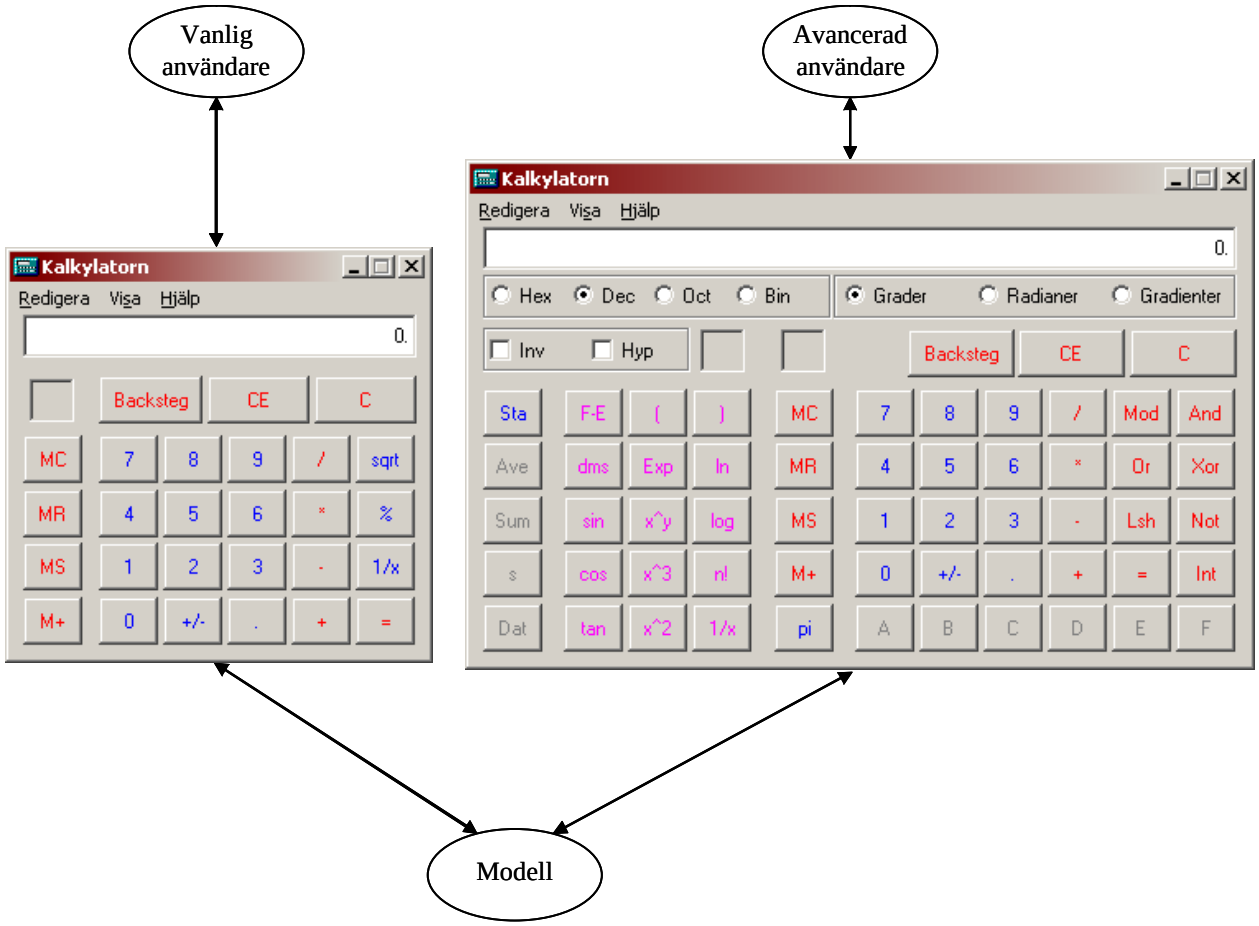
4.1.2.2 Underhållsvänligt

Den ökade modulariteten medför även underhållsmässiga fördelar genom att modellen kan ändras och uppdateras utan att användargränssnittet påverkas. Detta märks särskilt då klient/server-konceptet ovan används, uppdateringen behöver endast göras på den dator som innehåller servern.

4.1.2.3 Multipla användargränssnitt

Det går även att tänka sig en situation då en viss användargrupp behöver utökad funktionalitet. Genom att skapa ett nytt användargränssnitt för denna grupp och lägga till erforderliga funktioner i modellen kan denna ändring utföras utan att andra användare påverkas, dessa kan istället fortsätta använda det befintliga användargränssnittet och ändringen blir helt omärkbar för dessa.

Då logiken isolerats från användargränssnittet i en särskild modell medför detta att det är enkelt att skapa flera användargränssnitt och koppla dessa till modellen. Ett exempel på hur detta skulle kunna användas ges i figur 4.3.



Figur 4.3: En modell – flera användargränssnitt

4.1.2.4 Låg kopplingsgrad

MVC-konceptet leder automatiskt till en låg kopplingsgrad (som är ett mått på antal beroenden mellan olika delar av programmet), vilket är en positiv egenskap. Detta kan jämföras med en e-post-lista: En deltagare skickar ett meddelande till listan (modellen) och listan skickar detta till alla registrerade deltagare. Kopplingsgraden är fortsatt låg då fler deltagare tillkommer. Om en e-postlista däremot inte används måste varje deltagare i stället skicka sitt meddelande direkt till alla övriga deltagare. Detta medför att kopplingsgraden stiger otroligt snabbt och att mycket administrativt arbete måste utföras då deltagare tillkommer eller försvinner.

4.2 Beskrivning av uppgiften

Syftet med detta examensarbete är som tidigare nämnts att utveckla en laboration där studenter skall skapa ett grafiskt användargränssnitt föreställande en operatörspanel i en rymdfärja och koppla denna till en underliggande modell. Ett viktigt mål med denna

laboration är att ge en tydlig förståelse för MVC-konceptet och dess fördelar. I syfte att uppnå detta mål används COM-tekniken för att särskilja modell och användargränssnitt.

All logik som rör själva rymdfärjan kommer att ligga i modellen medan den som finns i användargränssnittet endast är till för att återspegla tillståndet i modellen samt åstadkomma erforderliga förändringar i gränssnittet såsom att göra en knapp icke valbar för att omöjliggöra för användaren att bryta mot ett förvillkor.

I uppsatsen ingår att utveckla både modell och användargränssnitt (klient). Modellen kommer att användas direkt av studenterna vid laborationen medan användargränssnittet huvudsakligen har två syften:

- Underlätta för laborationshandledare genom att illustrera hur modellen kan användas.
- Utveckling och testning av modellen.

Modellen skall tillhandahålla följande funktioner och information, samt notifiera alla klienter då någon av dessa ändras/inträffar:

- Aktuellt avstånd till rymden och jorden.
- Aktuell rakettemperatur och notifiera när temperaturen går från att vara för låg till att vara stabil.
- Aktuell fas.
- Återstående tid vid nedräkning.
- Ge larm då något av följande måste utföras:
    - Dörren måste stängas.
    - Rymdfärjan måste lossas från rampen.
    - Raketkraften måste ökas.
    - Rakettemperaturen är för hög och måste sänkas.
    - Rullning måste påbörjas.
    - Hjälpraketerna måste släppas.
    - Huvudraketerna måste stängas av.
    - Rymdraketerna måste slås på.
- Ge möjlighet till att ändra rymdfärjans raketkraft (hos såväl huvudraketerna som hjälpraketerna) samt återge aktuell kraft och notifiera då kraften går från att vara för låg till att vara tillräckligt stor.

- Ge möjligheten att återställa rymdfärjans tillstånd till det ursprungliga tillståndet.
- Ge möjlighen att utföra följande åtgärder:
    - Stänga rymdfärjans dörr.
    - Påbörja nedräkning.
    - Lossa rymdfärjan från rampen.
    - Lyfta från rampen.
    - Påbörja rullning.
    - Lossa hjälpraketerna.
    - Stänga av huvudraketerna.
    - Slå på rymdraketerna.

Användargränssnittet skall genom anrop till de ovan beskrivna funktionerna ge användaren möjlighet att kontrollera rymdfärjan. Genom att reagera på modellens händelser skall gränssnittet hela tiden hålla användaren underrättad om modellens tillstånd. Allt detta skall ske på ett sådant sätt att användaren inte kan krascha rymdfärjan genom felaktigt användande av gränssnittet. Dessutom skall användaren ha möjlighet att välja om varje enskilt larm skall visas eller ej. Funktionen att återställa rymdfärjans tillstånd behöver dock inte implementeras.

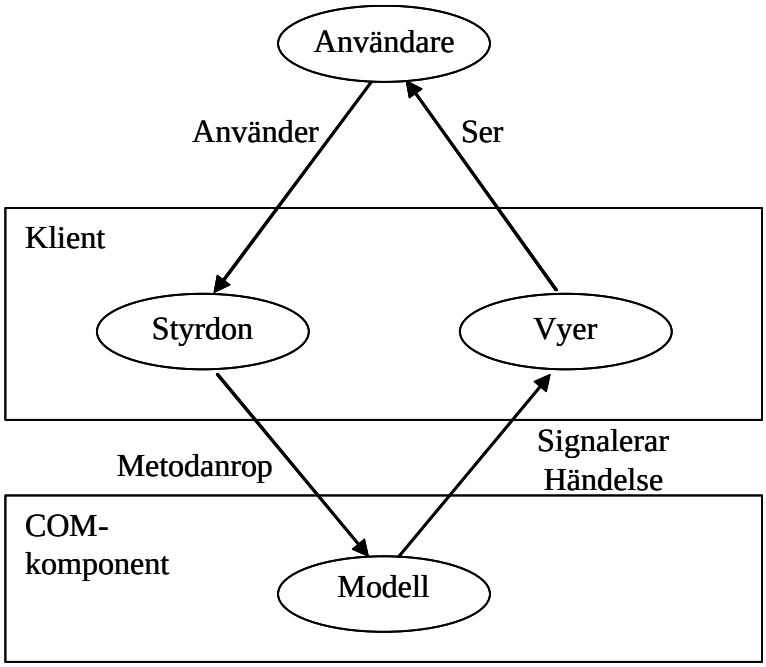
Förutom att ta fram en modell och ett användargränssnitt ingår att skriva en uppgiftsspecifikation för de studenter som skall utföra uppgiften. Denna återges i bilaga C.

4.3 Beskrivning av konstruktionslösningen

Som nämndes i kapitlets inledning är konstruktionslösningen uppbyggd enligt MVC-konceptet. För att realisera denna uppdelning mellan modell och klient används alltså COM-tekniken, närmare bestämt DCOM då det är önskvärt att kunna låta användargränssnittet ligga på en annan dator än själva logiken.

4.3.1 Generell struktur

Figur 4.4 visar hur MVC ser ut då COM används. Jämför detta med figur 4.2. Som synes lämpar sig COM väl för att implementera MVC-konceptet..



Figur 4.4: MVC med COM

För att påverka modellen använder klienten sig av metodanrop, dessa sker till modellens ingående interface. När vyerna skall uppdateras av modellen använder denna sig av det utgående interfacet för att skicka en signal till dessa vyer. Detta är en form av händelsehantering, så kallade events.

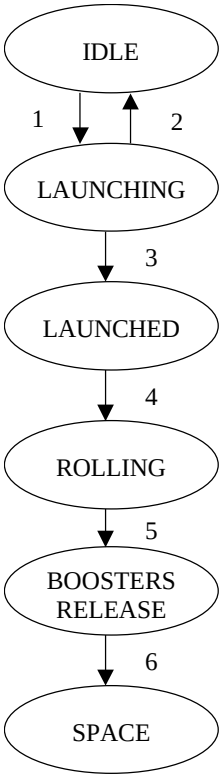
4.3.2 Modellens uppbyggnad

Målet för studenterna som utför laborationen är att implementera ett grafiskt användargränssnitt som kan interagera med modellen (rymdfärjan) på ett sådant sätt att en lyckad uppskjutning kan genomföras och färjan på ett säkert sätt kan lotsas ut i rymden.

Under sin färd från stillastående på marken till omloppsbana runt jorden kommer modellen att genomgå sex olika tillstånd, en översikt över dessa tillstånd ges i tabell 4.1. Hur tillstånden är relaterade till varandra och villkor för övergångar mellan dem åskådliggörs i figur 4.5 respektive tabell 4.2.

Fas	Beskrivning
IDLE	Rymdfärjan befinner sig på marken.
LAUNCHING	Avfyrningsstadium, färjan är redo att lyfta / befinner sig strax över marken.
LAUNCHED	Färjan har påbörjat sin färd mot rymden.
ROLLING	Rullningsfas.
BOOSTERSRELEASE	I denna fas skall hjälpraketerna släppas.
SPACE	Rymdfärjan har nått rymden

Tabell 4.1: Beskrivning av modellens tillstånd



Figur 4.5: Modellens tillstånd

Nr:	Beskrivning av tillståndsövergångar:
1.	Huvudraketer och hjälpmotorer har ett sammanlagt gaspådrag större än noll.
2.	Både huvudraketer och hjälpmotorer har gaspådraget noll, rymdfärjan har inte lyft från marken (avstånd till jord är lika med 0).
3.	Jordavståndet är lika med DISTANCE_LIFT_OFF (10 avståndsenheter).
4.	Jordavståndet är lika med DISTANCE_ROLL (1000 avståndsenheter).
5.	Jordavståndet är lika med DISTANCE_RELEASE_BOOSTERS (2000 avståndsenheter).
6.	Jordavståndet är lika med DISTANCE_SPACE (3000 avståndsenheter).

Tabell 4.2: Tillståndsövergångar i modellen

Modellen lagrar och uppdaterar kontinuerligt all relevant information såsom vilka larm som är aktuella. Syftet med detta är att en klient i vilket ögonblick som helst skall kunna koppla upp sig mot modellen och inhämta dess aktuella tillstånd. En klient skall alltså inte göra några egna antaganden om modellen, såsom att rymdfärjan skulle befinna sig på marken etcetera.

Då ett fel uppstår i modellen ges en signal om detta samtidigt som samtliga larm ges. I praktiken inträffar detta då ett förvillkor brutits. Att kontrollera förvillkor inuti en funktion strider till viss del mot principen med för- och eftervillkor men av pedagogiska skäl har detta ändå gjorts för att ge studenterna omedelbar respons då ett förvillkor inte uppfylls. Efter att dessa signaler givits kommer modellens fortsatta exekvering vara odefinierad.

4.3.3 Vyernas och kontrollernas uppbyggnad

Vyernas och kontrollernas uppgift är att återspegla respektive kontrollera modellen. I praktiken är dessa samlade i ett grafiskt användargränssnitt (”fönster”). Uppbyggnaden av vyer och kontroller i konstruktionslösningen baseras i huvudsak på tre saker:

- Modellens funktionalitet (in- och utgående interface, för- och eftervillkor).
- Interagerandet med användaren, med syfte på användbarhet och enkelhet.
- De skall fungera som ett slags skydd och hindra användaren från att bryta mot modellens förvillkor etcetera.

Detaljerna kring detta hör i hög grad ihop med implementationslösningen och beskrivs vidare i kapitel 5.

5 Implementation och test

I föregående kapitel gavs en principiell översikt över konstruktionslösningen, detta kapitel kommer att beskriva lösningen mera i detalj med avseende på både server och klient (där servern utgörs av modellen och klienten av det grafiska användargränssnittet). Kapitlet beskriver också rent praktiskt hur komponenten konfigureras för att fungera på en dator. Slutligen ges en beskrivning av den testning som gjorts för att i största möjliga mån säkerställa att laborationen är praktiskt möjlig att genomföra.

5.1 Serverdelen

COM-servern är skriven i VB, och innehåller två komponentklasser, ShuttleModel och Connector som beskrivs nedan.

5.1.1 Komponentklassen Connector

Connectorklassens har skapats i syfte att fullgöra en enda uppgift, att tilldela klienter en och samma instans av ShuttleModelklassen. Anledningen till att Connectorklassen måste finnas är att alla klienter skall återspegla samma modell. Då det, som tidigare nämnts, i VB inte är möjligt att lösa detta genom att implementera en egen klassfabrik har istället denna klass skapas för att sköta uppgiften.

Ett globalt objekt tillhörande ShuttleModelklassen har skapats i en särskild kodmodul, detta globala objekt används av alla Connectorobjekt och instansieras första gången ett Connectorobjekt skapats. Klienter får tillgång till detta ShuttleModelobjekt genom att anropa metoden *GetShuttleModel* i Connectorklassen.

5.1.2 Komponentklassen ShuttleModel

Klassen ShuttleModel utgörs av själva implementationen av rymdfärjan. Klienterna har alltså åtkomst till klassen ShuttleModel men kan inte själva skapa instanser av denna, då klassen har definierats som PublicNotCreatable. Därmed säkerställs att instansieringen måste ske med hjälp av Connectorklassen och att endast en instans kan existera.

5.1.3 Modellens interface

Komponentklasserna Connector och ShuttleModel implementerar varsitt ingående interface i form av publika metoder, ShuttleModel har dessutom ett utgående interface som utgörs av händelser (events). Dessa interface beskrivs närmare i bilaga D.

I bilaga D anges funktionerna på samma form som de skrivs i VB och inte på formen med HRESULTs och utparametrar. Om verktyget OLE View används för att studera modellens interface framgår att dessa ändå kommer att få denna standardform.

5.1.3.1 Modellens ingående interface

För att rymdfärjans klienter skall kunna kommunicera med modellen implementerar modellen de funktioner som beskrivs i bilaga D.1. Anrop till funktionerna i bilaga D.1.1 är det enda sättet för en klient att påverka modellen medan funktionerna i bilaga D.1.2 enbart är till för att låta klienter avläsa modellens tillstånd. I bilagan anges även de för- och eftervillkor som klienterna måste anpassa sig till för att en säker rymdfärd skall kunna genomföras utan missöden.

5.1.3.2 Modellens utgående interface

Förutom ett ingående interface har även modellen ett utgående sådant, detta för att kunna ta initiativ till en asynkron kommunikation med sina klienter. De klienter som önskar använda sig av detta måste implementera de funktioner som beskrivs i bilaga D.2.

5.1.4 Komponenten XTimers

Då ett flertal händelser (avståndsändring med flera) i modellen är tidsberoende är det nödvändigt att använda någon form av timer för detta. (En timer kan sägas vara en form av tidtagare som larmar efter en viss tid, periodiskt eller en enstaka gång.) VB har en inbyggd sådan som dock har den nackdelen att den måste bäddas in i ett formulär. Detta hade inneburit en nackdel då något formulär inte existerar i serverimplementationen och ett dolt sådant därför skulle ha behövt skapats. Vid praktiska prov visade det sig även att denna timer orsakade problem genom att servern fortsatte exekvera även då alla klienter kopplat ifrån. Med anledning av detta föll valet på att istället använda komponenten XTimers.

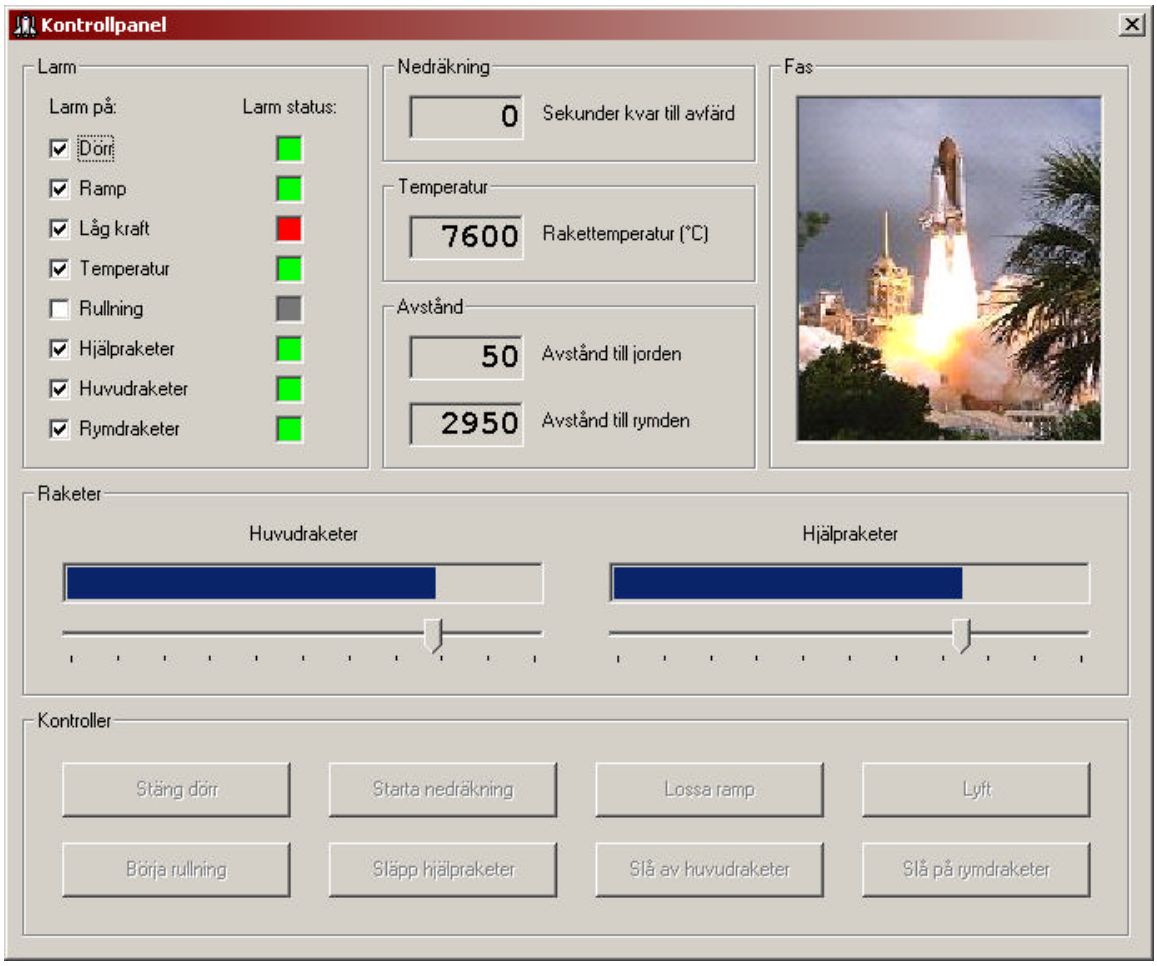
XTimers är en in-process COM-server (alltså implementerad i en DLL-fil). Denna behöver inte skapas i något formulär, utan kan instansieras och användas som en vanlig klass. XTimers är en del av projektet Coffee, som är ett av de många kodexempel som finns tillgängliga för VB, se [21].

5.2 Klientdelen

När det gäller klientdelen så motsvarar denna den del av laborationen som studenterna skall utföra. Detta medför att en ingående beskrivning av denna inte kan ges här, dock kommer en viss översikt att ges. Den som är intresserad av detaljer hänvisas istället till att kontakta den kursansvarige för kursen Grafiska användargränssnitt (kursansvarig när detta skrivs är Hannes Persson, hannes.persson@kau.se). Klientdelen har liksom serverdelen implementerats i VB, men den har även implementerats i VC++.

5.2.1 Grafiskt användargränssnitt

Det grafiska användargränssnitt som skapats till klienten som en del i uppsatsskrivandet visas i figur 5.1. Bilden visar det gränssnitt som skapats i VB men det som skapats i VC++ är uppbyggt på exakt samma sätt.



Figur 5.1: Klientens grafiska användargränssnitt

5.2.1.1 Mät- och styrdon

Gränssnittet innehåller följande vyer:

- Status för de olika larmen. Om ett larm är aktuellt eller ej indikeras med de traditionella färgerna rött och grönt för att ingen tveksamhet skall råda. Ett frånkopplat larm indikeras med grå färg. Larmen har placerats så att operatören omedelbart skall upptäcka att ett larm ändrat färg till rött, men ändå en bit ut i periferin då det bedömts som viktigare att andra mät- och styrdon placerats centralt för en god överblick. Larmen har implementerats med hjälp av bildrutor.
- En bild som representerar en extern kameravy visande aktuell tillstånd. Bilden har placerats längst upp i högra hörnet då det mänskliga ögat ändå lätt kommer att uppfatta då den ändras och att operatören annars skulle kunna uppfatta att den vore i vägen.
- Digitala mätare för visning av avstånd, temperatur och nedräkning. Dessa har placerats i gränssnittets mitt (horisontellt sett) för att vara lätta att överblicka. Nedräkningsmätaren har placerats längst upp i överkant då den endast har betydelse under en kort tidsperiod. Dessa mätare har implementerats med hjälp av textetiketter (labels).
- Indikering av det aktuella gaspådraget till huvud- och hjälpraketer, implementerad med hjälp av framdriftsindikatorer. Dessa har placerats centralt för att vara lätta att överblicka.

Följande styrdon har skapats i gränssnittet:

- Kryssrutor som användaren kan markera/avmarkera för att avgöra om larm skall visas eller ej. Dessa har placerats längst upp i hörnet på gränssnittet, dels för att minimera risken att användaren råkar koppla ur ett larm av misstag dels för att ändringar inte förväntas ske ofta. Vidare har de placerats i anslutning till respektive larm.
- Skjutreglage för att reglera kraften hos huvudraketer respektive hjälpraketer. Dessa har givits en central placering i gränssnittet då de är viktiga reglage som kommer att användas frekvent. Dessa har placerats i direkt anslutning till mätarna som visar det aktuella gaspådraget.
- Knappar för att utföra speciella åtgärder. Dessa har placerats närmast operatören för att en snabb och enkel åtkomst skall vara möjlig. För att ge största möjliga tydlighet

har knapparna placerats i den ordning som är naturlig för människor i västerlandet: från vänster till höger, uppifrån och ned. För att ytterligare öka tydligheten och minimera risken för missöden är knapparna nedtonade då de inte är aktuella.

5.2.2 Grafiskt användargränssnitt för handdator

I laborationen ingår som en extrauppgift att utforma ett grafiskt användargränssnitt för en handdator med en begränsad skärmyta. Den variant som tagits fram av uppsatsförfattarna visas i figur 5.2.

Figur 5.2: Grafiskt användargränssnitt för handdator

På grund av den begränsade ytan har vissa kompromisser fått göras. Ett alternativ som övervägdes var att skapa ett fliksystem för att få plats med ytterligare information, detta valdes dock bort på grund av att det skulle komplicera gränssnittet och försvåra att snabbt åtgärda larm etcetera. Möjligheten att välja bort vissa larm har slopats då den bedömts vara mindre viktig.

I huvudsak har två kompromisser gjorts:

- Om flera larm är aktuella visas endast ett. Detta är en viss brist, men att ge möjlighet att visa samtliga larm skulle uppta orimligt mycket plats.

- Kontrolldonet för att reglera gaspådraget måste dessutom agera vy och visa aktuellt gaspådrag.

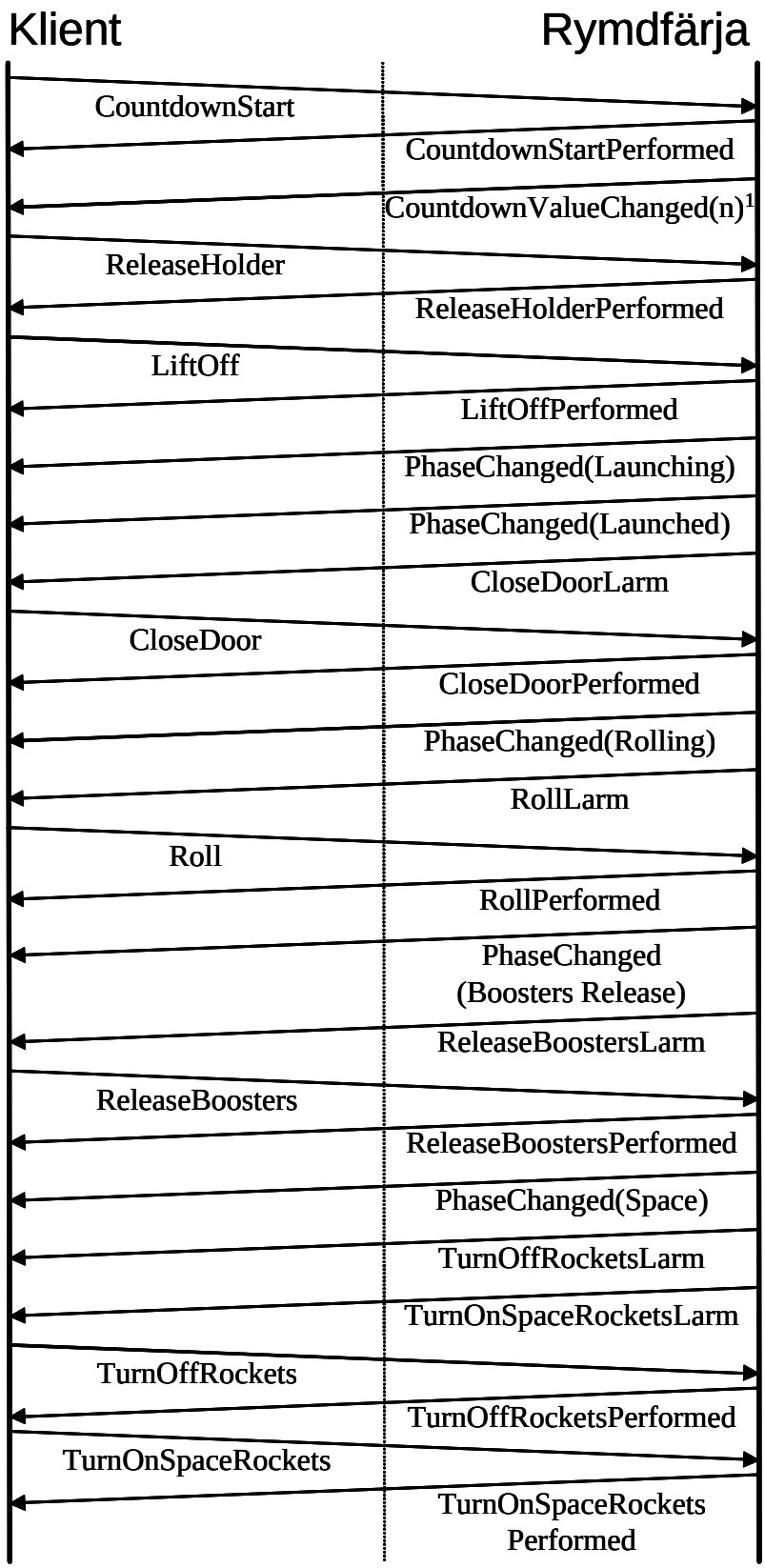
5.3 Kommunikation mellan klient och modell.

Figur 5.3 illustrerar hur kommunikationen mellan klienten och rymdfärjemodellen kan se ut. Observera att detta endast är ett exempel, det finns ett flertal olika sätt att korrekt genomföra en uppskjutning av rymdfärja på. Vidare har hanteringen av raketkraft och temperatur lyfts ut och illustreras separat i figur 5.4 och 5.5. Fasövergångarna som är inlagda i sekvensdiagrammet nedan kan tyckas uppstå spontant, men de sker då de villkor som tidigare specificerades i tabell 4.2 har uppfyllts. Alla illustrationer visar hur kommunikationen sker då det endast finns en klient som arbetar mot rymdfärjan, men det kunde lika gärna ha varit flera klienter. De signaler som rymdfärjan skickar ut tas nämligen emot av alla klienter.

Figur 5.4 visar vad som händer då rymdfärjan skall lyfta, men då raketerna inte har startats. Det hela börjar med att rymdfärjan skickar signalen *ForceLarm* som talar om för klienten att den måste öka raketkraften. I figuren ökar klienten huvudraketkraften till 95, och rymdfärjan skickar tillbaka en signal som indikerar att den har ökats till 95. Därefter utförs motsvarande med hjälpraketkraften. I och med detta har nödvändig kraftmängd uppnåtts, vilket leder till att rymdfärjan kan lyfta. Dessutom skickas signalen *ForceStable* till klienten. Under hela den tid då raketkraften är mer än noll skickar rymdfärjan signalerna *DistanceEarthChanged(m)* och *DistanceSpaceChanged(m)* till klienten, där m är antalet avståndsenheter. Eftersom det inte är möjligt att illustrera dessa signaler varje gång rymdfärjan skickar dem (eftersom de kontinuerligt då villkoret ovan är uppfyllt), så visas de endast första gången de skickas. Figur 5.4 visar slutligen hur signalen *TemperatureChanged(k)* fungerar, där k är temperaturen. Denna signal skickas ut 2 sekunder efter att antingen huvudraketkraften eller hjälpraketkraften har ändrats. Raketkraften kan när som helst under resans gång ändras av användaren.

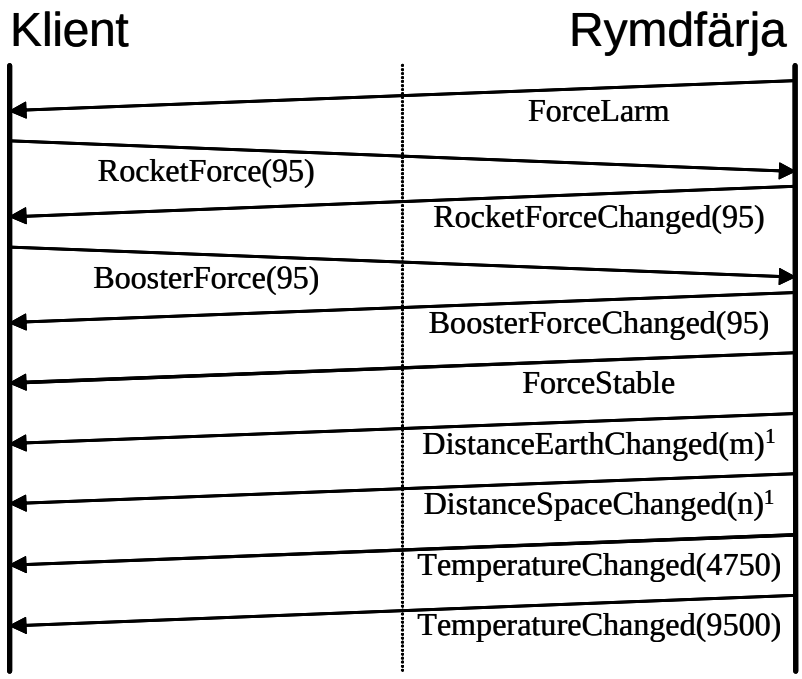
Figur 5.5 visar vad som händer då rymdfärjans tillstånd övergår till *Rolling*. Raketkrafterna antas ha de värden de blev tilldelade i figur 5.4. Om raketkrafterna är för stora, blir temperaturen för hög. När fasen rolling inleds ges signalen *TemperatureLarm* om temperaturen är 9000 grader eller högre. Som figur 5.5 illustrerar så sänker klienten raketkrafterna och därmed temperaturen. Precis som tidigare skickar rymdfärjan signalen *TemperatureChanged* först två sekunder efter någon raketkraft har ändrats. Om temperaturen understiger 9000 grader skickas signalen *TemperatureStable*, vilket indikerar att raketemperaturen inte längre är för hög.

För en utförligare beskrivning av modellens metoder och händelser, se bilaga D. Läsaren uppmanas även att ta en titt på bilaga E som innehåller den kod som studenterna får utgå ifrån.



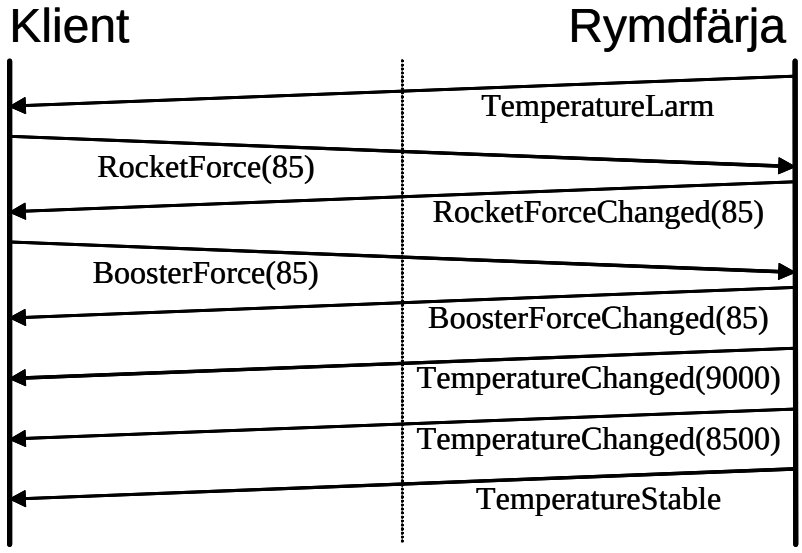
[1]: Upprepas 10 ggr, n räknas ner från 9 till 0

Figur 5.3: Sekvensdiagram över kommunikationen mellan klient och rymdfärja



[1] m och n är antalet avståndsenheter. Rymdfärjan skickar dessa signalerar kontinuerligt då den sammanlagda raketkraften är större än 0.

Figur 5.4: Sekvensdiagram över hanteringen av raketkraft och temperatur



Figur 5.5: Sekvensdiagram över hanteringen av temperaturlarm och raketkraft

5.4 Installation och konfigurering

I laborationen är det i första hand tänkt att DCOM ska användas. Detta orsakar två problem som är relaterade till de begränsade rättigheter som är knutna till studenternas datorkonton:

- Verktyget dcomcnfg kan inte användas för att konfigurera rättigheter för start och exekvering av servern.
- Package & Deployment Wizard kan inte användas för att registrera komponenten på klientdatorn och kopiera nödvändiga DLL-filer.

Lösningen på dessa problem blev att gå runt problemen genom att manuellt kopiera nödvändiga filer och manipulera systemregistret. För att studenterna enkelt skall kunna utföra dessa operationer har två batchfiler (en batchfil är ett exekverbart script) skapats:

- *server.bat* skall köras på den dator där servern skall exekvera för att kopiera och registrera komponenten (och även Xtimers-komponenten) samt sätta säkerhetsnivån.
- *client.bat* skall köras på klientdatorer för att kopiera serverns TLB-fil (nödvändig för att kunna kompilera klientprojektet) och en nödvändig DLL-fil, samt registrera servern. Då filen körs måste namnet på den dator där servern finns implementerad anges som en parameter, exempelvis *client b03*. Observera att om en klient skall exekvera på samma maskin som servern skall endast server.bat köras.

Då dessa batchfiler tagits fram på experimentell väg genom att undersöka systemregistret före och efter att respektive verktyg använts så är de känsliga för annorlunda konfigurationer. Ett krav för att de skall fungera är att VB finns installerat.

Dessa problem berör endast DCOM, då lokal COM används krävs endast rättigheter att utföra en manuell komponentregistrering (*regsvr32 xtimers.dll* och *SpaceShuttle /RegServer*). Denna komponentregistrering kan utföras med batchfilen *local.bat*.

5.5 Testning

För att i största möjliga mån säkerställa att den utvecklade laborationen är praktiskt användbar och för att minimera risken för fel i modellen har implementationen fått genomgå olika typer av tester. Dessa tester kan delas in i två kategorier:

- Allmän testning för att säkerställa att modellens funktioner fungerar korrekt. Denna testning har till största delen bedrivits på de datorer som är tillgängliga för datavetenskapsstudenter vid Karlstads universitet.
- Kompatibilitetstestning då implementationen testats under olika versioner av Windows. Detta är av vikt då en uppgradering från Windows NT till Windows XP planeras för de aktuella studentdatorerna.

De två kategorierna beskrivs mer ingående i följande avsnitt.

5.5.1 Allmän testning

Den allmänna testningen av modellens funktionalitet har främst utförts genom att genomföra ett antal flygningar med hjälp av de framtagna klienterna. Dessa tester omfattar:

- Kompletta färder från start till mål.
- Test av händelsehantering genom att flera klienter har använts samtidigt och konstaterats reagera korrekt på modellens händelser.
- Plötsligt avslutande av klienter i olika stadier (till exempel under pågående nedräkning) för att kontrollera att förbindelsen avslutas korrekt.
- ”Aptest”, det vill säga att trycka på diverse tangenter, trycka på slumpmässiga tangenter och så vidare.

Dessa tester har alltså utförts med hjälp av de framtagna klienterna, de klienter som studenterna kommer att utveckla kommer givetvis inte att ha en identisk funktionalitet. Modellens förvillkor medför dock att korrekta funktionsanrop endast kommer att kunna ske i motsvarande ordning även då andra klienter används och testresultaten kan därför anses gälla rent allmänt.

Testningen indikerar inte några problem med att installera och använda komponenten på den datorkonfiguration som finns i flertalet av de datasalar som institutionen disponerar.

Det finns dock en tänkbar situation då ett problem kan uppstå, eftersom en viss fördröjning föreligger (särskilt då DCOM används) är det möjligt att följande scenario kan uppstå:

1. Klient A anropar foo i modellen, foo är en funktion som bara får anropas en gång.
2. Modellen börjar underrätta klienterna i tur och ordning om att funktionen anropats.

3. Klient B anropar foo innan klienten hunnit motta händelsen att den redan inträffat, modellens förvillkor bryts och en signal om att rymdfärjan har kraschat sänds ut.

För att undvika problemet skulle det i detta fall vara nödvändigt att frångå principen med förvillkor och istället flytta kontrollen från klienterna till modellen. Då detta av pedagogiska skäl inte är önskvärt har denna brist helt enkelt accepterats.

5.5.2 Testning av windows-kompatibilitet

Modellen har testats under följande operativsystem:

- Windows NT 4.0
- Windows 2000
- Windows XP Professional

Testning mellan olika operativsystem (till exempel serverdator med NT och klientdator med XP) har endast förekommit sporadiskt då det är osannolikt att laborationen kan komma att utföras under sådana förutsättningar.

För att testningen enkelt skulle kunna genomföras användes programvaran VMware Workstation som används för att simulera flera datorer på en enda fysisk dator. För en närmare beskrivning av denna programvara hänvisas till [29]. Därefter installerades två exemplar av respektive operativsystemen på varsin virtuell dator. Dessa virtuella datorer förbereddes sedan enligt följande:

- Operativsystem installerades (inklusive senaste service pack)
- VB installerades (för att undvika att diverse DLL-filer senare skall behöva läggas till).
- Ett antal användare med olika rättigheter skapades.

Då dessa förberedelser gjorts konfigurerades VMware till att därefter återställa varje virtuell maskin till detta läge vid uppstart för att på så sätt möjliggöra experimenterande utan att lämna bestående registerinställningar etcetera.

Vid kompabilitetstestningen har endast modellen och VB-klienten använts, VC++-klienten och handdatorn har ej testats. Komponenter installerades och registrerades med hjälp av de batchfiler som nämndes i 5.4.

Vid testningen har dels en lokal klient startats på den dator där komponenten finns och dels har DCOM testats genom att en klient startats på en annan dator. Dessa har sedan exekverat samtidigt för att säkerställa att händelsehanteringen fungerar korrekt. Därefter har den andra datorn fått agera server i motsvarande test, detta för att säkerställa att inga kvarvarande inställningar skall ställa till problem då komponenten tidigare varit lokalt registrerad.

Testningen har visat att implementationen fungerar under följande förutsättningar:

- Windows NT 4.0: En användare tillhörande gruppen *Användare* (eller högre) kan installera komponenten (med hjälp av de batchfiler som skapats).
- Windows 2000: Användare tillhörande gruppen *Privilegierade användare* (eller högre) kan installera komponenten. Gruppen *Användare* kan inte installera komponenten på grund av begränsade rättigheter till systemregistret.
- Windows XP Professional: Av de två standardgrupperna kan endast gruppen *Datoradministratör* installera komponenten. För att en normalanvändare skall kunna genomföra installationen krävs att den systemansvarige genomför ändringar i systemets konfiguration.

Vid testerna har en gemensam arbetsgrupp använts, vid större företag och institutioner såsom Karlstads universitet är det normala istället att domäner används, se till exempel [15]. För att använda sig av domäner krävs tillgång till en serverversion av något av de ovanstående operativsystemen, med påföljd att detta av praktiska skäl inte har kunnat testas. Ett par punkter angående detta bör nämnas:

- Domäner ger större möjligheter till olika konfigurationer då det gäller rättigheter knutna till användare/grupper. De minimirättigheter som angivits i föregående lista hänvisar till de lokala grupper som finns i en standardinstallation av operativsystemet. Minimirättigheter för att självständigt kunna genomföra laborationen är att manuellt kunna registrera en COM-komponent och rättigheter att lägga till registerposter.
- När det gäller användare som skapats lokalt i en Windows XP-installation är det standard att gästkontot används vid DCOM, vilket leder till att autentiseringen i detta fall misslyckas. Denna inställning kan manuellt ändras av en administratör genom att starta verktyget *Administrationsverktyg* och sätta egenskapen *Lokal säkerhetsprincip – Lokala principer – Säkerhetsalternativ – Nätverksåtkomst: Delning och*

säkerhetsmodell för lokala konton till *Klassisk*. Detta bör inte få någon inverkan på laborationen men stöd för att ändra denna inställning har ändå lagts in i de batchfiler som skapats. (Genom att anropa dessa med kommandot *server.bat xp* respektive *client.bat <servernamn> xp*.)

6 Erfarenheter och rekommendationer

Som rubriken antyder ägnas detta kapitel åt att redogöra för de erfarenheter som uppsatsförfattarna tillgodogjort sig under arbetets gång och de rekommendationer som kan ges med utgångspunkt från dessa erfarenheter.

Dessa erfarenheter omfattar i huvudsak COM-teknologin generellt samt skillnader och likheter då det gäller VB och VC++. Denna jämförelse görs dels med avseende på hur COM-teknologin används i respektive miljö men även mera generellt angående till exempel händelsehantering och användarvänlighet.

Då kapitlet baseras på författarnas egna upplevelser och åsikter får det givetvis anses vara subjektivt. Läsaren uppmanas därför att bilda sig en egen åsikt.

6.1 COM

COM är en teknik som lämpar sig mycket väl för att separera modell och användargränssnitt och därigenom realisera MVC-konceptet. När det gäller händelsehanteringen får dock callbacks anses ha vissa svagheter då principen med att i tur och ordning synkront anropa funktioner i samtliga klienter inte känns särskilt optimal.

Vid användning av DCOM så krävs en hel del säkerhetsinställningar för att allt ska fungera. Dessa inställningar bör, förutom att medge användande av komponenten, även säkerställa att inga säkerhetsrisker uppstår. Under utvecklingen av laborationen krävdes en hel del testande för att få detta att fungera och då har ändå inga hänsyn till säkerhetsaspekten tagits.

6.2 Jämförelse mellan Visual Basic och Visual C++

Något som slagit författarna under uppsatsskrivandets gång är den stora skillnaden i kodens innehåll. I VB syns endast den kod som programmeraren själv skrivit, i VC++ däremot är koden förutom detta full av automatiskt genererad kod, makron och kompilatordirektiv. Detta innebär förstås större inflytande i VC++ men det största intrycket är att det försämrar läsbarheten och förståelsen för koden. Användandet av automatiska guider (så kallade wizards) i VC++ förenklar en hel del, men då programmeraren senare vill ändra något blir det lätt problem att rensa bort det som dessa guider producerat.

6.2.1 Inlärningströskel

När det gäller inlärningströskeln för respektive utvecklingsmiljön finns det bara en slutsats att dra – VB är enormt mycket lättare att komma igång med.

6.2.2 Händelsehantering

Rent principiellt fungerar händelsehanteringen på samma sätt i de båda språken. Då en signal om händelse tas emot påbörjas exekveringen av en bestämd funktion. I VB är namnet på funktionen som skall anropas förutbestämt, då det alltid är på formen *NamnPåObjekt_NamnPåHändelse*, exempelvis *MyShuttle_DoorClosePerformed*. I VC++ måste användaren själv specificera vilken funktion som skall anropas då en viss händelse har inträffat, detta görs med så kallade ”Message maps” eller ”Dispatch maps”. Detta leder givetvis till mer programkod i fallet med VC++.

För att få signaler om händelser från ett objekt kan *WithEvents* användas i VB i samband med deklaration av ett objekt. Detta fungerar med såväl händelser från vanliga objekt som händelser från COM-objekt. I VC++ är detta mera omständigt och kräver en mer komplex och omfattande kod, särskilt då COM finns med i bilden.

Att deklarera en händelse i VB kan göras på en kodrad, likaså att signalera en händelse. Detta är alltså mycket enkelt i VB. Modellen, som deklarerar och sänder ut händelser, är som bekant endast utvecklad i VB, och inte i VC++. Detta gör att en jämförelse inte är möjlig här.

För att sammanfatta skillnaderna i händelsehanteringen kan sägas att VB är smidigast att använda. Vanlig händelsehantering är enkel att implementera, och steget till att använda samma principer med COM är mycket litet.

6.2.3 Användning av COM/DCOM

Användningen av COM-komponenter skiljer sig mycket åt i de olika miljöerna. I VB är det egentligen bara anropet till *CreateObject* som avslöjar att en COM-komponent används och inte en intern klass. I VC++ krävs åtminstone trettio till trettiofem rader komplex kod för att initiera / avinitiera COM och ett COM-objekt med händelser.

När det gäller själva skapandet av komponenter så är även det enklare i VB, det enda som skiljer en COM-komponent från en intern klass är att projekttypen valts till *ActiveX* (EXE eller DLL) och att några projektinställningar bör göras (komponentens namn, princip för hur GUID-nummer skall genereras, om DCOM skall kunna användas). Inte heller i VC++ krävs någon avancerad kod för att få grundläggande funktionalitet, däremot krävs en hel del

navigerande i menyer och högerklickande på objekt för att skapa komponentklasser och deras interface.

De olika trådningsmodellerna i COM är inte helt lätta att förstå sig på. Vid VB-implementationen av rymdfärjemodellen innebar inte detta något problem då några extra trådar aldrig startas upp manuellt utan allt detta sköts av VB. Att implementera motsvarande modell i VC++ skulle dock ha blivit problematiskt, användandet av timers skulle ha medfört att flera trådar behövts och att modellen behövt signalera händelser från olika trådar.

7 Slutsatser

Arbetet har resulterat i en väl fungerande implementation som bör kunna komma till praktisk användning i en laboration. Som tidigare nämnts i uppsatsen finns vissa frågetecken angående studenternas framtida rättigheter till systemregistret men dessa ligger utanför författarnas kontroll. Ambitionen att i VC++, liksom i VB, utveckla både klient och modell (server) övergavs i ett relativt tidigt skede. Detta då implementationen skulle ha upptagit väsentligt mera tid och att den modell som utvecklats i VB är fullt användbar även för VC++-klienter. Av den anledningen utvecklades endast klientdelen i VC++.

En positiv överraskning har varit att några större avvikelser från tidsplanen inte har förekommit. Dock har dokumenteringen upptagit en större andel av tiden än planerat.

Arbetet har även resulterat i grundläggande kunskaper om COM-tekniker och deras användningsområde samt fördelar och nackdelar.

Vidare har en större insikt om fördelarna med MVC-konceptet erhållits. Författarna har även fått en viss inblick i de båda utvecklingsmiljöerna VB och VC++.

Den enda större svårigheten under arbetets gång har varit att få implementationen att fungera med hjälp av DCOM, detta krävde en hel del efterforskningar och experimenterande innan det fungerade korrekt.

Referenser

[1] Shoaib Ali, *A very simple COM server without ATL or MFC*, http://www.codeproject.com/com/com_server_without_mfc_atl.asp, 2000-08-04.

[2] Dan Appleman, *ActiveX: A Historical (but Technical) Perspective*, http://sunsite.iisc.ernet.in/virlib/activex/devvb/ch2.htm, 2003-02-19.

[3] Ranjan Banerji, *A Beginners Tutorial for Connection Points using VC++ and ATL*, http://www.codeproject.com/useritems/connection.asp, 2002-04-11.

[4] Cristobal Baray, *The model-view-controller (MVC) design pattern*, http://www.cs.indiana.edu/~cbaray/projects/mvc2.html, 1999-03.

[5] David Chappell, *Introducing ActiveX*, http://www.chappellassoc.com/articles/article_Intro_ActiveX.html, 1997.

[6] Santiago Comella-Dorda, *Component Object Model (COM), DCOM, and Related Capabilities*, http://www.sei.cmu.edu/str/descriptions/com_body.html, 2001.

[7] DevX, *COM + MTS = COM+*, http://archive.devx.com/free/mgznarch/vcdj/1999/febmag99/commts1.asp, 2003-02-05.

[8] Ryan D'Silva, *Part 1 - What MFC is all about*, http://www.cosc.brocku.ca/~cspress/HelloWorld/1999/02-feb/mfc_corner.html, 2003-03-22.

[9] Guy Eddon & Henry Eddon, *Inside Distributed COM*, Microsoft Press, 1998.

[10] David Goldbrenner, *What is Visual C++?*, http://www.weeno.com/art/0699/89.html, 2003-03-22.

[11] Brian Hart, *DCOM D-Mystified: A DCOM Tutorial*, http://www.codeproject.com/com/HelloTutorial1.asp, 2000-08-12.

[12] George Mack, *A History of Visual Basic*, http://csd1.dawsoncollege.qc.ca/~gmack/info/VBHistory.htm, 2003-02-20.

[13] Team Iseran, *History of the Windows API*, http://www.iseran.com/Win32/FAQ/history.html, 2003-04-02.

[14] Microsoft, *CLSCTX*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/cme_a2d_152w.asp, 2003-03-23.

[15] Microsoft, *Domains*, http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/server/sag_adintro_15.asp, 2003-04-10.

[16] Microsoft, *Fireev.exe Fires Events from a Second Thread*, http://support.microsoft.com/default.aspx?scid=kb;EN-US;157437, 2002-07-03.

[17] Microsoft, *Microsoft COM Technology*, http://www.microsoft.com/com/, 2003-02-10.

[18] Microsoft, *Microsoft Foundation Class Library*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/mfchm.asp, 2003-02-12.

[19] Microsoft, *PRB: Firing Event in Second Thread Causes IPF or GPF*, http://support.microsoft.com/default.aspx?scid=KB;en-us;q196026, 2001-07-24.

[20] Microsoft, *The #import Directive*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_predir_the_.23.import_directive.asp, 2003-03-23.

[21] Microsoft, *Visual Basic, Explore the Samples*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/VBRef98/html/vbsamppage.asp, 2003-04-03.

[22] Microsoft, *Visual Studio Home Page*, http://msdn.microsoft.com/vstudio/default.asp, 2003-02-10.

[23] Ted Pattison, *Programming Distributed Applications with COM+ and Microsoft Visual Basic Second Edition*, Microsoft Press, 2000.

[24] Jeff Prosise, *COM Security Primer, Part I*, http://www.codeguru.com/activex/COMSecurity1.html, 2000-11-13.

[25] Ash Rofail & Yasser Shohoud. *Mastering COM and COM+*, Sybex Inc, 1999.

[26] Dale Rogerson, *Inside COM*, Microsoft Press, 1997.

[27] TrollTech, *TrollTech - Qt - Overview*, http://www.trolltech.com/products/qt/index.html, 2003-02-03.

[28] Visual Basic Internet Programming, *Designing Business Objects*, http://www.vbip.com/books/186100107X/chapter_107X_02.asp, 2003-02-05.

[29] VMware, *VMware*, http://www.vmware.com, 2003-04-10.

A Förkortningsförteckning

AFX	Application Framework X
API	Application Programming Interface
ATL	Active Template Library
CLSID	Class IDentifier
COM	Component Object Model
DCOM	Distributed COM
DDE	Dynamic Data Exchange
DLL	Dynamic Link Library
GUI	Graphical User Interface
GUID	Globally Unique Identifier
IDL	Interface Definition Language
IID	Interface IDentifier
LIBID	Library IDentifier
LRPC	Local RPC
MAC	Media Access Control
MFC	Microsoft Foundation Classes
MIDL	Microsoft IDL
MTS	Microsoft Transaction Server
MVC	Model-View-Controller
OCX	OLE Custom Control
ODBC	Open Database Connectivity
OLE	Object Linking and Embedding
ORPC	Object RPC
OSF	Open Software Foundation
RPC	Remote Procedure Call
TLB	Type Library
UUID	Universally Unique Identifier
VB	Visual Basic
VC++	Visual C++
VBX	Visual Basic Extensions

B Exempel

B.1 Exempel på IDL-fil

```
/*Kommentar*/

  [
       object,
       uuid(EDD8BBC0-9240-11CF-9ED3-00AA004C120C),
       dual,
       helpstring(“Mitt Interface1”),
  ]
     interface IMittObj : IDispatch
     {
          import “oaidl.idl”;

          HRESULT MinMetod(
          [in]                    LONG parameter1,
          [in, string]            BSTR parameter2,
          [in, out]               LONG* parameter3,
          [out]                   LONG* parameter4);
     };
     [
     uuid(336c8c70-a62b-11d0-ad5f-00aa00a219aa),
     version(1.0),
     helpstring(“MittObj 1.0 Type Library”)
]
library MittObjLib
{
     importlib(“stdole2.tlb”);

     [
```

```
            uuid(53DEFDE0-9222-11CF-9ED3-00AA004C120C),
            helpstring("WPObj Class")
      ]
      coclass MinCoClass
      {
            [default] interface IMittObj;
      };
};
```

B.2 Exempel i Visual Basic

B.2.1 Server

1. Öppna dialogrutan *New Project* och välj *ActiveX EXE* (eftersom en out-of-process-server önskas).

2. Döp om Class1 till MinCoClass och skriv in följande kod:

```
Option Explicit

Private iColor As String

Public Event ServerCallback(sMess As String)

Public Function Add(A As Integer, B As Integer) As Integer
    RaiseEvent ServerCallback("Det är jag som är servern")
    Add = A + B
End Function

Public Property Get Color() As String
    Color = iColor
End Property
```

```
Public Property Let Color(newColor As String)
    iColor = newColor
End Property
```

3. Välj *Project -> Project1 Properties*. Skriv ”MinServer” i rutan *Project Name* och ”Min VB-server” i *Project Description*. Stäng dialogrutan med *OK*.

4. Välj *File -> Make MinServer.exe*.

5. Välj *Project -> MinServer Properties*, därefter fliken *Component*. Välj alternativet *binary compability*, detta för att inte skapa nya GUID-nummer varje gång servern kompileras om. En bock i kryssrutan *Remote Server Files* innebär att det blir möjligt att använda servern från en annan dator (DCOM).

B.2.2 Klient

1. Öppna dialogrutan *New Project* och välj *Standard EXE*.

2. Välj *Project -> References*, lägg till ” Min VB-server” och stäng dialogrutan

3. Lägg till en knapp i det automatiskt skapade formuläret och skriv in följande kod (som ser ut på samma sätt för såväl COM som DCOM):

```
Option Explicit

Dim WithEvents MittObjekt As MinServer.MinCoClass

Private Sub Command1_Click()
    On Error GoTo err1
    Set MittObjekt = CreateObject("MinServer.MinCoClass")
    MittObjekt.Color = "grön"
    MsgBox "Servern svarade att 2+2 är " _
        & MittObjekt.Add(2, 2) _
        & " och att den har lagrat färgen " & MittObjekt.Color
```

```
    Set MittObjekt = Nothing
    Exit Sub
err1:
    MsgBox "Fel"
End Sub

Private Sub MittObjekt_ServerCallback(sMess As String)
    MsgBox "Servern sa: " & sMess
End Sub
```

4. Kompilera projektet

Figur B.1: Callback från server

Figur B.2: Svar på funktionsanrop

B.3 Exempel i Visual C++

B.3.1 Server

1. Starta VC++ och skapa ett nytt projekt med hjälp av *ATL COM AppWizard*. Ange ”MinServer” som projektnamn och välj OK.

2. Välj servertyp *Executable (EXE)* och sedan *Finish*.

3. Välj *Insert -> New ATL Object* i menyn, därefter *SimpleObject* och *Next*. Ange “MinCoClass” i rutan *Short Name*. Bocka i kryssrutan *Support Connection Points* under fliken *Attributes*. Stäng därefter dialogen med *OK*.

4. Gå till fliken *ClassView*. Högerklicka på *IMinCoClass* och välj *Add Property*. Välj *Property Type* = ”BSTR” och *Property Name* = “Color”. Stäng dialogen med *OK*.

5. Högerklicka åter på *IMinCoClass* och välj *Add Method*. Ange *Method Name* = ”Add” och *Parameters* = “[in] int A, [in] int B, [out, retval] int *Result”. Stäng dialogen med *OK*.

6. Högerklicka på *_IMinCoClassEvents* och välj *Add Method*. Välj *Method Name* = ”ServerCallback” och *Parameters* = “[in] BSTR sMess”. Stäng dialogen med *OK*.

7. Gå till fliken *FileView*. Högerklicka på *MinServer.idl* och välj *Compile MinServer.idl*.

8. Återgå till fliken *ClassView*. Högerklicka på *CminCoClass* och välj *Implement Connection Points*. Bocka i *_IMinCoClassEvents*.

9. Öppna *MinCoClass.cpp* och redigera filen så att dess metoder får nedanstående utseende:

```
STDMETHODIMP CMinCoClass::get_Color(BSTR *pVal)
{
  *pVal = SysAllocString(m_Color);
  return S_OK;
}

STDMETHODIMP CMinCoClass::put_Color(BSTR newVal)
{
  m_Color = newVal;
  return S_OK;
}

STDMETHODIMP CMinCoClass::Add(int A, int B, int *Result)
{
  BSTR message = SysAllocString(L"Det är jag som är servern");
  Fire_ServerCallback(message);
  *Result = A + B;
  return S_OK;
}
```

10. Öppna filen MinCoClass.h. Leta rätt på raden:

```
CONNECTION_POINT_ENTRY(IID__IMinCoClassEvents)
```

Modifiera den så att det istället står:

```
CONNECTION_POINT_ENTRY(DIID__IMinCoClassEvents)
```

Lägg även till nedanstående kod sist i klassen:

```
private:
  BSTR m_Color;
```

11. Kompilera.

B.3.2 Klient

1. Skapa ett nytt projekt med hjälp av *MFC AppWizard (exe)*. Ange ”MinKlient” som projektnamn och välj *OK*. Välj att skapa en dialogbaserad tillämpning, acceptera övriga värden genom att välja *Finish*. (Då rutan *Automation* bockas i skapas automatiskt klassen *CMinKlientDlgAutoProxy* där serverdelens callback-interface kan implementeras, i detta exempel kommer istället klassen *CMinEventKlass* att skapas manuellt för detta ändamål, se punkt 3 nedan.)

2. Öppna filen ”StdAfx.h” och infoga de rader nedan som markerats med fet stil (anpassa sökvägen vid behov).

```
#include <afxcmn.h> // MFC support for Windows Common Controls
#include <atlbase.h> // Support for CComPtr< >
#include <afxctl.h>  // MFC support for Connection Points
#import   "../MinServer/MinServer.tlb"   no_namespace   named_guids
   raw_interfaces_only
#endif // _AFX_NO_AFXCMN_SUPPORT
```

3. Välj *View -> ClassWizard* i menyn. Klicka på knappen *Add Class* och välj *New* i popup-menyn. Ange ”CMinEventKlass” som namn och att klassen ska ärva *CCmdTarget* (i rutan *Base Class*). Se till att *Automation* är valt. När dialogen stängs dyker det upp ett varningsmeddelande om att ClassWizard inte kunde lägga till klassinformation till projektets ODL-fil, detta saknar i det här fallet betydelse

(Orsaken är att någon ODL-fil inte finns eftersom stöd för automatisering inte valdes under punkt 1.)

4. Öppna ” MinEventKlass.cpp”. och sök upp följande kod:

```
BEGIN_INTERFACE_MAP(CMinEventKlass, CCmdTarget)
   INTERFACE_PART(CMinEventKlass, IID_IMinKlient, Dispatch)
END_INTERFACE_MAP()
```

Ändra den så att det istället står:

```
BEGIN_INTERFACE_MAP(CMinEventKlass, CCmdTarget)
   INTERFACE_PART(CMinEventKlass, DIID__IMinCoClassEvents, Dispatch)
END_INTERFACE_MAP()
```

Inkludera även headerfilen för klassen som hanterar händelser:

```
#include "MinEventKlass.h"
```

5. Öppna “MinEventKlass.h” och lägg till den fetstilta texten nedan

```
class CMinEventKlass : public CCmdTarget
{
   friend class CMinKlientDlg;
   DECLARE_DYNCREATE(CMinEventKlass)
```

6. Välj åter *View -> ClassWizard* och gå till fliken *Automation*. Se till att klassen *CMinEventKlass* är vald och klicka på knappen *AddMethod*. Ange följande data: *External Name* = ”ServerCallback”, *Internal Name* = ”ServerCallback”, *Return type* = ”void”. Lägg även till parametern ”sMess” av typen ”LPCTSTR” i parameterlistan. Acceptera genom att välja *OK*.

7. Öppna filen ”MinEventKlass.cpp” och lägg till den fetstilta koden nedan:

```
void CMinEventKlass::ServerCallback(LPCTSTR sMess)
{
   CString mess = "Servern sa: ";
   mess += sMess;
   AfxMessageBox(mess, MB_ICONINFORMATION);
}
```

8. Välj fliken *ResourceView* och öppna dialogen *IDD_MINKLIENT_DIALOG*. Ta bort de kontroller som skapats automatiskt och lägg till en egen knapp i formuläret. Dubbelklicka på knappen för att lägga till en medlemsfunktion, acceptera det förvalda namnet *OnButton1* genom att välja *OK* och modifiera metoden *OnButton1()* så att den ser ut som koden nedan. Koden är avsett för att användas tillsammans med ett COM-objekt som ligger på samma dator som klienten. Den enda modifikationen som behövs för att instansiera ett COM-objekt som ligger på en annan dator är att ändra det tredje argumentet metoden `CoCreateInstance` från `CLSCTX_LOCAL_SERVER` till *CLSCTX_REMOTE_SERVER.*

```
void CMinKlientDlg::OnButton1()
{
   //Initiera COM
   CoInitialize(NULL);

   //Skapa en instans av MinCoClass och
   //lagra en interfacepekare
   IMinCoClass* pMinCoClass=NULL;
   HRESULT hr=CoCreateInstance(
      CLSID_MinCoClass,NULL,
      CLSCTX_LOCAL_SERVER,
      IID_IMinCoClass,
      reinterpret_cast<void**>
      (&pMinCoClass));

   //Kontrollera resultatet
```

```
if (FAILED(hr))
{
   MessageBox("Fel vid skapande av serverobjekt.",
   "Fel",MB_OK + MB_ICONSTOP);
}

//Upprätta connection point för callbacks
//från server
//Objekt som ska hand om callbacks:
CMinEventKlass m_events;
IUnknown* pUnkSink = m_events.GetIDispatch(FALSE);
//Id för förbindelsen
DWORD pdwCookie = 0;
IUnknown* pUnkSrc = NULL;
CComPtr<IUnknown> spUnk = (pMinCoClass);
pUnkSrc = spUnk.p;
//Upprätta connection point
BOOL bResult = AfxConnectionAdvise(
   pUnkSrc,
   DIID__IMinCoClassEvents,
   pUnkSink,
   TRUE,
   &pdwCookie);

if(!bResult)
   MessageBox("Callback-fel.",
   "Fel",MB_OK + MB_ICONSTOP);

CString msg = "Servern svarade att 2 + 2 är ";

//Testa metod
int* res = new int;
pMinCoClass->Add(2, 2, res);
char r[] = " \0";
msg += itoa(*res, r, 10);
```

```
   msg += " och att den har lagrat färgen ";

   //Testa get&set egenskap
   CString color = "grön";
   BSTR col = color.AllocSysString();
   pMinCoClass->put_Color(col);
   pMinCoClass->get_Color(&col);
   msg += col;
   SysFreeString(col);

   MessageBox(msg, "Info",MB_OK | MB_ICONINFORMATION);

   //Ta bort connection point
   AfxConnectionUnadvise(
      pUnkSrc,
      DIID__IMinCoClassEvents,
      pUnkSink,
      TRUE,
      pdwCookie);

   //Släpp serverförbindelsen
   pMinCoClass->Release();

   //COM har gjort sitt
   CoUninitialize();
}
```

9. Kompilera

Figur B.3: Callback från server

Figur B.4: Svar på funktionsanrop

B.4 Distribuera tillämpningen med Package and Deployment Wizard

B.4.1.1 Installationsprogram för server

Om server och klient inte skall installeras på samma dator måste ett separat installationsprogram för servern skapas enligt nedan. Om servern däremot är ett COM-objekt

som skall exekvera på samma dator som klienten är det bättre att skapa ett gemensamt installationsprogram, detta beskrivs i nästa avsnitt.

1. Välj det aktuella serverprojektet i dialogrutan som kommer upp när *Package and Deployment Wizard* startas.

2. Klicka på knappen *Package*.

3. Välj *Standard Setup Package* och klicka på *Next*.

4. Välj vart installationsprogrammet skall sparas och klicka *Next* igen.

5. Svara nej på frågan om du vill installera remote automation server files (dessa filer behövs då en äldre teknik än DCOM används).

6. Acceptera därefter alla förvalda alternativ.

B.4.1.2 Installationsprogram för klient/klient-server

Gör följande för att skapa installationsprogram för klientdelen eller ett gemensamt sådant för server och klient:

1. Använd återigen *Package and Deployment Wizard* och välj det aktuella klientprojektet.

2. Klicka på knappen *Package*.

3. Välj *Standard Setup Package*, klicka på *Next* och välj vart installationsfilerna skall sparas.

4. Nu kommer det eventuellt upp ett varningsmeddelande med titeln *Missing Dependency Information*, i detta exempel är det säkert att stänga ner den genom att välja *OK* eftersom alla erforderliga filer ändå kommer med. (Vid osäkerhet om detta kan *Package and Deployment Wizard* användas, med alternativet *Dependency File*, för att skapa en fil som lagrar beroendeinformation om projektet. Detta för att vara säker på att få med alla erforderliga filer. Detta måste göras både för serverprojektet och för

klientprojektet innan installationsprogrammet skapas.) Ta bort bocken bredvid *VBServer1.exe* i listan med filer som ska inkluderas om servern inte ska installeras på klientdatorn.

5. (Hoppa över detta steg om servern skall installeras tillsammans med klienten.) Klicka på knappen *Add* för att få upp en filväljardialog. Ändra filtyp till *Remote Server Files* och välj *VBserver1.vbr*.

6. Gå vidare genom att välja *Next*.

7. (Hoppa över detta steg om servern skall installeras tillsammans med klienten.) I nästa dialogruta kan adressen till den maskin där servern finns anges om så önskas. I annat fall får denna adress istället anges då installationsprogrammet körs.

8. Acceptera därefter alla förvalda alternativ.

C Laborationsinstruktioner för studenter

Efter olyckan med rymdfärjan Columbia har NASA beslutat att genomföra en större modernisering av samtliga rymdfärjor. I denna modernisering ingår bland annat att ta fram en ny användarvänlig operatörspanel och då endast det bästa är gott nog har man beslutat vända sig till Karlstads universitet och studenterna vid kursen Grafiska användargränssnitt för att utveckla denna operatörspanel.

C.1 Uppgiften

Uppgiften är att använda utvecklingsmiljön Visual Basic för att skapa ett grafiskt användargränssnitt till en befintlig modell (= rymdfärja). Det ingår alltså inte i uppgiften att skriva kod för själva modellen.

C.1.1 Grunduppgiften

Skapa en operatörspanel för att kontrollera och övervaka rymdfärjans färd ut i rymden. Operatörspanelen skall minst ha följande funktionalitet:

- **Vyer**: Återge antal sekunder kvar till avfyring, temperaturen på raketerna, avståndet till jorden respektive rymden, aktuellt gaspådrag för huvud- och hjälpraketer, återkoppling då larm har gått respektive åtgärdats.

- **Kontroller**: Möjlighet att starta nedräkning, justera gaspådraget för huvud- och hjälpraketer, lossa rymdraketen från rampen, avfyra rymdfärjan. Möjlighet att åtgärda larmen i form av att stänga dörren, lossa ramphållaren, starta rullningen av rymdfärjan, släppa hjälpraketerna, stänga av huvudraketerna, starta rymdraketerna. Möjlighet att välja om larm skall visas eller ej.

Operatörspanelen skall ha en skärmstorlek på minst 800x600 pixlar. Panelen skall utformas på ett sådant sätt att en användare inte kan utföra en operation som leder till att ett förvillkor i modellen bryts. Flera paneler skall när som helst under färd kunna startas och då genast återspegla modellens aktuella tillstånd.

C.1.2 Extrauppgift 1 (1 poäng)

Använd passande händelser i modellen för att uppdatera en vy av en extern simulerad stillbildskamera. Kameravyn ska integreras i det befintliga grafiska gränssnittet. Rymdfärjan kan befinna sig i sex olika faser som med fördel kan användas vid uppdateringen av kameravyn. Passande bilder för respektive fas kan hämtas från till exempel http://grin.hq.nasa.gov/.

De faser modellen genomgår under en flygning är följande (siffrorna inom parantes är de heltal som returneras av modellen för respektive fas):

- IDLE (0)
- LAUNCHING (1)
- LAUNCHED (2)
- ROLLING (3)
- BOOSTERSRELEASE (4)
- SPACE (5)

C.1.3 Extrauppgift 2 (1 poäng)

Designa ytterligare en operatörspanel, denna skall vara anpassad för att visas på en handdator. Ett tänkt scenario är att även personal på marken ska kunna följa och styra rymdfärjans färd. Panelens skärmstorlek skall vara högst 240x320 pixlar. Dokumentera uppgiften genom att lämna in en skärmdump på gränssnittet och en utförlig skriftlig redogörelse som innehåller motivering till varför du väljer att prioritera viss information framför annan. Förklara även eventuella tekniska detaljer eller flöden som inte framgår från gränssnittet. Från den skriftliga dokumentationen skall det även framgå med motivering till varför de använda grafiska gränssnittskomponenterna valts. Identifiera de svårigheter som dyker upp. Gränssnittet behöver inte kopplas till rymdfärjan, utan fokus ligger på din design. Tips vid designen: sätt dig in i rollen som användare och den miljö och situation tillämpning kommer användas i.

C.2 Arbetsgång

1. Kopiera erforderliga filer och registrera modellen i Windows systemregister genom att köra filen local.bat. Denna fil kan även modifieras för att installera filerna i en annan katalog, men katalogen får inte ligga på en nätverksenhet och den måste vara skrivbar.
2. Gör en design av gränssnittet (på papper eller i VB:s utvecklingsmiljö).
3. Stäm av designen med labhandledaren.

4. För att använda komponenten i VB måste den läggas till i listan över referenser. Detta görs genom att välja *Project -> References* i utvecklingsverktygets meny. Därefter skall *SpaceShuttle* läggas till.
5. Utgå från källkoden som återfinns i bilaga E.
6. Koppla operatörspanelen till modellens händelser och funktioner.
7. Kontrollera att panelen fungerar genom att genomföra en komplett färd.
8. Kontrollera att panelen hämtar all information från modellen genom att använda två paneler samtidigt och kontrollera att händelser återspeglas i båda panelerna.
9. Använd modellen som en del i ett distribuerat system. Börja med att exekvera filen server.bat för att installera modellen som en DCOM-server. I samband med att server.bat installerar modellen skrivs även namnet på datorn (hostname) ut, exempelvis ”b03”. Anteckna detta, då det behövs vid installation av klienten.
10. Starta upp operatörspanelen som tidigare på datorn, som nu skall agera server. Logga därefter in på en annan dator, som skall vara klient. Observera att inloggningsnamnet på de båda datorerna måste vara detsamma, då en användare måste ha rättighet att använda ett objekt på en annan dator. Med de aktuella säkerhetsinställningarna för DCOM har endast skaparen av serverprocessen rättigheten att använda den från en annan dator.
11. Kör filen client.bat på klienten med namnet på serverdatorn som parameter. Exempelvis ”client b03” om namnet på servern är b03. Filen client.bat kopierar erforderliga dll-filer och registrerar inställningar för komponenten i systemregistret. Observera att client.bat endast skall köras på den datorn där servern inte är installerad. Kontrollera att de båda panelerna fungerar på samma sätt som i föregående punkt.
12. Provkör.

Om filen client.bat av misstag exekveras på datorn som agerar server måste filen server.bat exekveras på nytt på denna dator.

C.3 Information om modellen

Modellen består av en COM-komponent (som även kan användas i ett distribuerat system, DCOM). Den kan i stort sett användas som ett lokalt objekt med några skillnader:

- Ett objekt kan inte skapas med nyckelordet *New* utan *CreateObject* måste användas för att skapa en instans av en klass.
- När ett objekt inte längre skall användas måste det avinitieras genom att sättas lika med ”Nothing”.

Komponenten innehåller klasserna *Connector* och *ShuttleModel*. ShuttleModel utgör själva modellen, den kan dock inte instansieras direkt utan det är här Connectorklassen kommer in i bilden. Connectorklassens enda uppgift är att vid anrop till funktionen *GetShuttleModel* returnera en referens till ett ShuttleModel-objekt, syftet med detta är att ge samtliga klienter en referens till samma modell.

COM-servern (SpaceShuttle.exe) startas automatiskt vid klientanrop och fortsätter exekvera så länge det finns klienter som använder sig av modellen för att sedan avsluta. Då ett misslyckat uppkopplingsförsök gjorts eller avinitieringar inte skett korrekt kan det hända att modellen fortsätter exekvera fast inga klienter längre använder den, i så fall måste den stoppas manuellt genom att döda processen via Windows aktivitetshanterare.

Om ett fel uppstår i modellen, kommer händelsen *ShuttleHasCrashed* att signaleras och samtliga larm kommer att ges.

C.3.1 Beskrivning av modellens interface

En fullständig beskrivning av modellens ingående och utgående interface ges i bilaga D.

C.4 Provkörning och dokumentation

C.4.1 Provkörning

Provkörning godkänns under följande förutsättningar:

- En stabil flygsekvens kan utföras från start till mål.
- Panelen återspeglar korrekt modellens tillstånd.
- Användaren kan inte bryta mot modellens förvillkor (genom klåfingriga knapptryckningar etcetera).
- Ytterligare paneler kan startas upp under pågående färd och ge en korrekt bild av modellen.
- Flera paneler kan startas upp på två olika datorer.
- COM-komponenten har avslutat sin exekvering då alla dess klienter stängts ned.

C.4.2 Dokumentation

- Alla antaganden skall dokumenteras.
- Koden skall vara väl dokumenterad med för- och eftervillkor.
- Någon skriftlig dokumentation utöver vad som beskrivits i ovanstående punkter erfordras ej.
- En skärmdump på användargränssnittet skall bifogas.
- I övrigt gäller ordinarie laborationsregler, se http://www.cs.kau.se/cs/students/lab_info/labrules_summary_swe.php och http://www.cs.kau.se/cs/students/lab_info/labmall.php.

D Beskrivning av modellens interface

D.1 Ingående interface

D.1.1 Funktioner för styrning av modellen

Funktion	Beskrivning:
Public Sub CloseDoor()	Stänger rymdfärjans dörr. Förvillkor: Får endast anropas då dörren är öppen. Eftervillkor: Dörren stängd.
Public Sub CountdownStart()	Påbörjar nedräkning. Förvillkor: Får endast anropas om nedräkning inte redan påbörjats. Eftervillkor: Nedräkningen har påbörjats.
Public Sub ReleaseHolder()	Lösgör färjan från avskjutningsrampen. Förvillkor: Färjan är fastgjord vid rampen, nedräkning till noll har skett. Eftervillkor: Rampen har lossats.
Public Sub LiftOff()	Ger kommando till rymdfärden att lyfta Förvillkor: Färjan har inte lyft, nedräkning till noll har skett. Eftervillkor: Färjan har lyft.
Public Sub Roll()	Påbörja rullning. Förvillkor: Färjan har nått sin rullningsfas men inte redan börjat rulla. Eftervillkor: Rullning påbörjad.
Public Sub ReleaseBoosters()	Släpper hjälpraketerna. Förvillkor: Färjan har nått rymdfasen och har ännu inte släppt hjälpraketerna. Eftervillkor: Hjälpraketerna har lossats.
Public Sub TurnOffRockets()	Stänger av huvudraketerna.

	Förvillkor: Färjan har nått rymdfasen, huvudraketerna är påslagna. Eftervillkor: Huvudraketerna har slagits av.
Public Sub TurnOnSpaceRockets()	Slår på rymdraketerna. Förvillkor: Färjan har nått rymdfasen, rymdraketerna är avslagna. Eftervillkor: Rymdraketerna har startats.
Public Sub RocketForce(Force As Integer)	Ändrar gaspådraget till huvudraketerna. Förvillkor: 0 <= angivet gaspådrag <= 100. Eftervillkor: Gaspådraget ändrat till det angivna värdet.
Public Sub BoosterForce(Force As Integer)	Ändrar gaspådraget till hjälpraketerna. Förvillkor: 0 <= angivet gaspådrag <= 100. Eftervillkor: Gaspådraget ändrat till det angivna värdet.
Public Sub ResetShuttle()	Återställer modellen till ursprungsläget. Förvillkor: - Eftervillkor: Modellen har återställts.

D.1.2 Egenskaper för avläsning av modellens tillstånd

Public Property Get CloseDoorState() As Boolean	Returnerar dörrens tillstånd. Förvillkor: - Eftervillkor: True returnerat om dörren stängts, annars false.
Public Property Get CountdownStartState() As Boolean	Returnerar nedräkningstillståndet. Förvillkor: - Eftervillkor: True returnerat om nedräkning påbörjats (inklusive genomförts och avslutats), annars false.
Public Property Get CountdownValueState() As Integer	Returnerar nedräkningsurets aktuella värde. Förvillkor: - Eftervillkor: Värdet returnerat.
Public Property Get ReleaseHolderState() As Boolean	Returnerar ramptillståndet. Förvillkor: -

	Eftervillkor: True returnerat om färjan har lösgjorts från avskjutningsrampen, annars false.
Public Property Get LiftOffState() As Boolean	Returnerar status för om rymdfärjan har lyft. Förvillkor: - Eftervillkor: True returnerat om rymdfärjan har lyft, annars false.
Public Property Get RollState() As Boolean	Returnerar rullningsstatus. Förvillkor: - Eftervillkor: True returnerat om rullning har påbörjats, annars false.
Public Property Get ReleaseBoostersState() As Boolean	Returnerar status för hjälpraketer. Förvillkor: - Eftervillkor: True returnerat om hjälpraketerna har släppts, annars false.
Public Property Get TurnOffRocketsState() As Boolean	Returnerar status för huvudraketer. Förvillkor: - Eftervillkor: True returnerat om huvudraketerna har stängts av, annars false.
Public Property Get TurnOnSpaceRocketsState() As Boolean	Returnerar status för rymdraketer. Förvillkor: - Eftervillkor: True returnerat om rymdraketerna har slagits på, annars false.
Public Property Get RocketForceState() As Integer	Returnerar det aktuella gaspådraget för huvudraketerna. Förvillkor: - Eftervillkor: Det aktuella gaspådraget returnerat.
Public Property Get BoosterForceState() As Integer	Returnerar det aktuella gaspådraget för hjälpraketerna. Förvillkor: - Eftervillkor: Det aktuella gaspådraget returnerat.
Public Property Get TemperatureState() As Integer	Returnerar den aktuella rakettemperaturen. Förvillkor: - Eftervillkor: Den aktuella temperaturen returnerad.

Public Property Get PhaseState() As Integer	Returnerar den aktuella fasen. Förvillkor: - Eftervillkor: Aktuell fas returnerad.
Public Property Get DistanceEarthState() As Integer	Returnerar aktuellt avstånd till jorden. Förvillkor: - Eftervillkor: Det aktuella avståndet till jorden returnerat.
Public Property Get DistanceSpaceState() As Integer	Returnerar aktuellt avstånd till rymden. Förvillkor: - Eftervillkor: Det aktuella avståndet till rymden returnerat.
Public Property Get CloseDoorLarmState() As Boolean	Status för larmet som indikerar att dörren måste stängas. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get ReleaseHolderLarmState() As Boolean	Status för larmet som indikerar att färjan måste lösgöras från avskjutningsrampen. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get RollLarmState() As Boolean	Status för larmet som indikerar att rullning måste inledas. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get ReleaseBoostersLarmState() As Boolean	Status för larmet som indikerar att hjälpraketerna skall släppas. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get TurnOffRocketsLarmState() As	Status för larmet som indikerar att huvudraketerna skall slås av.

Boolean	Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get TurnOnSpaceRocketsLarmState() As Boolean	Status för larmet som indikerar att rymdraketerna skall slås på. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get TemperatureLarmState() As Boolean	Status för larmet som indikerar att rakettemperaturen är för hög. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.
Public Property Get ForceLarmState() As Boolean	Status för larmet som indikerar att gaspådraget är för lågt. Förvillkor: - Eftervillkor: True returnerat om larmet är aktuellt, annars false.

D.1.3 Connectorklassens ingående interface

Funktion	Beskrivning:
Public GetShuttleModel() As ShuttleModel	Ger en referens till rymdfärjan. Förvillkor: - Eftervillkor: Referens till modellen returnerad

D.2 Utgående interface

Funktion	Beskrivning:
Event CloseDoorLarm()	Larmar då dörren är öppen men inte borde vara det.
Event ReleaseHolderLarm()	Larmar då färjan är fastgjord vid rampen men inte borde vara det.
Event RollLarm()	Larmar då färjan borde inlett sin rullning men inte gjort det.
Event ReleaseBoostersLarm()	Larmar då hjälpraketerna borde lossats men inte gjort

	det.
Event TurnOffRocketsLarm()	Larmar då huvudraketerna borde slagits av men inte gjort det.
Event TurnOnSpaceRocketsLarm()	Larmar då rymdraketerna borde startats men inte gjort det.
Event TemperatureLarm()	Larmar då rakettemperaturen är för hög.
Event ForceLarm()	Larmar då det sammanlagda gaspådraget (huvud- plus hjälpraketer) är för lågt.
Event TemperatureStable()	Anropas då rakettemperaturen stabiliserat sig.
Event ForceStable()	Anropas då det sammanlagda gaspådraget blivit tillräckligt.
Event CloseDoorPerformed()	Anropas då dörren stängts.
Event CountdownStartPerformed()	Anropas då nedräkning påbörjas.
Event ReleaseHolderPerformed()	Anropas då färjan lösgjorts från rampen.
Event LiftOffPerformed()	Anropas då rymdfärjan lyfter.
Event RollPerformed()	Anropas då färjan påbörjat sin rullning.
Event ReleaseBoostersPerformed()	Anropas då hjälpraketerna lossats.
Event TurnOffRocketsPerformed()	Anropas då huvudraketerna slagits av.
Event TurnOnSpaceRocketsPerformed()	Anropas då rymdraketerna slagits på.
Event CountdownValueChanged(Seconds As Integer)	Anropas varje gång nedräkningsklockan ändrar värde.
Event RocketForceChanged(Force As Integer)	Anropas då gaspådraget till huvudraketerna ändrats.
Event BoosterForceChanged(Force As Integer)	Anropas då gaspådraget till hjälpraketerna ändrats.
Event TemperatureChanged(Temperatur	Anropas då rakettemperaturen ändrats.

e As Integer)	
Event PhaseChanged(ShuttlePhase As Integer)	Anropas varje gång rymdfärjan går in i en ny fas.
Event DistanceEarthChanged(Distance As Integer)	Anropas då avståndet till jorden ändrats.
Event DistanceSpaceChanged(Distance As Integer)	Anropas då avståndet till rymden ändrats.
Event ShuttleHasCrashed()	Anropas då modellen har hamnat i ett feltillstånd.
Event ResetShuttlePerformed()	Anropas då modellen återställts i startläget

E Exempelkod för användning av modellen

```
Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''Konstanter m.m.''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'Modellens olika faser
Private Const PHASE_IDLE = 0
Private Const PHASE_LAUNCHING = 1
Private Const PHASE_LAUNCHED = 2
Private Const PHASE_ROLLING = 3
Private Const PHASE_BOOSTERSRELEASE = 4
Private Const PHASE_SPACE = 5

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''Privata variabler''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'Modellen
Private WithEvents MyShuttle As SpaceShuttle.ShuttleModel

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''Privata funktioner/subrutiner''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'Initieringar då programmet startas
Private Sub Form_Load()
    'Deklarera Connector-objekt
    Dim MyConnector As SpaceShuttle.Connector
    'Hämta Connector-objekt
    Set MyConnector = CreateObject("SpaceShuttle.Connector")
```

```
    'Använd Connector-objektet för att få rymdfärjemodellen
    Set MyShuttle = MyConnector.GetShuttleModel
    'Kontrollera om något gick fel
    If MyShuttle Is Nothing Then
        MsgBox ("Kontakt med rymdfärjan misslyckades.")
        'Avinitiera Connector-objektet
        MyConnector = Nothing
        End
    End If
    'Avinitiera Connector-objektet
    Set MyConnector = Nothing
End Sub

'Avinitieringar då programmet avslutas
Private Sub Form_Unload(Cancel As Integer)
    'Avinitiera rymdfärjan
    Set MyShuttle = Nothing
End Sub

'Inträffar då rymdfärjans dörr stängs.
Private Sub MyShuttle_CloseDoorPerformed()
    'Uppdatera GUI för att återspegla modellen
End Sub

'Stäng rymdfärjans dörr
Private Sub cmdCloseDoor_Click()
    MyShuttle.CloseDoor
End Sub
```