Avdelning för datavetenskap

Anders Olsson

Marcus Karlsson

Design och implementation av en Lagersystemssimulator

Design and Implementation of a Warehouse Simulator

Examensarbete 15 hp

Dataingenjörsprogrammet

Datum/Termin: 09-06-03

Handledare: Thijs Holleboom

Examinator: Martin Blom

Ev. löpnummer: C2009:01

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

Information@kau.se www.kau.se

Design och implementation av en Lagersystemssimulator

Anders Olsson & Marcus Karlsson

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 mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Anders Olsson

Marcus Karlsson

Godkänd, 20090603

Handledare: Thijs Holleboom

Examinator: Martin Blom

Sammanfattning

Prevas AB har utvecklat ett lagerhanteringssystem som kallas Snitcher Warehouse. Programmet körs på handdatorer och används av lagerarbetare för att underlätta det dagliga arbetet. Applikationen innehåller bland annat funktioner för att hantera gods som inkommer till lagret, det hanterar också flera olika typer av plocklistor samt funktioner för saldo och inventering. Lagerhanteringssystemet körs mot ett affärssystem som heter Movex/M3. Affärssystemet är stort och komplext samt innehåller många fler funktioner än vad Snitcher Warehouse har användning för. För att kunna demonstrera och felsöka applikationen behövs en uppkoppling mot affärssystemet samt tillgänglig testdata i det. Önskemålet från Prevas var att utveckla en simulator till deras lagerhanteringssystem, denna simulator har till uppgift att ta bort beroendet av ett stort affärssystem i bakgrunden vilket kommer leda till att Prevas produkt blir mycket lättare att demonstrera, utveckla och felsöka.

Syftet med examensarbetet var att utveckla en simulator i enlighet med Prevas kravspecifikation. För att skriva simulatorn använde vi programmeringsspråket Visual Basic, (VB), för att tillgodose Prevas önskemål om ett enhetligt språk genom hela systemet. Simulatorn är byggd för att vara dynamisk, det vill säga att den ska kunna anpassas för de olika behov och möjligheter som finns i lagerhanteringssystemet. Målet för simulatorn är att den ska bli enkel att använda men att den samtidigt ska fylla de flesta funktioner som en koppling mot ett riktigt affärssystem erbjuder.

Abstract

Prevas AB has developed a warehouse application that is called Snitcher Warehouse. The program is run on PPCs, Pocket PC, and it's used by stockroom workers to make the everyday work easier. The application has functions for handling goods that arrive at the warehouse. It also handles several different types of picking-lists and functions for balance and inventory. During execution, the warehouse system connects to a business system called Movex/M3. The business system is large and very complex, and it has a lot more functions than Snitcher Warehouse has use for. To do even the simplest form of demonstrating and troubleshooting with the application, you need a connection to the business system and specific test data in it. Prevas has wanted that a simulator be developed for the warehouse system. That is to remove the dependency of having a large business system running in the background. This will make the warehouse software a lot easier to demonstrate, develop and troubleshoot.

The degree project was to develop a simulator in accordance with the specifications provided by Prevas. Visual Basic was used as the programming language to implement the simulator. This was a desire from Prevas' side and it was the natural choice since the rest of the program also was written in VB. The construction of the simulator is meant to be dynamic. This means that it’s supposed to be able to adapt to the different needs and possibilities of the warehouse application. The goal for the simulator is to make it simple to use and at the same time make sure that it can support most of the functions that a connection to a real enterprise resource can.

Innehållsförteckning

Figurförteckning

1 Introduktion

Examensarbetet för vår del började med ett möte hos Prevas i Karlstad. Där berättade de om det lagerhanteringssystem de utvecklat, Snitcher Warehouse, hur det fungerade och vilka komponenter systemet bestod av. Efter genomgången av systemet gick de in på problemet vid demonstration av systemet, då det nuvarande systemet kräver uppkoppling mot ett affärssystem.

Uppdraget de önskade få hjälp med bestod i att implementera en simulator för deras Snitcher Warehouse system. Det är ett lagerhanteringssystem som har till uppgift att tillhandahålla de tjänster och operationer som underlättar och effektiviserar lagerarbete. Snitcher Warehouse systemet använder sig av handdatorer som kommunicerar trådlöst via Wireless Local Area Network, (WLAN), till en lokal server som upprätthåller en webbservice. Webbservicen tar emot anropen från handdatorerna och anropar i sin tur ett affärssystem som utför all databehandling.

Examensarbetet gick ut på att bygga en komponent som kan användas i Snitcher Warehouse webbservice. Komponenten ska sedan fungera som ett litet affärssystem, där den ska tillhandahålla affärssystemslogiken för de viktigaste operationerna och funktionerna i systemet. Genom att komponenten ska kunna simulera den viktigaste affärssystemlogiken så behövs ingen uppkoppling mot internet. Det skulle underlätta för Prevas vid demonstration av produkten samt vidareutveckling och felsökning.

2 Bakgrund

Kapitlet som följer innehåller en beskrivning av bakgrunden till examensarbetet, vad lagerhantering omfattar och en förklaring av vad en simulator är i det allmänna fallet. Det följs upp av historik och information gällande företaget som var vår uppdragsgivare under examensarbetet. Efter det beskrivs hur vi prioriterade arbetsuppgifterna och planerade upplägget på examensarbetet.

2.1 Examensuppgiften

Prevas har som vi tidigare nämnt utvecklat lagerhanteringssystemet Snitcher Warehouse. Detta system körs mot ett affärssystem som heter Movex/M3. Affärssystemet är stort och det innehåller ett stort antal funktioner som Snitcher Warehouse inte använder sig av. Det gör att även en enkel demonstration av Snitcher Warehouse blir jobbig att genomföra, då den kräver uppkoppling mot en riktig Movex/M3 server samt specifik testdata för demonstration. Examensarbetet gick ut på att utveckla en simulator som efterliknar de funktioner som Snitcher Warehouse systemet använder sig av i Movex/M3. Simulatorn var tänkt att underlätta vid en demonstration av systemet genom att man inte behöver ha en uppkoppling mot det riktiga affärssystemet. Dessutom blir det smidigare då man kan ha flera olika typer av testdata redan inlagda i simulatorn från början. Simulatorn var också tänkt att underlätta för Prevas vid felsökning och utveckling av Snitcher Warehouse produkten. Felsökning och utveckling av produkten skulle underlättas genom att man med simulatorn kan köra hela miljön lokalt. Det gör inte bara att körningarna går snabbare utan också att man kan följa anrop och parametrar i simulatorn vilket gör granskning av produkten enklare.

2.2 Lagerhantering

När man pratar om lagerhantering så handlar det om det arbete som krävs för att lagret ska hållas i god ordning. Det innebär i stort att alla de artiklar som behövs finns i lager, att inventeringslistorna är korrekta och att arbetet på lagret sker på ett effektivt sätt.

Mer specifikt handlar det om att skriva in, kontrollera och registrera inkommande varor. Placera och registrera varor på ny plats om de inte redan finns på plats i lagret eller att öka mängden på den plats där de redan finns. Dessutom ingår arbetet att registrera förflyttningar av varor mellan platser inom samma lager eller till ett annat lagerhus. Ett lager behöver inte bara bestå av ett lagerhus utan det kan bestå av flera lagerhus, som inte behöver ligga bredvid varandra. Det gör att det system som hanterar registreringen av varor och dess mängd måste vara centralt för hela lagret. Lagerhantering omfattar också inventering av varor, där man kan inventera hela lagret på en gång eller dela upp inventeringen efter lagerhus eller sektioner. Lagerhantering innefattar också att färdigställa order för utskick av varor.

För att förhindra att problem uppstår på grund av att flera lagerarbetare jobbar mot samma data så behövs någon typ av lås i lagerhanteringssystem. Låset ska genom att ge eller neka tillgång i vissa bestämda situationer se till att informationen som representerar lagret inte blir korrupt och inkonsistent. I Prevas programvara utförs detta endast i funktionerna som hanterar plocklistor. Med plocklistor menas de listor med artiklar som plockas ihop från olika sektioner på lagret för t.ex. tillverkning, det som då används är exklusiva lås. Det vill säga det är bara en handdator som kan arbeta med en plocklista åt gången.

Exklusiva lås används då ändringar ska göras i databasen, det vill säga när det handlar om UPDATE, DELETE och INSERT. Förutom exklusiva lås finns det dessutom något som heter delade lås. Dessa används när någon läser från databasen, det vill säga då det utförs SELECT kommandon. Syftet med delade lås är att ingen ska kunna ändra innehållet efter det att någon läst och inte släppt låset. Eftersom det skulle kunna skapa problem om personen som läste innan ändringen också ändrar samma data. Man vet då inte vilket av ändringarna som är den korrekta versionen.

Programvaran Prevas använder för att sköta lagerhantering har ett enkelt och användarvänligt gränssnitt. Med den begränsade kunskapen vi har inom lagerhantering så tycks programmet ha god funktionalitet för att på ett effektivt sätt hjälpa till med lagerhanteringen.

2.3 Simulator

En simulator skapas för att efterlikna delar av verkligheten [1][2]. Tanken bakom en simulator är att den ska återskapa de delar av original komponenten som fyller ett syfte för användaren. Eftersom man inte tar med allt, utan bara de delar man är intresserad av, så blir simulatorn mindre komplex och enklare att använda. En annan stor fördel med simulatorer är att de oftast är billigare än en riktig miljö. Tack vare att man gör en simulering av miljön så behöver man inte vara orolig för att krascher eller haverier ska förstöra något värdefullt. De flesta simulatorer används för utveckling och felsökning av en eller flera produkter och därför brukar de ha egenskapen att de sparar information om vad som gick fel eller kraschade i en loggfil eller liknande. Detta underlättar vid felsökning då man kan följa händelseförloppet innan felet uppstod. Ett bra exempel på vad en simulator är och syftet med dem är flygsimulatorer. De ger en verklighetstrogen flygupplevelse trots att simulatorns konstruktion som helhet inte liknar ett flygplan. Man har fokuserat på det som är viktigast för piloten, det vill säga cockpiten. En flygsimulator behöver inte bestå utav mer än en joystick och en skärm. Men funktionaliteten finns fortfarande med, det vill säga hur ett plan uppför och beter sig i olika situationer i luften. Nedan följer exempel på syften som simulatorer används för.

- Optimering av prestanda
- För att förbättra säkerheten
- Testning
- Träning och utbildning

När man arbetar för att förbättra prestandan hos ett system kan det vara jobbigt att arbeta med hela systemet på en gång. Speciellt om man bara vill fokusera på en mindre del av systemet. För att lättare kunna prova olika inställningar utan att behöva ta hänsyn till hela systemet så kan man då bygga en simulator. Den kan visa vilken inställning som ger den bästa prestandan för den specifika uppgiften. Med en simulator blir det smidigare och snabbare att testa en liten del eftersom simulatorn är byggd med en förbestämd uppgift som den ska lösa. I ett större sammanhang så kan det vara bättre att dela in systemet i flera simulatorer. Där man alltså testar systemet en del i taget istället för att använda hela det skarpa systemet på en gång. Det blir också säkrare om man använder simulatorer eftersom med dem kan man undersöka systemens gränser på ett säkert sätt. Man har ju inte utsatt det skarpa systemet för påfrestningen så ingen skada är skedd. Det blir särskilt tydligt om man försöker förbättra

säkerheten, eftersom man då vill se var gränserna går och det blir mindre skadligt om man använder simulatorer. De byggs just för det syftet, för att se vilka begränsningar som finns och var systemets svaga punkter ligger. Hade man istället använt hela systemet så hade man behövt reparera det som gått sönder innan man kan fortsätta med de andra testerna.

När nya anställda kommer in i bilden som behöver lära sig systemet så kan det bli ineffektivt att behöva ha hela systemet igång för att de ska kunna utbildas på sin del. Kan man då istället simulera den funktion som de ska tränas på blir det betydligt effektivare. Dessutom kan möjligheten finnas att köra flera simulatorer för olika saker på samma gång, om de byggs för att vara oberoende av varandra vill säga. Uppnår man detta så kan utbildning och testning göras mycket effektivt, eftersom man kan utföra så många fler funktioner samtidigt än om allt skulle gå mot ett stort system. Simulatorer brukar delas in i två olika grupper, *Fysisk Simulation* och *Interaktiv Simulation*.

Med fysisk simulation menas att det fysiskt objekt man vill simulera ersätts med en materiellsimulator som ofta är rätt lik originalet, se Figur 2-1. Det som är viktigt vid fysisk simulation är konstruktionen, vilken görs för att efterlikna originalet så långt det är möjligt.

Figur 2-1 Hästridningssimulator

För interaktiva simulatorer, se Figur 2-2, så är likheten rent konstruktionsmässigt inte så stor, man har istället fokuserat på det som är viktigt för användaren. Det som spelar roll vid interaktiv simulation är hur simulatorn uppför sig i olika situationer. Det är tänkt att simulatorn ska fungera som originalet, det vill säga den ska ge samma gensvar på

användarens handlingar som i en verklig situation. Det är viktigt att uppförandet blir så likt som möjligt, för att simulatorn ska vara ett effektivt verktyg vid testning och utbildning.

Figur 2-2 Flygsimulator

Vår simulator hamnar i den interaktiva gruppen, eftersom meningen med den är att den ska uppföra sig som att det fanns ett affärssystem i bakgrunden. Så beroende på vilken information som användaren matar in så kommer den ge samma svar som ett skarpt system.

2.4 Prevas AB

Vår uppdragsgivare under examensarbete var Prevas AB [11]. Det är ett IT-företag vars grundstomme är produktutveckling och utveckling av industrisystem. Prevas grundades 1985 i Västerås, där också företagets huvudkontor ligger. Prevas börsnoterades i Sverige 1998.

Prevas vision och affärsidé lyder som följande,

Figur 2-3 Prevas vision

Företagets har i dagsläget, (våren 2009), ca 560 anställda och har kontor bland annat i Stockholm, Göteborg, Malmö, Västerås och Karlstad. Karlstad kontoret där vi utförde vårt examensarbete ligger i dagsläget runt femte plats sett till antalet anställda. I Karlstad ligger Prevas kontor i Karolinen.

Prevas två huvudområden är;

- **Produktutveckling**
  Ledande i Norden inom inbyggda system
- **Industrisystem**
  Ledande i Norden inom industriell IT

2.5 Uppsatsens upplägg

När examensarbetet påbörjades informerades vi om att en stomme på en simulator påbörjats sedan tidigare. Snitcher Warehouse produkten har ändrats flera gånger efter det att stommen på simulatorn byggdes. Därför var den varken fungerande eller korrekt, men vi bestämde oss

för att utgå från den i alla fall. När examensarbetet startade gjordes en planering över hur vi tänkt lösa uppgiften. Upplägget för examensarbetet såg ut som följande:

- Studera och förstå strukturen i Snitcher Warehouse.
- Studera stommen i den påbörjade simulatorn.
- Få igång Snitcher Warehouse med simulatorn.
- Börja med de funktioner som skriver till databasen.
- Ta sedan de funktioner som bara läser från databasen.

Den första planeringen var inte längre än så då det säkert skulle komma upp nya saker man inte tänkt på från början. Men det gjorde att vi hade något att gå efter.

2.6 Sammanfattning

Prevas är ett it-företag som startade i Västerås, vi har under examensarbetet arbetat för deras Karlstadkontor. De har gjort en produkt som heter Snitcher Warehouse, som är ett handdatorbaserat lagerhanteringssystem. Systemet använder sig av en lokal server, som upprätthåller en webbservice. Handdatorerna kommunicerar via WLAN med webbservicen. Från webbservicen skickas sedan informationen vidare till affärssystemet Movex/M3 där all databehandlingen genomförs. Vid utveckling eller demonstration blir det jobbigt då det måste finnas uppkoppling mot affärssystemet samt fungerande testdata som man kan använda sig av. Vårt examensarbete går ut på är att implementera en komponent som simulerar det Application Programming Interface, (API), som Snitcher Warehouse använder sig av i affärssystemet.

3 Uppgiften

Följande kapitel kommer att handla om de specifikationer och riktlinjer som vi erhöll inför examensarbetet. Under detta kapitel kommer vi också att ta upp önskemålet från Prevas sida om att implementera en Windows Applikation för insättning av testdata till simulatorns databas.

3.1 Uppdraget

Designa och implementera en komponent som simulerar databehandlingen för det API Prevas produkt Snitcher Warehouse använder sig av i affärssystemet Movex/M3. Den nya komponenten behövs för att kunna köra det handdatorbaserade lagerhanteringssystemet Snitcher Warehouse utan koppling mot affärssystemet. Simulatorn är tänkt att underlätta för Prevas vid demonstration, utveckling och felsökning av produkten. Till examensarbetet tillkommer dokumentation i form av en användarmanual. Denna skall innehålla information om hur miljön sätts upp, funktionen av simulatorn samt om en konfigureringsfil behöver användas och hur inställningarna i konfigureringsfilen ändras. Alla tabeller i Structured Query Language, (SQL), databasen ska kunna ändras med SQL-manager eller liknande programvara.

3.2 Specifikation

Denna del innehåller en mer detaljerad specifikation som fastställer den minsta funktionaliteten som krävs av simulatorn. Vad simulatorn ska ge för svar på respektive anrop från handdatorn och vilken information som behöver lagras i databasen.

Godsmottagning: Kunna ladda ner listor med artikelbenämning, artikelnummer, antal mm från tabell med hjälp av sökning på ordernummer.
All återrapportering av rader till Movex kan svara OK hårdkodat

Lagerläggning: All rapportering till Movex kan svara OK hårdkodat.

Plocktillverkning: Kunna ladda ner lista av plocklisthuvud med plocklistnummer mm från tabell med hjälp av sökning på plocklisttyp, (tillverkning).
Kunna ladda ner plocklistrader med lagerplats, rad, artikelnummer mm från tabell med hjälp av sökning på plocklistnummer
All återapportering av rader till Movex kan svar OK hårdkodat.

Lagerinventering: All rapportering till Movex kan svara OK hårdkodat.

Plock slutprodukt/sp: Kunna ladda ner lista av plocklisthuvud med plocklistnummer mm från tabell med hjälp av sökning på plocklisttyp (slutprodukt),
Kunna ladda ner plocklistrader med lagerplats, rad, artikelnummer mm från tabell med hjälp av sökning på plocklistnummer
All återapportering av rader till Movex kan svar OK hårdkodat.

Gallring kassation: Kunna ladda ner lista av plocklisthuvud med plocklistnummer mm från tabell med hjälp av sökning på plocklisttyp (kassation),
Kunna ladda ner plocklistrader med lagerplats, rad, artikelnummer mm från tabell med hjälp av sökning på plocklistnummer
All återapportering av rader till Movex kan svar OK hårdkodat.

Lagersaldo: Skall kunna returnera en kvantitet utsökt med en lagerplats identitet från en tabell. Om man söker på en lagerplats som inte finns upplagd så ska man returnera "NOK Lagerplatsen existerar inte"

Lagerplatsflytt: All rapportering till Movex kan svara OK hårdkodat

3.3 Dokumentation

I dokumentationen för simulatorn ska det beskrivas vad som behöver ändras och hur man ändrar det för att få igång en fungerande Snitcher Warehouse miljö med simulatorn. I

dokumentationen ska det också beskrivas vart simulatorns filer ska läggas samt om konfigureringsfil används. Exempel på ändringar som ska stå med är ändringar som är utförda i Snitcher Warehouse system32 filer. Om simulatorn använder sig av konfigureringsfil så ska dokumentationen berätta hur den fungerar samt hur man ändrar i konfigureringsfilen.

3.4 Riktlinjer

För att underlätta vid demonstration och göra informationen simulatorn använder sig av i databasen mer lätthanterlig, så ska alla tabeller i databasen gå att ändra med SQL manager eller liknande programvara. Det innebär att en person som är insatt i Snitcher Warehouse och som känner till de element och typer produkten använder sig av, ska kunna lägga till och ändra informationen i simulatorns databas.

3.5 Windows Applikation

Om det skulle blir tid över vid slutet av examensarbetet så önskades en Windows applikation för insättning av testdata till databasen. Den funktionalitet Prevas önskade i Windows applikationen var bland annat att ha en översikt över hur tabellerna i databasen hänger samman samt innehållet i tabellerna. Applikationen var tänkt att underlätta insättningen och redigeringen av data i simulators databas. Användaren kan då välja mellan att antingen sätta in specifik testdata som den själv matar in eller slumpa fram data för att fylla upp databasen till tester eller demonstration. Genom applikationen ska det också gå att ändra och redigera de attribut som är associerade med artikeln, det kan exempelvis vara om en artikel har någon specifik information associerad till sig.

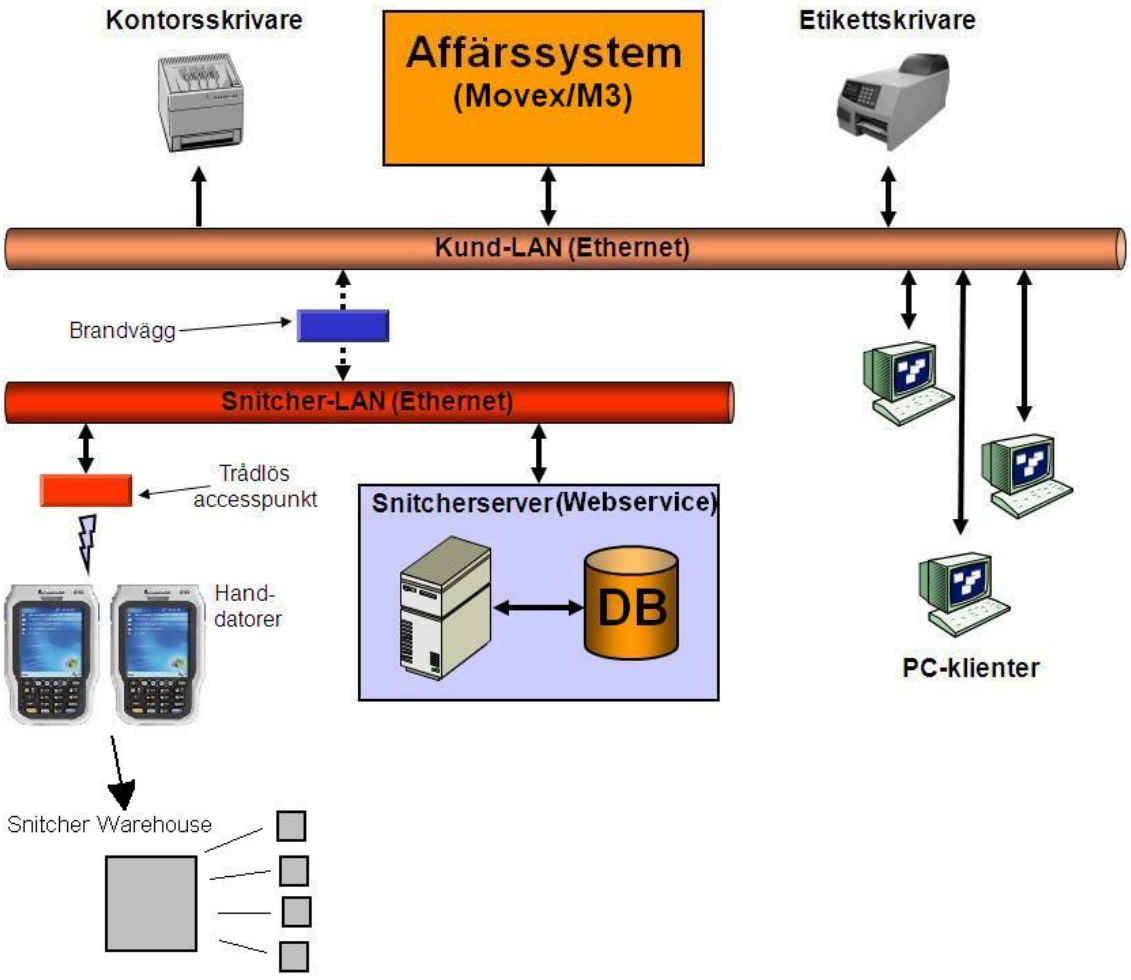
3.6 Sammanfattning

Kapitlet har beskrivit mer ingående vad examensarbetet består av, implementera en komponent för att simulera den affärssystemslogik Snitcher Warehouse behöver för att kunna köras utan uppkoppling mot affärssystemet Movex/M3. Det har berättats om den specifikation vi fick över hur simulatorkomponenten ska implementeras, det vill säga vilken funktion som minst önskas av funktionsanropen i simulator. I examensarbetet ska det ingå dokumentation,

där det ska beskrivas hur man ändrar ifrån den nuvarande Snitcher Warehouse miljön till att använda simulatorn. Att SQL-databasen simulatorn använder sig av ska kunna redigeras med hjälp av SQL-manager eller liknande programvara, så att användaren lätt kan ändra eller lägga till data i tabellerna. I kapitlet beskrevs också önskemålet om en Windows applikation för insättning och redigering av testdata i databasen, så att användaren enkelt kan antingen sätta in specifik testdata eller slumpa fram en vald mängd data.

4 Systemets delar

Här ges en kort beskrivning av delarna som ingår i Snitcher Warehouse miljön. I Figur 4-1 är alla delar i Prevas Snitcher Warehouse miljön med. Systemet är uppbyggt med handdatorer som kommunicerar via WLAN [12] med webbservicen på den lokal Snitcherservern. På det lokala intranätet kan det också finnas skrivarstationer och PC-klienter. Webbservicen på Snitcherservern kommunicerar sedan med affärssystemet Movex/M3 via internet.



Figur 4-1 Översikt över Snitcher Warehouse miljön

Snitcher Warehouse miljön består av en Snitcherserver kör en Internet Information Service, (IIS), server. IIS servern upprätthåller en webbservice. Handdatorerna som kör Snitcher

Warehouse kommunicerar med webbservicen via WLAN. För att handdatorerna ska kunna kommunicera med webbservicen krävs inloggning med användarnamn och lösenord. Komponenten MvxAPI.dll, som inkluderas i webbservicen sköter all kommunikation vidare mot affärssystemet Movex/M3 över internet. Snitcherservern har en SQL-databas där användaruppgifter och information som rör användningen av handdatorerna lagras. Användarhanteringen rör bland annat inloggning och tidskontroller, som gör att endast behöriga användare får tillgång till systemet. Tidsstämplar används för att se till att inga inaktiva användare finns i systemet. Tidsstämpeln uppdateras när någon funktion på handdatorn används. Med tidsstämpel och användarnummer låser man de operationer som ska vara atomära, det vill säga att de måste antingen göras helt och hållet eller inte alls. Låsningen är viktig i dessa operationer då det oftast finns flera handdatorer som arbetar parallellt med de olika varorna på de lager där Snitcher Warehouse används.

4.1 Warehouse

Snitcher Warehouse systemet bygger på att handdatorerna kommunicerar via WLAN med en lokal server som tillhandahåller en webbservice, webbservern som körs för att kunna upprätthålla webbservicen är IIS. Webbservicen sköter sedan all kommunikation vidare mot Movex/M3 med hjälp av komponenten MvxAPI.dll. Systemet gör ingen databehandling lokalt, utan all datahantering sker i affärssystemet. Lagerhanteringssystemet sparar information om de operationer och transaktioner som utförs. Syftet med systemet är att effektivisera materialflöden inom lager och lagernära verksamhet. En del av effektiviseringen sker genom att handdatorerna har streckkodsavläsare, vilket underlättar arbetet. Den största fördelen med lösningen är dock att man inte behöver skriva ner informationen på papper först, för att sedan mata in uppgifterna på en lokal arbetsstation. Man slipper mellansteget eftersom handdatorerna är lätta att ta med sig ut på lagret och de håller hela tiden kontakten med webbservicen eftersom kommunikationen sker via WLAN.

4.2 Movex/M3

Movex/M3 är ett affärssystem som utvecklades av företaget Movex i början av 80-talet. Företaget köptes senare upp av Intentia i mitten på 80-talet, för att sedan gå ihop med Lawson Software och bilda företaget Lawson. I dagsläget kallas Movex för M3, som står för Move

Make Maintain. Det är ett affärssystem där allt från ekonomi och lönehantering till lagerhantering och produktion ingår. Styrkan med Movex/M3 är att alla delar finns i ett och samma system. Den senaste versionen av affärssystemet är skriven i Java. Under examensarbetet fick vi möjligheten att träffa anställda från Lawson som har varit med från början och de berättade att idag använder cirka 3-4000 företag i Sverige Movex/M3. De berättade också att på senare år har antalet företag som använder Movex/M3 minskat i Sverige men ökat utomlands.

4.3 Webbservice

Webbservice möjliggör kommunikation mellan olika plattformar och programspråk, den löser detta genom att sammankoppla de olika språken och plattformarna på en gemensam grund, gränssnittet i en webbservice är eXtensible Markup Language, (XML). Det finns två olika varianter, ”Big Web Services” och RESTful Web Services [10].

Big Web Services följer standarden för SOAP. I system som använder sig av Big Web Services så brukar det finnas en beskrivning av de tillgängliga operationerna, denna är skriven i WSDL, Web Services Description Language. SOAP, WSDL och UDDI är de tre grundstenarna i Big Web Services.

RESTful Web Services, REpresentational State Transfer, använder sig av http och anropen sker till unika URLer som representerar objekt. Man använder sig av kommandona GET, POST, PUT och DELETE för att manipulera objekten.

4.4 Pocket PC

Pocket Personal Computer är en handdator som använder sig av Microsoft Windows Mobile som operativsystem [3]. Under utvecklingen av simulatorn har vi använt oss av en Intermec CN2, se Figur 4-2. Simulatorn och Warehouse systemet är dock inte bundna till något specifikt märke eller modell.

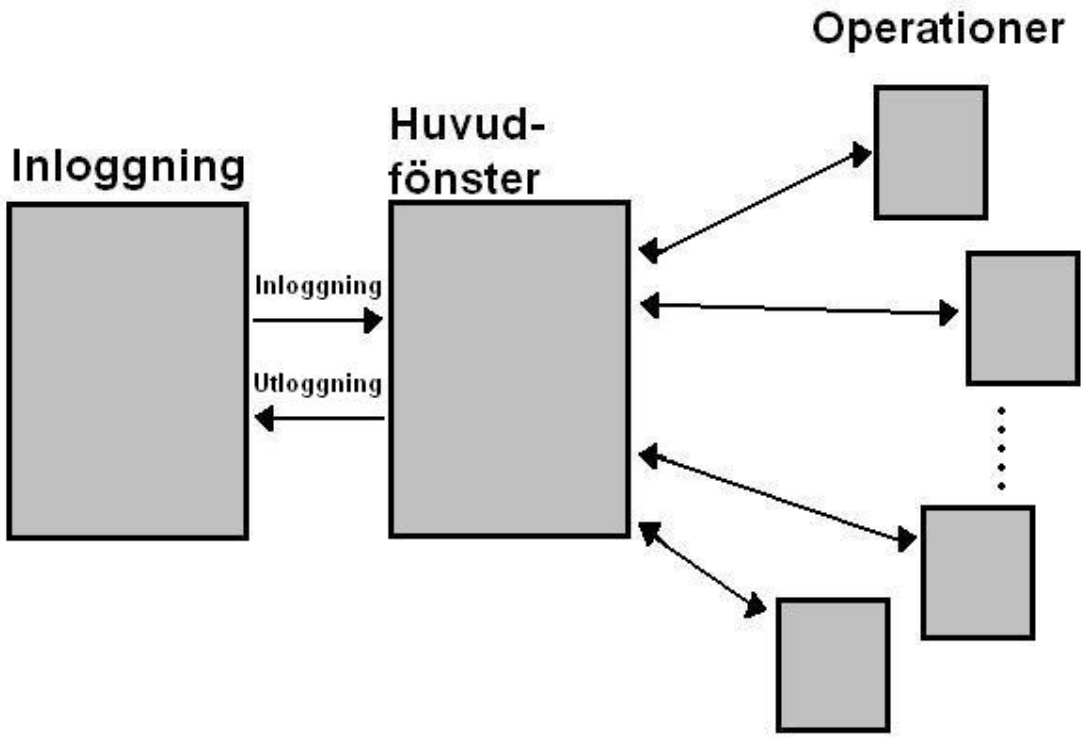
Figur 4-2 Intermec handdator

4.4.1 Tidssessioner i handdatorn

För att se till så att inga inaktiva handdatorer är inloggade i Snitcher Warehouse systemet, så använder sig programmet av tidssessioner. Tiden för hur lång en session ska vara ställs in i inställningarna för systemet. När handdatorerna loggar in mot webbservicen och registreras, startar första sessionen. Den återstående tiden en handdator har innan den loggas ut från systemet visas i nederkanten av programmet. Om någon operation utförs med handdatorn så att den kontaktar webb servicen, så uppdateras sessionstiden till det förbestämda värdet.

4.5 Användargränssnitt och fönster i Warehouse systemet

Användargränssnittet i handdatorerna är ett praktiskt och simpelt användargränssnitt där användaren klickar sig fram via knappar. För att underlätta och effektivisera så är menyerna och reglagen så stora som möjligt så att användaren inte behöver använda pekpinnen mer än nödvändigt. Knappar och reglage har alltid en tydlig beskrivning på vad de innebär.



Figur 4-3 Översikt över fönstren i handdatorerna

4.5.1 Inloggningsfönstret

Det första fönstret som kommer upp vid uppstart är inloggningsfönstret, se Figur 4-3 till vänster. Där får användaren skriva in användarnamn och lösenord samt välja vilken server han vill logga in på, om det nu finns flera alternativ.

4.5.2 Huvudfönstret

Efter inloggning visas huvudfönstret, se Figur 4-3 i mitten, här visas alla de operationer användarna kan utföra. I huvudfönstret visas tillsammans med alla operations alternativ, tiden användaren har tills han automatiskt blir utloggad för inaktivitet. Från huvudfönstret kan användaren klicka sig vidare till rätt fönster beroendet på ärendets karaktär.

4.5.3 Operationsfönstren

I operationsfönstren, se Figur 4-3 till höger, visas det som behövs för att användaren ska kunna utföra vald operation. I de fall där till exempel en lista behöver hämtas för att

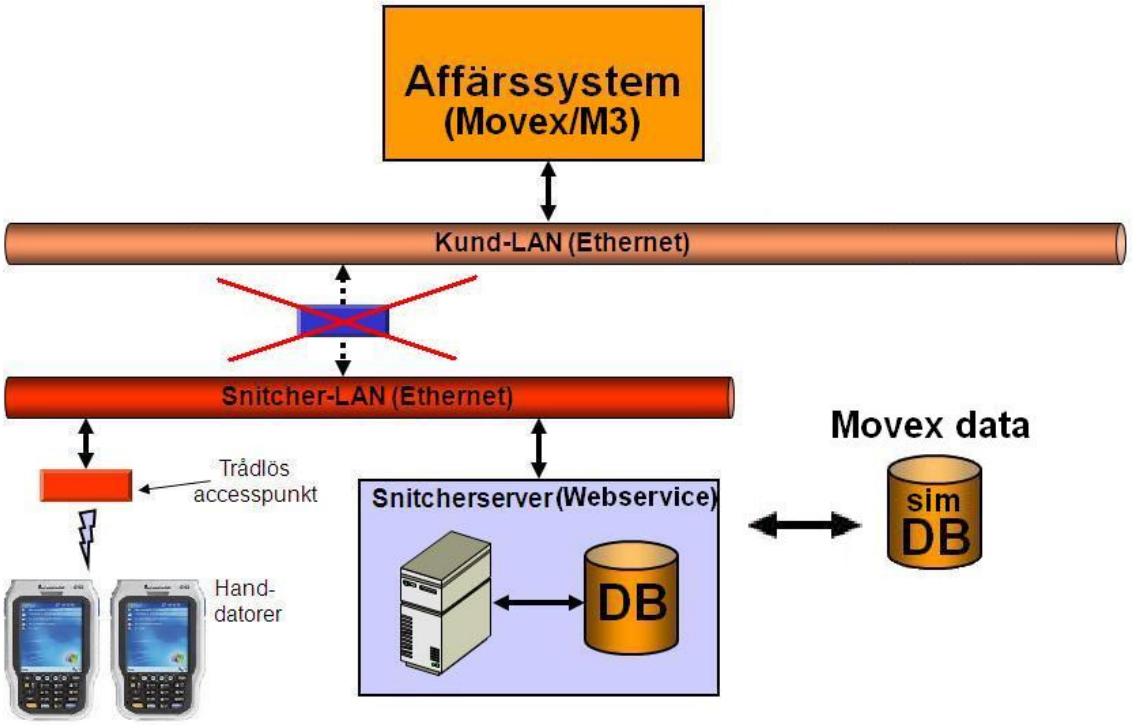
användaren skall kunna utföra operationer på listan finns det fält som användaren behöver fylla i för att hämta korrekt lista.

4.6 Prototyp

I denna del kommer vi att gå igenom ursprungsidén för upplägget på examensarbetet. Dessutom kommer programmeringsspråket som användes till implementationen av simulatorn att gås igenom, samt vår erfarenhet av det.

4.6.1 Grundidé

I Figur 4-4 kan man se hur vår första idé för hur vi tänkt oss att miljön med simulatorn kunde se ut. Efter det att kopplingen till Movex/M3 hade tagits bort så var den nya komponenten tänkt att kommunicera med en extern databas, *sim DB*, som skulle ligga på den lokala Snitcher servern. Den databasen var tänkt att bara innehålla den information vi behövde för att simulera affärssystemet. Genom att det nuvarande systemet bygger på stränghantering, stora strängar som man sedan har ett varierat antal substrängar i, så hade vi planer på att implementera objekt och använda oss av dem istället för stränghanteringen. Då vi vid en första överblick tyckte att anropen såg liknande ut och substrängarnas placering och storlek var lika. För information om vilket alternativ som sedan valdes till simulatorn se stycke 6.1.



Figur 4-4 Prototyp över miljön med simulatorn

4.6.2 Språk

Funderingar fanns att skriva simulator i C#. Efter önskemål från Prevas valdes Visual Basic, som är det språk Snitcher Warehouse systemet är skrivet i. Tack vare detta så kunde en stomme på en redan påbörjad simulator användas som grund för examensarbetet. Den påbörjade simulatorn var dock flera år gammal och hade aldrig blivit färdigställd. Stommen gav oss dock en grund att gå efter och underlättade vid arbetets start. Vid starten av examensarbetets var vår erfarenhet av VB minst sagt begränsad, men då vi tidigare programmerat i .Net miljön tog det inte lång tid innan vi var relativt bekanta med VB. Tid avsattes i början åt att gå igenom Snitcher Warehouse systemet och lära oss hur språket är uppbyggt och fungerar.

4.7 Sammanfattning

I det här kapitlet har vi försökt beskriva och förklara alla delar i systemet. För att läsaren ska ha nödvändig bakgrundsförståelse inför kommande kapitel. Målet med vårt examensarbete var att skapa en simulator för det API som Prevas Snitcher Warehouse system använder sig av

på affärssystemet Movex/M3. Det slutgiltiga resultatet av examensarbetet är tänkt att underlätta vid demonstration av Prevas produkt Snitcher Warehouse. Snitcher Warehouse körs på handdatorklienter, dessa kommunicerar via WLAN med webbservicen på den lokala Snitcherserver. Webbservicen skickar sedan informationen vidare till affärssystemet med hjälp av komponenten MvxAPI.dll. I simuleringsmiljön kommer lagringen av data ske i en lokal databas istället för i affärssystemet. Komponenten MvxAPI.dll kommer behöva ersättas med en komponent som istället för att kommunicera med affärssystemet simulerar databehandlingen affärssystemet utför.

5 Implementation

Uppgiften bestod i att designa och implementera en simulator till lagerhanteringssystemet Snitcher Warehouse. I den vanliga miljön kommunicerar Snitcher Warehouse programmet i handdatorerna med en webbservice, som upprätthålls av en lokal server. I webbservicen inkluderas komponenten MvxAPI.dll som sedan sköter kommunikationen mot affärssystemet. Informationen som skickas till affärssystemet kan till exempel vara listor av gods som lagret mottagit eller varor som placerats ut på respektive plats.

Genom att det nuvarande systemet bygger på att all kommunikation går genom komponenten MvxAPI.dll vidare till affärssystemet så fanns önskemål från Prevas att implementera en komponent som simulerar det API applikationen använder sig av i affärssystemet. Då behövs det inga ändringar i programmet, utan hela miljön förutom komponenten förblir intakt. Genom att man då lätt kan ersätta den riktiga komponenten med simulatorkomponenten så kan man smidigt övergå från skarpt läge till simulerat läge. Simulatorn ska innehålla en SQL-databas där testdata ska finnas lagrad. Vilket underlättar mycket vid demonstration såväl som felsökning och utveckling av produkten.

5.1 Databas

Webbservicens databas, som används för att lagra information om användarna samt deras aktivitet, användes också för att lagra tabellerna med testdata för simulatorn. Detta var ett önskemål från handledaren på Prevas då det verkade vara den enklaste och smidigaste lösningen. Nio tabeller fick läggas till i databasen för att på ett dugligt sätt kunna simulera funktionaliteten för det API Snitcher Warehouse använt sig av i affärssystemet Movex/M3. Nedan följer en kortare beskrivning av tabellernas innehåll och dess funktion. De tabellerna som finns beskrivna här är de som används i den slutgiltiga lösningen.

SIM_ARTICLE: I denna tabell lagras simulerade artiklar, informationen som lagras är artikelID och benämning på artikeln, där ID-numret är tabellens primärnyckel.

SIM_ORDERHEAD: I denna tabell lagras alla de order som finns i systemet. De två typerna som finns är *godsmottagningslistor* och *kundreturer.* För att skilja orderna åt i tabellen så används ordernumret som primärnyckel. Det finns också ett fält där man anger vilken typ av order det rör sig om. Dessa finns för att programmet ska veta vilka order som ska hämtas för vilken funktion. Dessutom lagras datumet då ordern skapades. En kolumn för kundnummer finns också med, men den sattes till *Allow Nulls* eftersom det bara är kundreturer som använder dessa.

SIM_ORDERROWS: I den här tabellen lagras information om raderna i orderna. Här är både ordernumret och orderraden primärnycklar. Den främmandenyckel som finns i tabellen är artikelID, som hänvisar till tabellen SIM_ARTICLE. Två fält används för att visa bekräftad och beställd kvantitet, det finns också ett specifikt fält för att visa om raden är klar.

SIM_LOCATION: Den här tabellen innehåller de tillgängliga lagerplatserna som finns. ID-numret som är primärnyckel sparas tillsammans med namnet på lagerplatsen.

SIM_STORAGE: Den här tabellen innehåller alla artiklarna som finns på lagret. Det vill säga det tillgängliga saldot av artikeln, vart de ligger, vilken sändning de kom med. Artikelnumret i denna tabell är främmandenyckel till *SIM_ARTICLE* tabellen, för att få kopplingen med namnet. För denna tabell finns ett separat StorageID, som ökas på efter behov med hjälp av *autoincrement*.

SIM_MESSAGE: Meddelandetabellen innehåller information om de specifika meddelande som tillhör funktionen Plock slutprodukt. Tabellen har messageID som primärnyckel och innehåller fält för meddelandetexten.

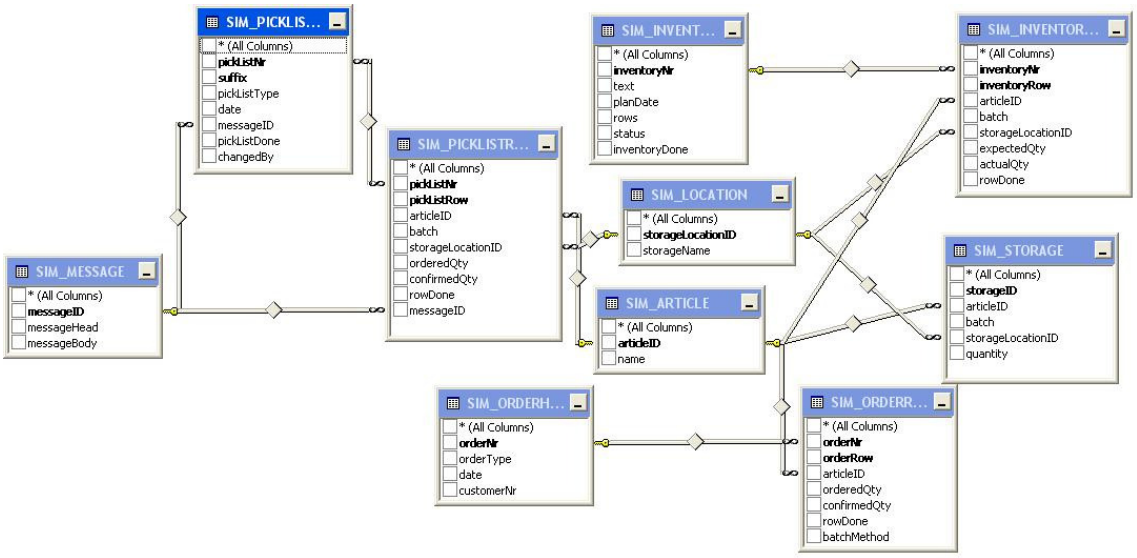
SIM_PICKLISTHEAD: I den här tabellen är pickListNr och suffix primärnycklar. Fältet pickListType beskriver vilken typ av plocklista det är. Dessutom finns ett fält för datum och det meddelande som hör ihop med **plocklist**an.

SIM_PICKLISTROWS: Primärnycklar i tabellen pickListRows är pickListNr och pickListRow. I tabellen finns det också fält för batchnummer, det fältet anger varifrån varan ska hämtas. Det finns också fält för beställd och plockad kvantitet samt meddelande. Ett specifikt fält används här också för att visa om raden är klar.

SIM_INVENTORYHEAD: Primärnyckel är inventoryNr. Det finns fält för specifik text associerad med inventeringen samt fält för datum då inventeringen är planerad. Fältet inventoryDone visar som namnet beskriver om inventeringen är gjord.

SIM_INVENTORYROW: Tabellens primärnyckel består av fälten inventoryNr och inventoryRow. Tabellen innehåller bland annat information om platser och artiklar, samt förväntad och aktuell kvantitet på de platserna.

För översikt över de tabeller simulatorn behöver i databasen se Figur 5-1.



Figur 5-1 Översikt över simulatorns databas

5.2 Databasens normalform

Normalformen för en databas anger en slags riktlinje för vilken design databasen är anpassad efter. Genom normalformen får man en överblick över hur god prestanda det är i databasen. Desto högre normalform en databas är i, desto mindre redundans finns det i den och desto snabbare är databasen. En databas bör bestå av så lite redundans som möjligt, då det försvårar uppdateringar och ändringar.

”The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized” [6].

De fyra vanligaste normalformerna är första, andra och tredje normalformen samt Boyce-Codds normalform. För att en databas ska vara i någon normalform måste den uppfylla kravet

för de tidigare normalformerna samt kravet för den aktuella normalformen. De fyra olika nivåerna av normalformer finns alla beskriva i databaslitteratur som till exempel [6].

Genom att vi använt oss av en existerande databas i vilken det finns flera tabeller i än de som lagts till för att simulera det API produkten använt sig av i affärssystemet, så vet vi inte vilken normalform hela databasen är i. Däremot är de tabeller som skapats till simulatorn gjorda med prestanda i åtanke, det vill säga minimera redundansen. Det passar väldigt bra då simulatorn bör använda sig av en databas med hög prestanda som är lätt att uppdatera. Genom att idén med simulatorn är att kunna ändra och redigera data i databasen.

5.3 Simulator

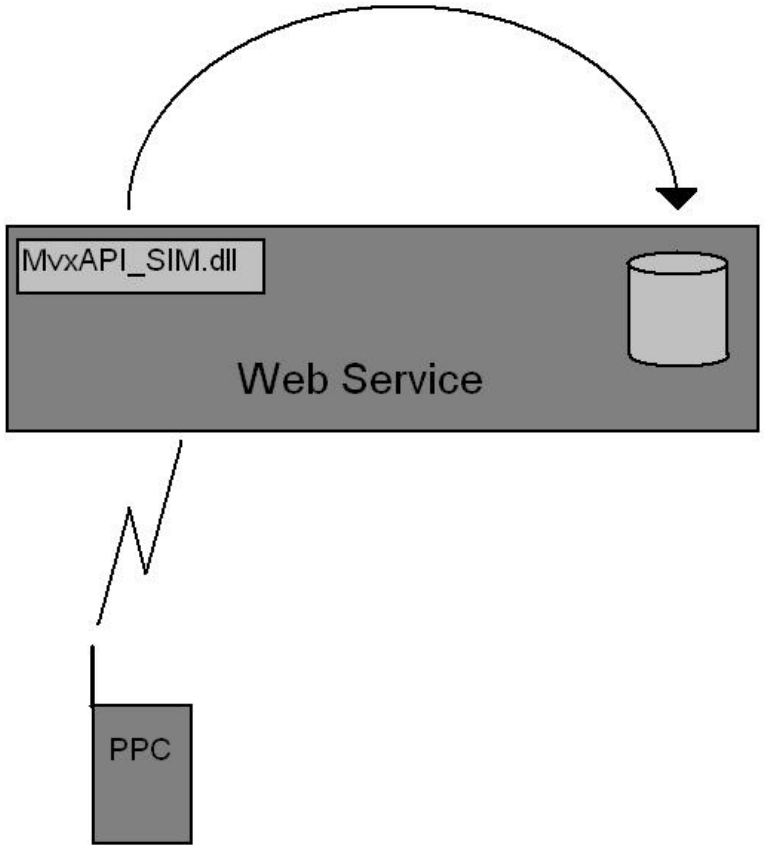
I den här delen kommer vi att beskriva vilken design som användes samt hur implementationen av simulatorn gick till. Denna beskrivning kommer att vara mer ingående då detta är kärnan i examensarbetet. Detta för att ge en tydlig bild över arbetets gång och hur vår tankesätt har sett ut.

5.3.1 Borttagning av Movex/M3 referenser

Första steget vid implementationen av simulatorn var att kommentera bort alla Movex/M3 referenser. Det gjordes för att undvika felmeddelande samt att programmet skulle avslutas på grund av att det inte hade kontakt med affärssystemet. När bortkommentering av referenser skett var nästa steg en fungerande inloggning. Användarnas uppgifter låg i webbservicens databas och kontrollerna för inloggningen gjordes i webbservicen, vilket gjorde att det fungerade på samma sätt som i skarpt läge. Om inloggningen godkändes anropade programmet affärssystemet för kontrollera klockornas överensstämmande. Klockproblemet löstes via att ta bort alla kontroller för anropet och returnera att klockorna var överensstämmande. Efter det gjordes ett inloggningsförsök till systemet, vilket gick bra. När Snitcher Warehouse systemets huvudmeny var tillgänglig valde vi att börja med de funktioner som skriver till databasen, för att på det sättet gradvis expandera databasen så den klarar de uppgifter som krävs. När de funktioner som skrev till databasen var färdiga, började vi med de funktioner som läste från databasen. De sista funktionerna som implementerade var de funktioner som både läste och skrev till databasen.

5.3.2 Ny databas struktur

I början av examensarbetet var planen att använda en ny extern databas för simulatorn. Med vägledning och önskemål från handledaren på Prevas valdes att placera de tabeller som behövdes till simulatorn i webbservicens databas, se Figur 5-2. Detta resulterade i att ingen separat databas behövde skapas vilket underlättade arbetet då en grundstruktur redan existerade. En positiv bieffekt var att företagets egna databashanteringssystem i Visual Studio kunde användas vilket underlättade utförandet av SQL-kommandon betydligt.



Figur 5-2 Länkning till webbservicens databas

5.3.3 Egna funktioner i simulator

I den här delen beskrivs de funktioner som lagts till för att underlätta och effektivisera simulatorn. Dessa funktioner har inget med handdatorn eller webbservicen att göra. Det är funktioner som bara används i och av simulatorn. Genom att strukturen i simulator komponenten skulle vara som i den riktigt komponenten så anropas först de systemfunktioner som är generella för både den riktiga komponenten och simulatorn. Därefter skiljer sig komponenterna åt.

5.3.3.1 InsertSubstring

En funktion för att hantera insättning av substrängar i en sträng implementerades då programmet bygger på stränghantering. Parametrarna som behövs vid anrop till funktionen är *baseString, subString* och *startValue*. Alla parametrarna är *ByVal*, det vill säga att det är värdeanrop som används. Efter att insättningen var färdig returnerades resultatet som en sträng.

baseString: Detta är grundsträngen, den sträng som insättningen sker i, den är av typen String.

subString: Det här är den nya delsträngen som man vill sätta in i den existerande strängen, den är av typen String.

startValue: Denna parameter anger vilken position som substrängen ska starta på i grundsträngen. Då substrängarna kontrolleras till innehåll och placeringen i strängen så tas storleken för den nya substrängen bort i grundsträngen. Detta gör att positionerna förblir korrekta efter insättningen.

5.3.3.2 GetLocationIdForLocationName

När användaren av handdatorn ombeds mata in lagerplats, är det namnet som ska anges. Men i databasen så används id-nummer, då det är lättare och mer praktiskt att arbeta med vid jämförelser och dylika operationer. Därför skapades denna funktion för att enkelt kunna ta reda på id-numret för ett visst givet lagerplatsnamn. Funktionen tar emot namnet på lagerplatsen, sedan kontrollerar den i databasen om namnet är giltigt och returnerar id-numret som en sträng om svar hittas. Anledningen till att svaret är en sträng och inte en integer är att programmet är uppbyggt runt hantering av strängar. Därav är det att föredra om svaret är i rätt format från början.

5.3.3.3 UpdateOrInsertFunction

Denna funktion användes vid de tillfällen man inte visste om det var en INSERT eller UPDATE operation som skulle utföras mot databasen. Genom att använda funktionen så undviker man att skriva kontroller av liknande typ flera gånger. Funktionen tar bort

redundansen genom att göra det generellt för alla insättningar eller uppdateringar. Det räcker då med att göra ett anrop till funktionen så kontrollerar den vilken operation som ska utföras och gör rätt ändringar i den angivna tabellen.

5.3.3.4 RemoveZerosInTheBegining

Funktionen tar bort alla nollor i början av strängen som skickas in. Denna funktion gjordes då vi upptäckte att programmet i några fall fyllde ut det angivna antalet positioner i substrängar med nollor i början om inte talet täckte alla positioner.

5.3.3.5 GetSpecificContentFromConfigFile

Funktionen tar in en parameter, som en sträng, med texten på det man vill hämta ut från konfigureringsfilen. Funktioner söker igenom konfigureringsfilen tills den hittar en matchning på parametern den tog in och returnerar värdet för den variabeln i filen. Hittas ingen matchning returnerar funktionen en tom sträng.

5.3.4 Implementation av handdatorns funktioner

Här följer en mer detaljerad beskrivning av hur implementationen av handdatorns funktioner i simulatorn gick till. Ändringar och tillägg som hör till varje funktion kommer att beskrivas mer ingående. Under utvecklingens gång har den enklaste lösningen valts, detta för att inte ödsla tid på att fundera och skapa funktionalitet som eventuellt inte behövs. Det har gjort att databasen liksom simulatorn har utökats då ett nytt behov uppstått. En del tabeller har bytt namn och funktionalitet under arbetets gång. Dessutom har en del tabeller tagits bort eller delats upp när andra behov uppstått. Det kan medföra att förloppet kan vara lite krångligt att följa, men beskrivningen har gjorts så utförligt som möjligt för att underlätta förståelsen.

5.3.4.1 Lagerläggning

Denna funktion används för att ange lagringsplats för artiklar som inkommer och registreras av lagret. Lagerläggning var den första funktionen som implementerades. Detta för att den vid en första anblick verkade relativt lätt att implementera, eftersom den inte behövde hämta någon information utan bara spara det som användaren matar in till databasen. En av de första

sakerna som gjordes var att skapa en tabell för att lagra informationen i. Den tabellen innehöll fyra kolumner; articleID, batchnumber, storagelocation och quantity se Figur 5-3. *SIM_GOODS* valdes som namn eftersom det var klart att den skulle innehålla information om gods. ArticleID och batchNr valdes som primärnycklar.

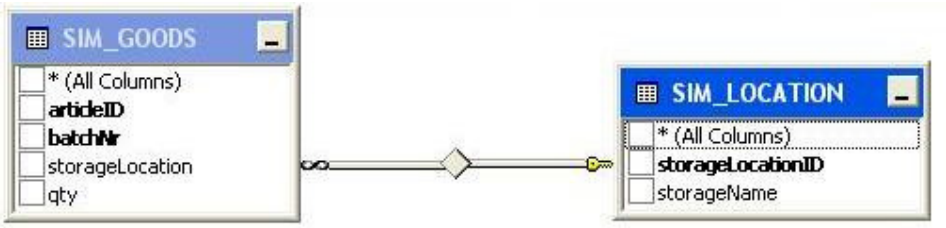
Figur 5-3 SIM_GOODS

5.3.4.2 Lagersaldo

Denna funktion används för att visa kvantiteten och lagerplatsen för det artikelnumret som användaren matar in. Lagersaldo var den andra funktionen som implementerades. Denna funktion valdes tidigt då den enbart läser från databasen, samt att vi insåg efter genomgång av specifikationen att funktionen inte skulle kräva några ändringar av databasen. Den klarade sig på tabellen som gjordes för lagerläggning, se Figur 5-3. Anledningen till att den klarade sig utan ändringar var att den endast hämtar information från de existerande artiklarna i tabellen *SIM_GOODS*.

5.3.4.3 Lagerplatsflytt

Denna funktion används för att byta lagerplats på existerande varor. Lagerplatsen dit en artikel ska flyttas måste vara en existerande lagerplats för att funktionen ska godkänna förflyttningen. Lagerplatsflytt valdes efter Lagersaldo därför att den ansågs klara sig på *SIM_GOODS*. Det visade sig dock av kontrollskäl att det blev lättare att flytta ut informationen om lagerplatsen från *SIM_GOODS* till en egen tabell *SIM_LOCATION,* se Figur 5-4. Flyttningen av lagerplatsen hade ändå gjorts senare för att reducera redundansen, så det ansågs lika bra att skapa tabellen direkt.

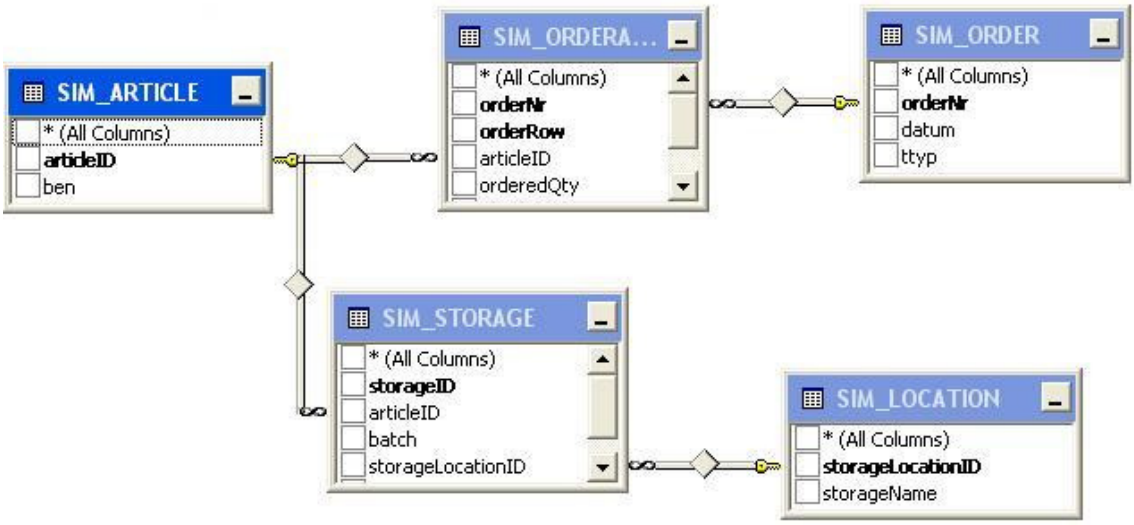


Figur 5-4 SIM_GOODS och SIM_LOCATION

5.3.4.4 Godsmottagning

Denna funktion används för att ta emot gods som inkommer till företaget. När funktionerna som gällde lagerplats och artiklar var klara prioriterades godsmottagningen. Det blev ganska tidigt klart att implementationen av denna funktion skulle innebära stora förändringar av databasstrukturen. Funktionen krävde som första indata ett ordernummer, något som tidigare inte använts i simulatorns databas. Initialt under implementationen av funktion gjordes en tabell för lagring av order, den döptes till *SIM_ORDER*. Från början innehöll den ordernumret, som valdes till primärnyckel, och datum. Efter skapande av *SIM_ORDER* insåg vi att vi skulle få ett många-till-många förhållande mellan en order och dess artiklar. Därför skapades *SIM_ORDERARTICLE,* en tabell som innehåller orderns artiklar.

Ytterligare behövdes en benämning på artiklarna, också här skapades en egen tabell för artiklar, SIM_ARTICLE, där användes artikelns identifikationsnummer som primärnyckel se Figur 5-5. Nu kunde information om ordernas innehåll visas. För att godsmottagningen skulle fungera optimalt behövdes en funktion för själva mottagningen och logiken för orderna. Med det menas bland annat en funktion som kollar om ordern är klar, genom att kontrollera att alla artiklar är med på ordern och korrekt till antalet. Då den funktionaliteten inte var direkt nödvändig för funktionen vid detta tillfälle, dokumenterades de delar som förmodligen skulle komma att behövas i funktionen och vi gick vidare med examensarbetet.



Figur 5-5 Databasens tabeller efter godsmottagning

5.3.4.5 Kundretur

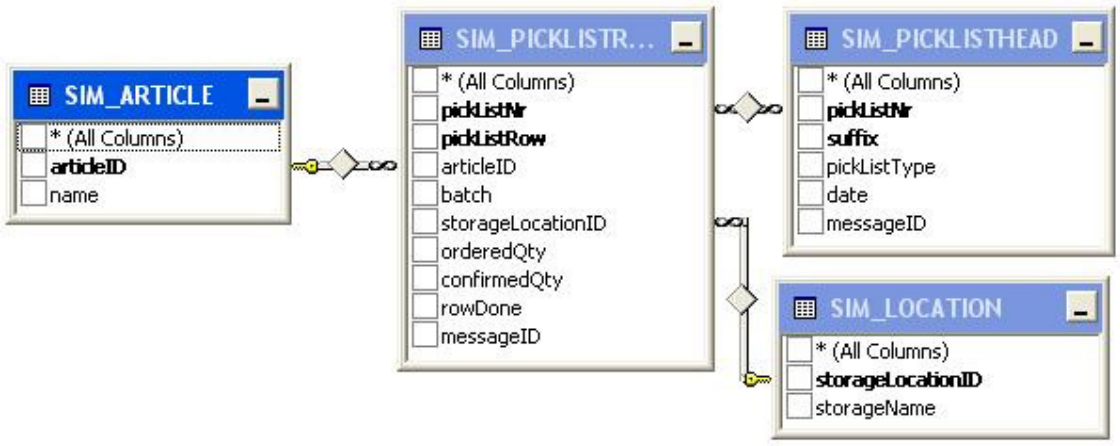
Kundretur valdes näst efter godsmottagning eftersom den verkade ha en likartad funktionalitet och att det därför inte skulle krävas stora ändringar i databasen. När implementationen av den här funktionen hade fortskridit blev det dock uppenbart att den existerande strukturen var felaktigt gjord. Därför gjordes några större ändringar för att komma till rätta med problemet. Felet bestod i en missuppfattning om hur godsmottagningar och kundreturer hanteras. *SIM_ORDER* och *SIM_ORDERARTICLE*, se Figur 5-5, ersattes av *SIM_ORDERHEAD* och *SIM_ORDERROWS*, se Figur 5-6. Dessutom togs kopplingen till *SIM_STORAGE* bort helt eftersom det rörde sig om artiklar som ännu inte lagts till på någon lagerplats.

Figur 5-6 Ny databasstruktur efter implementation av kundretur

Kopplingen till *SIM_ARTICLE* är kvar eftersom den används för att hämta ut artiklarnas namn.

5.3.4.6 Plock TO/Rekvisition

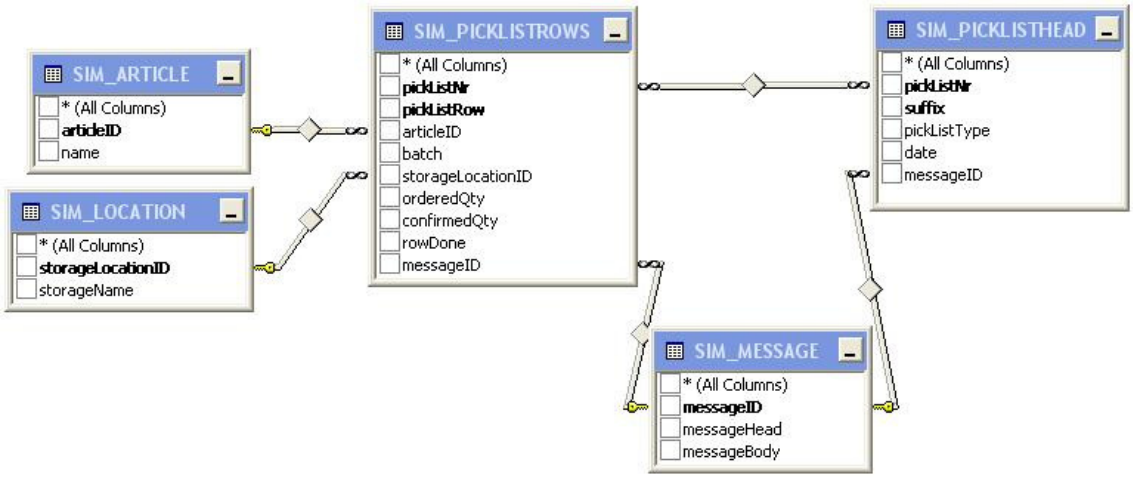
Nästa funktion som implementerades var Plock Tillverkningsorder. Eftersom det var första list funktionen, behövdes nya tabeller läggas till databasen för att den skulle fungera. Anledningen till de nya tabellerna var för att undvika att det skulle bli komplext och ineffektivt att försöka återanvända existerande struktur. Enligt specifikationen skulle databasen vara redigerbar med hjälp av SQL-manager eller dylik programvara, den bör då ha en struktur som är begriplig. För att databasen skulle få ett konsekvent utseende, användes en liknande design som för kundretur tabellerna, se Figur 5-6. Det slutgiltiga resultatet av de nya tabellerna kan ses i Figur 5-7.



Figur 5-7 Databasens tabeller efter Plock Tillverkningsorder/Rekvisition

5.3.4.7 Plock slut/SP

Efter plocktillverkningsorder implementerades plockslutprodukt. Det som skiljer sig mellan plocktillverkningsorder, se Figur 5-7, och plockslutprodukt, är att i den sistnämnda används en meddelande tabell för att kunna visa upp textinformation associerad med listan eller raden, se Figur 5-8. Meddelandet kan användas för att beskriva artiklar som har speciella krav, det kan tillexempel vara en artikel som ska hanteras varsamt eller att en plocklista måste ut innan ett visst datum.

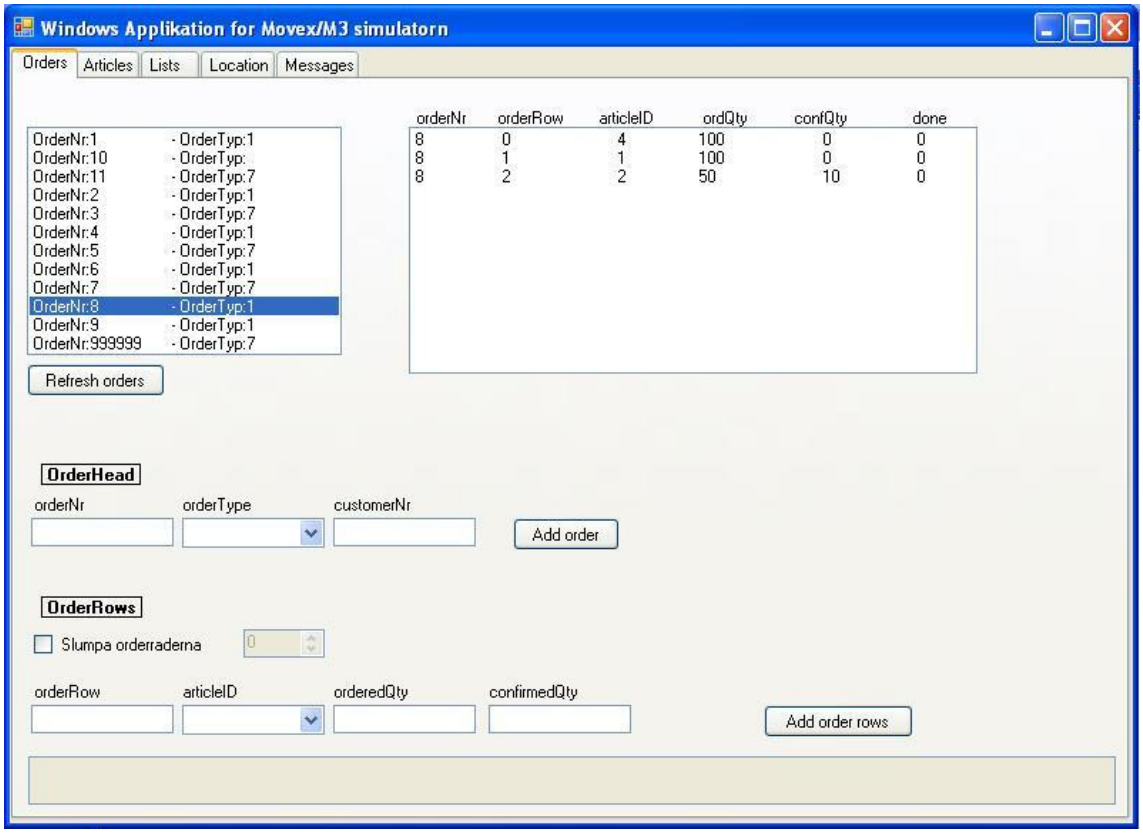


Figur 5-8 Databasensstruktur efter Plock Slutprodukt

5.4 Program för inmatning av testdata, Windows applikation

Problemet med manuell inmatning av data i databasen är att det kan vara svårt att veta vad alla värden betyder. För att lösa problemet var önskemålet från Prevas ett separat program för hanteringen av inmatning av testdata till databasen. Programspråket som användes var C# på grund av att vi är mer bekanta med det när det gäller programmering kopplat till ett *Graphical User Interface*, (GUI). Programmet låter användaren välja mellan fem olika flikar som vardera har hand om en egen typ av data. Med hjälp av programmet kan man skapa order för Godsmottagning och Kundretur, plocklistor för Plock Slutplock, Plock Tillverkningsorder och Kassation. Då en order/plocklista skapas kan användaren välja mellan att antingen mata in innehållet för hand eller genom att slumpmässigt generera innehållet baserat på de artiklar som finns i databasen. I de andra flikarna finns möjligheten att lägga till nya lagerplatser, artiklar och meddelanden. Då inmatningen av informationen färdigställts hanterar programmet all kommunikation med databasen och ser till att alla krav efterföljs. Nedan följer ett par användarexempel med bilder för att ge en ytterligare förståelse av programmet.

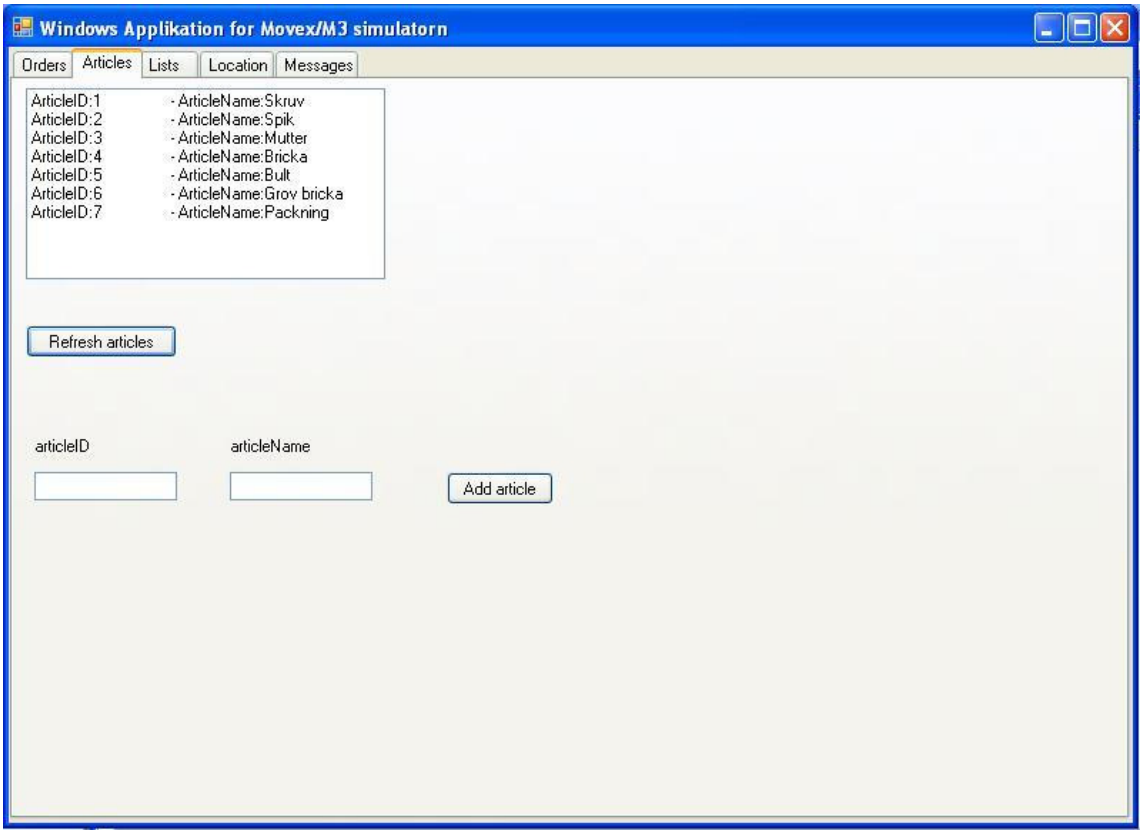
5.4.1 Inläggning av order



Figur 5-9 Windows Applikation order flik

I Figur 5-9 kan man se användarfönstret som används för inläggning av order till simulator databasen. I textrutan i övre vänstra hörnet visas alla order som finns lagrade. Ordernummer och orderns typ skrivs ut för att skilja dem åt, 1 = Godsmottagning och 7 = Kundretur. För att skapa en ny order fyller man i fälten som finns under *OrderHead*. Om en order markerats visas dennes rader i den högra textrutan förutsatt att det finns några. Då en order valts finns möjlighet att fylla i fälten under *OrderRows* för att lägga till en ny rad. Dessutom finns möjligheten att generera raderna automatiskt, då kryssar man i checkboxen och väljer det antal rader som ska läggas till.

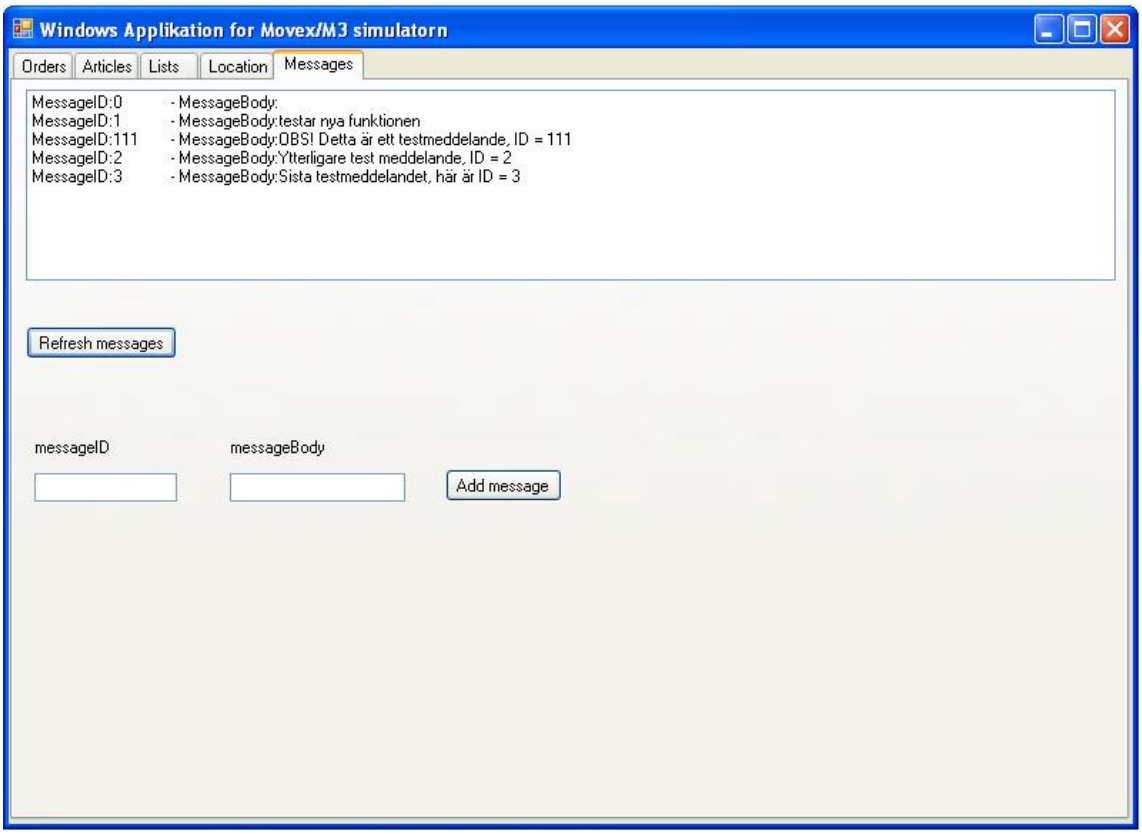
5.4.2 Inläggning av artiklar



Figur 5-10 Windows Applikation artikel flik

I Figur 5-10 kan man se fliken i Windows programmet som används för att lägga till artiklar. Tillgängliga artiklar visas i text rutan med ID och namn. Ny artikel läggs till genom att fylla i ett ID och ett namn, där ID numret måste vara unikt. Dessa används sedan bland annat i order och list flikarna.

5.4.3 Inläggning av meddelanden



Figur 5-11 Windows Applikation meddelande flik

I Figur 5-11 ovan kan man se gränssnittet som används då man vill lägga till nya meddelanden eller ändra existerande meddelanden. I textrutan visas meddelandena som redan finns i systemet. För nytt meddelande så fylls ID och meddelande fältet i. Informationsmeddelanden används för att upplysa användarna om specifika omständigheter eller villkor som gäller för vissa plocklistor och/eller artiklar på listorna. Meddelandet med "0" som ID används för att ett blankt meddelande ska finnas tillgängligt. Detta för minimering av antalet rader i tabellen som har tomma fält.

5.4.4 Återstående arbete på Windows applikationen

Windows Applikationen var ett önskemål från Prevas, som skulle prioriteras om ytterligare tid fanns efter grund uppgiften i examensarbetet. Detta resulterade i en minskning av funktioner och kontroller i applikationen. Stora delar av inläggningen av data till databasen slutfördes dock. Här följer några av de funktioner som prioriterades sist och inte hanns med;

- Lägga till funktion för att ta bort data från databasen så att detta inte behöver göras direkt i tabellen med SQL-manager eller dylik programvara.

- Lägga till funktion för att val av databas till vilken applikationen ska ansluta sig till. Detta för att underlätta arbetet mot icke lokala databaser.

- Lägga till felkontroller för att se till att villkoren i tabellerna efterföljs. Till exempel för att undvika duplicering av primärnycklar.

- Vid slumpning av orderrader och listrader bör kontroller tilläggas in så att de genererade raderna undviker krockar med unika attribut eller nycklar i tabellerna.

5.5 Simulatorns framtid

Då arbetet med simulatorn och examensarbetet närmade sig slutet såg man inte bara tillbaka på det som åstadkommits, man undrade också hur framtiden för simulatorn ska komma att se ut. I nuläget kommer simulatorn främst användas för demonstration i kontakten med nya kunder, felsökning av kod och för att underlätta vidareutveckling av Snitcher Warehouse produkten. Redan nu har mer åstadkommits än vad som stod i specifikationen i form av ett program för inmatningen av testdata.

Andra användningsområden som skulle kunna bli aktuellt i framtiden är till exempel utbildning. Utbildningen skulle rikta sig till arbetarna på företagen som köpt ett Snitcher Warehouse system. Möjligheten finns skriva ut etiketter och genomföra simulerade flyttningar, mottagning av nytt gods och många fler funktioner i systemet. Fördelen med det arbetssättet är möjligheten att testa och lära sig systemet utan att riskera några ekonomiska missöden, eftersom det är simulerad data som används. Dessutom får lagerarbetarna se och prova på systemet som de kommer att använda, vilket borde underlätta inlärningen. En simulering samtidigt med genomgångar om systemets funktioner borde ge en bra förståelse och en relativt kort inlärningskurva.

6 Problem

Följande kapitel handlar om de problem som uppdagats under utvecklingen av simulator. En förklaring följer över hur problemen uppstod och dess lösningar.

6.1 Objekt

I början såg funktionsanropen snarlika ut och substrängarna såg ut att ha samma placering och storlek i parametrarna. Efter genomläsning i pärmar och manualer samt granskning av koden insågs att så inte var fallet. Fördelarna med användningen av objekt blev försumbara i förhållande mot arbetet som skulle krävas för att läsa in alla olika substrängarna i objekten. Inläsningen av substrängarna skulle då behöva vara specialanpassad för varje anrop. Några anrop var mer lika varandra, men placeringen och storleken på substrängarna i dem kunde ändå variera så mycket att det inte gick att hitta en generell struktur i dem. Detta ledde till att valet blev att fortsätta med samma variant av stränghantering som övriga systemet hade.

6.2 Olika versioner av programvara

Ett problem som orsakade fördröjningar under utvecklingen var att webbservicen som användes i början inte var den senaste versionen. Detta upptäcktes först då vi skulle implementera första listfunktionen i simulatorn. Det visade sig då att handdatorn hade en nyare version av systemet än Webbservicen. För att låsa upp transaktionerna i databasen hade de som konstruerat systemet lagt in nya funktioner och ändrat om strukturen i jämförelse med den vi hade till vårt förfogande. Problemet verkade till en början inte vara så svårt att justera, men den gamla versionen hängde kvar och orsakade både problem och fördröjning i utvecklingen.

6.3 Otillgänglig uppdrags handledare

Något som orsakade problem och förvirring i början av examensarbetet var att handledaren på företaget under perioder varit svår att nå. Detta resulterade i en del egna beslut som vid provkörning visade sig vara felaktiga. Därför har vi vid upprepade tillfällen gått tillbaka långt i koden för att rätta till felen. Detta skapade i början en osäkerhet för hur långt arbetet kunde gå vidare eftersom man inte visste om det som gjorts tidigare motsvarade önskemålet.

6.4 Flera användare av arbetsstation

Arbetsstationen som fanns till förfogande har använts av andra personer till och från. Detta resulterade i förändringar av inställningar skett vilket gjorde att miljön inte längre var fullt fungerande. På grund av att vår dator använts av andra har mycket tid gått åt för att söka efter och rätta till fel som kunnat undvikas om vi haft en egen arbetsstation. Problemen har bland annat innefattat ändringar av versionen på IIS och framework i Visual Studio. Då ändringarna i sig inte varit svåra att justera till så har det gått åt mycket tid att hitta felkällan. Detta eftersom felen som uppstår varit svåra att tolka. Ytterligare problem som tog mycket tid var datumformatet i SQL krånglade, vilket ledde till problem med hämtning och skrivning till databasen. Även då flertalet anställda på Prevas varit och försökt hjälpa oss så visste ingen vad felet berodde på. Problemet löstes tillfälligt genom att skriva en funktion i webbservice delen av programmet som ändrar formatet på datumet. Detta påverkar dock inte projektet mycket som helhet eftersom problemet inte uppstått då simulatorn testats på andra datorer.

6.5 Svårtolkad testdata för ovana användare

Programmet använder sig av siffror för att bland annat identifiera olika typer av plocklistor. Utöver det använder vår simulator siffror för att ange speciella meddelanden som kan gälla för en viss plocklista eller för vissa artiklar på listan. Dessutom används siffervärden för att ange om en order i SIM_ORDERHEAD är av typen godsmottagning eller en kundretur. För den nye användaren kan alla dessa nummer göra det svårt att lägga in testdata. För att underlätta för användaren ombads vi skriva en användarmanual för simulatorn och dess tabeller i databasen. Dessutom har det arbetats på en bonusprogramvara, se avsnitt 5.4, som hjälper användaren med inmatningen/genereringen av testdata. Detta program är skrivet i C#.

Bonusprogrammet gör det betydligt enklare att skapa bra testdata då man undviker att skriva in direkt i databas tabellerna. Personer som kommer helt nya till systemet kan med hjälp av inmatningsprogrammet och manualen skapa den testdata som de behöver. Dessutom kan användaren generera slumpmässiga rader för plocklistor och order automatiskt baserat på de artiklar och lagerutrymmen som finns i systemet. Bonusprogrammet är ett bra alternativ då man snabbt vill kontrollera funktionaliteten i simulatorn eller skapa testdata för demonstration.

7 Resultat och Slutsatser

Vid starten av examensarbetet hade vi begränsade kunskaper om programspråket Visual Basic. Dessutom visste vi inte hur lagerhanteringssystem eller affärssystem fungerade. Dessa faktorer gjorde det svårt att bedöma hur mycket arbete som skulle krävas för att genomföra uppgiften examensarbetet bestod av. Den något bristande kunskapen gjorde att vi spenderade den första tiden med att bekanta oss med miljön och språket. Från början fick vi inte någon klar kravspecifikation, det gjorde det än svårare att bedöma uppgiftens omfattning. Vi pratade med handledaren och bad honom skriva ner de krav och önskemål de hade på den slutgiltiga funktionaliteten. Detta resulterade i att vi fick en sammanställd och tydlig kravspecifikation.

Resultatet av examensarbetet är en fungerande simulator som uppfyller samtliga krav som finns angivna på kravspecifikationen. Simulatorn som vi byggt kan simulera alla de användarfall som Prevas önskade. Utöver det som står angivet i kravspecifikationen har även ett program för att sköta inmatningen av testdata utvecklats. Det programmet skapades för att underlätta användningen för personer som inte är insatta i simulatorn.

Problemen som uppstått under projektets gång var mest relaterade till miljön och handledarens närvaro. Problemet med miljön handlade om att vi inte var ensamma om att använda arbetsstationen som vi blivit tilldelade. Detta gjorde att en del inställningar och frameworks var ändrade när vi kom tillbaka efter en helg. Det ultimata vore att arbeta på en egen arbetsstation. Om det inte finns möjlighet till det rekommenderar vi att man har god kunskap om hur de program som är inblandade i arbetet fungerar.

Vi rekommenderar också att man är aktiv och lyssnar när man får hjälp med felsökning och uppsättning av miljön, detta för att undvika att vara beroende av en annan part på grund av bristande kunskap inom den egna gruppen. Det gör också att om problem uppstår, går de snabbare att lösa om kunskap om miljön finns inom gruppen. Dessutom rekommenderar vi att man hör efter med handledaren hur det man gjort fungerar i förhållande till vad som efterfrågas om tvivel finns. Det är bättre att fråga direkt någon oklarhet kommer upp istället för att behöva ändra stora delar av projekten senare när det framkommer att lösningen inte motsvarade önskemålet eller kraven.
Vi rekommenderar också att man skriver anteckningar för de problem som uppkommer under arbetets gång. En bra dokumentation underlättar betydligt för skrivandet av rapporten, då kan

man gå tillbaka och läsa om hur man tänkt vid lösningar av problem. Det blir också lättare att utvärdera sin egen insats om man har en dokumentation där det står hur man tänkt och gjort.

Till sist rekommenderar vi att man börjar skriva på uppsatsen tidigt, så tidigt som möjligt. Även om det bara blir ett par rader eller ett litet avsnitt åt gången så underlättar det mycket att fördela arbetet då man slipper stressen över att skriva allt i slutet av examensarbetet.

8 Appendix - Förklaringar

API

Application Programming Interface är det regelverk för hur en programvara kan kommunicera med en annan programvara. Regelverket beskrivs oftast som en mängd funktionsanrop och fungerar på samma sätt som kontrakt. Anropas den korrekt så kommer den att fungera korrekt. Det är upp till användaren av regelverket att efterhålla och garantera funktionaliteten.

IIS

Internet Information Services är Microsofts webbserverprogramvara. Programvaran finns som tillval till de flesta Windowsbaserade operativsystem. I vårt examensarbete har vi använt oss av IIS v 5.1, som bland annat finns tillgängligt för XP Professional och XP Media Center Edition.

XML

Genom att använda eXtensible Markup Language får man ett gemensamt format för informationen. Detta underlättar spridningen av information för informationssystem, speciellt via Internet.

SOAP

Simple Object Access Protocol är ett protokoll som används då decentraliserade och distribuerade system ska utbyta information. SOAP bygger på XML och används oftast tillsammans med http, men kan även användas med andra protokoll.

WSDL

Web Services Description Language är ett språk som används för att beskriva Web Services och förklara hur den ska användas, det är baserat på XML.

UDDI

Universal Description Discovery and Integration är en standardiserad katalogmodell för webbtjänster som bygger på XML. Från början var det tänkt att bli en del av kärnan i Web Service. UDDI används av SOAP meddelanden för att få tillgång till WSDL dokument. En UDDI katalog innehåller information om webbtjänsters funktionalitet och beskrivning.

HTTP

HyperText Transfer Protocol används för kommunikation mellan en klient och en server. Klienten är slutanvändare och server är en webbsida. Informationen utbyts genom att klienten skickar förfrågningar, REQUEST, och servern svarar, RESPONSE. Protokollet är *stateless*, vilket innebär att varje förfrågning behandlas separat, det vill säga ingen status sparas på servern.

DLL

Dynamic Link Library är Microsofts lösning på att dela ut programbibliotek. Används för att dela funktioner mellan flera olika program. DLL-filer kompileras inte i runtime utan de refereras in till programmen.

TCP/IP

Transmission Control Protocol/Internet Protocol är två protokoll som används för nätverkskommunikation.

WLAN

Wireless Local Area Network är ett trådlöst nätverk. Fördelen med trådlös kommunikation är att man kan flytta omkring inom nätverket utan att behöva bryta kontakten och koppla upp den igen. Det förutsätter dock att man håller sig inom nätverkets täckning.

Referenser

[1] 2009-04-23 - http://sv.wikipedia.org/wiki/Simulator

[2] *2009-04-23* - http://en.wikipedia.org/wiki/Simulator

[3] 2009-04-23 - http://en.wikipedia.org/wiki/Pocket_PC

[4] Utgåva V2p3, 2007-01-31 - Funktionsspecifikation ADF-System 1.0

[5] Utgåva V2, 2006-11-23 - Användarmanual ADF Zebra 1.0 Handdatorfunktioner

[6] Utgåva 5, 2007 – Fundamentals of Database Systems, Ramez Elmasri & Shamkat B. Navathe

[7] 2009-05-04 – Transaktionslayouter Movex Java Warehouse

[8] 2009-05-04 – Prevas dokumentation om Snitcher Warehouse

[9] 2009-05-11 - http://www.bra.edu.stockholm.se/utbildning/program/teknikprogram/images/litensaabflygsimulator.jpg

[10] 2009-05-12 - http://sv.wikipedia.org/wiki/Web_service

[11] 2009-05-16 - http://www.prevas.se/

[12] 2009-05-16 - http://sv.wikipedia.org/wiki/Wlan

A Windows Applikationens kod

1. Main

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Warehouse_SimApp_CSharp
{
    public partial class Main : Form
    {
        public Main()
        {
            InitializeComponent();
            Init();
        }

        private void Init()
        {
            DisplayOrderTypeInCB();
            DisplayArticleIDInCB();
            DisplayPickListTypes();
            DisplayStorageLocation();
            UpdateArtIDCB();
            UpdateMessagesCB();
        }

//************************************************************************
//List                              funktioner                          start
//************************************************************************
        private void ListPickListType()
        {
            _listTypes = Mapper.Instance.GetPickListTypes();
        }
        private void ListStorageLocation()
        {
            _storageLocation = Mapper.Instance.GetLocationFromDbOrEmptyList();
        }
        private void ListArticles()
        {
```

```
            _article = Mapper.Instance.GetArticleFromDbOrEmptyList();
        }
        private void ListOrderTypes()
        {
            _orderTypes = Mapper.Instance.GetOrderTypesFromDbOrEmptyList();
        }
        private void ListOrderHead()
        {
            _orderHead = Mapper.Instance.GetOrderHeadFromDbOrEmptyList();
        }
        private void ListOrderRows()
        {
            _orderRows                                                              =
Mapper.Instance.GetOrderRowsFromDbOrEmptyList(_orderHead[orderLB.SelectedIndex]);
        }
        private void ListPickListHead()
        {
            _listHead = Mapper.Instance.GetListHeadFromDbOrEmptyList();
        }
        private void ListPickListRows()
        {
            _listRows                                                               =
Mapper.Instance.GetListRowFromDbOrEmptyList(_listHead[listLB.SelectedIndex]);
        }
        private void ListMessages()
        {
            _messages = Mapper.Instance.GetMessagesFromDb();
        }
//*************************************************************************
//List funktioner slut
//*************************************************************************

//*************************************************************************
//Display                   list                    funktioner                  start
//*************************************************************************
        private void DisplayPickListTypes()
        {
            ListPickListType();
            foreach (DataRow dr in _listTypes)
                if (!pickListTypeCB.Items.Contains(dr["pickListType"]))
                    pickListTypeCB.Items.Add(dr["pickListType"]);
        }

        private void DisplayStorageLocation()
        {
            ListStorageLocation();
```

```
            foreach (DataRow dr in _storageLocation)
                if (!sLIDCB.Items.Contains(dr["storageLocationID"]))
                    sLIDCB.Items.Add(dr["storageLocationID"]);
        }

        private void DisplayArticleIDInCB()
        {
            ListArticles();
            foreach(DataRow dr in _article)
                if (!articleIDCB.Items.Contains(dr["articleID"]))
                    articleIDCB.Items.Add(dr["articleID"]);
        }

        private void DisplayOrderTypeInCB()
        {
            ListOrderTypes();
            foreach (DataRow dr in _orderTypes)
                if (!orderTypeCB.Items.Contains(dr["orderType"]))
                    orderTypeCB.Items.Add(dr["orderType"]);
        }

        private void UpdateOrderLB()
        {
            orderLB.Items.Clear();
            messageTB.Text = "";
            ListOrderHead();
            foreach (DataRow dr in _orderHead)
            {
                orderLB.Items.Add("OrderNr:" + dr["orderNr"] + "\t - OrderTyp:" +
dr["orderType"]);
            }
        }

        private void UpdateOrderRowsLB()
        {
            orderRowsLB.Items.Clear();
            messageTB.Text = "";
            ListOrderRows();
            foreach (DataRow dr in _orderRows)
            {
                orderRowsLB.Items.Add(dr["orderNr"] + "\t      " + dr["orderRow"] +
"\t\t" + dr["articleID"] + "    " + dr["orderedQty"] + "\t" + dr["confirmedQty"] +
"\t      " + dr["rowDone"]);
            }
        }
```

```
        private void UpdateArticleLB()
        {
            articleLB.Items.Clear();
            messageTB.Text = "";
            ListArticles();
            foreach (DataRow dr in _article)
            {
                articleLB.Items.Add("ArticleID:"   +   dr["articleID"]   +   "\t   -
ArticleName:" + dr["name"]);
            }
        }

        private void UpdateListLB()
        {
            listLB.Items.Clear();
            messageTB.Text = "";
            ListPickListHead();
            foreach (DataRow dr in _listHead)
            {
                listLB.Items.Add("PickListNr:" + dr["pickListNr"] + "\t - Suffix:"
+ dr["suffix"] + "\t - PickListType:" + dr["pickListType"]);
            }
        }

        private void UpdateArtIDCB()
        {
            artIDCB.Items.Clear();
            ListArticles();
            foreach (DataRow dr in _article)
            {
                artIDCB.Items.Add(dr["articleID"]);
            }
        }

        private void UpdatePickListRowsLB()
        {
            listRowsLB.Items.Clear();
            ListPickListRows();
            foreach (DataRow dr in _listRows)
            {
                listRowsLB.Items.Add(dr["pickListRow"] + "\t" + dr["articleID"] +
"\t" + dr["batch"] + "\t" + dr["storageLocationID"] + "\t" + dr["orderedQty"] +
"\t" + dr["confirmedQty"] + "\t" + dr["rowDone"] + "\t" + dr["messageID"]);
            }
        }
```

```
        private void UpdateLocationList()
        {
            ListStorageLocation();
            locLB.Items.Clear();
            foreach (DataRow dr in _storageLocation)
                locLB.Items.Add("LocID   -   "   +   dr["storageLocationID"]   +   "
\t\tLocName - " + dr["storageName"]);
        }

        private void UpdateMessagesCB()
        {
            ListMessages();
            pLmessageIDCB.Items.Clear();
            pLRowMessCB.Items.Clear();
            foreach (DataRow dr in _messages)
            {
                pLmessageIDCB.Items.Add(dr["messageID"]);
                pLRowMessCB.Items.Add(dr["messageID"]);
            }
        }
//*************************************************************************
//Display list funktioner slut
//*************************************************************************

//*************************************************************************
//Order                                                                start
//*************************************************************************
        private void OrderB_Click(object sender, EventArgs e)
        {
            UpdateOrderLB();
        }

        private void OrderLB_SelectedIndexChanged(object sender, EventArgs e)
        {
            UpdateOrderRowsLB();
        }

        private void AddOrderB_Click(object sender, EventArgs e)
        {
            if  (orderNrTB.Text  !=  null  &&  orderTypeCB.SelectedIndex  !=  -1  &&
customerNrTB.Text != null)
            {
                string orderNr = orderNrTB.Text;
                string orderType = orderTypeCB.SelectedItem.ToString();
                string customerNr = customerNrTB.Text;
                Mapper.Instance.addNewOrderToDb(orderNr, orderType, customerNr);
```

```
                UpdateOrderLB();
            }
            else
                messageTB.Text = "Error - måste fylla i alla värden";
        }

        private void AddOrderRowsB_Click(object sender, EventArgs e)
        {
            if (orderLB.SelectedIndex != -1)
            {
                if (randomCB.Checked == true)
                {
                    string                          orderNr                          =
_orderHead[orderLB.SelectedIndex]["orderNr"].ToString();
                    int randomMax = Mapper.Instance.getNrOfArticles();
                    Random rand = new Random();
                    int lastIndex;

                    //Kontrollera om ordern är tom
                    if (_orderRows.Count > 0)
                        lastIndex    =    int.Parse(_orderRows[_orderRows.Count    -
1]["orderRow"].ToString());
                    else
                        lastIndex = 0;

                    //Slumpa fram antalet rader
                    for   (int   i   =   lastIndex   +1   ;   i   <=   lastIndex   +
numericUpDown1.Value; i++)
                    {
                        Mapper.Instance.AddNewOrderRowsToDb(orderNr,   i.ToString(),
rand.Next(1, randomMax).ToString(), 100.ToString(), 0.ToString(), 0.ToString());
                    }
                }
                else
                {
                    //Ta in användarens uppgifter
                    if (orderRowTB.Text != null && articleIDCB.SelectedIndex != -1
&& ordQtyTB.Text != null && confQtyTB.Text != null)
                    {
                        string                        orderNr                        =
_orderHead[orderLB.SelectedIndex]["orderNr"].ToString();
                        string orderRow = orderRowTB.Text;
                        string articleID = articleIDCB.SelectedItem.ToString();
                        string ordQty = ordQtyTB.Text;
                        string confQty = confQtyTB.Text;
```

```
                    Mapper.Instance.AddNewOrderRowsToDb(orderNr,       orderRow,
articleID, ordQty, confQty, 0.ToString());
                }
                else
                    messageTB.Text = "Error - måste fylla i alla värden";
            }
        }
        else
            messageTB.Text = "Error - ingen vald order";

        UpdateOrderRowsLB();
    }

    private void RandomCB_CheckedChanged(object sender, EventArgs e)
    {
        if (randomCB.Checked)
            numericUpDown1.Enabled = true;
        else
            numericUpDown1.Enabled = false;
    }
//*************************************************************************
//Order end
//*************************************************************************

//*************************************************************************
//Article                                                              start
//*************************************************************************
    private void ArticleB_Click(object sender, EventArgs e)
    {
        messageTB.Text = "";
        UpdateArticleLB();
    }

    private void AddArticleB_Click(object sender, EventArgs e)
    {
        messageTB.Text = "";
        if (addArticleIDTB.Text != null && addArticleNameTB.Text != null)
        {
            string articleID = addArticleIDTB.Text;
            string articleName = addArticleNameTB.Text;

            Mapper.Instance.AddNewArticleToDb(articleID, articleName);
            UpdateArticleLB();
        }
        else
            messageTB.Text = "Error - måste fylla i alla värden";
```

```
        }
//*************************************************************************
//Article end
//*************************************************************************

//*************************************************************************
//List                                                                start
//*************************************************************************
        private void ListB_Click(object sender, EventArgs e)
        {
            UpdateListLB();
        }

        private void ListLB_SelectedIndexChanged(object sender, EventArgs e)
        {
            ListPickListRows();
            UpdatePickListRowsLB();
        }

        private void AddPickListHeadB_Click(object sender, EventArgs e)
        {
            if (pickListNrTB.Text != null && suffixTB.Text != null &&
pickListTypeCB.SelectedIndex != -1)
            {
                string pickListNr = pickListNrTB.Text;
                string suffix = suffixTB.Text;
                string pickListType = pickListTypeCB.SelectedItem.ToString();

                Mapper.Instance.AddNewListHeadToDb(pickListNr,              suffix,
pickListType, "Exjobb");
                UpdateListLB();
            }
            else
                messageTB.Text = "Error - måste fylla i alla värden.";
        }

        private void RandPLCB_CheckedChanged(object sender, EventArgs e)
        {
            if(randPLCB.Checked)
                numericUpDown2.Enabled = true;
            else
                numericUpDown2.Enabled = false;
        }

        private void PickListAddRowsB_Click(object sender, EventArgs e)
        {
```

```
            if (listLB.SelectedIndex != -1)
            {
                string                          pickListNr                          =
_listHead[listLB.SelectedIndex]["pickListNr"].ToString();
                if (randPLCB.Checked == true)
                {
                    int randomMax = Mapper.Instance.getNrOfArticles();
                    Random rand = new Random();
                    int lastIndex;

                    //Kontrollera om ordern är tom
                    if (_listRows.Count > 0)
                        lastIndex     =     int.Parse(_listRows[_listRows.Count     -
1]["pickListRow"].ToString());
                    else
                        lastIndex = 0;

                    //Slumpa fram antalet rader
                    for   (int   i   =   lastIndex   +   1;   i   <=   lastIndex   +
numericUpDown2.Value; i++)
                    {
                        Mapper.Instance.AddNewPickListRowToDb(pickListNr,
i.ToString(),  rand.Next(1,  randomMax),  rand.Next(1,  8000).ToString(),  "1",  "100",
"0", 0, pLmessageIDCB.SelectedItem.ToString());
                    }
                }
                else
                {
                    int batchNr;
                    int articleNr;
                    if       (int.TryParse(artIDCB.SelectedItem.ToString(),       out
articleNr))
                    {
                        if (int.TryParse(pickListBatchTB.Text, out batchNr))
                        {
                            if (pickListRowTB.Text != null && pickListBatchTB.Text
!=  null  &&  sLIDCB.SelectedIndex  !=  -1  &&  pickListOrdqtyTB.Text  !=  null  &&
pickListConfqtyTB.Text != null)
                            {
                                Mapper.Instance.AddNewPickListRowToDb(pickListNr,
pickListRowTB.Text,                articleNr,                pickListBatchTB.Text,
sLIDCB.SelectedItem.ToString(),  pickListOrdqtyTB.Text,  pickListConfqtyTB.Text,  0,
pLRowMessCB.SelectedItem.ToString());
                            }
                            else
                            {
```

```
                                messageTB.Text = "Error - incorrect input";
                            }
                        }
                        else
                        {
                            messageTB.Text = "Error - incorrect batchnr";
                        }
                    }
                    else
                    {
                        messageTB.Text = "Error - incorrect articlenr";
                    }
                }
            }
            else
                messageTB.Text = "Error - ingen vald order";
            UpdatePickListRowsLB();
        }
//*************************************************************************
//List                                                                 end
//*************************************************************************

//*************************************************************************
//Location                                                           start
//*************************************************************************
        private void RefreshLocB_Click(object sender, EventArgs e)
        {
            UpdateLocationList();
        }

        private void AddLocB_Click(object sender, EventArgs e)
        {
            messageTB.Text = "";
            if (locIDTB.Text != null)
            {

                if (locNameTB.Text != null)
                {
                    string storageLocID = locIDTB.Text;
                    string storageLocName = locNameTB.Text;

                    Mapper.Instance.AddNewStorageLocation(storageLocID,
storageLocName);

                    UpdateLocationList();
                }
```

```
                else
                    messageTB.Text = "Location name måste anges!";
            }
            else
                messageTB.Text = "Location ID måste anges!";
        }
//*************************************************************************
//Location                                                              end
//*************************************************************************

//*************************************************************************
//Message                                                             start
//*************************************************************************
        private void UpdateMessLBB_Click(object sender, EventArgs e)
        {
            messageTB.Text = "";
            UpdateMessageLB();
        }

        private void UpdateMessageLB()
        {
            messageLB.Items.Clear();
            messageTB.Text = "";

            ListMessages();
            foreach (DataRow dr in _messages)
            {
                messageLB.Items.Add("MessageID:"   +   dr["messageID"]   +   "\t   -
MessageBody:" + dr["messageBody"]);
            }
        }

private void AddNewMessageB_Click(object sender, EventArgs e)
        {
            string newMessageID;
            string newMessageBody;

            if (messageIDTB.Text != null)
            {
                if (messageBodyTB.Text != null)
                {
                    newMessageID = messageIDTB.Text;
                    newMessageBody = messageBodyTB.Text;
                    Mapper.Instance.AddNewMessageToDb(newMessageID,
newMessageBody);
                    UpdateMessageLB();
```

```
                }
                else
                    messageTB.Text = "messageBody måste anges!";
            }
            else
                messageTB.Text = "messageID måste anges!";
        }
//*************************************************************************
//Message                                                                end
//*************************************************************************

        //Privates
        private List<DataRow> _orderHead;
        private List<DataRow> _orderRows;
        private List<DataRow> _orderTypes;
        private List<DataRow> _article;
        private List<DataRow> _listHead;
        private List<DataRow> _listRows;
        private List<DataRow> _listTypes;
        private List<DataRow> _storageLocation;
        private List<DataRow> _messages;
    }
}
```

2. Mapper

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data.Common;
using System.Data;

namespace Warehouse_SimApp_CSharp
{
    class Mapper
    {
        //Constructor
        public Mapper()
        {
            //DB - connection
            sqlConnection = new SqlConnection(@<address to database>);
        }

        //Singelton
        public static Mapper Instance
```

```
        {
            get
            {
                if (instance == null)
                    instance = new Mapper();
                return instance;
            }
        }
//**************************************************************************
//Public - add test and controls here before private functions call
//**************************************************************************

        //Pre: True
        //Post: Has returned a list with all the elements in SIM_ORDERHEAD
        public List<DataRow> GetOrderHeadFromDbOrEmptyList()
        {
            return GetOrderHeadsFromDB();
        }

        //Pre: True
        //Post: Has returned a list with all the elements for a order in
SIM_ORDERROWS
        public List<DataRow> GetOrderRowsFromDbOrEmptyList(DataRow orderHead)
        {
            return GetOrderRowsFromDB(orderHead);
        }

        //Pre: True
        //Post: Has added the order to the table SIM_ORDERHEAD
        public void AddNewOrderToDb(string orderNr, string orderType, string
customerNr)
        {
            AddOrderToDB(orderNr, orderType, customerNr);
        }

        public int GetNrOfArticles()
        {
            return GetNrOfArticlesFromDB();
        }

        public void AddNewOrderRowsToDb(string orderNr, string orderRow, string
articleID, string orderedQty, string confirmedQty, string rowDone)
        {
            AddOrderRowToDB(orderNr, orderRow, articleID, orderedQty, confirmedQty,
rowDone);
        }
```

```
        public List<DataRow> GetArticleFromDbOrEmptyList()
        {
            return GetArticlesFromDB();
        }

        public void AddNewArticleToDb(string articleID, string articleName)
        {
            AddArticleToDB(articleID, articleName);
        }

        public List<DataRow> GetListHeadFromDbOrEmptyList()
        {
            return GetListHeadsFromDB();
        }

        public List<DataRow> GetLocationFromDbOrEmptyList()
        {
            return GetStorageLocationsFromDB();
        }

        public List<DataRow> GetListRowFromDbOrEmptyList(DataRow dr)
        {
            return GetListRowsFromDB(dr);
        }

        public List<DataRow> GetOrderTypesFromDbOrEmptyList()
        {
            return GetOrderTypesFromDB();
        }

        public List<DataRow> GetPickListTypes()
        {
            return GetPickListTypesFromDB();
        }

        public void AddNewListHeadToDb(string pickListNr, string suffix, string
pickListType, string changedBy)
        {
            AddListHeadToDB(pickListNr, suffix, pickListType, changedBy);
        }

        public void AddNewPickListRowToDb(string pickListNr, string pickListRow,
int articleNr, string batch, string storageLocationID, string orderedQty, string
confirmedQty, int rowDone, string messageID)
        {
```

```
            AddNewPickListRowToDB(pickListNr,    pickListRow,    articleNr,    batch,
storageLocationID, orderedQty, confirmedQty, rowDone, messageID);
        }

        public     void     AddNewStorageLocation(string     storageLocID,     string
storageLocName)
        {
            AddNewStorageLocationToDB(storageLocID, storageLocName);
        }

        public List<DataRow> GetMessagesFromDb()
        {
            return GetMessagesFromDB();
        }
//************************************************************************
//Private         -         tests         and         controls         =         OK
//************************************************************************

        private List<DataRow> GetOrderHeadsFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            string[] stringObj = new string[2];

            SqlDataAdapter  da  =  new  SqlDataAdapter("SELECT  orderNr  AS  [orderNr],
orderType AS [orderType] FROM SIM_ORDERHEAD", sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
            }

            return result;
        }

        private List<DataRow> GetOrderRowsFromDB(DataRow orderHead)
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter  da  =  new  SqlDataAdapter("SELECT  *  FROM  SIM_ORDERROWS
WHERE orderNr = '" + orderHead["orderNr"] + "'", sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
```

```
                result.Add(dr);
            }

            return result;
        }

        private void AddOrderToDB(string orderNr, string orderType, string
customerNr)
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_ORDERHEAD (orderNr,
orderType, date, customerNr) VALUES ('" + orderNr + "', '" + orderType + "', '" +
DateTime.Now + "', '" + customerNr + "')", sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        private int GetNrOfArticlesFromDB()
        {
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM SIM_ARTICLE",
sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");

            return ds.Tables["Collection"].Rows.Count;
        }

        private void AddOrderRowToDB(string orderNr, string orderRow, string
articleID, string orderedQty, string confirmedQty, string rowDone)
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_ORDERROWS (orderNr,
orderRow, articleID, orderedQty, confirmedQty, rowDone) VALUES ('" + orderNr.Trim()
+ "', '" + orderRow + "', '" + articleID + "', '" + orderedQty + "', '" +
confirmedQty + "', '" + rowDone + "')", sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        private List<DataRow> GetArticlesFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM SIM_ARTICLE",
sqlConnection);
            DataSet ds = new DataSet();
```

```
            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
            }

            return result;
        }

        private List<DataRow> GetStorageLocationsFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM SIM_LOCATION",
sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
            }

            return result;
        }

        private void AddArticleToDB(string articleID, string articleName)
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_ARTICLE (articleID,
name) VALUES ('" + articleID + "', '" + articleName +"')", sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        private List<DataRow> GetListHeadsFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM
SIM_PICKLISTHEAD", sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
```

```
            }

            return result;
        }

        private List<DataRow> GetListRowsFromDB(DataRow datarow)
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM SIM_PICKLISTROWS
WHERE pickListNr = " + datarow["pickListNr"], sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
            }

            return result;
        }

        private List<DataRow> GetOrderTypesFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT orderType FROM
SIM_ORDERHEAD", sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
            }

            return result;
        }

        private List<DataRow> GetPickListTypesFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT pickListType FROM
SIM_PICKLISTHEAD", sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
```

```
            {
                result.Add(dr);
            }

            return result;
        }

        private void AddListHeadToDB(string pickListNr, string suffix, string
pickListType, string changedBy)
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_PICKLISTHEAD
(pickListNr, suffix, pickListType, date, messageID, pickListDone, changedBy) VALUES
('" + pickListNr + "', '" + suffix + "', '" + pickListType + "', '" + DateTime.Now
+ "', 0, 0, '" + changedBy + "')", sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        private void AddNewPickListRowToDB(string pickListNr, string pickListRow,
object articleNr, string batch, string storageLocationID, string orderedQty, string
confirmedQty, int rowDone, string messageID)
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_PICKLISTROWS
(pickListNr, pickListRow, articleID, batch, storageLocationID, orderedQty,
confirmedQty, rowDone, messageID) VALUES ('" + pickListNr + "','" + pickListRow +
"','" + articleNr + "','" + batch + "','" + storageLocationID + "','" + orderedQty
+ "','" + confirmedQty + "','" + rowDone + "','" + messageID + "')",
sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        private void AddNewStorageLocationToDB(string storageLocID, string
storageLocName)
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_LOCATION
(storageLocationID, storageName) VALUES (" + storageLocID + ", '" + storageLocName
+ "')", sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        public void AddNewMessageToDb(string newMessageID, string newMessageBody)
```

```
        {
            sqlConnection.Open();
            SqlCommand sc = new SqlCommand("INSERT INTO SIM_MESSAGE (messageID,
messageHead, messageBody) VALUES (" + newMessageID + ", ' ','" + newMessageBody +
"')", sqlConnection);
            sc.ExecuteNonQuery();
            sqlConnection.Close();
        }

        private List<DataRow> GetMessagesFromDB()
        {
            List<DataRow> result = new List<DataRow>();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM SIM_MESSAGE",
sqlConnection);
            DataSet ds = new DataSet();

            da.Fill(ds, "Collection");
            foreach (DataRow dr in ds.Tables["Collection"].Rows)
            {
                result.Add(dr);
            }

            return result;
        }

        //Privates
        private SqlConnection sqlConnection;
        private static Mapper instance = null;
    }
}
```