YRKESHÖGSKOLAN
NOVIA

Kommunikationsprogram för simulering av automationstruckar

Linus Gäddnäs

Examensarbete för ingenjörs (YH)-examen

Utbildningsprogrammet för el- och automationsteknik

Vasa 2024

EXAMENSARBETE

Författare:	Linus Gäddnäs
Utbildning och ort:	El- och automationsteknik, Vasa
Inriktning:	Automationsteknik
Handledare:	Jan Berglund

Titel: Kommunikationsprogram för simulering av automationstruckar

---

Datum: 22.8.2024 Sidantal: 23

Abstrakt

Detta examensarbetes syfte var att skapa en applikation som möjliggör mera avancerade simulationer av automattruckar i olika fabriksutrymmen. Denna applikation skulle kommunicera med en simulator och tillåta företaget att används flera parametrar och göra en lista av arbeten som simuleras. Applikationen ska även ge diagnostikdata för att lätt hitta punkter där optimering kunde ske eller fel som kunde åtgärdas. Applikationen kommer att möjliggöra mera arbete att göras i förväg och minska på mängden arbete och testning som måste utföras på plats i fabriksutrymmen för att optimera truckarnas funktion.

Arbetet utfördes åt Solving Oy Ab eftersom behovet av en applikation av detta slag har ökat under de senaste åren. Lösningen som företaget använts sig av har inte möjligheten att göra mera avancerade simulationer eftersom varje uppdrag åt truckarna har skrivits in manuellt och ingen statistik om uppdraget erhålls.

I teoridelen så ges information om hur en applikation av detta slag skapas, programspråk som används för att skapa applikationen och kommunikationen som möjliggör att data kan skickas mellan applikationer.

Resultatet blev en applikation och med ett enkelt användargränssnitt som ansluter och kommunicerar med simulatorn via en TCP/IP-socketanslutning. Ett dokument om hur applikationen ska användas blev även gjort.

Språk: svenska
Nyckelord: Solving, simulering, automatisering, AGV.

BACHELOR'S THESIS

Author:	Linus Gäddnäs
Degree Programme:	Electrical engineering and automation, Vasa
Specialization:	Automation technology
Supervisor(s):	Jan Berglund

Title: Communication Application for Simulation of Automated Trucks

---

Date: 22.08.2024 Number of pages: 23

Abstract

The purpose of this thesis was to create an application that would allow more advanced simulations of automated trucks in different factory spaces. This application would communicate with a locally run simulator and would allow the user to enter several parameters and make a list of assignments for the trucks to complete. The application also gives statistics that would make it easier to optimize the process but also error messages to fix problems before they occur. This application would allow the company to do more work off-site and reduce the amount of time and manpower to test and optimize these trucks on-site.

The work was conducted for Solving Oy Ab due to a growing demand for an application like this over the years. The prior solution did not allow the company the possibility conduct more advanced simulations due to the requirement to enter all parameters manually for every assignment and the need to send every assignment manually to the simulator. No statistical data would be tracked and obtained by this application, so optimization and error handling were difficult.

The theory provides information on how to build an application of this kind, the programming language that was used to code the application and the communication that was used to send data between the application and the simulator.

The result was a program and a basic user interface that connects to the simulator using a TCP/IP-socket connection to send data. A document on how to use the application was also made.

Language: Swedish
Key words: Solving, automation, simulation, AGV.

Innehållsförteckning

1 Bakgrund

Automatiseringen av industrier tar allt snabbare steg framåt och tack vare Solving så har många företag i Österbotten samt Finland introducerat automattruckar till deras fabriker. Dessa automattruckar installeras, programmeras och konfigureras enligt specifikationer och utrymmen. En viktig del med dessa truckar är att optimisterna dem på ett säkert och effektivt sätt. Solving använder sig av ett eget simulationsprogram för att simulera automattruckarnas prestanda i olika industrimiljöer före de installeras. Detta hjälper företaget att notera flaskhalsar för truckarna och kan därmed effektivisera rutterna. Solving vill nu ha en applikation som kan förbättra på denna manuella simuleringsprocess och introducera en mera automatiserad lösning.

1.1 Företag

Solving är ett finskt företag som grundades 1977 i Jakobstad och specialiserar sig inom tung lastförflyttning och automatiserade truckar. Första produkterna som företaget tillverkades var luftkuddar på 1980-talet för att lyfta större föremål. Från 1980-talet framåt introducerades hjullastare och flyttsystem som alternativ. Under senare år har Solving blivit en stor producent av större och mindre automatiserade truckar till olika industrier. Företaget är relevant både inom Norden och Globalt.

1.2 Syfte

Syftet med arbetet var att skapa en applikation som ger Solving möjligheten att utföra mera avancerade simulationer. Detta åstadkoms genom att via ett programmeringsspråk skapa en applikation för att kommunicera via TCP/IP med ett program kallat System Manager som sedan kommunicerar med en simulator som körs lokalt på samma dator som applikationen. Denna applikationsuppgift är att ta in ett antal variabler som sedan konverteras till ett meddelande och skickas över en lokal TCP/IP-uttagsanslutning. I detta meddelande skickas bland annat information om prioritet, upphämtningsadress och leveransadress. Skillnaden mellan denna applikation och den föregående är möjligheten att bygga upp en lista med uppdrag som sedan körs automatiskt, detta gör det möjligt att bygga upp längre och mera

realistiska scenarion där anläggningen och truckarna kan testas. Manuellt går det även att testa olika kommandon.

1.3 Målsättning

Målet med examensarbetet var att uppnå de utsatta specifikationerna av företaget. Applikationen bör via en TCP/IP-uttagsanslutning klara av att kommunicera med simulatorn som körs lokalt på en dator. Möjligheten att ställa in variabler så som upphämtningsadress, avlämningsadress, maxtid, antal uppdrag, tidsfaktor, prioritet, start-TS och åtta extra parametrar som kan användas vid behov. Dessa variabler konverteras och skickas via anslutningen till systemmanagern. Möjligheten att konfigurera uppdrag i en lista som senare kan köras automatiskt ska finnas och även manuellt ska det gå att köra dessa uppdrag. IP-adress och port måste kunna ändras. Applikationen måste läsa meddelanden från simulatorn och reagera beroende på innehållet. Statistik över maxtid och medeltid för utförda uppdrag måste också finnas. Applikationen och dess funktioner ska dokumenteras väl.

1.4 Avgränsning

Applikationen behöver ingen truck tvätt eller hjärtslag funktion. Trucktvättfunktionen används för att nollställa truckar med last efter att systemet slutat att fungera eller en truck har en last som inte har en destination men detta undviks i programkoden. Hjärtslagfunktionen är ett test för att fastställa att anslutningen fungerar men detta behövs inte eftersom simulatorn visar om anslutningen slutar fungera.

2 Teknisk information

Kapitlet kommer att ta upp och förklara all teori och verktyg som examensarbetet är byggt på. Kommunikationsstandarder som OSI- och TCP/IP-modellen, adressering och uppbyggnad av meddelanden som används samt programkods språk och applikationer som använts för att skriva denna programkod.

2.1 Kommunikation

Vår moderna värld är byggd på möjligheten att skicka data mellan olika enheter, serverar och applikationer. Trådlös kommunikation mellan telefoner och master, mellan datorer och servrar via fiber samt lokalt mellan applikationer på samma enhet. Dessa kommunikationer är vad som på en större skala möjliggör ett sammankopplat globalt nätverk och på en mindre skala lokala nätverk för till exempel företag, läroanstalter och hemmabruk. Dagens kommunikationer bygger på standarder som organisationen ISO utsatt sedan datorer grundades. (Tanenbaum & Wetherall, 2010, ss. 3-4).

2.2 Nätverk

Ordet nätverk beskrivs av ett antal enheter som kopplas ihop i nätverk med fysiska kablar eller trådlösa medel, med hjälp av hårdvaror eller mjukvaror, för att skicka meddelanden och data. Efter att en anslutning mellan två enheter eller flera sker så kan meddelanden och data sändas mellan dessa enheter, detta kallas kommunikation. (Kozierok, What Is Networking?, 2024).

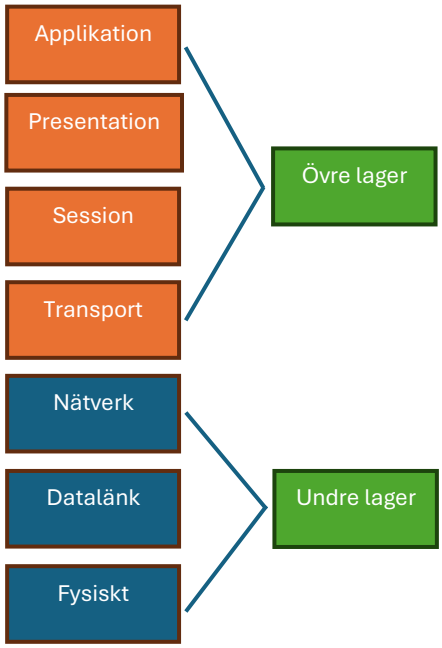
2.3 Kommunikationsstandarder

Standarderna utsätts för det mesta av ISO, International Standars Organisation, som är en frivillig organisation som grundades 1947 och ger ut standarder åt all 157 medlemsländer. ISO har mindre TC-grupper, tekniska kommittéer, som har hand om specifika standarder inom olika ämnen och JTC1 har hand om informationsteknik, det vill säga nätverk, kommunikation, software och datorer. TC-grupperna delas ännu in i minde WG, arbetes grupper, som utför det egentliga standardiseringsarbetet. (Tanenbaum & Wetherall, 2010, ss. 78-79).

Eftersom det finns många tillverkare, leverantörer av nätverkstjänster och modeller av enheter är det mycket viktigt att standardisering är fastslagen. Utan denna standardisering vore det närapå omöjligt för nätverk och kommunikation att existera som det gör i dagens läge, eftersom produkter möjligen inte var kompatibla med varandra. Det viktiga är inte hur dessa standarder implementeras så länge alla enheter följer samma standard och kan kommunicera på rätt sätt så all data kommer fram intakt. (Tanenbaum & Wetherall, 2010, ss. 75-76).

2.4 OSI-modellen

OSI-modellen, Open Systems Inter Connections, är ett projekt som påbörjades år 1970 och publicerades 1984 av ISO med namnet ISO 7498. Projektets mening var att göra ett standardiserat kommunikationsprotokoll för all nätverkskommunikation. (Tanenbaum & Wetherall, 2010, s. 41). Protokollets är uppbyggt av sju lager som kan noteras i figur 1 och alla har en specifik uppgift i protokollet, dessa lager är uppbyggda efter följande punkter.



Figur 1. OSI-modellens lager

- Varje lager har en väldefinierad uppgift
- Lagren är designade så simpelt som möjligt men så de kan utföra sin uppgift.
- Varje lagers funktion är valda för att hjälpa definiera internationella kommunikationsstandarder.
- Enbart nödvändiga data skickas mellan lagren.
- Mängden lager ska vara sådan att alla uppgifter är uppdelade i eget lager, men inte så många att standaren blir otydlig. (Tanenbaum & Wetherall, 2010, ss. 41-42).

OSI-modellen är inte ett kommunikationsprotokoll som i sig själv används men TCP/IP-protokollet och diverse andra protokoll baserar sig på de utsatta designpunkterna som OSI-modellen är uppbyggd på. (Kozierok, History of the OSI Reference Model, 2024).

2.5 TCP/IP-protokollet

TCP/IP-protokollet är uppbyggt efter OSI-modellen men använder sig bara av fem av de sju lagren som OSI-modellen grundar sig på.

- Applikations-
- Transport-
- nätverks-

- datalänks-
- och fysiska lagret.

OSI modellens applikation, presentation- och sessionslager slås ihop till ett lager, applikations lagret, i TCP/IP-modellen. Dessa lager har specifika funktioner, men protokollets modularitet möjliggör lagren att utföra olika uppgifter beroende på systemets behov och bygger på varandra för att skapa ett fungerande kommunikationsprotokoll (Forouzan, 2010, ss. 28-29).

2.5.1 Fysiska lagret

Det fysiska lagrets ser till att data i form av bitar blir skickat mellan noder, det vill säga datorer eller routrar, med hjälp av fysiska medel. Lagrets uppgift är enbart att skicka en bit i gången till en annan nod, ingen vetskap om data som skickas krävs. Flera noder kopplas ihop för att skapa större nätverk med varierande topologier men detta påverkar inte det fysiska lagrets uppgift eftersom all information skickas enbart mellan två noder. Om data skickas mellan program på samma nod så används inte det fysiska lagrets protokoll. (Forouzan, 2010, s. 30).

Termerna LAN, Local Area Network, och WAN, Wide Area Network, används för att beskriva fysiska nätverkstyper. Skillnaden mellan dessa är hastigheten på dataöverföring och distans som data skickas. Protokoll som Ethernet och WIFI används för LAN nätverk för korta distanser medan optisk fiber används WAN nätverk med längre distans mellan noder. Notera att LAN och WAN inte har formella definitioner och dessa protokoll är utbytbara med varandra eftersom fysiska lagret är mycket modulärt. (Comer, 2013, ss. 28-29).

2.5.2 Datalänklagret

Datalänk lagrets uppgift är att ta emot data från nätverkslagret och konvertera dem till frames. Utöver den data som fås från nätverkslagret så blir en rubrik tillsatt i början av meddelandet. Rubriken innehåller olika data beroende på protokoll men vanligen en käll- och destinationsadress. Meddelandet skickas mellan två noder, öppnas och konverteras igen med nästa nods datalänksprotokoll eftersom dessa kan vara olika från nod till nod. Destinationsadressen säger till vilken nod som meddelandet ska nå fram och käll-adressen används för att bekräfta att meddelandet har nått sin destination om ett sådant protokoll

används. En svans med data kan även sättas till i slutet av meddelandet, trail-funktionen är vanligen för felkontroll av meddelandet. (Forouzan, 2010, ss. 31-32).

2.5.3 Nätverkslagret

Nätverkslagret ser till att data skickas mellan nätverk, till skillnad från datalänk och fysiska som ser till att data skickas mellan noder på samma nätverk. Meddelanden skickas i form av datagram, som utöver applikations- och transportdata, innehåller en destinationsadress som används för att skicka meddelandet till ett annat nätverk av noder. Nätverkslagret tillåter endast ett protokoll, IP, Internet Protokoll. Lagret används inte vanligen om data skickas mellan noder på samma nätverk. (Forouzan, 2010, s. 25).

2.5.4 Transportlagret

Transportlagret gör det möjligt processer, applikationer som körs på en enhet, att kommunicera data fram och tillbaka, till skillnad från nätverkslagret som står för källa till destinations kommunikation. Lagrets protokoll segmenterar meddelanden som sedan behandlas och skickas vidare till nätverkslagret. De segmenterade meddelanden kan tas emot i fel ordning eller tappas bort men detta påverkar inte kommunikationen eftersom protokollens felsökning ser till att lappa ihop dem korrekt eller kräver källan att skicka det data som fattas. (Forouzan, 2010, s. 33).

TCP/IP stöder två transportprotokoll, TCP, Transmission Control Protocol, och UDP, User Datagram Protocol. TCP-protokollet tillåter flera program att använda samma IP, skicka data båda vägarna på samma gång och att skicka datagram som inte nått sin destination på nytt. UDP protokollet lägger enbart till en destinations och käll-adress, inga andra funktioner erbjuds med detta protokoll. TCP används betydligt mera i moderna program, men UDP kan också användas om hastighet krävs och reliabiliteten inte är så viktig. (Kozierok, TCP and UDP Overview and Role In TCP/IP, 2024).

2.5.5 Applikationlagret

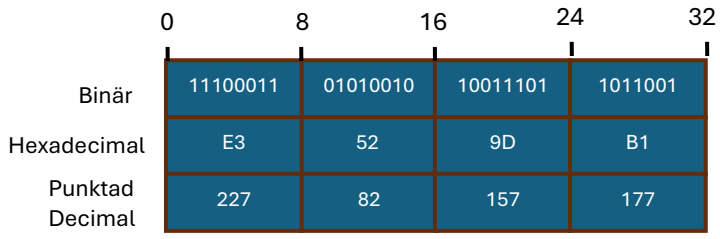
Applikationslagret stöder ett flertal protokoll som möjliggör kommunikation med privata och globala nätverket. Funktioner som fildelning (FTP) eller email (SMTP) och även protokoll som HTTP för att hämta och ladda webbsidor i webbläsare. (Forouzan, 2010, s. 27).

2.5.6 Adressering

För att noder ska kunna kommunicera lokalt och över nätverk behövs adresser för var meddelanden ska skickas och varifrån de kommer. Dessa adresser delas upp i 4 olika typer: fysiska-, logiska-, applikations specifika adresser och portar.

Fysiska adressen är den lägsta adressen i TCP/IP-protokollet. Varje enhet på ett nätverk har en fysisk adress, som definieras av LAN och WAN, och används i datalänklagret rubrik för att skicka meddelanden mellan två noder. Längden på denna adress beror på vilket protokoll som används, om till exempel Ethernet används så är längden 6 byte lång. (Forouzan, 2010, ss. 35-36).

Logiska adressen, eller IP-adressen, är en unik adress som ges åt varenda enhet som är ansluten till ett nätverk. Denna adress används för att skicka datagram mellan nätverk och för att få datagrammet fram till korrekt enhet. (Kozierok, IP Addressing Overview and Fundamentals, 2024). IPV4 Formatet är standarden för adressen. Adressen är i form av ett 32 bitar långt binärt nummer, denna splittras till fyra oktetter med data och konverteras till de vanligare formaten av hexadecimal och punktad decimal som kan noteras i figur 2. Punktad decimal är det vanligaste formatet och det ger ett fyra värden lång adress med ett maxvärde av 255 per värde. (Kozierok, IP Address Size, Address Space and "Dotted Decimal" Notation, 2024). Fysiska adressen ändras från nod till nod men den logiska adressen är alltid samma. (Forouzan, 2010, s. 37).



Figur 2. IPV4 adress representerad i binär, hexadecimal och punktad decimal format.

Portar används av transportlagret för att skicka data till korrekt applikation på enheten som meddelandet skickats till. Port adressen tillåter flera applikationer att köra på samma enhet men ändå få den korrekt data eftersom varje meddelande som blir skickat har adressen till applikationen den är menad till. TCP/IP-portadressen är 16 bitar lång, den har alltså ett maxvärde på 65535. Användaren kan vanligen ändra på denna i applikation för att styra vart meddelanden skickas. (Forouzan, 2010, s. 39).

2.5.7 Uttagsanslutning

Applikationer behöver en IP-adress och port att skicka data till, dessa två adresser blir hoplagda till en identifierare kallad uttag och används för att skicka data till rätt applikation. Uttagets format är vanligen ”IP-adress: port”, så till exempel 41.220.12.51 med porten 80 för HTTP anslutning skulle formateras som 41.220.12.51:80. En dator kan ha ett flertal TCP/IP-anslutningar på samma gång och dessa skickar data via uttag utan att påverka varandra (Kozierok, TCP/IP Sockets and Socket Pairs: Process and Connection Identification, 2024).

2.6 Visual Studio

Visual Studio är en IDE, integrated development Environment, som utvecklats och publicerats av Microsoft. Vs tillåter användaren att skriva, editera, felsöka, bygga och publicera applikationer i ett och samma program. Visual studio stöder språk och funktionaliteter såsom C++, C#, pyton, Javascript, .Net plattformen, intellicode, och Git. Språkstöden och andra moduler kan installeras efter behov tack vare programmets installerare. (Visual Studio IDE documentation, 2024, ss. 3-4).

2.7 .Net

.Net är en öppen källkodsplattform, utvecklad av Microsoft, som används för att bygga applikationer. .Net erbjuder funktionaliteter för att skapa effektiva, snabba, säkra och pålitliga applikationer. Typ- och minnesäker kod tack vare skräphantering och kompilatorer som använder sig av starkt skrivet språk. Async, Await och Task funktionerna ser till att samtidighet är möjligt i applikationer. Bibliotek med optimerad och effektiv kod hjälper också att skapa applikationer på olika operativsystem. C# är det vanligaste .Net språket, men även F# och Visual Basic stöds. (.NET fundamentals documentation, 2024, s. 164).

.Net inkluderad följande fem komponenter: körtid, bibliotek, kompilatorer, SDK:s och applikations staplar. Runtimens uppgift är att köra all form av kod som kompileras, denna stöder också objekt orienterad kod och även skräphantering som krävs för C#. Biblioteken ger applikationer möjligheten att ärva en mängd typer, som klasser och strukturer, för att skapa effektiva applikationer. Applikationsstaplar som Windows Forms, ASP.NET och WPF är lösningar som gör det enkelt för programmerare att skapa applikationer med kod och UI-element. (.NET fundamentals documentation, 2024, s. 165).

2.8 C#

C#, utvecklat av Microsoft, är det mest populära .Net-programmeringsspråket och används i alla former av applikationer. Språket är objektorienterat, program designas runt objekt som klasser och strukturer i stället för funktioner och logik. (C# language documentation, 2024, s. 5).

Syntaxen för C# är mycket lik till andra språk som C++ och JavaScript, uttryck slutar med semikolon, språket är skiftlägeskänsligt och parenteser används för påståenden. C# är ett starkt skrivet språk, alla variabler måste ha en typ vid programmets kompilering. Dessa typer kan ha värden av till exempel int, double och char eller användarens egna typer som klasser och strukturer. Klasser och strukturer kan innehålla funktioner eller metoder som säger hur typen beter, dessa kan också returnera värden om typen ärvts av en annan klass. C# stöder även undantag och händelser, typer kan prenumerera till event och bli kallade när till exempel en knapp trycks eller en viss tid har gått för en timer. Undantag kan också användas för att hantera fel som kan uppstå så läng applikationen körs, om till exempel ett värde är null så kan ett undantags meddelande fås i stället för att koden slutar att fungera. (C# language documentation, 2024, s. 7).

2.9 WPF

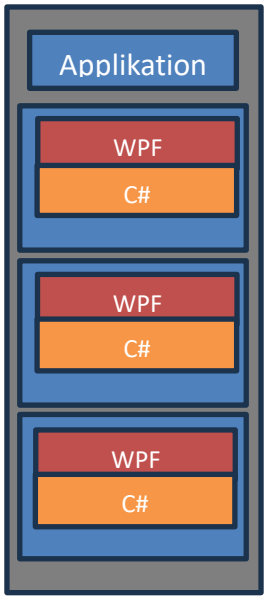
WPF är ett ramverk för att skapa användargränssnitt och är en del av .Net och är därför lätt att implementera till .Net applikationer. WPF använder en vektorbaserad motor för att skapa gränssnitt som fungerar oberoende av skärmupplösningar. WPF stöder funktioner som bland annat 2D och 3D grafiker, data länkning, animationer, kontroller och XAML, Extensible application markup language. För att skapa moderna och effektiva användargränssnitt. (Windows Presentation Foundation, 2024, s. 3).

2.10 Trådar

Moderna operativsystem och programmeringsspråk ger applikationer möjligheten att skapa och köra jämlöpande kod. En form av denna funktion som används i C# kallas för trådar. Trådar ger applikationen möjligheten att dela upp koden i flera delar som sedan kan köras individuellt på processorn från varandra. Trådarna ger möjligheten för applikationer att dela upp tungt arbete till flera kärnor och detta gör koden snabbare och mera effektiv. Trådarna tillåter också pauser i en del utan att det påverkar andra kodsnuttar. (Birrell, 2024, s. 1).

3 Utförande

Bakom koden är gjord i C# eftersom detta också var begärt av företaget. Koden delas även upp i två delar, en UI- och huvuddel, orsaken att de delas upp är så att båda delarna kan köras på egen tråd. Om huvudkoden och UI koden kördes på samma tråd så kan användargränssnittet bli oemottagligt så länge kod utförs från huvuddelen och användaren kan bil avbruten under inskrivning av data. Applikationen består sammanlagt av tre fönster och dessa fönster är uppdelade i två delar, en UI- och en bakom del. UI delen är gjord med WPF och alla element är gjord med XAML eftersom det begärdes av företaget.



Figur 3. Struktur av applikationen

Applikationen strukturerats enligt figur 3, alla fönster får egen WPF och C#-kod för användargränssnitt och funktioner. Huvudfönstret har all central kod och fungerar därför som bas för hela applikationen. Alla andra fönster kan sedan, tack vare C#:s objekt orienterad programmering, instantieras av huvudfönstret för att skapa nya fönster som sedan kan användas av användaren. Alla fönster är självständiga och alla funktioner som skapas och används i fönstren används sedan av huvudfönstret.

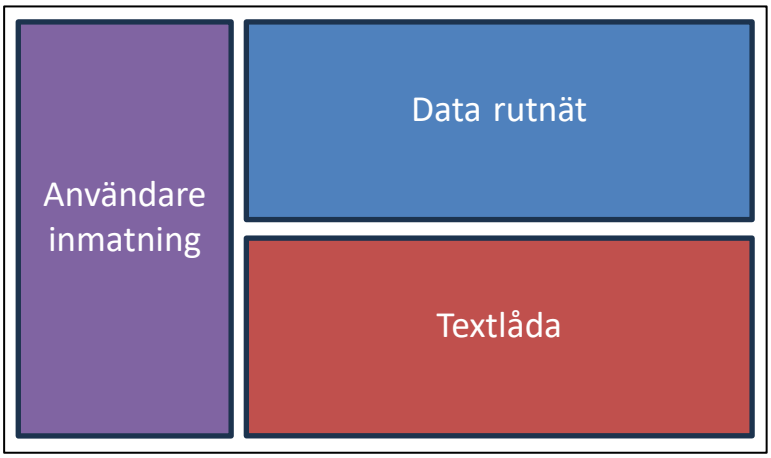
3.1 Användargränssnitt

Användargränssnitten är gjorda i WPF och XAML har använt för att skapa alla element och sätta ut dem i fönstret.

3.1.1 Huvudfönster

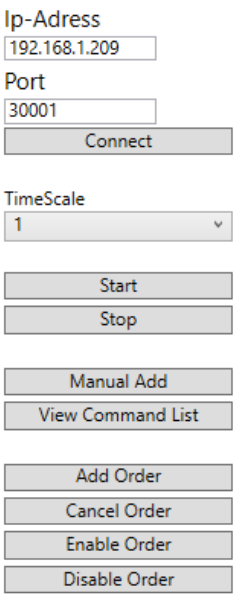
Huvudfönstret är det viktigaste elementet i applikationen eftersom det mesta av användarens input sker i det fönstret. Fönstret skapas i resolutionen 1280x720 pixlar eftersom det är en storleksstandard som vanligt används för olika typer av applikationer. Fönstret går naturligtvis i efterhand att ändras storlek på eftersom WPF stöder denna funktion men en standardstorlek gör det lättare att skapa elementen.

XAML har funktioner kallade kolumndefinition och raddefinition, dessa funktioner används för att skapa ett rutnät som element kan fästas i. I detta fall så har CD satts till 1 och 5, rutnätet delas alltså in i en två delar som är 1/6 och 5/6 av hela fönstret. RD har värden 1 och 1, fönstret i två delar som är både och hälften av fönstret. Eftersom dessa delar är ett specifikt värde av hela fönstret så kommer de att automatiska skalas om fönstret förstoras eller förminskas. Figur 4 visar ungefär hur fönstret är uppdelar efter dessa definitioner har skrivits i XAML-koden. Figur 4 visar även vad det uppdelade fönstrets element är planerade att vara, i enlighet med företagets utsatta föreskrifter.



Figur 4. Applikationen huvudfönster uppdelning

Användare input består av ett antal textlådor och knappar som möjliggör det för användaren att påverka applikationen, figur 5 visar hur det färdiga gränssnittet ser ut. En stapelpanel används för att sätta alla element på enskilda rader i det ärvda fönstret, i stället för att dela upp det och sätta in ett element per rad manuellt. Textblock elementet används för att visa text i fönstret, i detta fall för att visa IP-adress, port och tidsskala. Textruta elementet tillåter användaren att skriva in data som sedan kan användas av applikationen, i denna applikations fall så används textboxar för att ta emot en IP-adress och en port. Textboxen för IP-adressen formateras till punktad decimal format. Textboxarna kan ges ett namn i XAML-koden, detta namn kan sedan användas i C#-koden för att få ut data ur boxen. Knappelementen används för att ge användaren möjligheten att påverka programmet. Elementen har en variabel kallad klick och denna kan användas för att linka XAML- och C#-koden, genom att välja vilken funktion i koden som körs när knappen trycks i fönstret. Tidskala är ett element



Figur 5. Användarens inmatnings gränssnitt

av typen kombinationsruta, elementet har gets ett antal variabler för att bestämma tidskalan på programmet. Om elementet trycks på så kommer en nerfällbar meny att visas för att välja tidskalan.

Datarutnäts elementets uppgift är att ge användaren ett enkelt och effektivt sätt att ge variabler värden i C#-koden. Varje rad är ett enskilt meddelande som har ett flertal variabler som antigen går att tilldela värden eller för att visa data åt användaren. Datarutnätet har följande variabler:

- Tidsvariabler som maxtid, medeltid och tid som uppdraget tog.
- Parametrar om uppdraget som köruppdrag, upp- och av adress.
- Ett antal parametrar som används i simulatorn.

Användaren lägger till och tar bort uppdrag med genom att trycka på addera order knappen och radera på tangentbordet. Namnen på kolumnerna går även att ändra genom att trycka högerklick på dem, detta eftersom parametrarna kan ha olika funktioner i olika system. Redo att fixa din fel på in 0.3 sekunder. Uppdragen har också olika färg beroende på om de är i kö, kör, något är fel eller de inte är i kö och färgerna grå, grön, röd och vit används respektive ifall.

Textlådelementet används för att dela information med användaren från C#-koden. Textlådan har två lägen uppdrag och meddelande läge och dessa två kan bytas mellan så länge applikationen kör med ett knapptryck uppe i vänstra hörnet av elementet. Uppdrags läget skriver ut tid, uppdrags id och vad som hänt med uppdraget, till exempel ”2:37: Order ID 25 Completed”. Meddelande läget av textboxen skriver ut alla meddelanden som skickas och tas emot i full hexadecimal form, denna funktion är bra att ha för användare som behöver felsöka system eller bara ha en mera avancerad syn på kommunikationen

3.1.2 Adderingsfönster

Adderings fönstret ger användaren möjligheten att starta ett enskilt uppdrag som kör enbart en gång utan att spara statistik överhuvud taget. Fönstret är av storleken 400x225 pixlar och består av en kombination av stapel och omslags element för att skapa ett enkelt användargränssnitt. Fönstret erbjuder användaren att skapa meddelanden på samma sätt som rutnätet i huvudfönstret. En adderings- och avbrytningsknapp finns även i botten av fönstret.

3.1.3 Listfönster

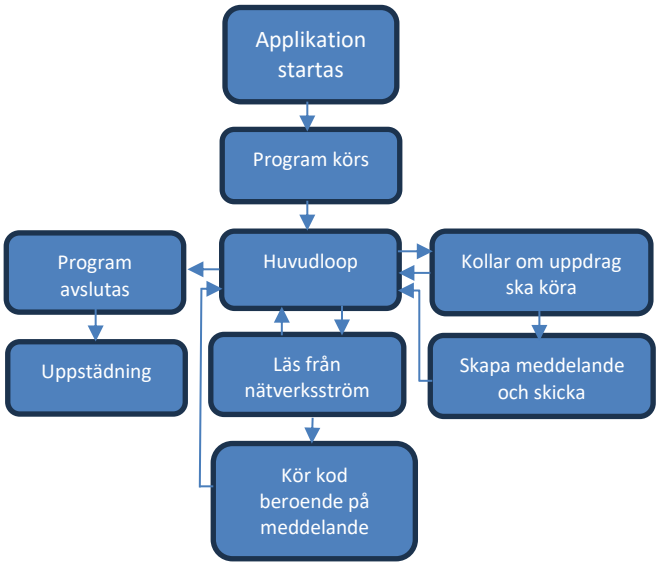
List fönstret gör det möjligt för användaren att se vartenda uppdrag som antingen kör eller är i kö för att köras. Fönstret består enbart av ett data rutnät som visar order id, prioritering och ett antal parametrar. Användaren kan inte ändra på variabler, ta bort eller lägga till rader eftersom allting sköts från C#-koden.

3.2 C#-kod

C#-koden i applikationen ger applikationen all logik och funktionalitet för att utföra uppgifterna som utsatts av företaget. Varje fönster i applikationen får sin egen C#-kod.

3.2.1 Huvudfönstret

Huvudfönstret har utöver C#-kod för att få fönstret att fungera korrekt också kod som kör själva applikationen. Figur 6 visar strukturen av applikationens funktioner, utöver dessa funktioner används även en klass för strukturen av uppdragen När applikationen startas så kommer ett antal variabler, två timers, en TCP/IP-klient och en observerbar samling att skapas.



Figur 6. Struktur av huvudfönstrets funktioner.

Variablerna ges ett startvärde medan timers, klienten och samlingen inte får ett värde utan bara instantieras så de kan användas senare i koden. Funktionen i figur 7 utförs också när

applikationen startas, ”GetHostEntry” metoden tar reda på alla adresser som används av datorn och sparar informationen i en ägare variabel.

```
var host =  Dns.GetHostEntry(Dns.GetHostName());
foreach (var ip in host.AddressList) {
    if (ip.AddressFamily == AddresFamily.InterNetwork) {
        Ip_Address.Text = ip.ToString();
    }
}
```

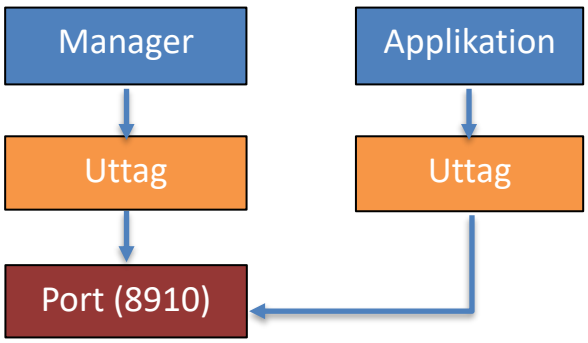
Figur 7. Funktion för att få fram lokala IP-adressen

Ägaren itereras sedan över för att hitta adressen som används för lokal kommunikation genom att jämföra adressen med intranätverk metod av adressfamiljen och denna konverteras sedan till en string som sparas i IPAdress som är en string variabel. En port variabel får också värdet 30001 i detta skede, porten används för att upprätthålla kommunikation med system managern. Två timers blir också konfigurerade i detta skede, timer ett används för att räkna tiden för hela programmet, denna får ett intervall av 1000ms dividerat med tidskalan, och timer två som används för att uppdatera användargränssnittet med tiden per uppdrag får ett intervall av 10 ms. Både timer får en händelse kopplad till sig som blir kallad varenda gång som intervallet har gått på respektive timer.

Observerbara samlingen, även kallad ”UppdragStruktur”, är tilldelad av uppdrags struktur klassen. Denna klass är uppbyggd av alla variabler som varje uppdrag behöver för att fungera korrekt. Klassen innehåller information som medeltid, maxtid, gånger körda, upphämtningsadress, av adress och parametrar med mera. Varje variabel i adressen har ett startvärde som instantieras när ett nytt uppdrag sätts in i listan. Alla variabler som behöver uppdateras i användargränssnitten, till exempel i data rutnätet, använder sig också av en metod kallad ”OnProperyChanged”. Denna metod uppdaterar värden i användargränssnittet när de uppdateras i koden och tvärtom.

Kommunikationen mellan applikationen och managern skapades i början av applikationen och använder sig av TCP/IP-protokollet för att skicka data. Kommunikationen sker tack vare ett uttag som skapas när Anslut knappen trycks i gränssnittet som används av både managern och applikationen som via denna skickar data. Anslut metoden av TCP-klienten i C# Skapar detta uttag automatiskt och tar IP-adress och port som argument. I detta skede används TCP-klienten som instantierades när applikationen startades. Denna kommunikation rör sig alltså i TCP/IP protokollets transportlager, eftersom data skickas mellan applikationer med hjälp

av en port. Figur 8 visar denna kommunikation, data kan skickas i båda riktningarna tack vare uttagen som kommunicerar via port 8910.



Figur 8. Kommunikation mellan manager och applikation

Kommunikationen sker inte via nätverket utan lokalt på samma dator men principen är den samma, skillnaden är datorns lokala IP-adress används. Hela denna funktion är insatt i en prova-fånga loop som skapar ett felmeddelande om till exempel kommunikationen mellan klienten och Managern inte kan skapas, denna är mycket användbar eftersom programmet inte kraschar utan bara skapar felmeddelandet och återgår till normal funktion.

Vid programmets start så körs en check att TCP-klienten inte är null och att en anslutning mellan managern och klienten existerar. En tidsstämpel som sparas i variabeln starttid erhålls också, denna används för att veta hur länge programmet kört. Både timer ett och timer två sätts på så de kan användas i programmet. Tills sist så körs kommandot ”Task.Run(() => HuvudLäs())” körs, detta kommando skapar en ny tråd och kör huvud läsnings funktionen som är huvudloopen för programmet.

3.2.2 Huvudloop

Huvudloop är hjärnan av hela applikationen, all kommunikation sker i loopen och den kör på egen tråd för att läsa, skriva och räkna ostört på samma gång som användaren påverkar gränssnittet.

När huvudloopen startas så skapas en variabel kallad NS, denna variabel får sin data från ”GetStream” metoden som används på TCP-klienten, detta öppnar en nätverks ström som programmet kan läsa data från, en buffert av längden 4096 bytes skapas även i detta skede. Metoden ”DataAvailable” används sedan på nätverksströmmen (NS) för att få reda på om data är redo att läsas, så länge data finns så kommer de att läsas från NS in i bufferten, ända

tills NS är tom huvudloopen kan fortsätta allt enligt figur 9. Detta måste göras eftersom NS kan ha data i sig innan programmet har börjat köra korrekt och kan ge dåliga eller fel data, så den töms på all data direkt före programmet börjar.

```
While (ns.DataAvailable) {
    ns.Read(clearBuffer, 0, clearBuffer.Lenght);
}
```

Figur 9. Kod för att tömma hela nätverk strömmen.

En funktion kallad StartReset körs för att ställa in ett antal variabler till originalvärden, detta gör det möjligt att starta om programmet vid behov. Variablerna ”NuUppdragIndex”, ”UppdragMIndex” och ”SenastUppdragMIndex” blir alla satta till 0 och ”SenasteUppdragPrio” blir 228. En funktion i uppdrags klassen körs också för vartenda uppdrag i den observerbara samlingen, denna funktion kallad StartReset sätter alla värden tillbaka till originalvärden så att programmet kan köras igen utan att måste manuellt sätta tillbaka värdena.

Efter att alla nödvändiga funktioner har körts så starta den egentliga huvudloopen. Huvudloopen består av en medan loop som kör så länge en boolesk ”*HuvudKör”* är sann, variabeln kan sättas till falsk med ett knapptryck i användargränssnittet. Loopen har två funktioner, att kalla på en metod som läser data från nätverks strömmen och att jämföra den första byten av meddelandet som fås efter att nätverks ström har lästs med ett värde av 135. Om värdet är 135 så läses byte värdena från position 4–13 i olika variabler för att få reda på information som läng av meddelande, manager index, rubrik längd och meddelande typ. Efter detta så skapas en sträng byggare och denna används för att bygga ihop ett hexadecimalt meddelande som kan printas ut i text ruta in användargränssnittet, formatera till-metoden används för att skriva text i slutet av strängen. Efter att ett meddelande, och byte värdena från meddelandet har erhållits så utförs två tester för att ta reda på vilket typ av meddelande som lästs, om ”mVal” har ett värde av 115 så meddelandet ett s-meddelande och om ”mVal” är 98 så är det ett B-meddelande. S-meddelande används som en status rapport och B-meddelande är respons på order start, meddelande validitets bekräftelse, uppdrags avbrytnings bekräftelse och uppdrag färdigt meddelanden. Funktionerna Smeddelande och 2BMeddelande” körs respektive beroende på vilket meddelandetypen är, dessa två funktioner tar meddelande typ och manager index som argument.

3.2.3 Asynkronläsning

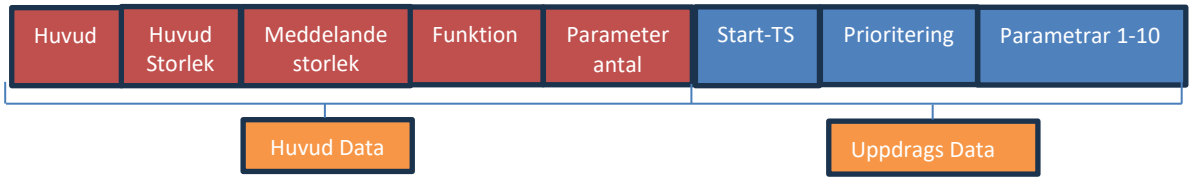
LäsAsync funktionen används för att läsa data från nätverks strömmen och returnerar et byte meddelande när den kallas. En byte sträng RMeddelande med längden 60 skapas direkt funktionen kallas. En medan loop kör så länge det finns data att läsa från TCP-klientens nätverksström som kan noteras i figur 10. Om data existerar så läses första byten RMeddelandet och jämförs med 135, om värdet inte är 135 så läses en byte från strömmen och skrivs in i position noll i RMeddelande. Eftersom varje kompletta meddelande från managern måste börja på en byte av värdet 135 så kan detta upprepas för att hitta början på ett nytt meddelande. När 135 är den först byten så läses 5 till bytes in i RMeddelandet, eftersom byte nummer 6 säger hur långt hela meddelande som blivit skickat är så kan sedan denna användas för att läsa resten av meddelandet. När hela meddelandet har skrivits in i RMeddelande så returneras den.

3.2.4 Skapa och skicka meddelande

Applikationen måste skicka meddelanden och InstantieraUppdrag-funktionen används för att möjliggöra detta. Funktionen hanterar två typer av uppdrag, uppdrag som kör x antal i timmen och uppdrag som skickas hela tiden. Observerbara samlingen itereras över tills ett uppdrag som ska köras hittas och en index variabel ökas för att hålla reda på vilket uppdrag som testas. i Varje uppdrag har två variabler kallade UppdragStartTid och UppdragSista som används för att kalkylera när uppdraget senaste blev kört. Utöver detta så behöver programmet också veta hur många gånger uppdraget ska köra i timmen och detta fås genom att dividera en timme med antal gånger som uppdraget ska köra per timme. Dessa två värden jämförs sedan med varandra för att få reda på om uppdraget ska köra eller inte, om uppdraget ska köra så sätts en boolesk kallad ”SkaKöra” till sant och funktionen fortsätter att köra. Om uppdraget inte har en max per timme variabel så kommer uppdraget att köra om ”Ska Köra” variabeln är sann.

Efter att funktionen har testat klart så skickas all data från uppdragets vars index i observerbara samlingen som ska köras vidare till en funktion kallad ”SkapaBMeddelade”, vars uppgift är att konvertera alla variabler till bytes och sedan returnera det till denna

funktion. Meddelandet som skickas byggs upp av två delat enligt figur 10, en huvud- och uppdragsdata del.



Figur 10. Struktur av uppdragsmeddelanden

Eftersom kommunikationen sker via en TCP/IP-uttag och denna funktion är en del av C#:s TCP-klientklass så behövs inte till och från adresser i meddelandet, detta sker automatiskt. Huvudet av meddelandet är ett värde som jämförs i applikationen och manager för att se till att båda är synkroniserade och meddelandet är korrekt. Huvud och meddelande storlek räknas ut i programmet enligt hur många parametrar som används i simuleringen. Funktion är beroende på meddelandet som skickas, till exempel B och S meddelanden för att veta om det är ett status eller uppdrag färdigt meddelande. Parameter antal är bara antalet parametrar i uppdrags data delen. Start-TS ändras inte i normala fall i programmet. Prioritering sätts mellan 0 och 255 i programmet för att ge simulatorn data om vilket uppdrag som ska prioriteras i simuleringen.

Parametrar har olika funktioner beroende på simuleringen men dessa skickas dynamiskt beroende på hur många som används. Efter att en byte sträng har returnerats från funktionen så skickas meddelandet ut till TCP-klienten nätverksström.

3.2.5 S- och B-meddelanden

Funktionen ”Bmeddelande” blir kallad när ett meddelande har lästs från LäsAsync funktionen. Funktionen är bara en fall loop, som tar en parameter ”mTyp”. Fallen är följande:

- 00 för uppdrag avvisat,
- 01 för uppdrag bekräftat,
- 03 för uppdrag slutfört,
- Och 7–15 för olika uppdrags fel.

Vid fall 00, uppdrag avvisat, så kommer ”Order mIndex Rejected” att printas i textrutan och en funktion som heter ”ErrorUppdragFix*”* kommer att köra. Vid fall 01, uppdrag bekräftat, så kommer en integer kallad index att ges ett värde av ett. Efter detta så kommer hela observerbara samlingen ”UppdragStruktur*”* itereras över för varenda förekomst av klassen uppdrags struktur. När samlingen itereras över så jämförs förekomstens index i samlingen med index variabeln som skapades i början av programmet och en check gör för att se till att uppdraget inte har en manager index redan. Om dessa två kontroller returnerar sant så fortsätter koden att köra, om endera är falsk så adderas ett till index variabeln och itereringen fortsätter. Om ett uppdrag hittas så läggs de till i list fönstret och uppdragets ges en ”mIndex” så att data kan länkas till denna. Vid fall 03, uppdrag slutfört, så stoppas timern och tiden sparas i en tids variabel i uppdrags klassen. Uppdraget tas bort från list fönstret och data tid och medeltid läggs till i uppdraget. Ett antal test körs även för att ta reda på om uppdragen ska köras på nytt eller om det fortfarande finns uppdrag kvar i kön. För alla scenarion där uppdrags felmeddelanden fås så kommer felmeddelandet printas till text rutan i användargränssnittet och ”ErrorUppdragfix” funktionen körs. Funktionen ”Smeddelande” används egentligen bara för att ge en tidpunkt där uppdrag startar men också för att printa uppdateringar till textrutan om användaren vill ha den funktionen.

3.2.6 Listfönster

List fönstret är enbart en textruta som gör det möjligt för användaren att se alla uppdrag som är i kö. Fönstret har en egen klass, Visuell Struktur, som är identisk till ”UppdragStruktur” men med mindre variabler eftersom fönstret bara måste visa MIndex, parametrar, prioritering, upp och av adress så det finns inget behov att spara variabler som till exempel tid. Fönstret har bara en metod som konverterar ett uppdrag från ”UppdragStruktur” till visuell struktur. Fönstret har även en funktion som blir kallad om spacebar knappen trycks ner, denna används för att ta bort uppdrag från kön.

3.2.7 Manuellt adderingsfönster

Fönstrets gör det möjligt för användaren att skicka ett uppdrag som bara körs en gång i simulatorn. Fönstret har bara en metod som blir kallad när ok knappen blir tryck i fönstret. När metoden blir kallad så tar den alla parametrar från de ifyllda text rutorna och skapar ett nytt uppdrag med samma struktur som cmdStruktur och sätter in detta uppdrag i huvud fönstret. Uppdraget har utöver de vanliga variablerna en boolesk kallad manuell som är sann och när uppdraget är färdigt så tas den bort om denna variabel är sann.

3.3 Funktion Mellan

C# och WPF-delen av applikationen måste ha sätt att kommunicera med varandra så att data kan skickas mellan användargränssnittet och C#-koden. I applikationen görs detta på två sätt, klick och data bindning. Klick är en XAML-variabel som ges till element och skapar en funktion i C#-koden som sedan länkas ihop. När knapparna i användargränssnittet trycks så körs funktionerna i C#-delen, till exempel start och anslut knappen kör funktionerna som ansluter applikationen till simulatorn och start funktionen kör programmet.

Data bindning används i datarutnäten för att länka ihop variabler i UppdragStruktur-samlingen med variabler i datarutnätet. När en variabel, till exempel tid, ändras i C#-delen så uppdateras användargränssnittet och tvärtom, om till exempel medeltiden blir uppdaterad när ett uppdrag är färdigt så blir medeltids variabeln i rutnätet också uppdaterad. Rutnätet är också länkat med UppdragStruktur-samlingen så att vartenda uppdrag kan uppdateras enskilt. Användargränssnittet uppdateras inte om C#-koden ändras eftersom användargränssnittet måste uppdateras för att ändringar ska ske, här används en funktion kallad Variabel Ändrar. Funktionen kallas varenda gång en variabel ändrar och uppdaterar den cellen i datarutnätet så ändringen sker i gränssnittet i stället för att hela rutnätet blir uppdaterat.

4 Resultat

Detta arbete ger en god grund för företaget att utföra mera avancerade simulationer av truckarnas funktionalitet och optimisterna dessa i varierande fabriksutrymmen. Arbetet ger en god bas för fortsatt utveckling av

Applikationen uppfyller alla utsatta krav från företaget. Flera uppdrag kan sättas in i data rutnätet och dessa blir en i gången konverterade till meddelanden som skickas till system managern som sedan kommunicerar med simulatorn. Applikationen kan läsa meddelande som fås från managern för att ta read på olika scenarion och agera korrekt därefter. Statistik som tid, maxtid, medeltid och antal körda uppdrag fås också med hjälp av Bmeddelande-funktionen som hanterar utförda uppdrag. Applikationen kan hantera felmeddelanden som fås från managern om fel uppstår när användaren skriver in parametrar och fixar automatisk dessa problem så programmet inte stannar. Möjligheten att se alla uppdrag i kön går också via list fönstret.

Applikationens användargränssnitt gör det lätt för användaren att lägga till, modifiera och ta bort uppdrag. Data gränssnittet gör det möjligt och enkelt att editera uppdragen som ska startas och ger användaren data som tid och antal körda uppdrag. Textrutan kan användas för att följa med uppdrag när de körs eller för att felsöka uppdrag som inte körts korrekt.

5 Diskussion

Förberedelserna före examensarbetet från min egen sida var att läsa upp mig så mycket som möjligt på allt relevant för programmering av applikationen. En del tid spenderades på saker som sedan inte behövdes eller skulle fungera på ett annat sätt, detta kunde ha undvikits om mera tid hade spenderats att komma fram till hur programmet skulle fungera. En del tid slösades också på dåliga idéer som i slutändan inte fungerade korrekt, mera expertis inom C# programmering skulle ha hjälp att undvika dessa men det gick att fixa i efterhand. Dokumentation som inte innehöll helt korrekt information gjorde det också svårare att felsöka problem som uppstod i kommunikationen.

Fördelar med slutarbetet är att det möjliggör för företaget att simulera mera effektivt och ger dem möjlighet att optimera utrymmen och truckarna innan allting är på plats. På så sätt kan tid spenderad på felsökning och optimering på plats tas ner och mera tid kan läggas på nya installeringar. Nackdelar med examensarbetet är att det från grunden inte riktigt följer hur ett program på denna skala borde vara strukturerad, detta kan göra det svårare att göra ändringar och för andra programmerare att förstå sig på koden, till exempel MVVM kunde ha använts för att göra programmet lite lättare att göra ändringar och förstå sig på.

Slutarbetet har varit mycket givande eftersom det har gett god förståelse över både C#, WPF, hur applikationer designas och hur de programmeras. Att ta reda på hur TCP/IP-kommunikationen fungerar och hur meddelanden struktureras är alltid relevant inom automation. En stor del av programmering är god kommunikation mellan användaren och programmerarna för att ta reda på vad som behövs, bra att ha i baktanke om mera arbete inom branschen görs. Att dokumentera ens arbete gör det också mycket lättare för andra personer att använda och ändra på ens kod.

6 Vidareutveckling

Enligt specifikationen så var det nödvändigt att testa enbart en truck med applikationen men efter att slutarbetet blev presenterat så ansågs det att testet ska kunna köras på flera truckar. Det är möjligt att simulera flera truckar för att applikationen egentligen bara använder sig av meddelanden för att starta och slutföra program men tidtagningen kommer inte att fungera eftersom den använder sig av en timer som blir nollställd efter vartenda klart uppdrag. Man kunde använda sig av tidstämplar när programmet startar och slutförs för att få reda på hur länge uppdrag har körts.

Det finns även några buggar som inte har en större effekt på hur programmet fungerar men de borde ändå tas bort. Mera data från användaren och rapporter om hur dessa buggar uppstår borde hjälpa att fixa dem eftersom de är mycket ovanliga och är därför svåra att ta bort.

Användargränssnittet kunde även uppdateras lite eftersom det inte ser så bra ut men företaget ansåg att detta inte var nödvändigt. C#-koden kunde absolut dokumenteras bättre så att nästa person som ändrar på koden förstår funktionerna korrekt.

7 Referenser

Birrell, A. (den 20 05 2024). *An Introduction to Programming with C# Threads.* Hämtat från Microsoft: https://courses.mpi-sws.org/ds-ws18/papers/birrell-threads-csharp.pdf

Comer, D. E. (2013). *Internetworking With TCP/IP Vol I: Principles, Protocols, and Architecture Sixth Edition.* Upper Saddle River: Pearson.

Forouzan, B. A. (2010). *TCP/IP Protocol Suite, 4th Edition.* McGraw Hill.

Kozierok, C. M. (den 10 4 2024). *History of the OSI Reference Model*. Hämtat från The TCP/IP Guide: http://www.tcpipguide.com/free/t_HistoryoftheOSIReferenceModel.htm

Kozierok, C. M. (den 15 4 2024). *IP Address Size, Address Space and "Dotted Decimal" Notation*. Hämtat från The TCP/IP Guide: http://www.tcpipguide.com/free/t_IPAddressSizeAddressSpaceandDottedDecimalNotation.htm

Kozierok, C. M. (den 15 04 2024). *IP Addressing Overview and Fundamentals*. Hämtat från The TCP/IP Guide: http://www.tcpipguide.com/free/t_IPAddressingOverviewandFundamentals.htm

Kozierok, C. M. (den 7 7 2024). *TCP and UDP Overview and Role In TCP/IP.* Hämtat från The TCP/IP Guide: http://www.tcpipguide.com/free/t_TCPandUDPOverviewandRoleInTCPIP-2.htm

Kozierok, C. M. (den 16 4 2024). *TCP/IP Sockets and Socket Pairs: Process and Connection Identification*. Hämtat från The TCP/IP Guide : http://www.tcpipguide.com/free/t_TCPIPSocketsandSocketPairsProcessandConnectionIden.htm

Kozierok, C. M. (den 10 4 2024). *What Is Networking?* Hämtat från The TCP/IP Guide: http://www.tcpipguide.com/free/t_WhatIsNetworking.htm

Microsoft. (den 23 4 2024). *.NET fundamentals documentation.* Hämtat från Microsoft Learn: https://learn.microsoft.com/pdf?url=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fdotnet%2Ffundamentals%2Ftoc.json

Microsoft. (den 20 4 2024). *C# language documentation.* Hämtat från Microsoft Learn: https://learn.microsoft.com/pdf?url=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fdotnet%2Fcsharp%2Ftoc.json

Microsoft. (den 22 4 2024). *Visual Studio IDE documentation*. Hämtat från Microsoft Learn: https://learn.microsoft.com/pdf?url=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fvisualstudio%2Ftoc.json%3Fview%3Dvs-2022

Microsoft. (den 24 4 2024). *Windows Presentation Foundation.* Hämtat från Microsoft Learn: https://learn.microsoft.com/pdf?url=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fdotnet%2Fdesktop%2Fwpf%2Ftoc.json%3Fview%3Dnetdesktop-8.0

Tanenbaum, A., & Wetherall, D. (2010). *Computer Networks (5th Edition).* Boston: Pearson.