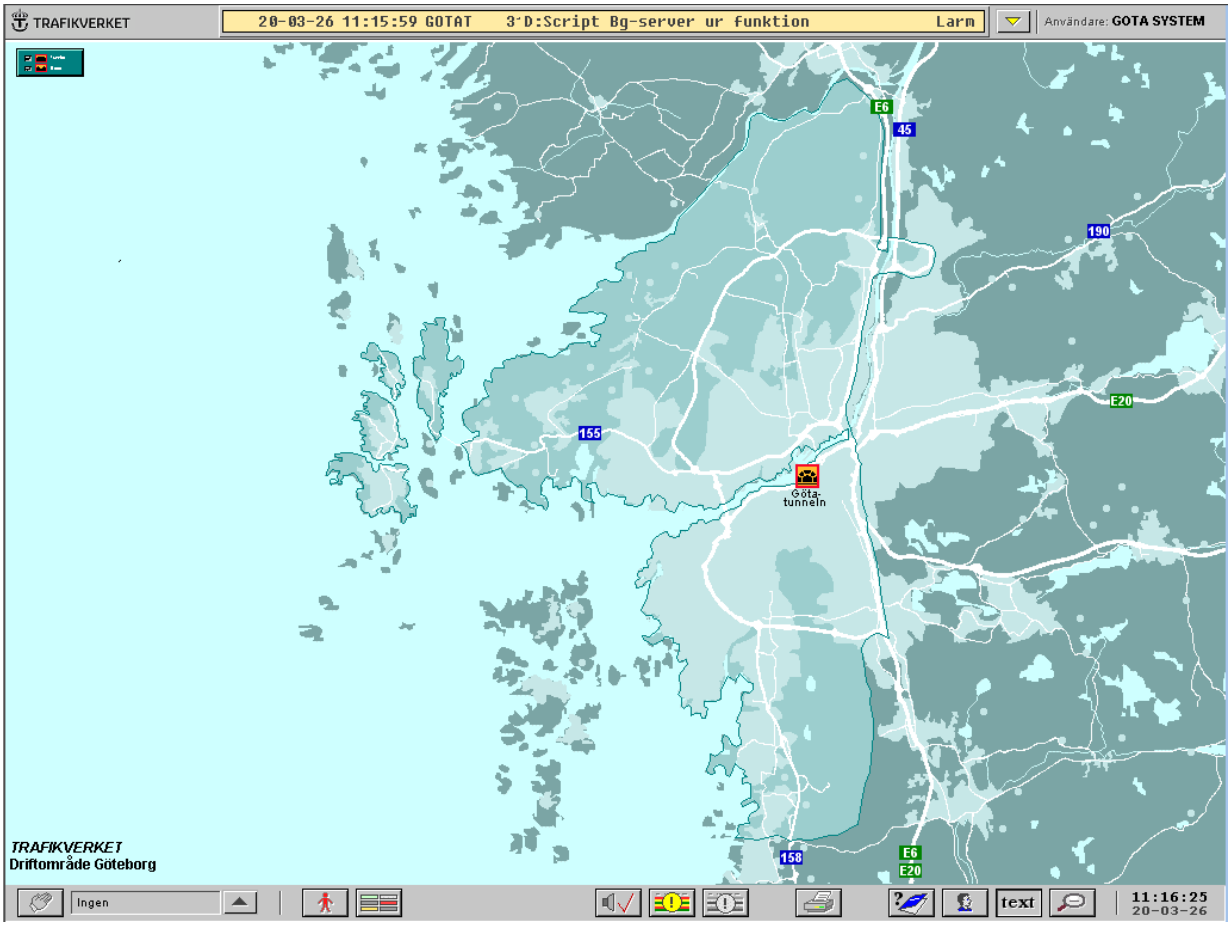
CHALMERS



Utveckling av larmhantering i iFIX-SCADA miljö

Development of Alarm Management in iFIX-SCADA environment

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

JAKOB ANDERSSON

ELIAS ISENSTIERNA

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2020

EXAMENSARBETE INOM HÖGSKOLEINGENJÖRSPROGRAMMET
MEKATRONIK

Utveckling av larmhantering i iFIX-SCADA miljö

JAKOB ANDERSSON
ELIAS ISENSTIERNA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2020

Utveckling av larmhantering i iFIX-SCADA miljö

JAKOB ANDERSSON
ELIAS ISENSTIERNA

Handledare: Anton Bergholtz, Midroc Automation

Examinator: Veronica Olesen, Institutionen för elektroteknik

Institutionen för Elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telefon: +46 (0)31-772 1000

Förstasida: Skärmdump från styrsystemet för Götatunneln, publicerad med tillstånd från Trafikverket

Förord

Detta projekt har utförts som examensarbete för utbildningen Mekatronikingenjör, på Chalmers Tekniska Högskola. Arbetet har utförts åt atuomationsföretaget Midroc Automation på plats i kontoret i Mölndal.

Vi vill tacka Anton Bergholtz som varit vår handledare på Midroc och som tagit sig tiden att vara behjälplig och besvarat våra frågor under arbetets gång. Vi vill även tacka alla anställda på Midroc Automation i Mölndal som gett oss ett varmt välkomnande och gjort att vi känt oss delaktiga i företaget. Sedan vill vi självklart rikta ett stort tack till Veronica Olesen, vår handledare och examinator på institutionen. Sist men inte minst vill vi tacka Hans Jönsson på Novotek för mycket god teknisk support.

Jakob Andersson och Elias Isenstierna, juni 2020, Göteborg

Sammanfattning

En tillförlitlig infrastruktur är en hörnsten för att dagens samhälle skall fungera. Eftersom infrastruktur finansieras med skattemedel, är alla åtgärder som sänker kostnader för service och underhåll av samhällsnytta. I Göteborgsområdet ansvarar Midroc Automation för drift och underhåll av styrsystemen i tunnlarna för fordonstrafik. I detta ingår att vid eventuella larm utreda orsaken samt åtgärda felet. I vissa fall väljer man att blockera larm, antingen för att åtgärder är planerade eller av annan anledning. Styrsystemet är uppbyggt med två servrar som lagrar all relevant information i en databas. Servrarna verkar redundant för att styrsystemet skall fungera felfritt även om en server är avstängd. Men en bristande funktion är då båda servrar stängs ner så förloras informationen om vilka larm som varit blockerade. Det leder till att Midroc på nytt behöver starta en utredning för att åtgärda något som redan är känt sedan tidigare. Syftet med projektet är således att ta fram en lösning till problemet att informationen om vilka larm som varit blockerade kan förloras. Resultatet av projektet är en fungerande lösning som lagrar de blockerade larmen och behåller informationen även när båda servrarna stängs ner. Detta leder i sin tur till att resurser i form av tid och pengar sparas. Eftersom larm som blockerats och redan hanterats inte riskerar att bli aktiva igen och orsaka en ny undersökning. Projektet har utförts i en testmiljö som utgår från det befintliga systemet i Götatunneln, men endast med de viktigaste komponenterna för att kunna utveckla en lösning. Det har således inte testats eller applicerats i verklig miljö under projektets gång, men målet är att lösningen skall appliceras i alla tunnlarna i Göteborgsområdet som använder samma system som Götatunneln.

Abstract

A reliable infrastructure is a cornerstone for today's society to function. Since the infrastructure is financed with tax funds, all measures that reduce the expenses of service and maintenance are of social benefit. In the Gothenburg area, Midroc Automation is responsible for the operation and maintenance of the control systems in the tunnels for vehicle traffic. This includes investigating the cause and rectifying the error in case of alarms. In some cases, the operator can choose to block an alarm if there is a planned measure in the future. The control system is constructed with two servers that store all relevant information in a database. The servers act redundantly to each other so the control system will function flawlessly even if one server is turned off. But a flawed feature in the system is when both servers are shut down, the information about which alarms have been blocked will be lost. This will cause Midroc to open an errand and examine a problem that is already known. The purpose of the project is thus to develop a solution to the specific problem, that is how to save the alarms that are blocked when both servers are shut down. The result of this project is a working solution that stores the blocked alarms and retains the information even when both servers are shut down. This in turn means that resources in the form of time and money can be saved, as alarms that have been blocked and already managed do not run the risk of being active again and causing another investigation. The project has been carried out in a test environment based on the existing system in the Göta tunnel, but only with the primary components in order to develop a solution. Thus it has not been tested nor applied in the real environment during the course of this project. But the goal is for the solution to be applied in all the tunnels in the Gothenburg area that use the same system as the Göta tunnel.

Innehållsförteckning

Figurförteckning

Tabellförteckning

Terminologi/ Förkortningar

Failover - Driftsäkerhetsfunktion som automatiskt lämnar över arbetet till en annan maskin.

HMI – Human-Machine Interface, någon form av display eller inmatningsenhet för mänsklig interaktion med en maskin.

I/O-enheter - Input/Output-enhet, någon form av periferiell enhet exempelvis en givare eller motor.

KVM - En förkortning för keyboard, video och mouse,

NSD - Network Status Display, en uppsättning av systemtaggar i iFIX som innehåller all information som rör nätverket.

NTS - Nationellt Trafikledningsstöd, ett system/operatörsstöd för vägtrafikledning och väganläggningsövervakning.

OPC - Open platform Communication, standarder för industriell kommunikation mellan serverar och klienter

PLC – Programmable Logic Controller, beräknings- och styrenhet för industriella applikationer. Används tillsammans med I/O-enheter.

RDBMS - Relational Database Management System (relationsbaserad databas), en databas där all data som lagras knyts till olika tillhörigheter och objekt.

SCADA - Supervisory Control And Data Acquisition, ett system för övervakning och styrning av servrar och PLCar.

1. Inledning

Denna uppsats beskriver hur man på bästa sätt kan lösa befintliga problem hos styrsystemen för säkerhetskritisk infrastruktur. Arbetet är utfört på uppdrag av Midroc Automation. En lösning på de problem som gett upphov för arbetet, bidrar till ett mer hållbart nyttjande av skattemedlen.

1.1. Bakgrund

Infrastruktur och fordonstrafik är centrala delar i dagens samhälle och det är kritiskt att dessa delar fungerar felfritt. Dessutom läggs stora delar av skattemedlen inom dessa områden [1]. Mycket styrs av automatiserade styrsystem och på dem ställs höga krav på funktionalitet och tillförlitlighet.

I Göteborgsområdet ansvarar Midroc Automation för drift och underhåll av styrsystemen i Trafikverkets tunnlar. Styrsystemen till varje tunnel är uppbygga av två redundanta servrar som sparar all information från all utrustning. En av servrarna är i drift medan den andra ligger i stand-by och kopierar information från driftservern.

När det uppstår ett fel hos någon utrustning aktiveras ett larm till operatörspanelerna. Där hanteras larmen av en tekniker som kan kvittera larmen. Om ett larm är frekvent återkommande och en åtgärdsplan finns, så kan larmet blockeras i en blockeringslista för att förbättra larmöversikten.

Om båda servrar stängs ner till följd av en oväntad händelse eller planerat underhåll förloras informationen om vilka larm som ligger i den blockerade listan. När servrarna startas igen kommer de läsa in samtliga larm som nya. Detta resulterar i att en ny utredning påbörjas och en tekniker behöver åka ut till berörd utrustning för att hantera dessa larm återigen. Följden blir ökade kostnader för skattebetalarna och ur ett hållbarhetsperspektiv är detta mycket negativt. För att möta framtidens utmaningar är det av största vikt att minimera all form av slöseri med skattemedlen så att dessa kan utnyttjas optimalt.

1.2. Syfte

Projektet syftar till att undersöka möjliga lösningar för att spara information om blockerade larm, samt applicera den bäst lämpade lösningen. Systemet skall spara information om larmstatus även om båda servrar fallerar. Detta skall leda till att servrarna läser av larminformationen vid återstart och samma larm som legat i den blockerade listan hamnar där efter återstart. En mycket viktig detalj för lösningen är att den skall vara robust och felsäker. Robust innebär i det här avseendet att lösningen inte får ha en negativ inverkan på styrsystemets tillförlitlighet.

Styrsystemen till Trafikverkets tunnlar drivs med programvaran Proficy iFIX, utvecklad av GE Digital. De äldre tunnlarna använder programversion 5.5, vilket är en äldre version. Därmed saknar systemet vissa funktioner som är standard i nyare versioner av programvaran. En av dessa funktioner är möjligheten att kommentera de blockerade larmen, vilket leder till bristande dokumentation gällande larmhantering. Ett önskemål från Midroc är därför att undersöka hur man gör det möjligt att lägga in kommentarer såsom blockeringsorsak samt ansvarig.

1.3. Avgränsningar

Testanläggningen kommer enbart bestå av en klient och två servrar. Således kommer ingen testning att ske mot PLC och I/O-enheter. Projektet kommer arbeta utifrån befintliga processbilder och kommer i största möjliga mån undvika att göra förändringar på dessa. Lösningen behöver inte vara direkt applicerbar på andra versioner av iFIX än version 5.5. Men den kan komma att användas som en mall för andra versioner. Lösningen kommer ej att appliceras på verklig anläggning under det här projektet. Detta eftersom genomförande av förändringar måste godkännas hos Trafikverket. Men målet är ändå att det skall komma till användning.

1.4. Precisering av frågeställningen

Frågeställningen för rapporten är följande:

- Vilka metoder finns för att hantera och spara undan larm i iFIX 5.5 och vilka är för/nackdelar med dessa?
- Hur lägger man till möjligheten att kommentera blockerade larm?
- Hur fungerar synkroniseringen mellan servrarna?
- Hur är servrarna fysiskt kopplade till varandra och resterande utrustning?
- Vilka potentiella risker tillkommer med att modifiera befintlig programkod samt införa ny kod?

2. Teoretisk / Teknisk bakgrund

I detta kapitel redovisas viktig teori och begrepp för projektet.

2.1. Server och klient

En server är en dator som tillgodoser någon form av tjänst eller resurs. Det kan vara till exempel en webbserver som lagrar webbsidor eller en databasserver som kör ett databashanteringsprogram [2].

I systemet används 2-lagers klient-server systemarkitektur. Denna består av en eller flera klientdatorer som är kopplade över ett nätverk till en eller flera servrar. På klienternas begäran utför servrarna olika uppgifter eller hämtar och skriver data [3].

2.2. Relationsdatabaser

En databas är en samling information som används för att lagra och organisera stora mängder data. I en relationsbaserad databas (RDBMS) knyts all data som lagras till olika tillhörigheter och objekt, vilket gör den lätt sökbar, och möjliggör snabb åtkomst av önskade data. Ett urval av de vanligaste objekten som används i en databas är tabeller, funktioner, lagringsprocedurer och granskning [4]. Användning av dessa objekt utökar användarvänligheten och produktiviteten mycket vid utformning av databasen.

2.3. Systemavbildning

En systemavbildning (system image, engelska) är en fullständig kopia av en dators hårddisk som sparas på en nätverksdisk eller på ett flyttbart media. Denna används sedan för att återskapa en dator till exakt samma utförande som när avbilden var tagen. En systemavbild innehåller även de drivrutiner som krävs för att köra operativsystemet [5].

2.4. PLC och I/O-enheter

En PLC är en dator som används för logiska beräkningar och är vanligt förekommande inom industrin. Populariteten hos PLCar i industrin grundar sig i de är enkla till sitt utförande och robusta för att verka även i tuffa miljöer. En PLC består till grunden av en beräkningsenhet dit olika moduler ansluts för att möjliggöra kontroll och styrning av yttre enheter, även kallat I/O-enheter. Dessa moduler kan vara ingångs- eller utgångsmoduler för olika typer av signaler. Modulerna innehåller ett antal portar dit signalerna för en I/O-enhet ansluts [6] .

2.5. Kommunikationsprotokoll

För kommunikation mellan PLC och I/O-enheter används olika protokoll beroende på vilket som är bäst lämpat för aktuellt användningsområde. De kommunikationsprotokoll som används i systemet är Modbus TCP/IP, Profibus, Profinet, samt seriellt RS232 och RS485.

2.6. SCADA

SCADA (Supervisory Control And Data Acquisition) är ett system för övervakning och styrning av fabriker, fastigheter, infrastruktur och utrustning. En SCADA-enhet kan bestå av ett flertal datorer, servrar och PLCar. Detta kombinerat med ett grafiskt användargränssnitt ger ett överskådligt utförande för att visualisera informationen [7].

2.7. NTS – Nationellt trafikledningsstöd

Nationellt trafikledningsstöd (NTS) är ett system/operatörsstöd för vägtrafikledning och väganläggningsövervakning. Systemet är en nivå över alla SCADA-enheter och

sammanställer informationen från dessa. Ifrån NTS har man åtkomst till Trafikverkets samtliga anläggningar [8].

2.8. OPC – Open Platform Communication

Open platform Communication (OPC) är ett antal standarder för industriell kommunikation mellan server och klient samt server till server [9]. Vid användning av OPC är alla servrar och klienter kopplade till OPC-applikationen som hanterar all begäran om läsning och skrivning. OPC kan kommunicera med alla fabrikat av PLC och man väljer vilket protokoll som används för varje PLC. För varje PLC är alla ingångar och utgångar kopplade till separata taggar i OPC-applikationen. Dessa innehåller det värde som är avläst eller som skall skrivas. Värdet kopieras sedan till databasservrarna. Trafikverkets tunnlar använder sig av Mitsubishis OPC programvara kallat MXOPC.

2.9. KVM - Keyboard, Video and Mouse

KVM (Keyboard, Video and Mouse) är en funktion som gör det möjligt att styra flera datorer med samma mus, tangentbord och skärm från samma plats. Tekniken är användbar när man från en avlägsen arbetsstation vill få åtkomst till flera servrar i en serverhall [10].

2.9.1. KVM switch

En KVM switch gör det möjligt att på ett smidigt sätt växla mellan olika datorenheter. Den har en port för en arbetsstation samt flera portar för de enheter som skall manövreras. Modellen i testmiljön, Aten CS1708A KVM Switch, har åtta portar och kan således styra lika många enheter [11].

2.9.2. KVM extender

En KVM extender används för att skicka KVM-signaler över längre avstånd, exempelvis till en konsol som inte är i direkt anslutning till ett serverrum. Extendern består av två enheter, en som kopplas lokalt och en avlägsen. Den lokala kopplas in vid arbetsplatsen och fjärrenheten kopplas tillsammans med en KVM-switch in vid utrustningen som skall styras. Signalen skickas mellan enheterna genom en nätverkskabel. Modellen i testmiljön är Aten CE700A med en räckvidd på upp till 150m [12].

2.10. Proficy iFIX

Proficy iFIX är en HMI/SCADA-plattform som är utvecklad av GE Digital. Plattformen används för automation av processer och anläggningar. Det är ett kraftigt verktyg som består av många komponenter, varav några beskrivs nedan [13].

2.10.1. Proficy License Manager

Proficy License Manager är ett program som används för att hantera licenser till iFIX och andra GE-programvaror. Hanteringen består av aktivering, returnering och förnyande av licenser. Beroende på vilken typ av licens och system som skall användas finns det olika metoder för detta. Exempel på metoder är licenser på USB-nycklar och licensservrar [14].

2.10.2. iFIX HMI/SCADA-funktioner

Några av de SCADA-funktioner som ingår i iFIX är möjligheten att övervaka en anläggning, hantera larmsignaler samt möjliggöra reglering av dess processer. De kan vara av kontinuerlig, batchvis eller statistisk karaktär. Regleringens omfattning kan skräddarsys efter behov.

SCADAN samlar in data från all utrustning och presenterar den på ett överskådligt sätt genom processbilder. Bilderna ger användaren möjlighet att ändra börvärden och styra varje enskild utrustning i realtid. Det går att införa olika behörighetsnivåer för olika användare, till exempel operatör, ingenjör och utvecklare.

Systemet kan identifiera och underrätta operatören om särskilda händelser i anläggningen genom meddelanden och larm. Dessa genereras utefter förinställda gränser och villkor. Normalt krävs en högre behörighet i systemet för att justera dessa larmgränser.

Många systemfunktioner kan ske automatiskt och tar bort behovet av mänsklig interaktion. Styrningen sker utefter förprogrammerade algoritmer för att bibehålla utrustningen inom bestämt driftområde [15].

2.10.3. Database Manager

Database Manager är det programmet som man använder för att överblicka och göra förändringar i databasen. Sök- och filtreringsfunktioner möjliggör enkel åtkomst av önskade taggar.

Databasen byggs upp av olika block, där vissa är direkt kopplade till en tagg. Blocken kan vara av enklare typ såsom digital I/O, analog I/O, larmblock och triggerblock. Men det finns även mer avancerade block som kan användas för beräkning, loggning av data, eller programblock som utför en sekvens [17].

2.10.4. Serversynkronisering

Synkronisering mellan servrar är en viktig säkerhetsfunktion för att förhindra förlust av data i system, och det är denna som skapar redundans. I iFIX finns en inbyggd funktion för detta kallad Enhanced Failover. Funktionen bygger på att ena servern har status aktiv och den andra status standby. Servern som är aktiv har tillgång till att göra ändringar i databasen. Servern med status standby kopierar kontinuerligt informationen från den aktiva servern med ett inställt intervall. I iFIX finns det specifika systemtaggar kallade NSD (Network Status Display) som innehåller all information om anslutningen mellan servarna, till exempel om servrarna är anslutna och vilken server som är Aktiv [16].

Ett larm aktiveras i SCADAn när en av servrarna förlorar anslutningen. Om den aktiva servern förlorar anslutning kommer en failover att ske, vilket medför att standby servern övergår till aktiv. Denna förblir aktiv tills en manuell failover utförs. Om båda servrar förlorar anslutning så kommer den först anslutna servern att bli aktiv [16].

2.10.5. VBA- Visual Basic for Applications

Visual Basics for Applications är ett programspråk utvecklat av Microsoft anpassat för att skriva script. Ett script är en typ av programkod men som skiljer sig i det avseendet att det inte kompileras som vanlig programkod. Istället för att processorn omvandlar hela koden till maskinkod som kan tolkas, så tolkas scriptet rad för rad direkt av ett program som i det här fallet är iFIX [18]. Normalt används script för att utföra en begränsad programfunktion eller uppgift [19].

I iFIX används språket för att skräddarsy automationslösningar som kan vara både enkla och mer komplicerade. Antingen skrivs nya script från grunden eller så redigeras befintliga funktioner i iFIX. Scripten kopplas ofta till någon händelse i HMI-bilderna eller

bakgrundsfunktioner. Exempelvis när en knapp i en HMI-bild trycks på eller när en tagg i databasen ändras [20].

2.10.6. Scheduler

För att utföra ett script vid en särskild tidpunkt eller vid en händelse kan iFIX Scheduler användas. Händelser kan aktiveras när en tagg får en positiv flank, negativ flank eller för båda flanker. Händelsen kan också förbli aktiverad kontinuerligt medan en tagg behåller värde ”sann” eller ”falsk”. Tidsbaserade Schedules aktiveras vid ett angivet datum och tidpunkt, antingen återkommande eller vid ett engångstillfälle. Samtliga Schedules kan köras samtidigt och helt oberoende av varandra. I applikationen åskådliggörs alla Schedules i ett kalkylark som är uppdelat i tidsaktiverade eller händelseaktiverade. Där demonstreras vilka som är startade eller stoppade, samt hur många exekveringar som utförts [21] [22].

2.10.7. Workspace

Workspace är det övergripande iFIX-programmet som innehåller alla bilder och applikationer. Där bläddrar man mellan HMI-bilderna, larmen och Schedules. Med rätt behörigheter i Workspace kan man förändra HMI-bilderna, samt ta fram all programkod kopplad till dessa och redigera vid behov. Det är även i Workspace som man bygger upp och lägger till nya bilder [23].

3. Metod

Metodavsnittet förklarar vilka tillvägagångssätt som använts för att uppnå det slutgiltiga resultatet.

3.1. Arbetsgång

En testanläggning upprättades för utveckling och testning av styrsystemet. Anläggningen bestod av två servrar samt en klientdator. Kommunikationen mellan servrar och klientdator utfördes över ett lokalt nätverk. Systemavbilder av servrar och klient ifrån verklig anläggning installerades på testanläggningen. Detta gav en realistisk utvecklingsmiljö.

Testanläggningen utgick ifrån Götatunnelns styrsystem, men då övriga tunnlar var snarlika blev således lösning även kompatibel med Lundby-, Gnistäng- och Tingstadstunneln. Lösningen utvecklades i iFIX 5.5, vilken var den mjukvara som användes i det verkliga styrsystemet. Med utvecklingsbehörighet aktiverat i iFIX Workspace fick man fullkomlig åtkomst och redigeringsbehörighet i programmet.

3.2. Beslutsmatris

Vid beslutandet av vilka lösningarna som valdes att vidareutveckla användes en oviktad beslutsmatris. Matrisen hjälpte till att sortera ut de lösningar som inte uppnådde de ställda kraven. Om första beslutsmatrisen hade gett många möjliga lösningar så skulle en viktad Pughs matris använts för att sålla urvalet ytterligare.

I tabell 3.1 redovisas de krav och meriter som använts till beslutsmatrisen för att sålla bland de möjliga lösningarna till problemet. Beslutsmatrisen redovisas i tabell 3.2. De lösningar som uppfyllde kraven var lösning 1, 2 och 3. Dessa lösningar var de som vidareutvecklades och redovisas i kapitel 5.

Tabell 3.1: De krav och meriter som använts för sållning av lösningar

Krav 1	Ingen kostnad	Meriterande 1	Enkel lösning
Krav 2	Hög säkerhet (känslig information)		
Krav 3	Tillförlitlig/ felsäker		
Krav 4	Aktuell status av blockerade larm skall uppdateras automatiskt		
Krav 5	Lösningen skall vara enkel att underhålla		

Tabell 3.2:Beslutsmatris som visar om lösningarna uppfyller krav och meriter

Lösningar	**Krav 1 (ja/nej)**	**Krav 2 (ja/nej)**	**Krav 3 (ja/nej)**	**Krav 4 (ja/nej)**	**Krav 5 (ja/nej)**	**Meriterande 1 (ja/nej)**
1: Använda programblock i databasen för att spara informationen på servrarna.	ja	ja	ja	ja	ja	ja
2: Skriva ett script i VBA som sparar informationen i en textfil på klientdator	ja	ja	ja	ja	ja	ja
3: Skriva ett script i VBA som sparar informationen i en textfil på servrarna	ja	ja	ja	ja	ja	ja
4: En molnbaserad lösning för att spara information	nej	nej	ja	ja	ja	ja
5: Installera en till fysisk server	nej	ja	ja	ja	ja	ja
6: Spara bitar för larmen i varje PLC som läses in till SCADA vid uppstart	ja	ja	ja	nej	nej	ja
7: Spara informationen i en loggfil och manuellt lägga in på nytt	ja	ja	nej	nej	ja	ja

3.3. Testning och verifiering av lösning

För verifiering av lösningens funktionalitet upprättades testprotokoll för de olika funktionerna. Genomförande av tester utifrån dessa protokoll gav ett underlag för att kunna ta beslut om lösningens funktionalitet var korrekt eller inte.

Protokollen utformades för att genomföras steg för steg och ovanför varje tabell anges förutsättningarna. I kolumnen ”Beskrivning” så beskrivs vad som skall göras för att testa funktionen och i ”Förväntat resultat” vad som skall hända om funktionen fungerar. Därefter skall ett resultat på testet fyllas i som ”OK” eller ”NOK” (Not OK) tillsammans med initialer för vem som utfört testet. Eventuella kommentarer anges även i protokollet, främst om någonting inte skulle fungera.

I tabell 3.3 visas testprotokoll 1 som testar funktionerna att blockera samt avblockera larm.

Tabell 3.3: Testprotokoll 1 för verifiering av funktion blockera/avblockera larm

Test 1: blockering/avblockering av larm				
Förutsättningar: Servrar anslutna till Viewklient, fixbackgroundserver är igång på alla enheter				
Steg	**Beskrivning**	**Förväntat resultat**	**Händelse**	**Kvittens**
1	Blockera valfritt larm	Larm blockeras, motsvarande blocktagg sparas i txt-fil på aktiv server	(OK/NOK)	(JA/EI)
2	Avblockera valfritt larm	Larm avblockeras, motsvarande blocktagg tas bort från txt-fil på aktiv server		
3	Blockera fem valfria larm i snabb följd	Fem larm blockeras, motsvarande blocktaggar sparas i txt-fil på aktiv server		
4	Avblockera fem valfria larm i snabb följd	Fem larm avblockeras, motsvarande blocktaggar tas bort från txt-fil på aktiv server		
5	Blockera alla larm i snabb följd	Alla larm blockeras, motsvarande blocktaggar sparas i txt-fil på aktiv server		
6	Avblockera alla larm i snabb följd	Alla larm avblockeras, motsvarande blocktaggar tas bort från i txt-fil på aktiv server		
7	Skriv en rad som är över 80 tecken på första raden i Blocklarm.txt	Meddelanderuta skall dyka upp med hänvisning till blockeringsfilen		
8	Skriv en rad som är över 80 tecken på första raden i Unblocklarm.txt	Meddelanderuta skall dyka upp med hänvisning till avblockeringsfilen		

I tabell 3.4 så visas testprotokoll 2 som testar funktionen att synkronisera textfilen mellan servrarna.

Tabell 3.4: Testprotokoll 2 för verifiering av funktion att synkronisera textfil mellan servrar

Test 2: Synkronisering av txt-fil mellan servrar

Förutsättningar: Servrar anslutna till varandra, SAC körs på båda servrar med fungerande kommunikation

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	GOTA_C1 aktiv, GOTA_C2 backup. Stäng av GOTA_C2, blockera fem larm, starta upp GOTA_C2	Identisk txt-fil som på GOTA_C1 med de fem motsvarande larmen skall finnas på GOTA_C2 efter uppstart	(OK/NOK)	(JA/EI)
2	GOTA_C2 aktiv, GOTA_C1 backup. Stäng av GOTA_C1, blockera fem larm, starta upp GOTA_C1	Identisk txt-fil som på GOTA_C2 med de fem motsvarande larmen skall finnas på GOTA_C1 efter uppstart		
3	GOTA_C1 aktiv, GOTA_C2 backup. Stäng av GOTA_C2, avblockera fem larm, starta upp GOTA_C2	Identisk txt-fil som på GOTA_C1, de fem avblockerade larmen skall vara raderade på GOTA_C2 efter uppstart		
4	GOTA_C2 aktiv, GOTA_C1 backup. Stäng av GOTA_C1, avblockera fem larm, starta upp GOTA_C1	Identisk txt-fil som på GOTA_C2, de fem avblockerade larmen skall vara raderade påGOTA_C1 efter uppstart		
5	Starta om både GOTA_C1 och GOTA_C2	Txt-fil på GOTA_C1 och GOTA_C2 skall vara identisk före och efter omstart		

I tabell 3.5 nedan så beskrivs arbetsgången för att testa funktionen att larm läses in från textfil och blockeras när bägge servrarna varit avstängda.

Tabell 3.5: Testprotokoll 3 för verifiering av funktion att läsa in blockerade larm vid omstart av båda servrarna

Test 3: Omstart av servrar och inläsning av blockerade larm från txt-fil

Förutsättningar: Larmtaggar återfinns i txt-fil, båda servrar startas om ungefär samtidigt

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	GOTA_C1 aktiv, GOTA_C2 backup. Starta om GOTA_C1 först, sedan GOTA_C2.	Motsvarande larm i txt-fil skall blockeras i iFIX	(OK/NOK)	(JA/EI)
2	GOTA_C2 aktiv, GOTA_C1 backup. Starta om GOTA_C2 först, sedan GOTA_C1.	Motsvarande larm i txt-fil skall blockeras i iFIX		
3	GOTA_C1 aktiv, GOTA_C2 backup. Starta om GOTA_C2 först, sedan GOTA_C1.	Motsvarande larm i txt-fil skall blockeras i iFIX		
4	GOTA_C2 aktiv, GOTA_C1 backup. Starta om GOTA_C1 först, sedan GOTA_C2.	Motsvarande larm i txt-fil skall blockeras i iFIX		

I tabell 3.6 visas testprotokoll 4 som testar Timeout-funktionen på servrarna.

Tabell 3.6: Testprotokoll 4 för verifiering av Timeout-funktion på servrarna

Test 4: Timeout funktion på servrar				
Förutsättningar: Servrar anslutna till Viewklient, fixbackgroundserver är igång på alla enheter				
Steg	**Beskrivning**	**Förväntat resultat**	**Händelse**	**Kvittens**
1	GOTA_C1 aktiv. Sätt tagg GOTA_BLOCK_QUEUE "FRÅN" i databas	Timeout funktionen skall återställa ("TILL") tagg: GOTA_BLOCK_QUEUE	(OK/NOK)	(JA/EI)
2	GOTA_C2 aktiv. Sätt tagg: GOTA_BLOCK_QUEUE "FRÅN" i databas	Timeout funktionen skall återställa ("TILL") tagg: GOTA_BLOCK_QUEUE		
3	GOTA_C1 aktiv. Simulera att Schedulern för blockering har hängt sig genom att stoppa GOTA_larm_blockering på aktiv server. Blockera sedan ett antal larm från klient	Motsvarande larm som blockerats på klient ska återfinnas i txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout		
4	GOTA_C2 aktiv. Simulera att Schedulern för blockering har hängt sig genom att stoppa GOTA_larm_blockering på aktiv server. Blockera sedan ett antal larm från klient	Motsvarande larm som blockerats på klient ska återfinnas i txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout		
5	GOTA_C1 aktiv. Simulera att Schedulern för avblockering har hängt sig genom att stoppa GOTA_larm_avblockering på aktiv server. Avblockera sedan ett antal larm från klient	Motsvarande larm som avblockerats på klient ska raderats från txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout		
6	GOTA_C2 aktiv. Simulera att Schedulern för avblockering har hängt sig genom att stoppa GOTA_larm_avblockering på aktiv server. Avblockera sedan ett antal larm från klient	Motsvarande larm som avblockerats på klient ska raderats från txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout		

I tabell 3.7 visas testprotokoll 5 för att testa funktionerna i fullständiga systemet med alla klienter anslutna.

Tabell 3.7: Testprotokoll 5 för verifiering funktionalitet i fullständigt system

Test 5: Funktionalitet i fullständigt system

Förutsättningar: Alla klienter kopplade till systemet, all kod uppdaterad, fixbackgroundserver är igång på alla enheter

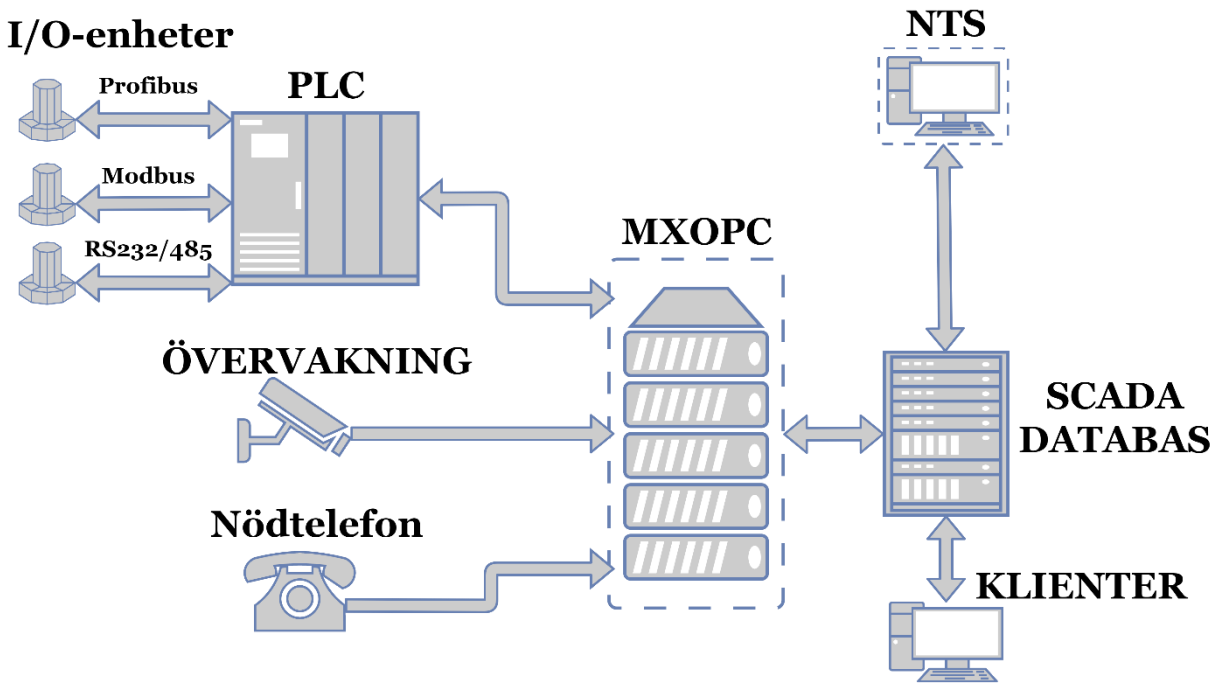
Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	Blockera larm från flera klienter samtidigt	Motsvarande larm i txt-fil skall blockeras i iFIX	(OK/NOK)	(JA/EI)
2	Avblockera larm från flera klienter samtidigt	Motsvarande larm i txt-fil skall avblockeras i iFIX		

4. Styrsystem

I följande kapitel beskrivs den verkliga miljön för styrsystemet i tunneln samt hur testanläggningen är uppbyggd. Även hur delar av systemet ser ut, samt några av de program det består av som varit relevanta för utförandet av projektet.

4.1. Verklig miljö i Götatunneln

Systemet i Götatunneln är uppbyggt av flera komponenter och enheter som åskådliggörs i figur 4.1. Klienterna och NTS-systemet övervakar tunneln genom att kontinuerligt titta i SCADAns databas. Databasen är i sin tur kopplad till underliggande utrustning såsom PLCar, övervakningsenheter och nödtelefoner via en OPC.



Figur 4.1: Bilden visar hur komponenterna i den verkliga anläggningen är sammankopplade.

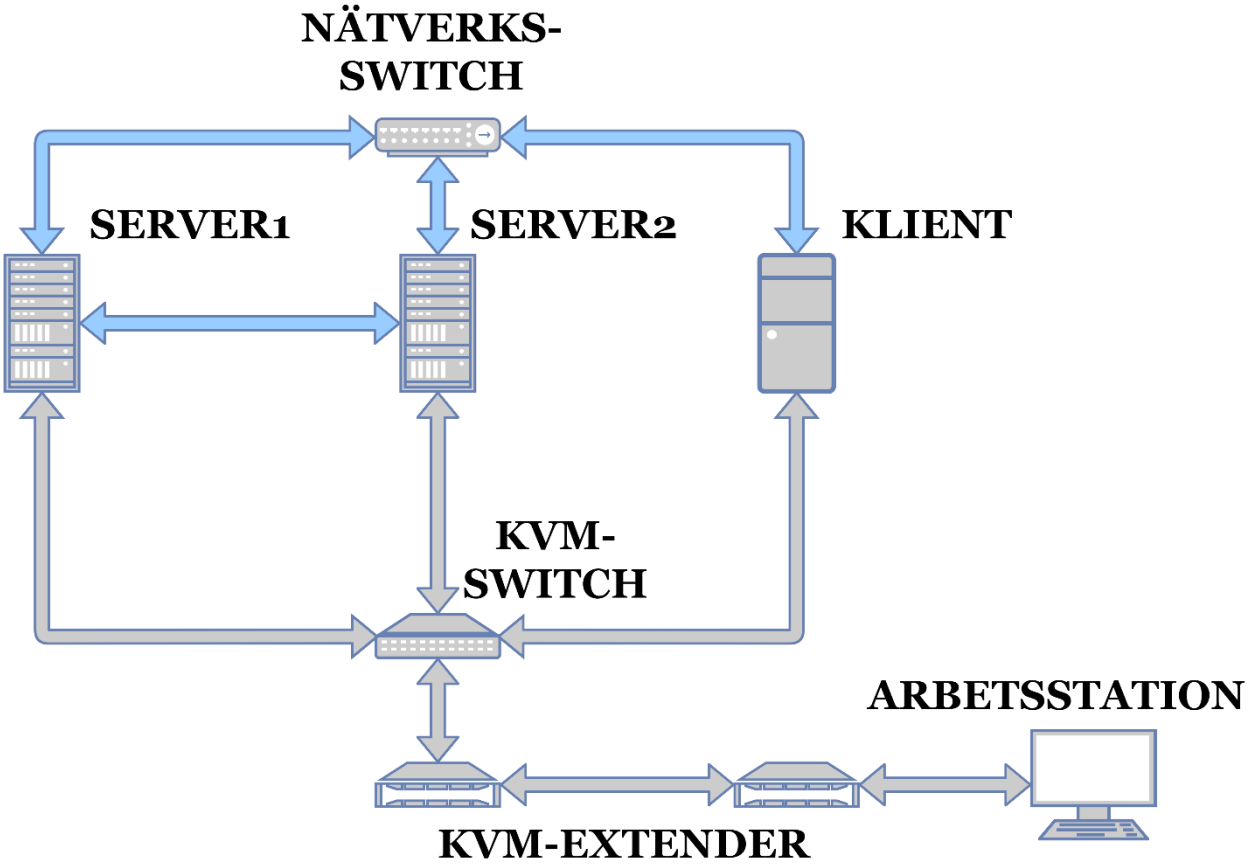
4.2. Testmiljö

De komponenterna som behövdes för att kunna upprätta en testmiljö för larmhanteringen var en klient och två servrar. För att kunna simulera den verkliga miljön togs systemavbilder från den verkliga hårdvaran. Dessa installerades sedan på två servrar av samma modell som i det verkliga systemet. Systemavbildningen från klienten i tunneln installerades på en annan klientdator som används i testmiljön.

Servrarna kör operativsystemet Windows Server 2008 64-bit och klienten Windows 7 32-bit. De installerades med respektive installationsskiva tillsammans med systemavbilderna, vilket gav att en testmiljö snarlik det verkliga systemet.

De tre enheterna är sammankopplade i ett lokalt nätverk med en nätverksswitch, för att iFIX och dess applikationer skall kunna kommunicera. Ytterligare en kabel är kopplad direkt mellan servrarna för synkroniseringsfunktionen, se figur 4.2. Detta är möjligt genom att servrarna har två separata nätverkskort installerade.

För att skapa en hållbar arbetsmiljö med avseende på ljudnivå, placerades servrar och klient på en avskild yta. Detta möjliggjordes genom att ansluta de tre enheterna till en KVM-switch, som i sin tur kopplades vidare genom en KVM-extender. På mottagarsidan av KVM-extendern kopplades en arbetsstation upp.



Figur 4.2: En översikt för testanläggningen. De ljusblå pilarna symboliserar det lokala nätverket och de grå pilarna KVM-komponenterna.

4.3. Licensnycklar

Alla Proficy-programmen som systemet är uppbyggt av måste köras med giltiga licensnycklar för att ge tillgång till de fullständiga funktionerna. De licensnycklar som det verkliga systemet använder är av en äldre version, medan de utvecklarnycklarna Midroc har är av nyare version. Systemavbildningarna som installerades var anpassade för att köras med den äldre versionen av licensnycklar, vilket blev ett problem som behövde lösas.

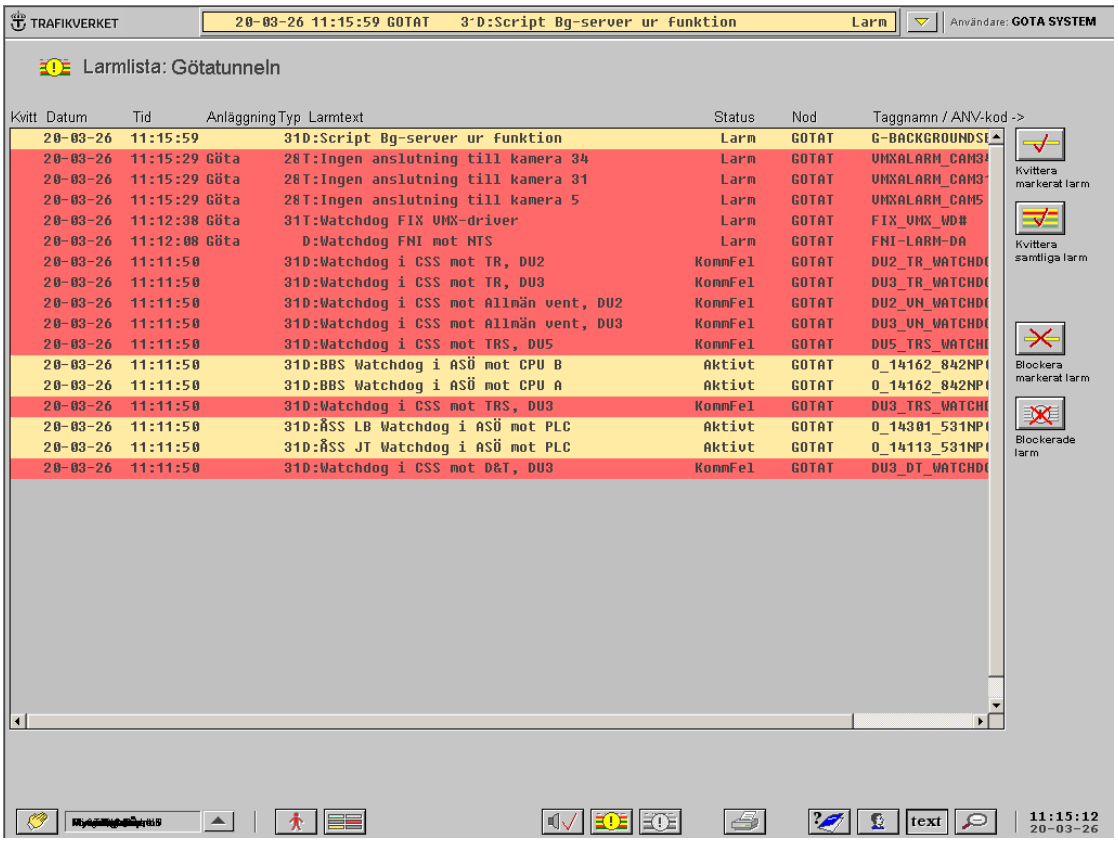
För att använda sig av de nyare licensnycklarna krävdes en installation av Proficy License Client. På servrarna gick det att installera programmet men på klientdatorn så uppstod komplikationer. Programmet gav felmeddelanden att Windows-versionen var gammal och saknade särskilda säkerhetsuppdateringar. Installation av program och funktionspaketet .NETframework version 4.7.1 löste problemet och programmet kunde köras [25]. Det medförde även att den nya versionen av licensnycklarna kunde läsas in och tillgång till de fullständiga funktionerna var i bruk.

4.4. Gränssnitt larmhantering

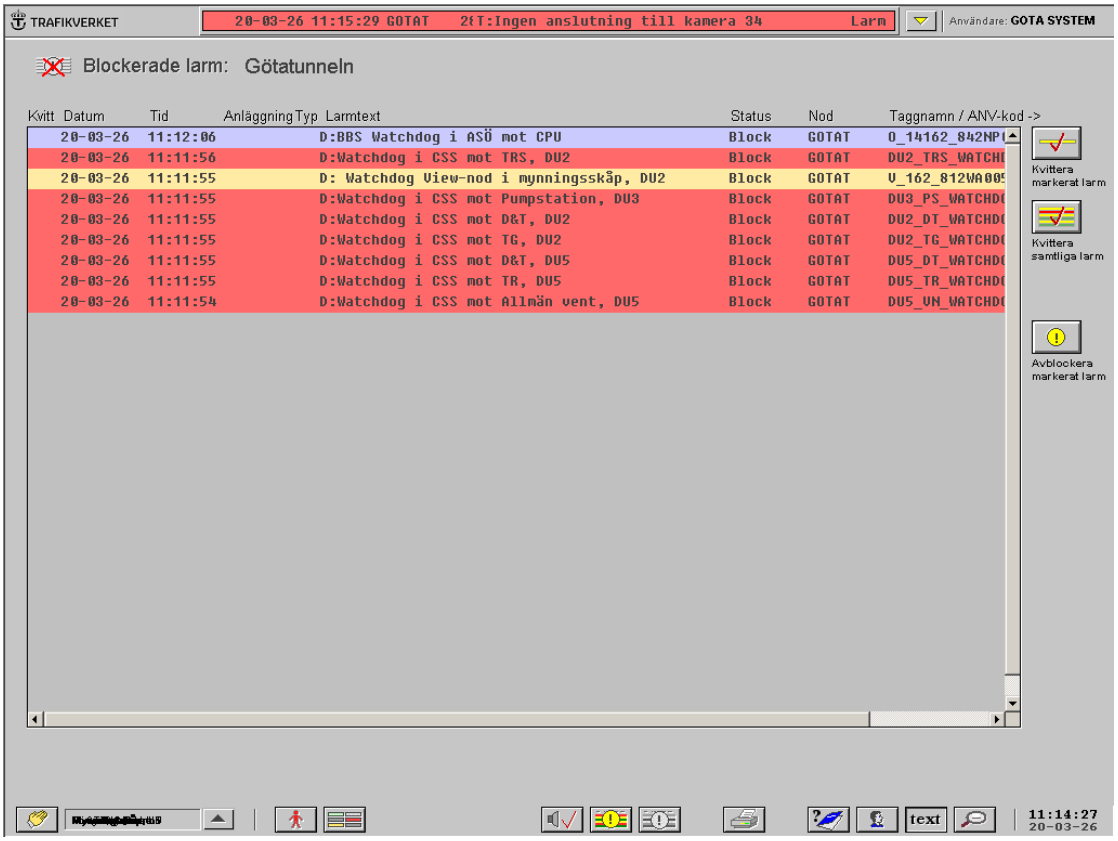
HMI-bilderna används för att åskådliggöra all information i ett användarvänligt gränssnitt. Detta för att operatören skall ges bästa möjliga förutsättningar att styra anläggningen. Bilderna består av en översikt för varje tunnel samt bilder uppdelade i tunnelns olika sektioner som är mer detaljerade. Det finns också bilder som visar hur kommunikationen är sammankopplad samt aktuell driftstatus på de olika servrarna.

Det finns separata bilder för larmhanteringen se figur 4.3. I larmlistan visas all nyckelinformation som är kopplat till larmet. Såsom tid och datum larmet aktiverades. Vilken anläggning som larmet tillhör samt en larmtext som verkar som en kort beskrivning av larmet. I listan presenteras larmets aktuella status, vilken nod/databas som larmet är sparat i, samt fullständigt taggnamn i databasen. I larmbilden markerar man ett larm och trycker på kvitteringsknappen för att bekräfta att man sett larmet. Om man vill dölja ett aktivt larm markerar man larmet och trycker på blockeringsknappen. Larmet hamnar då i blockeringslistan, se figur 4.4.

Larmen som ligger i blockeringslistan ligger kvar i listan oavsett status på larmet. För att ta bort ett larm från blockeringslistan markerar man aktuellt larm och trycker på avblockeringsknappen.



Figur 4.3: Larmlistan i Götatunneln från iFIX, publicerad med tillstånd från Trafikverket.



Figur 4.4: Listan för blockerade larm i Götatunneln från iFIX, publicerad med tillstånd från Trafikverket

4.5. Hållbarhetsaspekter

Vikten av ett säkert och tillförlitligt styrsystem för infrastrukturanläggningar är av kritiska mått. Detta eftersom ett felande styrsystem i en tunnel kan direkt eller indirekt vara en bidragande orsak till att en trafikolycka uppstår, med ett stopp i trafiken som följd. Detta resulterar i sin tur till samhällsekonomiska kostnader. Dessa kostnader kan vara i form av förseningar, utebliven produktion, samt kostnader för omledning av trafik med mera [24]. Dessutom tillkommer de direkta kostnaderna för själva olyckan.

Störningskostnaderna för ett stopp i vägtrafiken kan beräknas genom, uppskattning av antal förseningstimmar, vilka fordon som trafikerar vägarna (fordonsfördelning), beläggningsgraden på vägen och ärendefördelningen för trafiken [24]. Utöver dessa data används även ett antal viktade faktorer i beräkningen. Dessa faktorer redovisas i ASEK (Analysmetod och samhällsekonomiska kalkylvärden för transportsektorn) [25]. Formeln som kan användas för att räkna ut störningskostnader blir då enligt (1) [24].

$$Störningskostnad = förseningstimmar \times fordonsfördelning \times ärendefördelning \times beläggningsgrad \times åktidsvärde \times förseningstidsvärde \times trängseltidsvärde \quad (1)$$

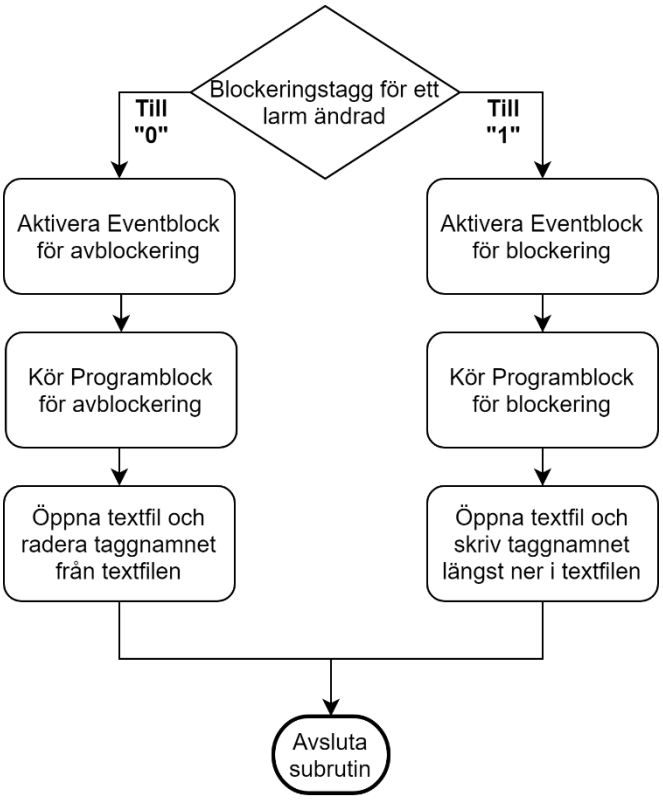
I rapporten nämns dock svårigheterna med att göra en noggrann och tillförlitlig beräkning på grund av brister och osäkerheter i dataunderlaget [24].

5. Analys av möjliga lösningar

I detta kapitel redovisas alla möjliga lösningar där utveckling påbörjats.

5.1. Lösning 1 – Programblock i databasen

Den första förslaget till en lösning bygger på att använda ett programblock i Database Manager. Programblock kan användas för att utföra en sekvens, samt är det enda block som har möjlighet att öppna filer utanför databasen. Blocket skall köra en sekvens som visas i figur 5.1, vilken sorterar ut alla taggar för blockerade larm. De taggar som ligger aktiva skrivs sedan till en separat textfil på servern. Ett eventblock används för att köra programblocket när en blockeringstagg blir ändrad.



Figur 5.1: Flödesschema över lösningen med programblock i databasen

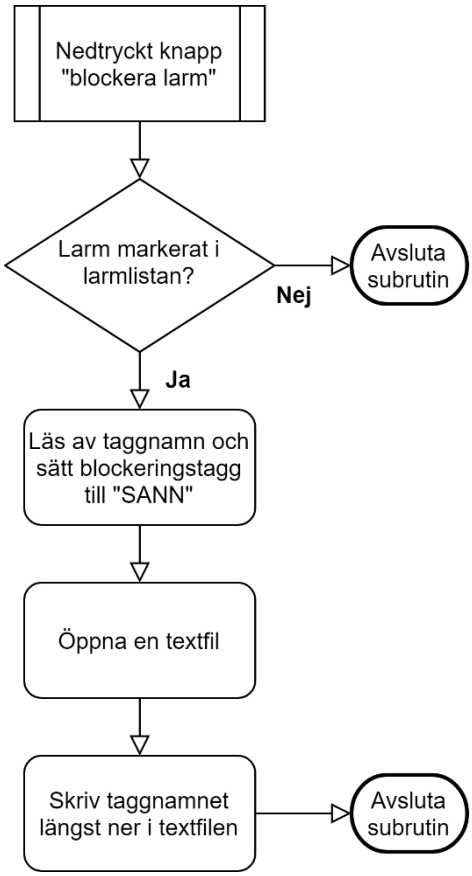
Denna lösning har tyvärr inte varit möjlig att genomföra eftersom programblocken inte fungerat som avsett. Det har inte på något sätt varit möjligt att öppna en textfil eller någon annan extern fil med ett programblock. Följaktligen faller hela lösning och därav finns ingen ”produkt” att redovisa för denna lösning.

5.2. Lösning 2 - Programmering i VBA på klientdator

Det andra förslaget till en lösning implementeras i klientdatorn med programmering i VBA.

5.2.1. Spara blockerade larm

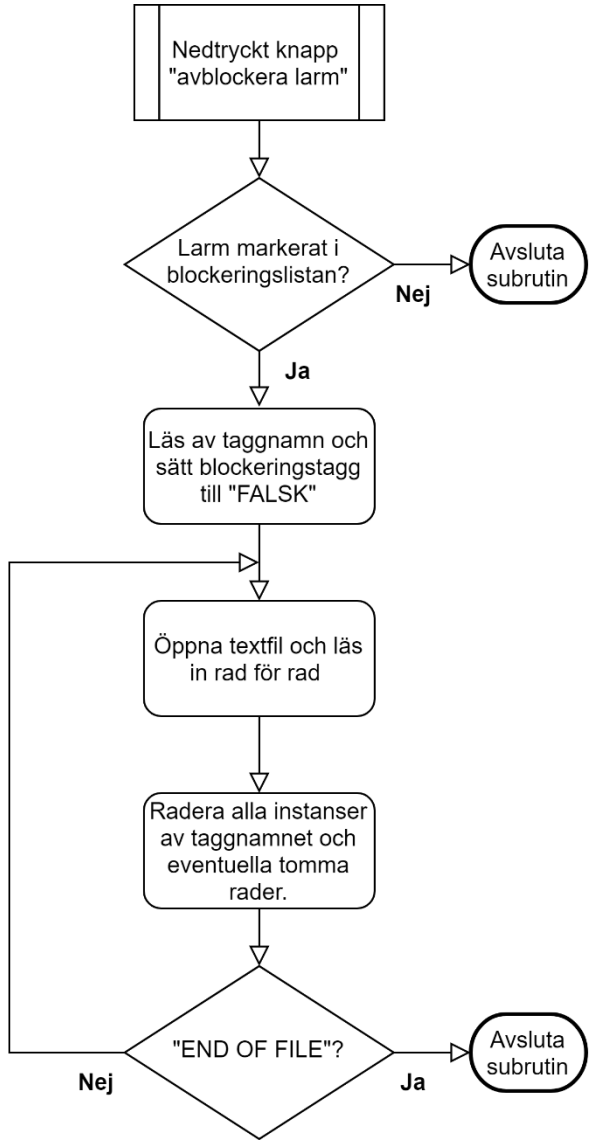
För att blockera ett larm måste man markera larmet och sedan trycka på “blockera larm” knappen. När knappen trycks ner läser man in larmets tagg i databasen och aktiverar den kopplade blockeringstaggen för det markerade larmet. Taggnamnet för blockeringstaggen skrivs sedan till en textfil. Detta är en modifiering av det befintliga programmet med tillägget att det blockerade larmet skrivs till en textfil i det sista steget. Scriptet beskrivs översiktligt med ett flödesschema i figur 5.2.



Figur 5.2: Lösning i klient med Visual Basic. Flödesschema av subrutinen för blockering

5.2.2. Radera avblockerade larm

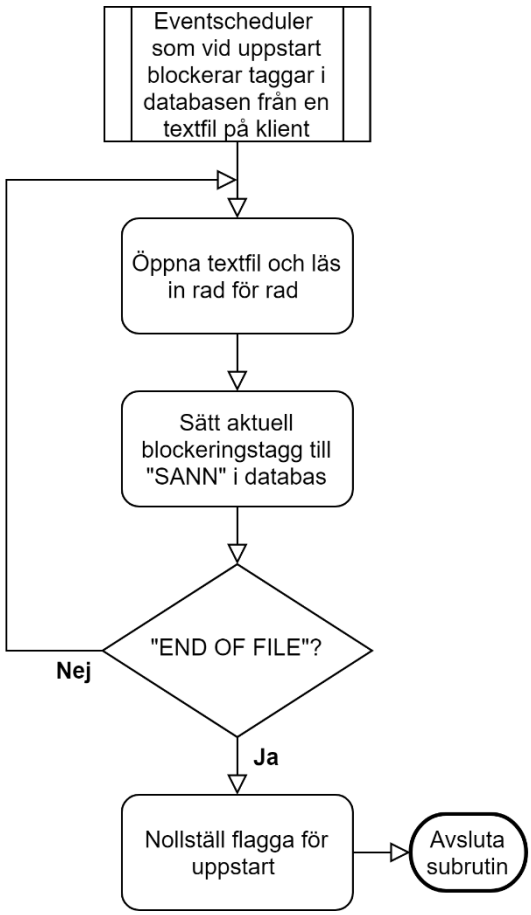
Funktionen att avblockera ett redan blockerat larm fungerar på ett nästan identiskt tillvägagångssätt som vid blockering av ett larm. Man markerar det önskade larmet och klickar sedan på knappen ”avblockera larm”, då nollställs blockeringstaggen för det valda larmet i databasen och larmet dyker upp som aktivt igen. Subrutinen öppnar sedan textfilen som innehåller alla blockerade taggar, där raderas sedan den nollställda blockeringstaggen. Detta är en modifiering av det befintliga programmet med tillägget att larmet som avblockerats raderas från en textfil i det sista steget. Scriptet beskrivs översiktligt med ett flödesschema i figur 5.3.



Figur 5.3: Lösning i klient med Visual Basic. Flödesschema av subrutinen för avblockering

5.2.3. Automatisk uppdatering av databas vid uppstart

I databasen finns en digital utgång som används som en flagga för att indikera när båda servrar har varit avstängda. Vid uppstart initieras flaggan som ”aktiv” vilket aktiverar ett event i iFIX. Detta event läser in hela textfilen med blockeringstaggar och uppdaterar dessa taggar i databasen. Samma larm som låg blockerade innan serveruppkopplingen försvann skall då hamna som blockerade när uppkopplingen återetableras. Slutligen återställs flaggan när eventet är exekverat. Scriptet beskrivs översiktligt med ett flödesschema i figur 5.4.



Figur 5.4: Lösning i klient med Visual Basic. Flödesschema av subrutinen för inläsning av blockerade larm vid uppstart

Denna lösning fungerade bra vid användning av enbart en klient men vid applicering i ett system med flera klienter skulle det innebära ett antal nackdelar och utmaningar. Detta i form av mycket kod som behöver läggas till på varje klient samt att textfilerna som finns på alla klienter behöver vara synkroniserade med varandra. Eftersom det verkliga systemet bestod av ett flertal klienter var denna lösning inte god nog.

5.3. Lösning 3 – Programmering i VBA på Servrarna

Det tredje förslaget till en lösning är mycket lik lösning 2, men med skillnaden att informationen om de blockerade larmen samlas på servrarna istället för klienterna. Lösningen är uppdelad i två liknande delar, en för blockering och en för avblockering.

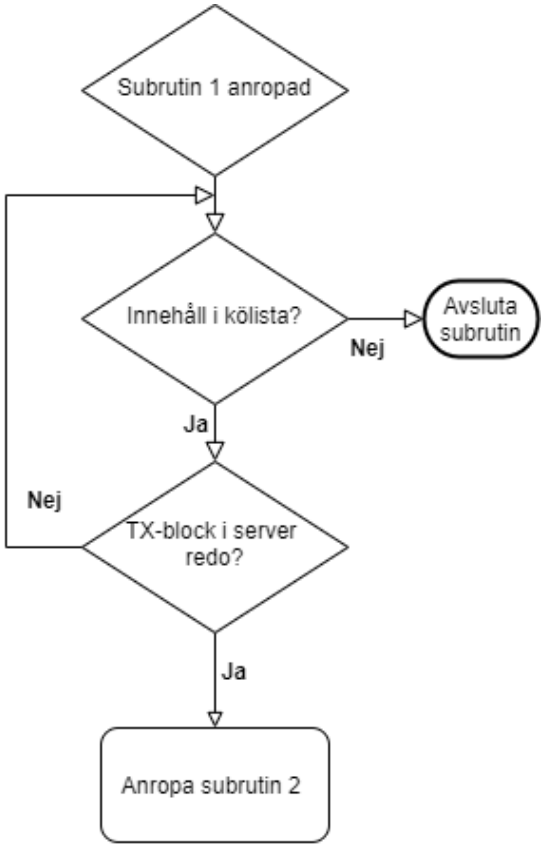
När ett larm blockeras från en klient sparas namnet på aktuellt larm i ett textblock i databasen. En skrivning till textblocket aktiverar en Scheduler på servern som skriver texten från textblocket till en separat textfil. När Schedulern är klar nollställs textregistret för att servern skall kunna ta emot nästa larm.

Vid avblockering skrivs taggnamnet till ett annat textblock och en annan Scheduler används som istället tar bort motsvarande text från textfilen.

Skrivningen från klientdatorn till databasen går snabbare än skrivningen från databasen till textfilen. Detta medför en risk att larm förloras vid blockering eller avblockering av flera larm i snabb följd. För att hantera den risken infördes en FIFO-funktion (First In First Out) för att kunna invänta servern och skriva först när den är redo. I följande delkapitel så beskrivs de lösningar som testades.

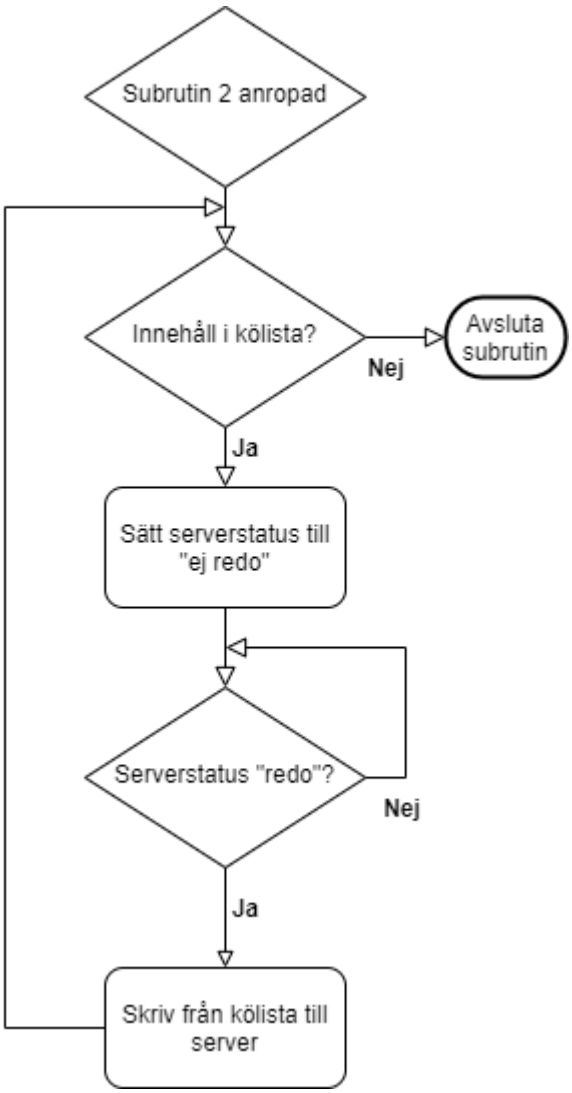
5.3.1. Subrutiner med While-loopar

Den första lösningen som testades var subrutiner med ett flertal If-satser och While-loopar. De flödesscheman som visas nedan i figur 5.5 och figur 5.6 beskriver programsekvenserna som utförs för att nå önskad funktionalitet.



Figur 5.5: Flödesschema över subrutin 1 för lösningen med While-loopar

Vid programmering av lösningen med While-loopar testades det först att lägga alla villkor i en och samma subrutin. Men då flaggorna som villkoren var kopplade till ändrades i If-satserna och While-looparna hängde sig programmet, vilket medförde ett behov av två subrutiner.



Figur 5.6: Flödesschema över subrutin 2 för lösningen med While-loopar

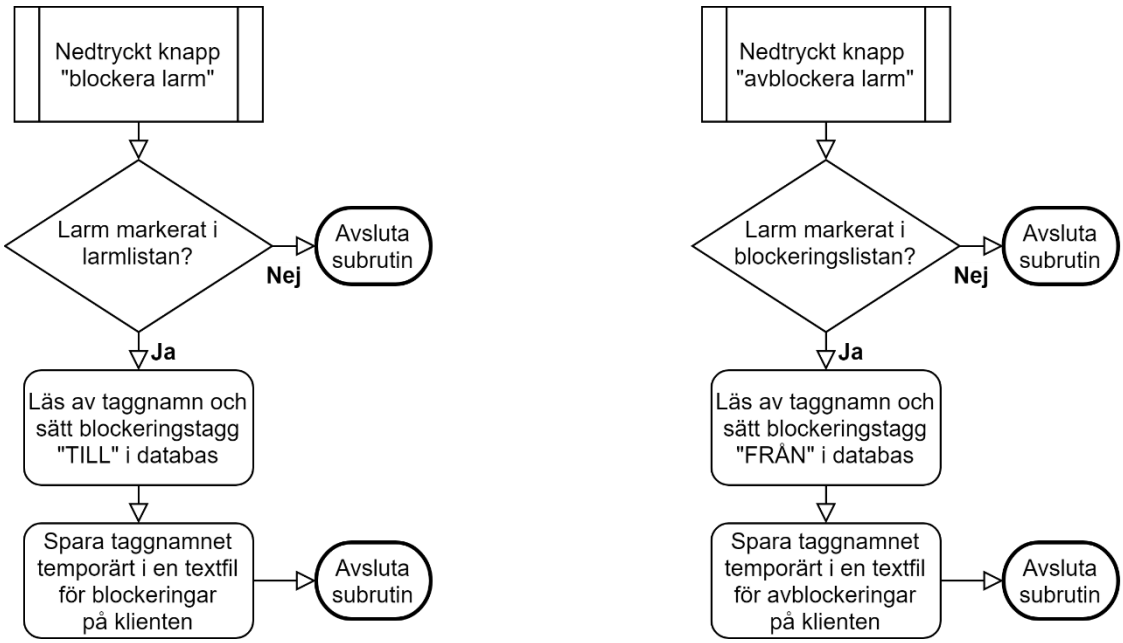
Lösningen med subrutiner och While-loopar löste problemet med att skrivningen till servern gick för fort. Men detta är inte en lämplig lösning, då hela iFIX-systemet blev långsammare och låste programmet tills dess villkoren för att gå vidare uppfyllts. Ett sådant utförande kan leda till oönskade konsekvenser för resterande funktioner i systemet, vilket inte kan accepteras då det påverkar tillförlitligheten.

5.3.2. Subrutiner med temporära textfiler på klienter

Den slutgiltiga lösningen är en vidareutveckling från ursprungsidén med Collections som beskrivs i kap 5.3.3.

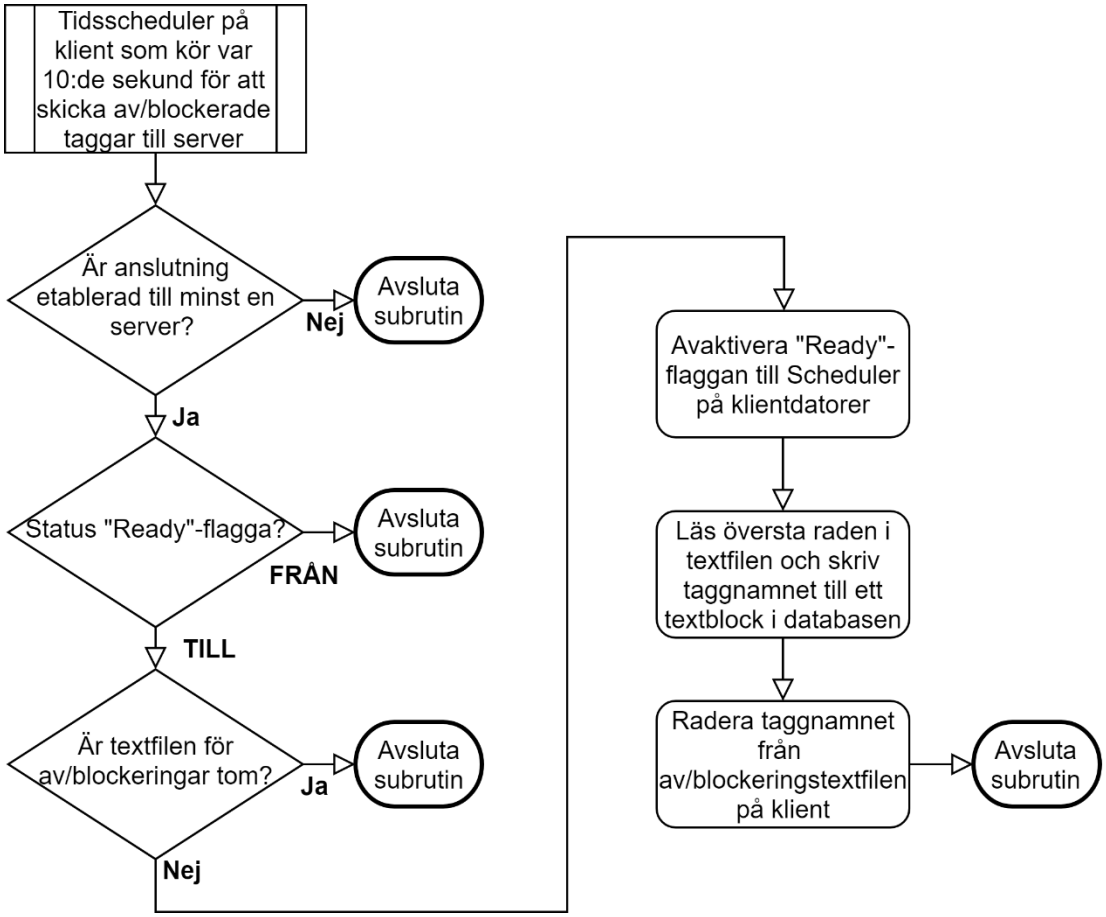
Istället för att temporärt spara de larmtaggar som blockeras i variabler så skrivs de till en textfil. Därefter så läses de av ett efter ett och skrivs till databasen samt tas bort från textfilen. För att kunna hålla isär vilka larm som skall blockeras samt avblockeras finns två textfiler på klienterna. För att inte problem skall uppstå om det både blockeras och avblockeras larm samtidigt så skrivs endast ett larm från klient till server åt gången.

Nedan beskrivs funktionerna i flödesdiagram och i figur 5.7 åskådliggörs funktionen för skrivning till textfilerna på klienten.



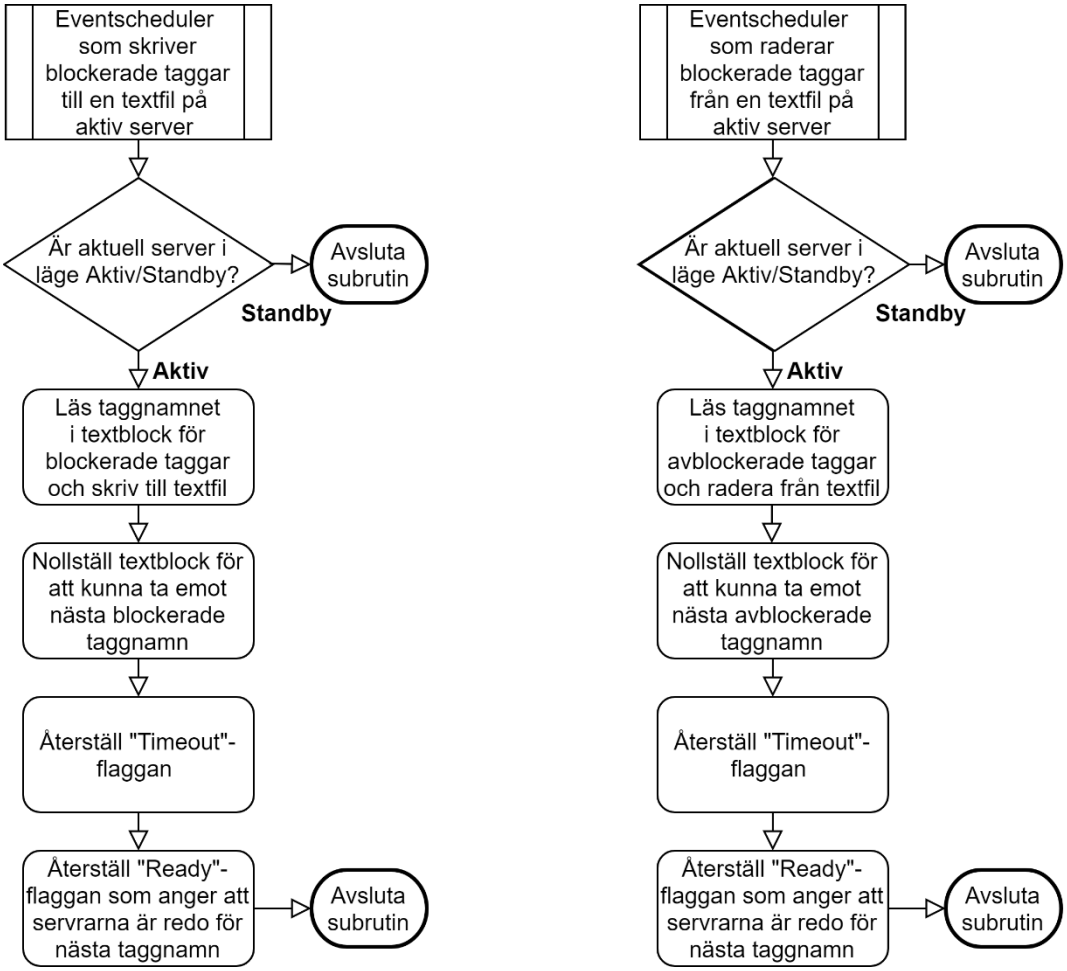
Figur 5.7: Flödesschema programkoden för blockering- och avblockeringsknapparna

Efter att ovanstående har genomförts kan programmet gå vidare till att skriva från klienten till servern. Det görs med en Scheduler som var tionde sekund skriver till databasen om den aktiva servern är redo samt om det finns något att skriva, som visas i figur 5.8. Anledningen till att 10 sekunder är valt beror på att skrivningarna annars sker för tätt för att synkroniseringen av textfilerna på servrarna skall fungera. Synkronisering av textfilerna beskrivs i efterföljande kapitel, 5.4.



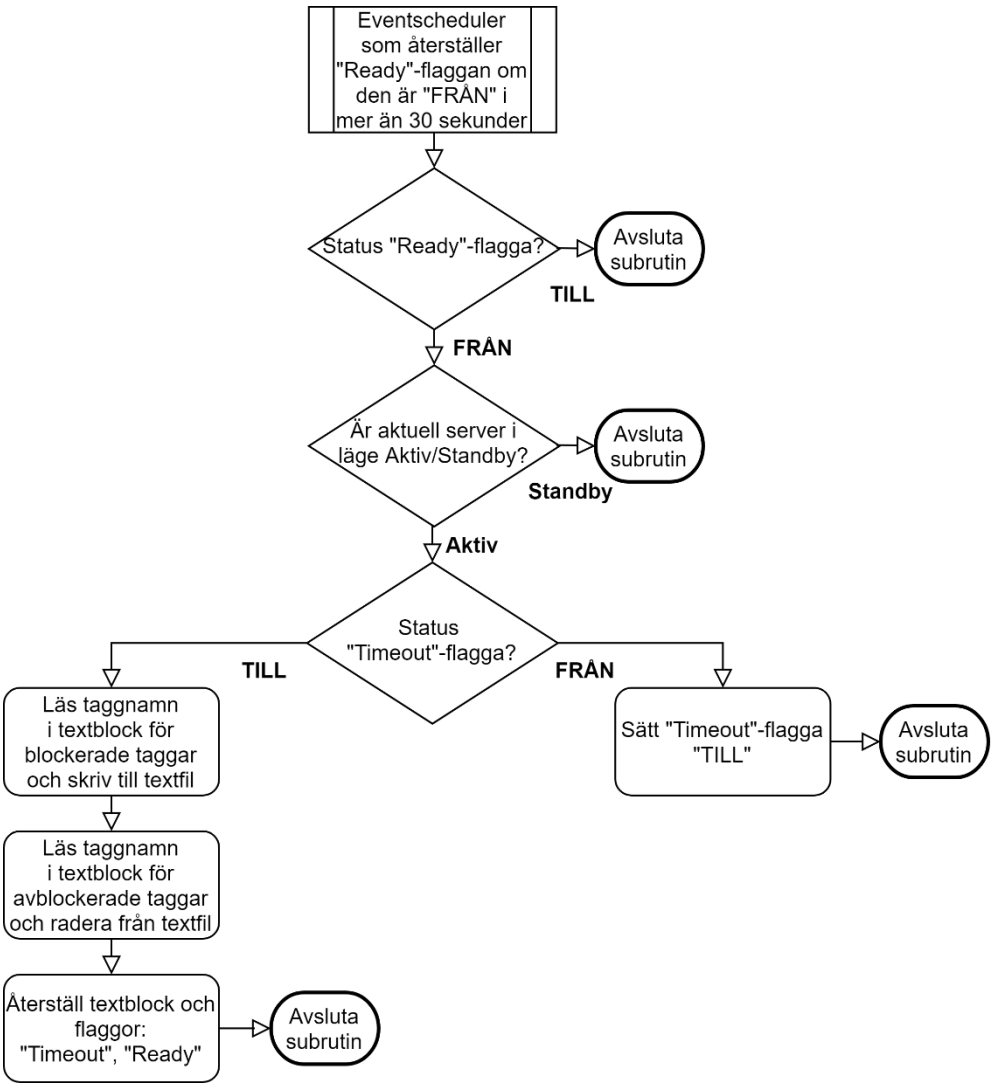
Figur 5.8: Flödesschema programkod för vidarebefordran av taggnamn från klient till server

Följande flödesscheman beskriver de funktioner som återfinns på servrarna. I figur 5.9 beskrivs programflödet för funktionerna som läser av taggarna från textblocken i databasen för att sedan skriva till, eller radera från textfilen som innehåller alla blockerade larm. Eventet aktiveras då textblocken får något innehåll som är skilt från ”0”, vilket är fallet när ett taggnamn har skrivits till något av blocken. Enbart den aktiva servern kommer utföra koden och detta kontroleras via en NSD-tagg. Innehållet i textblocket skrivs till textfilen, därefter återställs textblocket genom att ”0” skrivs till det. Då är textblocket redo för nästa skrivning från en klient och sist innan rutinen avslutas så återställs ”Timeout”-flaggan och ”Ready”-flaggan så klienterna får signalen att servern är redo för nästa skrivning.



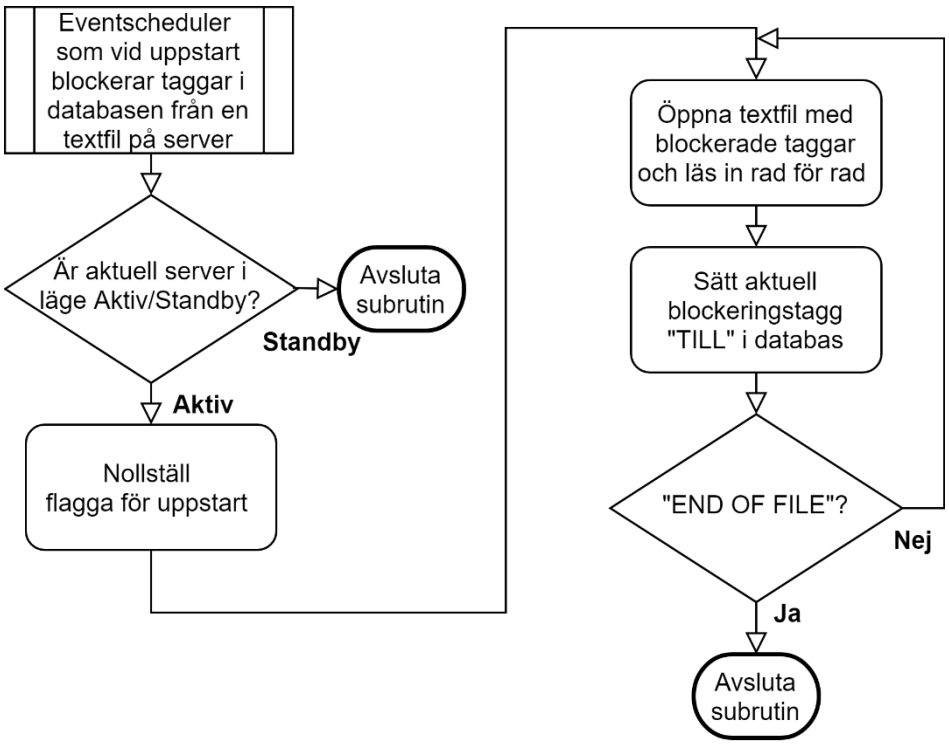
Figur 5.9: Flödesscheman för koden som skriver taggnamn till eller raderar taggnamn från textfilen på server

Flödesschemat i figur 5.10 nedan visar programflödet för Timeout-funktionen. Om flaggan ”Ready” är ”FRÅN” i mer än 30 sekunder så har något hänt, exempelvis en failover mitt under en skrivning. Därav behövs en felhantering för detta scenario. Schedulern är tidsbaserad och körs var 30:e sekund då den kontrollerar status på ”Ready”-flaggan. Enbart den aktiva servern kommer utföra koden. Därefter kontrolleras om ”Timeout” flaggan är ”TILL” eller ”FRÅN”. Är den ”FRÅN” så sätts den ”TILL” innan rutinen avslutas. Detta är för att avgöra om det verkligen har skett en Timeout eller inte. Om det har skett en Timeout så kommer flaggan vara ”TILL” nästa gång rutinen körs. Om innehåll finns i textblocket så läses det av för blockering och skrivs till textfilen, samma sak sker med textblocket för avblockering och motsvarande larm raderas. Sist så återställs ”Timeout” och ”Ready” flaggorna.



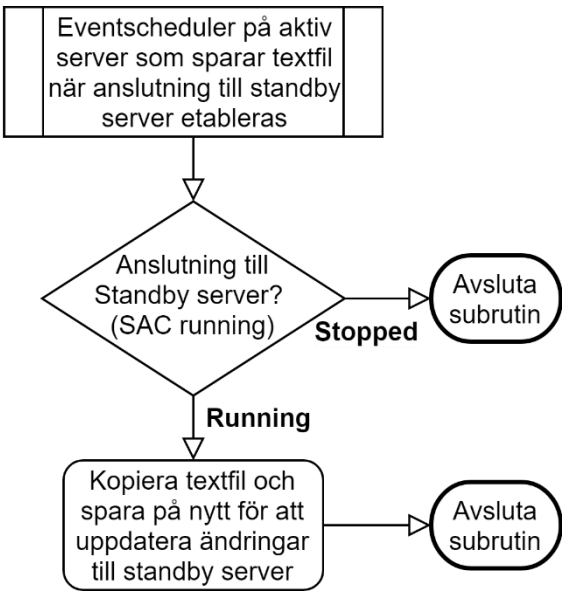
Figur 5.10: Flödesschema programkod för Timeout-funktion

Flödesschemat i figur 5.11 demonstrerar huvudsyftet med hela lösningen, det vill säga hur servrarna läser in blockerade larm efter att databasen blivit nollställd. I databasen finns en ”Startup”-flagga som alltid initieras som ”TILL” när databasen laddas in på nytt, detta sker när båda servrar har varit avstängda. På båda servrar finns en Scheduler som triggas när flaggan är ”TILL” men enbart den server som är aktiv kommer utföra hela programkoden. Den aktiva servern nollställer flaggan och läser sedan in och blockerar varje taggnamn som finns i textfilen.



Figur 5.11: Flödesschema programkod för inläsning av blockerade larm vid uppstart

I figur 5.12 demonstreras flödesschemat för Eventschedulern för synkronisering av textfil då den aktiva servern får anslutning till standby-servern. Vad denna funktion gör är att spara textfilen med samma innehåll för att få en uppdatering av filen. Anledningen till detta är att filerna som finns i mappen PDB endast synkroniseras när ändringar sker. Ändringar som sker i textfilen medan en server är avstängd skulle annars medföra en inaktuell version på standby-servern även efter uppstart. Därför är denna funktion nödvändig för att synka filen på nytt när servrarna får ny anslutning.



Figur 5.12: Flödesschema programkod för synkronisering av textfil mellan servrar vid uppstart av standby server

Denna lösning uppfyllde de krav som ställdes och innebar inga stora tillägg eller förändringar i ursprungskoden.

5.3.3. Subrutiner med Collections

Detta förslag till en lösning var ursprungsidén till den slutgiltiga lösningen i föregående kapitel (kap 5.3.2). För denna lösning används två listor med variabler på varje klientdator. Dessa listor kallas ”Collections” i VBA. I ena listan sparas taggnamnen för de larm som blockeras och i den andra sparas taggnamnen för de larm som avblockeras. På alla klienter finns även två tidsbaserade Scheduler som startar var tionde sekund. Den ena Schedulern läser av listan för blockeringar och den andra läser av listan för avblockeringar. När en Schedule läser av ett taggnamn i listan skickas taggnamnet vidare till ett av textblocken i databasen och raderas sedan från listan.

De Scheduler som används på klienterna kör i bakgrunden av iFIX. När en Schedule kör som en bakgrundstjänst så är den en helt egen process med egna minnesareor jämfört med iFIX Workspace där HMI-bilderna bearbetas. Av detta skäl har Schedulern ingen åtkomst till samma listvariabler som HMI-bilderna, där variablerna sparas. På grund av detta vidareutvecklades inte denna lösning. Istället användes samma tankesätt men med lokala textfiler som ersättning till Collections, och resultatet beskrivs i kap 5.3.2.

5.4. Delning av textfil mellan enheter

Något som är gemensamt för alla lösningar som utvecklats är att en textfil används för att spara informationen. Härav uppstår ett behov att dela textfilen mellan enheterna för att alla skall kunna uppdatera denna. Vilka enheter som behöver åtkomst skiljer sig mellan de två lösningarna som vidareutvecklats.

För lösning 2 behöver samtliga klienter få åtkomst till textfilen, för att lägga till och ta bort larm som blockerats eller avblockerats.

Flera potentiella metoder är möjliga för att åstadkomma gemensam åtkomst av textfilen. Man kan använda sig av en nätverkshårdisk som är kopplad till nätverket och spara textfilen på denna. Fördelen med en nätverkshårdisk är att det bara finns en enskild instans av textfilen, vilket resulterar i att det inte finns någon risk för osynkade dokument.

Man kan också dela en nätverksmapp ifrån den lokala klienten. Normalt skall denna klientdator alltid vara igång, vilket gör att alla kan få åtkomst till textfilen. Men det existerar ändå en viss risk att denna klientdator kan stängas av, detta skulle då resultera i att textfilen ej är åtkomlig och eventuella blockeringar och avblockeringar som utförs ej uppdateras i textfilen. När klientdatorn sedan startas och nätverksmappen återfår anslutning kommer textfilen att vara inaktuell. Detta kan lösas genom att dela en nätverksmapp med en textfil på varje klientdator. Varje förändring som utförs på blockeringstaggarna uppdateras från aktuell klientdator till samtliga textfiler. Om en klientdator har varit avstängd kommer textfilen på denna att vara inaktuell, härav uppstår ett behov av att synkronisera textfilen mot övriga vid uppstart. Som en följd av att man inför flera instanser av samma textfil på de olika enheterna införs också en stor risk att dokumenten ej stämmer överens. Därav förlorar lösningen egenskaper såsom tillförlitlighet och felsäkerhet.

För lösning 3 gäller att båda servrar har åtkomst till textfilen eftersom den aktiva servern kan växla mellan de två. Metoderna för att dela textfilen mellan servrarna är liknande som för lösning 1. Metoden att använda sig av en nätverkshårdisk blir identisk. Om man istället delar en nätverksmapp på respektive server kommer man likt lösning 1 fortfarande få flera instanser av samma textfil. Men eftersom det bara är två servrar så blir riskerna betydligt mindre för osynkroniserade dokument. En annan fördel med lösning 3 är när båda servrar är avstängda kommer sålunda även databasen vara avstängd och inga förändringar blir möjliga att göra tills någon av servrarna är uppstartad.

Det finns även ett tredje alternativ, vilket är alternativet som applicerats i den slutgiltiga lösningen. I iFIX finns en funktion där alla filer i en förbestämd mapp övervakas och alla filer som får en ändring synkroniseras från den aktiva till standby-servern [26]. Då filerna i denna mapp endast synkroniseras vid ändringar och från den aktiva till standby-servern krävdes funktioner för att hantera de fall en server var avstängd.

De fallen hanteras med en Scheduler som körs när servrarna får anslutning till varandra igen. Scriptet som då körs sparar textfilen som skall synkas på nytt med samma innehåll. Det tolkar datorn som en ändring vilket aktiverar synkroniseringen mellan servrarna och fallet är hanterat.

Risken för osynkroniserade textfiler finns fortfarande kvar då det är två kopior av samma textfil och om kommunikationen mellan servrarna sätts ur spel så kan problem uppstå. Men

det är ändå den bästa lösningen utan att tillföra ytterligare hårdvara till systemet, till exempel i form av en nätverkshårddisk eller server.

5.5. Risker med att göra ändringar i programkod

När man står inför uppgiften att modifiera befintlig kod kan det vara frestande att radera det gamla och skriva om nytt ifrån början. Anledningen till detta är att det generellt är enklare att skriva kod än att läsa den. Således innebär det inte att kod som vid en första anblick upplevs som rörig faktiskt är rörig. Vidare gäller att en rörig kod inte är dålig per automatik. Man måste även ta hänsyn till att en gammal kod har testats i verkligheten där buggar har upptäckt och byggts bort [27].

När man gör förändringar i programkoden för ett befintligt program finns det alltid en risk att införa oönskade effekter. Detta eftersom den ursprungliga koden inte är skapad med hänsyn tagen till de eventuella tillägg och förändringar som utförs vid senare tillfälle. O'Neill [28] skriver att 42% av alla fel kan kopplas till bristande spårbarhet från koden till design, krav eller slutprodukt. Av detta skäl är det av största vikt att programmeraren är insatt i befintlig kod och förstår den till fullo, samt att de modifieringar eller tillägg som skall utföras är väl genomtänkta.

Det är lika viktigt att dokumentera samt ta kopior på alla ändringar som görs för att kunna gå tillbaka till tidigare versioner. Om problem uppstår blir det enklare att se vad som gått fel och åtgärda det, eller gå tillbaka till tidigare kod. För att underlätta eventuell felsökning så är det även en god idé att göra en ändring åt gången och testa den innan fler ändringar utförs. Annars blir det snabbt väldigt komplext att felsöka då många olika saker kan orsaka problemen. Om det är programkod som andra kan tänkas jobba med är det även viktigt att följa programstrukturen, för att underlätta kommande ändringar och uppdateringar [29].

Att fel upptäcks i ett tidigt stadie är av största vikt för att minimera kostnaderna att åtgärda felen [30]. Kostnaden för att åtgärda de fel som tar sig vidare till nästa fas i utvecklingen kan vara över 10 gånger högre än om felen upptäckts tidigare [28]. Dessutom är det enklare att åtgärda felen i ett tidigt stadie, och i vissa fall kan det till och med vara omöjligt att hitta felorsaken i den färdiga produkten då felen maskeras [30].

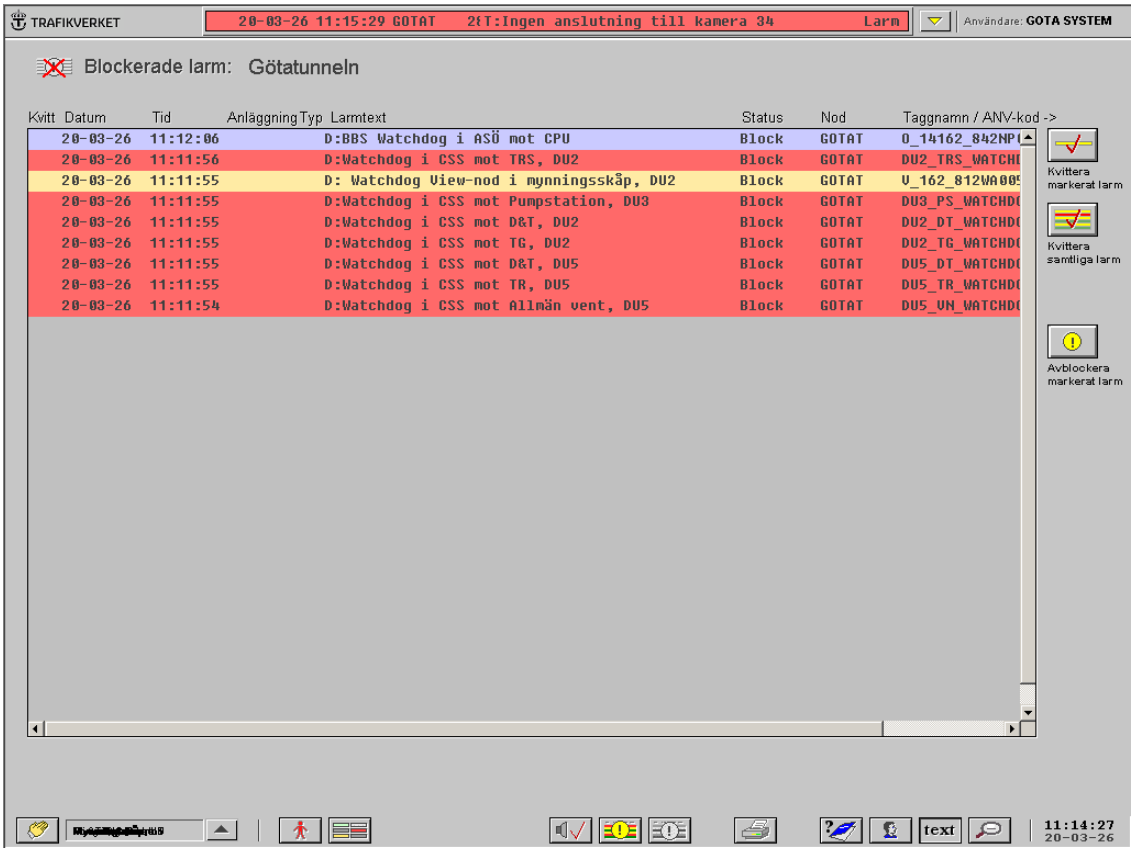
Vid programmering under projektet har dessa viktiga aspekter varit centrala. Först och främst studerades det befintliga programmets uppbyggnad och den tillhörande dokumentationen, för att ge en förståelse innan ändringar och tillägg började göras. Dokumentationen bestod av iFIX-beskrivningar, dokumentation från skapandet av programmet samt Microsofts webbsidor om Visual Basic. När det väl påbörjades infördes endast en ändring åt gången, för att kunna verifiera lösningen och se att det inte påverkade befintligt program. Det fanns inget smidigt sätt att ta totala backuper, så för att hålla isär ändringarna från det befintliga programmet kommenterades koden väl för att kunna återgå till tidigare versioner.

5.6. Kommentarfunktion för blockerade larm

En av funktionerna som efterfrågades av Midroc var möjligheten att lägga till kommentarer till de larm som blockeras. Kommentarer som beskriver vem som blockerat larmet, när och varför hade förbättrat dokumentationen och historik gällande larmhantering. I dagsläget finns ingen sådan funktion i iFIX 5.5, färdigt stöd för det kom först i senare versioner.

Tanken var att lägga till ytterligare en kolumn för kommentarer i HMI-bilden som visas i figur 5.13. Men att modifiera befintligt program för att möjliggöra kommentarer hade

inneburit en större förändring i koden än vad som behövts för att lösa ursprungsproblemet. Detta eftersom det enbart var en meriterande funktion och inte ett grundkrav för ursprungsproblemet. Att inte göra fler förändringar än absolut nödvändigt har varit av största prioritet för att minimera antal fel. Dessutom är målet med detta arbete att det skall komma till användning och större förändringar leder till mindre chans att lösningen appliceras på verkligt system. Därför har en kommentarfunktion inte utvecklats.



Figur 5.13: Översikt av blockerade larmlistan och dess kolumner

6. Test av lösning

För verifiering av lösningen genomfördes testprotokollen presenterade i kap 3.3.

Den lösning som testerna utfördes på är den slutgiltiga som beskrivs i kapitel 5.3.2, där larmen sparas temporärt i en textfil på klienterna. Som tidigare nämnt består denna lösning av ett antal olika funktioner som samverkar och tillsammans bildar en lösning. De funktioner som lösningen består av är:

- Ett script som temporärt lagrar blockeringstaggen som skall blockeras eller avblockeras i en textfil på klientdatorn.
- En Schedule som skriver från ovan nämnd textfil till ett textblock i databasen.
- En Schedule som skriver från ovan nämnt textblock till slutgiltig textfil på server.
- En Schedule som återställer systemet vid eventuell timeout.
- En Schedule som läser från slutgiltiga textfilen vid systemuppstart och aktiverar de blockeringstaggar som finns i textfilen.
- En Schedule som säkerställer synkroniseringen av den slutgiltiga textfilen mellan servrarna.

Testerna utfördes i den upprättade testanläggningen och har alltså inte gjorts i det verkliga systemet i Götatunneln.

Testprotokoll 1 - 4 har genomförts med gott resultat, som redovisas i bilaga 1-4. Då alla tester gav förväntat resultat för de testade funktionerna kan lösningen anses fungera korrekt. Testprotokoll 5 genomfördes aldrig då det i testanläggningen endast är en klient inkopplad och förutsättningarna för testet kunde ej uppfyllas. Därför kan inte korrekt funktionalitet av lösningen i sin helhet verifieras.

Ett scenario som lösningen inte kan hantera är fallet då en server kör utan anslutning till den andra under en längre tid och larm blockeras eller avblockeras. Om den server som körs då stängs ner och systemet sedan startar upp på den andra så förloras de ändringarna. Anledningen till det är att den server som varit avstängd och innehåller den icke-uppdaterade textfilen kommer bli aktiv och sedan skriva över den uppdaterade filen på den första servern när den ansluts igen. Det är som vi förstått det inte troligt att det scenariot uppstår, men om det gör det så kan blockerade larm förloras.

7. Resultat och diskussion

I detta kapitel diskuterar vi vad vi kommit fram till samt diskuterar om frågeställningarna i kapitel 1.4 är besvarade eller ej.

7.1. Besvarande av frågeställningen

- Vilka metoder finns för att hantera och spara undan larm i iFIX 5.5 och vilka är för/nackdelar med dessa?

Under projektets gång så har flera idéer på olika lösningar dykt upp och dessa redovisas i kapitel 5. Ett fåtal av lösningarna har varit lyckade, medan andra har avfärdats efter att vi har prövat oss fram samt läst i manualerna till iFIX och VBA. Alla lösningar som är beskrivna i rapporten utgick vi från att de skulle fungera innan vi blev bevisade motsatsen.

Lösningen med programblock i databasen var den idé som vi hade störst förhoppningar på och även den vi började att utveckla. Denna lösning var den som innebar minst ändringar i befintligt program då den endast skulle implementeras på servrarna, och således inte medföra någon förändring på klienterna. Vid informationssökning i dokumentationen om databasens funktioner så verkade det finnas goda möjligheter att bygga en lösning med programblock. Men programblocken fungerade inte i praktiken som de var beskrivna och vi fick kassera idén.

Den andra lösningen som utvecklades var lösning 2, vilken innebar att larmen sparades lokalt på klientdatorerna. Det är den mest simpla lösningen i VBA som vi kom fram till vad gäller ändringar i befintligt program. Men utmaningarna att dela textfilen mellan klienter och samtidigt säkerställa att den uppdateras korrekt innebar för många riskmoment, så även denna lösning valdes bort.

Det var utmaningarna med att dela textfilen som gav idén om att lagra larmen på servrarna istället, då den endast behöver delas och uppdateras av två enheter. Den stora utmaningen med den slutgiltiga lösningen (lösning 3) var att skrivning från klienter till databasen gick snabbare än skrivningen från databas till textfil. Vilket ledde till, i jämförelse med lösning 2, mycket mer ny kod och tillägg i systemet. Bortsett från nackdelen att mycket kod och taggar har tillagts uppfyller lösningen de krav vi hade på en eventuell lösning.

Eftersom lösningen med Collections var helt inbyggd i iFIX utan externa filer på klientdatorerna, hade detta troligtvis varit en bättre lösning än med textfilerna om den varit möjlig att genomföra enkelt. Under utvecklingen kontaktade vi iFIX support om möjliga metoder att dela åtkomst till samma variabler från olika processer. Dem gav då ett förslag till en eventuell lösning på problemet. Förslaget innebar att skapa ett VB6 ActiveX-program som ”instansieras” från båda processerna, vilket skulle möjliggöra delning av globala Collections mellan processerna. Men eftersom kunskapen inom området saknades så hade vi inte möjlighet att genomföra förslaget.

- Hur lägger man till möjligheten att kommentera blockerade larm?

Funktionen att kommentera blockerade larm hade varit ett bra tillägg i styrsystemet för att förbättra översikt och dokumentation. Men vi valde att inte utveckla denna funktion eftersom det hade inneburit en större förändring än nödvändigt på ursprungsprogrammet, och skulle resultera i en lägre chans att den utvecklade lösningen faktiskt appliceras och används. Vidare gäller att utveckling av funktionen hade krävt mer tid och möjligen äventyrat vår deadline för projektet.

- Hur fungerar synkroniseringen mellan servrarna?

Synkroniseringen av servrarna beskrivs i teoriavsnittet 2.10.4. Den slutgiltiga lösningen använder sig specifikt av synkroniseringen vid uppdatering av textfil mellan servrar.

- Hur är servrarna fysiskt kopplade till varandra och resterande utrustning?

I kapitel 4 beskriver vi i detalj hur all utrustning är sammankopplad och därför anser vi att denna fråga är besvarad. Värt att notera är dock att testanläggningen skiljer sig från verklig anläggning även om vi försökt att spegla systemet i största möjliga mån. Detta är en verklighet när man bygger upp en testmiljö då det är omöjligt att få till en exakt kopia utan att faktiskt bygga en ny anläggning.

- Vilka potentiella risker tillkommer med att modifiera befintlig programkod samt införa ny kod?

Risker vi funnit som berör modifiering av programkod redovisas i kapitel 5.5. De risker och tips som vi hittat har varit ett bra stöd för oss under projektets genomförande för att undvika och minimera antalet fallgropar som berör programmering. På grund av tidsbrist har vi inte gjort någon djupare undersökning om vilka metoder som normalt används vid mjukvaruutveckling för att felsöka och verifiera programkod.

7.2. Utvecklingsmöjligheter

Om det funnits möjlighet under projektet att installera ytterligare hårdvara till systemet så hade detta gjorts i form av en nätverkshårddisk på synkroniseringsnätverket mellan servrarna. Då skulle båda servrarna haft åtkomst till samma textfil och riskerna med att ha två instanser av samma fil eliminerats. Det hade även åtgärdat risken som kommenterats i kapitel 6, när ändringar sker på en server som inte har anslutning till den andra. Därför hade detta troligtvis varit den absolut bästa lösningen för lagring av textfilen innehållande de blockerade larmen.

Utöver att besvara frågeställningen hade vi funnit det intressant att beräkna de faktiska kostnaderna som uppkommer vid ett stopp som orsakats av styrsystemet, eller från trafikstörningar som uppstår vid åtgärdande av fel på utrustning. Men en sådan beräkning är svår att utföra eftersom den data som krävs är svår att hitta exakt, annat än uppskattning. Vidare har vi inte funnit några rapporter om olyckor som orsakats av styrsystemet i tunnlarna, inte heller hur stor påverkan som underhållsarbete av tunnlarnas styrsystem påverkar trafiken.

Slutligen anser vi att projektet har lett fram till ett gott resultat i form av en tilläggsfunktion som förbättrar larmöversikten för operatörerna vilket leder till ökad säkerhet och förbättrad arbetsmiljö. Funktionen leder också till reducering av onödiga underhållskostnader.

Referenser

[1] Regeringen, “Regeringen,” 2019. [Online]. Available: https://www.regeringen.se/4adae5/contentassets/c689564aa19c4d29bcebb1c037a2e37b/utgiftsomrade-22-kommunikationer.pdf.

[2] M. McQuillan, “What Is SQL Server,” in *Introducing SQL Server*, Berkley, Apress, 2015, pp. 1-5.

[3] H. S. Oluwatosin, “Client-Server Model,” *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67-71, 2014.

[4] M. McQuillan, “Database Basics,” in *Introducing SQL Server*, Berkley, Apress, 2015, pp. 33-51.

[5] HP, “HP Customer Support,” [Online]. Available: https://support.hp.com/us-en/document/c02053303?jumpid=reg_r1002_uken_c-001_title_r0002. [Accessed April 2020].

[6] K. Erickson, “Programmable Logic Controllers,” *IEEE Potentials*, vol. 15, no. 1, pp. 14-17, 1996.

[7] Communication Technologies, Inc., “Supervisory Control and Data,” Chantilly, 2004.

[8] Trafikverket, “Trafikverket.se,” [Online]. Available: https://trafikverket.ineko.se/Files/sv-SE/10347/RelatedFiles/100348_nationellt_trafikledningsst%C3%B6d_nts.pdf. [Accessed Mars 2020].

[9] OPC Foundation, “What is OPC?,” [Online]. Available: https://opcfoundation.org/about/what-is-opc/. [Accessed Mars 2020].

[10] Aten International Co., “What is KVM,” [Online]. Available: https://www.aten.com/ext_data/eu_en/solution/what_is_kvm/what_is_kvm.htm. [Accessed Mars 2020].

[11] Aten International Co., “8-Port PS/2-USB VGA KVM Switch with Daisy-Chain Port and USB Peripheral Support,” [Online]. Available: https://www.aten.com/global/en/products/kvm/rack-kvm-switches/cs1708a/. [Accessed Mars 2020].

[12] Aten International Co., “USB VGA Cat 5 KVM Extender (1280 x 1024@150m),” [Online]. Available: https://www.aten.com/global/en/products/kvm/kvm-extenders/ce700a/. [Accessed Mars 2020].

[13] GE Intelligent Platforms Inc., “Introduction to iFIX,” in *Understanding iFIX*, GE Intelligent Platforms Inc., 2012, pp. 3-5.

[14] General Electric Company, Common Licensing Quick Start Guide, General Electric Company, 2018.

[15] GE Intelligent Platforms, Inc., “HMI and SCADA Functions,” in *Understanding iFIX*, 2012, pp. 22-23.

[16] GE Intelligent Platforms, Inc., “Database Synchronization for Enhanced Failover,” in *Enhanced Failover*, 2012, p. 14.

[17] GE Intelligent Platforms, Inc., “Understanding a Database,” in *BUILDING A SCADA SYSTEM*, 2012, pp. 3-7.

[18] M. Rouse, “DEFINITION: Script,” [Online]. Available: https://whatis.techtarget.com/definition/script. [Accessed April 2020].

[19] D. Hemmendinger, “Computer scripting language,” 2 Februari 2015. [Online]. Available: https://web.archive.org/web/20160425225434/https://www.britannica.com/technology/computer-scripting-language. [Accessed April 2020].

[20] GE Intelligent Platforms, Inc., “Visual Basic for Applications (VBA),” in *UNDERSTANDING I FIX*, 2012, pp. 5-6.

[21] GE Intelligent Platforms, Inc., “Scheduler,” in *UNDERSTANDING I FIX*, 2012, p. 13.

[22] GE Intelligent Platforms, Inc., “Scheduler,” in *MASTERING I FIX*, 2012, pp. 3-34.

[23] GE Intelligent Platforms, Inc., “Understanding the Proficy iFIX WorkSpace,” in *UNDERSTANDING I FIX*, 2012, pp. 27-33.

[24] MSB, “Myndigheten för samhällsskydd och beredskap,” Augusti 2015. [Online]. Available: https://rib.msb.se/filer/pdf/27940.pdf. [Accessed Maj 2020].

[25] Trafikverket, “Trafikverket,” 1 April 2018. [Online]. Available: https://www.trafikverket.se/contentassets/4b1c1005597d47bda386d81dd3444b24/asek-6.1/asek_6_1_hela_rapporten_180412.pdf. [Accessed Maj 2020].

[26] General Electric Company, Enhanced Failover, 2013, p. 2.

[27] J. Spolsky, “Things You Should Never Do, Part One,” in *Joel on Software*, Berkley, Apress, 2004, pp. 183-187.

[28] D. O'Neill, “Issues in software inspection,” *IEEE Software*, vol. 14, no. 1, pp. 18-19, 1997.

[29] J. Aycock, Reading and Modifying Code, 2008, pp. 19-25.

[30] A. Aurum and H. W. C. Petersson, “State-of-the-Art: Software Inspections after 25 Years,” *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133-154, 2002.

Bilaga 1 – Blockering/avblockering av larm

Test 1: blockering/avblockering av larm

Förutsättningar: Servrar anslutna till Viewklient, fixbackgroundserver är igång på alla enheter

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	Blockera valfritt larm	Larm blockeras, motsvarande blocktagg sparas i txt-fil på aktiv server	OK	JA/EI
2	Avblockera valfritt larm	Larm avblockeras, motsvarande blocktagg tas bort från txt-fil på aktiv server	OK	JA/EI
3	Blockera fem valfria larm i snabb följd	Fem larm blockeras, motsvarande blocktaggar sparas i txt-fil på aktiv server	OK	JA/EI
4	Avblockera fem valfria larm i snabb följd	Fem larm avblockeras, motsvarande blocktaggar tas bort från txt-fil på aktiv server	OK	JA/EI
5	Blockera alla larm i snabb följd	Alla larm blockeras, motsvarande blocktaggar sparas i txt-fil på aktiv server	OK	JA/EI
6	Avblockera alla larm i snabb följd	Alla larm avblockeras, motsvarande blocktaggar tas bort från i txt-fil på aktiv server	OK	JA/EI
7	Skriv en rad som är över 80 tecken på första raden i Blocklarm.txt	Meddelanderuta skall dyka upp med hänvisning till blockeringsfilen	OK	JA/EI
8	Skriv en rad som är över 80 tecken på första raden i Unblocklarm.txt	Meddelanderuta skall dyka upp med hänvisning till avblockeringsfilen	OK	JA/EI

Bilaga 2 – Synkronisering av txt-fil mellan servrar

Test 2: Synkronisering av txt-fil mellan servrar

Förutsättningar: Servrar anslutna till varandra, SAC körs på båda servrar med fungerande kommunikation

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	GOTA_C1 aktiv, GOTA_C2 backup. Stäng av GOTA_C2, blockera fem larm, starta upp GOTA_C2	Identisk txt-fil som på GOTA_C1 med de fem motsvarande larmen skall finnas på GOTA_C2 efter uppstart	OK	JA/EI
2	GOTA_C2 aktiv, GOTA_C1 backup. Stäng av GOTA_C1, blockera fem larm, starta upp GOTA_C1	Identisk txt-fil som på GOTA_C2 med de fem motsvarande larmen skall finnas på GOTA_C1 efter uppstart	OK	JA/EI
3	GOTA_C1 aktiv, GOTA_C2 backup. Stäng av GOTA_C2, avblockera fem larm, starta upp GOTA_C2	Identisk txt-fil som på GOTA_C1, de fem avblockerade larmen skall vara raderade på GOTA_C2 efter uppstart	OK	JA/EI
4	GOTA_C2 aktiv, GOTA_C1 backup. Stäng av GOTA_C1, avblockera fem larm, starta upp GOTA_C1	Identisk txt-fil som på GOTA_C2, de fem avblockerade larmen skall vara raderade påGOTA_C1 efter uppstart	OK	JA/EI
5	Starta om både GOTA_C1 och GOTA_C2	Txt-fil på GOTA_C1 och GOTA_C2 skall vara identisk före och efter omstart	OK	JA/EI

Bilaga 3 – Omstart av servrar och inläsning av blockerade larm från txt-fil

Test 3: Omstart av servrar och inläsning av blockerade larm från txt-f

Förutsättningar: Larmtaggar återfinns i txt-fil, båda servrar startas om ungefär samtidigt

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	GOTA_C1 aktiv, GOTA_C2 backup. Starta om GOTA_C1 först, sedan GOTA_C2.	Motsvarande larm i txt-fil skall blockeras i iFIX	OK	JA/EI
2	GOTA_C2 aktiv, GOTA_C1 backup. Starta om GOTA_C2 först, sedan GOTA_C1.	Motsvarande larm i txt-fil skall blockeras i iFIX	OK	JA/EI
3	GOTA_C1 aktiv, GOTA_C2 backup. Starta om GOTA_C2 först, sedan GOTA_C1.	Motsvarande larm i txt-fil skall blockeras i iFIX	OK	JA/EI
4	GOTA_C2 aktiv, GOTA_C1 backup. Starta om GOTA_C1 först, sedan GOTA_C2.	Motsvarande larm i txt-fil skall blockeras i iFIX	OK	JA/EI

Bilaga 4 – Timeout funktion på servrar

Test 4: Timeout funktion på servrar

Förutsättningar: Servrar anslutna till Viewklient, fixbackgroundserver är igång på alla enheter

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	GOTA_C1 aktiv. Sätt tagg GOTA_BLOCK_QUEUE "FRÅN" i databas	Timeout funktionen skall återställa ("TILL") tagg: GOTA_BLOCK_QUEUE	OK	JA/EI
2	GOTA_C2 aktiv. Sätt tagg: GOTA_BLOCK_QUEUE "FRÅN" i databas	Timeout funktionen skall återställa ("TILL") tagg: GOTA_BLOCK_QUEUE	OK	JA/EI
3	GOTA_C1 aktiv. Simulera att Schedulern för blockering har hängt sig genom att stoppa GOTA_larm_blockering på aktiv server. Blockera sedan ett antal larm från klient	Motsvarande larm som blockerats på klient ska återfinnas i txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout	OK	JA/EI
4	GOTA_C2 aktiv. Simulera att Schedulern för blockering har hängt sig genom att stoppa GOTA_larm_blockering på aktiv server. Blockera sedan ett antal larm från klient	Motsvarande larm som blockerats på klient ska återfinnas i txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout	OK	JA/EI
5	GOTA_C1 aktiv. Simulera att Schedulern för avblockering har hängt sig genom att stoppa GOTA_larm_avblockering på aktiv server. Avblockera sedan ett antal larm från klient	Motsvarande larm som avblockerats på klient ska raderats från txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout	OK	JA/EI
6	GOTA_C2 aktiv. Simulera att Schedulern för avblockering har hängt sig genom att stoppa GOTA_larm_avblockering på aktiv server. Avblockera sedan ett antal larm från klient	Motsvarande larm som avblockerats på klient ska raderats från txt-fil på server genom Timeout schedulern: GOTA_larm_tagg_timeout	OK	JA/EI

Bilaga 5 – Funktionalitet i fullständigt system

Test 5: Funktionalitet i fullständigt system

Förutsättningar: Alla klienter kopplade till systemet, all kod uppdaterad, fixbackgroundserver är igång på alla enhe

Steg	Beskrivning	Förväntat resultat	Händelse	Kvittens
1	Blockera larm från flera klienter samtidigt	Motsvarande larm i txt-fil skall blockeras i iFIX		
2	Avblockera larm från flera klienter samtidigt	Motsvarande larm i txt-fil skall avblockeras i iFIX		