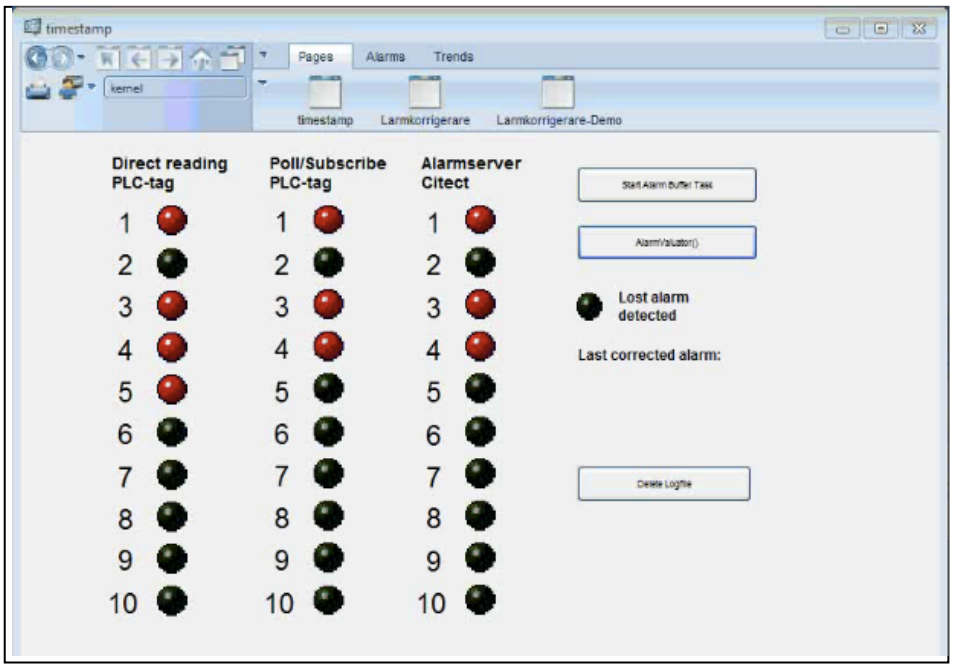
CHALMERS



Driftsäker larmhantering mellan PLC och CitectSCADA

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

PHILIP FORSTING

Institutionen för signaler och system
Avdelningen för reglerteknik, automation och mekatronik.
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige, 2013

FÖRORD

Detta examensarbete har utförts som avslutande moment inom mekatronikingenjörs-programmet på Chalmers Tekniska Högskola. Arbetet utfördes på AcobiaFLUX i Göteborg.

Jag vill passa på att tacka Martin Bonnevier på AcobiaFLUX för feedback på arbetet och stöttning i tekniska frågor.

Ett särskilt tack till uppdragsgivare Martin Esping för sin tilltro och sitt stöd.

Till sist vill jag ge ett stort tack till Morgan Osbeck vid institutionen Signaler och System för sin ovärderliga handledning under arbetets alla faser.

Göteborg den 8 februari 2013.

Philip Forsting

SAMMANFATTNING

Då en PLC kommunicerar och övervakas av ett SCADA-system finns ett behov av att PLC:n ska kunna skicka larm till SCADA-systemet så att en operatör kan underrättas om något oväntat inträffar. En sådan metodik existerar redan men den är inte helt vattentät eftersom ett överföringsfel som inträffar under ett kritiskt skede under avläsningen kan medföra att larmet tappas helt och är förlorat för alltid. För att eliminera den risken har ett larmkontrollprogram konstruerats som är uppbyggt att upptäcka och korrigera larmdifferenser mellan en PLC och dess övervakande SCADA-system, i detta fall av fabrikat Citect.

Arbetet har utförts på AcobiaFLUX i Göteborg som har fått uppdrag att byta ut det styrsystem som används på Öresundsbron. Programmet är bl.a. tänkt att användas av de system som styr trafikövervakning, trafikljus och tunnelventilation.

Programmet är skrivet i Citects egna programmeringsspråk Cicode och gränssnittet är utformat till att representera den larmanalys som sker i larmkontrollen. Om ett larm har tappats blir operatören uppmärksammad genom en varningstext i displayen och mer detaljerad information om felets uppkomst sparas i en loggfil.

För att testa programmet har nätstörningar simulerats genom larmgenerering i PLC samt bortkoppling av den befintliga handskakningsmetodiken. Programmet har klarat av att detektera och korrigera de förlorade larmen utan svårigheter.

SUMMARY

As a PLC communicates with a supervising SCADA system there is a need for the PLC to send alarms to the SCADA system so that an operator can be notified if something unexpected occurs. There is already an existing method of transmission but it is not completely reliable since an alarm can be lost because of an error during the transfer. If an alarm gets lost, it will be irretrievable. To eliminate the risk of losing an alarm, a control program has been designed to detect and correct alarm differentials between the PLC and its supervisory SCADA systems, in this case manufactured by Citect.

The project has been performed at AcobiaFLUX in Gothenburg. The company has been given the task to replace the control system operating at the Oresund Bridge. The program is meant to be used by the systems that control traffic surveillance, traffic lights and tunnel ventilation.

The program is written in Citect's own programming language Cicode and the interface is designed to represent the alarm analyzing process during the alarm inspection. If an alarm has been lost a warning sign on the screen will notify the operator and further information is saved in a log file.

To test the program, network disruptions have been simulated by generating alarms in the PLC and by disconnecting the existing handshake process. The program has been able to detect and correct the lost alarms without difficulty.

INNEHÅLLSFÖRTECKNING

BETECKNINGAR

Bit	Digitalt värde. 1 eller 0
Byte	8 bitar. $0 – 255_{10}$.
Cell	Sammanslagning av byte. Jämför med word
Gantt	En typ av flödesschema som används under projektledning
HMI	Human Machine Interface. Sv. Användargränssnitt
IBAN	International Bank Account Number. Sv. Internationellt bankkontonummer
PC	Persondator
PLC	Programmable logic controller. Sv. Programmerbart styrsystem
SCADA	Supervisory Control And Data Acquisition. System för styrning och övervakning av processer
XOR	Exclusive OR. Sv. Exklusiv ELLER-operation

1 INLEDNING

1.1 Bakgrund

AcobiaFLUX är ett konsultföretag inom automation och industriell IT som har flera uppdrag inom industri, infrastruktur och fastighetsautomation. De har fått i uppdrag av företaget Öresundsbron att byta ut hela det styr- och reglersystemet som analyserar trafikflödets mätdata vilket bland annat används till att reglera trafikljus, vägbelysning och ventilations-fläktar. Styrsystemet består av flera samverkande PLC:er som en operatör kan interagera med via ett CitectSCADA-system. Skulle någon mätgivare visa ett otillåtet värde, t.ex. att det blåser för kraftiga vindar över bron, kommer PLC:n att generera ett larm som skickas till SCADA-systemet.

Det finns ett handskakningssytem mellan PLC och Citect som har hand om överföringen av larm, trender och event. Överföringen är meddelandebaserad och har checksumma för att kontrollera att informationen som sänds är korrekt. Det finns en viss risk att Citect kan ”förlora” ett meddelande som sänds ifrån PLC, t ex om nätuppkopplingen tappas under ett kritiskt skede i handskakningen. Skulle det ske så kommer den informationen att bli borttappad för alltid och är det borttappade meddelandet ett larm kan det innebära katastrof.

1.2 Syfte

Det som skall uppnås i projektet är konstruktionen av ett bakgrundsprogram som ser till att larmen alltid når operatören trots eventuella handskakningsmissar. Programmet skall ständigt kontrollera om det uppstår skillnader i utlösta alarm mellan PLC och Citect och om någon skillnad upptäcks ska programmet korrigera larmet och via det grafiska gränssnittet varna operatören om att kommunikationsmissar har uppstått. Programmet ska vara generellt anpassat och fungera oberoende av PLC:s styrfunktion.

1.3 Avgränsningar

Målet med arbetet är inte att eliminera felrisken vid handskakningsfel mellan Citect och PLC eller att analysera varför larmmissar uppstår utan enbart att i efterhand rätta till de alarmmissar som inträffar de gånger då hanskakningsproblem uppstår.

2 TEORETISK BAKGRUND

2.1 PLC

I början av den digitala automationen användes reläer som kunde kopplas samman för att bilda mer komplicerade styrsystem och de användes i större anläggningar som bilfabriker och trafikljussystemen i större städer enligt förbestämda algoritmer. Nackdelen var att styrfunktionerna programmerades genom fasta individuella tråddragningar mellan reläerna. Det kunde göra systemet svåröverskådligt och det var mycket tidsödande att ställa om systemet från att styra en process till en annan.[1]

Under 70-talet började datorer och programmerbara styrsystem ta över och ersätta de reläsystem som varit etablerade inom automation. Programmeringen kunde då göras via en datorskärm vilket underlättade dokumentation och modifiering av systemet. Konstruktörerna var vid den här tiden redan inskolade på relälogik så PLC-tillverkarna såg till att efterlikna programmeringen så långt som möjligt till relälogiken och den utformningen lever kvar än idag.[2]

Dagens PLC:er är väletablerade inom fastighetsautomation, industriautomation och styrning av infrastrukturanläggningar och har hög tolerans mot yttre påfrestningar såsom fukt, temperaturdifferenser och luftbundna partiklar jämfört med en PC. PLC:er är utrustade med omfattade I/O-anslutningar för att kunna koppas mot externa givare och styrenheter.

2.2 CitectSCADA

SCADA (*Supervisory Control And Data Acquisition*) är samlingsnamnet för system via vilket en operatör styr och övervakar processer inom industri och fastighetsautomation. Ett SCADA-system kan genom sitt HMI (*Human-Machine Interface*) ge operatören grafisk driftinformation och manövreringsmöjligheter från en bildskärm. Genom att koppla samman flera PLC till samma SCADA-system kan hela processen styras centraliserat och ytterligare informationsmöjligheter som ekonomi-, kvalitet- och underhållsuppföljning kan implementeras. Kopplas systemet mot internet finns möjlighet till fjärrstyrning av anläggningen från en mobil enhet.[3]

SCADA-systemet som har utnyttjas i det här projektet är Citect's CitectSCADA vilket är det som AcobiaFLUX använder.

2.3 Cicode

Cicode är det programmeringsspråket som nyttjas av CitectSCADA vars syntax påminner om Pascal och MatLab. Språket har flera biblioteksfunktioner som är avsedda att användas i en typisk styrsystem-miljö.

2.4 Befintlig handskakningsmetodik

Detta arbete bygger på en redan befintlig handskakningsmetodik som i fortsättningen kommer kallas buffertöverföringen.

Den befintliga buffertöverföringen grundar sig på att den generella PLC:n genererar tidsstämplade larm beroende på dess funktionalitet. Det kan vara att CO_2-halten i tunneln är för hög eller att ett styrdon inte svarar på de styrsignaler som PLC skickar. CitectSCADA tar emot PLC:s larmmeddelande och lagrar det i sin interna larmserver där alla mottagna larmvärden sparas. [5]

SCADA-systemet är sammankopplat med flera PLC:er och det är inte säkert att SCADA-systemet kan ta emot larmet direkt när det skapas. Det kan även hända att uppkopplingen är instabil och då måste varje PLC kunna lagra undan sina larmmeddelanden i väntan på betjäning. För ändamålet finns i PLC:n en dedikerad minnesarea som används som en buffert där PLC:n sparar undan sina larmmeddelanden tills de kan bli avlästa av CitectSCADA. Efter avläsning raderas larmmeddelandena från bufferten. PLC:n förutsätts klara av att lagra upp till 4000 larm i minnet och buffertöverföringen läser av 30 larm i taget.

2.4.1 Buffertminnet

Varje larm tilldelas en rad i buffertminnet ihop med sin tillhörande information som är organiserad i 8 stycken celler om 2 byte enligt tabell 2.1. Följaktligen upptar varje rad 16 byte i minnet. Bufferten medger att PLC kan tidsange larmets uppkomst med millisekunds-precision.

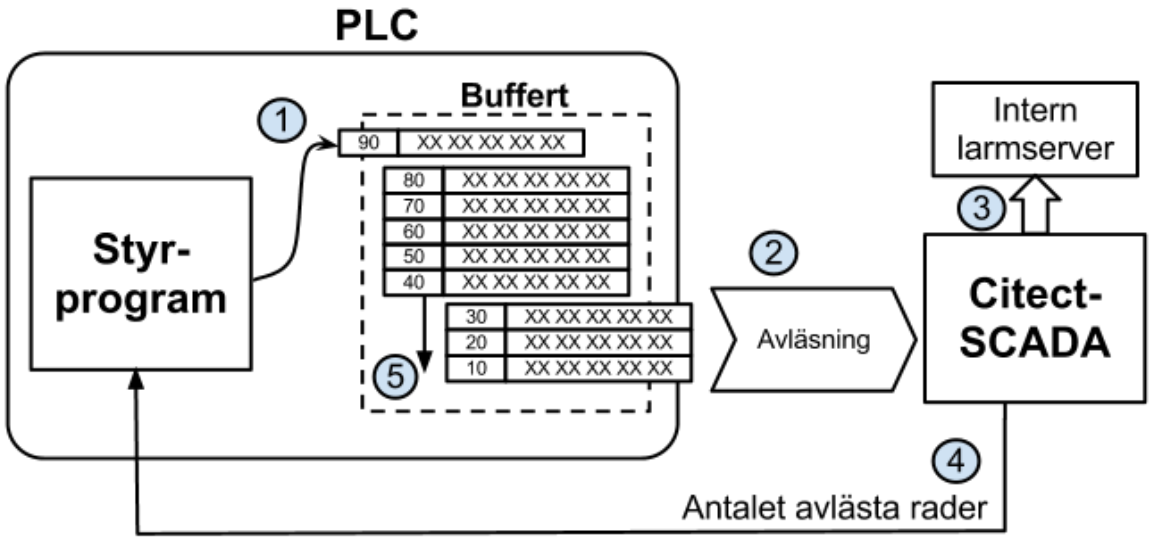
Tabell 2.1 – Utformingen av komponenterna i larmmeddelandet

Funktion	**Cell**	**Beskrivning**
Könummer	Count	Larmets könummer i PLC:ns buffert. Räknas från 0 till 32767 och börjar om på 0 igen
Tid då larmet triggades	YearMonth	Hög byte = År 2000 + (0-255) Låg byte = Månad
	DayHour	Hög byte = Dag i månad Låg byte = Timme
	MinSec	Hög byte = Minut Låg byte = Sekund
	Milli	Millisekunder
Address	ID	Addressregistret i PLC för det larmtagg som har aktiveras (0-100). Adressen används till att knyta larmet till en tagg
Värde	Value	b_1 = Värdet för det larm som har aktiverats (0-1)
Checksumma	Checksum	Checksumma på hela raden genom XOR mellan varje cell

2.4.2 Handskakning

PLC nyttjar tre register till att handskaka över sina alarm till Citects alarmsserver: ReadTrigger, NrOfRowsInBuffer och RowsRead. Överföringen illusteras i figur 2.1

1. Till att börja med sätter PLC registret *ReadTrigger* till 1 så fort det finns en rad redo att läsas i bufferten medan registret *NrOfRowsInBuffer* innehåller antalet larm som finns i bufferten.
2. När Citect har reagerat på att det finns larm som väntar på betjäning nollställer den *ReadTrigger* och läser sedan av hur många rader som finns ur *NrOfRowsInBuffer* och läser av dessa. Överstiger antalet 30 betjänas resterande larm nästa avläsning.
3. Citect börjar med att läsa in raden i sin helhet och därefter lägga cellerna i egna variabler och kör bitvis XOR mellan dem för att kontrollera att de stämmer mot Checksumman. Om Checksumman stämmer fortsätter exekveringen, annars avläses raden på nytt. Tidscellerna används till att skapa en ny variabel av typen TIMESTAMP som används tillsammans med taggnamnet och larmets värde till att uppdatera larmservern.
4. När raderna är färdiglästa meddelar Citect hur många larm som blev avlästa genom att skriva till registret *RowsRead.*
5. PLC tömmer de avlästa raderna och skiftar fram alla kvarvarande larm i bufferten. Samtidigt uppdateras antalet rader i *NrOfRowsInBuffer* och först nu kan PLC ettställa *ReadTrigger* om det finns något kvarvarande larm eller om nya uppkommer.



Figur 2.1 – Den befintliga överföringsfunktionen

2.5 Risk för överföringsfel

Trots att buffertöverföringen med checksumman ger ett gott skydd mot missad data finns ändå en risk att fel kan gå förbi odetekterade eller missas helt.

Överföringen är eventbaserad så PLC:n skickar enbart larmmeddelanden om det har skett en förändring i något larm. Om det larmmeddelandet blir felaktigt avläst eller tappas helt kommer felet aldrig bli upptäckt.

En annan potentiell risk är att XOR-checksumman kan missa ett avläsningsfel av larmvärdet, som finns på b_1 i cellen Value (se tabell 2.1), om det uppstår ett likadant avläsningsfel i någon annan cell på motsvarande bit.

3 CICODE-SPECIFIK FUNKTIONALITET

Detta kapitel beskriver de funktioner som har utnyttjats i utformningen av kontroll-programmet som är mer eller mindre unika för Cicode [4].

3.1 I/O-taggar

Citect utnyttjar taggnamn till att kommunicera med externa I/O-enheter, t.ex en PLC. Taggnamnen är kopplade till de gemensamma variabler som delas mellan I/O-enheten och Citect och taggfunktionerna erbjuder extra funktionalitet som är till nytta vid larmhantering.

TagSubscribe()
Vid anrop börjar Citect prenumerera på en tagg och kontrollerar varje pollperiod om dess värde har förändras. Dess return-värde är en så kallad handle, unik för prenumerationen, som används som parameter vid anrop av andra taggfunktioner. En handle kan liknas vid en pekare som markerar en minnesarea som är reserverad för taggen och dess tillhörande information.

SubscriptionGetValue()
Funktionen returnerar värdet från en prenumererad tagg och används vid manuell pollning.

SubscriptionGetTimestamp()
Funktionen returnerar den tidstämpeln av variabeltypen TIMESTAMP då taggen senast ändrades.

AlarmNotifyVarChange()
Funktionen ser till att uppdatera Citects interna larmserver om tidsstämpeln i den anropande parametern är av senare datum än tidstämpeln i larmservern. Om anropet har äldre datumstämpel än den befintliga i larmservern sker ingen uppdatering.[4]

3.2 Övriga biblioteksfunktioner

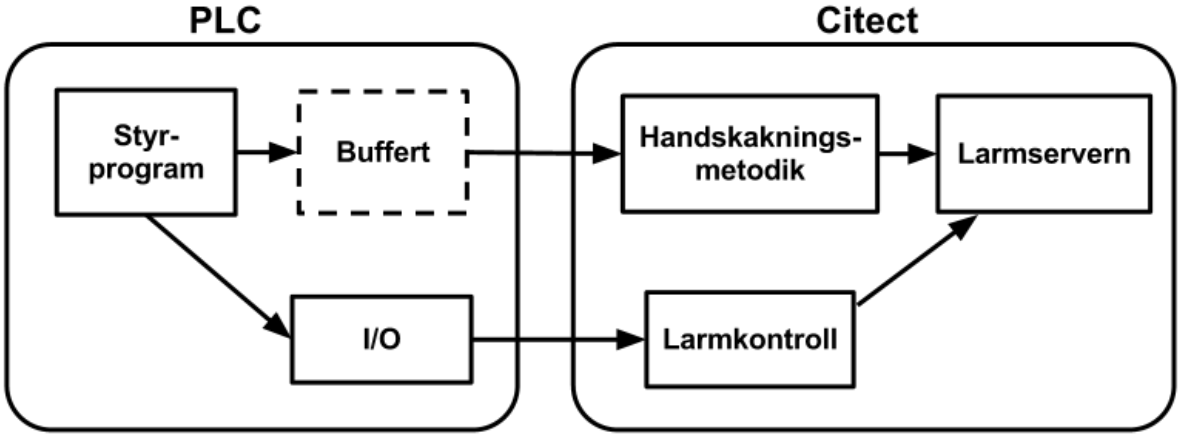
Variabeltypen TIMESTAMP
För att nyttja tid/datum-relaterade variabler används variabeltypen TIMESTAMP som är en integer som räknar antalet millisekunder som har passerat sen 1601-01-01. Det finns fördefinierade biblioteksfunktioner som tillåter initiering och konvertering av TIMESTAMP-variabler.

Sleep() / SleepMS()
Då Sleep() eller SleepMS() anropas sätts en timer som avbryter programexekveringen och återupptar den då tiden har passerat. I evighetsloopar eller stora FOR-loopar är det lämpligt att använda Sleep() för att göra paus och låta andra program eller trådar köras.

4. MJUKVARUDESIGN

Detta kapitel beskriver det program som har skrivits för att detektera larmdifferenser mellan PLC och Citects larmserver och dess funktioner som knutna till programmet. Programmet benämns larmkontroll i rapporten. Kapitlet kan med fördel läsas jämte källkoden ur bilagan.

Programmet använder sig av pollning (tidsstyrd statuskontroll) av I/O-taggarna för att jämföra deras värden i jämna tidsintervall. Systematiskt går programmet igenom varje larmtagg i PLC och Citect och jämför dem mot varandra och om en larmdifferens hittas så korrigerar programmet den felaktiga taggen, skriver felet i en loggfil samt varnar operatören via HMI:t.

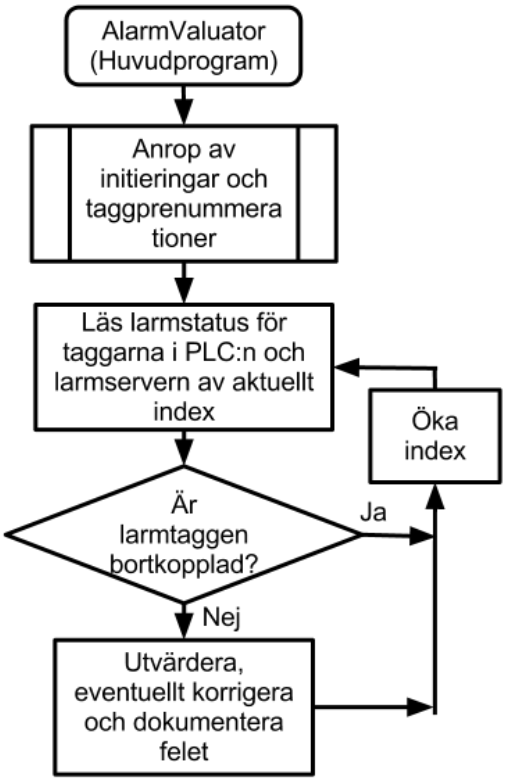


Figur 4.1 – Schematisk skiss av driften då lamkontrollen körs samtidigt som handskakninsmetodiken

Figur 4.2 – Exempel på larmkontrollens arbetsmetodik då ett fel hittas på larmtagg 0004

4.1 Larmkontrollprogrammet - AlarmValuator()

Huvudprogrammets syfte är att sköta indexeringen och att anropa de funktioner som har hand om initiering, inläsning av larmstatus och larmanalys enligt figur 4.3. Om den aktuella larmtaggen inte har blivit inaktiverad så utvärderas den och korrigeras beroende på om det har förekommit en larmdifferens. För att programmet inte ska stjäla för mycket processorkraft från andra parallella processer utnyttjas funktionerna *Sleep()* och *SleepMS()* som sätter en timer som avbryter och återupptar exekveringen när tiden har runnit ur.



Figur 4.3 – Flödesdiagram över funktionen av AlarmValuator()

4.2 Initieringar - SubcsriptionAndArrayInitiation()

Larmtaggarna är namngivna efter ett givet mönster där PLC:s larm kallas *ReadAlarmTag_0XXX* och Citects larmserver kallas *AlarmTag_0XXX*. XXX står för larmens indexnummer och de är numrerade från 1 till 100. Den rådande namngivningsstrukturen gör det möjligt att använda indexering för att systematiskt analysera alla larm.

Funktionen börjar med att definiera de fält som behövs för programmets fortsatta drift. Fyra fält sätts till att innehålla namnen på de taggar och taggfunktioner som kommer anropas senare i programmet. Därefter sätter Citect upp prenumerationer (subscribes) på de taggar som är intressanta att övervaka:

- De larm som är satta av PLC och som ännu inte har handskats över till Citects alarmserver
- Aktiva larm i Citects interna larmserver
- Avstängda larm.

Alla prenumerationer returnerar en s.k. handle som fungerar som en pekare för prenumerationen. De behövs vid anrop av subscribe-relaterade funktioner.

Till sist anropas *ReadAlarmStatus()* för att ge det grafiska gränssnittet aktuella värden att visa upp.

4.3 Läs larmstatus - ReadAlarmStatus()

Varje prenumeration från PLC och Citect anropas av *SubscriptionGetValue()* som returnerar värdena av de indexerade larmtaggarna vilka sparas i sina respektive fält. Fälten avläses senare i de funktioner som jämför larmstatusen. PLC:s värden inverteras innan analys vilket förklaras mer ingående i kapitel 5.3.

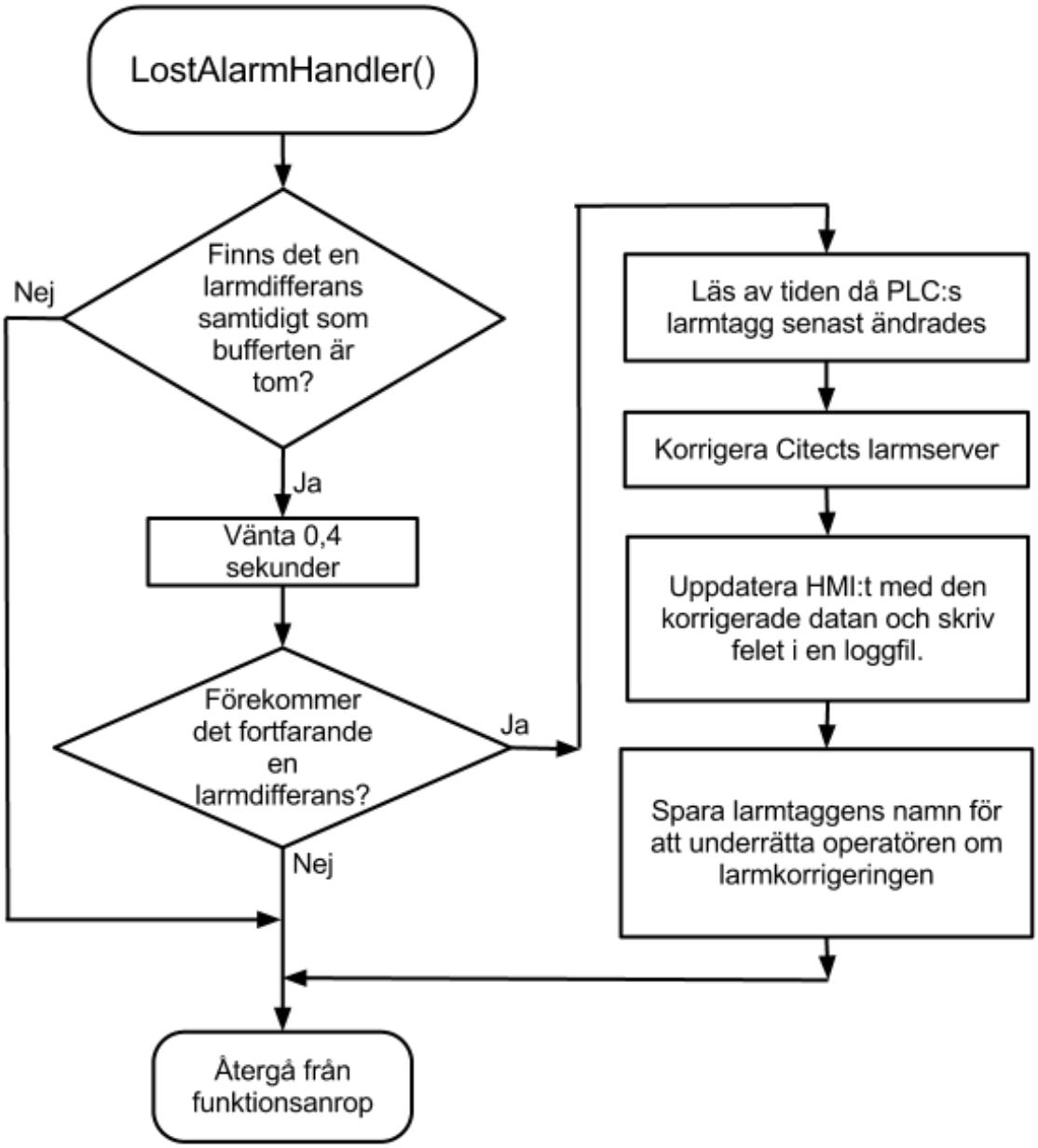
4.4 Utvärdera, rätta larm och skriv fellogg - LostAlarmHandler()

Flödesdiagramet i figur 4.4 beskriver funktionen av LostAlarmHandler(). Funktionen börjar med att i en IF-sats testa om statusen skiljer sig mellan larmen i PLC och Citects larmserver. En differens kan bero på att PLC:s larmmeddelande ligger i bufferten och väntar på betjäning så som bivillkor måste bufferten vara tom. Det finns också en möjlighet att larmmeddelandet håller på att betjänas i samma stund av larmhanteraren *AlarmBufferTask()* och att programmet inte har hunnit uppdatera status i larmserven, därför sätts *LostAlarmHandler()* att vänta en sekund.

Om det fortfarande finns en differens mellan de bägge larmen läser funktionen av tidstämpeln då taggprenumerationen upptäckte att PLC-larmet senast ändrades. Tidstämpeln omtolkas till en textsträng och till en integer för att kunna användas av efterföljande funktioner. *AlarmNotifyVarChange()* anropas för att uppdatera larmserven med det senaste värdet.

Då *WriteErrorToLogFile()* anropas skriver den ett felmeddelande till en loggfil i projektkatalogen. Inledningsvis stängs Citects felkoll av, annars kommer Citect generera ett programavbrott då loggfilen öppnas och exekveringen avbryts. Därefter kan filen öppnas med anrop av *FileOpen()* och av dess sökväg. Om det inte redan finns en loggfil i sökvägen skapar Citect en ny vid anropet.

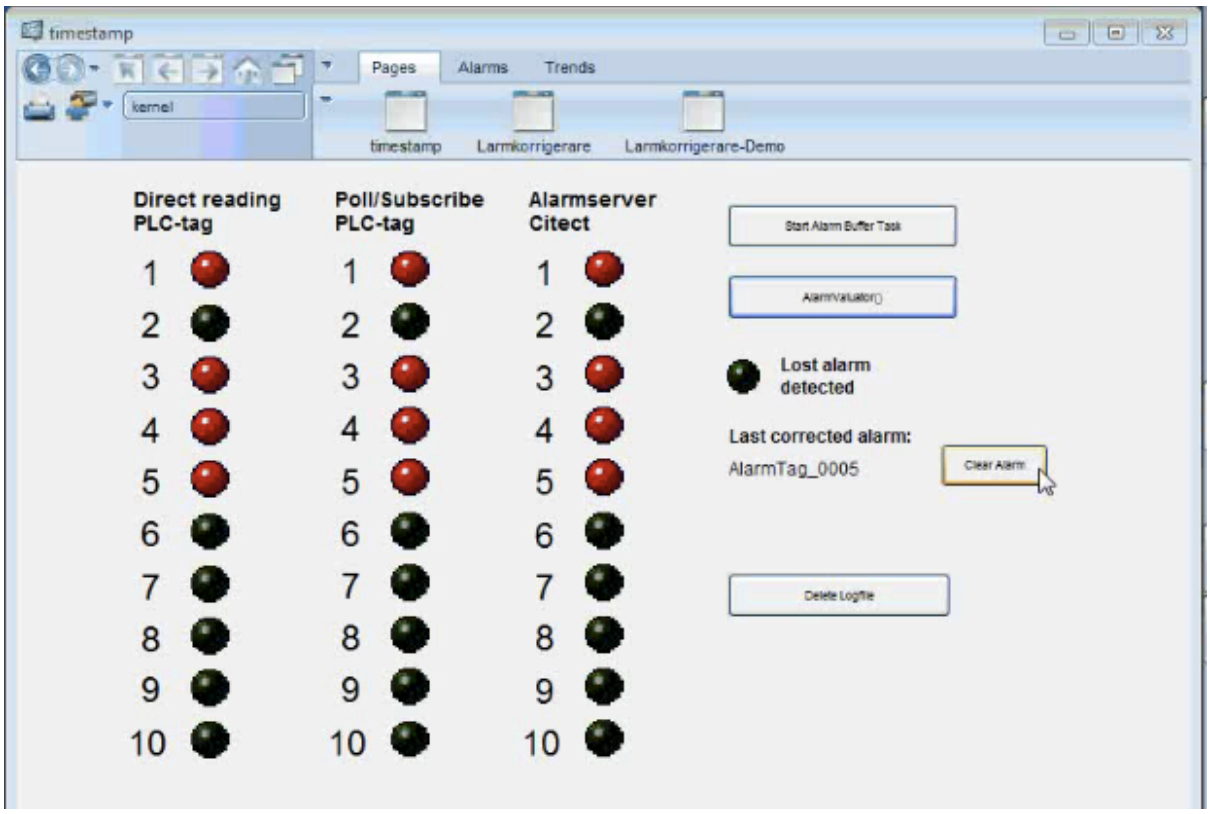
Själva felloggen består av en .txt-fil som beskriver mellan vilka larmtaggar felet har uppstått och tiden då felet det upptäcktes. Då det är klart stängs filen och Citects felkoll återaktiveras. Larmanalysen är nu klar och huvudprogrammet ökar index och läser av nästa larmtagg.



Figur 4.4 – Flödesdiagram över funktionen av AlarmValuator()

4.5 Grafisk representation – HMI

Operatören har tillgång till ett grafiskt gränssnitt där denne kan avläsa statusen i PLC:s larmminne och Citects larmserver och se om något larm har blivit korrigerat av *AlarmValuator()*. Larmstatusvisningen visar de tio första larmtaggarna i respektive minnesarea. HMI:t är i huvudsak tänkt att ge en snabb inblick i hur *AlarmValuator()* arbetar men det är däremot inte tänkt att ge operatören en komplett överblick över statusen för samtliga larm.



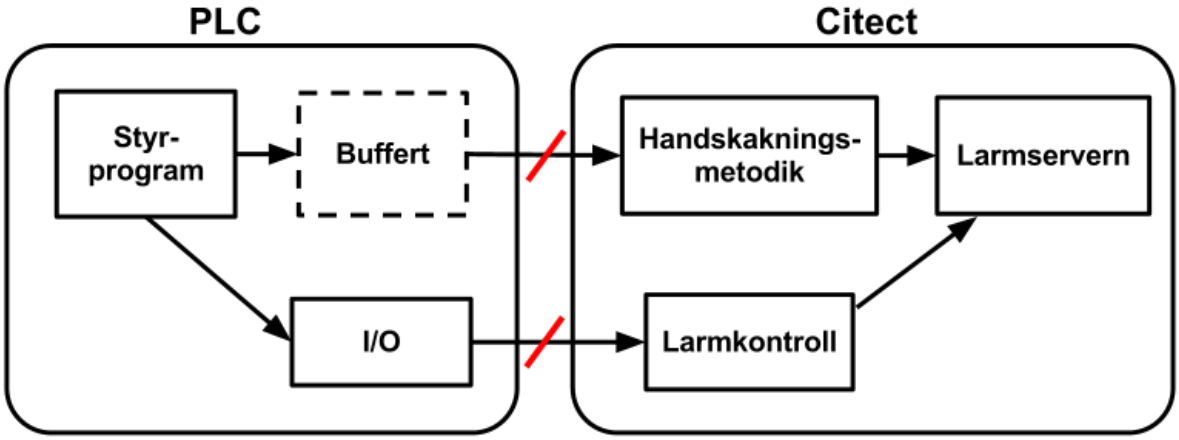
Figur 4.5 – Utformning av HMI:t

Från HMI:t är det bara möjligt att anropa funktioner eller taggar, så för att kunna redovisa statusfälten används funktionerna *RevealAlarmValuePLC()* och *RevealAlarmValueCitect()* som returnerar värdet för det anropade indexet i fältet.

Under programmets gång kommer operatören kunna se att Citects larmserver rättas till om en larmdifferens uppstår. Det senast korrigerade larmet redovisas under texten ”Last corrected alarm” i figur 4.5. Om operatören väjer att klicka på knappen “Clear Alarm” kommer larmets namn och även “Clear Alarm”-knappen att döljas tills det uppstår en ny larmdifferens. Om operatören vill tömma loggfilen kan denne klicka på “Delete Logfile”-knappen som på samma sätt döljs om det inte finns någon loggfil.

5 Testmetodik och resultat

För testning av larmöverföringen finns en PLC med ett testprogram som genererar larm på de aktuella taggarna viket under testet var tio stycken. Larmtaggen slumpas fram och växlar dess larmstatus: om den är nollad blir den triggad och om den är triggad så nollas den. Då statusen är växlad lägger PLC larmmeddelandet på bufferten enligt kapitel 2.4.



Figur 5.1 – Bortkoppling av nätverkskabel

För att simulera nätverksbrott kopplades nätverkskabeln mellan PC och PLC ur enligt figur 5.1. Under avbrottet fortsätter PLC:n exekveringen av sitt program och genererar därmed nya larm som den lägger på bufferten.

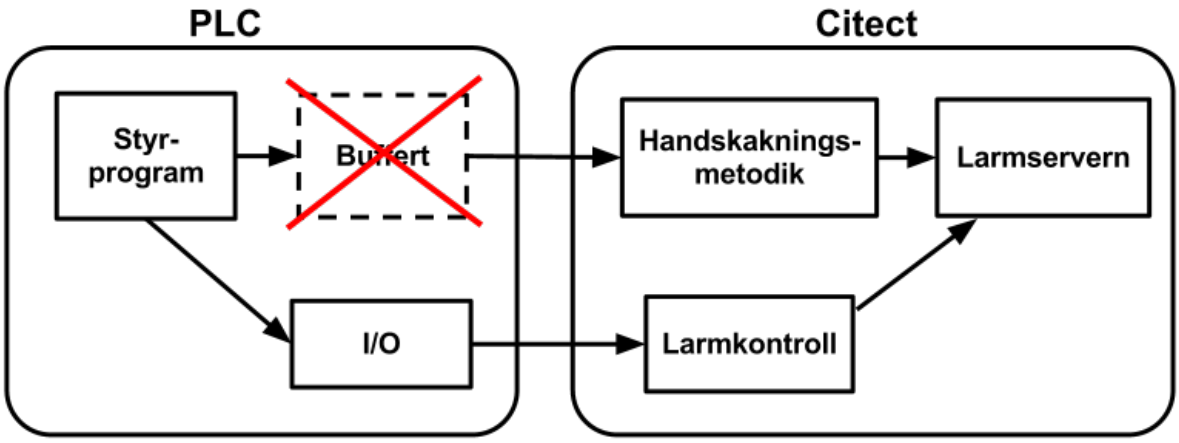
Utöver att anropa *RevealAlarmValuePLC()* och *RevealAlarmValueCitect()* läser HMI:t av PLC:s utgångar direkt (utan subscription) och redovisar dem i kolumnen Direct reading PLC-tag ur figur 4.5. Då den direkta avläsningen redovisades upptäcktes att värdena från utgångsavläsningen är inverterade mot de värden som PLC:n genererar. För att lösa inverteringsfelet återinverteras värdena innan de redovisas eller används för vidare logikanalys.

5.1 Felgenerering med handskakning och nätverksavbrott

Då larm genereras löste den befintliga handskakningsmetodiken larmöverföringen enligt kapitel 2.4.2. Vid nätverksbrott lades de nya larmen på bufferten och handskakningsmetodiken klarade att återuppta överföringarna så fort nätuppkopplingen var återupprättad.

5.2 Felgenerering utan handskakning

För att simulera ett nätverksavbrott som inträffar under en kritisk tidpunkt i handskakningsförloppet kopplades PLC:s bufferthantering bort. Det medförde att registret *ReadTrigger* aldrig blev satt och handskakningsmetodiken blev aldrig varse om att ett larm har blivit triggat. Citect får därmed aldrig sin interna larmserver uppdaterad så larmet förloras och kommer aldrig nå operatören.



Figur 5.2 – Bortkoppling av buffert

AlarmValuator() som läser direkt från PLC:s egna minnesadresser uppmärksammade dock att det förekom en differens mellan larmstatus mellan PLC och Citect och rättade till felet enligt kapitel 4.4. Operatören kunde göras uppmärksam på att det har ferekommit en larmmiss och att det redan har blivit rättat. Längsta tiden mellan felgenerering och korrigering i fallet med tio larmtaggar är ca åtta sekunder vilket främst beror på huvudloopens och *LostAlarmHandler()*'s Sleep()-anrop som pausar processen. Då felet korrigerat skriver programmet ett loggmedelande i en textfil.

5.3 Inverting av PLC:s värden

Under testkörningen upptäcktes att då Citect direktavläser PLC:s larmtaggar, utan utnyttjande av buffertfunktionen, blir de avlästa värdena inverterade mot hur PLC:n genererar sina larm. Det visade sig bero på att larmtaggarna är satta som passivt höga för att t.ex. ett kabelbrott ska tolkas som ett larm. Om signalen hade varit satt till aktivt hög är det stor risk att det aldrig hade blivit känt att ett brott hade inträffat. Det värdet som skickas till buffert är det som anses vara det riktiga.

6 Slutsats

6.1 Uppfyllande av mål

Tack vare komplettering av handskakningsmetodiken med larmkontrollprogrammet är de larmen som sätts av PLC:n garanterade att nå fram till operatören trots uppkopplingsstörningar. Om ett missat larm upptäcks meddelas operatören dels via HMI:t och en loggfil. Programmet uppfyller därmed syftet väl. Nackdelen är att larmtaggarna bara pollas var femte sekund så millisekundprecisionen av tidsangivelsen då larmet skapades förloras men det primära är att larmet faktiskt når operatören.

Programmet är uppbyggt med indexerade fält och funktionsanrop, dels på grund av kravet på generalitet och dels för att koden ska vara lätt att modifiera och för att en utomstående lätt ska kunna sätta sig in i programmets uppbyggnad. Högst upp i programmet finns parametrar som lätt kan ändras av användaren och variablerna är givna så beskrivande namn som möjligt. Programmets generella struktur innebär att programmet inte är knutet till Öresundsbron utan kan användas till andra projekt med liknande struktur.

6.2 Förbättringsområden

Då loggfilen blir uppdaterad med nya korrigeringsrapporter fylls de på under de redan befintliga rapporterna så de senaste rapporterna hamnar längst ner i filen. Det vore önskvärt att nya rader påbörjades högst upp i dokumentet så att den som läser en gammal rapport inte behöver skrolla ner till botten av filen. I dagsläget (Citect v7.20) stödjs inte nyradskrivning till fil på annat sätt än längst ner och det enda sättet att ta bort gamla meddelanden är att radera hela filen eller att gå in manuellt och radera rader. Om senare versioner tillåter mer avancerad filhantering kan det vara värt att anpassa koden.

En av Citect styrkor är dess HMI med grafiska animationer som kan representera schematiska processfölden på skärmen. Kravet på generalitet gör att animering av ett godtyckligt processsflöde blir omöjligt att representera i detta skede men det kan realiseras på anläggning då processen är känd.

Den checksumma som PLC och Citect använder till sin handskakning genereras genom bitvis XOR-summering mellan alla radceller och det finns en önskan från företagets sida att hitta en bättre och säkrare checksumma. En algoritm som kan vara lämplig är den som används vid generering av IBAN-numrens checksumma som utnyttjar modulusdivision av ett kontonummer.

6.3 Erfarenheter

Innan projektets början gjorde jag ett Gantt-schema över hur jag skulle disponera tiden och jag hade lämnat en vecka till att orientera mig i uppgiften och bekanta mig i Citect. I kontrast till förväntningarna behövde jag lägga tre veckor på att lära mig Citect och på att få en tydlig uppfattning av hur den befintliga handskakningen fungerade och vilka verktyg jag hade till mitt förfogande. Själva kodningen gick däremot fortare än vad jag hade förväntat och jag slutade på samma vecka som jag hade avsett.

Jag lärde mig att det är viktigt att ge orienteringsfasen den tiden den behöver för att senare kunna skriva koden på ett effektivt sätt.

REFERENSER

[1] PLC-fakta
Wiklund, I., Hult, G. och Osbeck, M (2011) *Underlag för Styrprojekt*.

[2] PLC-fakta
http://en.wikipedia.org/wiki/Programmable_logic_controller
(2013-01-09)

[3] SCADA-fakta
http://en.wikipedia.org/wiki/SCADA
(2013-01-09)

[4] Cicode referensmanual
Citect Corporation (2004) *CitectSCADA User Guide*.
http://www.vius.ru/files/CitectSCADAUserGuide.pdf (2013-01-09)

[5] Buffertöverföring
Implementation Proposal ÖSB/12-1133