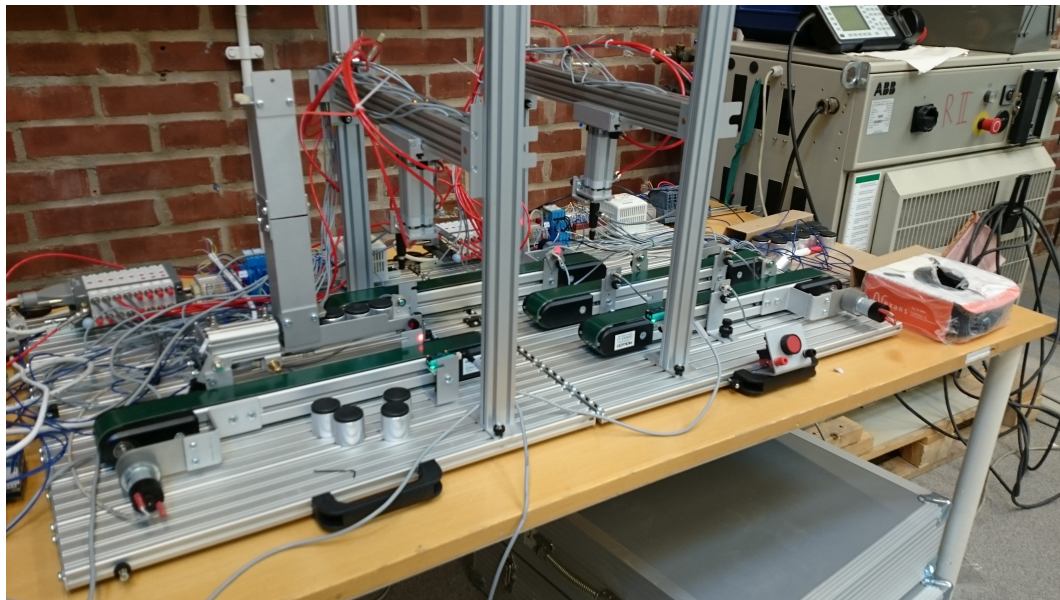




CHALMERS



Återstart och felhantering för produktionssystem

Kandidatarbete inom civilingenjörsprogrammet Automation och Mekatronik

JOEL ALBINSSON
JONAS GARSTEN
CHRISTIAN JERN
ERIK RISFELT
EMELIE VIKINGSSON

Inst. för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2017

Förord

Den här rapporten är en del av kandidatarbetet *Återstart och felhantering för produktionssystem* som utfördes under vårterminen 2017 på Chalmers Tekniska Högskola. Vi vill tacka följande personer för deras samarbete.

- *Kristofer Bengtsson* för handledning och hjälp med Sequence Planner, uppbyggnad av produktionssystemet samt hjälp med PLC.
- *Martin Dahl* för hjälp med Sequence Planner
- *Patrik Bergagård* för hjälp med felhantering och återstart
- Vår examinator *Martin Fabian*

Abstract

In today's society many customers demand products which can be customized. To be able to supply customized products that is constructed without faults the production of customized products is done manually. Manual labor increases time and consequently cost for producing customized products in larger amounts. It would be more time and cost effective if this process could be automatized; since automatizing usually would lead to lower production time and cost for production industry. Automatization of flexible production systems demands much knowledge in automation, especially with handling unplanned errors. Today most of the errors are handled manually which takes both time and skill.

This paper aims to research how error handling and restart can be implemented in a production system to be able to handle a great variety of errors.

A theory based solution for identification och handling of errors is presented. The solution is based of a master system that is developed. This master system has internal variables as representations for the variables of the physical system. The physical variables are handled by a PLC. The internal variables are compared to the physical and if an error has happened the variables will not correspond. The system has to handle this error. A user interface for facilitating the error handling and restart process for the operator is developed.

Sammanfattning

I dagens samhälle kräver många konsumenter att produkter ska bli mer och mer skräddarsydda. För att kunna erbjuda skräddarsydda produkter som är konstruerade korrekt sker mycket av det skräddarsydda arbetet manuellt. Manuellt arbete ökar tiden och därmed kostnaden för att producera skräddarsydda produkter i större volymer. Det skulle vara mer tids- och kostnadseffektivt om denna process gick att automatisera, då kostnaden och tiden för varje enskild produkt att producera hade minskat. Automatisering av flexibla produktionssystem kräver mycket kunskap inom automatisering, speciellt vid oplanerade fel. I dagens läge löses dessa fel ofta manuellt av operatörer, vilket kräver både tid och kunskap.

Rapporten syftar till att undersöka hur felhantering och återstart kan implementeras i ett produktionssystem för att kunna hantera en stor variation av fel.

En teoribaserad lösning för identifiering och hantering av fel presenteras. Lösningarna grundar sig i att ett övergripande system utvecklas. Detta system har interna variabler som representationer för variablerna i det fysiska systemet. De fysiska variablerna i systemet hanteras av en PLC. De interna variablerna och PLC:ns variabler jämförs med varandra och om ett fel inträffat så kommer variablerna inte att överensstämma. Detta fel måste systemet hantera. Ett användargränssnitt för att underlätta felhanteringen och återstarten för operatören har tagits fram.

Ordlista

Backend	Delen av programmet som gör alla beräkningar
CodeSys	Utvecklingsmiljö för att programmera PLC
CPU	Central Processing Unit. Processor som utför beräkningarna i en dator
Fotoelektrisk sensor	Sensor som kan snabbt kan upptäcka objekt på avstånd
Frontend	Gränsnittet som systemet styrs från
Induktiv givare	Detekterar metaller inom avkänningsområdet
Stegkod	En digital ersättare till att programmera med relän
SFC	Sequential Function Chart
Magnetisk induktiv givare	Magnetisk sensor som kan upptäcka magnetiska objekt på avstånd
MVC	Model-View-Controller
OPCUA	Klient/server baserad kommunikationsarkitektur
Petrinät	En riktad grafstruktur
PLC	Programmable logic controller
Relälogik	Den mest grundläggande hårdvaruprogrameringen
Scala/ScalaJS	Ett objektorienterat och funktionellt programmeringsspråk
Scan-cykel	En hel genomgång av ett PLC-program
SP	Sequence Planner
SPA	Single page application
Tjänst	Program utan grafisk representation
Svart låda	En låda vars inre funktion är dold, endast in- och utsignal är känt. Används inom programmering för att förenkla abstraktion
Widget	Window gadget. På svenska: grafisk användargränssnittskomponent
Återstartspunkt	Ett tillstånd där systemet kan återstartas ifrån
Meddelandebuss	Ett sätt att skicka information mellan tjänster och datorer

Innehållsförteckning

1	Introduktion	1
1.1	Syfte	2
1.2	Problemformulering	2
2	Bakgrund till återstart och felhantering	3
2.1	PLC - introduktion	3
2.2	Överordnat system	5
2.3	Felhanteringsprocessen för produktionssystem	6
2.3.1	Vanliga fel i industrin	6
2.3.2	Diagnostik	7
2.3.3	Återstart	8
3	Metod för verifiering av återstart	10
3.1	Produktionssystem för testning av felhantering	10
3.2	Felscenario för verifiering av återstart	11
4	Resultat	13
4.1	PLC:ns styrning av produktionssystemet	13
4.2	Sequence Planner	14
4.2.1	Tjänster i Sequence Planner	14
4.2.2	Modellering av det fysiska systemet i SP	17
4.2.3	Widgets	18
4.3	Lösningar på felscenarion	22
5	Diskussion	26
5.1	Utvärdering av projektets mål	27
5.2	Problem och svårigheter med återstart	27
6	Slutsats	29
	Referenser	30

1 Introduktion

Många fabriker har svårt att klara av att automatisera system där produkter av många olika varianter förekommer. Det är ett problem då det blir allt vanligare att konsumenter vill kunna anpassa produkten efter eget tycke och smak[1]. I dagsläget görs mycket utav detta arbete manuellt. Manuell produktion ökar kostnad och tid för att producera. Därför går priserna upp för produkter med stor variation. För att kunna hålla nere priset på specialanpassade produkter så krävs det att det går att automatisera processen.

Industrier och produktionslinor stöter ofta på stora problem vid oplanerade fel. Det kan vara att en produkt välter, något fel uppstår i en av maskinerna eller att det sker ett strömavbrott. Vissa fel kan innebära att hela systemet måste tömmas och återstartas manuellt. Att det måste tömmas beror på att systemet självt har svårt att hålla reda på vart i systemet det finns produkter då det återstartas. Produkterna behöver då köras om för att systemet ska uppdateras med deras positioner. Ofta är det inte möjligt att ha sensorer överallt i systemet då det finns för många delar som kan gå sönder. Det som då ofta händer är att olika delar i systemet tappar synkroniseringen med varandra [2]. Alltså att en del av systemet säger att det finns en produkt på en plats medans produkten i verkligheten står vid en annan plats.

Problemet vid en manuell återställning är att det tar lång tid och kan ibland tvinga en hel produktionskedja att ligga nere medan åtgärder utförs. Med hjälp av en mer automatiserad återstartsprocess så kan längden på produktionsstopp sänkas. Systemet kan då hjälpa operatören att identifiera vad för fel som uppstått och var felet har skett, också vad som ska göras för att åtgärda felet kan erbjudas[3].

Begreppet industri 4.0 har introducerats för att beskriva den här typen av smarta fabriker[4]. Med det menas fabriker där det i största möjliga mån endast sker automatiserad produktion och helst inga människor behöver vara inblandade. För dessa fabriker krävs det att en övergripande övervakning finns. Med övervakning menas att det finns ett överordnat system som håller reda på produkterna och resurserna i systemet. Operatören kan också bli informerad om var alla produkter är i systemet och vilka tillstånd resurserna befinner sig i. Att bygga upp smarta fabriker är något som självklart ökar utvecklingskostnader då det är många extra timmar mjukvaruutveckling som behövs. De extra timmarna arbete kan långsiktigt innebära att stopptiderna i produktionen blir kortare, vilket i sin tur innebär mycket sparade pengar då stopptider i produktionen leder till ökade kostnader[5].

1.1 Syfte

Syftet med projektet är att utvärdera hur det med hjälp av ny mjukvara och ett användarvänligt gränssnitt går att förenkla återstart av ett produktionssystem och därmed minska dess stopptider. Vidare ska det också undersökas i vilken mån detta kan ske automatiskt och vad som behöver lämnas till en operatör.

1.2 Problemformulering

Arbetet går ut på att med stöd av tidigare forskning inom felhantering och återstartsmetoder utvärdera olika sätt att hantera återstart på för att göra processen så simpel som möjligt för operatören. Vid fel behöver inte lösningen vara uppenbar eller kända sedan tidigare. Därför är det viktigt att operatören kan få tillgång till relevant information och att åter kunna få systemet i ett körbart läge. Det är även viktigt att produktionssystemet är uppbyggt på ett sätt som möjliggör återstart.

Frågor som ska besvaras:

- Kan information om felet presenteras till operatören på ett sådant sätt att det förenklar återstartsprocessen?
- Är det möjligt att åtgärda fel utan att en operatör ingriper?
- Är det möjligt att utforma ett produktionssystem på ett sätt som gör det möjligt att identifiera och åtgärda oförutsedda fel?

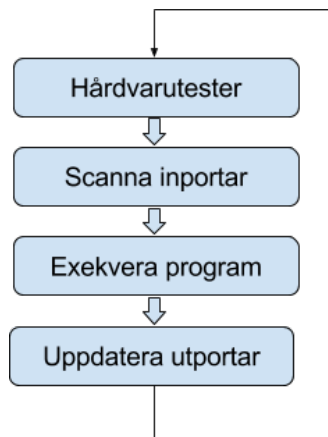
2 Bakgrund till återstart och felhantering

För att förstå återstartsproblemet behövs kunskap om hur hård- och mjukvaran fungerar. Det behövs även kunskap om olika möjligheter för att upptäcka fel och vilka typer av fel som finns. Det kommer också presenteras hur redan befintlig forskning angriper problemet.

2.1 PLC - introduktion

Programmable logic controller (PLC) är något som idag är väldigt vanligt för styrning och insamling av data ute i industrin. PLC är ett robust och säkert sätt att styra både små och stora produktionssystem[6]. En PLC består utav en Central Processing Unit (CPU) som tar in insignaler och utför beräkningar. Beräkningarna som genomförs beror på det aktuella programmet som är programmerat i dess minne. Den skickar sedan signaler till dess utportar beroende på vad programmets logik säger. I CPU:n finns enheter för diagnostisering som vet om alla delar i PLC:n fungerar som de ska, om de inte fungerar som de ska varnar PLC:n användaren. CPU:n har flera olika tillstånd den kan vara i. Bland annat har den ett programmeringstillstånd där den laddar ner ny programkod från användaren. Den har också ett körtillstånd där aktuell kod exekveras[7].

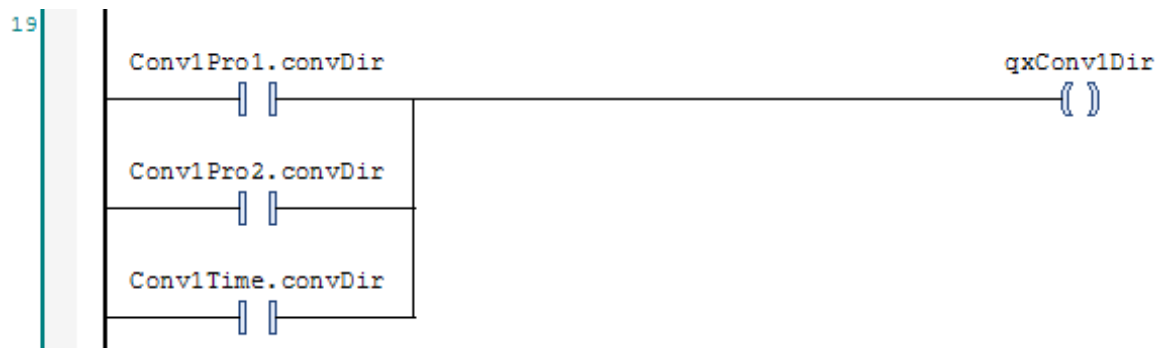
PLC:n har en väldigt kort scan-cykel. Med scan-cykel menas hur ofta datorn läser av insignaler och skickar nya utsignaler. Vad en PLC gör under varje scan-cykel visas i figur 1. Den korta scan-cykeln gör att den snabbt kan reagera på nya signaler ifrån det fysiska systemet[8]. Att reaktionstiden är kort är viktigt vid många applikationer där ett system ska regleras korrekt. Om ett system reagerar för långsamt på till exempel en sensorsignal skulle det kunna innebära att en produkt åker av ett transportband.



Figur 1: PLC:ns scancykel

PLC:n har ett gränssnitt för in- och utgångar. Detta gränssnitt går att anpassa efter eget valt användningsområde. Med hjälp av olika moduler kan både utformning och antal portar varieras för att det ska anpassas efter just det systemet. Dessa moduler kopplas till PLC:n och kan då läsas av eller skrivas till[7]. Det går att styra väldigt många portar med en enda PLC och därför kan en ensam PLC ofta styra relativt stora system. I de fall där det inte är tillräckligt med portar eller där avstånden blir för långa, som i ett större produktionssystem, så kan flera PLC:er behöva användas. PLC:erna kan sedan kopplas in till samma nätverk där de kan kommunicera direkt med varandra. De har också möjligheten att kommunicera med ett övergripande system. Det övergripande systemet styr och kommunicerar sedan med alla de andra PLC:erna[9].

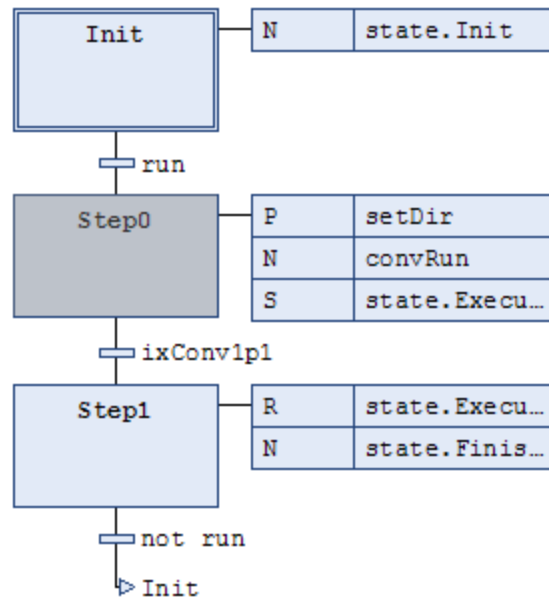
För att PLC:n skall kunna styras krävs ett programmeringsgränssnitt, ett sätt för användaren att berätta för PLC:n hur den ska uppföra sig. PLC:n kan hantera många olika typer av kod och språk. När PLC:n först utvecklades så ersatte den något som kallas för relälogik. Relälogik är föregångaren till digital programmering och ett sätt att programmera fysiskt. Olika brytare och logiska enheter kopplas samman med kablar. Olika kombinationer av kopplingar och enheter kan då styra ett system på önskat sätt. Då de första användarna var vana vid relälogik så designades detta PLC-programmeringsspråk att efterlikna relälogik. Resultatet blev något som kallas stegkod[10], som i stor utsträckning liknar hur relälogik byggdes upp, se figur 2 för ett exempel på stegkod. Denna kod sätter qxConv1Dir, som är riktningen på band 1, till sann om någon utav de 3 booleanerna, `***.convDir`, är sann. En boolean är en variabel som kan anta två värden, sann eller falsk. Anledningen till stegkodens uppkomst var att det utan vidare programmeringskunskaper skulle vara lätt att programmera en PLC[7].



Figur 2: Exempel på stegkod

Stegkod är något som fungerar i stor utsträckning kombinatoriskt. Kombinatoriskt innebär menas att samma kombination av insignaler alltid kommer generera samma kombination av utsignaler. Systemets utfall påverkas alltså inte av i vilken sekvens något sker.

Om däremot följderna av en mängd sekvenser är viktigt och det är förutbestämt i vilken ordning saker ska ske så används Sequential Function Chart (SFC) för att programmera en PLC. SFC gör det mer överskådligt för programmeraren att sätta upp en sekvens med startvillkor och slutvillkor för varje del av sekvensen. Figur 3 visar ett exempel på SFC-kod. Där står hela sekvensen i initsteget tills dess att körsignalen, run, sätts hög. Då kan den gå vidare in i nästa steg, Step0, och där utföra de 3 handlingarna. Den står i steg0 tills dess att ixConv1p1 går hög. På detta sätt går det lätt att bygga väldigt tydliga sekvenser av händelser i systemet. Det finns dock viss risk att fastna i koden om ett villkor aldrig blir uppfyllt. Det kan generera problem som antingen kräver en återstart av kodblocket eller hela programmet.



Figur 3: Exempel på SFC-logik

För att kunna få en överblick över ett större program så förpackas dessa kodstycken in i små lådor, så kallade svarta lådor. Svarta lådor är ett koncept som används för att förenkla överblicken över kod [11]. Det innebär att faktiskt kod döljs och det enda som finns kvar är vilka parametrar lådan vill ha som insignaler, och vilka signaler den skickar ut. Lådan skall skapas så att det är otvetydigt vad den gör alternativt så skrivs en kommentar med dess inverkan på signalerna. Dessa svarta lådor underlättar framförallt kod där många personer ska jobba i samma projekt och ska använda sig utav kod som de andra skapar. Då behövs inte samma insyn i de andras arbete utan det går att återanvända deras svarta lådor utan att behöva lägga lång tid på förståelse. Även i ett projekt där endast en programmerare jobbar så är det högst användbart. Implementeras lådorna på ett allmänt vis så får de också stort återanvändningsvärde och kan spara många timmars arbete[12].

2.2 Överordnat system

Ett överordnat system är ett system som bevakar och styr ett eller flera maskinnära system, såsom en PLC. Överordnade system kan struktureras på flera olika sätt. En struktur som kan användas för ett överordnat system är Model-View-Controller (MVC)[13]. Det är en struktur som delar in systemet i tre tydliga delar.

- En modell byggs upp med klasser och underklasser som alla representerar någon del av det fysiska systemet. Att notera är att modellen inte alltid stämmer överens med det fysiska systemet men synkroniseras vid vissa intervall.
- Controller-delen är den del som bestämmer vilka resurser som kan köra just nu och vad de ska göra. Det är här systemets logik ligger och det är även här som själva styrningen av systemet finns. Controllern skickar sina utsignaler till det maskinnära systemet där de sedan realiserar.
- I view-delen, vyn, så presenteras den relevanta informationen från de två ovannämnda delarna för användaren. Oftast beskrivs någon övergripande bild utav modellen men här kan också olika meddelanden komma upp med ny information. Till exempel att en ny resurs har börjat användas, en produkt är klar eller liknande.

2.3 Felhanteringsprocessen för produktionssystem

Felhanteringsprocessen kan beskrivas i fyra steg: upptäckt, analys, korrektion och återstart[14]. Upp-täckt är att ett fel i systemet identifieras. Analys är att med hjälp av den tillgängliga datan hitta och identifiera det upptäckta felet. Med korrektion menas att det som är fel behöver åtgärdas. Om ett verktyg gått sönder måste denna lagas eller om en sensor är trasig behöver den ersättas. Vid återstart behöver systemet försättas i ett läge där körning kan återupptas. Exempel på detta kan vara att det fysiska systemet och det överordnade systemet måste försättas i ett läge där de överensstämmer eller att systemet måste backas till ett tidigare stadiet. I industrin finns det idag stor efterfrågan på hög kompetens som kan vara behjälplig vid felhantering i produktionen. Felen bidrar till förluster i produktionen då detta innebär att produktionen, i de flesta fall, står helt still. Produktion kan då ej återupptas förrän felet är löst. Detta bidrar till att produktionen kommer drabbas av stora förluster eftersom väntan, enligt lean produktion [15], är ett av slöserierna. Av denna anledning är felhantering idag en viktig fråga för de flesta produktionsföretag.

2.3.1 Vanliga fel i industrin

Farooqui har identifierat sju vanliga produktionsfel där väl implementerad felhantering skulle behövas[14]. Det första felet är verktygsbrott. Verktygsbrott innebär att till exempel ett skär eller ett borr blir oanvändbart till följd av slitage eller att verktyget går av. Om produkten som verktyget arbetar på är inne i en lång process på flera minuter och det sker ett verktygsfel, medför detta att processen måste startas om och går mycket tid går till spillo. Effektiv återstart skulle kunna göra det möjligt att starta mitt i processen istället för att börja om på nytt.

Nästa fel Farooqui tar upp är avsaknad av produkt eller bufferbrist. Det nämns att det för tillfället inte är ett så stort problem då stationerna blir matade manuellt. Men med högre automationsgrad kommer även den manuella matningen att försvinna. Det problem som uppstår när en produkt saknas är att det överordnade systemet inte längre kommer ha en korrekt uppfattning om hur det verkliga systemet ser ut och kan då starta oönskade sekvenser.

Resursfel är ett fel som ofta liknar verktygsbrott i hur det behöver åtgärdas. En resurs är däremot annorlunda från ett verktyg och kan ha större påverkan på produktionen. En robot eller en hel arbetsstation kan vara en resurs, vilket betyder att om dessa går sönder är det inte lika enkelt att fixa som om det bara är verktyget som har gått sönder.

Mjukvarufel är relativt vanliga och svåra att åtgärda. Oftast krävs mycket kunniga och erfarna individer för att hitta och åtgärda felet. Vid varje åtgärdat fel måste försiktighet iakttas så att inga nya fel har uppstått som konsekvens av det tidigare fixade felet. Åtgärd av mjukvarufel kommer förmodligen kräva en total omstart av produktionen för att det överordnade systemet inte ska få fel uppfattning av de fysiska systemet.

Strömavbrott är ett ovanligt fel och de flesta typer av strömavbrott kan industrin redan hantera med hjälp av redundans. Däremot kan strömavbrott till följd av blixtnedslag vara problematiskt. Vid denna typ av strömavbrott kan en produktionsstation komma ur synkronisering med dess styrsystem. Denna avsaknad av synkronisering kan orsaka stor skada på både produkter och resurser under återstartsprocessen.

Vid nödstop kan en enskild station eller en hel produktionslina stanna. För att starta produktionen igen måste hela linan hamna i ett läge där återstart är möjligt och säkert, vilket i värsta fall betyder total återställning av linan.

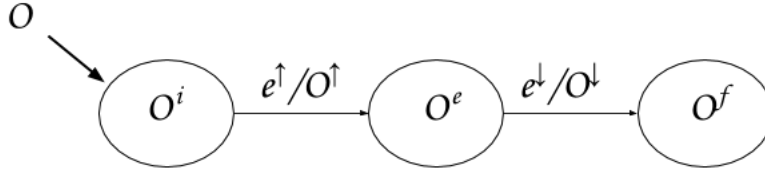
En oförutsedd sekvens av operationer är i nuläget inget stort problem då dagens produktion i störta

grad är statisk på det sättet att en resurs kan bara göra en enda sekvens av operationer. I framtiden däremot kommer produktionen bli mer dynamisk och en resurs kan utföra många olika typer av sekvenser. När denna typ av produktion är implementerad kommer det bli viktigt att kunna hantera denna typ av fel.

Alla dessa olika typer av felscenarion kan mer generellt delas upp i två grupper. Fel som har med det fysiska systemet att göra, exempelvis en saknad produkt eller ett verktygsbrott, eller fel där det överordnade systemet inte längre stämmer överens med det fysiska systemet.

2.3.2 Diagnostik

För att kunna diagnostisera möjliga fel i ett produktionssystem vars styrenhet är en PLC finns det stora fördelar med att använda sig av operationer. Operationer är kopplade till de olika uppgifterna som systemets resurser kan utföra, exempelvis att en robot plockar en produkt på ett visst ställe [16]. I figur 4 beskrivs av Bengtsson en modell av en operation[17]. Modellen består av start- och slutvillkor, start- och sluthändelser samt tillstånd. Villkor beskriver när operationen kan gå till nästa tillstånd och representeras av $O^\uparrow, O^\downarrow$. Händelser utförs när ett villkor blir sant innan operationen hamnar i nästa tillstånd. Händelser är representerade av $e^\uparrow, e^\downarrow$. En operationerna kan vara i tre olika tillstånd, det initiala tillståndet (O^i), exekveringstillståndet (O^e) och tillståndet när den är klar med en operation (O^f). Operationen kan enbart vara i ett tillstånd samtidigt.



Figur 4: Illustration av en operationsmodell med start- och sluthandlingar

Operationer kan också bli ihopsatta i sekvenser, det vill säga, att ett flöde av operationer måste exekvera efter varandra [16]. En sådan sekvens av operationer kan exempelvis vara färdigställandet av en produkt, från dess början till dess slut. Dessa sekvenser av operationer kan bara utföra en operation i taget, alltså inte köra operationer parallellt. En operation kan dessutom enbart vara med i en enda sekvens av operationer. Sekvenserna utav operationer kan vara antingen aktiva eller inaktiva. Sekvenserna är aktiva när en av operationerna är i exekveringstillståndet och inaktiva när ingen av operationerna i sekvensen är i exekveringstillståndet.

När en sekvens av operationer körs kommer det finnas ett tillstånd som sekvensen förväntas befinna sig i [16]. När det sker ett fel i produktionen kommer det förväntade tillståndet och det verkliga tillståndet skilja sig från varandra. Det förväntade tillståndet är alltså det tillstånd som modellen har som finns beskrivet i avsnitt 2.2. Detta problem uppstår när det blir fel i det fysiska systemet och något förväntas ha hänt eller något oväntat händer. Exempelvis, en sekvens med operationer körs och det är den andra operationen i sekvensen som är i exekvering. Här uppstår ett fel som gör att det fysiska systemet inte kan göra klart operationen, det vill säga, operationens tillstånd sätts aldrig till klar. Det överordnade systemet vet däremot inte att felet uppstått och förväntar sig att operationen är klar, vilket innebär att nästa operation kan köras. För att hitta ett fel kan man alltså jämföra det verkliga, fysiska tillståndet med tillståndet som det överordnade systemet förväntar sig att sekvensen av operationer är i. När dessa två tillstånd skiljer sig kan man diagnostisera att det uppstått ett fel

och förhoppningsvis vart felet uppstått.

Ett enklare men mer statiskt sätt att diagnostisera fel på är funktionsbaserad diagnostik som returnera ovanliga värden från funktioner när något inte stämmer[18]. Det kan vara att en funktion som förväntas att bara returnera positiva värden skickar ett negativt värde. Till exempel om en robot åker utanför dess tillåtna område så ändras positionen till ett negativt värde när alla tillåtna positioner är definierade som positiva. Den här typen av diagnostisering behöver programmeras in i själva funktionerna och kan därför bara hantera förutsedda fel. Oförutsedda fel kommer då behöva upptäckas och korrigeras av operatören till maskinen.

Tillsammans med funktionsbaserad diagnostik är grafbaserad diagnostik ett av det vanligare sättet att göra det på[16]. Dessa är ofta byggda av Petrinät som är en typ av graf men har även fördelen att kunna representeras matematiskt. Petrinät modelleras med platser, övergångar och riktade bågar. Petrinät kan användas på många olika sätt för att hitta fel. Vanligt är att man använder en observatör som kollar efter otillåtna övergångar i det Petrinät som representerar systemet[19]

2.3.3 Återstart

Återstart beskrivs kort som "the process by which the system returns to a state where production can restart after an abnormal and disruptive condition has occurred"[20]. Alltså, återstart är den process när systemet återvänder till ett tillstånd där produktionen kan återstarta efter att en abnormal och störande händelse har hänt. Denna process innebär till stor del om att synkronisera det fysiska systemet med modellen i det överordnade systemet efter att man identifierat samt korrigerat ett fel[16]. Det fysiska systemet måste befinna sig i rätt motsvarande tillstånd som modellen för att kunna få korrekta och relevanta instruktioner för vidare körning. Skulle modell och fysiskt system vara ur synkronisering vid återstart kan detta orsaka en krasch eller vidare fel. Om till exempel en produkt befinner sig på en plats där modellen tror det skall vara tomt finns risken för fel när ytterligare en produkt skall placeras på samma plats. Saknas en produkt vid en position där modellen tror det skall finnas en, kommer vidare instruktioner sannolikt inte ge förväntad effekt och vara direkt felaktiga.

Då ett system stoppas till följd av att fel uppstått kan såväl felet i sig som själva korrigeringen av felet ofta innebära att systemets fysiska tillstånd ändras utan modellens vetskap[21]. Alltså kan det fysiska systemet befinna sig i ett tillstånd som inte har något motsvarande tillstånd i modellen.

Återstart är inte självklart lämpligt i alla tillstånd ett system kan anta. Ett tillstånd som befinner sig i en serie tillstånd där tidsvarianta processer pågår kan ofta vara olämpligt att återstarta ifrån. Bergagård beskriver ett exempel när ett rådande tillstånd innebär att en produkt har blivit limmad och väntar på att monteras. Appliceras nytt lim igen på det gamla är det inte säkert att önskad kvalitet efterlevs varför detta tillstånd bedöms som olämpligt återstartspunkt [3].

En återstartspunkt är ett tillstånd för ett system var det kan vara möjligt att återstarta för att sedan kunna fortsätta produktion[16]. En återstartspunkt måste inte vara ett tillstånd som tillhör ordinarie produktion, utan det räcker att det är möjligt att ta sig vidare till ett tillstånd som leder till att ordinarie produktion kan återupptas [21].

En simpel punkt att starta ifrån kan vara då hela systemet befinner sig i startposition, vilket innebär ett väntande tillstånd då hela systemet är tomt. Men vid mer komplexa sammanhängande system med många produkter i omlopp så förlorar man mycket tid vid omstart om hela systemet först måste återställas till starttillstånd dvs tömmas. Genom att införa fler återstartspunkter som inte innebär lika tidsödande åtgärder som tömning av hela systemet kan återstart på så vis effektiviseras.

Många metoder för återstart bygger på att först backa systemet, köra fram systemet eller bara återstar-

ta från det tillstånd där felet identifierades och åtgärdades [22], [23]. Vid flexibla system där sekvenser av operationer inte är lika tydliga blir det inte längre självklart till vilket tillstånd som skall backas respektive köras fram till [21]. Bergagård föreslår också generella metoder för att hitta lämpliga återstartpunkter vid flexibla system där man på förhand beräknar vilka återstartpunkter som dels är lämpliga och dels orsakar minst besvär för operatören.

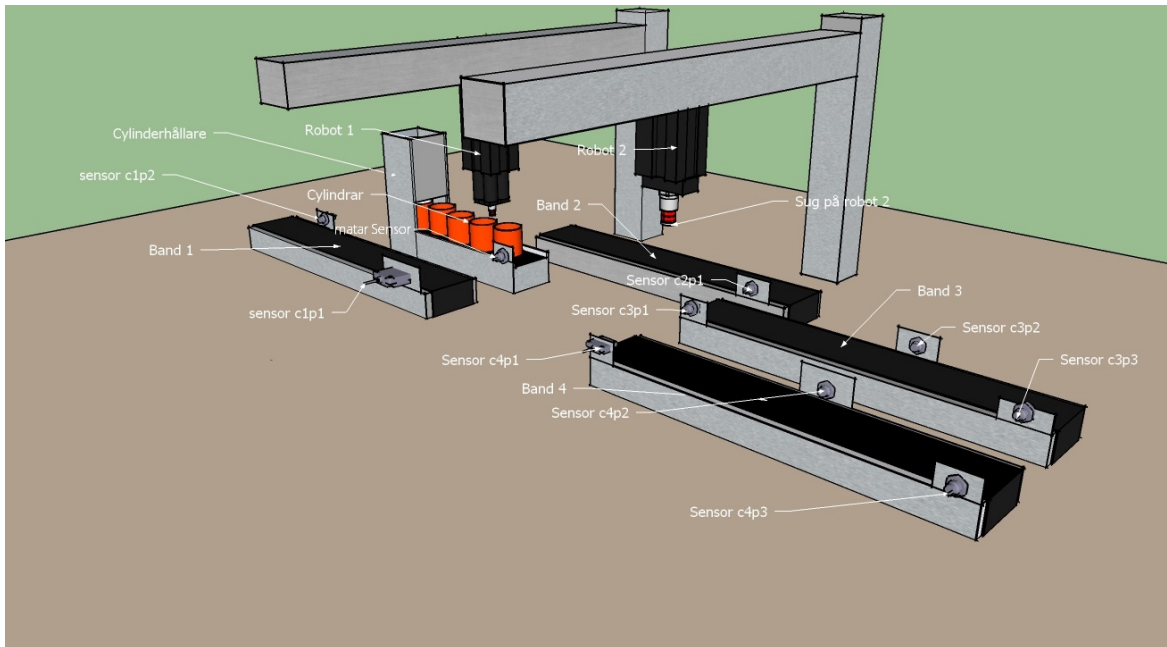
3 Metod för verifiering av återstart

För att fysiskt kunna visa att återstartsmetoder fungerar används ett litet produktionssystem med ett förbestämt flöde för produkter. I detta produktionssystem kan olika typer av fel och processen för att identifiera samt återstarta systemet demonstreras.

3.1 Produktionssystem för testning av felhantering

Det fysiska produktionssystemet, se figur 5, består av fyra rullband, två linjärrobotar, en produkthållare, två fotoelektriska sensorer, åtta induktiva givare, tolv magnetiskt induktiva givare och ett antal cylindrar. Robotarna har en ställning som gör att de har möjligheten att styras fram och tillbaka mellan de olika positionerna som roboten ska hämta eller lämna cylindrar på. För att veta robotens position används de magnetiskt induktiva givarna, sex stycken per robot. Detta gör att robotarna kan vara i tre olika positioner. Robot 1 kan vara vid mataren, band 1 och vid band 2. Robot 2 kan vara vid band 2, band 3 och vid band 4. De två robotarna har också ett sugverktyg som gör att roboten kan flytta cylindrar från en position till en annan. Rullbanden används för att transportera cylindrarna mellan olika sensorer. Sensorerna representerar vardera varsin process som cylindrar kan passera igenom för att förädlas. Produkthållaren matar ut produkter, en efter en, av de produkter som finns i produkthållaren. Produkterna i produkthållaren fylls på manuellt. När produktionssystemet har modellerats har produkterna antagits vara tunga, det vill säga, att det inte går att lyfta produkterna för hand från en position till en annan. Detta har antagits eftersom denna forskning och utveckling, för ett produktionssystem, är alldeles för dyr om inte produkterna i produktionssystemet är av stort värde, exempelvis bilar.

Flödet i processen är beroende av vilka processer som valts för varje produkt. Men det finns vissa krav på i vilken ordning de kommer ske då produktionssystemet ser ut som det gör. Planeringen är inget som operatören berörs av utan det är systemet som själv planerar upp produktens väg genom dess processer och tar produkten från start till mål genom produktionssystemet. Band 1 har en process som är valfri, vilket innebär att alla produkter inte behöver passera denna process. In- och utflödet på band 1 är vid samma position, alltså vid sensor c1p1 (se figur 5). Band 2 kommer däremot alla produkter att passera eftersom det är ett transportband som för produkten vidare i systemet. Band 3 har tre stycken processer där produkten kan välja att gå igenom en, två eller tre processer. Den kan också välja att gå igenom ingen av processerna. Produkten matas sedan ut ur systemet. På band 4 finns det två stycken processer som kan utföras oberoende av varandra. Sedan finns möjlighet att mata ut produkten ur systemet om produkten är färdig. Men om den inte är färdig kan den åka tillbaka till inpositionen på bandet, sensor c4p1 i figur 5. Därifrån kan den sedan plockas upp för att kunna åka till band 3 för att genomföra de sista processerna för att därefter matas ut. De två enda utgångarna ur produktionssystemet är alltså band 3 och 4. En produkt som ska till både band 3 och 4 kommer aldrig först åka till band 3 då det bandet inte har möjlighet att mata tillbaka.



Figur 5: Virtuellt representation av det fysiska systemet

3.2 Felscenarion för verifiering av återstart

För att testa och demonstrera systemets felhantering har fem scenarion tagits fram. Dessa fem scenarion baseras på de vanliga fel som Farooqui har presenterat och som är beskrivna i avsnitt 2.3.1[14].

I det första scenariot demonstreras ett fel där en produkt inte kommer fram till dess förväntade position, alltså avsaknad av produkt. Att en produkt saknas kan ha flera orsaker. Bland annat kan det bero på att råvaran ej blivit påfylld korrekt eller att en produkt har fallit bort. Det är ett problem som lätt kan uppstå då det finns många saker som kan orsaka det. Scenariot demonstreras i produktionssystemet genom att produkthållaren har slut på råvara. Det kommer innebära att mataren kommer försöka mata ut råvara när det är tomt.

I det andra scenariot demonstreras ett fel där en av resurserna går sönder. Resursfel kan vara omfattande fel där en hel robot kan vara ur funktion och behöver lagning av mekaniker. Det kan också vara något så enkelt som att en resurs har blivit strömlös och därmed bara behöver ström. Resursfel blir därför en väldigt generell beskrivning av flera olika feltyper. Felet kommer att demonstreras i systemet genom att transportband 4 går sönder, vilket realiserar genom att göra bandet strömlöst. När bandet är strömlöst kommer operationen för att flytta bandet att startas men bandet på produktionssystemet kommer inte röra på sig. Systemet kommer då fastna i ett tillstånd där det försöker flytta en produkt men inget händer.

I det tredje scenariot demonstreras en mer specifik typ av resursfel, nämligen sensorfel. I nyare system där mycket automatiserad produktion skall kunna ske krävs det flera nya sensorer. Sensorer kan gå sönder, börja ge fel data eller sluta svara helt. Det kan bero på trasiga kablar eller att smuts kommit i vägen. Ju fler sensorer som finns i ett helt system desto oftare kommer den typen av sensorfel att uppstå. Felet kommer att demonstreras utifrån att en sensor är trasig och ej kommer kunna reagera på att en cylinder passerar.

I det fjärde scenariot demonstreras ett fel där ett verktygsfel sker. Oftast när detta sker så beror det på att verktygen är en form av förbrukningsvara som måste bytas kontinuerligt. Till exempel kan det vara ett skär som ej är skarpt nog eller en gummidel som blir torr och får sprickor. Detta kan orsaka stora fel om de ej upptäcks i tid och möjligheten finns att flera produkter förstörs innan felet upptäcks. I systemet kommer felet att demonstreras genom att luftslangen till sugluften att kopplas ur från en robot. Det representerar att gummit har gått sönder och därmed kommer roboten inte kunna plocka upp några produkter. Det kommer att demonstreras på robot 2. Även om roboten blir obrukbar så är det fortfarande endast ett verktygs fel då själva resursen, roboten, fortfarande kan förflytta sig och förlänga eller dra ihop armen. Det enda som gått sönder är egentligen verktyget på roboten.

I det femte scenarion så demonstreras ett strömbavbrott. Strömbavbrott kan ske av olika orsaker, åsknedslag eller kabelbrott är några utav dem. Ett strömbavbrott ger direkta problem i att allting stängs av och ingen produktion kan ske. Efter återstart kan det även finnas stora problem med synkroniseringen igen. Ofta är programmen uppbyggda i olika nivåer där det finns en hierarki vilket program som styr vilket. De olika delarna kan då ha svårt att hitta tillbaka till samma bild av verkligheten och därmed hamna osynkroniserat med varandra. Strömbavbrottet kommer realiseras genom att koppla bort strömmen från det fysiska systemet men fortfarande ha kvar ström till datorn och det överordnade systemet. Det är så det oftast sker i befintlig produktion då det överordnade systemet har möjlighet att spara sin information i en molntjänst och därmed ej kraschar totalt.

4 Resultat

Efter att de nödvändiga litteraturstudierna genomförts har det byggts upp ett produktionssystem där olika återstartsscenarioer ska kunna testas. Detta system styrs av en PLC som kodats med CodeSys. PLC:n styrs i sin tur utav ett överordnat system som är skrivet i språket ScalaJS. Det har byggts upp ifrån en kodbas som handhålls av Chalmers, Sequence Planner (SP). SP innehåller flera verktyg som har kunnat återanvändas i projektet. För att möjliggöra återstart så har en operationsmodell byggts upp. Operationsmodellen finns där för att kunna upptäcka och identifiera fel genom att jämföra de två tillståndet som det överordnade och verkliga systemet har. För att sedan styra och felhantera systemet så planeras ett grafiskt användargränssnitt som möjliggör för just återstart.

4.1 PLC:ns styrning av produktionssystemet

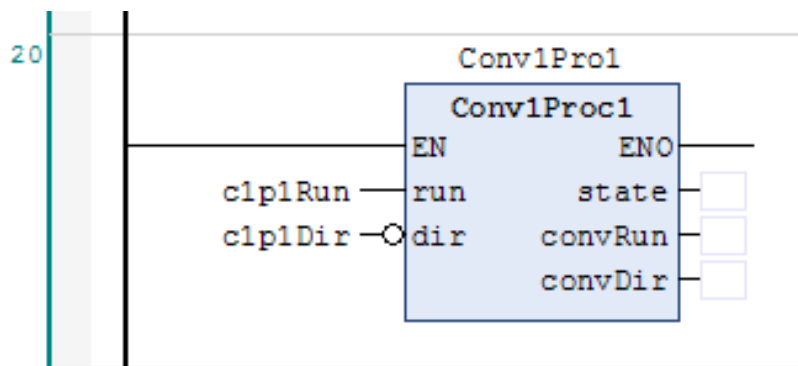
Att använda sig utav en PLC för att styra systemet var det smidigaste och mest relevanta sättet att styra systemet på. Ute i dagens industri så är det i princip uteslutande PLC:er som står för styrningen av maskiner[24]. Därför är det just ihop med en PLC som det är högst relevant att kunna realisera återstart.

I PLC:n skapas det kodblock som vardera styr sin respektive operation. Det är PLC:n som skickar de faktiska signalerna som startar robotarna och transportbanden i produktionssystemet. Ingen annan del i koden har möjligheten att kommunicera med produktionssystemet direkt utan att först gå genom PLC:n.

Koden som styr PLC:n skapas i CodeSys med både SFC- och stegkod. Ibland krävs de tydliga sekvenserna, eftersom följden är viktig, då används SFC. Annars är det stegkod som används. Koden skapas ur ett perspektiv där varje kodblock utför en operation vardera. Detta för att lätt kunna koppla det till det överordnade systemet och för att underlätta felsökning. Ibland skapas även vissa hjälpfunktioner som endast anropas av andra kodblock i CodeSys. De kan ses som hjälpkod som det överordnade systemet inte behöver ha kontakt med.

Det är viktigt att operationer skapas på rätt abstraktionsnivå. Om varje kodblock gör för lite blir det mycket för ett överordnat system att styra. Detta försvårar kodningen utav den överordnade logiken då mycket mer måste tas hänsyn till. Det överordnade systemet har också för stor fördröjning i kommunikationen, vilket gör att den är långsam på att reagera på nya sensorsignaler. Om det däremot ligger väldigt mycket logik i PLC:n så kommer det överordnade systemet ha svårare att hålla uppsikt över det fysiska systemet i samma utsträckning. Då PLC:n stängs av för en återstart så innebär det också att hela dess minne töms och alla variabler blir återställda. Att ha mycket logik i PLC:n kan då leda till att felhantering och återstart försvåras då ingen del i systemet längre håller reda på i vilket tillstånd det senast var. Det är därför viktigt att här hitta en bra balans i uppgifterna. Nya beslut för varje enskild operations utformning måste därför tas så abstraktionsnivån blir rätt. I största möjliga utsträckning så är operationerna i PLC:n skapade så att de gör så lite som möjligt, med just sensorreaktioner som den undre gränsen.

Ett exempel på ett kodblock visas i figur 6. Detta kodblock kör en cylinder på transportband 1 (Conv1) till process 1 (Proc1). Det är en relativt kort sekvens men som gör precis tillräckligt för att klara av just sensorhanteringen. Vid process 1 så finns en sensor som går hög när cylindern anländer. Direkt då måste bandet stängas för att cylindern inte ska åka för långt. Det fungerar endast om hanteringen sker inom PLC:n utan att behöva vänta på signaler från det överordnade systemet. Kodblockets insignaler är de till vänster i figur 6 och dess utsignaler är de till höger.



Figur 6: Exempel på kodblock i CodeSys

För att inte systemet ska tappa några cylindrar genom att ett band kör för långt så måste timers läggas in. Dessa timers stänger helt enkelt av ett band om det har kört för långt. Dessa timers har högre precision i en PLC och om en timer löper ut behöver reaktionen vara nästintill omedelbar. Därför implementeras även dessa i PLC:n och inte i det överordnade systemet.

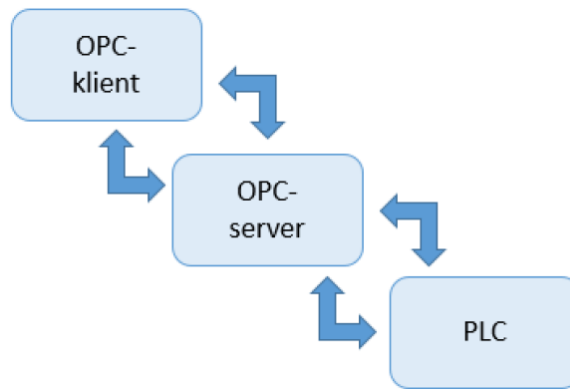
4.2 Sequence Planner

Sequence Planner (SP) är en mjukvara utvecklad på Chalmers med avseende att förenkla modellering och styrning utav automationssystem[25]. Det är en mjukvara under ständig utveckling. SP har hjälpmedel för optimering, schemaläggning och planering utav produktion. SP har också ett väl underbyggt material för att underlätta för användaren att bygga sitt egna webbaserade användargränssnitt med hjälp av widgetar. En widget är en fönsterruta som har ett förutbestämt syfte, det kan vara till exempel att ha några knappar för styra systemet, eller för att presentera relevant data för användaren. Med SP kan flera sådana läggas intill varandra i ett enda webbfönster så att alla kan iakttas och användas samtidigt. Som frontend kommer alltså det grafiska användargränssnittet skapas i SP med hjälp av dessa widgets. Frontend är den delen av programmet som användaren ser och interagerar med. I backend så kommer alla logik som styr PLC:n att finnas. Med backend menas helt enkelt det som händer i programmet som användaren inte ser. Den här uppdelningen används för att användaren endast ska presenteras den väsentliga informationen.

4.2.1 Tjänster i Sequence Planner

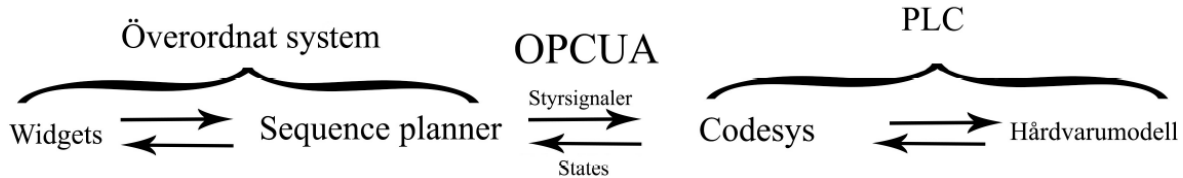
Många av SP:s widgetar använder sig av så kallade tjänster som bäst beskrivs som separata program. Dessa tjänster har ingen grafisk representation som widgetarna utan är till för att sköta beräkningar, algoritmer, kommunikation med mera. Tjänsterna kan skicka data mellan varandra, men även skicka data till widgetarna så att dessa ständigt är uppdaterade.

För kommunikationen mellan SP och PLC används Object Linking and Embedding for Process Control Unified Automation (OPCUA). Detta är ett kommunikationsgränssnitt som används för att kommunicera mellan olika enheter inom industrin[26]. Det är möjligt att köra detta på alla vanliga plattformar. OPCUA bygger på att data läggs ut på en server. Andra enheter kan sedan få tillgång till datan via servern. Ett exempel på hur kopplingen går till visas i figur 7. Det går att koppla flera PLC:er, och andra enheter, som alla skickar data till servern.



Figur 7: OPCUA - överblick

I figur 8 så kan det ses en övergripande struktur över hur de olika systemen kommunicerar med varandra. Strukturen bygger på en ram liknande ett MVC-system, beskrivet i avsnitt 2.2. Widgets tar in data från användaren och skickar data till backend i SP. SP skickar då ut styrsignaler på OPCUA-servern där PLC:n läser av dem. Strukturen för hur PLC:n läser av den datan och PLC:ns logik skrivs i CodeSys. PLC:n är sedan direkt kopplad till hårdvaran. När något nytt händer i hårdvaran skickas hårdvarans signaler till PLC:n som i sin tur lägger upp sitt nya tillstånd på OPCUA-servern. SP läser då av tillståndet och uppdaterar sin modell samt widgetsen så de visar den nya, korrekta informationen. Användaren kan då läsa av den nya datan och göra nya val.



Figur 8: Modell av systemet

För att kommunikationen ska fungera så måste PLC:ns in- och utvariabler struktureras upp på ett tydligt sätt. Det är viktigt att de heter samma sak i alla de system som är intresserade av att hantera variablerna. Därför är namnkonventioner viktiga för att det inte ska bli några problem. Ett exempel på hur en sådan lista med variabler ser ut i CodeSys ser ut syns i figur 9. Där visas namnen på strängarna och pilarna som är dubbelriktade. Detta visar att PLC:n skickar ut variablerna på servern.

☒	c1p1Dir			BOOL
☒	c1p1Run			BOOL
☒	c1p1Sensor			BOOL
☒	c1p1State			INT
☒	c1p2Dir			BOOL
☒	c1p2Run			BOOL
☒	c1p2Sensor			BOOL
☒	c1p2State			INT
☒	c2p1Dir			BOOL
☒	c2p1Run			BOOL
☒	c2p1Sensor			BOOL
☒	c2p1State			INT
☒	c3TimeRun			BOOL
☒	c3TimeState			INT
☒	c3TimeTime			INT
☒	c3p1Dir			BOOL

Figur 9: Del av variabellistan i CodeSys

De variabler som PLC:n skickar ut på servern kan det överordnade systemet välja att läsa av och skriva till. Det har därför i SP skapats en motsvarande lista med variabler som SP prenumererar på, se figur 10.

```
val nodes = List("feedRun", "feedFinish", "feedState", "feedError", "robot1Run", "r
"robot1ResetState", "robot2Run", "robot2Target", "robot2State", "robot2ResetRun", "
"c1p1State", "c1p1Sensor", "c1p2Run", "c1p2Dir", "c1p2State", "c1p2Sensor", "c1Time
"c2p1Dir", "c2p1State", "c2p1Sensor", "c3p1Run", "c3p1Dir", "c3p1State", "c3p1Senso
"c3p3Run", "c3p3Dir", "c3p3State", "c3p3Sensor", "c3TimeRun", "c3TimeTime", "c3Time
"c4p2Run", "c4p2Dir", "c4p2State", "c4p2Sensor", "c4p3Run", "c4p3Dir", "c4p3State",
```

Figur 10: Variabellistan i SP

Kommunikationen med kodblocken i PLC:n måste också följa en viss standard så det blir lätt att bygga upp logiken i SP utan att behöva ha större inblick i den maskinnära koden i PLC:n. Kommunikationsstandarden är uppbyggd som så att SP skickar en specifik run-signal till PLC:n som då startar en operation. Varje operationsblock i PLC:n har sin respektive signal som den väntar på ska gå hög. PLC:n håller inte reda på om operationen får starta, det ansvaret ligger hos SP. Sedan körs operationen i sin helhet tills dess att den är i sitt färdigtillstånd. Där väntar PLC:n på att run-signalen från SP går låg och därmed är operationen avslutad och den operationen kan nu utföras igen. Detta kallas för en handskakning och är ett sätt att hålla de två systemen synkroniserade med varandra. Då de både opererar inom olika tidsramar så är det viktigt att PLC:n inte bara skickar signaler utan att få bekräftat att SP läst dem. Utan handskakningen skulle SP helt tappa vart det fysiska systemet befinner sig, vilket innebär att modell och verklighet skulle vara helt skilda.

I figur 6 ses ett kodblock med dess insignaler: kör- och riktningssignal. Det visas också dess utsignaler: tillstånd som skickas till SP samt två interna styrsignaler för produktionssystemet. Alla operationer är implementerade på liknande sätt med standardiserade in- och utsignaler. Det som skiljer dem åt är deras insida och i visst fall någon extra in- eller utsignal. Tanken är att de för SP inte ska vara mer än svarta lådor. SP skall alltså inte behöver veta något mer om operationen än vad som krävs för att den ska börja köras, vad som förväntas ha hänt när operationen är klar och när den är klar. Kodblocket i figur 6 kan för SP ses som just en svart låda där innehållet inte är relevant.

För kommunikation mellan tjänster använder sig SP av en meddelandebuss. Meddelandebussen fungerar så att en aktör antingen kan skicka ut meddelanden inom ett visst ämne eller lyssna efter meddelanden inom ett annat ämne genom att prenumerera på det. Fördelen med detta är att alla aktörer inte behöver ta del av all information utan behöver bara ta del av den information som är väsentlig.

4.2.2 Modellering av det fysiska systemet i SP

Modelleringen av systemet är baserad på teorin om operationer som beskrivits tidigare i avsnitt 2.3.2. Valet att använda operationer baseras på att det är en modell som är enkel att förstå samtidigt som det går att hantera oförutsedda fel, givet att felhanteringen implementeras på rätt sätt. Modellen är uppdelad i två delar, förmågor och operationer.

Förmågor

En förmåga är avsedd att representera en fysisk förmåga i systemet. Förmågan består av ett startvillkor, en händelse och ett slutvillkor. Exempel på detta finns i figur 11.

```
val robot1toFeedCylPick = abapi.Ability(name = "robot1toFeedCylPick ", id = ID.newID,
  precondition = prop(allVars, s"${robot1State.name} == 1 and not ${robot1Run.name} and not" +
    s" ${robot1gripping.name} and ${feedSensor.name}"),
  List(s"${robot1Run.name} := true", s"${robot1Target.name} := 3")),
  postCondition = prop(allVars, s"${robot1State.name} == 3", List(s"${robot1Run.name} := false")),
  started = prop(allVars, s"${robot1State.name} == 2", List())
)
```

Figur 11: Ett exempel på kod för en förmåga

Förmågor är utformade på samma sätt som operationer beskrivs under 2.3.2 med tre olika tillstånd. I figur 11 visas förmågan som representerar att hämta en produkt vid mataren. Startvillkoret säger att roboten som ska hämta produkten måste vara i sitt initiala läge, representerat av en etta. Roboten ska heller inte vara i exekveringstillstånd eller hålla i en produkt. Det ska också finnas något att hämta i mataren. I listan finns de starthändelser som sker när förmågans startvillkor blir sant. En körsignal sätts till sann för roboten och vilken position den ska åka till sätts som ett heltal. Started säger att vi är i exekveringsläge om roboten är i läge 2. Till sist har vi slutvillkoret som säger att om robotens läge är 3, alltså färdig, ska körsignalen sättas till falsk.

Operationer

Operationerna är länkade ett till ett med förmågorna, en operation representerar en förmåga. Operationer behöver inte implementeras på detta sätt. En operation skulle kunna representera någon kombination av förmågor eller alternativt ingen förmåga alls. En operation utan förmåga skulle till exempel kunna vara en operation där modell och verkligt system jämförs. Där sker inget i det verkliga systemet och därmed behöver operationen ej kopplas till en förmåga då förmågor är något som är kopplat till resursernas förmågor. Som exempel på en kombination skulle en operation för att förflytta en produkt från a till b kunna innehålla två förmågor. Förmågan att hämta en produkt från a och sen förmågan att lämna den vid b.

I figur 12 kan det observeras att en stor skillnad mellan förmågor och operationer är att operationer inte har någon koppling till det fysiska systemet utan använder sig av interna variabler. Vissa av namnen på de interna variablerna är samma som för variablerna för förmågorna men representerar nu en position istället för ett värde.

```

robot1toFeedCylPick.copy(conditions = List(
  prop(vars, s"${things("feedSensor").id} == $cylName &&" +
    s" ${things("C1").id} == available && ${things("R1").id} == available",
    List(s"${things("R1").id} := unavailable"), "pre"),
  prop(vars, "", List(s"${things("feedSensor").id} := empty",
    s"${things("robot1").id} := $cylName"), "post")
)),

```

Figur 12: Ett exempel på kod för en operation

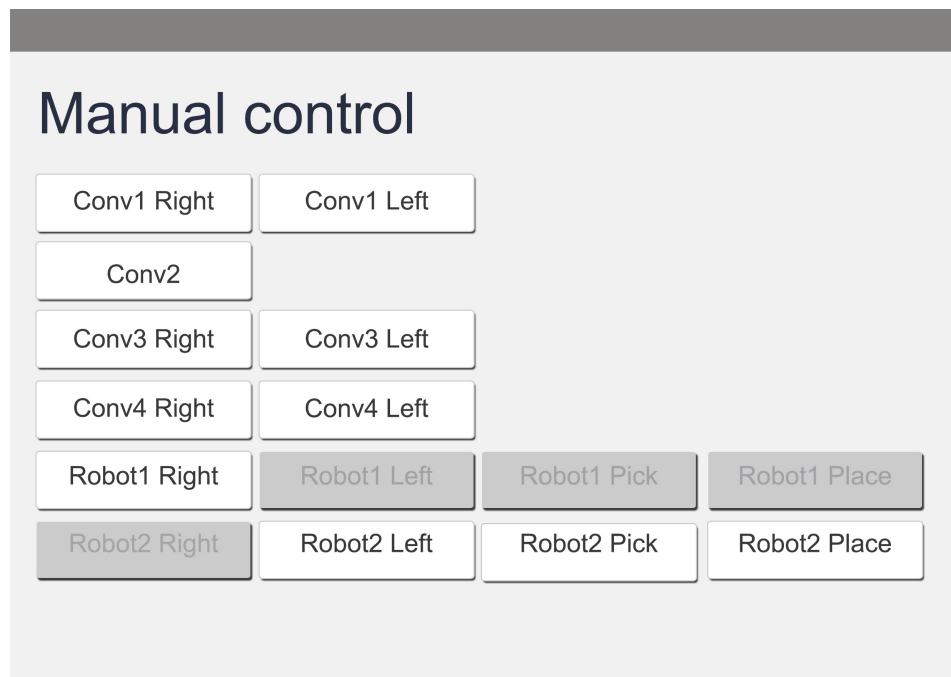
I figur 12, som är operationen som representerar förmågan i figur 11, kan det observeras att operationer har ett start och slutvillkor. Dessa villkor ser något annorlunda ut i jämförelse med villkoren i förmågorna. En del av startvillkoret säger att "feedSensor" ska vara lika med "cylName". CylName är en produkt, i produktionssystemet en cylinder. Villkoret säger att det ska stå en specifik produkt på platsen feedSensor. De interna variablerna C1 och R1 representerar resurser i produktionssystemet. C1 representerar Band 1 och R1 robot 1 i figur 5. Dessa interna variabler används för resursbokning. Resursbokning innebär vad namnet antyder, att man bokar resursen så att den bara hanterar en produkt i taget. För operationen i figur 12 ska resurs C1 och R1 vara tillgängliga för att starta operationen. När operationen startat sätts R1 till otillgänglig eftersom resursen nu används. Det finns inget villkor för att operationen ska vara färdig eftersom operationer inte har tillgång till sensorvärden. Istället blir operationen färdig när förmågan den är länkad till blir färdig. När det händer görs sluthändelserna, i exemplet att det inte står något på position feedSensor och att produkten nu är på position robot1.

4.2.3 Widgets

SP är uppbyggt på ett sätt där den visuella presentationen sker i webbläsaren. Skalet som finns uppbyggt i SP bygger på en så kallad single page application (SPA). En SPA ger möjligheten att presentera olika typer av innehåll och data för användaren i ett enda fönster på en webbsida. Det gör det lättillgängligt då det enda som behövs för att köra applikationen på sin dator är en webbläsare. Fönstret är sedan uppdelat i flera små program, så kallade widgets. Operatören kan själv anpassa fönstret så att rätt widgets ligger på rätt plats. Endast relevant information ska visas och därför ska olika operatörer ha möjligheten att anpassa fönstret efter deras arbetsuppgifter och preferenser. Några koncept till widgets för att testköra systemet och möjliggöra återstart har tagits fram.

Widget för manuell styrning

I figur 13 visas en widget där en användare har tillgång till att styra de operationer som existerar i systemet. Denna widget hjälper användaren att lösa problem som systemet inte kan lösa själv, sådana problem som är mer avancerade och kräver att en användare gör en visuell inspektion av det fysiska systemet för att kunna lösa problemet. Detta kräver också en grundläggande kunskap om systemet, då användaren ska veta vilka operationer som kan och bör köras. Däremot finns det en säkerhetsfunktion implementerad som gör att de operationer som inte går att köra, ska användaren inte kunna köra. Detta för att det ska vara så enkelt och användarvänligt som möjligt att styra systemet manuellt, men också för att det inte ska finnas någon risk att användaren ska kunna skada systemet. Exempel på detta ses också i figur 13 då de knappar som är gråa inte går att använda, detta kan exempelvis vara eftersom robot 1:s position är längst till vänster på dess ställning, vilket innebär att den enbart kan köras till höger. Det är inte heller möjligt för roboten att plocka eller lämna en produkt vid denna position då det vid denna position redan står en cylinder enligt sensorn.



Figur 13: Widget för att styra systemet manuellt

Widget för att lägga beställning

I figur 14 syns en widget där man kan lägga en beställning på en produkt. Möjligheten finns också att skräddarsy produkten. Det ska alltså vara möjligt att kunna bestämma vilka processer produkten ska genomgå för att få det slutresultat man önskar. Här beräknar sedan backend produktens specifika väg genom produktionssystemet beroende på vilken beställning som användaren har valt. Det är inget som presenteras för användaren utan något som endast sker i backend. Här ska det också visas vilka processer som är möjliga att göra och inga omöjliga val ska kunna genomföras. De beställda cylindrarna och vilka processer de ska passera visas sedan som en lista.

Order a cylinder

Cylinder 5

- ☒ Process 1
- ☒ Process 2
- ☒ Process 3
- ☒ Process 4
- ☒ Process 5
- ☒ Process 6

Make cylinder

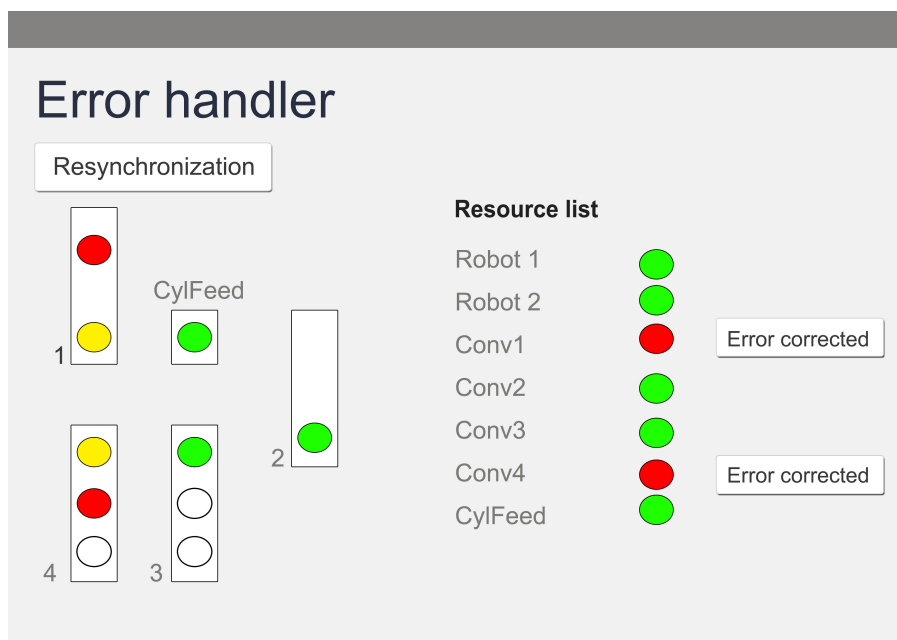
- Cylinder 0 Process 1 : true Process 2 : false Process 3 : false Process 4 : true Process 5 : false Process 6 : false
- Cylinder 1 Process 1 : false Process 2 : true Process 3 : false Process 4 : false Process 5 : true Process 6 : false
- Cylinder 2 Process 1 : true Process 2 : true Process 3 : false Process 4 : true Process 5 : false Process 6 : false
- Cylinder 3 Process 1 : false Process 2 : false Process 3 : false Process 4 : false Process 5 : true Process 6 : true
- Cylinder 4 Process 1 : false Process 2 : true Process 3 : false Process 4 : true Process 5 : false Process 6 : false
- Cylinder 5 Process 1 : true Process 2 : true Process 3 : true Process 4 : true Process 5 : true Process 6 : true

Figur 14: Widget för att beställa cylindrar

Widget för felhantering

Denna widget är den widget som sköter och visualiserar felhantering för användaren. Widgeten kan ses i figur 15. Ett grafiskt gränssnitt underlättar för en användare att ta till sig information[27]. Det första som användaren gör när den ska kontrollera om systemet är felfritt är att trycka på knappen "Resynchronization". Då kommer de andra delarna av widgeten att visas, alltså den visuella representationen av systemet samt resurslistan.

läs och kolla
efetr cirkel
lampa



Figur 15: Exempel på en widget som kan användas vid felhantering

Till vänster i figur 15 ses de olika rullbanden, numrerat ett till fyra, samt CylFeed. Det visas också de olika punkterna i systemet där fel kan upptäckas, det vill säga vid sensorerna. Vid de gröna cirklarna på rullbanden är allt som det ska vara. Den gröna färgen motsvarar att där ska det stå en cylinder och sensorn detekterar en cylinder. Detta kan ses vid den gröna lampan vid CylFeed, vilket är det rullband som matar produkter från en produkthållare. Om det hade varit något fel på produkthållaren och den ej har kunnat mata en produkt, skulle denna lampa ändra färg till röd. Om färgen på lampan är röd innebär det att en sensor förväntar sig att den ska ha en produkt, men inte har någon. Produkten har alltså aldrig nått dess slutstation, något har hänt och det är fel i systemet. När lampan är gul innebär det att systemet har en produkt, men egentligen inte ska ha någon på den platsen. Som det kan ses på band 1 är felet troligtvis att bandet inte har lyckats rulla, alltså att produkten aldrig rört sig från den ena till den andra sensorn.

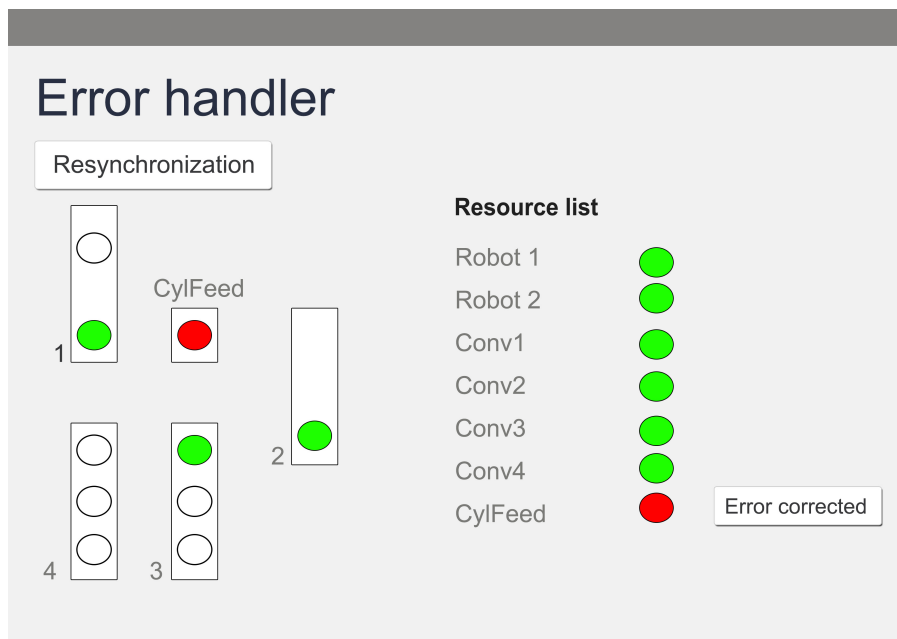
När lampan är fylld vit är det ingen produkt vid motsvarande plats, men det ska det heller inte vara, alltså har inget fel inträffat. När hela systemet fungerar som det ska så ska alla punkter vara gröna eller vita.

I den högra delen av figur 15 ses en överblick över de resurser som systemet använder. När de har en röd lampa efter dess namn innebär det att det är något fel gällande den resursen. Som vi ser är det enligt denna lista fel på band 1 (conv1 enligt figuren), det kan vi även se i den högra delen av bilden där, som skrivet ovan, är något fel som hänt på det bandet. Knappen bredvid den röda lampan där det står "Error corrected" är till för att operatören kan bekräfta för systemet att det kan göra ett återstarts försök igen. För att åtgärda ett fel så kan antingen operatören köra systemet manuellt och åtgärda felet för att sedan trycka på "Error corrected". För vissa lättare fel ska det räcka att gå in och laga det som är trasigt och sedan trycka på "Error corrected" och systemet kommer då automatiskt att göra om den senaste operationen som den misslyckades med. När det är en grön lampa vid en resurs betyder det att det inte är något fel.

4.3 Lösningar på felscenarion

För att lösa scenarierna som är specificerade i avsnitt 3.2 så krävs det oftast implementation både i PLC:n och i det överordnade systemet men i vissa fall räcker det med implementation i det överordnade systemet. Lösningar är implementerade så att de ska fungera relativt generellt. De ska alltså inte endast vara specialimplementerade för just den delen i just detta systemet. En generell lösning som fungerar för att upptäcka många fel är att systemet jämför sin mjukvarumodell med verkligheten för att se så att de är synkroniserade.

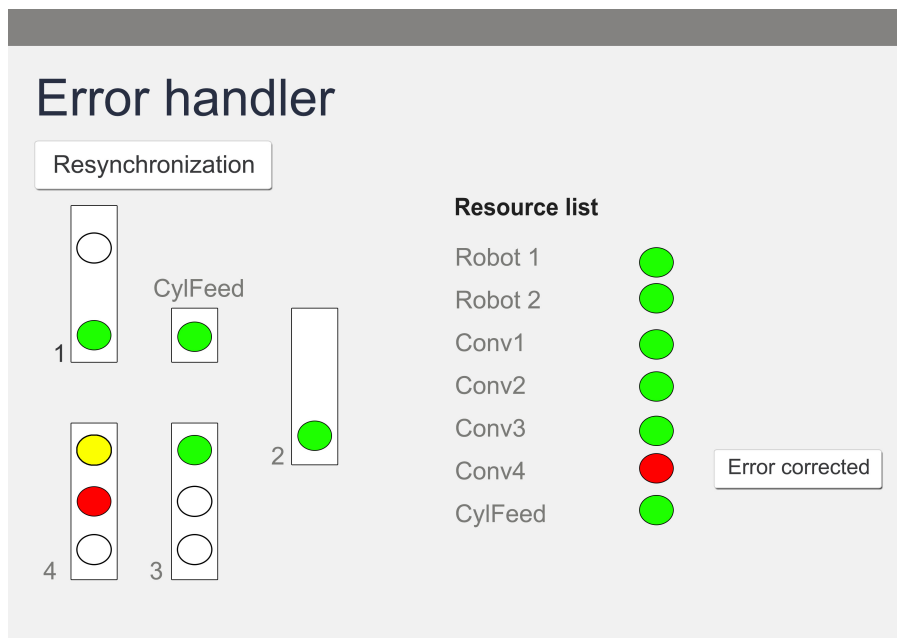
För att lösa det första scenariot, det är slut på cylindrar i mataren (feedsensor i figur 5), krävs det endast implementering i det överordnade systemet. Om tornet är tomt så kommer mataren att mata en gång och sedan gå över till sluttillstånd. Det som då händer är att modellen kommer att uppdateras till att det just nu står en cylinder vid matarsensorn. I det fysiska systemet står där dock ingen cylinder. Det överordnade systemet kommer då att upptäcka felet genom att jämföra modellens tillstånd med det fysiska tillståndet. Sedan kommer vyn uppdateras i widgeten för felhantering, avsnitt 4.2.3. Det kommer då rödmarkeras på mataren och operatören vet då att det saknas produkter i mataren enligt figur 16. Det fylls sedan på manuellt av operatören. Att felet är åtgärdat bekräftas i widgeten och systemet kan starta upp igen och återstartar då vid punkten strax före att den försökte mata ut en cylinder.



Figur 16: Utseende på felhanteringswidget för scenario 1: Slut på cylindrar

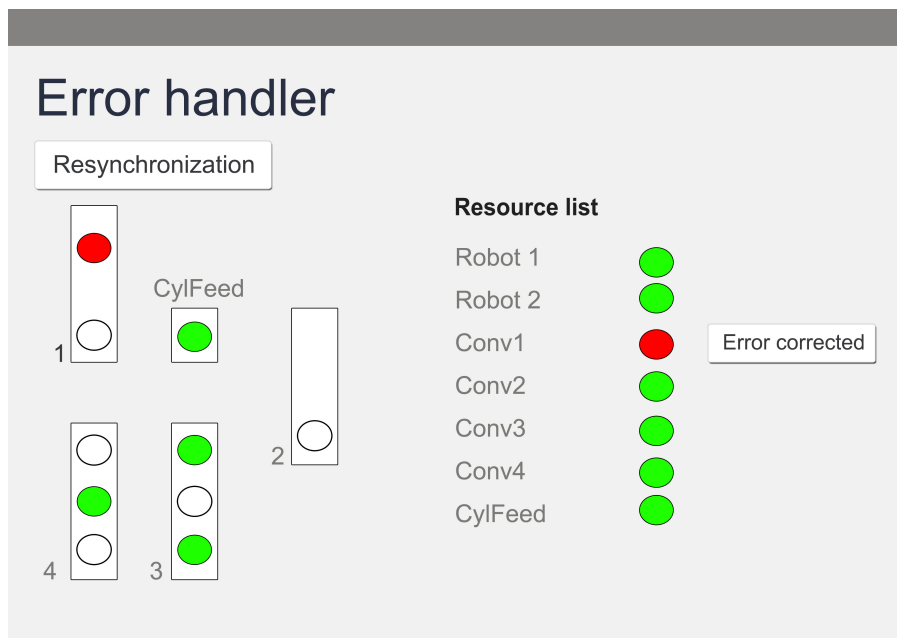
Det andra scenariot, där det är ett rullband som blir strömlöst, krävs implementering både i PLC:n och i det överordnade systemet. I PLC:n behöver det kodas in att ett rullband aldrig försöker köra längre tid än vad som är tillräckligt för att nå en sensor. Om bandet skulle rulla längre skulle det tillslut rulla en cylinder av bandet och det är något som inte får hända då produkten skulle kunna gå sönder. Därför måste specifika tider för varje förflyttning mätas upp och sedan läggas till i koden. PLC:n kommer då att rulla banden antingen tills sensorn går hög eller tills den bestämda tiden har gått. Den kommer då gå in i sluttillståndet som att operationen var klar. Det som då händer är att efter de två systemen handskakat så kommer det överordnade att jämföra sin modell med verkligheten. Den kommer att ha uppdaterat modellen att det numera ska stå en cylindern på nästa sensor på

bandet. Detta stämmer inte med verkligheten där produkten antingen fortfarande står kvar på den tidigare sensorn eller någonstans mittemellan, oavsett vad så saknas den på sin rätta plats. Widgeten för felhantering kommer då att lysa med en röd lampa vid den sensorn som ska ha cylindern medans den där den står kvar kommer att lysa gul enligt figur 17. Lampan vid resursen conv4 kommer också att bli röd. Detta uppmärksammar operatören på att något är fel vid band 4 och de två lamporna på bandet ger även information om vad som kan tänkas vara fel. Operatören behöver laga det som är trasigt vid bandet för att sedan bekräfta i widgeten att systemet är körbart igen. Antingen har operatören själv kört fram bandet med den manuella widgeten för att testa så att bandet fungerar igen eller så låter operatören systemet att återstarta självt genom att köra den senaste operationen.



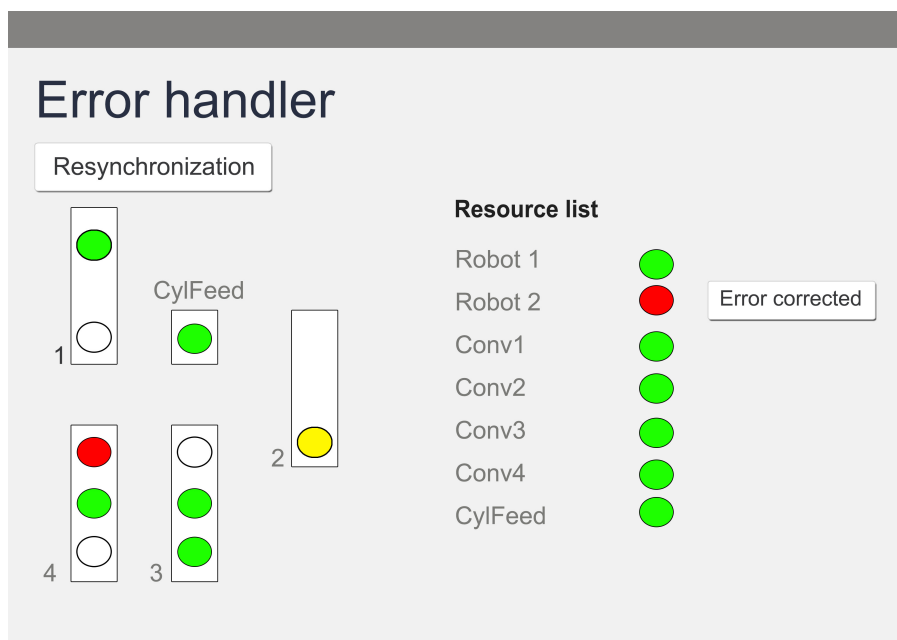
Figur 17: Utseende på felhanteringswidget för scenario 2: Band 4 är trasigt

Det tredje scenariot, sensorfel, innebär att en sensor ej längre reagerar på cylindrarna. För att lösa detta krävs samma typ av implementering som i scenario två. Det behövs maxtider för körning för att ej riskera att hamna utanför det tänka rörelseområdet där skador på produkter och resurser kan uppstå. Det är svårt att särskilja ett sensorfel ifrån andra fel då sensorerna är något som används för att upptäcka även andra fel. Sensorfelet där en sensor inte längre reagerar på en cylinder kommer att upptäckas genom att bandet rullar sin maxtid och operationen hamnar i sitt sluttillstånd. Efter synkronisering av modell och verklighet så kommer det överordnade systemet att upptäcka att det saknar en cylinder vid slutmålet för operationen. Operatören uppmärksammas då på det genom att den sensors lampor tänds till röd. I exemplet i figur 18 syns hur en sensor vid band 1 är trasig. Operatören får då gå ut till produktionssystemet och identifiera felet. Här är det operatören som får avgöra om det är ett sensorfel eller ett fel på bandet. Skillnaden på de två felen följer ett mönster som är lätt att förstå. Har cylindern åkt för långt är det ett sensorfel. Står däremot cylindern någonstans tidigare på bandet mellan de två sensorerna så är det ett fel på bandet. Sensorn får bytas eller lagas och sedan kör operatören manuellt fram cylindern för att testa om sensorn fungerar igen. Därefter bekräftas det i widgeten att systemet är körbart igen och det kan fortsätta där det slutade.



Figur 18: Utseende på felhanteringswidget för scenario 3: Sensor vid rullband 1 är trasig

För att angripa det fjärde scenariot där sugfunktionen hos roboten slutar att fungera krävs det endast den tidigare beskrivna implementeringen i det överordnade systemet. Roboten kommer att åka och försöka plocka upp en cylinder. Antingen så lyckas roboten plocka upp cylindern men kanske tappar den på vägen eller så får roboten inte ens med sig cylindern från bandet. Det som då händer är att när roboten sedan åkt och lämnat så övergår operationen till sitt sluttillstånd. Men som vid de tidigare scenariorna så kommer inte modellen och det verkliga systemet att stämma överens med varandra. Då måste vyn i felhanteringswidgeten uppdateras. Mottagarsensorns lampa blir röd, lampan vid band 2 lyser gult och även robotens lampa lyser rött som syns i figur 19. Det överordnade systemet vet att den resursen som skulle ha flyttat cylindern till sensorn är trasig då det ej kommit fram någon cylinder. Just att systemet vet vilken resurs som jobbar gör att den lättare kan urskilja fel och då ge mer information till operatören. Operatören måste sedan gå ut och laga sugverktyget på roboten för att sedan bekräfta i widgeten att systemet är fixat och kan återstartas. Om roboten har tappat cylindern kan det krävas mer jobb för operatören att återställa produkten. Att tappa cylindern får egentligen inte ske då detta kan ge stora skador om produkterna är tunga och dyra så verktygen ska servas regelbundet för att undvika skador. Om däremot cylindern står kvar på sin tidigare plats som i figuren så kan systemet självt köra den senaste operationen den genomförde och därmed vara igång igen.



Figur 19: Utseende på felhanteringswidget för scenario 4: Sugan på robot 2 trasig

Det femte och sista scenariot, strömavbrott i produktionen, kräver implementering i det överordnade systemet. Då det blir ett strömavbrott så kommer alla sensorer att gå låga och inga resurser fortsätter köra. När detta händer kommer modellen och det fysiska systemet att hamna helt ur synkronisering och det kommer att bli många röda lampor i felhanteringswidgeten. När systemet sedan återfår strömmen och sensorerna börjar reagera igen så kommer antagligen några lampor att ändra färg och bli gröna och gula. Det som krävs vid ett strömavbrott är att operatören får gå in och manuellt styra cylindrar till de platser där en röd lampa lyser. Det överordnade systemet vet vilken cylinder som den förväntar sig vid varje sensor och utifrån den informationen så kan operatören köra rätt cylinder till rätt plats. När systemet endast har gröna och vita lampor vid sensorerna så kan operatören bekräfta via widgeten att systemet är redo för att återuppta normal drift.

5 Diskussion

Stora delar av resultatet är fortfarande i koncept- och utvecklingstadiet och således föremål för vidare utveckling. I korthet så har grunderna för ett återstartsbart system byggts upp och studier utförts för att vidare utveckla systemet för att kunna analysera återstart och problemen runt detta.

En operationsmodell har blivit implementerad med bra resultat. Eftersom våra operationer är implementerade för att göra endast en sak var och att vara isolerade ifrån resten av systemet är de enkla att förstå och felsöka. Dessutom går det enkelt att lägga till och ta bort både operationer och förmågor från systemet om det fysiska systemet skulle ändras. Operationerna som skapats gör så pass lite i varje operation och att använda sig av dem har gjort det lätt att kunna köra systemet manuellt utan vidare kunskap om systemet. Denna typ av lätta små operationer gör det även lätt att hitta återstartspunkter. Vid granskning av de uppsatta felscenarierna och dess lösningar så upptäcks det att med de små korta operationerna så blir återstartspunkterna lättare att hitta. Återstartspunkterna blir antingen att gå tillbaka till innan operationen och försöka igen eller att operatören har kört klart operationen manuellt och systemet kan då fortsätta som att inget har hänt.

Användningen av operationer i PLC:n har gjort koden enklare och mer överskådlig. Att koden i PLC:n och det överordnade systemet följer en gemensam modell har också gjort koden i PLC:n lättförståelig. Det går enkelt att ta bort och lägga till nya operationer utan att ändra mycket i koden. De går enkelt eftersom operationsblocken bara använder sig av grundläggande funktioner för systemet och inte beroende av andra operationer. Frikopplingen ger ett stort återanvändningsvärde av operationerna.

Tidigare forskning som studerat återstart visar att om systemet kan beskrivas med en graf går det också att identifiera oförutsedda fel. Den implementerade operationsmodellen är en typ av graf med platser, övergångar och bågar vilket gör det möjligt att implementera en generell felsökningsmetod baserad på grafsökning. Vidare hittades också en metod där flera återstartspunkter används. Metoden med återstartspunkter är något som liknar lösningen i projektet. Återstartsoperationer behöver implementeras för att kunna ta sig till återstartspunkterna och sedan fortsätta produktionen därifrån. Det är en utmaning att få systemet att automatiskt göra detta och det är i många lägen lättare att låta en operatör göra det. Operatören krävs ofta ändå för att kunna laga vad som är trasigt vid fel, det blir då naturligt att operatören själv utför återstartsoperationen som tar systemet tillbaka till en återstartspunkt och därmed korrekt synkronisering.

Som det syns i resultatet så är felhanteringen ganska så generell. Efter att felen är åtgärdade ser återstartsprocessen liknande ut för de olika fallen. Det viktiga är att hitta tillbaka till en position där det övergripande systemet kan åter synkroniseras med det fysiska systemet. Tack vare modellen som kan jämföras med det verkliga systemet så går det att upptäcka även oplanerade fel. Felen behöver ej vara kända och planerade för innan. Så länge det fysiska systemet ej gjort vad operationen förväntar sig så ska programmet säga vart felet uppstått och en operatör får sedan göra ytterligare identifikation.

Ett tydligt och bra användargränssnitt tros underlätta tillräckligt mycket för att även en oerfaren operatör skall kunna återstarta systemet med den hjälp denne får i från de utvecklade widgetarna. Att utforma ett användargränssnitt som är lättförståeligt är dock inte alltid det lättaste utan kräver mycket eftertanke och testning. Ett försök att jobba grafiskt i widgeten har gjorts för att underlätta förståelsen och identifieringen av fel. Det kan vara lättare att se en grafisk bild som representerar systemet och vart ett fel har uppstått än att utläsa det från en lista med variabler. Nackdelen med en grafisk uppbyggnad är att det krävs en ny specifik grafisk presentation för nya produktionssystem.

5.1 Utvärdering av projektets mål

Ett produktionssystem kan utformas för felidentifiering och åtgärd av oförutsedda fel. Om produktionssystemet utformas på ett sätt så att det går att representera som en graf och har återstartsövergångar gör detta det möjligt att implementera felidentifiering och åtgärd av oförutsedda fel. Det kräver en tydlig och bra operationsmodell som lätt kan jämföras med verklighetens sensorvärden.

Ett förslag på grafisk presentation av information har utvecklats men någon studie för verifiering av att det skulle förenkla har ej utförts. Den grafiska representationen bör med stöd i tidigare studier[27] göra det enklare för operatören att ta till sig information och förstå problemet.

Det är möjligt att åtgärda fel automatiskt. Baserat på behandlade metoder så kan fel identifieras med en observatör som kollar efter otillåtna tillstånd utifrån en graf som beskriver systemet. Felen kan sen åtgärdas med hjälp av algoritmer som hittar möjliga återstartspunkter och möjliga vägar till dessa. Om felet uppstått i det fysiska systemet kommer dock fortfarande en operatör att krävas för att laga det. Det som kan ske automatiskt är snarare själva återgången till ett tillstånd varifrån det kan fortsätta produktion.

5.2 Problem och svårigheter med återstart

Då felhantering har undersökts så har det uppdagats flera problem som gör att felhantering ofta inte går att se som en generell lösning utan som en kombination utav flera avvägningar. Vid implementering av återstart arbetas det med ett begränsat antal sensorer. Detta ger en begränsad mängd punkter där fel kan identifieras. Detta har sina fördelar och nackdelar. För få sensorer gör det svårt att identifiera vart ett fel har uppstått och för många sensorer ökar risken för sensorfel. Med få sensorer kan det också bli problem att inte bara upptäcka felet men också identifiera vilket typ av fel det är. Sker det för många händelser mellan två platser i systemet som kan kontrolleras så kan mängden av möjliga fel bli för stor. Många möjliga fel gör att ingrepp av en kunnig operatör blir nödvändigt för att styra upp situationen. Kunniga operatörer är inte alltid lätt att få tag på och därför fördelaktigt om det går att underlätta för en operatör.

De sensorer som har använts är mycket enkla och kan bara bekräfta att produkter är på diskreta ställen i systemet. Trasiga enkla sensorer, som bara visar ett sant eller falskt värde, är enklare att felsöka just för att de genererar begränsade data. Med mer kontinuerlig och avancerad bevakning av systemet genereras mer data som kan vara till hjälp att analysera närmare vad som har gått fel. Men mer avancerad teknik kan också medföra mer svåranalyserad data. Om övervakningen blir avancerad nog så att systemet helt beror på att övervakningen ska fungera ställer detta höga krav på att övervakningen är felsäker. Med till exempel kameror som kontinuerligt skickar stora mängder data finns också mer data som kan vara felaktig. Skulle all eller del av denna data vara felaktig kan detta få större konsekvenser på systemet baserat på dess beroende av informationen. Det finns risker med att ju fler sensorer och system som införs så införs också potentiellt fler felkällor.

Att implementera återstartsprinciper i redan befintliga system kan vara svårt. PLC:n behöver vara programmerad med en speciell struktur motsvarande vald återstartsметод för att möjliggöra återstart. Med en tydlig struktur bestående av enkla operationer går det att koppla PLC:n till ett överordnat system som utvärderar och hanterar fel. Många system är inte utvecklade med återstart i åtanke från början utan har gjort denna implementationen i efterhand. Om befintliga system ska implementera bättre återstartsметодer kan PLC:n behöva programmeras om helt. Varje minut är värdefull i industrin och den tiden produktionen behöver vara stoppad under uppdatering gör företag omotiverade att göra förändringar.

Vad som också kan vara svårt är att faktiskt ta sig tillbaka till en bra återstartspunkt. Det finns

inget uppenbart sätt att låta systemet självt hitta tillbaka. Det finns så många olika fall och vissa av fallen kan systemet lösa automatiskt medan vissa kan vara omöjliga. Vad mer är att två olika fel kan se ut på samma sätt i det överordnade systemet, såsom i våra felscenario 2 och 3. I vårt system kommer dessa två fel upptäckas som att en cylinder inte nått fram till en förväntad punkt, även om orsaken till detta fel är olika i de två fallen. Det är svårt att bygga upp systemet på ett sätt där det självt kan avgöra när det har möjlighet att lösa det utan inverkan av operatör. Med det sagt verkar de flesta hårdvarufel, som tidigare nämnt, kunna åtgärdas på liknande sätt. Systemet upptäcker vart något är fel och gör operatören uppmärksam på detta. Operatören går sedan in och lagar det som gått sönder varefter systemet antingen behöver backa ett steg och göra om operationen som misslyckades eller fortsätta direkt till nästa operation.

Vidare undersökningar och forskning som varit intressant inom ämnet hade varit att undersöka hur många sensorer som är nödvändiga för vår typ av felhantering. Ett system med enkla och isolerade operationer som bara gör en sak och sensorer emellan dessa verkar räcka för vårt system. Detta behöver dock ej vara fallet för ett riktigt produktionssystem i industrin. Noggrannheten för hur man behöver kontrollera att varje steg av processen har genomförts och att produkterna har nått ett förväntat läge är ej självklar. Det kan vara något som inte går att generalisera utan snarare blir specifikt till olika steg av produktionen. Detta utgör centrala punkter att tänka på för hur generell man kan göra felhanteringen. Det blir en optimeringsfråga mellan hur många delar som krävs för att systemet ska kunna genomföra självkontroll kontra hur många ytterligare delar som kan gå sönder.

6 Slutsats

Det är möjligt att utveckla mjukvara som underlättar för återstart av ett system. Ett bra sätt att implementera det på är att implementera en välstrukturerad operationsmodell som tillsammans med interna variabler beskriver det verkliga systemet. Mjukvarumodellen och det verkliga systemet måste sedan jämföras med varandra efter varje operation och kontrollera att det som modellen förväntar sig ska ha skett verkligen har skett i det fysiska systemet. Om olikheter uppstår här så har systemet en relativt klar bild av vilken typ av fel som kan ha uppstått. Vissa fel kan specificeras medans andra istället får pekas ut vart de har hänt men där ingen åtgärd kan föreslås. Att bygga en perfekt operationsmodell är inget som är helt självklart hur det ska göras utan behöver värderas för situationen. Men med idén att operationerna ska göra så lite som möjligt så underlättas processen med att hitta återstartspunkter och därmed felhanteringen.

Att genomföra en totalt automatiserad återstart är ofta inte det lättaste då hårdvaruresurser för att åtgärda fel sällan finns tillgängliga. Det krävs alltså en operatör för att laga felen och därmed blir det också naturligt att operatören har en viktig del i identifiering av fel. Som en del i identifiering och lagning kan operatören behöva styra manuellt för testning. Då kan också systemet iordningställas av operatören till ett läge varifrån det sedan kan återstartas eftersom denne redan styr systemet manuellt.

Referenser

- [1] Deloitte, “The Deloitte Consumer Review Made-to-order: The rise of mass personalisation”, Deloitte, London, tekn. rapport, 2015. URL: <https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/consumer-business/ch-en-consumer-business-made-to-order-consumer-review.pdf>.
- [2] J. Kyung, C. Park, D. Park, H. Do och T. Choi, “Real-time synchronisation method in multi-robot system”, *Electronics Letters*, vol. 50, nr 24, s. 1824–1826, nov. 2014, ISSN: 0013-5194. DOI: 10.1049/el.2014.2959. URL: <http://digital-library.theiet.org/content/journals/10.1049/el.2014.2959>.
- [3] P. Bergagård, P. Falkman och M. Fabian, “Modeling and automatic calculation of restart states for an industrial windscreen mounting station”, i *IFAC-PapersOnLine*, vol. 48, 2015, s. 1030–1036. DOI: 10.1016/j.ifacol.2015.06.219. URL: <http://linkinghub.elsevier.com/retrieve/pii/S2405896315004589>.
- [4] BMBF, *Industri 4.0 - BMBF*. URL: <https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html> (hämtad 16 febr. 2017).
- [5] M. Tabikh, “Downtime cost and Reduction analysis: Survey results”, diss., Mälardalen University, 2014, s. 50. URL: <http://www.diva-portal.org/smash/get/diva2:757534/FULLTEXT01.pdf>.
- [6] E. R. Alphonsus och M. O. Abdullah, “A review on the applications of programmable logic controllers (PLCs)”, *Renewable and Sustainable Energy Reviews*, vol. 60, s. 1185–1205, 2016, ISSN: 13640321. DOI: 10.1016/j.rser.2016.01.025.
- [7] T. Young, *How PLCs Work*. URL: http://www.plcdev.com/how%7B%5C_%7Dplcs%7B%5C_%7Dwork (hämtad 21 febr. 2017).
- [8] A. Lakshmi Sangeetha, B. Naveenkumar, A. Balaji Ganesh och N. Bharathi, “Experimental validation of PID based cascade control system through SCADA–PLC–OPC and internet architectures”, *Measurement*, vol. 45, nr 4, s. 643–649, 2012, ISSN: 02632241. DOI: 10.1016/j.measurement.2012.01.005.
- [9] PLC-manual, *PLC Networks, Industrial LANs / PLC Manual*. URL: <http://www.plcmanual.com/plc-networks> (hämtad 21 febr. 2017).
- [10] W. Bolton, *Programmable logic controllers*. Elsevier/Newnes, 2006, ISBN: 9780750681124.
- [11] W. M. Gentleman, “Off-the-Shelf Black Boxes for Programming”, *IEEE Transactions on Education*, vol. 12, nr 1, s. 43–50, 1969.
- [12] R. G. Lanergan och C. A. Grasso, “Software Engineering with Reusable Designs and Code”, *IEEE Transactions on Software Engineering*, vol. SE-10, nr 5, s. 498–501, 1984.
- [13] T. Reenskaug, *Models -Views -Controllers*, 1979. URL: http://heim.ifi.uio.no/%7B%5C_%7Dtrygver/1979/mvc-2/1979-12-MVC.pdf (hämtad 9 maj 2017).
- [14] A. Farooqui, P. Bergagard, P. Falkman och M. Fabian, “Error handling within highly automated automotive industry: Current practice and research needs”, i *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2016-Novem, 2016, ISBN: 9781509013142. DOI: 10.1109/ETFA.2016.7733628.
- [15] P.-O. Börnfelt, *Arbetsorganisation i praktiken : en kritisk introduktion till arbetsorganisationsteori - P-O Börnfelt - Häftad (9789144118529) / Bokus*, 2. utg. Lund: Studentlitteratur AB, 2016, s. 156.
- [16] K. Andersson, B. Lennartson och M. Fabian, *Figoperations*, 2010. URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5357474> (hämtad 2 maj 2017).

- [17] K. Bengtsson, C. tekniska högskola. och Chalmers tekniska högskola. Institutionen för Signaler och System., *Flexible design of operation behavior using modeling and visualization*. Chalmers University of Technology, 2012, s. 215, ISBN: 9789173857390. URL: <http://chans.lib.chalmers.se/record=b1564899>.
- [18] I. Cox och N. Gehani, "Exception handling in robotics", *Computer*, vol. 22, nr 3, s. 43–49, mars 1989, ISSN: 0018-9162. DOI: 10.1109/2.16224. URL: <http://ieeexplore.ieee.org/document/16224/>.
- [19] M. P. Fanti och C. Seatzu, "Fault diagnosis and identification of discrete event systems using Petri nets", i *2008 9th International Workshop on Discrete Event Systems*, IEEE, 2008, s. 432–435, ISBN: 978-1-4244-2592-1. DOI: 10.1109/WODES.2008.4605985. URL: <http://ieeexplore.ieee.org/document/4605985/>.
- [20] C. Syan och Y. Mostefai, "Status monitoring and error recovery in flexible manufacturing systems", *Integrated Manufacturing Systems*, vol. 6, nr 4, s. 43–48, aug. 1995, ISSN: 0957-6061. DOI: 10.1108/09576069510088961. URL: <http://www.emeraldinsight.com/doi/10.1108/09576069510088961>.
- [21] P. Bergagård och M. Fabian, "Calculating Restart States for Systems Modeled by Operations Using Supervisory Control Theory", *Machines*, vol. 1, s. 116–141, 2013, ISSN: 2075-1702. DOI: 10.3390/machines1030116. URL: www.mdpi.com/journal/machines.
- [22] M. Zhou och F. Dicesare, "Adaptive design of Petri net controllers for automatic error recovery", *Proceedings IEEE International Symposium on Intelligent Control 1988*, s. 652–657, 1988. DOI: 10.1109/ISIC.1988.65508.
- [23] P. Loborg och A. Tiirne, "Manufacturing Control System Principles supporting Error Recovery", i *AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, 1994, s. 101–108.
- [24] V. Hajarnavis och K. Young, "An investigation into programmable logic controller software design techniques in the automotive industry", *Assembly Automation*, vol. 28, nr 1, s. 43–54, febr. 2008, ISSN: 0144-5154. DOI: 10.1108/01445150810849000. URL: <http://www.emeraldinsight.com/doi/10.1108/01445150810849000>.
- [25] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian och P. Falkman, "Sequence Planner: Supporting Integrated Virtual Preparation and Commissioning",
- [26] OPC Foundation, *Unified Architecture - OPC Foundation*. URL: <https://opcfoundation.org/about/opc-technologies/opc-ua/> (hämtad 9 maj 2017).
- [27] N. Staggers, D. Kobus, W. R. S. LA, D. M och P. DJ, "Comparing Response Time, Errors, and Satisfaction Between Text-based and Graphical User Interfaces During Nursing Order Tasks", *Journal of the American Medical Informatics Association*, vol. 7, nr 2, s. 164–176, mars 2000, ISSN: 1067-5027. DOI: 10.1136/jamia.2000.0070164. URL: <https://academic.oup.com/jamia/article-lookup/doi/10.1136/jamia.2000.0070164>.