Afstudeerverslag

Ethernet communicatie tussen PC en PLC

Naam: Rik Bulthuis

Studienummer: 08057567

Datum: 27-3-2013

Inhoudsopgave

1. Inleiding

Het bedrijf Weighpack International te Den Haag, maakt machinelijnen die aangestuurd worden door één of meerdere PLC's. Naast sommige machinelijnen staat een PC die extra functionaliteit toevoegt aan de machinelijn. Deze extra functionaliteit wordt toegevoegd in de vorm van software. Deze software heet OMS. Op dit moment wordt er door de PC met de machinelijn gecommuniceerd door middel van een profibus-kaart in de PC. Deze kaart wordt aangestuurd door middel van OPC software.

Weighpack wil graag gaan werken met een ethernet verbinding tussen de PC en de PLC. De redenen hiervoor zijn dat er dan geen aparte OPC software meer geïnstalleerd hoeft te worden er geen aparte kaart meer geleverd hoeft te worden en de verbinding stabieler is.

Voordat de software aangepast gaat worden is het belangrijk dat gekeken word naar hoe gecommuniceerd gaat worden met de PLC. Daarom moet er eerst een protocol opgesteld worden waardoor duidelijk wordt wat voor gegevens er over de ethernet verbinding gaan lopen.

Om de ethernetverbinding tot stand te brengen moet de PLC software aangepast worden en een aanpassing gemaakt worden in OMS.

2. Het bedrijf

Weighpack International is een bedrijf in de metaalbewerkingsindustrie. Weighpack International is fabrikant van Weeg-, Tel-, Verpakkings- en Inspecteermachines. Deze machines worden ontwikkeld voor bedrijven die technische producten maken zoals diverse soorten bevestigingsmaterialen, metaal en kunststofproducten.

Weighpack International is opgericht in 1974. Het is een relatief klein bedrijf met ongeveer 40 werknemers. Het bedrijf is internationaal actief, haar producten worden over de hele wereld verkocht .

Het bedrijf heeft de volgende afdelingen:

- Management
- Inkoop
- Verkoop
- Mechanische engineering
- Elektrische engineering
- Software engineering
- Magazijn

Ik ben zelf werkzaam op de afdeling Software engineering. Het project wordt ook uitgevoerd op deze afdeling. In diagram 1 staat het organisatiediagram van weighpack.

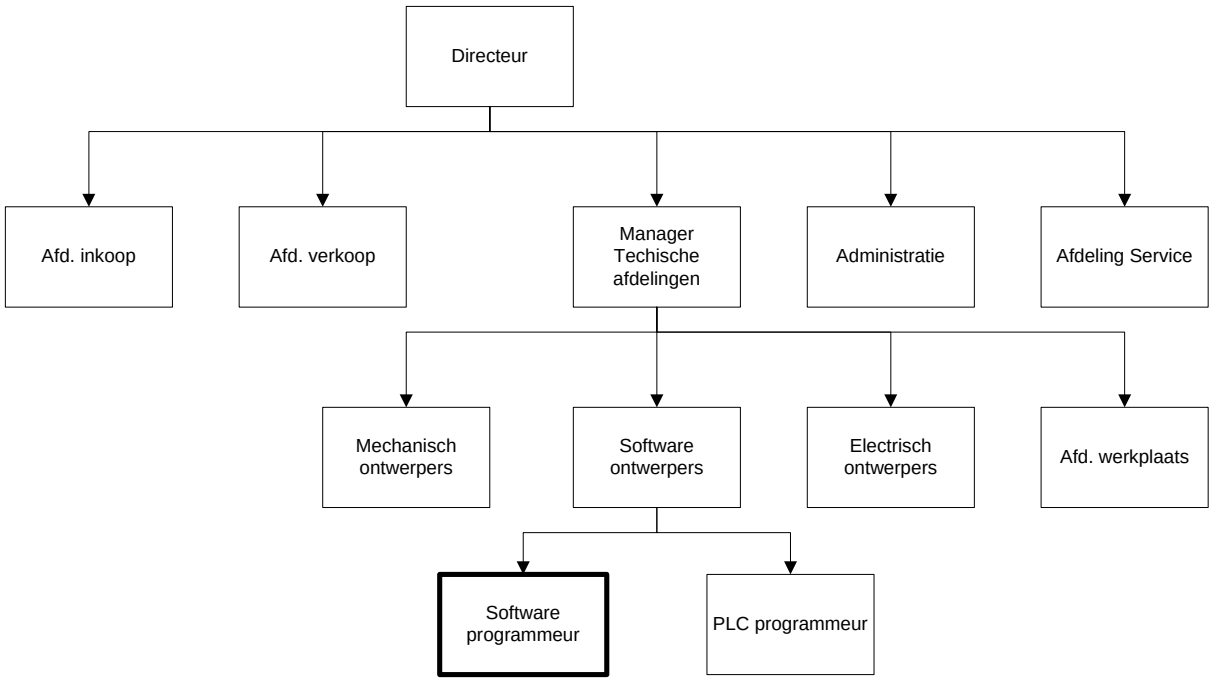


Diagram 1 – Organogram Weighpack

3. Doelstelling en Plan van aanpak.

3.1 Doelstelling.

De doelstelling is het gemakkelijker en betrouwbaarder maken van communicatie van de PC naar de PLC en andersom door middel van ethernet. Dit wordt bereikt door software te ontwikkelen die het voor de PC mogelijk maakt te communiceren met de PLC.

De te ontwikkelen producten zijn:

- Product 1: Een protocol waarmee gecommuniceerd gaat worden tussen PC en PLC.
-
- Product 2: Een functieblok voor de PLC waardoor de PLC via ethernet kan communiceren met de PC
-
- Product 3: Een versie van het programma OMS dat via ethernet kan communiceren met de PLC

3.2 Plan van aanpak

3.2.1 Achtergrond

In de meeste machinelijnen die worden ontwikkeld zijn PLC's opgenomen die de besturing van de lijn regelen. Deze PLC's hebben een apart scherm waar diverse instellingen van de machinelijn op kunnen worden ingegeven. Deze instellingen verschillen per product dat verwerkt word door de lijn. De verzameling instellingen per product wordt een **recept** genoemd. In diagram 2 staan twee voorbeelden van een recept weergegeven.

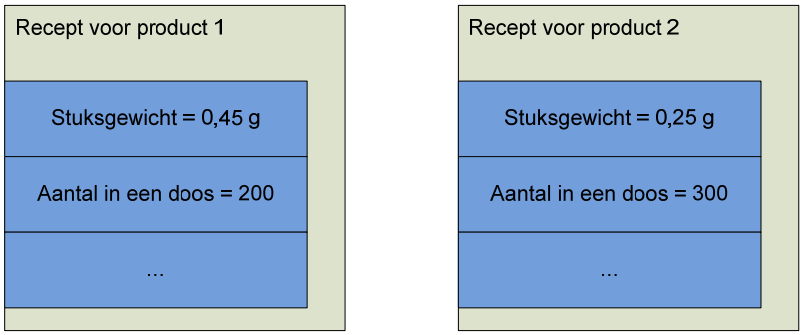


Diagram 2 – Recepten voor producten

PLC's hebben de mogelijkheid om een beperkt aantal recepten op te slaan. Het is hierdoor mogelijk om het recept van een bepaald product te laden uit de PLC. Dit voorkomt dat de **operator**, de persoon die de lijn aanstuurt, alle instellingen handmatig moet aanpassen als er van product gewisseld word. Hij kiest gewoon het recept dat bij het nieuwe product hoort en de instellingen worden automatisch goed gezet.

De PLC heeft een beperkte opslagcapaciteit en kan maximaal ongeveer 50 recepten opslaan, afhankelijk van het type PLC en de opslagcapaciteit van de geheugenkaart die erin zit. Omdat bij sommige lijnen met heel veel verschillende recepten word gewerkt is er ongeveer 10 jaar geleden besloten om een computer naast de machinelijn te plaatsen en hier de recepten in op te slaan.

Voor het opslaan van recepten op de computer in een computerprogramma ontwikkeld dat *OMS* heet. Dit staat voor Operator Management System. OMS heeft als doel recepten voor een machinelijn op te slaan en deze recepten naar de machinelijn te sturen.

OMS communiceert op dit moment via een Profibus-kaart, de CP5611, met de PLC. Deze kaart communiceert doormiddel van het protocol Profibus DP met de PLC. Voordat de Profibus-kaart goed werkt moet aparte software, de OPC software, geïnstalleerd worden op de computer.

In het verleden zijn er bij Weighpack veel problemen geweest met de communicatie naar de PLC met de OPC kaart. De software is lastig in te stellen en niet altijd betrouwbaar. Om deze reden heeft Weighpack besloten om over te gaan op ethernet modules op de PLC zodat er via ethernet gecommuniceerd kan worden met de PLC.

3.2.2 Opdrachtomschrijving

3.2.2.1 Opdrachtgever en opdrachtnemer

Opdrachtgever

Naam bedrijf:	Weighpack International BV
Naam opdrachtgever:	Kevin Kloosterman
Naam contactpersoon:	Kevin Kloosterman

Opdrachtnemer

Naam:	Rik Bulthuis

3.2.2.2 Probleemstelling

Het bedrijf Weighpack heeft een probleem met het sturen van instellingen naar de PLC's van machinelijnen. De huidige manier van communiceren met de machinelijnen gaat door middel van seriële communicatie met een OPC kaart. Er moet ingewikkelde software geïnstalleerd worden voordat dit werkt en het is onbetrouwbaar omdat de OPC kaart soms niet goed is ingeprikt of de kabel losraakt. Daarom wil het bedrijf overgaan op ethernet communicatie naar de PLC.

3.2.2.3 Doelstelling opdracht

De doelstelling is het gemakkelijker en betrouwbaarder maken van communicatie van de PC naar de PLC en andersom door middel van ethernet. Dit wordt bereikt door software te ontwikkelen die het voor de PC mogelijk maakt te communiceren met de PLC.

3.2.2.4 Opdrachtbeschrijving

Allereerst moet een protocol samengesteld worden om vast te stellen hoe de PC met de PLC zal communiceren. Aan de hand van dat protocol moet er aan de PC en PLC kant software ontwikkeld worden.

De volgende eisen hangen aan dit project:

- De communicatie mag niet te lang duren (Het mag niet langer dan een paar seconden duren voordat alle instellingen van een recept zijn doorgestuurd
- De PC software moet betrouwbaar zijn (Bij de vorige communicatiemethode kwam het nog wel eens voor dat de computer opnieuw opgestart moest worden of dat het programma crashte)
- De PC software mag niet te zwaar zijn (Het moet kunnen draaien op een Windows 7 machine met een 2GHz processor met 2GB geheugen
- De PLC code mag maximaal één functieblok en een paar datablokken gebruiken
- De PLC code moet kunnen draaien op een Siemens CPU 313C met CP 343-1 ethernet module

3.2.2.5 Op te leveren producten

De volgende producten moeten worden afgeleverd:

- Een communicatieprotocol.
- Een module in de PLC die de PLC in staat stelt via ethernet te communiceren.
- Een aanpassing van OMS waardoor deze in plaats van via Profibus DP via ethernet communiceert met de PLC.

3.2.2.6 Projectgrenzen

- OMS zelf hoort niet bij het project, alleen de aanpassing die ervoor zorgt dat deze met de PLC kan communiceren.

3.2.2.7 Randvoorwaarden

Er zijn geen niet-inhoudelijke eisen gesteld aan dit project.

3.2.3 Aanpak

3.2.3.1 Methodes en technieken

V-Model

De project aanpak gaat volgens het V-Model. In de documentatie voor het V-Model staan nog meer stappen om het ontwerp op te splitsen in verschillende componenten en deze dan apart te testen. Deze zijn weggelaten omdat die niet handig zijn om toe te passen in dit project omdat het niet zo groot is.

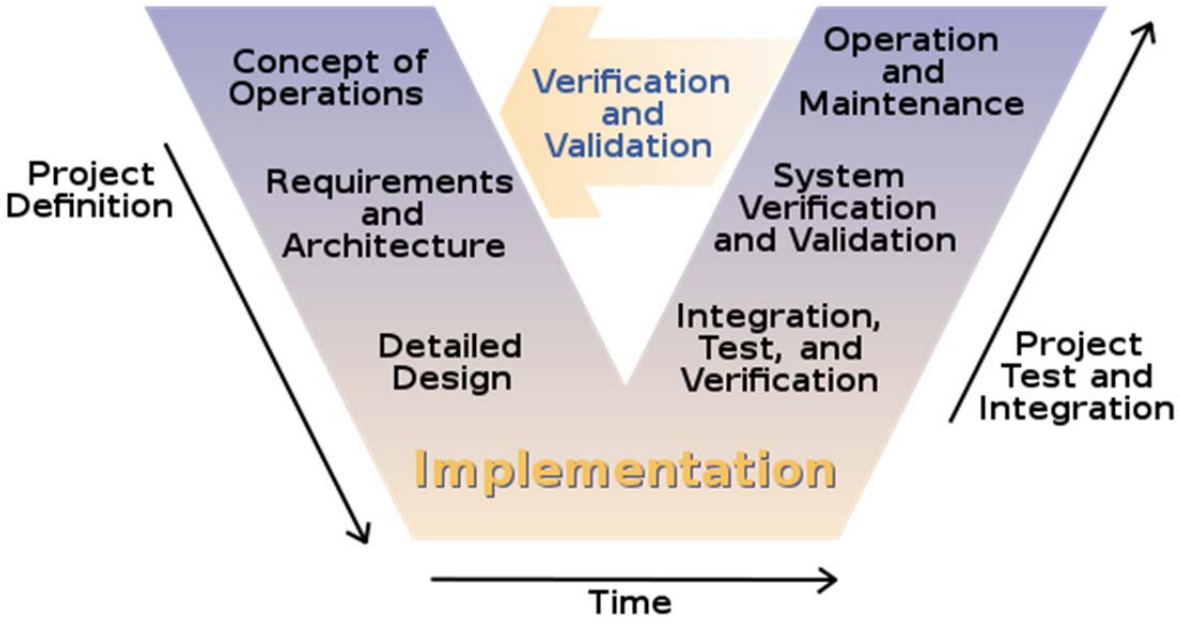


Diagram 3 - V-Model

Dit model bestaat uit een aantal fases[1]:

a. *Concept*
   In deze fase worden de verwachtingen van het systeem op papier gezet.

b. *Eisen*
   In deze fase wordt het pakket van eisen samengesteld, zowel de functionele als de niet-functionele eisen. In deze fase wordt een systeemspecificatie samengesteld.

c. *Systeem ontwerp*
   Aan de hand van de systeemspecificatie wordt een ontwerp samengesteld. In deze fase wordt het volledige ontwerp van het systeem gemaakt.

d. *Implementatie*
   Aan de hand van het ontwerp wordt het systeem geïmplementeerd.

e. *Systeem test*
   Aan de hand van de systeemspecificatie wordt het systeem getest. Hieruit moet blijken of het systeem werkt zoals beschreven is.

f. *Acceptatie test*
   Dit is een gebruikerstest. Aan de hand van een test door een extern persoon moet gekeken worden hoe het systeem werkt in een testomgeving.

[1] *Bron: http://www.coleyconsulting.co.uk/testtype.htm*

g. *Release test*
   Als het systeem draait bij een klant moet er getest worden hoe het draait door middel van een interview bij de klant.

De reden dat voor deze ontwerpmethode is gekozen komt voort uit de aard van dit project:

- Het systeem heeft vaste eisen.
- Er is een doordacht ontwerp nodig, er kan geen lijst met punten afgebrand worden.
- Het wordt gemaakt door één persoon.
- Het systeem moet goed getest worden.

Van de verschillende project aanpak methodes is het V-Model de aanpak die het beste past bij dit project.

- Scrum werkt beter bij het wegwerken van een grote lijst met kleine punten
- Extreme Programming is geoptimaliseerd voor werken in projectgroepen
- Test Driven Development is handig op het moment dat de klant niet veel vertrouwen heeft in de ontwikkelaar
- Bij de watervalmethode wordt het testen pas gedaan op het laatste moment, wanneer het moeilijker is om structurele wijzigingen door te voeren

3.2.3.2 Werkzaamheden

Fase 1: Concept
Het concept in dit project is het plan van aanpak. Ook worden in deze fase analyses gemaakt van de bestaande systemen.
Fase 2: Eisen
In deze fase wordt het communicatieprotocol opgesteld.
Fase 3: Ontwerp
In deze fase wordt het systeem ontworpen.
Fase 4: Implementatie
In deze fase wordt de software geschreven.
Fase 5: Systeemtest
Een gedetailleerde test van het systeem.
Fase 6: Gebruikerstest
Een test van het volledige systeem door een gebruiker.

3.2.3.3 Standaarden

Gebruikte software voor PLC programmeren

- Siemens Step7 SIMATIC Manager v5.5 sp2.
- Simatic WinCC Flexible 2008 (voor de HMI schermen, alleen indien nodig).

Gebruikte software voor PC programmeren

- Microsoft Visual Studio 2008

- .Net Framework v4
- C# .Net als programmeertaal
- Windows Presentation Foundation als GUI framework
- SQL Compact Edition als database
- Fluent Nhibernate als database framework

De lay-out van de documenten is:

1) Het afstudeerdocument
2) De bijlage voor het communicatieprotocol

3.2.3.4 Planning

Dit is de planning voor het project:

Week 36	Inleiding in het bedrijf, planning, opzet
Week 37	Inleiding PLC programmeren (Bodycote)
Week 38	Inleiding PLC programmeren (Bodycote)
Week 39	Inleiding PLC programmeren (Bodycote)
Week 40	Opzetten protocol
Week 41	Ontwerp PLC programma
Week 42	Implementatie PLC programma
Week 43	Implementatie PLC programma
Week 44	Test PLC programma
Week 45	Oriëntatie OMS (Barcodescanners aansturen)
Week 46	Oriëntatie OMS (Barcodescanners aansturen)
Week 47	Ontwerp test programma en ontwerp koppeling binnen OMS
Week 48	Ontwerp test programma en ontwerp koppeling binnen OMS
Week 49	Implementatie koppeling binnen OMS
Week 50	Implementatie koppeling binnen OMS
Week 51	Test OMS software

3.2.4 Projectinrichting

3.2.4.1 Projectorganisatie

Het project bestaat uit een opdrachtgever, Kevin Kloosterman, en een opdrachtnemer, Rik Bulthuis.

3.2.4.2 Informatievoorziening

Aan het begin van elke week is er telkens een vergadering waar de voortgang wordt besproken.

3.2.4.3 Faciliteiten

De faciliteiten zijn:

- Een ontwikkel PC

- Een test PLC (CP313C met een CP343-1 Ethernet module)
- De PC software, Visual studio 2010
- De PLC Software, Step7 Simatic Manager

3.2.4.4 Kwaliteitsborging

Aan het einde van dit traject zal het systeem worden getest. Er zijn twee testfases, een systeemtest om fouten uit het systeem te halen en een gebruikerstest om te beoordelen of er goed met het systeem te werken is.

3.2.4.5 Risicofactoren

Er zijn een aantal risico's aan dit project verbonden:

- Het project loopt uit. Om dit te voorkomen wordt elke week vergaderd en de wordt de planning bijgesteld.
- Het project of een deel ervan is technisch onmogelijk. In de concept fase wordt door middel van analyses vastgesteld of wat gewenst is ook mogelijk is. Indien nodig worden de eisen bijgesteld.

3.2.4.6 Kosten en baten

De kosten van dit project betreffen de tijd die is gemoeid met de ontwikkeling. Deze wordt geschat op 300 uur.

De baten van dit project zijn:

- Voor elke afgeleverde machine hoeft er geen OPC kaart met software meer meegeleverd te worden. Er hoeft alleen maar een ethernet module gekocht te worden (Precieze kosten opvragen)
- Er hoeft geen aparte software geïnstalleerd worden (dat scheelt een paar uur x 100 euro)

4 Afstudeerproces

Het afstudeerproces is verdeeld in de volgende onderdelen:

Week 36	Inleiding in het bedrijf, planning, opzet
Week 37	Inleiding PLC programmeren (Bodycote)
Week 38	Inleiding PLC programmeren (Bodycote)
Week 39	Inleiding PLC programmeren (Bodycote)
Week 40	Opzetten protocol
Week 41	Ontwerp PLC programma
Week 42	Implementatie PLC programma
Week 43	Implementatie PLC programma
Week 44	Test PLC programma
Week 45	oriëntatie OMS (Barcodescanners aansturen)
Week 46	oriëntatie OMS (Barcodescanners aansturen)
Week 47	Ontwerp test programma en ontwerp koppeling binnen OMS
Week 48	Ontwerp test programma en ontwerp koppeling binnen OMS
Week 49	Implementatie koppeling binnen OMS
Week 50	Implementatie koppeling binnen OMS
Week 51	Test OMS software

Allereerst is er tijd gespendeerd aan het leren kennen van de PLC software door mee te helpen aan het programmeren van een machine voor een klant, Deze klant heet Bodycote. Daarna is het communicatieprotocol opgezet, waarna de PLC software is ontworpen en geïmplementeerd. Daarna is er een implementatie voor een barcodescanner geschreven om zo meer inzicht te krijgen in het programma OMS. Dan volgt het ontwerp en implementatie van de aanpassing in OMS die zorgt dat deze kan communiceren via ethernet met de PLC, en de verschillende tests die daarbij horen.

4.1 Inleiding PLC programmeren

Om kennis op te doen hoe een PLC programma bij Weighpack werkt is de eerste opdracht geweest om een PLC programma aan te passen voor een machinelijn voor de klant Bodycote. De klant heeft een grote lopende band. Deze lopende band gaat naar een oven. Het doel is om een bepaald aantal bouten per minuut op de lopende band te laten vallen. Diagram 3 geeft schematisch aan hoe de lijn er uitziet.

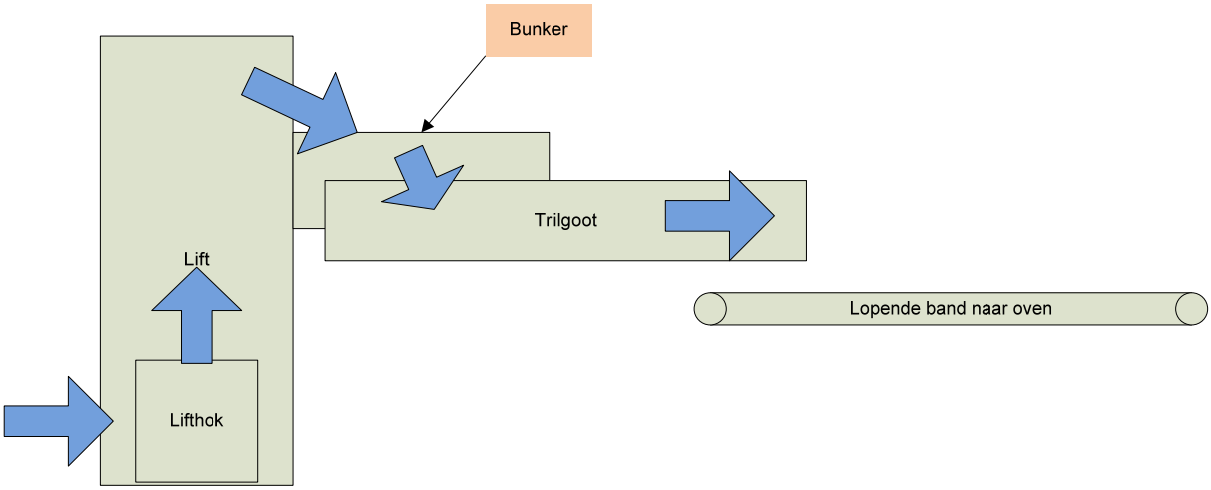


Diagram 4 – Bovenaanzicht project Bodycote

Wat niet goed te zien is in diagram 4 is dat de bunker achter de tril goot zit. In diagram 5 staat daarom het bovenaanzicht.

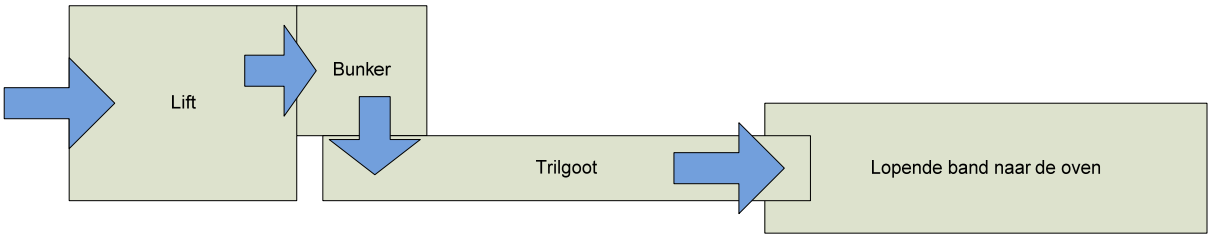


Diagram 5 – Zijaanzicht bodycote

De machinelijn voor Bodycote bestaat uit de volgende onderdelen:

- Een lift om containers met product naar boven te brengen, deze daar te legen en de lege containers weer te laten zakken.
- Bovenaan de lift is een bunker, een opvangbak die het product opvangt en door te trillen het product op de tril goot laat vallen.
- Een tril goot, dit is een goot waar het product uit de bunker in valt en door te trillen het product op de lopende band naar de oven laat lopen. Onder deze tril goot zit een aantal loadcellen. deze loadcellen kunnen het gewicht van de band bepalen. Deze loadcellen zijn aangesloten op een siwarex weegmodule die het gewicht van de loadcellen door kan geven aan de PLC.

Omdat een andere klant van Weighpack bijna dezelfde installatie heeft staan is de PLC code van dat project gekopieerd. Daarna moesten alle input- en output signalen nagelopen worden om te kijken of deze overeenkomen met degene van het gekopieerde project. Om dat te doen wordt er bij Weighpack gebruik gemaakt van een elektro tekening, waarin de input- en outputsignalen staan van de machine. In de PLC zijn alle inputs en outputs van de verschillende functieblokken nagelopen en zijn ook in deze functieblokken alle inputs en outputs gecontroleerd . Dit is in eerste instantie gedaan door te kijken naar de naam van een in- of output en deze te vergelijken met de signalen uit de elektro tekening.

Uiteindelijk was er één schakelaar die extra in deze lijn zat en niet in de lijn van de gekopieerde code en dat was een stopschakelaar, halverwege de lift. Het was lastig om alle inputs en outputs te vinden omdat de benaming in het gekopieerde PLC project niet altijd overeenkwam met het nieuwe project.

De volgende stap was het controleren van de inputs en outputs om te zien of ze goed de PLC binnenkomen. Dit gebeurde door bij elke schakelaar en lichtsensor te kijken of het signaal goed binnenkwam. Bij elke output werd gekeken of de vibrator (de module die de goten laat trillen) of de motor goed werkte.

In het begin waren er wat fouten in de elektrische bedrading waardoor niet alle signalen goed binnenkwamen.

Daarna werd als eerste de lift in zijn geheel getest. In het begin kwam deze niet eens van zijn plaats, en nadat de code werd doorgespit bleek dat een aantal inputs niet aan de goede functieblokken en functies gekoppeld waren. Na een heleboel tests werkte de lift eindelijk naar behoren, alleen was er nog één probleem over: Er miste een stuk functionaliteit.

In diagram 5 staat hoe de lift eerst werkte. Dit is overigens de basis, er zijn nog een aantal failsafes, bijvoorbeeld wat er gebeurt als de bunker vol is.

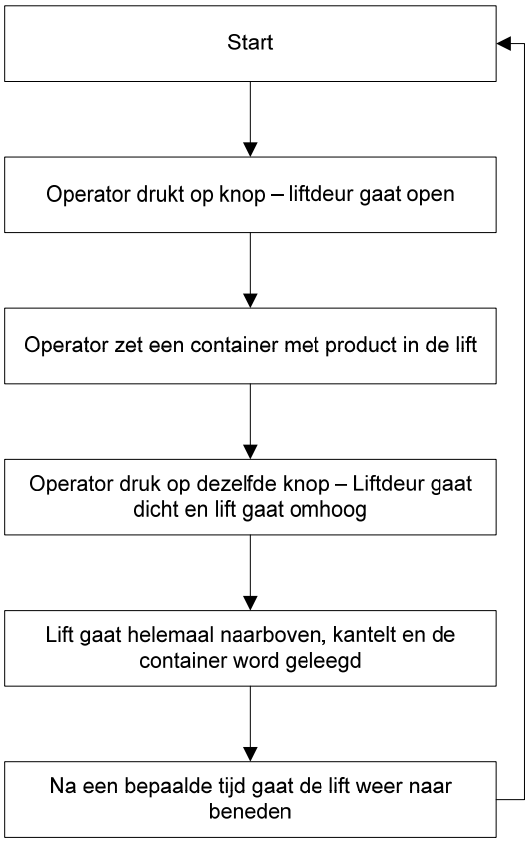


Diagram 6 – Workflow lift

De functionaliteit die ontbrak was dat in het programma voor het HMI scherm een optie aanwezig was om de lift meerdere keren te laten legen en niet in één keer de container leeg te maken. In diagram 6 staat het aangepaste schema.

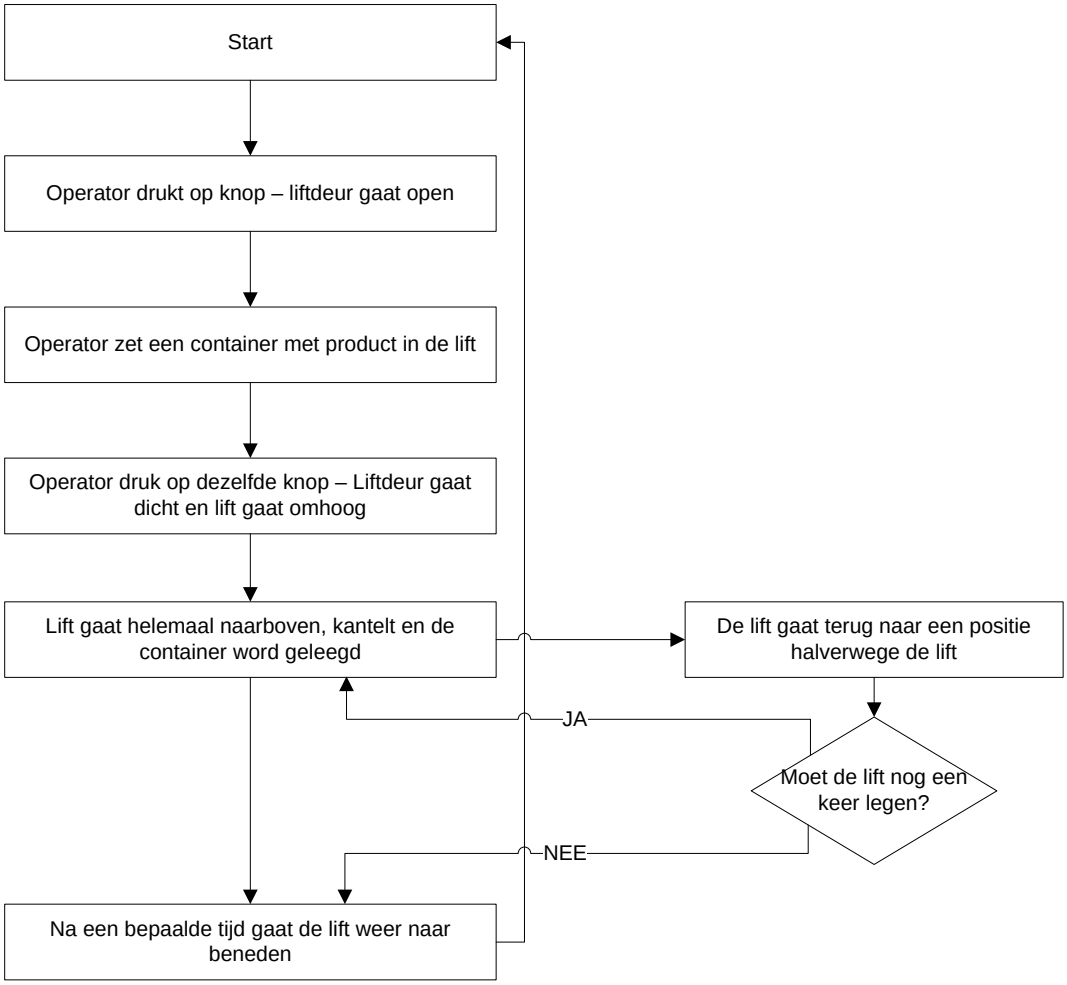


Diagram 7 – Workflow lift (2)

In diagram 7 staan aan de zijkant de nieuwe blokken. De lift gaat na de eerste keer legen terug naar een positie halverwege de lift en controleert of deze niet nog een keer hoeft te legen.

Deze functionaliteit is ingebouwd in de lijn door het stappenprogramma te onderbreken en een variabele aan te maken die gelijk wordt gemaakt aan de instelling voor het aantal keer legen van de lift, elke keer aan het begin van dit stappenprogramma. Bij elke keer legen word deze variabele verminderd met 1, en als deze 0 is mag de lift weer terug naar beneden. Het grootste probleem was de lift stilzetten halverwege. Er zit een schakelaar halverwege die aangeeft of de lift de laatste keer dat deze langs de schakelaar is gekomen naar boven is gegaan of naar beneden. In eerste instantie werd aangenomen dat als de schakelaar op 1 staat de lift op weg is naar boven en bij 0 op weg naar beneden, alleen was het precies andersom. Toen dit omgedraaid werd bleef de lift stilstaan nadat deze geleegd had. Het kostte tijd om uit te vinden wat dit precies veroorzaakt had.

Een andere interessante fout was het per ongeluk tegelijkertijd de lift naar boven en naar beneden laten gaan. Er is een output die er voor zorgt dat de lift naar boven gaat als deze 1 is en een andere output die zorgt dat de lift naar beneden gaat. Als deze tegelijkertijd aangaan gaat de lift eerst een klein stukje naar boven, daarna weer een stukje naar beneden, enzovoorts.

De bunker en de trilgoot werkten precies zoals de machinelijn waar de gekopieerde PLC code vandaan kwam, dus daar hoefde na testen niets aan te veranderd worden. Ook de inputs en outputs

waren correct aangesloten, dat kwam omdat de benaming zo duidelijk was en er niet zoveel inputs en outputs waren.

Een interessante vraag was, hoe het PLC programma er voor zorgt dat er een bepaalde hoeveelheid product per minuut op de band naar de oven komt te liggen. Door de hoeveelheid aan instellingen is het lastig om precies uit te leggen hoe het werkt.

In het kort komt het hier op neer:

Allereerst wordt in het HMI panel een aantal bouten per minuut ingegeven. Nadat de lijn gestart wordt, komt er via de lift product in de bunker. De bunker gaat een tijdje trillen (deze tijd is instelbaar) en laat een grote hoeveelheid op de trilgoot vallen.

De trilgoot meet hoeveel product de bunker zojuist op hem/haar heeft gedumpt. Daarna gaat de trilgoot een tijdje trillen en er valt x kilo product van de trilgoot af op de band die richting de oven gaat. De trilgoot meet nu nog een keer hoeveel product er in de goot ligt en berekent hoeveel er van de goot is gevallen. De hoeveelheid die uit de goot is gevallen is gelijk aan de hoeveelheid die op de band naar de oven is gekomen. Daarna wordt berekend hoeveel product er op de band komt te liggen als de goot daar een minuut lang mee doorgaat. Is dit groter dan wat de instelling (bouten/min) aangeeft, dan word de tijd dat de trilgoot trilt korter gemaakt en anders langer.

Het is een PID-regeling. Er kan bijvoorbeeld ingesteld worden hoe snel de PLC moet reageren op afwijkingen en er kan een startwaarde ingegeven worden.

Daarna is met een ander programma (specifieke software voor de Siwarex module) de Siwarex module gekalibreerd.

Na een uitgebreide test met de leidinggevende erbij werd de machinelijn software in orde bevonden.

4.2 Opzetten communicatie protocol

In bijlage 1 staat het communicatieprotocol tussen PC en PLC.

Bij het opzetten van het communicatieprotocol zijn de Use Cases gebruikt met OMS (de PC software) als actor. Dat wil zeggen dat gekeken is naar welke functionaliteit OMS nodig heeft van de PC. In diagram 6 staat het use case diagram van de PLC.

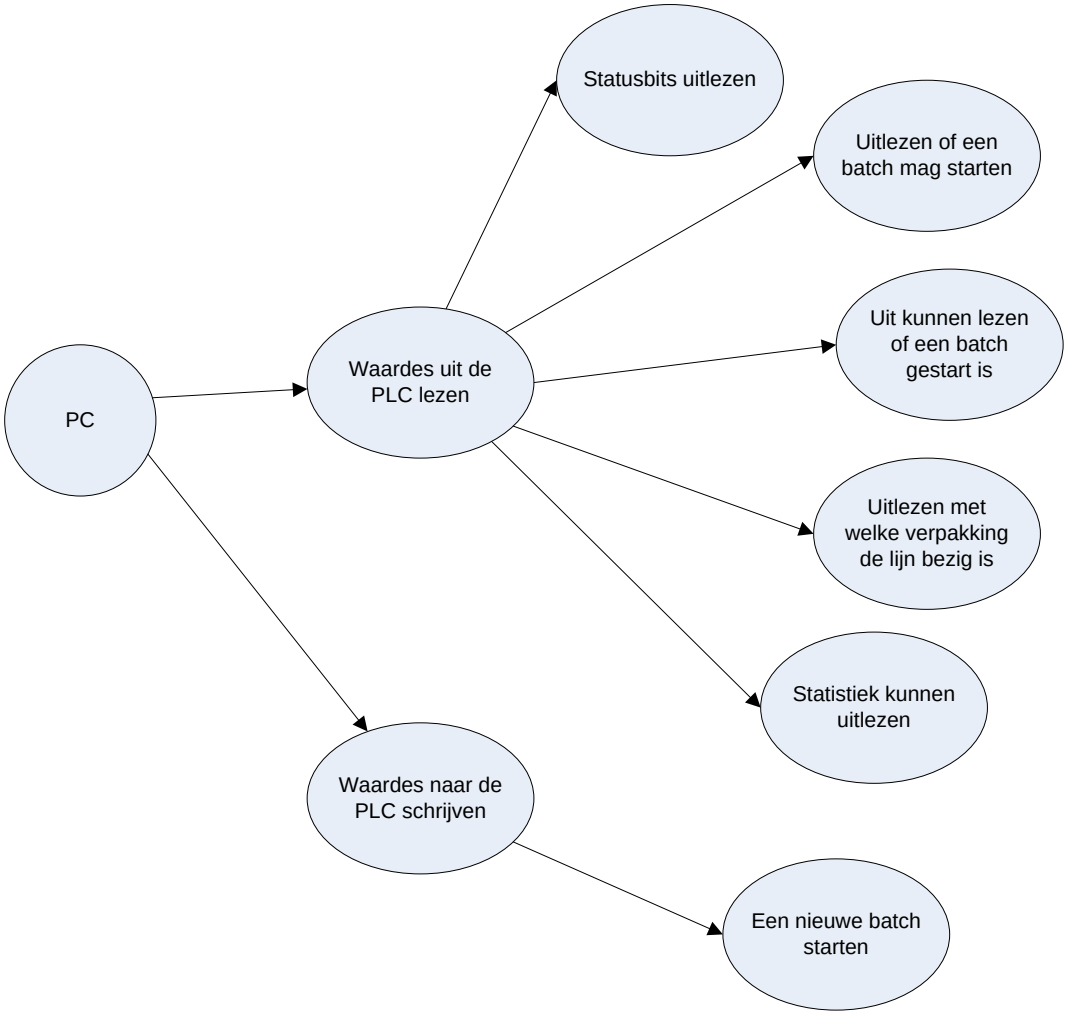


Diagram 8 – Use case diagram PC – PLC communicatie

De basis van de communicatie tussen PC en PLC is natuurlijk het kunnen lezen en schrijven van waardes in de PLC.

De PC moet de statusbits van de PLC uit kunnen lezen. Statusbits zijn bitjes in de PLC die aangeven of de machine is gestart, is gestopt, in storing is of een andere status heeft.

Daarna moet OMS uit kunnen lezen of er een batch (een opdracht) gestart is. Een batch is een opdracht die gestart word door OMS. Deze opdracht is bijvoorbeeld “vul x aantal dozen met het aangegeven product”.

OMS moet een batch kunnen starten of stoppen. Hoe OMS een nieuwe batch (opdracht) stuurt is als volgt. Als eerste controleert OMS of een batch gestart mag worden. Daarna stuurt OMS alle instellingen naar de PLC, waarna de batch gestart word door een bitje hoog te maken.

OMS moet ook uit kunnen lezen met welke verpakking de lijn bezig is.

OMS moet statistiek uit kunnen lezen. De PLC slaat bij een dozenlijn de precieze gewichten op van de laatste 50 gevulde dozen. Deze moet OMS uit kunnen lezen.

Het lastige bij het maken van een use case diagram voor dit systeem is dat alle use cases, afgezien van het starten van een batch, allemaal het lezen van een bepaalde waarde betreft en een keuze gemaakt moet worden tussen wat een relevante actie is en wat gewoon het lezen van een bepaalde waarde is. OMS kan alle waardes uit de PLC lezen, maar niet elke waarde is een use case. Daarom is ervoor gekozen om de waardes die in alle machinelijnen met OMS (uitzonderingen daargelaten) voorkomen te nemen als use case.

4.3 Ontwerp en implementatie PLC software

Allereerst moest gekeken worden naar hoe er precies gecommuniceerd gaat worden met de PLC.

Er zijn drie verschillende communicatiemethodes onderzocht waarmee gecommuniceerd kan worden naar de PLC:

1) UDP Communicatie
   Het is mogelijk om via UDP naar de PLC te communiceren. Het nadeel van UDP is dat er geen garantie is dat de communicatie aankomt bij de PLC. Dit kan handig zijn als het gaat om, in het geval van de opdracht, de statusbits. Alleen alle andere communicatie vereist een gegarandeerde aankomst van de gegevens. Het is alleen mogelijk om de verschillende communicatiemethodes naast elkaar te laten draaien door middel van meerdere ethernet modules aan de PLC te koppelen. Dit is niet gewenst in verband met de kosten voor een ethernet module.
2) TCP Communicatie SEND/RECEIVE
   De SEND/RECEIVE methode is een TCP verbinding met de PLC. Doordat het een tcp verbinding is, is het gegarandeerd dat een pakket aankomt. In diagram 9 staat uitgelegd hoe dit in zijn werk gaat.

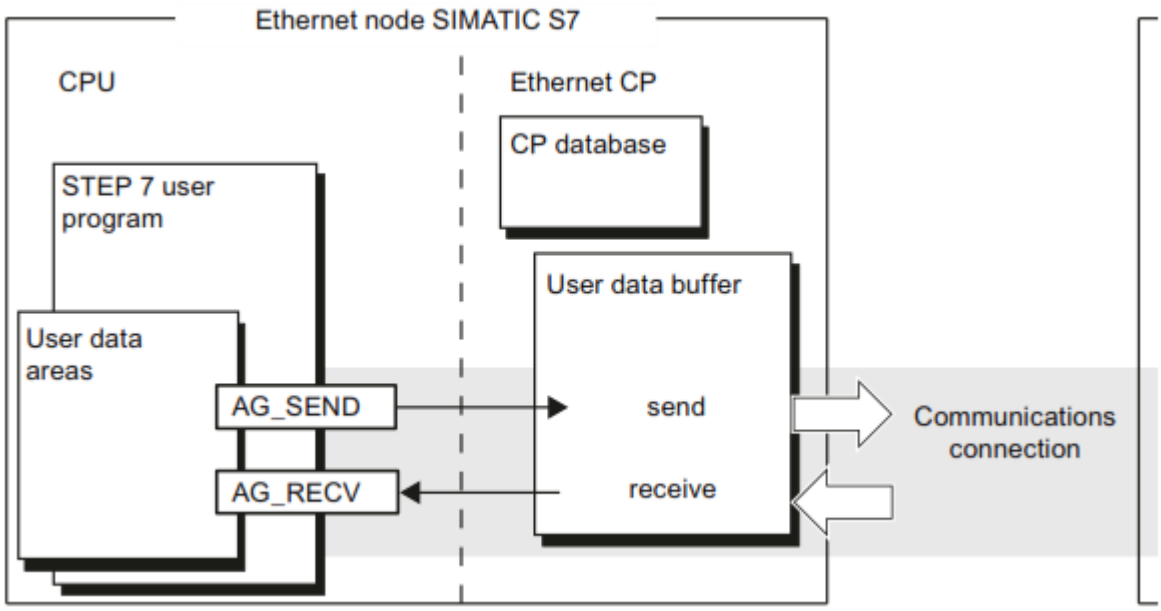


Diagram 9 - Send-receive methode

Het idee is dat als er TCP data naar de PLC verzonden word dit opgevangen word in een functieblok (AG_RECV). Dit blok kan dan de data verwerken en eventueel data

terugsturen via een ander functieblok, AG_SEND. Dit zorgt voor een enorme vrijheid in het omgaan met de TCP communicatie, alleen moet er wel een handshake systeem gemaakt worden omdat niet duidelijk is wanneer de PLC klaar is met het verwerken van binnenkomende data. Als er bijvoorbeeld 5 TCP pakketten binnenkomen, en het eerste pakket geeft een opdracht om iets te doen in de PLC dat een aantal kloktikken duurt, dan kan het voorkomen dat het tweede pakket te snel er achteraan komt.
Daarnaast moet er een protocol verzonnen worden die de TCP data omzet in een DB nummer, een adres daarin en de data die geschreven of gelezen moet worden.

3) TCP Communicatie FETCH/WRITE
   De FETCH/WRITE communicatiemethode is een methode die het mogelijk maakt rechtstreeks waardes naar de datablokken van de PLC te schrijven. Door een TCP pakket volgens een bepaald protocol naar de PLC te sturen kan een bepaald adres, merker of ingang/uitgang gelezen of geschreven worden.
   Dit werkt op dezelfde manier als de SEND/RECEIVE methode, echter met het bijbehorende voordeel dat het protocol waarmee de TCP data word omgezet in een DB nummer, een adres en de data niet meer gemaakt hoeft te worden.
   Deze methode voorkomt niet het probleem met de handshake, het is nog steeds een probleem dat de PLC niet klaar kan zijn met een bepaald pakket voordat het volgende komt.
   Een nadeel van deze methode is dat DB nummers alleen in bytes kunnen worden doorgegeven. Dit limiteert het aantal DB's dat kan worden uitgelezen tot maximaal 255. Omdat de opdrachtgever wil dat bestaande software kan werken met het ethernet functieblok zal hier een oplossing voor gevonden moeten worden.

   Omdat methode 1 niet garandeert dat een pakket aankomt en methode 2 moeilijker te implementeren is dan methode 3 word methode 3 gebruikt.

Use cases

Omdat het communicatieprotocol een use case diagram is, kunnen we deze mooi gebruiken als basis voor het ontwerp voor de PLC software. In diagram 10 staat deze nog een keer weergegeven.

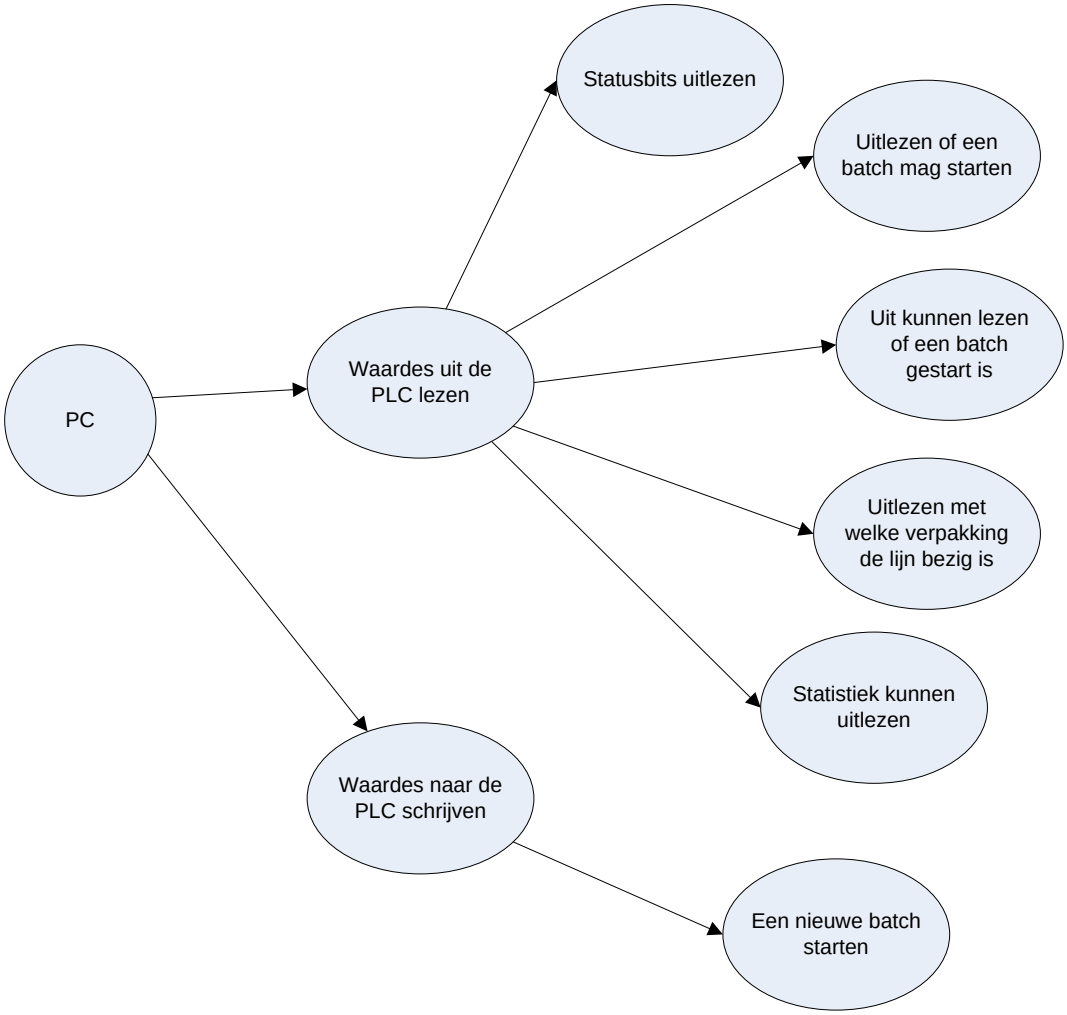


Diagram 10 – Use case diagram PC – PLC communicatie

Allereerst een diepere uitleg van de verschillende use cases.

Statusbits uitlezen

Er zijn een aantal statussen in de PLC die uitgelezen moeten worden:

- Een foutcode
- Een meldingscode
- Staat de machine in automaatbedrijf. Als de machine in automaatbedrijf staat wil dat zeggen dat deze is gestart en bezig is. Dit hoeft niet te betekenen dat de machine ook draait, als er bijvoorbeeld geen product in zit dan detecteert de machine dat en stopt deze.
- Is de machine gestopt. Op het moment dat de machine handmatig gestopt word maakt de machine eerst alle taken af waar deze mee bezig is en dan stopt deze.
- Staat de machine in handmatige mode. In deze methode draait de machine niet, alleen is het via het HMI scherm mogelijk om de verschillende onderdelen aan te sturen zoals bijvoorbeeld een lift naar boven en naar beneden of een cilinder open of dicht te doen.
- Fataal alarm. Op het moment dat er iets serieus mis gaat word dit bit hoog, zoals bijvoorbeeld:

- Een cilinder die er te lang over doet om open of dicht te gaan. Dit kan een indicatie zijn dat er wat tussen zit zoals een arm.
- Een noodstop knop wordt ingedrukt
- Een circuit breaker (stop) word overbelast en stopt ermee

Uitlezen of een batch mag starten

Er zijn een aantal voorwaardes voor het starten van een nieuwe opdracht voor de machine:

- Er mag niet nog een opdracht bezig zijn
- De machine mag niet aan het leegdraaien zijn
- De machine mag geen fataal alarm hebben

Uitlezen met welke verpakking de lijn bezig is

De meeste machinelijnen kunnen verschillende verpakkingen over de lijn hebben lopen, met verschillende afmetingen. Het is belangrijk om te weten met welke verpakking de lijn bezig is, want als er een andere verpakking door de machinelijn moet gaan moet deze wordt omgesteld. Bij dit omstellen verandert de machinelijn bijvoorbeeld de breedte van de lopende banden of hoogte van machines die de verpakking vullen.

Batch statistiek kunnen uitlezen

Aan het einde van een batch moeten er bepaalde gegevens kunnen worden uitgelezen. Dit verschilt per machinelijn en er zijn geen vaste waardes die worden uitgelezen. Een voorbeeld hiervan is dat bij een klant van de opdrachtgever de wens was om aan het einde van een batch het aantal gevulde dozen uit te lezen en door te sturen. Het gaat er om dat er de mogelijkheid is om aan het einde van een batch bepaalde waardes uit de PLC uit te lezen.

Dump statistiek kunnen uitlezen

Dump statistiek is de statistiek van één dump, dit is het vullen van een bepaalde verpakking met product. Dus na dat een doos gevuld is moeten er gegevens in de PLC opgeslagen worden over dit doosje, wat de PC weer uit kan lezen. In principe is dit alleen het netto gewicht van het product dat in de doos gaat.

Een nieuwe batch (opdracht) starten

De pc moet een nieuwe opdracht kunnen sturen. Dit gebeurt door de lijn instellingen naar de PLC te schrijven, waarna de PC een start commando geeft aan de PLC. Op het moment dat er geen nieuwe batch gestart kan worden moet er een negatief antwoord terugkomen.

Sequentie diagram

Met de FETCH/WRITE communicatiemethode wordt alle communicatie geïnitieerd door de PC. Zowel de leesbewerking als de schrijfbewerking bestaan uit een vraag van de PC en een antwoord van de PLC. Een aantal use cases bestaan uit enkel lees of enkel schrijf methodes. Dit zijn:

- *Statusbits uitlezen*
- *Uitlezen of een batch mag starten*

- *Uitlezen met welke verpakking de lijn bezig is*
- *Batch statistiek kunnen uitlezen*
- *Dump statistiek kunnen uitlezen*

Deze use cases zijn niet relevant om weer te geven in het sequentiemodel. Daarom zal alleen een nieuwe batch starten en het lezen of schrijven naar de PLC met een DB nummer hoger dan 255 weergegeven worden in het sequentiediagram. Leesbewerkingen zijn in diagram 11 weergegeven als communicatie van de PC richting de PLC en schrijfbewerkingen vice versa.

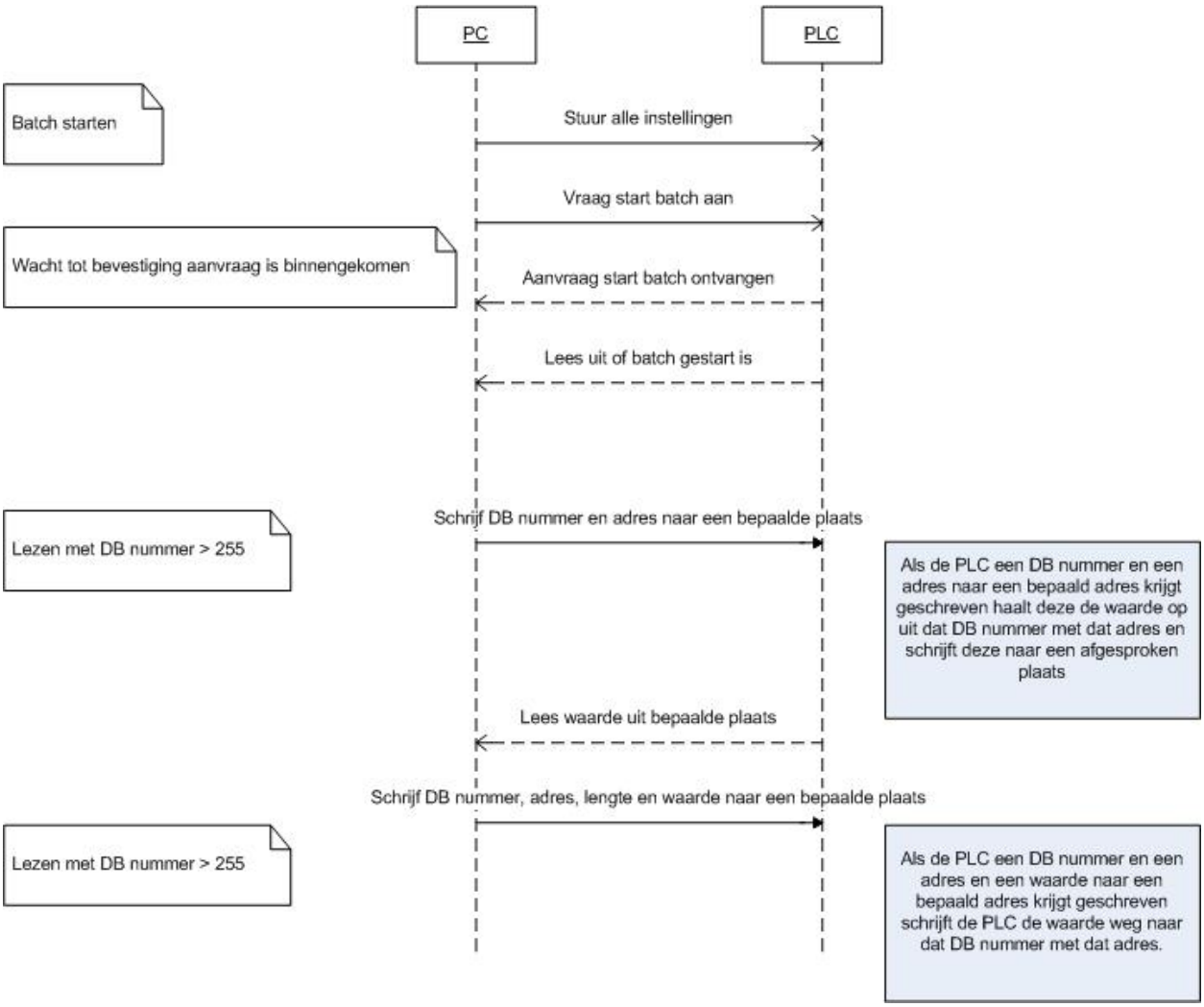


Diagram 11 – Sequentie diagram PC – PLC communicatie

Het starten van een nieuwe batch

Het starten van een nieuwe batch bestaat uit het sturen van alle instellingen naar de PLC, waarna een aanvraag wordt gestuurd om een batch te starten. De PC wacht tot de aanvraag verwerkt is en controleert dan of er een nieuwe batch gestart is. Zo niet, dan mag een nieuwe batch niet starten.

Het lezen van data uit een datablok met een nummer groter dan 255

Het lezen uit een datablok met een waarde groter dan 255 gaat door middel van het schrijven van een DB nummer, een adres uit dit DB nummer en een lengte naar een afgesproken plaats in de PLC. De PLC ziet dit, kopieert de waarde van dit adres naar een plaats waar de PC wel bij kan en de PC kan dit dan ophalen.

Het schrijven van data naar een datablok met een nummer groter dan 255

Het schrijven gaat op dezelfde manier als het lezen waarbij er ook data meegestuurd wordt.

Implementatie

Ontwerp in bestaande PLC applicatie

Omdat dit project een onderdeel wordt van een bestaande PLC applicatie moet gekeken worden hoe dit zodanig ontworpen moet worden zodat bestaande applicaties zo min mogelijk aangepast hoeven te worden om deze functionaliteit, het mogelijk maken van ethernet communicatie, toe te voegen. De beste aanpak hiervoor is een nieuw functieblok maken dat deze functionaliteit in zich heeft.

Het schrijven van gegevens naar de PLC met een DB-nummer hoger dan 255

Door de manier van communiceren met de PLC via de FETCH/WRITE methode kunnen geen waardes geschreven worden naar DB-nummers hoger dan 255. Doordat bestaande PLC applicaties wel werken met deze nummers is hier een oplossing voor verzonnen. Voordat er verder gegaan word met het ontwerp van een oplossing voor dit probleem komt eerst een ander probleem dat meteen meegenomen word.

Het schrijven van een bitwaarde naar de PLC

Tijdens de implementatie is het probleem opgetreden met het schrijven van één bit naar de PLC. Omdat hiervoor het ontwerp aangepast moet worden, wordt hier aandacht aan besteed in het ontwerp. Er kunnen alleen bytes geschreven worden naar de PLC. Om één bit aan te passen in de PLC moet dus een lees en schrijf operatie gedaan worden in de PLC. Een lees operatie om de bits van de hele byte uit te lezen, de juiste bit aan te passen, en dan de hele byte met het aangepaste bit terugsturen naar de PLC. In diagram 12 staat dit omschreven.

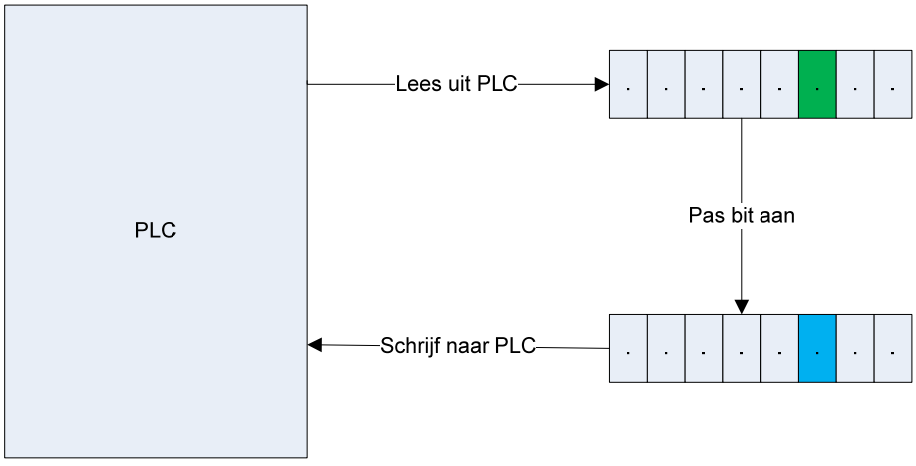


Diagram 12 – Schrijven van bit naar PLC

Omdat de lees en schrijf operaties communicatiepakketten via ethernet zijn, nemen deze relatief veel tijd in beslag. De leesoperatie zou in principe niet nodig hoeven te zijn, de PC weet welk bit deze wil aanpassen en heeft niets nodig met de rest van de bits in de byte.

Om deze twee problemen, het lezen en schrijven naar een DB-nummer groter dan 255 en het schrijven van een enkel bit, op te lossen is een systeem ontwikkeld waarbij de PC alle informatie naar de PLC schrijft die voor de actie nodig is. Er zijn drie verschillende acties:

1) Het lezen van gegevens uit een datablok met het nummer groter dan 255
2) Het schrijven van informatie naar een datablok met het nummer groter dan 255
3) Het schrijven van een enkel bit naar een willekeurig datablok, ook groter dan 255

De aanpak van dit probleem is als volgt. De gegevens die nodig zijn om de actie uit te voeren worden naar een datablok geschreven waar de PC wel bij kan. Daarna verwerkt de PLC deze data. In het geval van actie één, de leesoperatie, zet de PLC de data op een plaats waar de PC wel bij kan en verwerkt deze de data. In het geval van de andere twee acties schrijft de PLC de meegegeven data naar het bit of de DB. In diagram 13 staan schematisch de relevante bytes weergegeven die de PC moet lezen/schrijven om deze operaties uit te voeren.

Diagram 13 – Lees- en schrijfpakket voor DB nummer groter dan 255

DB-Nummer
Dit is het DB-nummer van de gegevens die gelezen of geschreven moeten worden. Er zijn twee bytes voor gereserveerd, zodat DB-nummers hoger dan 255 meegegeven kunnen worden.

Adres
Dit is het adres van de gegevens die gelezen of geschreven worden

Moet er een bit geschreven worden?
In het geval van actie drie, het schrijven van een enkele bitwaarde, geeft deze waarde aan welk bit er geschreven moet worden.

Klaar?
Dit is een bit, en geeft aan of de operatie klaar is.

Schrijven?
Dit bit moet hoog zijn als er een waarde geschreven word. Deze waarde is 0 voor actie 1, het lezen en 1 voor actie 2, het schrijven. Voor actie 3 moet dit bit ook hoog zijn.

Bitwaarde?
In het geval van actie 3, het schrijven van een bitwaarde, geeft dit bit aan welke waarde het bit moet hebben.

Lengte van de gegevens?
In het geval van actie 1 en actie 2 geeft deze byte aan hoeveel bytes er gelezen of geschreven moeten worden. Er kunnen maximaal 4 bytes gelezen of geschreven worden. Er is hiervoor gekozen omdat de enige waardes die langer zijn dan 4 bytes teksten zijn en die bijna nooit worden gelezen of geschreven.

Waardebyte 1 tot en met 4
In het geval van actie 1 komen hier de data te staan die uitgelezen moet worden. In het geval van actie 2 moeten hier de data komen te staan die geschreven moeten worden.

Om dit te verduidelijken is van elke actie een diagram gemaakt met een typisch pakket voor deze actie. In diagram 14 is het pakket te zien dat de PC moet sturen om INT30 uit de DB340 te lezen. Als de PLC deze gegevens krijgt, dan worden Waardebyte 1 en 2 gevuld met de waarde van INT30 uit DB340.

Diagram 14 – Voorbeeld leespakket DB nummer groter dan 255

In diagram 15 is te zien wat de PC moet sturen om de waarde 123 naar INT30 uit DB340 te sturen.

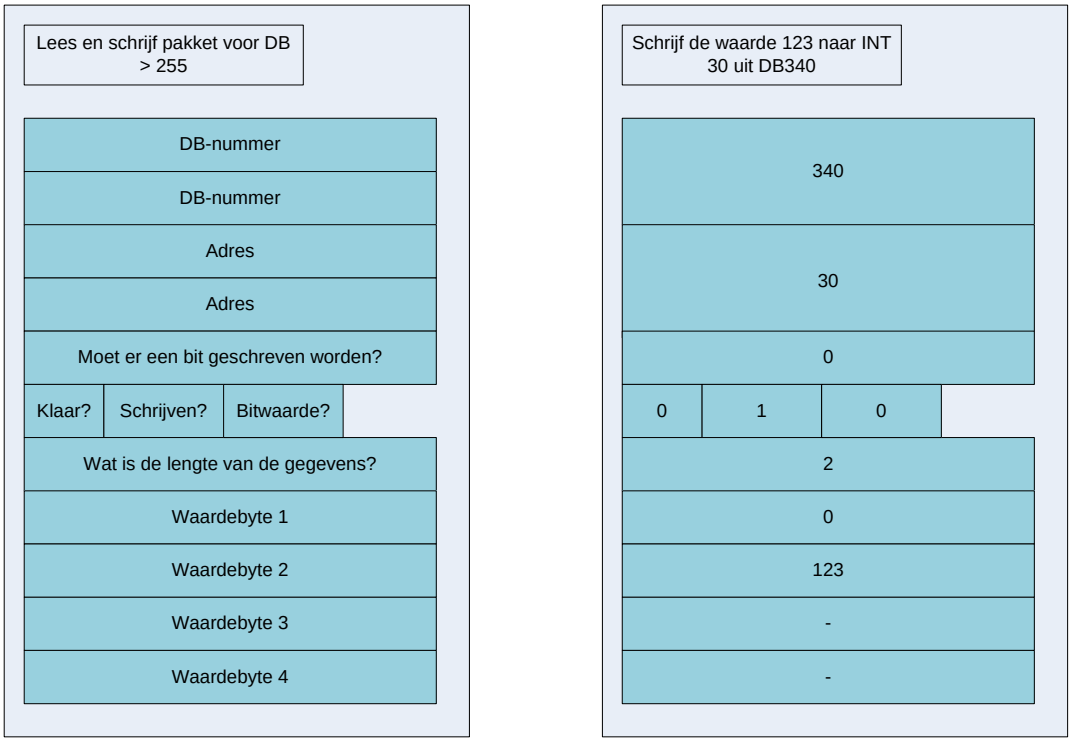


Diagram 15 – Voorbeeld schrijfpakket DB nummer groter dan 255

In diagram 16 is te zien welke gegevens naar de PLC gestuurd moeten worden om bit 5 van INT30 uit DB340 hoog te maken.

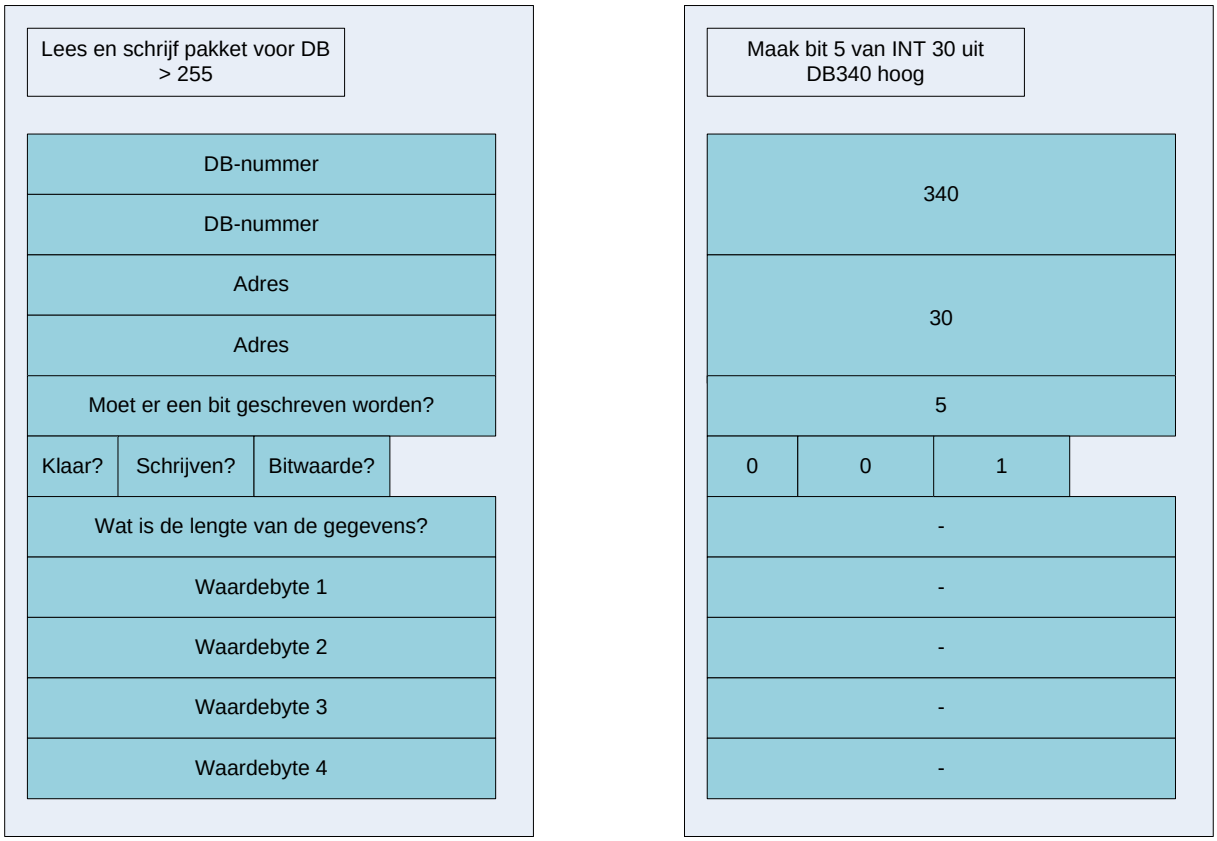
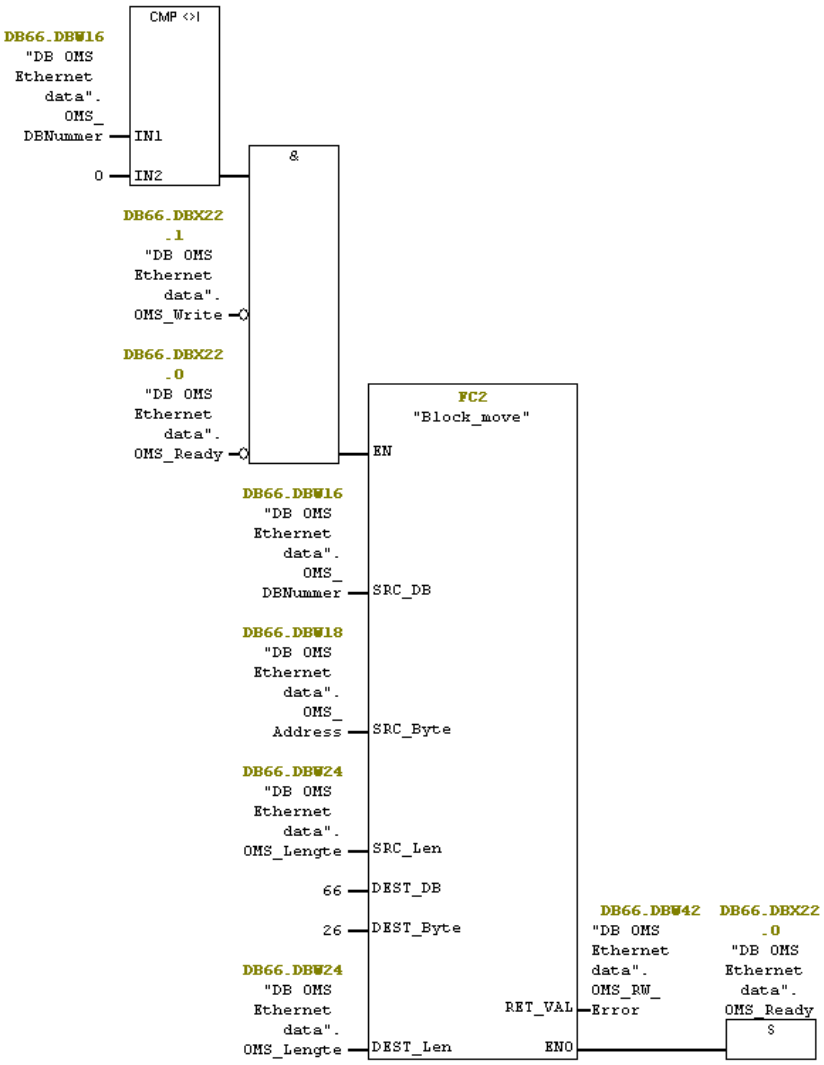


Diagram 16 – Voorbeeld schrijfpakket bitwaarde voor DB nummer groter dan 255

Implementatie actie 1: Het lezen van een waarde uit een DB groter dan 255

In codevoorbeeld 1 staat het PLC blok dat verantwoordelijk is voor het lezen uit een DB groter dan 255.

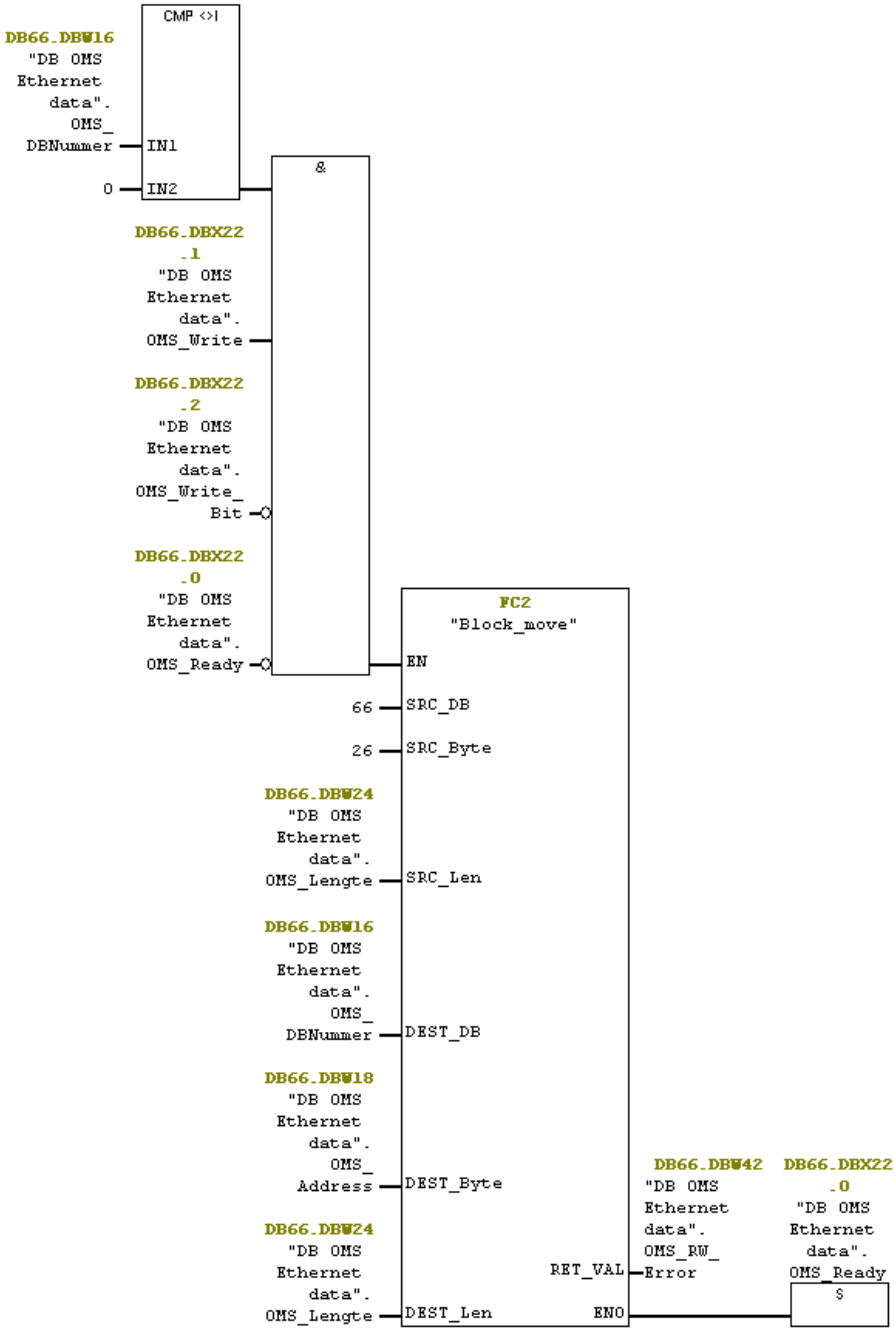


Codevoorbeeld 1 – Lezen uit een DB met DB nummer groter dan 255

Voordat de actie uitgevoerd mag worden moet eerst aan drie voorwaarden voldaan worden. Het DB nummer moet gevuld zijn, het mag geen schrijfoperatie zijn en de operatie mag niet al klaar zijn. Daarna worden er met het blok *Block_move* de gegevens gelezen en geschreven naar de plaats waar de PC ze kan ophalen, in dit geval DB66 byte 26. Tot slot wordt het *OMS_Ready* bit op 1 gezet zodat de PC weet dat deze de gegevens kan lezen en de PLC weet dat de actie niet nog een keer moet worden uitgevoerd.

Implementatie actie 2: Het schrijven van een waarde naar een DB groter dan 255

In codevoorbeeld 2 staat het PLC netwerk dat verantwoordelijk is voor het schrijven naar een DB groter dan 255.



Codevoorbeeld 2 – Schrijven van waarde naar DB met DB nummer groter dan 255

Eerst wordt er gekeken of aan alle voorwaarden voldaan is zoals bij lezen. Nu word ook gekeken of er geen bit geschreven moet worden, want dat is een andere actie. Daarna word met *Block_move* de waarde gelezen die door de PC geschreven is, en deze gekopieerd naar de meegegeven DB. Tot slot wordt het *OMS_Ready* bit op 1 gezet zodat de PC weet dat deze klaar is en de PLC weet dat de actie niet nog een keer moet worden uitgevoerd.

Implementatie actie 3: Het schrijven van een bit van een int op een bepaald adres

Het schrijven van een bit gebeurt door middel van een stappenprogramma van 5 stappen. Het had in één netwerk gekund alleen dan was dit netwerk dusdanig groot geworden dat het niet meer overzichtelijk is. In diagram 17 staat dit stappenprogramma.

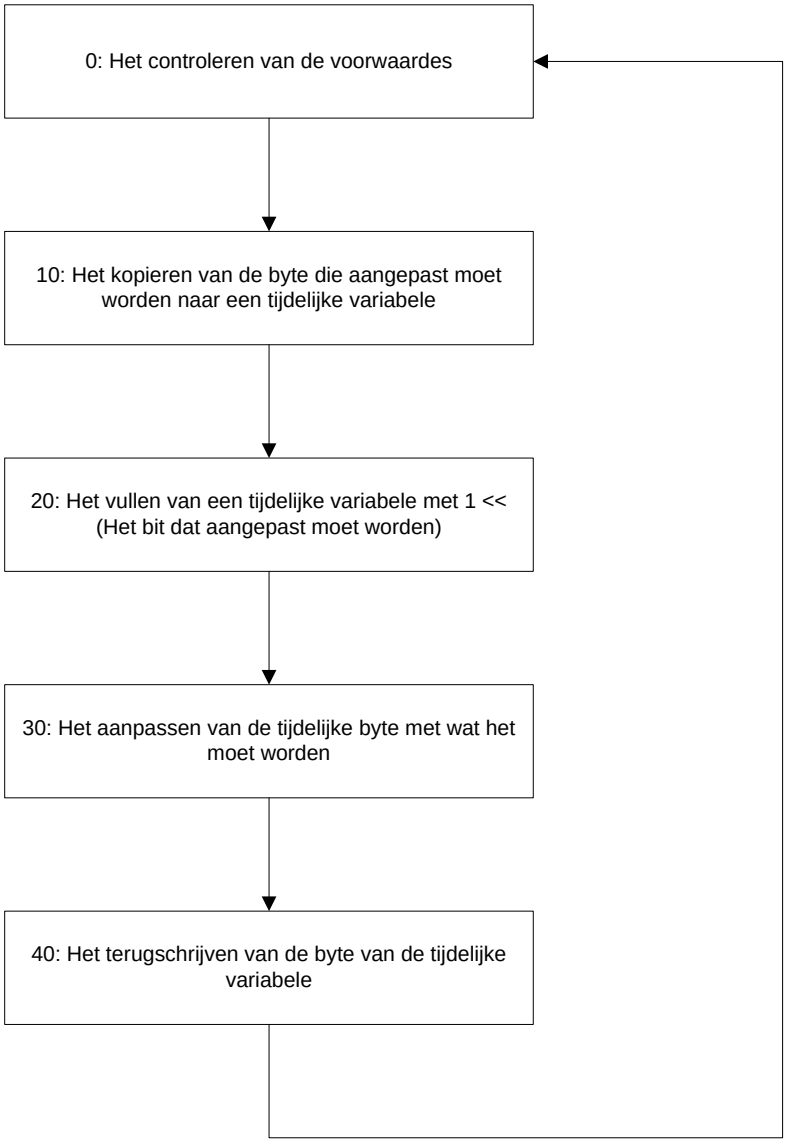


Diagram 17 – Stappenafloop schrijven van een bit naar een bepaald adres in de PLC

Stap 0: Het controleren van de voorwaardes

De voorwaardes voor het beginnen van de actie zijn dat het bit dat geschreven is groter dan 0 moet zijn, het DB-nummer groter dan 0 moet zijn en de actie niet al klaar moet zijn.

Stap 10: Het kopiëren van de byte die aangepast moet worden naar een tijdelijke variabele

De waarde op het adres waarvan één bit aangepast moet worden word gekopieerd naar een tijdelijke variabele.

Stap 20: Het vullen van een tijdelijke variabele met 1 << Bit

Er zijn geen methodes in Siemens die direct een bit aan kunnen passen in een variabele. In plaats daarvan wordt er gebruik gemaakt van logische operaties door middel van *or* en *xor* om het bit aan

te passen, zoals uitgelegd word in stap 30. Om dit te kunnen doen moet een 16 bit waarde gegenereerd worden met allemaal nullen en een één op de plaats van welk bit aangepast moet worden. Dus als het 5e bit aangepast moet worden word de waarde van de tijdelijke variabele 0000100000000000.

Stap 30: Het aanpassen van het bit in de tijdelijke word
In diagram 18 staat het hele programma schematisch uitgelegd.

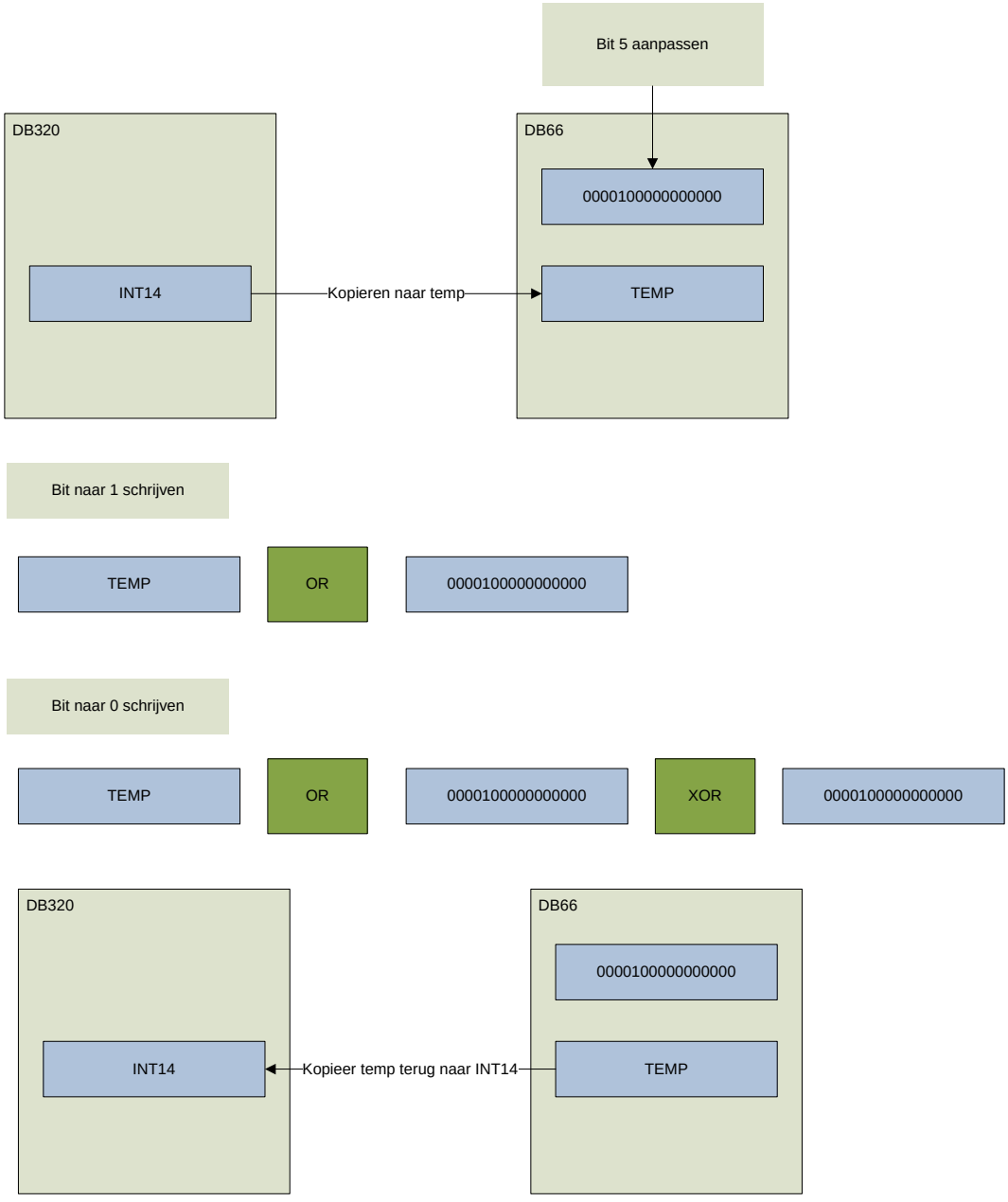


Diagram 18 – Schematische weergave van het schrijven van een bit naar de PLC

In stap 30 word de daadwerkelijke bit aanpassing gedaan. De enige bit operaties die in siemens beschikbaar zijn, zijn AND, OR en XOR. In een tijdelijke variabele staat een waarde met een 1 op de plaats van welk bit aangepast moet worden. Waar ook rekening mee gehouden moet worden is dat het bit een 0 of een 1 moet worden. In het voorbeeld uit diagram 10 word bit 5 van INT14 uit DB320

aangepast. Deze integer word gekopieerd naar een tijdelijke variabele TEMP. Dit is een 16bit waarde. In een andere tijdelijke variabele staat 1 << 5, dit is TEMPBIT.

Als het vijfde bit hoog moet worden gebeurd er een OR operatie met TEMP en TEMPBIT, dit maakt het vijfde bit van TEMP hoog.

In het geval dat temp laag moet worden word er na de OR operatie nog een keer een XOR operatie overheen gehaald, dit maakt het vijfde bit laag.

Stap 40: Het terugschrijven van de tijdelijke INT
In stap 40 word zoals aan het einde van diagram 17 te zien is de TEMP variabele weer teruggeschreven naar de plaats waar die vandaan kwam, in het geval van het voorbeeld DB320, INT14. Ook word het klaar bitje hoog gemaakt en het stappenprogramma weer op 0 gezet.

4.3 Test PLC software

Allereerst is er een test voor de PLC software uitgevoerd, om te controleren of alle functionaliteit van de PLC software goed werkt. De functionaliteit van de PLC software is:

1) Een batch starten
   Het starten van de batch is getest in de PLC door een aanvraag voor de batch hoog te maken en dan te kijken of het bitje dat aangeeft dat de batch gestart is hoog word.
   Uitkomst: geen problemen
2) Batch beëindigen
   Het beëindigen van de batch is getest door het bitje voor de aanvraag van beëindig batch hoog te maken en dan te controleren of het bitje dat aangeeft dat de batch gestart is laag is.
   Uitkomst: geen problemen
3) Een waarde lezen uit een adres met een datablok nummer groter dan 255.
   Er zijn een 16bit waarde en een 32bit waarde opgevraagd uit een adres met een dbnummer groter dan 255.
   Uitkomst: De 32bit waarde faalde in eerste instantie. De oplossing hiervoor was dat dat de lengte die meegegeven word vast op 2 stond in de code, waardoor wel de 16bit waarde goed ging, alleen niet de 32bit waarde. Dit probleem zat ook in het schrijf netwerk
4) Een waarde schrijven naar een adres met een datablok nummer groter dan 255
   Er zijn een 16bit waarde en een 32bit waarde geschreven naar een adres met dbnummer groter dan 255.
   Uitkomst: geen problemen
5) Een enkel bit schrijven van een adres
   Er is een 1 en een 0 geschreven naar een willekeurig adres met het dbnummer boven 255.
   Uitkomst: Het schrijven van een 1 werkte wel, 0 niet. In de implementatie van de PLC software staat dat om de bewerking te doen voor de 1 de OR operatie gebruikt word en voor de 0 de operaties OR en XOR. In eerste instantie was dit alleen de XOR operatie en het leek te werken. Het probleem is dat dit alleen werkt als de waarde van het bit dat veranderd moet worden 1 is, als deze al 0 is dan wijzigt de XOR operatie het bitje juist naar een 1.

4.4 Inleiding OMS

De PC software is een programma genaamd OMS. OMS is software die de functionaliteit van de PLC aanvult. Sommige taken kan de PC software gemakkelijker doen dan de PLC software. Op het moment zijn deze taken:

1. Het opslaan van lijn instellingen
2. Het opslaan en weergeven van statistieken
3. Het aansturen van labelprinters
4. Het aansturen van randapparatuur die niet of ingewikkelder aan te sturen zijn met de PLC dan met de PC

1. *Het opslaan van lijninstellingen*

De instellingen van een machinelijn wordt een recept genoemd. Een recept is een verzameling instellingen voor de machinelijn voor het maken van een bepaald product. In het verleden werden zulke recepten altijd opgeslagen in de PLC zelf. Omdat bij sommige klanten de hoeveelheid recepten te groot werd om in de PLC op te slaan ontstond er de behoefte om deze recepten ergens anders op te slaan en zo ontstond het receptgedeelte van OMS.

Bij het ontwerpen van PLC software worden bepaalde waardes instelbaar gemaakt om verschillende producten van verschillende omgang over de machinelijn te kunnen laten gaan. Als producten bijvoorbeeld zwaarder worden moet de machine meer kracht leveren om deze te verwerken. Deze waardes verschillen per machinelijn. Al deze waardes zijn instelbaar in het PLC scherm, maar kunnen ook doorgestuurd worden vanuit OMS naar de PLC.

2. *Het opslaan en weergeven van statistieken*

Statistieken zijn op dit moment vooral het precieze gewicht van een doos nadat deze gevuld is. De meeste klanten willen precies terug kunnen zien hoeveel er in een doos zit in verband met labels die erop moeten worden gedrukt.

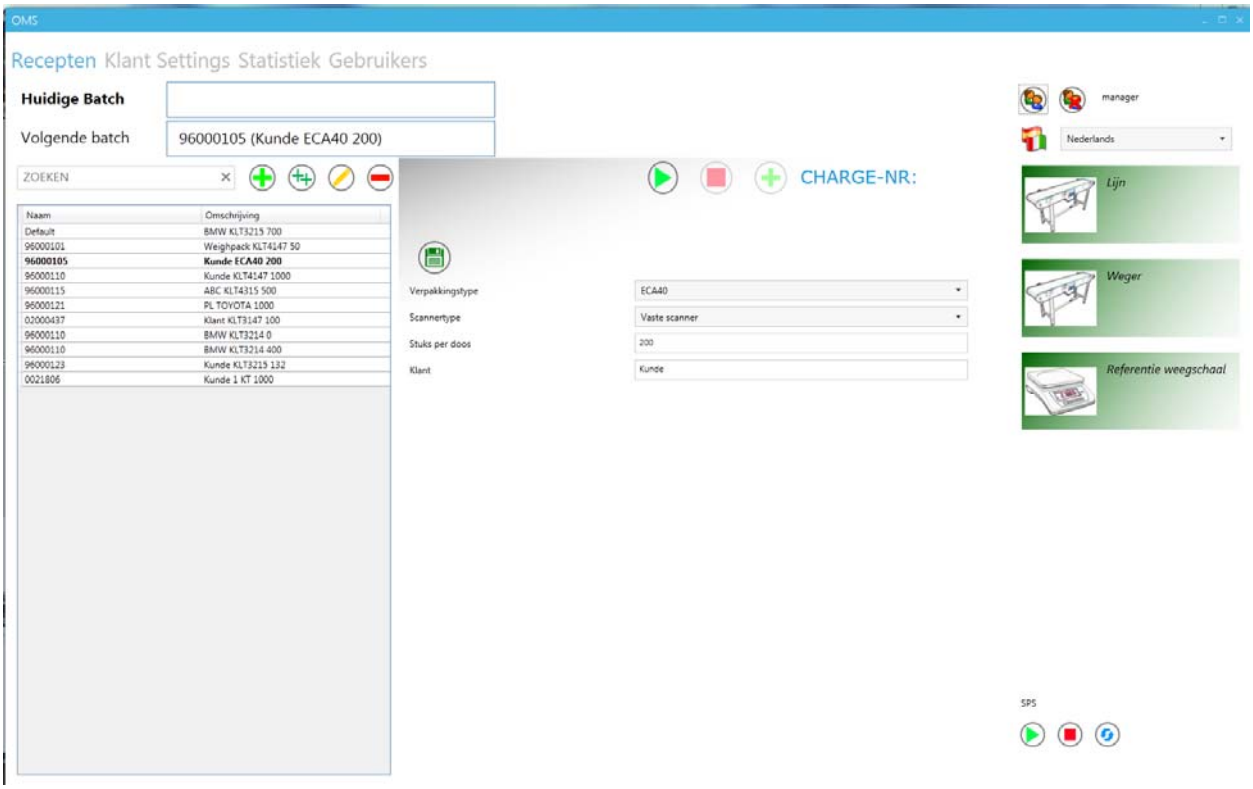
3. *Het aansturen van labelprinters*

OMS heeft de mogelijkheid om een printer aan te sturen die labels afdrukt

4. *Het aansturen van randapparatuur*

Bepaalde randapparatuur wordt niet door de PLC maar door de PC aangestuurd. Het gaat hier bij de meeste machinelijnen alleen om een referentieschaal. Dat is een weegschaal naast de machinelijn waarop de operator (de persoon die de machinelijn aanstuurt) het precieze stuk gewicht van één product af kan wegen. Dit is nodig voor het geval de klant een aantal producten in een doos wil hebben in plaats van een gewicht. Hierbij stuurt OMS het stuk gewicht en het aantal producten per doos naar de machinelijn en de PLC rekent dan uit hoeveel gewicht er in de doos moet komen. Ook andere randapparatuur kan aangesloten worden, maar daar moet aparte software voor geschreven worden.

Werking OMS

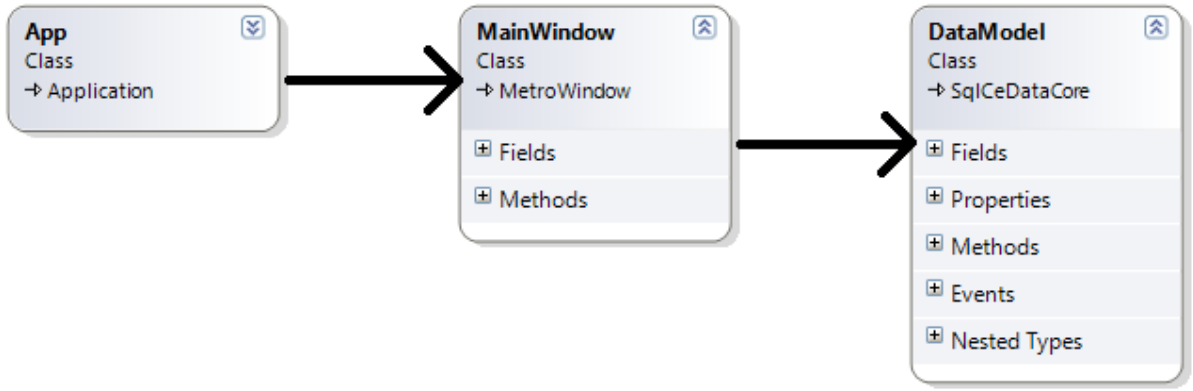


Figuur 1 – hoofdscherm OMS

In figuur 1 is het hoofdscherm van OMS te zien. Links staan de recepten waar uit gekozen kan worden. In het midden staan zogenaamde globale variabelen. Dit zijn algemene instellingen die belangrijk zijn voor het product waarmee gedraaid gaat worden. Aan de rechterkant staat een lijst met machines die aangestuurd worden door OMS. In dit geval zijn dit twee PLC's en een referentie weegschaal. Aan de bovenkant staan wat menu's en is te zien welk recept geselecteerd is (volgende batch) en welk recept aan het draaien is (huidige batch). Onderin zijn drie knoppen te zien die de PLC start, stopt of reset.

Software OMS

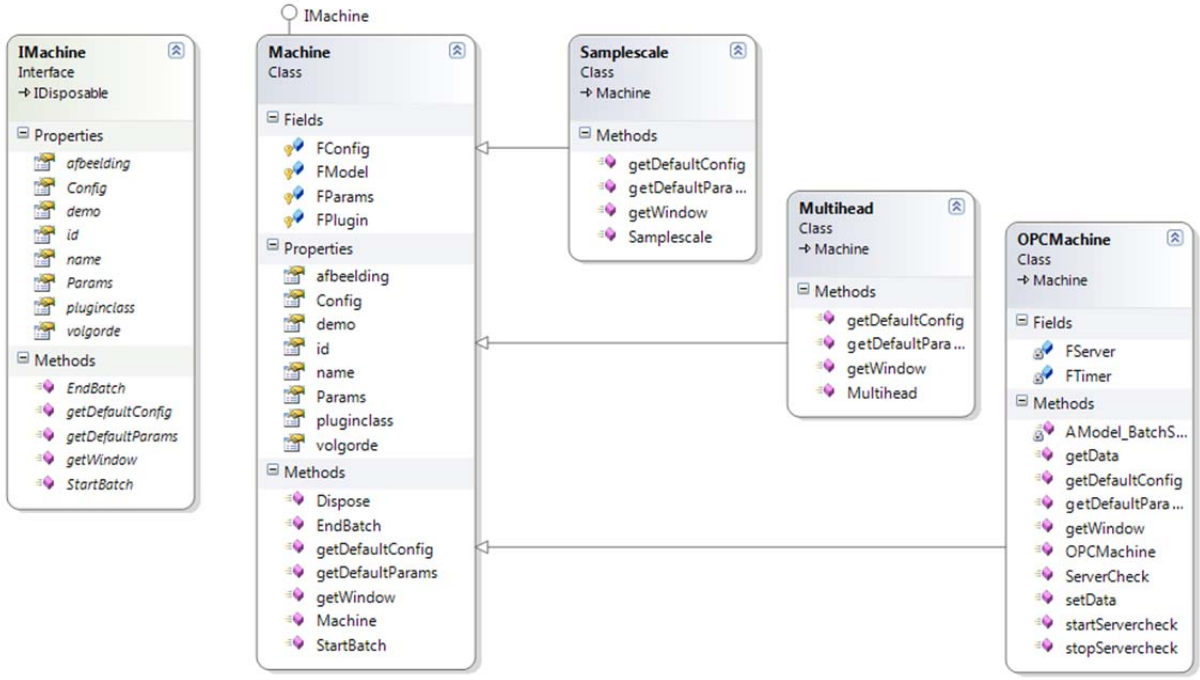
OMS is gemaakt met *Windows Presentation Foundation (WPF)* en .NET met als ontwikkelingstaal c#. In WPF bestaan alle schermen uit een lay-out bestand (gemaakt in XAML, een soort HTML) met een code bestand erachter (de c#). De schermen zijn de basis van de applicatie. Bij de start van de applicatie wordt het hoofdscherm aangemaakt, en dit hoofdscherm roept de andere schermen op. De applicatie heeft één klasse om alle data op te slaan en te verwerken. Dit is *DataModel.* Deze klasse werkt met *Nhibernate*, een database framework. Alle schermen hebben dezelfde instantie van DataModel waarmee ze gegevens uit de database kunnen ophalen of gegevens uit de database kunnen wijzigen. In klassediagram 1 is een klein deel van het klassediagram te zien waarin staat hoe de applicatie, het hoofdscherm en het DataModel gerelateerd zijn.



Klassediagram 1 - MVC

Machines in OMS

Om verschillende externe apparaten aan te kunnen sturen met OMS is er ruimte gemaakt om codes toe te voegen om een dergelijk extern apparaat aan te kunnen sturen. Een extern apparaat, zoals een referentieweegschaal of een PLC wordt een machine genoemd. Een machine is afgeleid van de interface *IMachine*. In klassediagram 2 is een deel van het klassediagram te zien met deze interface en de huidige machines.



Klassediagram 2 – interface IMachine met implementaties

In klassediagram 2 zien we dat de verschillende machines (de referentieweegschaal *samplescale,* een andere weegmachine *multihead* en een PLC *OPCMachine*) afgeleid zijn van *Machine*. *Machine* is op zijn beurt weer afgeleid van *IMachine*. De klasse *Machine* geeft een standaard implementatie van de meeste machines. Machines kunnen ook direct afgeleid worden van *IMachine*, maar de klasse *Machine* heeft wat code in zich die het gemakkelijker maakt een nieuwe machine te implementeren.

IMachine

Om *IMachine* te implementeren moeten een aantal properties en methodes geïmplementeerd worden.

Afbeelding (string)
De afbeelding die bij de machine hoort

Config(Ilist<DConfig>)
Dit is een lijst met configuratie namen en waardes. Elke machine kan een lijst met instellingen meegeven zoals namen van COM poorten of ip adressen. *IList* is een template klasse (dit heet generic in .NET) welke een lijst voorstelt van het meegegeven type. *DConfig* is een configuratie.

Demo(bool)
Als deze waarde waar is loopt de machine in demonstratiemode. Dit is ingebouwd om de machine te kunnen testen zonder dat deze is aangesloten.

Id (int)
Het nummer waaronder deze machine is opgeslagen in de database.

Name (string)
De naam van de machine.

Params (IList<DParam>)
Dit is een lijst met parameter namen en eigenschappen. Elke machine kan een lijst met parameters meegeven zoals namen van PLC instellingen. De waarde van een parameter is altijd afhankelijk van het geselecteerde recept. *IList* is een template klasse (dit heet generic in .NET) welke een lijst voorstelt van het meegegeven type. *DParam* is een parameter.

Pluginclass
De naam van de klasse de geladen word voor deze machine. Het is mogelijk om meerdere machines toe te voegen van hetzelfde type, om bijvoorbeeld meerdere PLC's aan te sluiten. Dit zijn dan meerdere instanties van dezelfde machine met andere eigenschappen.

Volgorde
De volgorde van deze machine in de lijst met machines op het hoofdscherm.

Void EndBatch()
De implementatie van deze methode wordt uitgevoerd als er een opdracht beëindigd word.

IDictionary<string, string> getDefaultConfig()
Deze methode wordt aangeroepen als een nieuwe machine van dit type wordt aangemaakt. Deze methode geeft een lijst terug met instellingen en de standaardwaardes hiervan. *IDictionary* is een interface voor een *Dictionary<type1,type2>.* Een dictionary is een lijst met naam – waarde paren (Key Value pairs). Er moet dus een lijst teruggegeven worden met alle instellingsnamen en hun standaardwaardes.

IList<DParam>getDefaultParams()
Deze methode wordt aangeroepen als een nieuwe machine van dit type wordt aangemaakt. De methode moet een lijst teruggeven met parameters die met de nieuwe machine moeten worden aangemaakt.

Window getWindow()
Deze methode moet een scherm teruggeven dat word geopend als op de knop van deze machine wordt gedrukt in het hoofdscherm.

Void StartBatch()
De implementatie van deze methode wordt uitgevoerd wanneer een nieuwe batch gestart word.

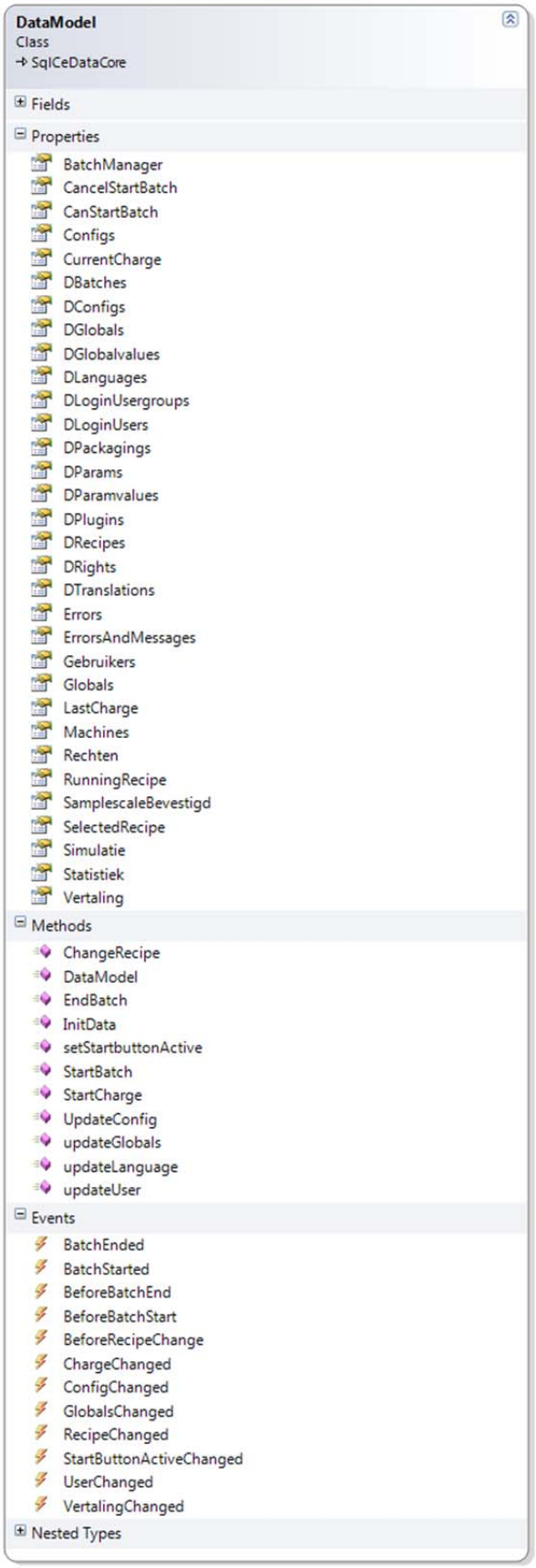
Dispose()
Omdat *IMachine* is afgeleid van *IDisposable* moet deze methode ook geïmplementeerd worden. *IDisposable* is een interface in het .NET framework die één methode heeft, *Dispose*. Deze methode wordt altijd aangeroepen als de instantie weggegooid word.

DataModel

De klasse DataModel is het toegangspunt tot de database en verbind de verschillende klasses met elkaar. Bij het opstarten word er één instantie van deze klasse aangemaakt. Deze instantie wordt aan de verschillende machines meegegeven zodat deze toegang hebben tot de verschillende tabellen in de database. In diagram 3 staat de inhoud van de klasse DataModel.

Database toegang
Voor database toegang gebruikt OMS *Fluent NHibernate. Fluent NHibernate* is een database framework dat op basis van klasses in een bepaalde namespace een database configuratie opzet. Dit gebeurt door een klasse aan te maken met velden die overeen komen met de kolommen van een tabel in de database.

Als voorbeeld is genomen de klasse DConfig die verwijst naar de table DConfigs in de database. In klassediagram 3 is deze klasse te zien.



Klassediagram 3 – Datamodel klasse

```
namespace OMSpro.Model.Entities
{
    public class DConfig : NHRow
    {
        public virtual int pluginid { get; set; }
        public virtual string datatype { get; set; }
        public virtual string name { get; set; }
        public virtual string value { get; set; }
        public virtual bool visible { get; set; }
    }

    public class DConfigMap : ClassMap<DConfig>
    {
        public DConfigMap()
        {
            Table("DConfigs");
            Id(x => x.id).GeneratedBy.Native();
            Map(x => x.pluginid, "fpluginid");
            Map(x => x.datatype, "fdatatype");
            Map(x => x.name, "fname");
            Map(x => x.value, "fvalue");
            Map(x => x.visible, "fvisible");
        }
    }
}
```

Codevoorbeeld 3 – database tabel mapping

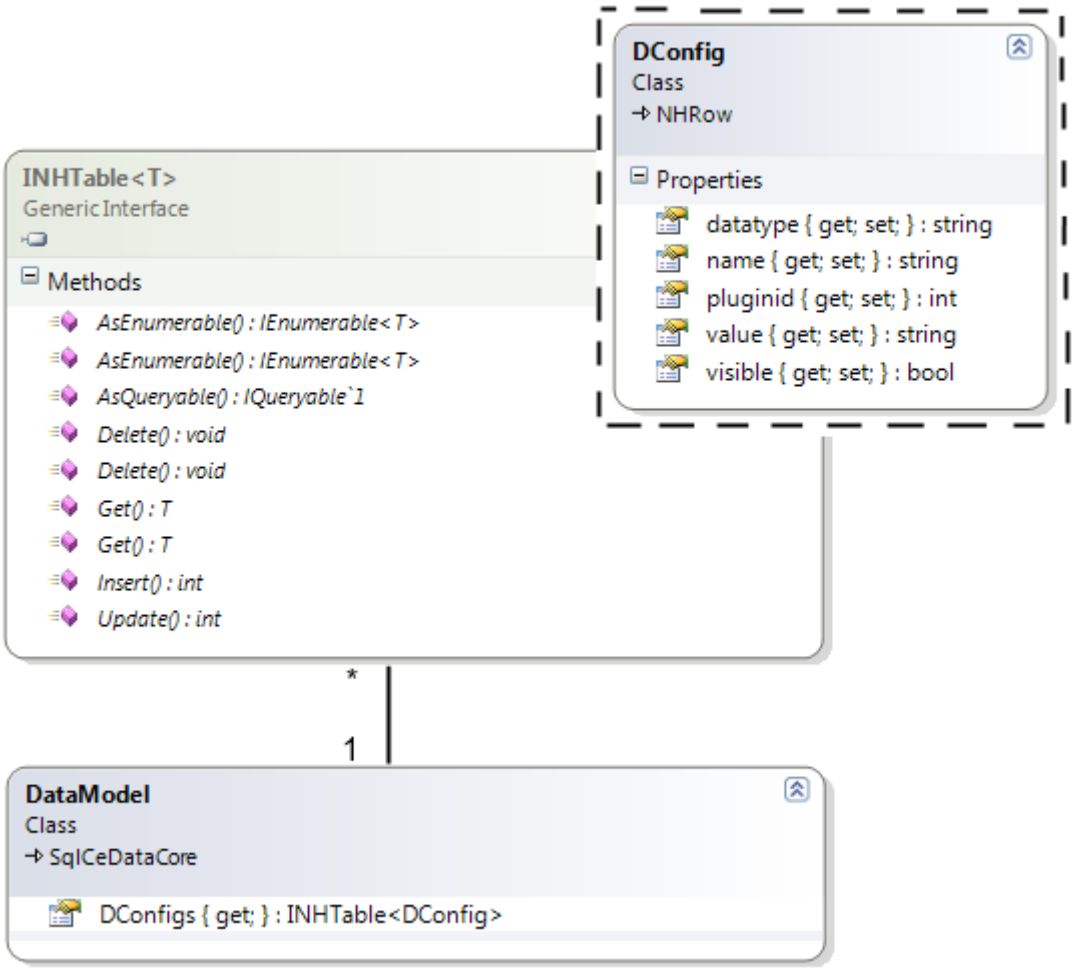
De klasse DConfig stelt een rij in de tabel DConfigs voor met de velden van de tabel als properties. De klasse DConfigMap is gevuld met de configuratie van het framework en geeft door middel van bepaalde methodes aan het framework door hoe deze tabel aan de klasse gekoppeld is:

Table("DConfigs")
Deze methode geeft de naam aan van de tabel in de database

Id(x => x.id).GeneratedBy.Native();
Deze methode geeft aan wat de Primary Key van de tabel is en hoe deze gegenereerd moet worden. In dit geval wordt aangegeven dat de database zelf een sleutel moet genereren.

Map(x => x.pluginid, " fpluginid");
Deze methode geeft aan welk veld van de database aan welke property van de klasse is gekoppeld.

De klasse DataModel heeft voor elke tabel in de database een instantie van het type *INHTable. INHTable* is een interface voor een tabel in de database. In diagram 4 staat als voorbeeld de property DConfigs van DataModel. DConfigs is een INHTable. Omdat INHTable een template is, moet er een type meegegeven worden. Dit is de klasse DConfig, die een rij van de database tabel voorstelt.



Klassediagram 4 – methoden om rijen uit database te trekken

De interface *INHTable* heeft een aantal methodes om rijen uit de database te trekken. In sommige methodes word een template teruggegeven met type T. T is de klasse die meegegeven wordt aan de template *INHTable* en stelt de klasse voor die de rij van de database is. Dus als er bijvoorbeeld een *List<T>* teruggegeven wordt, is dit een lijst van rijen uit de database.

IEnumerable<T> AsEnumerable()
IEnumerable is de meest eenvoudige vorm van een lijst. Deze methode geeft alle rijen uit de database terug.

IEnumerable<T> AsEnumerable(Func<T, bool> APredicate)
Deze methode geeft een lijst met rijen terug die voldoen aan de voorwaarde *APredicate*. *APredicate* is een functiepointer. Deze functiepointer moet een methode zijn waar een T, een rij van de database, in gaat en waar of onwaar uitkomt. In Codevoorbeeld 4 staat een voorbeeld waarin een lijst wordt gemaakt met alle rijen die zichtbaar zijn.

```
void main(DataModel AModel)
{
    IEnumerable<DConfig> lijst = AModel.DConfigs.AsEnumerable(zichtbareRijen);
}

bool zichtbareRijen(DConfig row)
{
    return (row.visible == true);
}
```

Codevoorbeeld 4 – methode om rijen uit database te halen

IQueryable<T> AsQueryable()
Deze methode geeft de volledige tabel terug als een *LINQ Query*.

void Delete(int AId)
Deze methode verwijdert de rij uit de database waar de primaire sleutel gelijk is aan *AId.*

void Delete(Func<T, bool> APredicate)
Deze methode verwijdert alle rijen uit de database die voldoen aan de voorwaarde *APredicate.*

T Get(int AId)
Deze methode geeft de rij terug met de meegegeven primaire sleutel.

T Get(Func<T, bool> APredicate)
Deze methode geeft de eerste rij terug die voldoet aan de voorwaarde *APredicate.*

void Insert(T row)
Deze methode voegt de rij *row* toe aan de database

void Update(T row)
Deze methode update de rij *row* in de database.

Database tabellen
Deze tabellen zitten er in de database van OMS:

1) *DBatches*
   In deze tabel worden gegevens over elke batch opgeslagen. Een batch is een opdracht, van bijvoorbeeld een aantal , dat gevuld moeten worden.
2) *DConfigs*
   In deze tabel zijn alle instellingen van OMS opgeslagen. Ook alle machine instellingen staan in deze tabel

3) *DGlobals*
   In deze tabel staan de verschillende globale parameters. Globale parameters zijn parameters die niet bij een machine horen maar wel ingevoerd moeten worden voor bijvoorbeeld de statistieken.
4) *DGlobalValues*
   In deze tabel staan de waardes van de globale parameters. Deze waardes zijn afhankelijk van de globale parameter en het recept.
5) *DLanguages*
   In deze tabel staan de verschillende talen.
6) *DLoginUsers*
   In deze tabel staan de verschillende gebruikers die kunnen inloggen
7) *DLoginUsergroups*
   In deze tabel staan de verschillende gebruikersgroepen waar gebruikers lid van kunnen zijn. Elke groep heeft een set bijbehorende rechten van dingen die ze mogen aanpassen.
8) *DPackagings*
   In deze tabel staan de verschillende verpakkingsmiddelen zoals verschillende dozen of zakken.
9) *DParams*
   In deze tabel staan de verschillende parameters van de machines. Deze parameters kunnen per machine verschillend zijn. Parameters zijn toegevoegd om machines flexibeler te kunnen instellen. In het geval van bijvoorbeeld een PLC zijn parameters de verschillende instellingen van de machinelijn in de vorm van PLC DataBlok adressen. Elke parameter heeft een adresveld waar een DB nummer en een adres meegegeven kunnen worden.
10) *DParamValues*
    Dit zijn de waardes van de parameters. Deze waardes zijn afhankelijk van het recept.
11) *DPlugin*
    In deze tabel staan de verschillende machines met een machinetype. Zo weet OMS welke instanties van welk machinetype aangemaakt moet worden.
12) *DRecipe*
    In deze tabel staan de naam en nummer van alle recepten.
13) *DRights*
    In deze tabel staan de rechten van bepaalde objecten binnen OMS of van een gebruikersgroep die deze bijvoorbeeld mag aanpassen. Als een gebruikersgroep bijvoorbeeld het recht krijgt om een bepaalde knop te zien, dan wordt een rij aan deze tabel toegevoegd met het nummer van de gebruikersgroep en het nummer van het recht om deze knop te zien.
14) *DTranslation*
    In deze tabel staan alle vertalingen van namen en omschrijvingen binnen OMS.

Barcode Scanner CLV621

Om bekend te worden met het aanmaken van een nieuw machinetype voor OMS is er een SICK CLV621 Barcode Scanner geïmplementeerd.

Pakket van eisen CLV621

- Op het moment dat de barcode een barcode scant welke begint met een “S” moet een nieuwe rij in de statistiek van OMS aangemaakt worden.
- Er mag alleen statistiek geschreven worden als er een opdracht draait.

Ontwerp

In diagram 21 staat het ontwerp voor de barcode scanner. De barcode scanner is via ethernet gekoppeld aan de PC met OMS. De scanner heeft een bepaald IP adres en elke keer dat een barcode binnenkomt wordt er een TCP pakket gestuurd naar de PC (Dit is een instelling van de barcode scanner). Deze TCP pakketten moeten worden onderschept.

Om het systeem overzichtelijk te houden, in overeenkomst met het *Single Responsability Principle,* is het systeem in twee klasses opgesplitst. *CLV621Scanner* regelt de scanner communicatie en *CLV621* regelt de koppeling met OMS.

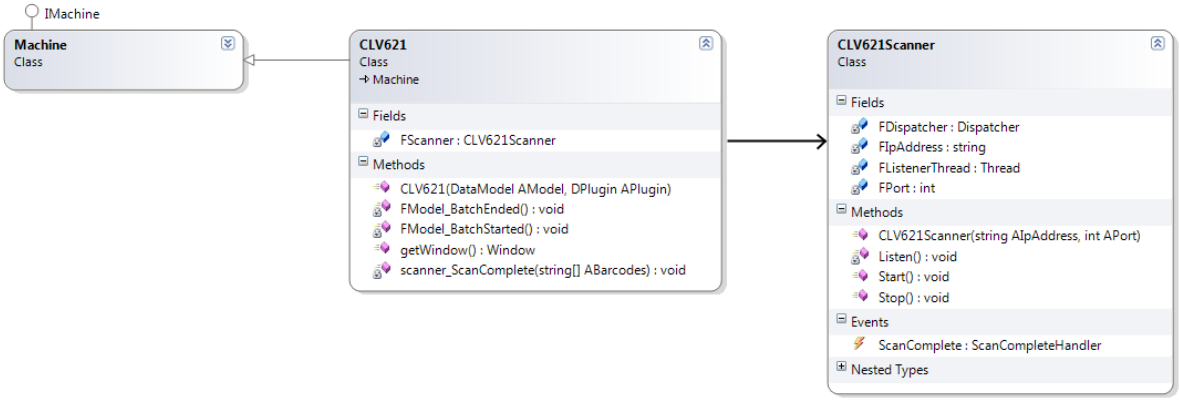


Diagram 20 – implementatie scanner communicatie

De klasse *CLV621Scanner* start een nieuwe thread waarop geluisterd wordt naar binnenkomende pakketten. Als een er een barcode binnenkomt word de *event ScanComplete* gelanceerd. Deze geeft door aan de klasse *CLV621* dat er een barcode binnen is gekomen. *CLV621* regelt dan verder dat er een rij aan de statistiek moet worden toegevoegd.

Implementatie CLV621Scanner

In Codevoorbeeld 5 staat de implementatie van de methode *Listen().* Dit is de kern van de klasse, *Listen()* draait op een aparte thread. De andere methodes zijn er om deze te starten of stoppen.

```
/// <summary>
/// Dit is de thread methode die luistert naar binnenkomend verkeer
/// </summary>
private void Listen()
{
    try
    {
        //Maak een nieuwe TcpClient aan en verbind met het ipadres met het poortnummer
        TcpClient client = new TcpClient();

        client.Connect(FIpAddress, FPort);

        NetworkStream stream = client.GetStream();

        while (true)
        {
            string result = "";

            //als er binnenkomend verkeer is, lees alle data als tekst uit en stop dit in de string result
            while (stream.DataAvailable)
            {
                int b = stream.ReadByte();

                string s = Encoding.ASCII.GetString(new byte[] { (byte)b });

                result += s;
            }

            if (result != "")
            {
                //splits de string result als er meerdere barcodes zijn binnengekomen
                IEnumerable<string> stukjes = result.Split(' ', '\r', '\n').Where(p => !string.IsNullOrWhiteSpace(p));

                //er komt een heleboel data via tcp binnen, en we hebben alleen het 5e stukje van elke 10 stukken nodig. dat zijn de barcodes.
                string[] barcodes = stukjes.Where((t, i) => i % 10 == 5).ToArray();

                //lanceer de event ScanComplete
                if (ScanComplete != null)
                {
                    FDispatcher.Invoke(ScanComplete, DispatcherPriority.Normal, new object[] { barcodes });
                }
            }

            Thread.Sleep(100);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("CLV621Scanner Listen: {0}", e);
    }
}
```

Codevoorbeeld 5 – implementatie methode “listen”

Listen begint met het aanmaken van een TcpClient voor de TCP verbinding. Nadat er een TCP verbinding is gemaakt met de scanner word de TCP data in een lus uitgelezen.

De gegevens die binnenkomen bij elke barcode zijn 10 regels met gegevens waaronder de barcode. Deze wordt uit de data gehaald en een event word gelanceerd met als parameter de binnengekomen barcodes.

Implementatie CLV621

De klasse CLV621 implementeert *IMachine.* Deze kan dus worden aangemaakt door OMS. De klasse CLV621 maakt een *CLV621Scanner* aan in zijn constructor en als een opdracht gestart wordt start de klasse het peilen voor barcodes. In codevoorbeeld 6 staat de code voor de methode die uitgevoerd wordt als er een barcode binnenkomt.

```
/// <summary>
/// Deze methode word gelanceerd als er barcodes binnenkomen
/// </summary>
/// <param name="ABarcodes">Lijst met barcodes</param>
void scanner_ScanComplete(string[] ABarcodes)
{
    foreach (string barcode in ABarcodes)
    {
        if (barcode != "" && barcode.StartsWith("S"))
        {
            string barcode1 = barcode;
            FModel.Dispatcher.Invoke(
                new Action(() => FModel.Statistiek.Add(barcode1, "")));
        }
    }
}
```

Codevoorbeeld 6 – implementatie scancomplete event

De methode *scanner_ScanComplete* gaat alle binnengekomen barcodes bij langs. Als de barcode met een “S” begint, wordt deze opgeslagen in de statistiek. Het DataModel heeft een klasse statistiek die verder regelt hoe dit wordt opgeslagen.

De *Dispatcher.Invoke* methode zorgt dat de *Statistiek.Add* methode uitgevoerd wordt op de juiste thread.

Bij het maken van deze code is er een bepaald probleem opgetreden door de manier waarop het *foreach* statement werkt. Op de 11e regel staat een vreemde lijn: string barcode1 = barcode; Op het moment dat deze lijn weggelaten zou worden krijgen we een probleem dat in het engels *Access to modified closure* heet.

Access to modified closure

Het probleem heeft alles te maken met de manier waarop C# met functiepointers omgaat.

```
new Action(() => FModel.Statistiek.Add(barcode1, ""))
```

Is een afkorting van

```
void methode()
{
    FModel.Statistiek.Add(barcode1, "");
}
```

Dit worden zogenaamde anonieme methodes genoemd, methodes zonder naam. Zoals de compiler goed opmerkt in het laatste voorbeeld bestaat daar geen variabele barcode1. *Op dezelfde manier waarop normale methodes toegang hebben tot globale variabelen, hebben anonieme methodes*

toegang tot lokale variabelen. Echter zit hier een maar aan. Deze lokale variabelen worden als verwijzing meegegeven aan de anonieme methode. Er kan niet gegarandeerd worden dat tegen de tijd dat de methode uitgevoerd word, de lokale variabele nog dezelfde waarde heeft.

Om dit probleem tegen te gaan wordt voor elke anonieme methode een nieuwe variabele, barcode1 aangemaakt.

4.4 Ontwerp en implementatie PC software

Use case diagram

Het use case diagram van de PC Software is exact hetzelfde als het use case diagram van de PLC met in plaats van de PC als actor de gebruiker als actor. In diagram 22 staat dit use case diagram. Voor verdere uitleg van de verschillende use cases kan bij het use case diagram voor de PLC gekeken worden.

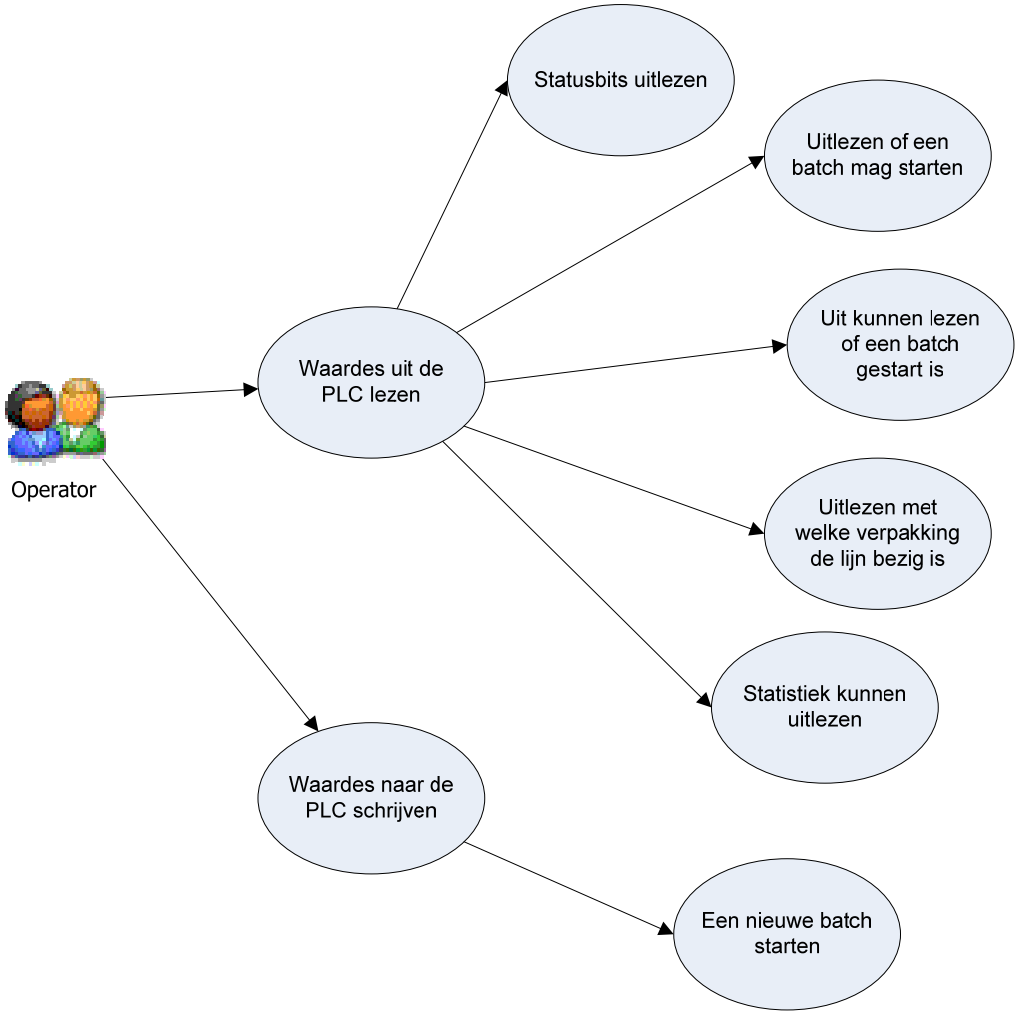
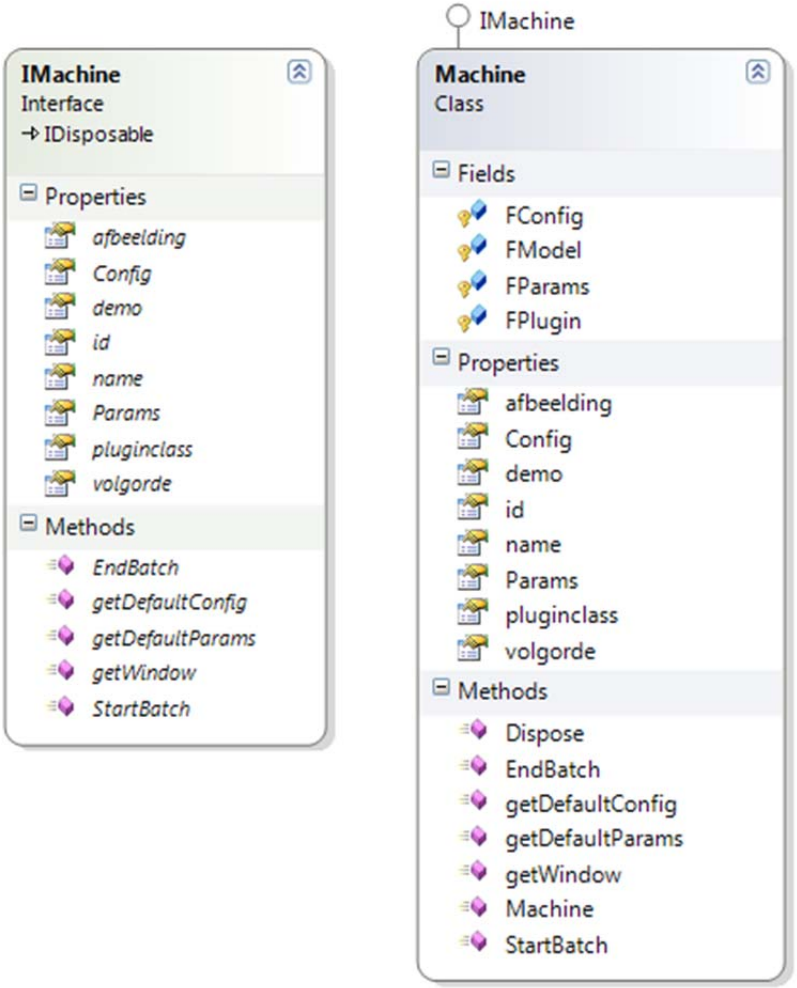


Diagram 21 – Use case diagram PC - PLC communicatie

Ontwerp

Om tot een goed ontwerp te komen moet er eerst gekeken worden naar hoe een machine in OMS geïmplementeerd moet worden. Alle machines in OMS moeten van het type *IMachine* zijn. Voor het gemak is er een standaard implementatie van *IMachine*, de klasse *Machine* gemaakt die een heleboel

functionaliteit in zich heeft die dan niet meer geïmplementeerd hoeft te worden. In klassediagram 5 staan deze.



Klassediagram 5 – standard implementatie IMachine

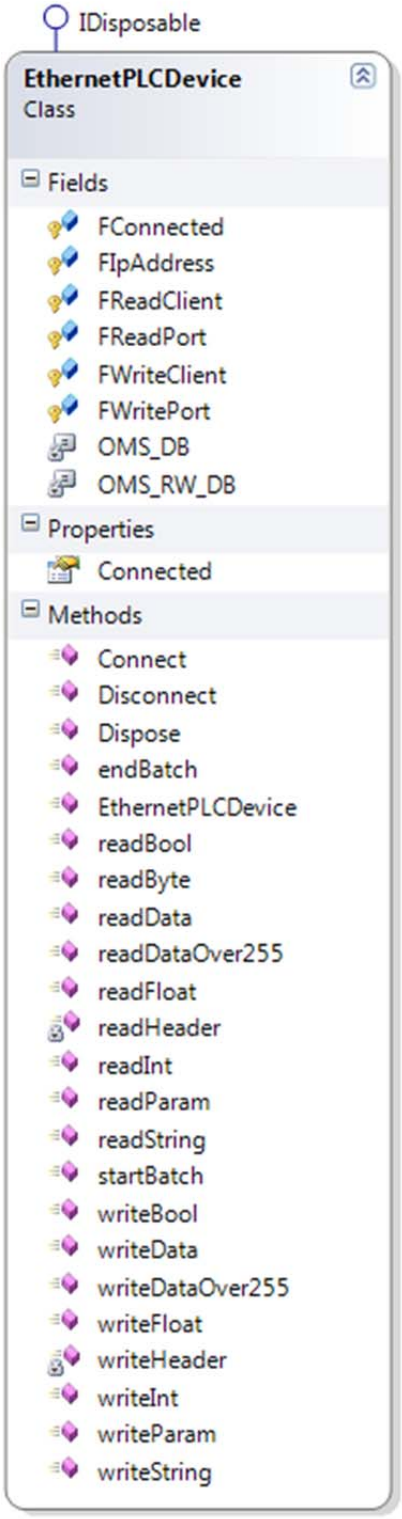
De klasse machine moet overerft worden door een nieuwe klasse met de functionaliteit om te kunnen communiceren met de PLC via ethernet. Omdat niet alle logica in één klasse past en niet in overeenkomst is met het *Single Responsability Principle,* is het handig om de logica te scheiden in verschillende klassen. Er is niet echt een goede verantwoording af te leggen waarom deze scheiding op een bepaalde plaats ligt anders dan de lengte van een klasse. In dit geval is ervoor gekozen om de code op te splitsen in deze drie klassen:

- EthernetPLC, de klasse die *Machine* overerft en de andere klassen aanroept
- EthernetPLCDevice, deze klasse is verantwoordelijk voor de daadwerkelijke communicatie met de PLC.
- EthernetPLCWindow, deze klasse maakt het scherm aan waar de instellingen van het recept aangepast, gelezen en geschreven kunnen worden.

De keuze voor deze splitsing is arbitrair omdat voor hetzelfde geld gekozen had kunnen worden om bijvoorbeeld een aparte lees- en een aparte schrijf klasse naar de PLC had kunnen worden gemaakt,

of een klasse die voor elke use case een methode heeft. De standaard regel die gehandhaafd word is wanneer een klasse onoverzichtelijk word om door te lezen word deze gesplitst.

Voor de uitbreidbaarheid van de logica had gekozen kunnen worden om overal nog een interface toe te voegen, maar dit is niet gedaan omdat er ten eerste geen reden is om te vermoeden dat er ooit nog een andere ethernet PLC machine geïmplementeerd hoeft te worden en ten tweede dat er pas een interface ontworpen moet worden op het moment dat er meerdere implementaties van de ethernet PLC machine zijn omdat het project anders vol staat met interfaces die eigenlijk maar één keer geïmplementeerd worden. Dit maakt de code onoverzichtelijk.



Klassediagram 6 - EthernetPLCDevice

EthernetPLCDevice
Deze klasse is verantwoordelijk voor de daadwerkelijke communicatie. Alle logica om bytes heen en weer te sturen naar de PLC zit in deze klasse. Om te zorgen dat de klasse EthernetPLC daadwerkelijk wat kan doen met deze klasse, moet er voor elk datatype een lees- en schrijfmethode geïmplementeerd worden.

In klassediagram 6 staat het klassediagram voor de klasse EthernetPLCDevice.

Voor degene die niet bekent zijn met de weergave van klasses in Visual Studio:

- Fields zijn private variabelen
- Onder de fields staan twee variabelen met een = teken, dit zijn constantes
- Een property werkt als een publieke variabele aan de buitenkant van de klasse, aan de binnenkant wordt dit behandeld als een set en een get methode. In dit geval is dit alleen *Connected,* een alleen lezen variabele die gelinkt is aan *FConnected* en aangeeft of de klasse verbinding heeft met de PLC of niet.
- De methodes met een slotje zijn private
- Soms staan onderaan nog events, dit zijn functiepointers
- Het bolletje aan de bovenkant geeft aan welke interface deze klasse implementeert.

De klasse implementeert *IDisposable.* Voor deze interface moet je de methode *Dispose* implementeren. Het is de bedoeling dat dispose aangeroepen word op het moment dat het object weg gegooid gaat worden om de klasse de kans te geven om resources die de klasse in gebruik heeft netjes weg te gooien, dat is in dit

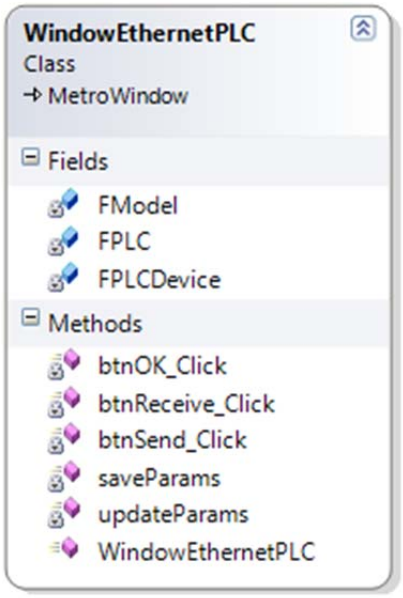
geval de TCP verbinding afsluiten.

TCP verbindingen moeten opgezet en verbroken worden. Daarom is er een *connect* en *disconnect* methode. Er had gekozen kunnen worden om het bestaan van de verbinding te koppelen aan het object, dat als het object aangemaakt wordt dat er dan meteen een verbinding is en als het object weggegooid wordt de verbinding verbroken wordt. Maar dan ontstaan er twee problemen. Ten eerste is het processor intensief, omdat voor elke nieuwe verbinding er een nieuw object aangemaakt moet worden, ten tweede bestaat de kans dan dat er twee EthernetPLCDevices zijn die tegelijkertijd een TCP verbinding met de PLC willen maken, en dit kan niet.

Dan is er voor elk datatype een lees- en schrijfmethode. Ook voor de het starten en stoppen van een batch is er een aparte methode. De laatste twee methodes kunnen ook in de EthernetPLC geïmplementeerd worden als gekeken word naar de verantwoordelijkheid van de klasses.

Daarnaast zijn er nog twee methodes, readParam en writeParam, die het mogelijk maken om receptparameters naar de PLC te kunnen schrijven. Een recept is een verzameling instellingen. Deze instellingen worden in OMS gedefinieerd als een object van het type *DParam*. Zo'n parameter heeft onder andere de eigenschappen adres en datatype en op basis daarvan kan deze direct naar de PLC gestuurd worden.

EthernetPLCWindow
In klassediagram 7 staat het ontwerp voor het scherm dat weergegeven wordt. Omdat er geen ontwerpkeuzes voor deze klasse gemaakt hoeven te worden zal er pas in de implementatie verder aandacht aan besteed worden. Dit komt omdat het scherm een standaard opmaak heeft die gebruikt wordt voor alle machines en deze grotendeels gekopieerd kan worden.



Klassediagram 7 - WindowEthernetPLC

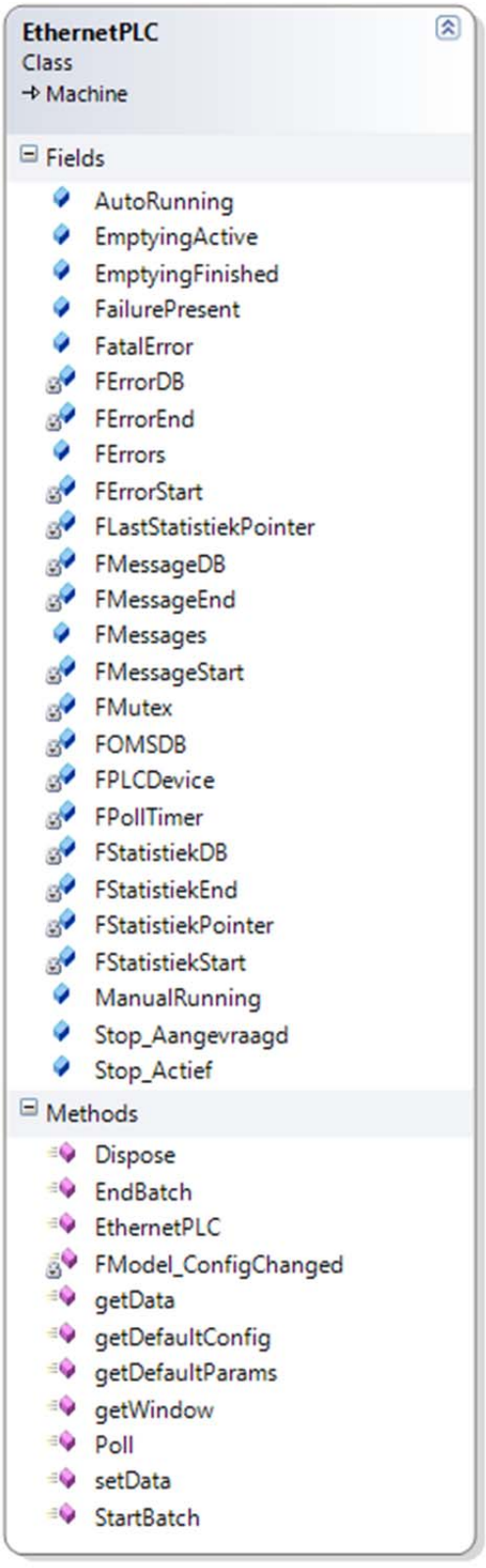
EthernetPLC
De verantwoordelijkheid van deze klasse is het implementeren van de verschillende use cases. Omdat deze klasse is afgeleid van machine moet door middel van het implementeren van deze klasse de use cases worden geïmplementeerd. In klassediagram 8 staat het klassediagram van EthernetPLC.

De klasse machine heeft een aantal methodes die rechtstreeks te koppelen zijn aan use cases.

De methodes startBatch en stopBatch worden gekoppeld aan het starten en stoppen van een batch. De methodes getData en setData halen en sturen de parameters voor een recept naar de PLC.

De andere use cases gaan over gegevens die uit de PLC gehaald moeten worden. Dit moet om de zoveel secondes gebeuren, om OMS op de hoogte te houden van de status van de PLC en de andere use cases. Om dit te doen zal er in de constructor een timer aangemaakt moeten worden die om de zoveel tijd een methode aanroept die al deze data uit de PLC haalt. Deze methode is *Poll.* Poll haalt de gegevens van de andere use cases uit de PLC en geeft deze door aan OMS. Deze gegevens zijn in overeenstemming met de overige use cases:

- De statusbits
- Of een batch mag starten
- Of een batch gestart is
- Met welke verpakking de lijn bezig is
- De statistiek

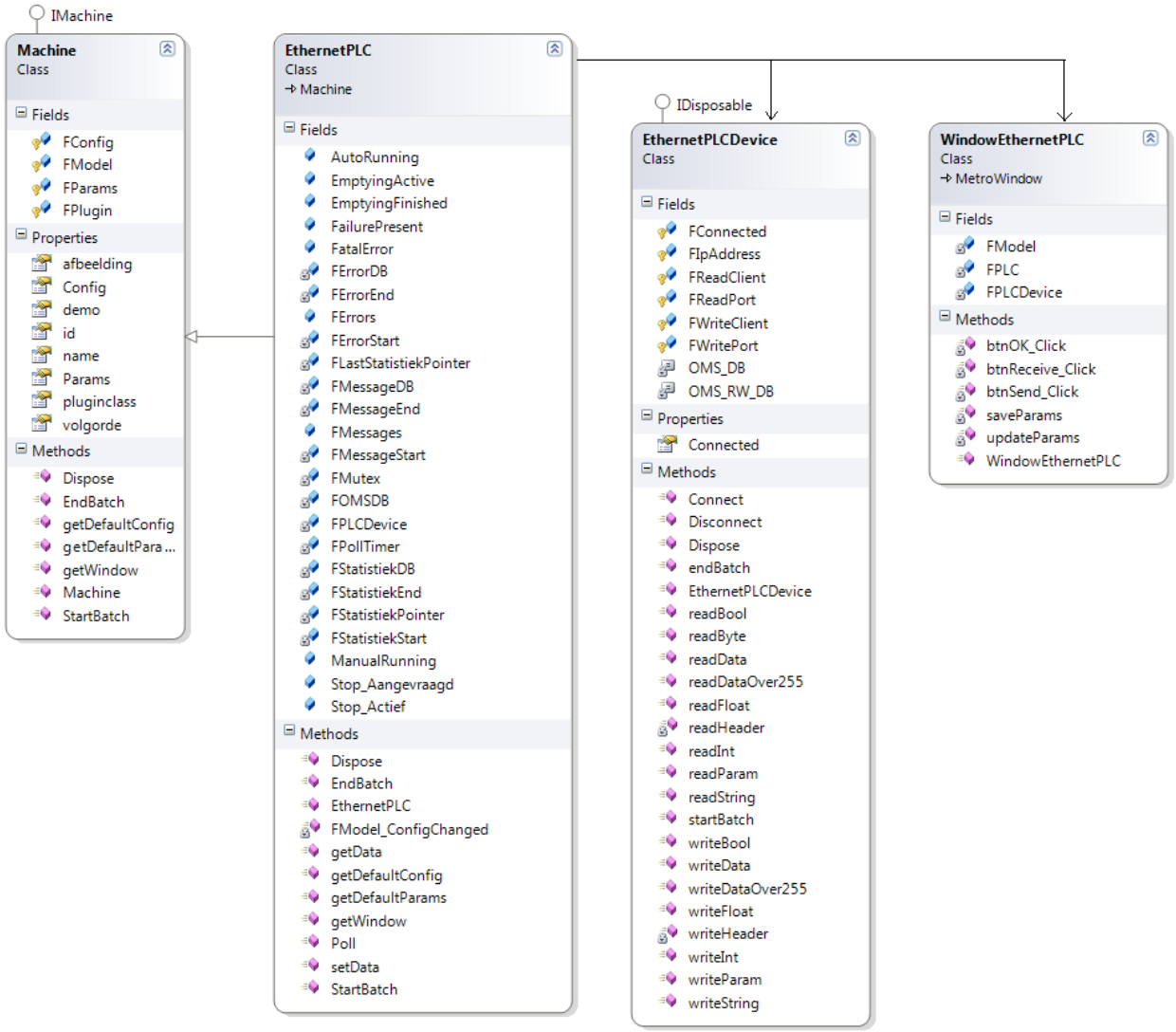


Klassediagram 8 - EthernetPLC

Implementatie

Voor de implementatie van de PC software moet een klasse van het type IMachine geïmplementeerd worden. In klassediagram 8 staat deze klasse, in combinatie met de klasse Machine.

De klasse Machine is een implementatie van IMachine die overerft kan worden door andere klasses en bepaalde standaardfunctionaliteit heeft die het gemakkelijker maakt om nieuwe machinetypes te implementeren. In klassediagram 9 staat het volledige klassediagram van de ethernet PLC software.



Klassediagram 9 – Overzicht klassen EthernetPLC

De klasse EthernetPLC overerft de klasse machine. Deze heeft een instantie van de EthernetPLCDevice die als verantwoordelijkheid de daadwerkelijke communicatie met de PLC heeft, en een instantie van WindowEthernetPLC, dit is het scherm dat weergegeven wordt om de waardes aan te kunnen passen in de PLC.

De klasse EthernetPLC

Als er een machine aangemaakt word in OMS, dan word een instantie van deze klasse aangemaakt.

Een aantal methodes moet worden geïmplementeerd om de gewenste functionaliteit, het communiceren met de PLC, te bereiken.

De constructor (EthernetPLC)
De constructor initialiseert de instantie, roept de constructor aan van de *Machine* en start een thread die om de 10 seconden de Poll methode aanroept om de status en (fout)berichten uit de PLC te halen. Ook word er in de constructor de methode *FModel_ConfigChanged* aan de functiepointer van de klasse *FModel* gekoppeld. Dit zorgt ervoor dat in runtime, iedere keer dat de configuratie wijzigt, deze methode aangeroepen word zodat configuratiegegevens zoals ip adressen en poortnummers gewijzigd worden deze direct worden aangepast, en dat daarvoor niet het programma opnieuw opgestart moet worden.

FModel_ConfigChanged()
Deze methode haalt de configuraties uit de database op zoals het ipadres en het poortnummer van de PLC en het DB nummer van het OMS datablok in de PLC. Deze methode initialiseert ook de instantie van de *EthernetPLCDevice,* verantwoordelijk voor de daadwerkelijke communicatie.

Poll(object state)
Deze methode haalt de statusbits, de errorcodes, berichtcodes en statistiek op uit de PLC. De actie die deze methode doet wordt behandeld als een kritieke sectie, omdat er nog twee andere methodes zijn die ook kunnen communiceren met de PLC en deze methode op een ander thread draait en elke 10 seconden uitgevoerd wordt. Daarom wordt er een Mutex gebruikt die er voor zorgt dat er niet in de kritieke sectie gegaan mag worden als een andere methode deze Mutex heeft.

setData(int ARecipeId)
Deze methode stuurt alle parameters van deze machine, uit het meegegeven recept, naar de PLC. Ook hier wordt gebruik gemaakt van dezelfde Mutex als in de methode *Poll* zodat de methodes niet tegelijk gegevens naar de PLC sturen.

getData(int ARecipeId)
Deze methode haalt de waardes van alle parameters van deze machine uit het meegegeven recept uit de PLC en slaat deze op in het recept. Ook hier wordt gebruik gemaakt van dezelfde Mutex als in de methode *Poll* zodat de methodes niet tegelijk gegevens naar de PLC sturen.

getWindow()
Deze methode geeft een instantie terug van *WindowEthernetPLC*, het scherm dat de parameters weergeeft.

StartBatch()
Deze methode wordt uitgevoerd als een opdracht gestart wordt, en stuurt de start batch opdracht naar de PLC. Ook hier word de Mutex gebruikt.

endBatch()
Deze methode wordt uitgevoerd als een batch beëindigd wordt, en stuurt de beëindig batch opdracht naar de PLC. Ook hier word de Mutex gebruikt.

De klasse EthernetPLCDevice

De klasse EthernetPLCDevice is verantwoordelijk voor de daadwerkelijke communicatie met de PLC. Deze heeft de volgende methodes.

De constructor EthernetPLCDevice(int AIPAdress, int AReadPort, int AWritePort)

Deze methode initialiseert de klasse en slaat het IP-adres en de poorten op zodat de methode *Connect* erbij kan.

Connect()

Deze klasse zet twee verbindingen op met de PLC, een voor lezen en de ander voor het schrijven van data naar de PLC. Als dit niet lukt, geeft de methode *false* terug.

Disconnect()

Deze methode verbreekt de verbindingen met de PLC.

byte[] readHeader(int ADBNummer, int AAdres, int ALengte)

Deze methode maakt de TCP header aan voor het lezen van gegevens uit de PLC. In de documentatie van de FETCH/WRITE methode voor TCP communicatie met de PLC is weergegeven hoe deze header eruit ziet. Dit is schematisch weergegeven in diagram 23.

0	"S"
1	"5"
2	Lengte van de header = 16
3	1
4	3
5	5
6	3
7	8
8	ORG_ID = 1
9	DB Nummer
10	Startadres
11	
12	Lengte
13	
14	255
15	2

Diagram 22 – Fetch TCP-pakket

De methode is statisch omdat deze altijd dezelfde waarde terugstuurt. De lengte die meegegeven wordt is de lengte in *WORDs,* 16bit waardes.

byte[] writeHeader(int ADBNummer, int AAdres, int ALengte)

Deze methode maakt de TCP header aan voor het schrijven van gegevens naar de PLC. In diagram 24 staat dit schematisch weergegeven.

0	“S”
1	“5”
2	Lengte van de header = 16
3	1
4	3
5	3
6	3
7	8
8	ORG_ID = 1
9	DB Nummer
10	Startadres
11	
12	Lengte
13	
14	255
15	2

Diagram 23 – Write TCP-pakket

In de code wordt de lengte gehalveerd omdat in de PC code gerekend wordt met byte arrays en de PLC werkt met WORD arrays, 16bit waardes.

byte[] readData(int ADBNummer, int AAdres, int ALengte)
Deze methode leest *ALengte* bytes uit de PLC uit DBNummer *ADBNummer* beginnend op adres *AAdres.* Dat gebeurt door een header aan te maken met de methode *readHeader* en deze naar de PLC te sturen. Daarna wordt er gewacht tot er TCP gegevens ontvangen worden. In diagram 15 staat de workflow weergegeven voor deze methode.

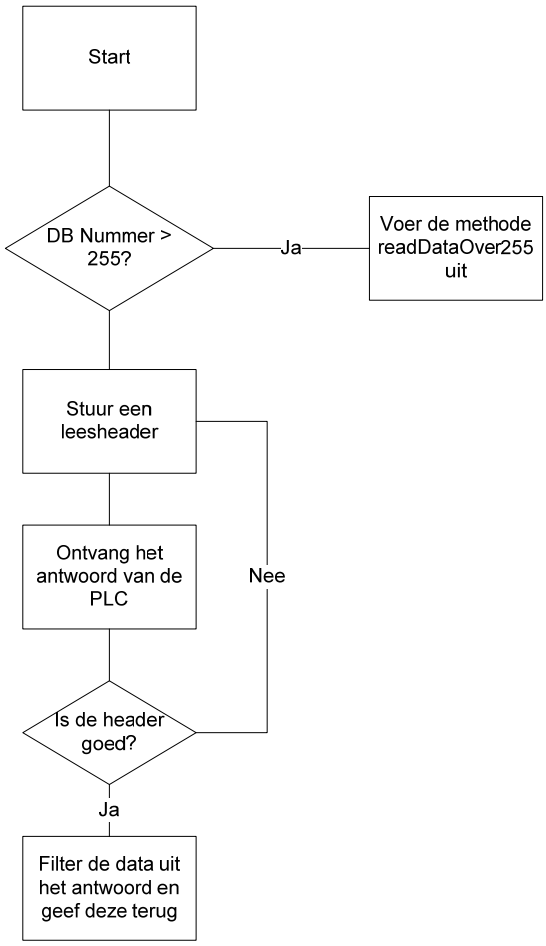


Diagram 24 – Workflow lezen PLC data

Interessant om op te merken is dat de lus die in diagram 25 staat bij de controle of de header wel goed is, eigenlijk niet meer nodig is. In de eerste tests werd er tegen het probleem aangelopen dat er soms hele rare gegevens terugkwamen, die opgelost werden door het meerdere keren te proberen. Het probleem was, dat bij het ontvangen van TCP data de lengte meegegeven moet worden. Omdat de PLC met words werkt en de PC met bytes, werd structureel de verkeerde lengte meegegeven. Het gevolg hiervan was dat er nog gegevens in de TCP buffer stonden, die meegenomen werden met de volgende keer dat de ontvangen TCP gegevens uitgelezen werden. De korte termijn oplossing hiervoor was om te controleren of de ontvangen data wel de goede header hadden , en net zo lang pakketten op te vragen totdat er een goede header terugkwam. Deze fout is opgelost, de lus is erin blijven zitten omdat deze ook gebruikt wordt om fouten in de .NET methode die TCP pakketten zend en ontvangt op te vangen. Als dit gebeurt word één keer geprobeerd om opnieuw verbinding te maken met de PLC, daarna wordt een foutcode teruggegeven.

int writeData(int ADBNummer, int AAdres, byte[] AData)
Deze methode schrijft de data *AData* naar de PLC naar DBNummer *ADBNummer* beginnend op adres *AAdres.* Als dit niet lukt word een errorcode teruggegeven, als dit wel lukt word 0 teruggegeven. Dit gebeurt op precies dezelfde manier als bij *readData* alleen in plaats van de leesheader wordt een schrijfheader met de gegevens gestuurd, en uit de ontvangen data word de eventuele foutcode gefilterd en teruggestuurd.

byte[] readDataOver255(int ADBNummer, int AAdres, int ALengte)
Deze methode handelt het lezen van gegevens af met een DB Nummer hoger dan 255. In de implementatie van de PLC code staat hoe dit in de PLC in zijn werk gaat. In diagram 26 staat de workflow voor de PC kant.

Diagram 25 – Workflow lezen data uit DB met DB nummer groter dan 255

Het DB Nummer, het adres en de lengte worden op een voorgedefinieerde plaats in de PLC geschreven. Daarna wacht de PC tot het klaar-bitje hoog is, waarna de gegevens uit de PLC gelezen worden en teruggegeven worden.

void writeDataOver255(int ADBNummer, int AAdres, byte[] AData)
Deze methode zorgt voor het schrijven van gegevens naar een datablok met nummer hoger dan 255. In diagram 27 staat hoe dit in zijn werk gaat.

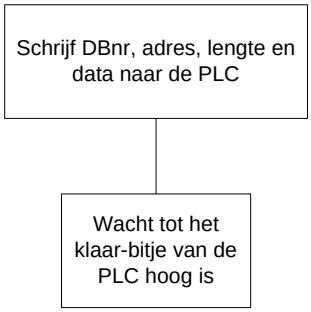


Diagram 26 – Workflow schrijven naar DB met DB nummer groter dan 255

Er hoeft op zich niet gewacht te worden tot de PLC klaar is, alleen als deze methode twee keer achter elkaar uitgevoerd wordt, terwijl er niet gewacht word en de PLC traag is om een bepaalde reden, dan is er een kans dat het mis gaat.

De datatype methodes
Per datatype dat naar de PLC gestuurd kan worden is een lees- en schrijfmethode om de datatypes naar een byte array om te zetten en vice versa voor de leesmethode. Dit is voor de meeste datatypes niet heel interessant, behalve voor *readFloat, writeFloat* en *writeBool.*

float readFloat(int ADBNummer, int AAdres), writeFloat(int ADBNummer, int AAdres, float AValue)
Deze methode leest of schrijft een 32bit single precision floating point uit of naar de PLC. Na uitgebreid onderzoek naar het IEEE 754 single-precision floating point formaat werd uitgevonden dat er een hele simpele .NET methode is die een c# float naar een byte array omzet met het IEEE 754 formaat. Toch is het interessant om weer te geven hoe dit formaat in elkaar zit. In diagram 28 staat weergegeven hoe dit formaat opgeslagen word in zowel PC als PLC.

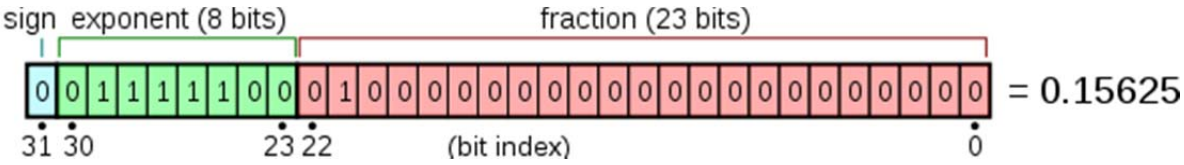


Diagram 27 – IEEE 754 floatingpoint lay-out

Nadat dit probleem verholpen was werkte het nog steeds niet, en dat heeft te maken met *Endianness.* Endianness is de volgorde van bits in een byte. Er zijn twee verschillende methodes, *Little-endian* en *Big-endian.* Big endian is normaal, little endian is van achter naar voren. De PLC gebruikt big-endian, de PC blijkbaar niet.

writeBool(int ADBNummer, int AAdres, bool AValue)
Deze methode schrijft een bit (boolean) naar de PLC. In de implementatie van de PLC staat uitgelegd hoe dit in de PLC in zijn werk gaat, de PLC kant is vrij simpel. In diagram 29 staan de bytes die gestuurd moeten worden naar de PLC om het bitje aan te passen.

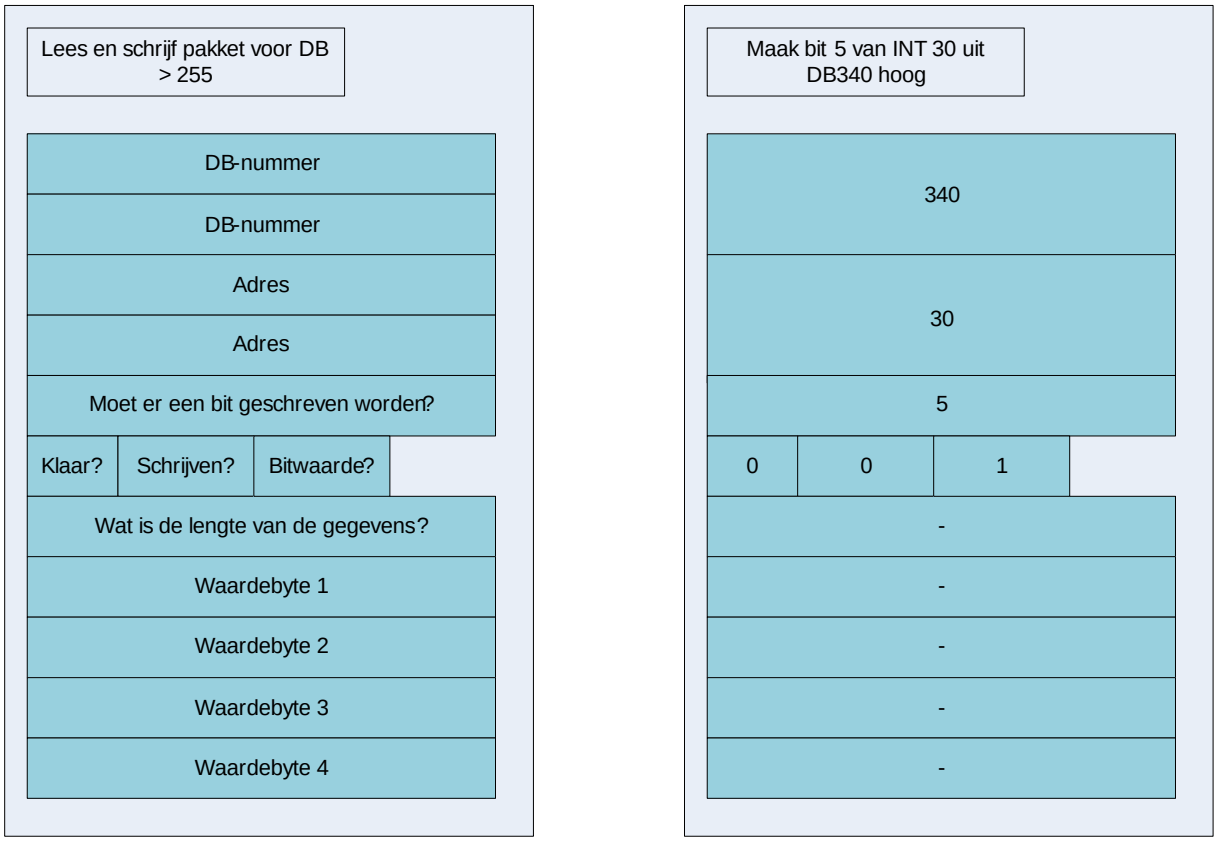
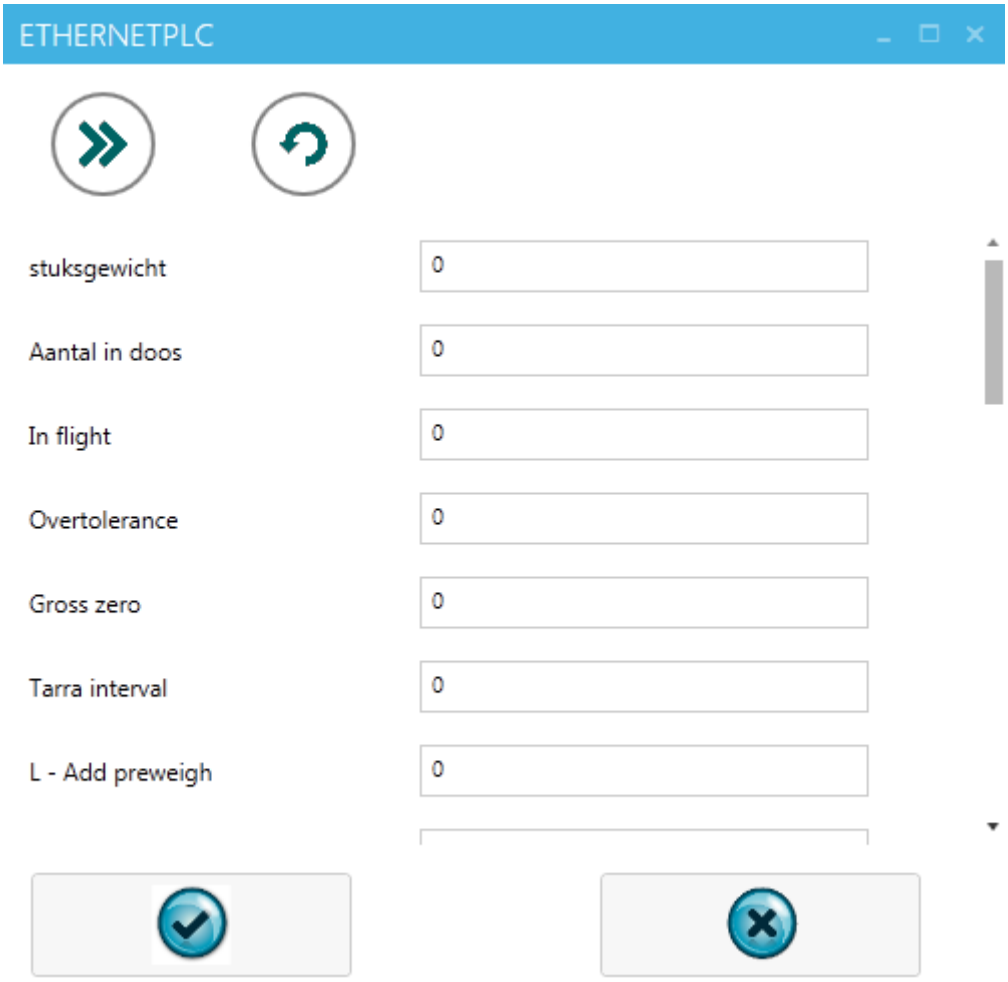


Diagram 28 – Voorbeeld schrijf bit naar DB met DB nummer groter dan 255

startBatch(), endBatch()
Deze twee methodes sturen respectievelijk het startsignaal voor een opdracht en het eindsignaal voor een opdracht naar de PLC.

string readParam(DParam AParam), writeParam(DParam AParam, string AValue)
Deze methodes lezen en schrijven parameters, van het type DParam, uit en naar de PLC. Elke instantie van DParam heeft onder andere een adresveld en een datatype veld. De methodes filteren het DB Nummer en adres uit het adresveld en eventueel de bitwaarde (opgeslagen als DBNummer,Adres[,Bit]) en sturen dan afhankelijk van het datatype de juiste gegevens naar de PLC of halen deze op.

De klasse EthernetPLCWindow
Deze klasse maakt het scherm aan om handmatig parameterwaardes naar de PLC te sturen of te ontvangen. In figuur 2 staat hoe dit scherm eruitziet.



Figuur 2 – Ethernet PLC scherm

Het scherm heeft twee knoppen bovenaan het scherm, de eerste om alle parameterwaardes te sturen en de tweede om alle parameterwaardes op te halen uit de PLC. Onderaan het scherm staan twee knoppen om de aanpassingen in de lijst met parameters op te slaan of te annuleren. De lijst met parameters kan aangepast worden in de instellingen van OMS.

4.5 Systeemtest PC software

Om alle functionaliteit te testen van de PC software moeten we eerst nagaan wat die functionaliteit dan precies is. Alleen de verschillende use cases testen met willekeurige waardes is onvoldoende want dan worden niet per definitie alle speciale gevallen getest, zoals het sturen van een kommagetal of het schrijven van een waarde naar een datablok met nummer groter dan 255. Wat moet dan allemaal getest worden? Heel simpel, de use cases plus alle speciale gevallen.

Use cases

- Parameters (instellingen van een recept) schrijven naar de PLC
  Van alle datatypes is één parameter aangemaakt en deze is verstuurd naar de PLC.
  Uitkomst: geen problemen
- Parameters lezen uit de PLC
  Van alle datatypes is één parameter aangemaakt en deze is gelezen uit de PLC
  Uitkomst: geen problemen
- Een batch starten
  Er is in OMS een batch gestart, en daarna gecontroleerd wat de PLC met de gegevens heeft gedaan
  Uitkomst: Debatch werd gestart en in de PLC stond dat de batch gestart was
- Een batch beëindigd
  Er is in OMS een batch beëindigd, en daarna gecontroleerd of de batch ook beëindigd was in de PLC.
  Uitkomst: geen problemen
- Alle statusbits zijn gewijzigd in de PLC, en daarna is gekeken of OMS de waardes goed uitlas
  Uitkomst: geen problemen
- Uitlezen of een batch mag starten
  Uitkomst: geen problemen
- Uitlezen of een batch gestart is
  Uitkomst: geen problemen
- Uitlezen met welke verpakking de lijn bezig is
  Uitkomst: geen problemen
- Er is statistiek aangemaakt in de PLC, en daarna gekeken of OMS deze ontvangt
  Uitkomst: De verkeerde statistiek werd in eerste instantie gelezen door een telfout in de code.

Speciale gevallen

- Het lezen en schrijven van waardes naar een datablok met nummer groter dan 255
  Er zijn meerdere waardes gestuurd naar meerdere datablokken met het nummer groter dan 255.
  Uitkomst: Het ging goed, alleen de communicatie is wel trager dan normaal

- Het schrijven van een bit naar de PLC
  Dit was in principe al getest met het zenden van een parameter van het type boolean bij de use cases, en het ging weer goed.

4.6 Gebruikerstest en acceptatietest

Omdat dit systeem nog niet bij een klant geïnstalleerd is, zullen de gebruikerstest en acceptatietest niet in dit document meegenomen worden.

5. Evaluatie

5.1 Productevaluatie

Dit zijn de producten die opgeleverd dienen te worden:

- Product 1: Een protocol waarmee gecommuniceerd gaat worden tussen PC en PLC.
- Product 2: Een functieblok voor de PLC waardoor de PLC via ethernet kan communiceren met de PC
- Product 3: Een versie van het programma OMS dat via ethernet kan communiceren met de PLC

Deze zijn allemaal opgeleverd, getest en werken naar behoren.

5.2 Procesevaluatie

Inkomen in software die anderen geschreven hebben
Het is lastig om software van anderen te analyseren en te begrijpen waarom het zo ontworpen is. Daarnaast moet gekeken worden of de functionaliteit die ontworpen moet worden toevallig ook niet ergens anders in de code zit. Door een gemakkelijker project eerst te doen voor zowel de PLC als de PC software kan de verschillende functionaliteit van beide heel snel eigen worden gemaakt, omdat de programmeur gedwongen wordt om er achter te komen hoe het werkt. Wat bijvoorbeeld ook gedaan had kunnen worden is de programmeur een berg papierwerk en de code mee te geven, zonder opdracht, en deze te analyseren. Het was misschien sneller gegaan, maar het is een stuk saaier en minder motiverender dan een *hands-on* aanpak.

Het ontwerpen van software aanpassingen
Opgevallen is, dat in zowel de PLC als de PC software de toegevoegde code los stond van de bestaande code. Dit komt eigenlijk niet vaak voor. Het is wel zo dat bijvoorbeeld in de PLC de hardware configuratie aangepast moest worden en in de PC de ethernet PLC aan een lijst met constanten moest worden toegevoegd. Alleen afgezien daarvan hoefde er niet in bestaande klassen en functieblokken gerommeld te worden om de functionaliteit toe te voegen.

Het ontwerpen van de aanpassingen zelf is heel gemakkelijk, tot het punt dat verantwoord moet worden waarom gekozen is voor een bepaalde opzet.

Daarnaast zijn het ontwerp en de implementatie in werkelijkheid twee processen die dwars door elkaar heenlopen. Het is onmogelijk en verspilde tijd om eerst de software volledig te ontwerpen en het daarna te gaan implementeren. Halverwege de implementatie kan tegen problemen aangelopen worden, die bij het ontwerp nog niet helemaal duidelijk waren. Het is wel goed om een ontwerp te maken, maar dan in de vorm van use cases en sequentiediagrammen. Klasse-diagrammen ontwerpen voor de implementatie worden toch gewijzigd. Deze zijn overigens wel gemaakt alleen heel vaak gewijzigd tijdens de implementatie.

Scheiding tussen implementatie en tests
Het punt van het testen is dat de uitkomst bijna altijd de gewenste uitkomst is. Dat maakt de testdocumenten niet heel interessant. Dit komt omdat tijdens de implementatie elke methode nadat deze gemaakt is standaard word getest. De systeemtest is de laatste *line of defense* voor codefouten. Deze zouden ook bij de gebruikers- en acceptatietest kunnen voorkomen, maar dat is niet aannemelijk als deze niet bij de systeemtest naar voren komen.

Multithreading
Op het moment dat in de PC software een timer aangemaakt wordt, draait de methode van de timer op een andere thread. Dit was in eerste instantie niet helemaal duidelijk, omdat als een breakpoint in de code geplaatst wordt, alleen de hoofdthread word stilgezet bij het debuggen. Toen dit duidelijk werd is het snel opgelost.

Documentatie bijhouden
Omdat het lijkt alsof de meeste ontwerp- en implementatie beslissingen triviaal zijn is het lastig om bij te houden wat voor een buitenstaander relevante beslissingen zijn. Daarnaast kunnen problemen ook op een andere manier aangepakt worden alleen doordat aangewend is om problemen op een bepaalde manier aan te pakken komen deze niet eens in gedachte.

6. Conclusie

Aan het einde van dit project zijn alle producten opgeleverd. Er is een protocol opgesteld dat duidelijk maakt welke informatie er heen en weer gestuurd word tussen de PC en de PLC. Zowel de PLC als de PC software werkt volgens deze specificaties. Aan het einde van dit project is OMS in staat om op dezelfde manier als via de profibus-kaart te communiceren met de PLC, alleen dan over ethernet. Door de systeemtests zijn de meeste problemen uit het systeem verholpen.

Wat echter nog niet gebeurd is zijn de gebruikers- en acceptatietest. Om zeker te kunnen zijn van een goede werking van de software zullen deze nog uitgevoerd moeten worden.

7. Competenties

- **C8: Ontwerpen van een technisch informatie systeem**
  Doordat nog niet precies vaststaat hoe het protocol vormgegeven is zal hier eerst een analyse van worden moeten gemaakt. Dit zal in kaart worden moeten gebracht en daardoor kan deze competentie getest worden.

  Door het protocol op te stellen en de bijbehorende informatiestromen te analyseren blijkt dat deze competentie behaald is.

- **C10: Ontwerpen van een systeem architectuur**
  Door een volledige analyse van het klasse-diagram en gebruikte ontwerp patronen zal aannemelijk gemaakt worden dat deze betreffende competentie behaald is.

  Door een software ontwerp te maken voor zowel de PC als de PLC software blijkt dat deze competentie behaald is.

- **D16: Het realiseren van software**
  Uit de opdrachtomschrijving blijkt dat dit project voldoende code op zal leveren om deze competentie te testen**.**

  Uit de implementatie van de PC en de PLC software blijkt dat deze competentie behaald is.

8. Literatuurlijst

- Website over uitleg V-Model, auteur onbekend:
  http://www.coleyconsulting.co.uk/testtype.htm
  Oktober 2012
- Siemens manual over de CP343, auteur onbekend:
  https://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objID=10806075&subtype=133300
  September 2012
- Artikelen over de verschillende functies van c#:
  http://www.msdn.com
  September 2012
- Overige handleidingen van siemens Step7 Functieblokken
  https://support.automation.siemens.com
  September 2012

9. Bijlages

9.1 Protocol PC PLC communicatie

Inleiding

Dit document beschrijft het protocol waarmee de PC met de PLC communiceert. Om dit te verduidelijken allereerst een aantal termen:

Statusbits
Een aantal bit waardes die aangeven in wat voor status de machine is, zoals gestart, gestopt of in storing.

Batch
Een batch is een opdracht van de machinelijn waarbij een bepaalde hoeveelheid van een bepaald product verwerkt wordt.

Recept
Een verzameling instellingen van de machinelijn voor een bepaald product.

Eisen

Eisen van de communicatie van de PLC naar de PC

1. De PC moet de statusbits kunnen uitlezen
    a. Berichten
    b. Errors
2. De PC moet uit kunnen lezen of een batch gestart is
3. De PC moet een batch kunnen starten en beëindigen
4. De PC moet uit kunnen lezen met welke verpakking de lijn bezig is
5. De PC moet alle instellingen die in een recept zitten kunnen lezen en schrijven
6. De PC moet statistieken uit kunnen lezen