



FHX PARSING TOOL

PBA in de Elektromechanica
Automatisering

Eindwerk voorgedragen tot het behalen van
de graad en het diploma van bachelor in de
Elektromechanica

Door: Adwin Dushi
Promotor hogeschool: Ronny Van de Voorde
Promotor bedrijf: Matthias Van Gysel

Academiejaar 2021-2022
Campus De Nayer, Jan De Nayerlaan 5, BE-2860 Sint-Katelijne-Waver

VOORWOORD

Voor u ligt de scriptie “FHX Parsing Tool”, deze bespreekt hoe de gelijknamige tool werkt om een FHX file uit te lezen en weg te schrijven naar een database. De scriptie is geschreven in functie van mijn stage om af te studeren aan Thomas More De Nayer opleiding Elektromechanica – Automatisering. Het project was in opdracht van stagebedrijf Process Automation Solutions. Hier heb ik in de periode van februari 2022 – mei 2022 het project uitgevoerd.

Samen met mijn stagebegeleider, Matthias Van Gysel, is de basis en start van het scriptie onderwerp bedacht. Het project was complex. Door veel onderzoek en inzet is de tool tot een goed einde gebracht. Tijdens het project stonden mijn stagebegeleider en mijn schoolbegeleider, Ronny Van de Voorde, altijd voor mij klaar. Ze waren er steeds als ik met vragen zat waardoor het project vlot kon verlopen.

Bij deze wil ik daarom mijn begeleiders bedanken voor hun ondersteuning en begeleiding. Ook wil ik alle contactpersonen bedanken die mij hebben geholpen het project tot een goed einde te brengen. Zonder hun hulp had ik het project nooit kunnen voltooien.

Evenzeer wil ik vrienden en familie bedanken voor hun morele steun bij het schrijven van deze scriptie. Deze motiverende woorden hebben geholpen deze scriptie tot een goed einde te brengen.

Ik wens u veel leesplezier toe.

Adwin Dushi

Zemst, 21 mei 2022

SAMENVATTING

In automatisering projecten bij het bedrijf Process Automation Solutions gebeuren er vandaag de dag veel aanpassingen aan projecten manueel. Dit neemt veel tijd in beslag en kost de klant hierdoor veel geld. Er is dus nood aan een tool die deze aanpassingen sneller en efficiënter kan uitvoeren.

Het doel van dit project is om een tool te kunnen ontwikkelen, om sneller en efficiënter aanpassingen aan projecten te kunnen maken, waardoor hier veel minder tijd aan moet worden gespendeerd. Zo kunnen de kosten voor de klant lager liggen en kan het bedrijf Process Automation Solutions nog beter concurreren met andere bedrijven en kunnen ze ook grotere projecten aannemen.

Eerst lichten we toe wat DCS Systemen zijn en de context hierrond. Daarna komt de FHX Parsing Tool aan bod en wat deze inhoud. Hierin kijken we naar de werking van de tool alsook het resultaat. Uiteindelijk krijgen we een database met overzicht van eigen gekozen attributes van de instance modules.

De tool ligt aan de basis van een verdere ontwikkeling naar later toe. Op de dag van vandaag kan er alleen maar data weggeschreven worden naar een database. Later wilt het bedrijf hier aan verder werken en zorgen voor een mogelijkheid tot bulk editing en verslagen maken van een FHX File.

INHOUDSTAFEL

VOORWOORD	2
SAMENVATTING	3
INHOUDSTAFEL.....	4
LIJST VAN ILLUSTRATIES	5
LIJST VAN AFKORTINGEN	6
INLEIDING.....	7
1 CONCEPT	8
1.1 Process Automation Solutions	8
1.2 DCS Systeem	9
1.3 FHX Parsing Tool.....	11
1.4 Opleidingen.....	12
2 BASIC DESIGN.....	13
2.1 Basis doelstellingen	13
2.1.1 De FHX structuur	13
2.1.2 Databases	15
2.2 Uitgebreide doelstellingen.....	16
2.2.1 Good practices	16
2.2.2 Planning FHX Parsing Tool	17
3 CLASSES.....	18
3.1 Class Structure	18
3.2 Structure analyzer	21
3.2.1 Pop-up Windows	21
3.2.2 Classlogger Program	24
3.3 Class Database	26
4 INSTANCE MODULES	27
4.1 Instance Structure	27
4.2 Structure analyzer	28
4.3 Attribute adaptations.....	30
4.4 Instance Database	32
5 KWALIFICATIE	34
5.1 Testing.....	34
5.1.1 Fase 1.....	34
5.1.2 Fase 2.....	37
5.1.3 Fase 3.....	39
5.2 SQL Server	40
BESLUIT.....	41
LITERATUURLIJST	42

LIJST VAN ILLUSTRATIES

Figuur 1.1 Functional levels of a manufacturing control operation	9
Figuur 3.1 Proces Model	19
Figuur 3.2 Fysiek Model	19
Figuur 3.3 Keuze Overview/Parse	21
Figuur 3.4 Bestand Selectie	21
Figuur 3.5 FHX File Kiezen	22
Figuur 3.6 Vorige Parsing Verwijderen	22
Figuur 3.7 Parsing Progressbar	23
Figuur 3.8 Class Module Voorbeeld	24
Figuur 3.9 Class Programma	25
Figuur 3.10 Class Database	26
Figuur 4.1 Instance Aanpassing Voorbeeld	27
Figuur 4.2 Voorbeeld Controle Toevoeging Bij Parsing	28
Figuur 4.3 Voorbeeld Voorwaarde Toevoeging Parsing	29
Figuur 4.4 Instance Programma	30
Figuur 4.5 Instance Database	31
Figuur 4.6 Keuze Overview/Parse	32
Figuur 4.7 Nieuw Overview Window	32
Figuur 4.8 Overview Window	32
Figuur 4.9 Overview Database	33
Figuur 5.1 Class Voorbeeld	34
Figuur 5.2 Description Attribute Test	35
Figuur 5.3 Description Attribute Controle.....	35
Figuur 5.4 Datatable Test	36
Figuur 5.5 Datatable Controle	36
Figuur 5.6 Instance Voorbeeld	37
Figuur 5.7 Instance Description Attribute Test	37
Figuur 5.8 Instance Description Attribute Controle.....	37
Figuur 5.9 Overview Controle	39

LIJST VAN AFKORTINGEN

<i>Afkorting</i>	Betekenis
<i>FHX file</i>	Database export file
<i>GMP</i>	Good Manufacturing Practise
<i>PA Solutions</i>	Process Automation Solutions
<i>ATS</i>	Automation Tooling Systems
<i>DCS</i>	Distributed Control System
<i>E&I</i>	Electrical en Instrumentation
<i>PLC</i>	Programmable Logic Controller
<i>SFC</i>	Sequential Function Charts

INLEIDING

De software applicatie en ontwikkel omgeving Emerson DeltaV wordt binnen de proces automatisering gebruikt om de aansturing en interpretatie van hardware equipment (ventielen, motoren, sensoren, ...) via een computerized systeem mogelijk te maken met een minimale noodzaak aan fysieke interacties (operator control).

De volledige configuratie van dergelijk systeem is enkel mogelijk via de specifieke software tools uitgegeven door Emerson, waarvoor vaak gespecialiseerde trainingen noodzakelijk zijn. Ook zijn er meerdere add-on applicaties beschikbaar, maar deze zijn beperkt en vaak niet toegepast op de noden van individuele bedrijven die een dergelijk automatisering systeem operationeel hebben.

Door middel van een zelf geschreven software applicatie, willen we de interne database structuur van DeltaV beschikbaar stellen voor externe tools die we nadien ook zelf kunnen ontwikkelen. Deze software applicatie zal een database export file (FHX file) als input nemen en deze ontleden en wegschrijven in een open source en toegankelijke externe database structuur. Van hieruit kunnen dan via reeds bestaande tools (bulk editing in onder andere Microsoft Access) of zelf ontwikkelde tools aanpassingen uitvoeren. Er kunnen rapporten gegenereerd worden en documentatie opgebouwd worden voor Design of good manufacturing practice (GMP) doeleinden.

De scriptie is opgedeeld in functie van de stappen dat het project doorlopen heeft. Het eerste hoofdstuk gaat over de concept fase. Hierin zullen we het hebben over hoe het project tot stand is gekomen. Hierin gebeuren ook de afbakeningen, aangezien het grotere project meerdere onderdelen kent. Hoofdstuk twee gaat over het basic design, hier zijn de doelstellingen tot stand gekomen en afspraken gemaakt omtrent de programmeer omgeving en code.

Na het basic design komen we bij een opsplitsing, hierin gaan we zeer gedetailleerd kijken naar de werking van de tool zelf en naar de uitwerking hiervan. Dit is opgesplitst in twee delen: de classes en de instance modules. Dit zijn tevens ook de grootste hoofdstukken aangezien hier ook het meeste tijd aanbesteed is in het project zelf.

Tot slot hebben we de kwalificatie waarin er getest wordt op projecten en de voordelen die dit met zich meebrengt. Kijken of alle doelstellingen wel degelijk behaald zijn, of er enige limitaties zijn en voorwaarden die moeten voldaan worden vooraleer er met de tool gewerkt kan worden.

1 **CONCEPT**

1.1 **Process Automation Solutions**

De software applicatie en ontwikkel omgeving Emerson DeltaV wordt binnen de proces automatisering gebruikt om de aansturing en interpretatie van hardware equipment (ventielen, motoren, sensoren, ...) via een computerized systeem mogelijk te maken met een minimale noodzaak aan fysieke interacties (operator control).

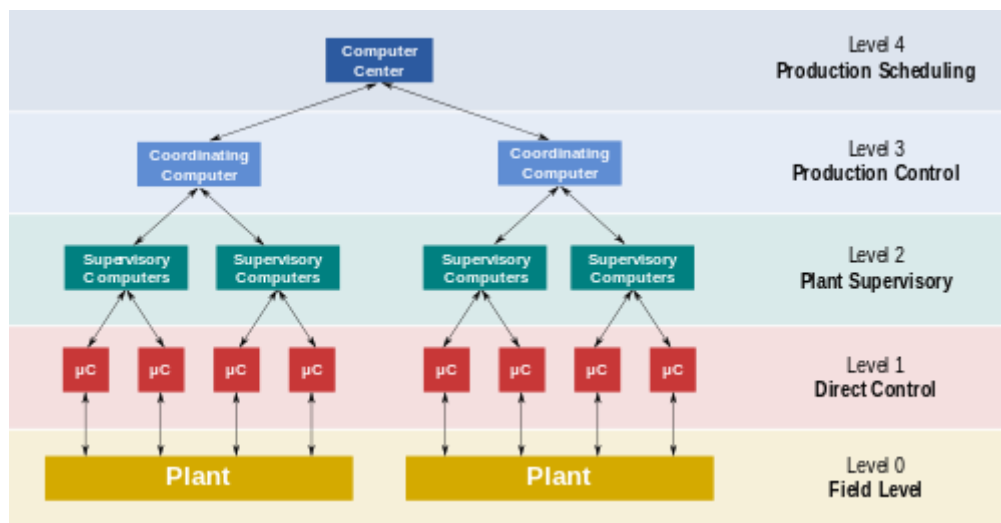
De volledige configuratie van dergelijk systeem is enkel mogelijk via de specifieke software tools, uitgegeven door Emerson, waarvoor vaak gespecialiseerde trainingen noodzakelijk zijn. Ook zijn er meerdere add-on applicaties beschikbaar, maar deze zijn beperkt en vaak niet toegepast op de noden van individuele bedrijven die een dergelijk automatisering systeem operationeel hebben.

Als onafhankelijke integrator is Process Automation Solutions (PA Solutions) actief bij een uitgebreid portfolio van klanten met een dergelijk automatisering systeem, waarbij ze veelvuldig geconfronteerd worden met de limieten van het DeltaV software pakket. Bij dit bedrijf wordt de scriptie volbracht.

PA Solutions is een onderdeel van het Canadese bedrijf Automation Tooling Systems (ATS). ATS is vooral actief in Europa en Azië en biedt diensten aan zoals het automatiseren van processen, integratie van machines bij klanten en de bouw van de machines zelf, maar ook de software en hardware implementatie. PA Solutions is sinds 2014 onderdeel van ATS. Zelf is PA Solutions wereldwijd vooral actief in de automotive sector, met op de tweede plaats de chemische sector. Ze houden zich vooral bezig met procescontrole systemen of Distributed Control System (DCS). Er is ook een tak binnen het bedrijf die zich bezighoudt met Electrical en Instrumentation (E&I). Het bedrijf werkt systeemafhankelijk met verschillende DCS systemen. Denk daarbij bijvoorbeeld aan deze van Emerson, Honeywell of Siemens. Een van deze systemen is het Emerson DeltaV systeem.

1.2 DCS Systeem

Een DCS¹ (Wikipedia, 2020) systeem wordt vooral gebruikt bij continue processen, meer daarover in hoofdstuk 3.1, waarbij er vele analoge signalen worden gebruikt en veel procesregelingen. De naam zelf verwijst al naar het feit dat er geen centrale controller is, maar verschillende die samenwerken en elk een deel van een proces krijgen toegewezen. Op deze manier faalt niet de hele plant als er een controller faalt, maar enkel dat proces. Een groot voordeel is dat de controllers dicht bij de processen kunnen staan waardoor I/O connecties zeer kort kunnen zijn.



Figuur 1.1 Functional levels of a manufacturing control operation

Een programmable logic controller² (PLC) (Wikipedia, 2022) wordt vooral gebruikt bij discrete processen, ook meer daarover in hoofdstuk 3.1, waar vooral gebruikt gemaakt wordt van digitale signalen.

¹ A distributed control system (DCS) is a computerised control system for a process or plant usually with many control loops, in which autonomous controllers are distributed throughout the system, but there is no central operator supervisory control.

² Een programmable logic controller (PLC, programmeerbare logische sturing) is een elektronisch apparaat met een microprocessor, dat op basis van de informatie op zijn diverse ingangen zijn uitgangen aanstuurt.

Het verschil tussen een DCS en PLC systeem is het volgende. De cycletimes bij een PLC zijn korter dan die bij een DCS, doordat er bij een PLC gebruikgemaakt wordt van een low level coding (denk hierbij aan Ladder Diagrams). Hierdoor is er vaak minder code die gecompileerd wordt waardoor er sneller kan worden gerekend. Bij DCS systemen wordt er gebruikgemaakt van high level coding (denk hierbij aan functional en sequential charts). Hierbij komt vaak meer code in de compiler en duurt dit langer.

Het grote voordeel van DCS systemen is dat het veiliger is in functie van falen en het een simpelere connectie kan maken met een human-machine interface (HMI). Bij een DCS systeem kan het monitoring design eenvoudig aangemaakt worden en zit dit al verweven in het DCS programma. Bij een PLC moet die nog apart gebeuren.

Over het algemeen verkiezen we DCS systemen boven PLC systemen als er een grote hoeveelheid I/O's aanwezig zijn.

1.3 FHX Parsing Tool

Door middel van een zelfgeschreven software applicatie willen we de interne database structuur van DeltaV beschikbaar stellen voor externe tools die we nadien ook zelf kunnen ontwikkelen. Deze software applicatie, genaamd FHX Parsing Tool, zal een FHX file als input nemen. Dit is een gewoon tekstbestand met daarin alle nodige data om, na een export naar DeltaV, het project terug op te zetten. Zo een tekstbestand kan echter wel enkele miljoenen lijnen tekst bevatten.

Binnen DeltaV gebeuren er in grote projecten veel manuele aanpassingen die veel tijd innemen. Daardoor wordt er meer arbeidstijd aangerekend aan de klant en verlopen de projecten ook veel trager. Het doel van de FHX Parsing Tool is om van aanpassingen die tot enkele weken kunnen duren, de duur terug te brengen tot enkele uren of dagen. Zo kan PA Solutions beter concurreren met de bedrijven die soortgelijke tools reeds bezitten en zo grotere projecten binnenslepen.

De Parsing Tool zal het FHX bestand ontleden en wegschrijven in een open source en toegankelijke externe database structuur. Van hieruit kunnen we dan via reeds bestaande tools (bulk editing in onder andere Microsoft Access) of via zelf ontwikkelde tools aanpassingen uitvoeren, rapporten genereren en documentatie opbouwen voor Design of GMP doeleinden. De Parsing Tool is dus de basis van een veel groter project en bekijkt eveneens welke andere tools mogelijk zijn om hierop verder te werken.

Er ontstaat dus een grote database met alle gegevens die we uit de FHX file willen halen. Van hieruit kunnen we vele andere tools ontwikkelen die met deze data aan de slag gaan. Zo kan er in een volgende fase van het grotere project, een andere tool gemaakt worden die de data uit deze database terug wegschrijft in een FHX file en zo terug importeert in het DeltaV programma. Zo kan men bulk editing mogelijk maken op een eenvoudige en tijdbesparende manier.

Een andere mogelijkheid is om programma's visueel weg te schrijven naar bijvoorbeeld Microsoft Visio. Zo kunnen we documentatie opbouwen die nodig is voor sommige projecten. In FHX files zitten coördinaten van de opbouw van een programma en de verbindingen die nodig zijn om dit na te kunnen bouwen in Visio.

Zo zijn er nog vele mogelijkheden voor uitbreidingen met aan de basis de database die door deze FHX Parsing Tool ontstaat.

1.4 Opleidingen

De Tool wordt geschreven in de programmeertaal C#, in het programma Visual Studio. Aangezien ik geen ervaring had met C#, heb ik hiervoor een online opleiding gevolgd zodat ik hier mee aan de slag kon. Dit werd gedaan aan de hand van de website w3schools.com die online verschillende opleidingen aanbiedt voor programmeertalen, onder andere C#.

C# zelf werd opgelegd vanuit het IT-departement van PA Solutions, eveneens de Visual Studio omgeving. Na de opleiding te hebben gevolgd, werd er een IT deskundige aangewezen om het team te vervolledigen, zodat ik met vragen en problemen bij hem terecht kon.

Een tweede opleiding die werd gevolgd, is DeltaV Implementation I. Dit is een cursus van Emerson die de basis van het DeltaV programma uitlegt. Zo werden de volgende onderwerpen aangehaald: Controle modules, alarmen, motor controle, sequential function charts (SFC), cascade control en equipment modules. Het DeltaV programma zelf bevat veel meer dan dat, maar dit gaf toch al een basis waarmee ik aan de slag kon gaan.

Het DeltaV programma is een van de meest gebruikte binnen PA Solutions, waardoor het al snel duidelijk werd dat de tool hier op moest kunnen parsen³ (Wikipedia, 2019). Een parser converteert de ingevoerde tekst in een datastructuur. Siemens heeft ook soortgelijke DCS systemen, maar hier is de tool niet voor ontworpen.

De laatste opleiding die gevolgd moest worden, is er een van trial en error. De FHX file werd voordien nooit gebruikt binnen het programma om rechtstreeks data uit te lezen. Wel werd deze zelf al gebruikt aangezien dit de methode is om programma's te exporteren en naar ergens anders over te brengen. De structuur van zo'n files was dus nog onduidelijk en moest ontleed worden vooraleer er mee aan de slag gegaan kon worden. Hoe de structuur opgebouwd wordt, zal duidelijk worden in het volgende hoofdstuk 'Basic Design'. Tijdens dit proces is er ook een DeltaV deskundige bij het team gekomen om te ondersteunen bij vragen en problemen.

Er werden enkele weken uitgerekend voor deze opleidingen om ze tot een goed einde te brengen en zo goed van start te kunnen gaan met de volgende stap in het project, het Basic Design.

³ Een parser (van het Engelse to parse, ontleden, en het Latijnse pars, deel) is een computerprogramma, of component van een programma, dat de grammaticale structuur van een invoer volgens een vastgelegde grammatica ontleedt (parset).

2 BASIC DESIGN

2.1 Basis doelstellingen

2.1.1 De FHX structuur

Aan het begin van het project zijn er enkele doelstellingen besproken die zeker moeten behaald worden door de tool. Zo waren er duidelijke richtlijnen over de werking van de tool.

Het stond al vast dat het programma geschreven moest worden in C# in de programmeeromgeving Visual Studio. In deze omgeving kunnen tekstbestanden gemakkelijk inladen en kunnen we zo lijn per lijn uitlezen. Om er zeker van te zijn dat het programma de juiste data leest, maken we gebruik van notepad of notepad++ om deze bestanden manueel te lezen. Een FHX file is in wezen niet meer dan een tekstbestand.

Uit de FHX file moeten minstens volgende zaken geparst worden: analoge metingen, digitale metingen en device controls. Zo'n file bevat meer dan alleen deze zaken maar dit zijn de belangrijkste. Als de data geparst is, worden deze weggeschreven in een Excel spreadsheet. Oorspronkelijk was dat het plan maar na verloop van tijd is Excel veranderd in Access. Dit omdat werken met query's⁴ (Wikipedia, 2021) efficiënter werkt en de tool zo database driven is.

De FHX file bestaat, zoals eerder vermeld, uit een vaste structuur. Deze structuur is rechtstreeks afkomstig uit het DeltaV programma en werkt als volgt. In DeltaV bestaan er verschillende soorten modules: class modules, instance modules, modules die geen gebruikmaken van classes (meer over de laatste later),... De class module is in feite een library die wordt gebruikt door een instance module. Het is een vooraf gemaakte module die men hergebruikt als module instance zodat deze niet elke keer vanaf nul moet worden opgebouwd.

Een module is een onderdeel waarin programmatie vervat zit. Dit kan onder andere voor analoge metingen zijn. In een hele installatie zitten vaak duizenden van zo'n modules vervat, dus is het handig om deze niet elke keer opnieuw te moeten schrijven. Hier komen de classes aan te pas, deze bezitten de standaard programmatie en instellingen. Deze kunnen dan gebruikt worden in een instance, krijgen hierin nog aanpassingen en zullen dan eventueel nog extra instellingen verkrijgen.

⁴ Met een query (Engels voor vraagstelling) wordt in de informatica een opdracht bedoeld die aan een database wordt gegeven om een bepaalde actie uit te voeren, die ook potentieel gegevens teruggeeft.

Naast de instances en de classes hebben we de modules die geen gebruikmaken van een class. Deze gebruikt men om aparte gevallen te behandelen die niet onder bepaalde classes vallen. Deze behandelen we niet verder in de rest van de scriptie.

Ook zit er in de FHX file nog een batch equipment unit module. Deze behandelt alle programmatie voor batchprocessen⁵ (Wikipedia, 2018). Aangezien het niet nuttig is om hier op te bulk editen, komt ook deze module niet meer aan bod in het verdere verloop van het eindwerk.

Naast de modules zitten er ook nog allerlei andere attributen in de FHX file. Zo bevatten deze bijvoorbeeld ook nog instellingen over hoe de programmatie in elkaar zit van deze modules: de coördinaten van de locatie van de functieblokken, hoe bepaalde connecties gelegd zijn van deze functieblokken, programmatie van de functieblokken zelf, Veel van deze instellingen hebben we niet nodig voor het huidige onderdeel van de tool. Naar de uitwerking van de toekomst van de tool toe kunnen deze zaken dan wel weer handig zijn. Denk daarbij aan het documenteren van de structuur van de programma's voor good documentation practice (GDP).

⁵ Een batchproces is een productiewijze waarbij telkens een afgeronde partij of hoeveelheid van een product wordt gefabriceerd, een batch genaamd. Denk hierbij (in huis-, tuin- en keukengebruik) aan het maken van een deeg voor brood of koekjes of het koken van een pan soep. In de procesindustrie worden batchprocessen toegepast om allerlei producten te maken die niet of moeilijk met een continu proces te maken zijn.

2.1.2 Databases

Nu we hebben bepaald wat we uit de FHX file moeten halen, is het tijd om te kijken naar waar we het gaan loggen. Dit gebeurt, zoals eerder vermeld, aan de hand van Access databases. Er is gekozen om de data weg te schrijven naar 3 verschillende databases.

De eerste database is diegene waarin we alle classes verzamelen. Hierin worden alle classes weggeschreven met hun attributen en hun waarden. Dan maken we een table aan per class met hierin alle data ervan.

Een tweede database dient om alle instance modules in te verzamelen. Deze worden gesorteerd per structuur: analoge metingen, digitale metingen en device controls. Eveneens worden hier de attributen en hun data weggeschreven zodat alles gegroepeerd is.

Als laatste database hebben we er een dat een overview geeft van alle gekozen instances met hun gekozen attributen. In deze database gebeurt dan, naar later toe en het verdere verloop van de tool gedacht, de bulk editing en/of maken we overzichten om mee te doen wat men wilt.

Een FHX file kan miljoenen lijnen tekst bevatten en vatten we uiteindelijk samen in een database die zeer beknopt is. Dit zorgt voor een enorme verbetering van leesbaarheid van zo'n FHX file en is zeker een enorme verbetering naar aanpassingen toe.

2.2 Uitgebreide doelstellingen

2.2.1 Good practices

Naast de basis doelstellingen, zijn er ook nog uitgebreide doelstellingen vastgelegd voor de tool. Men moet rekening houden met de manier van programmeren. Zo zijn er enkele “good practices” waar men zich moet aan houden.

Er moet op een bepaalde manier te werk worden gegaan, zodat er optimaal gebruikgemaakt wordt van de class based en object georiënteerde manier van programmeren. Op deze manier kunnen extensies op de tool later eenvoudig aan te brengen zijn. De tool moet voor een database kunnen zorgen waarop andere tools eenvoudig inwerken.

De tool moet ook allerhande FHX files kunnen behandelen. Zo mag men er niet vanuit gaan dat alle benamingen van modules en dergelijke in alle bedrijven en programma's dezelfde zijn. Men moet streven naar een zo dynamisch mogelijke parsing, zodat deze uiteindelijk FHX files kan parsen van onafhankelijke DeltaV systemen.

Als uitbreiding op de huidige basis doelstellingen (als de tool erin slaagt om de analoge metingen, digitale metingen en device controls uit te lezen) mag de tool andere structuren uitlezen.

In de tool zelf moet men rekening houden met een leesbare structuur, witregels tussen code in en een ordelijke structuur in het programma. Ook moet men commentaren bijvoegen waar dit nodig is om de uiteenzetting van de tool goed te kunnen begrijpen.

2.2.2 Planning FHX Parsing Tool

We gaan in dit eindwerk in verschillende fases werken. Zo starten we met het schrijven van een programma dat de classes uitleest en deze wegschrijft naar een database. Eens dit werkt, beginnen we aan het uitlezen van de instance modules. In de tussentijd nemen we op wekelijkse basis contact op met het team om te kijken of alles goed verloopt en bekijken we in detail wat de huidige stand van zaken omtrent de tool is.

Als de tool klaar is, zal er een uitvoerige test fase volgen. Hierin stress testen we op verschillende FHX files en kijken we of de tool dit allemaal goed behandelt. Eveneens testen we of de tool gebruiksvriendelijk is.

Tijdens de ontwikkeling van de tool doen we af en toe praktijkervaring op door on site loopchecks te doen. Deze zorgen ervoor dat er een beter inzicht verworven wordt omtrent het DeltaV programma en de data die men gaat parsen.

3 CLASSES

3.1 Class Structure

Zoals eerder vermeld, zitten er verschillende structuren in een FHX file. Eén daarvan is de class structure. Deze bevat belangrijke data die we nodig hebben als de module instances later geparst worden. De class bevat alle data van de module die we gaan uitlezen. Vooraleer we gaan kijken naar de structuur hiervan in de FHX file zelf, is het duidelijker om eerst te kijken naar de structuur van de class module in DeltaV. De structuren hangen namelijk zeer nauw samen.

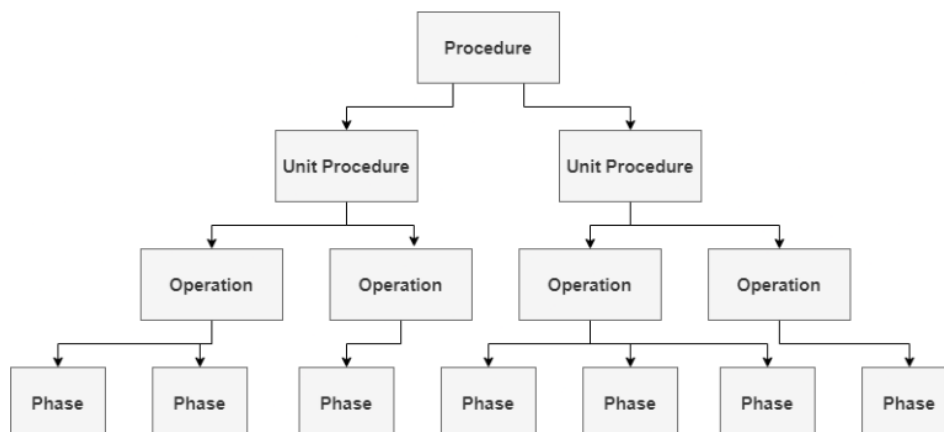
Het DeltaV volgt het ISA-S88⁶ (Eoin-Fox, 2021) model. Er zijn 3 soorten processen: batch-, continue-⁷ (Wikipedia, 2020) en discrete⁸ (Wikipedia, 2022) processen. De batch- en continue processen vallen onder de procesindustrie. Bij deze processen veranderen we de chemische en fysische eigenschappen van een product. Daar tegenover staat de discrete productie waarbij ook de vorm van het product verandert. Wij gaan ons focussen op het continu proces. Hiervoor kijken we naar het batch model. Volgens ISA-88 zijn er 2 modellen: proces- en fysiek model. Eerst gaan we ons toespitsen op het proces model.

Het proces model focust zich op de controle rangorde en bestaat uit procedures. Op figuur 3.1 tonen we deze rangorde aan. De unit procedure bestaat uit een aantal procedures die zich uitwerken op een enkele unit. Als voorbeeld van een unit zullen we een tank nemen die gevuld is met water. Een operation is een set van fases die uitgevoerd worden in een enkele unit. Een voorbeeld voor een operation kan het vullen van de watertank zijn. Zo'n operation kan verschillende fases bevatten zoals het vullen van de tank en deze dan verwarmen. Als laatste hebben we de phase of fase die simpele onafhankelijke taken kan uitvoeren. Als voorbeeld nemen we het vullen van de tank tot een bepaald niveau.

⁶ The ISA S88 standard is a guideline for batch control. It focuses on batch processes and their control. The concepts and terminology defined within make the design and operation of batch plants much easier.

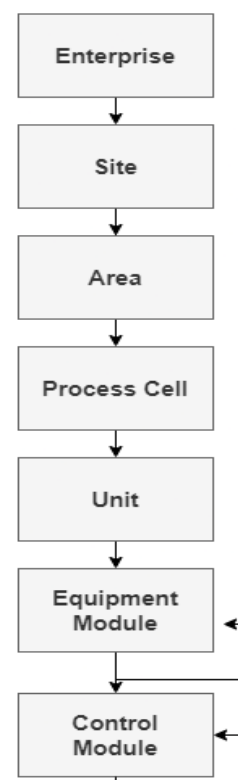
⁷ Een continu proces of flowproces is in de procestechnologie van een proces dat zonder onderbreking verloopt.

⁸ Maakindustrie is dat deel van de industrie dat materialen tot nieuwe producten verwerkt. Deze tak wordt soms ook discrete industrie of discrete productie genoemd en onderscheidt zich van de procesindustrie.



Figuur 3.1 Proces Model

Naast het proces model hebben we dus ook het fysiek model. Dit is ook een rangorde van verschillende niveaus. De lagere levels maken samen de hogere. Ook hier overlopen we weer even de verschillende lagen. Enterprise is het bedrijf zelf, beslissingen voor wat we willen produceren maken we hier. De Site is de plant zelf, dit spreekt voor zichzelf. Een Area is een onderdeel van de site waarin een batch verloopt. Als voorbeeld kan je een clean room nemen. De Process Cell bevat alle apparatuur die een recept moeten verwerken. Dan komen we bij de Equipment Module, deze runt alle procedure logica. Deze koppelen we aan apparatuur via controle modules en kan andere apparatuur en controle modules bevatten. Als laatste hebben we de Control Modules. Deze runnen alle basis logica zoals states en kan rechtstreeks apparatuur bedienen. Controle modules kunnen ook andere controle modules bevatten. Voor de FHX Parsing Tool is enkel de laatste module momenteel van toepassing.



Figuur 3.2 Fysiek Model

In het batch model bevindt zich zoals eerder vermeld een controle module. In deze controle module worden de continue processen uitgevoerd. Denk hierbij aan regelingen en kleine logica die kleppen automatisch openen bij bepaalde setpunten. Op deze continue processen en dus controle modules gaan we ons verdiepen met de FHX Parsing Tool.

Nu de structuur uitgelegd is, kunnen we gaan kijken naar het grotere geheel. Uit de FHX file willen we 2 belangrijke delen halen: de Module Instances en de Module Classes. In dit hoofdstuk gaan we het verder hebben over de Module Classes. Dit zijn templates die de module instances gebruiken (de module instances gaan we bespreken in een volgend hoofdstuk). De module classes worden aangemaakt en dan in een library gestopt, die de module instances daarna gebruiken als template.

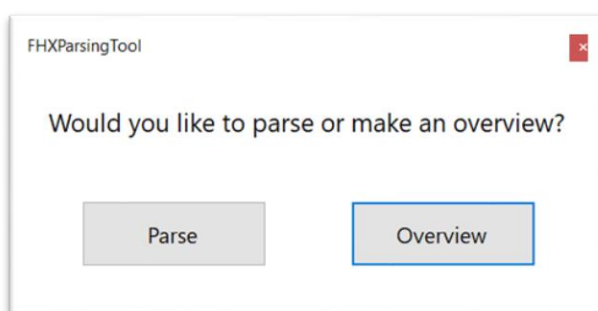
In de FHX File zelf zit alle data die nuttig is voor ons om eruit te halen. Zo komen alle instellingen van de modules aan bod, maar ook van extra modules die we niet gaan gebruiken. Coördinaten van bepaalde programma's en codering zelf is voor ons niet nuttig.

In het volgende deel gaan we ons toespitsen op de structuur van de class module.

3.2 Structure analyzer

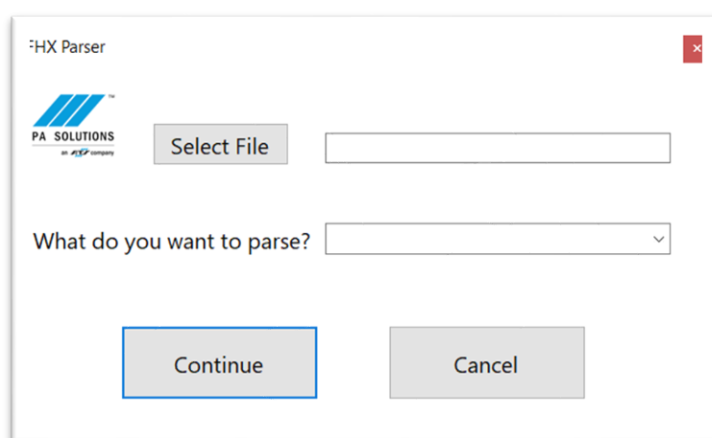
3.2.1 Pop-up Windows

Nu gaan we het hebben over het programma zelf, hoe gaat dit te werk? Eerst moeten we een paar belangrijke keuzes maken. Zo beslissen we eerst of we een parsing maken of een overview (meer over het maken van een overview in het hoofdstuk 4.5 Overview Attributes).

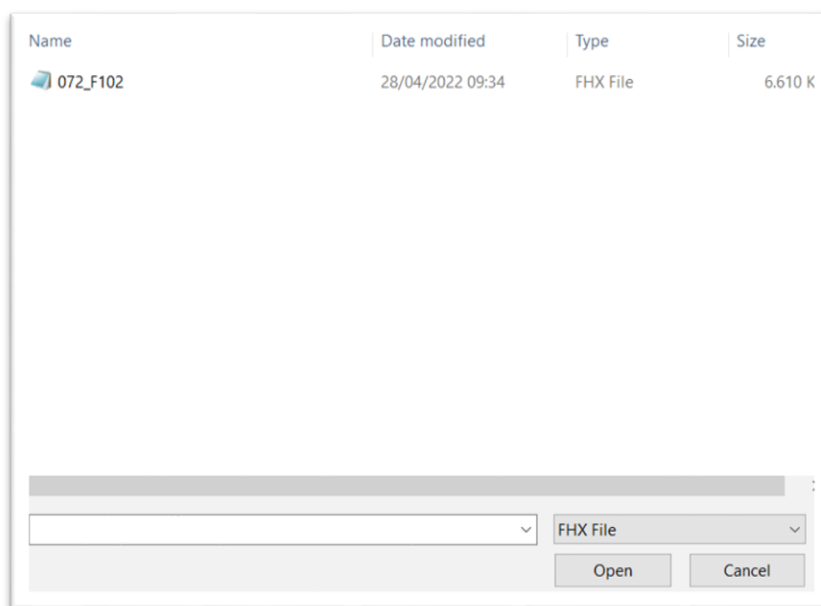


Figuur 3.3 Keuze Overview/Parse

Momenteel kiezen we voor de Parse optie. Nadat we deze gekozen hebben, moet we een bestand kiezen. We maken gebruik van Windows Forms, deze hebben ingebouwde functies die we kunnen aanroepen. In dit geval is er een functie "OpenFileDialog", deze toont een pop-up window waarin we een bestand kunnen kiezen. Deze pop-up stuurt de filepath door zodat hier later een connectie mee gemaakt kan worden. Eveneens is ingesteld dat we enkel geldige FHX Files kunnen aanklikken.



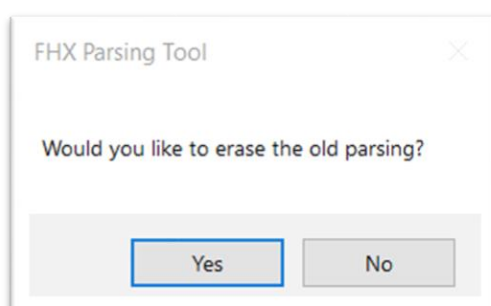
Figuur 3.4 Bestand Selectie



Figuur 3.5 FHX File Kiezen

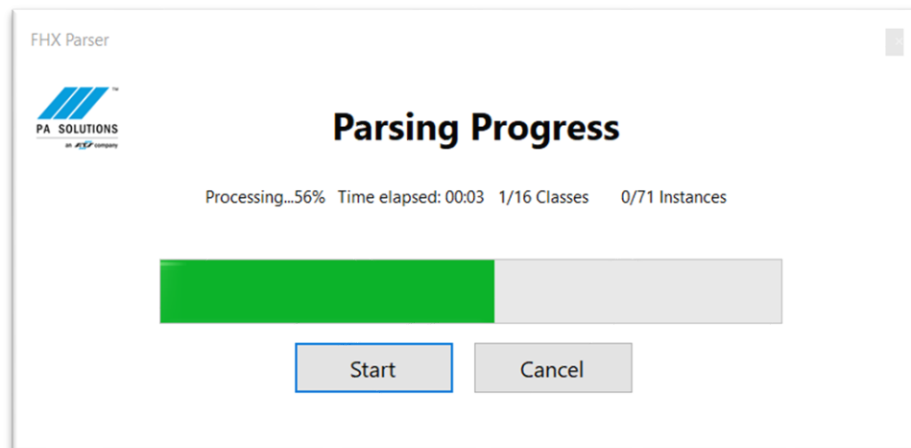
Nadat een FHX File gekozen is, kan men kiezen tussen wat we parsen: instances, classes of both. Dit betekent dat men kan kiezen tussen enkel de instances parsen, enkel de classes of beiden. Een voorwaarde voor het enkel parsen van de instances, is dat de nodige classes al eens geparst werden. Eerst was er enkel gekozen om beide samen te parsen. Zo konden we dus niet kiezen tussen enkel instances en classes. Snel werd echter duidelijk dat, als er ondertussen toch een aanpassing gebeurd was in het DeltaV programma aan de instances, ook de classes mee opnieuw geparst moesten worden, ook al is dit niet nodig.

Dan wordt er op “Continue” gedrukt en krijgen we een volgende keuze. Hier komt de vraag of de oude parsing verwijderd moet worden. Dit zorgt ervoor dat eerder geparste FHX Files verwijderd worden uit de database. Er is voor gekozen om dit te implementeren omdat we op deze manier extra modules kunnen parsen bij de huidige.



Figuur 3.6 Vorige Parsing Verwijderen

Daarna volgt het parsing proces. Hier kunnen we het proces starten worden en tonen we de vooruitgang. Men kan zien hoe ver de tool in het tekstbestand zit, hoeveel tijd er al verstreken is en hoeveel classes/instances er al geparst zijn. Er kan ook op elk moment gecancelled worden. Als het parsing proces afgelopen is, sluit het programma af. De parsing is dan compleet.



Figuur 3.7 Parsing Progressbar

Achter het visuele gedeelte zit veel meer programmatiewerk. Hier gaan we mee verder in het volgende deel.

3.2.2 Classlogger Program

Het programma gaat lijn per lijn door het tekstbestand. Als een lijn start met "MODULE_CLASS" gaan we hier op verder. Dit betekent dat de lijnen dat hierop volgen van toepassing zijn op de class module. De lijn zelf bevat enkel de naam van de class module, deze slaan we op.

```

    }
  }
}
MODULE_CLASS NAME="USM_01_DT" CATEGORY="Library/Control Module Classes/CM_HR"
user="Administrator" time=1615331243/* "10-Mar-2021 00:07:23" */
{
  DESCRIPTION="Dosing Tank Unit Support Module with Device Failure Monitoring"
  PERIOD=1
  PRIMARY_CONTROL_DISPLAY=""
  INSTRUMENT_AREA_DISPLAY=""
  DETAIL_DISPLAY=""
  TYPE="Unit Support Module"
  SUB_TYPE="USM_01_DT_T"
  NVM=F
  FUNCTION_BLOCK NAME="RESET_MSG" DEFINITION="ACT"
  {

```

Figuur 3.8 Class Module Voorbeeld

Een voorwaarde voor het programma verdergaat: is deze parsing een toevoeging aan een andere parsing? Zo ja, dan controleren we eerst of de huidige class module al in de database zit. Als deze er nog niet in zit, maken we een nieuwe table aan in Access met de naam van deze module. Eveneens wordt er een nieuwe table aangemaakt onder de vorm van: "module-naam_Exposed_Parameter". Dit zijn parameters, voor de specifieke class, die zichtbaar zijn bij de module instances onder een andere naam. In deze table schrijven we de verwijzingen weg tussen parameter naam in de class en de parameter naam in de instance.

Nadat deze 2 tables zijn aangemaakt, kunnen we alle attributes toevoegen. De tool zoekt lijn per lijn naar bepaalde woorden. Hieronder enkele van de belangrijkste opgesomd:

Attribute_Instance: Dit zijn de attributen die we eruit willen halen, of met andere woorden de instellingen van het DeltaV programma. Op deze lijn staat de naam van de attribute, gevolgd op de volgende lijnen door de waarden hiervan.

Value: Dit zegt dat er waarden gaan volgen, deze kan meer dan 1 waarde bevatten. Denk hierbij aan alarmen die een alarm priority hebben, een high limit, een low limit, ...

Description: Hier wordt er duidelijk gemaakt wat de class module doet of voorstaat. Dit is de beschrijving van de class. Dit staat apart van de attributes, dus deze moeten we ook apart zoeken.

Al deze waarden zitten binnenin één class en zullen we dus allemaal wegschrijven naar de class table, de exposed parameters naar een andere. Eens we bij de volgende class module komen, maken we weer nieuwe tables aan en schrijven we de data die hierop volgt hiernaar weg. Op deze manier zitten alle attributes netjes class per class gesorteerd.

```
//Module Klasse
if (fileLine.StartsWith("MODULE_CLASS")){...}

else if (fileLine.StartsWith("MODULE_INSTANCE")){...}

//Write to classTable
if (fileLine.StartsWith("}") & ModuleNameString.Contains("Class:")){...}

//Attributes
if (fileLine.StartsWith(" ATTRIBUTE_INSTANCE") & ModuleNameString.Contains("Class:")){...}

//Values Single Line
if ((fileLine.StartsWith(" VALUE") || ValueNotDone) & ModuleNameString.Contains("Class:")){...}

//Expose
if ((fileLine.StartsWith(" EXPOSE") || fileLine.StartsWith(" EXPOSE_IS_OVERRIDDEN") || fileL

//Values Multiple Lines
if (Value & ModuleNameString.Contains("Class:")){...}

//Class Info
if (fileLine.StartsWith(" SUB_TYPE") & ModuleNameString.Contains("Class:")){...}

if (fileLine.StartsWith(" DESCRIPTION") & ModuleNameString.Contains("Class:")){...}

//Log Exposed Parameters
if (fileLine.StartsWith(" EXPOSED_PARAMETER") & ModuleNameString.Contains("Class:")){...}
```

Figuur 3.9 Class Programma

3.3 Class Database

Het nut van deze aparte database met classes is dat we deze later kunnen oproepen bij de database van instance modules, aangezien de instances gebaseerd zijn op de classes en er achteraf aanpassingen zijn gemaakt. In onderstaand voorbeeld is een parsing gemaakt van een analoge input class. Dit voorbeeld heeft 175 attributes en delen we op in drie kolommen. De linkse kolom bevat de naam van de attribute. De middelste bezit de instelling en de rechtste de waarde hiervan.

VERSION	VALUE	5
VERSION	EXPOSE	T
VERSION	EXPOSE_IS_OVERRIDDEN	T
BLOCK_ERR	ENUM_SET	\$blk_err_opts
EXEC_TIME	VALUE	0
ONTIME	VALUE	F
VERSION_CLASS	VALUE	2
IOBAD_ALM	PRIORITY_NAME	PROCESS
IOBAD_ALM	ENAB	T
IOBAD_ALM	INV	F
IOBAD_ALM	ATYP	General I/O Failure
IOBAD_ALM	MONATTR	[EMPTY]
IOBAD_ALM	ALMATTR	CND_BAD_IO/OUT_D
IOBAD_ALM	LIMATTR	[EMPTY]
IOBAD_ALM	PARAM1	[EMPTY]
IOBAD_ALM	PARAM2	[EMPTY]
IOBAD_ALM	SUPPTIMEOUT	1438560
IOBAD_ALM	MASK	65535
IOBAD_ALM	ISDEFAULTMASK	T
IOBAD_ALM	ALARM_FUNCTIONAL_CLASSIFICATION	0
IOBAD_ALM	EXPOSE	T
IOBAD_ALM	EXPOSE_IS_OVERRIDDEN	T

Figuur 3.10 Class Database

Zoals je kan zien, zijn er meerdere instellingen voor het alarm “IOBAD”. Soms zijn er wel instellingen in het programma, maar zijn deze leeg gelaten. Deze worden weergegeven als “[EMPTY]”. Er is voor gekozen om deze niet weg te laten ook al zijn ze leeg. Zo kan men zien dat de instelling erin zit en deze leeg is, maar vooral om naar later toe aanpassingen te doen zodat we deze toch kunnen aanpassen.

Nadat de class modules geparst zijn komen de instance modules aan de beurt.

4 INSTANCE MODULES

4.1 Instance Structure

Bij de module instances blijft dezelfde structuur als bij de class modules van kracht. Het grote verschil is dat bij de class modules alle instellingen in de FHX File staan, maar niet bij de instance modules. Bij de instance modules tonen we enkel de aanpassingen. Dit zorgt ervoor dat de instance modules afhankelijk zijn van de class modules en de FHX Parsing Tool dezelfde structuur moet handhaven in zijn databases.

```

ATTRIBUTE_INSTANCE NAME="AI1/HI_LIM"  ATTRIBUTE_INSTANCE NAME="AI1$HI_LIM"
{                                       {
  VALUE { CV=0 }                       VALUE { CV=80 }
  EXPLICIT_OVERRIDE=T                  }
}
```

Figuur 4.1 Instance Aanpassing Voorbeeld

Figuur 4.1 is een simpel voorbeeld hiervan. Aan de linkerkant staat de waarde van de class module, hier staan 2 instellingen. Aan de rechterkant zie je echter maar 1 instelling waarvan de waarde afwijkt. Hier kan je zien waarom de `exposed_parametes` in de class database zo belangrijk zijn. Ook al gaat het om dezelfde attribute, toch hebben ze een verschillende naam.

Een ander verschil is natuurlijk dat bij de instance module een verwijzing staat naar de class module die we hebben gebruikt bij het aanmaken. In de FHX Files zullen weinig class modules te vinden zijn en zeer veel instance modules. Daardoor zal het parsen van deze laatste veel langer duren en zal, doordat de instances gebaseerd zijn op de modules, de code hier veel complexer door zijn.

4.2 Structure analyzer

Het programma voor instance modules werkt deels hetzelfde als dat voor de class modules. Eerst controleren we of een lijn start met "MODULE_INSTANCE". Zo ja, dan kijken we eerst aan welke voorwaarden voldaan zijn. Dit zijn er enkele meer dan bij de classes. Eerst kijken we of het de eerste instance is die we tegenkomen. Als dit het geval is dan schrijven we de data weg naar een datatable in C# in plaats van naar de database zelf. Hier zaten we met een performance issue: het is sneller om in de datatable waarden aan te passen en toe te voegen dan elke keer een query te runnen. Zo belasten we de database zelf ook minder en is er minder kans op fouten.

Als het niet de eerste instance is die we tegenkomen, schrijven we eerst de vorige datatable naar de database weg, daarna maken we deze leeg en kunnen we er terug data in stoppen voor de volgende instance.

Een andere voorwaarde is er als we aan een bestaande database, nieuwe instances gaan toevoegen. Dan kijken we eerst of deze nog niet in de database zit, als dit echter wel het geval is, dan schrijven we de huidige instance niet weg. Hieronder enkele voorbeelden van deze voorwaarden uit de tool.

```
//Check if instance is already added when dealing with old parse
if (!newParse & fileLine.StartsWith("MODULE_INSTANCE"))
{
    ValueBool = false;
    ClassLogger.ClassLogger.ModuleNameString = CleanupModule.CleanupModule.Main(fileLine);

    try
    {
        OleDbCommand InstanceInDatabase = new()
        {
            Connection = InstanceDatabaseConnection,
            CommandText = "SELECT Instance FROM Instances WHERE Instance = '" + ClassLogger.ClassLogger.ModuleNameString + "'"
        };
        OleDbDataReader Reader = InstanceInDatabase.ExecuteReader();
        using (Reader)
        {
            if (Reader.HasRows)
            {
                InstanceExists = true;
            }
            else
            {
                InstanceExists = false;
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Figuur 4.2 Voorbeeld Controle Toevoeging Bij Parsing

```
//InstanceExists is false by default when dealing with new parse  
if (!InstanceExists)  
{
```

Figuur 4.3 Voorbeeld Voorwaarde Toevoeging Parsing

De eerste foto is het algoritme dat geschreven is om te kijken of er al een instance aanwezig is in de database. Eerst schrijven we de naam van de module weg, daarna komt de try/catch methode. Deze methode zorgt ervoor dat we, bij een error in de try code, de catch code uitvoeren. Hier maak ik gebruik van door eerst aan de hand van een query te testen of de naam van de module al in de database zit. Als er een error optreedt, zal het programma niet vastlopen, maar een pop-up geven voor de error en verdergaan. Als de query een resultaat oplevert, met andere woorden er is al een module met dezelfde naam in de database gevonden, zetten we de "InstanceExist" bool op true. Als de query geen resultaat oplevert, zetten we de bool op false.

Als de bool op true staat, runnen we de code om instance modules weg te schrijven naar de database niet.

4.3 Attribute adaptations

We hebben dus al de class, die we gebruiken bij de instance, weggeschreven naar een datatable in C#. Nu zijn dit enkel de standaard waarden die we inladen in de datatable. Een instance module heeft zijn eigen aanpassingen hieraan en deze moeten we ook opslaan in de datatable. Hiervoor hebben we een apart programma dat gelijkaardig loopt aan die van de class modules.

```
//Attributes
if (fileLine.StartsWith(" ATTRIBUTE_INSTANCE") & !ClassLogger.ClassLogger.ModuleNameString.Contains("Class:"))

//Values Single Line
if (fileLine.StartsWith(" VALUE") & !ClassLogger.ClassLogger.ModuleNameString.Contains("Class:") & ClassLog

//Expose
if ((fileLine.StartsWith(" EXPOSE") || fileLine.StartsWith(" EXPOSE_IS_OVERRIDDEN")) & !ClassLogger.Clas

//Values Multiple Lines
if (ValueBool & !ClassLogger.ClassLogger.ModuleNameString.Contains("Class:"))...

//Instance Info
if (fileLine.StartsWith(" SUB_TYPE") & !ClassLogger.ClassLogger.ModuleNameString.Contains("Class:") & ClassLo

if (fileLine.StartsWith(" DESCRIPTION") & !ClassLogger.ClassLogger.ModuleNameString.Contains("Class:") & Clas
```

Figuur 4.4 Instance Programma

Zoals duidelijk zichtbaar is in figuur 4.4 hierboven, zoeken we op dezelfde attributen. Dit komt ook omdat we enkel de attributen die al in de class zitten, kunnen aanpassen. Als er een wijziging is, schrijven we deze weg naar de datatable. De huidige waarde voor de attribute gaan we simpelweg overschrijven en vervangen door de nieuwe waarde.

Als alle aanpassingen zijn uitgelezen, schrijven we de datatable naar de database weg. Dit gebeurt volgens een specifieke volgorde. De tables in de instance database zijn al aangemaakt per class. Dan kijken we naar de naam van de huidige instance module, daarna maken we een nieuwe kolom aan in de juiste class table met de module als naam. Tot slot lezen we alle data uit de datatable uit en schrijven deze weg naar de kolom. Zo bekomen we het resultaat zoals op onderstaande figuur 4.5.

Attributes	091J001FI01	091J001FI02	091J001FI03
DESCRIPTION	W001 debiet cip 92	W002 debiet cip 72	W003 debiet cos
SUB_TYPE	AI	AI	AI
MCOMMAND/SET	\$module_states	\$module_states	\$module_states
MCOMMAND/STRING_VALUE	In Service	In Service	In Service
MCOMMAND/CHANGEABLE	F	F	F
MSTATE/SET	\$module_states	\$module_states	\$module_states
MSTATE/STRING_VALUE	In Service	In Service	In Service
MSTATE/CHANGEABLE	F	F	F
MERROR/ENUM_SET	\$module_error_opts	\$module_error_opts	\$module_error_c
MSTATUS/ENUM_SET	\$module_status_opts	\$module_status_opts	\$module_status_s
MSTATUS_MASK/ENUM_SET	\$module_status_opts	\$module_status_opts	\$module_status_s
MSTATUS_MASK/EXPOSE	T	T	T
MSTATUS_MASK/EXPOSE_IS_OVERRIDDEN	T	T	T
MERROR_MASK/ENUM_SET	\$module_error_opts	\$module_error_opts	\$module_error_c
MERROR_MASK/EXPOSE	T	T	T
MERROR_MASK/EXPOSE_IS_OVERRIDDEN	T	T	T
BAD_ACTIVE/VALUE	F	F	F
ABNORM_ACTIVE/VALUE	F	F	F
VERSION/VALUE	1	1	1
VERSION/EXPOSE	T	T	T
VERSION/EXPOSE_IS_OVERRIDDEN	T	T	T
BLOCK_ERR/ENUM_SET	\$blk_err_opts	\$blk_err_opts	\$blk_err_opts
EXEC_TIME/VALUE	0	0	0
ONTIME/VALUE	F	F	F
VERSION_CLASS/VALUE	2	2	2
RATE_ALM/PRIORITY_NAME	WARNING	WARNING	WARNING
RATE_ALM/ENAB	F	F	F
RATE_ALM/INV	F	F	F
RATE_ALM/ATYP	Rate of Change	Rate of Change	Rate of Change
RATE_ALM/MONATTR	(EMPTY)	(EMPTY)	(EMPTY)
RATE_ALM/ALMATR	RATE/RATE ACT	RATE/RATE ACT	RATE/RATE ACT

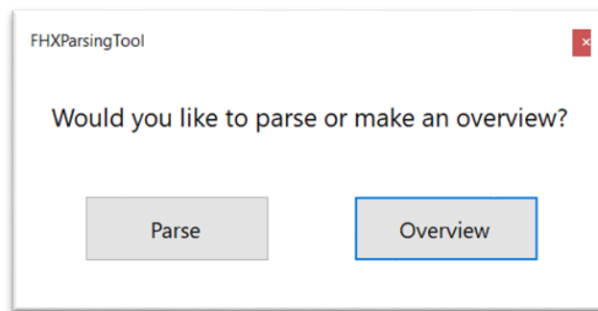
Figuur 4.5 Instance Database

Nadat de datatable is weggeschreven, maken we deze terug leeg en kunnen we de volgende instance opslagen. Eens alle instance modules aan de beurt zijn gekomen, sluit het programma af. Er komt nog een pop-up window met daarin de melding dat de parsing is afgelopen.

Eén van de problemen was dat alle ruwe data in de database niet heel overzichtelijk werkt als we aanpassingen maken aan attributes. Om een duidelijk overzicht te kunnen hebben, is er een extra programma gecreëerd. Dit programma komt in het volgende deel aan bod en zal een duidelijk overzicht maken voor wat we willen bekijken.

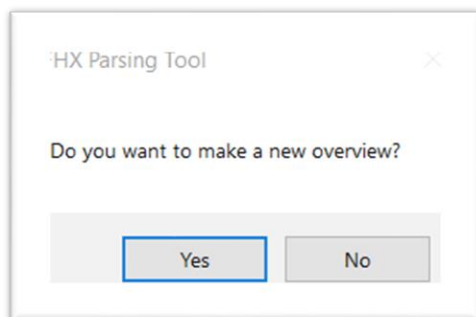
4.4 Instance Database

In hoofdstuk 3.2.1 Pop-up Windows hadden we een keuze gemaakt tussen parsen en overview. Hieronder tonen we diezelfde figuur 4.6, alleen gaan we deze keer niet kiezen voor de “Parse” optie, maar voor “Overview”.

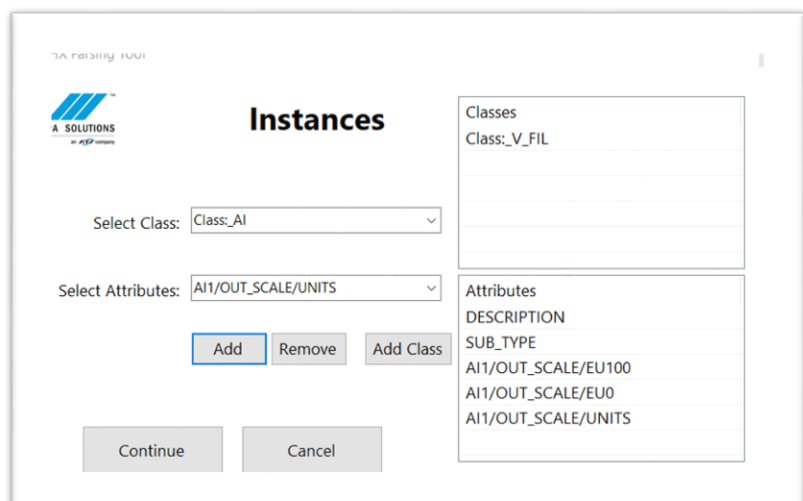


Figuur 4.6 Keuze Overview/Parse

Eens we hebben gekozen, komt er een nieuwe pop-up tevoorschijn. Deze keer moet we kiezen of we een nieuwe overview moeten maken. Dit is geïmplementeerd zodat de huidige overview kan blijven bestaan en we aanpassingen kunnen maken aan de selectie van de attributes en classes.



Figuur 4.7 Nieuw Overview Window



Figuur 4.8 Overview Window

Nu kunnen we kiezen wat er allemaal in de overview moet terechtkomen. Eerst wordt er een “Class” gekozen, dit zijn de classes die we gebruiken in de instance database waarin alle instance modules zijn onderverdeeld. Als een class is geselecteerd, kunnen de attributes hiervoor worden toegevoegd aan het overzicht. Eens alle selecties voor de class zijn geselecteerd, kunnen we ook de class met de geselecteerde attributes toevoegen aan het overzicht “Classes”. Daarna kunnen we nog classes en attributes toevoegen naar eigen noodzaak. Eens de keuzes gemaakt zijn, kunnen we verdergaan. Tot slot verkrijgen we een mooi overzicht met alle geselecteerde attributen en instances per class.

All Access ...						
Search...						
Tables						
Class: AI						
Class: V_Fil						
Instance	DESCRIPTION	SUB_TYPE	AI1/OUT_SCALE/EU100	AI1/OUT_SCALE/EU0	AI1/OUT_SCALE/UNITS	
091J001FI01	W001 debiet cip 92	AI	250	0	l/h	
091J001FI02	W002 debiet cip 72	AI	250	0	l/h	
091J001FI03	W003 debiet cosa pur 84	AI	250	0	l/h	
091J001FI04	CIP Debietmeter	AI	40	0	m3/h	
091J001PI01	Druk warme stikstof	AI	2	0	barg	
091J001PI03	CIP drukmeter	AI	10	-0.6	barg	
091J001TI13	P006 temperatuur sealpot	AI	100	0	°C	
091J001PI06	T001 Drukmeting	AI	1000	-1000	mBarg	
091J001TI01	T001 temperatuurmeting	AI	150	0	°C	
091J001TI11	P001 temperatuur sealpot	AI	100	0	°C	
091J001PI07	T002 Drukmeting	AI	1000	-1000	mBarg	
091J001TI02	T002 temperatuurmeting	AI	150	0	°C	
091J001TI12	P002 temperatuur sealpot	AI	100	0	°C	
091D003FI01	ING1 naar E001/E003	AI	1500	0	m3/hr	
091D003PI21	Meting trommeldruk	AI	10000	0	mBarg	
091D003PT05	Drukmeting axiaal membraan	AI	10000	0	mbar	
091D003PT06	Drukmeting radiaal membraan	AI	10000	0	mbar	
091D003TI01	Temperatuursmeting dichting	AI	160	0	°C	

Figuur 4.9 Overview Database

5 KWALIFICATIE

5.1 Testing

5.1.1 Fase 1

Nu we de classes en de instances kunnen uitlezen, is het tijd om te gaan testen. In de eerste fase testen we de classes, in fase twee de instances en in de derde fase de combinatie van de twee in de overview. Deze testen tonen aan of de tool wel de doelstellingen heeft behaald en of deze beperkingen heeft. Ook wordt de consistentie getest.

Bij de classes zijn er verschillende zaken die we moeten onderzoeken: uitlezen van de classes, dataverwerking en wegschrijven naar de database. Om te controleren of we de classes juist uitlezen, kijken we naar de data in de FHX file. Dit maken we concreet aan de hand van onderstaand voorbeeld.

```
MODULE_CLASS NAME="_AI_" CATEGORY="Library/Control Module Classes/Analoog"
user="BE04424" time=1551169230/* "26-Feb-2019 09:20:30" */
{
  DESCRIPTION="AI basis module met alarmering"
  PERIOD=1
  PRIMARY_CONTROL_DISPLAY=""
  INSTRUMENT_AREA_DISPLAY="S_AI_FP"
  DETAIL_DISPLAY="S_AI_DT"
  TYPE="BIGAN2"
  SUB_TYPE="_AI-"
  NVM=F
```

Figuur 5.1 Class Voorbeeld

Figuur 5.1 toont een class module genaamd “_AI_”. Als controle kijken we naar de “DESCRIPTION” attribute en controleren we of we de data juist uitlezen, verwerken en wegschrijven. We beginnen met de uitlezing.

We kijken naar de code die uitgevoerd wordt als we een description tegenkomen. Zoals we in figuur 5.2 tonen, zoeken we op specifieke attributes. De description is er een van, met een vrij eenvoudige code die twee onderliggende functies oproept. Deze codering van de functies zelf is niet van belang, echter hun functie wel. De “CleanupModule.CleanupModule.Main(fileLine)” stuurt de uitgelezen lijn door naar de functie waar we de “fileLine” inkorten naar enkel de beschrijving. De controle die we doen, is kijken of we de juiste lijn wel degelijk uitlezen.

```
if (fileLine.StartsWith(" DESCRIPTION") & ModuleNameString.Contains("Class:"))
{
    string data = CleanupModule.CleanupModule.Main(fileLine);

    //Datatable invullen
    Datasets(classTable, "DESCRIPTION", "", data);
    return;
}
```

Figuur 5.2 Description Attribute Test

Name	Value
fileLine	" DESCRIPTION=\"AI basis module met alarmering\""
data	"AI basis module met alarmering"

Figuur 5.3 Description Attribute Controle

Zoals duidelijk zichtbaar is in figuur 5.3 hierboven, lezen we de juiste lijn in. We beschouwen de test als geslaagd. De volgende testen die we gaan doen, controleren de dataverwerking. Hiervoor gaan we weer naar het voorbeeld kijken. We gaan testen of de fileLine ook effectief wordt ingekort tot enkel de nuttige beschrijving. Daarnaast bekijken we ook of het invullen van de datatable correct verloopt. We gaan dus 2 testen doen.

Voor de eerste test kijken we naar de variabele “data” waar het resultaat van de functie in terecht komt. Deze is correct en dus kunnen we deze test aanvinken als geslaagd. Bij de tweede test kijken we naar de Datasets functie. Hier stoppen we data in de datatable. We kijken voor deze test dan ook of we de data juist in de datatable stoppen.

```
//Add rows to DataTable
DataRow row = table.NewRow();
row["Attribute"] = Attribute1;
row["Multiple_Attributes"] = Attribute2;
row["Data"] = Attribute3;
table.Rows.Add(row);
```

Figuur 5.4 Datatable Test

Attribute1	"DESCRIPTION"
Attribute2	""
Attribute3	"AI basis module met alarmering"

Figuur 5.5 Datatable Controle

Op figuren 5.4 en 5.5 is te zien dat de variabele "Attribute3" de data is die we aan het controleren zijn in de test. Eveneens kunnen we controleren of de data bij de juiste attribute wordt gestopt, variabele "Attribute1". Zowel de attribute als de data zijn correct en dus kunnen we ook deze test als geslaagd zien.



Op alle verschillende soorten attributen zijn deze testen uitgevoerd aan de hand van verschillende FHX files en al deze testen zijn geslaagd. Dit zorgt ervoor dat we fase 1 afronden en we door kunnen gaan naar de volgende fase.

5.1.2 Fase 2

In de tweede fase testen we de instance modules. Dit doen we op dezelfde manier als de classes: uitlezen van de instance aanpassingen, dataverwerking en wegschrijven naar de database. We gaan dit op dezelfde manier aanpakken als bij de classes. We gebruiken hetzelfde voorbeeld en gaan hier op verder.

```
MODULE_INSTANCE TAG="FI_J301_D" PLANT_AREA="J_ALG/J372_1-6" MODULE_CLASS="_AI_" CATEGORY=""
user="MAXIM" time=1561030249/* "20-Jun-2019 13:30:49" */
{
  DESCRIPTION="Debiet Ed naar J310/1-2"
  WORK_IN_PROGRESS=F
  PERIOD=1
  PRIMARY_CONTROL_DISPLAY="Javelstockage_verlading"
  INSTRUMENT_AREA_DISPLAY="S_AI_FP"
  DETAIL_DISPLAY="S_AI_DT"
  TYPE="BIGAN2"
  SUB_TYPE="_AI-"
  NVM=F
  PRINT_BANNER_TITLE1=""
  PRINT_BANNER_TITLE2=""
  PERSIST=NONE
  USES_CLASS_SIGNATURE_POLICY=T
}
```

Figuur 5.6 Instance Voorbeeld

Name	Value
 fileLine	" DESCRIPTION=\"Debiet Ed naar J310/1-2\""
 data	"Debiet Ed naar J310/1-2"

Figuur 5.7 Instance Description Attribute Test

	Attributes	FI_J301_D
1	DESCRIPTION	Debiet Ed naar J310/1-2
2	SUB_TYPE	_AI-
3	MCOMMAND/SET	\$module_states
4	MCOMMAND/STRING_VALUE	In Service

Figuur 5.8 Instance Description Attribute Controle

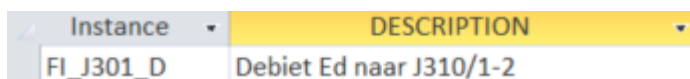
De eerste test is weer controleren of we de juiste data uitlezen. Ook in dit geval klopt dit en beschouwen we deze test als geslaagd. Een tweede test is de controle op het verwerken van de data. Dit werkt op dezelfde manier als op in fase 1 en op figuur 5.7 kunnen we het resultaat zien in de variabele "data". De data is correct ingekort en ook deze test kunnen we aanzien als geslaagd. Als laatste test gaan we kijken of de instance juist is weggeschreven naar de database. Zoals getoond op figuur 5.8 is ook dit weer correct.

Dit is weer getest op alle attributen van verschillende FHX. Zo zijn deze allemaal geslaagd voor de instances en gaan we verder naar de volgende fase.

5.1.3 Fase 3

Als laatste testen we de overview. Bij deze reeks testen gaan we kijken of alle classes uit de instance database worden gelezen en of we alle attributes vanuit deze classes kunnen selecteren. Daarnaast gaan we kijken of, na de keuzes van de classes en attributes, we de juiste data wegschrijven naar deze database.

Dit doen we weer aan de hand van het voorbeeld "DESCRIPTION". We selecteren in de overview pop-up window de juiste class en de juiste attributes. Dan kijken we naar de data die tevoorschijn komt in de database. Zo selecteren we in dit geval de juiste class en attribute "DESCRIPTION". We verwachten dat de instance data hierin zit.



Instance	DESCRIPTION
FI_J301_D	Debiet Ed naar J310/1-2

Figuur 5.9 Overview Controle

Zoals duidelijk zichtbaar in de figuur 5.9 hierboven is ook deze test geslaagd. Eveneens zijn de testen voor de uitlezing van de classes en de uitlezing van alle attributes ook geslaagd. Bij deze zijn alle testen gebeurd en geslaagd en is de testing fase afgerond.

5.2 SQL Server

Tijdens de testing fase waren alle bugs uit het programma gewerkt buiten één: “Access Violation”. Dit is een error die getoond wordt als we geheugen op een verkeerde manier aanspreken. Waarom de error exact voorkwam is niet duidelijk, maar dit is een gekend probleem bij Access databases als we werken met query’s. De error komt enkel voor bij grotere FHX files. Na uitvoerig testen en optimalisatie van de code werd duidelijk dat de bug in het programma zou blijven zitten als we bleven gebruikmaken van een Access database.

Als oplossing voor dit probleem gaan we gebruikmaken van een SQL Server voor de databases en gebruiken we de overview database nog wel van Access. Op deze manier kunnen we grote FHX files probleemloos uitlezen en wegschrijven naar databases in SQL Server. Deze werken op dezelfde manier als Access en hiervoor zijn minimale aanpassingen nodig in de code aangezien ze beide werken aan de hand van SQL. Wel zijn er kleine syntax verschillen, maar deze zijn minimaal.

We gebruiken wel nog een Access database voor de overview, deze gebruiken we als front-end⁹ en werkt gemakkelijker voor de gebruiker. SQL Server bevat dan de back-end databases. Alle testen zijn opnieuw uitgevoerd met SQL Server en zijn allemaal geslaagd.

Aangezien het project nu klaar is, gaan we kijken of alle doelstellingen behaald zijn.

⁹ In software engineering, the terms frontend and backend (or sometimes referred to as back end or back-end) refer to the separation of concerns between the presentation layer (frontend), and the data access layer (backend) of a piece of software, or the physical infrastructure or hardware

Besluit

In deze scriptie is de FHX Parsing Tool uitgelegd. Deze schrijft aan de hand van een FHX file, de data uit een DeltaV programma weg naar een database. Hier in kan men snel de data terugvinden die men wil. De werking hiervan is uitgelegd aan de hand van de classes en instances.

De tool kan alle attributes uit de FHX file lezen en deze wegschrijven naar een back-end database. Aan de hand van een Access front-end database kan men kiezen welke classes getoond worden en eveneens welke attributes hiervan. Er zijn vele mogelijkheden naar de toekomst van de tool toe, aangezien deze een database aanmaakt waardoor andere tools van deze data gebruik kunnen maken.

Naar mijn mening zijn alle doelstellingen behaald:

1. Werken met C# in Visual Studio op een georganiseerde manier en volgens de PA Solutions regelgeving van IT.
2. Verschillende onderdelen uit een FHX file kunnen halen, deze zijn: analoge metingen, digitale metingen en device controls.
3. Deze onderdelen wegschrijven naar Excel.

Voor doelstelling één kijken we naar de leesbaarheid van de code, met andere woorden het ontwijken van de zogenaamde “spaghetti” code. Comments toevoegen waar nodig zodat de code leesbaar is voor de programmeur en ervoor zorgen dat de code optimaal werkt. Uit de testfase volgt dat deze zaken zijn behaald.

Doelstelling twee zegt dat er bepaalde onderdelen uit de FHX file gehaald moeten worden. De tool haalt niet alleen enkel die zaken eruit, maar alle instellingen en attributes die er in de FHX file zitten. Deze doelstelling is dus zeker en vast behaald blijkt uit de testen.

Als laatste doelstelling moest alles weggeschreven worden naar Excel. Zoals vermeld in hoofdstuk twee is dit al snel veranderd naar Access voor efficiëntie redenen. Op het einde van het project is dan weer overgeschakeld van Access naar SQL Server voor de back-end databases. Dit in samenspraak met het bedrijf dat hier snel akkoord meeging. Dit is voor de toekomst van het bedrijf nog beter. Deze doelstelling is daarmee dus ook behaald.

LITERATUURLIJST

Eoin-Fox. (2021, Mei 17). *An introduction to ISA-88 Batch Control in DeltaV*.

Retrieved from APP Tech Blog:

<http://techblog.appliedprojectsengineering.com/post/2021/05/17/recipes-in-deltav>

Eoin-Fox. (n.d.). An introduction to ISA-88 Batch Control in DeltaV. *Process Model*. APP Tech Blog.

Eoin-Fox. (n.d.). An introduction to ISA-88 Batch Control in DeltaV. *The Physical Model*. App Tech Blog.

Pugliesi, D. (n.d.). Function levels of a manufacturing control operation. *Function levels of a manufacturing control operation*. Wikipedia, Wikipedia.

Wikipedia. (2018, Juli 13). *Batchproces*. Retrieved from Wikipedia:

<https://nl.wikipedia.org/wiki/Batchproces>

Wikipedia. (2019, April 21). *Parser*. Opgehaald van Wikipedia:

<https://nl.wikipedia.org/wiki/Parser>

Wikipedia. (2020, Oktober 27). *Continu Proces*. Retrieved from Wikipedia:

https://nl.wikipedia.org/wiki/Continu_proces

Wikipedia. (2021, Mei 2). *Query*. Retrieved from Wikipedia:

<https://nl.wikipedia.org/wiki/Query>

Wikipedia. (2022, April 23). *Distributed control system*. Retrieved from Wikipedia:

https://en.wikipedia.org/wiki/Distributed_control_system

Wikipedia. (2022, Januari 16). *Frontend and backend*. Retrieved from Wikipedia:

https://en.wikipedia.org/wiki/Frontend_and_backend

Wikipedia. (2022, Februari 28). *Maakindustrie*. Retrieved from Wikipedia:

<https://nl.wikipedia.org/wiki/Maakindustrie>

Wikipedia. (2022, Mei 19). *Programmable logic controller*. Retrieved from

Wikipedia: https://nl.wikipedia.org/wiki/Programmable_logic_controller