Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Joonas Väänänen

Rakennusautomaatiojärjestelmän ohjelma- ja grafiikkakirjasto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Opinnäytetyö

24.4.2019

Tekijä Otsikko Sivumäärä Aika	Joonas Väänänen Rakennusautomaatiojärjestelmän ohjelma- ja grafiikkakirjasto 35 sivua + 2 liitettä 18.3.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Sähkö- ja automaatiotekniikka
Suuntautumisvaihtoehto	Automaatiotekniikka
Ohjaajat	Lehtori Kristian Junno Aluejohtaja Harri Bamberg

Tutkimuksen lähtökohtana oli tilanne, jossa projektisuunnittelija tekee kutakin automaatioprojektia varten erillisen ohjelman ja käyttöliittymälle oman grafiikan. Menettely on suhteellisen työläs ja vie paljon projektisuunnittelijan arvokasta työaikaa. Tavoitteeksi asetettiin vähentää olennaisesti automaatioprojektin toteuttamiseen tarvittavaa ohjelmointiaikaa luomalla kirjastot tyypillisille ohjelmille ja käyttöliittymiin tarvittaville grafiikoille.

Opinnäytetyössä kehitettiin kirjastot rakennusautomaatio projekteissa yleisesti käytetyistä ohjelmista ja grafiikoista. Osana työtä tutkittiin standardissa IEC 61131-3 määritettyjen ohjelmointikielten soveltuvuutta rakennusautomaatio-ohjelmointiin ja yrityksen käyttämän grafiikkatyökalun pohjana toimivaan HTML-kieleen.

Tavoitteena oli saada aikaan toimivat kirjastot avuksi projektikohtaista alakeskusten ohjelmointia ja grafiikan luomista varten. Oikein käytettynä kirjastot säästävät ohjelmointiin ja graafisen käyttöliittymän toteutukseen kuluvaa aikaa.

Opinnäytetyössä tuotettiin ohjelmakirjastoon noin kolmekymmentä valmisohjelmaa ja grafiikkakirjastoon luotiin noin viisikymmentä grafiikkaa.

Ohjelmakirjastoa varten kerättiin ohjelmia yrityksen kokeneilta projektipäälliköiltä ja aluejohtajalta. Ohjelmakirjaston pohjana käytettiin isojen suunnittelutoimistojen tuottamia luetteloita tavallisesti ohjelmaa edellyttävistä tapahtumista. Suurin osa näistä tapahtumaohjelmista koodattiin osana tätä opinnäytetyötä.

Grafiikkakirjaston luomisen pohjana käytettiin Fidelix Oy:n toteuttaman mittavan projektin grafiikkakuvia. Grafiikkakuvien pohjalta luotiin grafiikat eri varustelutasoisille ilmanvaihtokoneille, lämmitysjärjestelmille sekä erillisyksiköille.

Opinnäytetyössä päästiin tavoitteeseen. Yrityksessä on nyt sekä ohjelmakirjasto että grafiikkakirjasto, joita laajentamalla automaatioprojektien ohjelmointityötä voidaan vielä nopeuttaa ja tehostaa.

Avainsanat	RAU, PLC ohjelmointi, Grafiikoiden piirto

Author	Joonas Väänänen
Title	Program- and Graphic Library for Building Automation System
Number of Pages	35 pages + 2 appendices
Date	18 March 2019
Degree	Bachelor of Engineering
Degree Programme	Electric- and automation engineering
Specialisation option	Automation Technology
Instructors	Kristian Junno, Senior Lecturer Harri Bamberg, Regional Manager

The purpose of this thesis work was to create a program and graphical user interface library from commonly used programs and graphical user interfaces in building automation projects to save project leaders time. The aim of the thesis work was to develop functional libraries in order to decrease the time spent to create programs and graphical user interface for projects.

One part of the thesis examines the feasibility of the standard IEC 61131-3 language for building automation programming and the HTML based graphic tool developed by the Fidelix Company.

During the thesis work over thirty function block programs and over fifty graphical user interfaces were created. The graphical user interfaces were created for different variations of ventilations units, heating systems, and separated electric points that were linked to function block programs.

The source of information of the program library were experienced project managers of the Fidelix company and already made event programming lists by the leading designing agencies.

As a result, the aim of developing functional libraries for programming and graphical user interface was reached. All the desired graphical user interfaces with building automation program were successfully implemented into a real-life in terms of usability and time spent to create programs and graphical user interface for projects.

Keywords	Building Automation, PLC programming

Sisällys

Lyhenteet ja käsitteet

Ala-asema	PLC-yksikkö, joka toimii valvonta-alakeskuksena.
I/O-moduli	Input/Output moduli. Tiedonsiirtoon käytetty kortti, joka muuntaa fyysisen tiedon ohjainyksikön ymmärtämään digitaaliseen muotoon.
IOT	Internet of things. Laitteet kytketään internetiin ja niistä kerätään tietoa.
IV-kone	Ilmanvaihtokone. Kone, jonka puhaltimet puhaltavat ilmaa sisään ja ulos.
LTO	Lämmöntalteenotto, Ilmastointikoneen osa, jolla poistoilmasta kerätään lämpöä tuloilmaan.
OPC	OLE for prosess control, Eri automaatiosovellusten kommunikointitapa.
PLC	Programmable logical controller eli ohjelmoitava logiikka. Automaatiojärjestelmän ohjainyksikkö.
RAU	Rakennusautomaatio. Automaation osa-alue, jolla hoidetaan muun muassa asuin- ja liiketalojen ilmastoinnin ja lämmityksen ohjausta.
VAK	Valvonta-alakeskus. Automaation ohjausyksikkö.

1 Johdanto

Opinnäytetyö tehtiin Fidelix Oy:lle, joka on Suomen markkinajohtaja rakennusautomaatiourakoinnissa. Yrityksessä on noin 200 työntekijää ja sen liikevaihto oli vuonna 2017 noin 23 miljoonaa euroa.

Kilpailu rakennusautomaatiourakoista kiristyy jatkuvasti. Tämän takia yrityksillä on tarve pienentää projektikohtaisia kulujaan voidakseen tarjota rakennusten automaatiourakat kilpailukykyiseen hintaan. Projekteihin on perinteisesti tehty kaikki ohjelmat ja grafiikat erikseen. Tekijän projektiin käyttämät työtunnit aiheuttavat projektikohtaisia kuluja, joita kovenevan kilpailun takia olisi saatava vähennetyksi.

Useissa toteutetuissa kohteissa on samanlaisia ohjelmia, jotka tehdään joka kohteessa uudelleen. Tarkoitus on saada tämän kaltaista toistotyötä vähennettyä, jolloin projektikohtaiset kustannukset pienenevät ja kannattavuus paranee

Opinnäytetyön tavoitteena on muodostaa ohjelma- ja grafiikkakirjasto helpottamaan rakennusautomaatioprojektin johtamista. Projektiin kuuluu laitteiden ja järjestelmien saattaminen käyttövalmiiksi. Tällöin järjestelmä sisältää oikein toimivan, helppokäyttöisellä graafisella käyttöliittymällä varustetun ohjelman.

Automaatioprojektin ohjelma- ja grafiikkakirjaston luominen valikoitui opinnäytteen aiheeksi siksi, että käytännössä toteutettavan projektin ei tarvitse olla kovinkaan laaja, kun ohjelmien ja tarvittavan grafiikan laadintaan kuluu aikaa useita päiviä, joista parhaimmillaan voidaan säästää olennainen osa.

Opinnäytetyö sisältää kolme toisistaan osittain erotettavissa olevaa osaa: ohjelmakirjasto, grafiikkakirjasto ja dokumenttien laatiminen.

Jo ennen tätä opinnäytetyötä ohjelmakirjaston kokoaminen oli aloitettu muutaman kerran, mutta aina oli ilmaantunut kiireellisempiä töitä, joiden takia ohjelmakirjastotyö oli jäänyt kesken. Edellisten aloitusten jäljiltä oli kuitenkin hyvät suunnitelmat, miten kirjastoa kannattaisi toteuttaa. Ohjelmien ulkoasuakaan ei tarvinnut enää miettiä, kun joitakin ohjelmia oli tehty jo valmiiksi.

2 Yritysesittely

Fidelix Oy, joka antoi opinnäytetyö aiheen, on Suomen suurin rakennusautomaatiourakoitsija. Sen henkilöstömäärä Suomessa on noin 200 henkilöä. Fidelixillä on kaksitoista aluekonttoria, joista suurimmat sijaitsevat Vantaalla, Tampereella, Oulussa ja Vaasassa. Sillä on Suomessa noin neljäkymmentä jälleenmyyjää. Suomen lisäksi Fidelix toimii Ruotsissa, jossa sillä on tytäryhtiö. Fidelix Sverige AB:lla työskentelee 20 henkilöä. Lisäksi tuotteita viedään muun muassa Norjaan, Intiaan, Isoon Britanniaan, Baltiaan, Koreaan, Kiinaan, Lähi-Itään ja Venäjälle. [1.]

Fidelix Oy pyrkii säilyttämään markkinajohtajan asemansa Suomessa. Siksi se otti vuonna 2017 tavoitteekseen kaksinkertaistaa henkilöstönsä ja liikevaihtonsa vuoden 2020 loppuun mennessä. Liikevaihto vuonna 2017 oli 23,2 miljoonaa euroa. Tilikauden tulos oli tuolloin 2,9 miljoonaa euroa. [2.]

Yritystä voi tarkastella kokonaisuutena, joka koostuu seitsemästä osastosta. Ne ovat yritysjohto, myynti, huolto, projektien toteutus, tuotekehitys, taloushallinto ja tuotanto. Vantaalla sijaitsevassa pääkonttorissa kaikki osastot ovat edustettuina. Sen lisäksi aluekonttoreissa tehdään projektin toteutusta ja huoltoa. Opinnäytetyö tehtiin projektintoteutusosastolle.

Fidelix Oy:n suurimpia valmistuneita projekteja ovat olleet Helsingin Musiikkitalo, Espoon sairaala, Lasten sairaala, Stockmann Helsinki ja Kehärata. Tällä hetkellä käynnissä on Fidelix Oy:n historian suurin projekti, Tripla. Tripla on myös pohjoismaiden suurin käynnissä oleva rakennusautomaatiohanke. Fidelix Oy:n henkilöstöä Triplassa on tällä hetkellä 10 henkilöä ja lisäksi on tarkoitus palkata useita kesätyöntekijöitä.

Yrityksen ovat vuonna 2002 perustaneet viisi automaatio-osaajaa. Heidän visionsa oli tuottaa älykäs nettipohjainen rakennusautomaatiojärjestelmä ja luotettava turvajärjestelmä. Fidelix Oy:n ensimmäisiä tuotteita oli FX2020. Ensimmäisenä nettipohjaisena, graafisen käyttöliittymän sisältävänä rakennusautomaatioalakeskuksena FX2020 oli alallaan edistyksellinen laite. [3.]

Vuonna 2006 Fidelix Oy toi markkinoille kosketusnäytölliset ala-asemat. Tämä uudistus oli yritykselle ja koko rakennusautomaatioalalle iso edistysaskel kohti nykyisin käytettävää teknologiaa. Kosketusnäytölle oli entistä helpompi helposti tuoda informaatiota sitä kautta tarkasteltavista ja säädettävistä prosesseista. [3.]

Ohjelmistoja on koko Fidelix Oy:n olemassa olon ajan kehitetty ja muokattu paremmin markkinoihin sopivaksi kuunnellen asiakkaiden toiveita. Tämän takia yrityksellä onkin oma tuotekehitysosasto. Tuotekehitysosasto pyrkii jatkuvasti kehittämään muun muassa projektinhallintaan käytettävää työkalua sekä käyttöliittymää. Yhtenä tärkeimmistä osastoista tuotekehitys Fidelix Oy:ssä voimakkaan panostuksen kohde. [3.]

Vuonna 2014 sijoitusyhtiö Procuritas osti pääosan yrityksestä. Omistajuuden vaihdos näkyi pian selkeytyvinä kehitystavoitteina. Pääomasijoittaja mahdollisti suuremmat panostukset toiminnan kehittämiseen. Tämä näkyi myös liikevaihdon ja henkilöstömäärän nousuna. [3.]

Fidelix Oy sai uuden brändiarkkitehtuurin ja logon vuonna 2016. Uusi logo on kuvassa 1. [3.]

Kuva 1. Fidelixin uusi logo [1].

Vuosina 2017 ja 2018 Fidelix Oy on kehittänyt palveluitaan entistä paremmin vastaamaan tunnistettuja markkinatarpeita. Myös huoltotoiminnan painotusta on lisätty. Vuonna 2017 lanseerattiin Compact-tuoteperhe. Näinä vuosina resursseja on keskitetty IOT:hen, energian säästöön ennakoivilla säädöillä, pilvipalveluihin ja koneellisen oppimisen kautta tapahtuvaan säädön kehittämiseen.

3 Ohjelmointi

3.1 PLC -ohjelmointi

Ohjelmointi tarkoittaa tietokoneelle annettavia toimintaohjeita. Tietokoneen suoritin ymmärtää vain konekieltä. Konekieli on ykkösistä ja nollista koostuvia merkkijonoja. Kun ohjelmointi pelkillä numeroilla on todella haastavaa, on ohjelmointityöhön jo kauan ollut saatavana erityisiä ohjelmointikieliä, joiden käskyt ovat käyttäjälle binäärilukuja havainnollisempia. Ohjelmointikieli tuottaa ohjelmoijan tekemästä ohjelmointikielen syntaksia noudattavasta koodauksesta prosessorin ymmärtämiä binäärilukumuotoisia komentoja.

Tulkkaamisella tarkoitetaan, että ohjelman lauseita käännetään yksi kerrallaan konekielelle. Ohjelman kääntämisessä koko ohjelma käännetään kerralla. Ohjelmasta käytetään myös nimitystä lähdekoodi. Sovelluksen ohjelmointikielet ovat ylemmän tason kieliä siihen kieleen nähden, jolle sovellus ohjelman kääntää. Ohjelmoitaessa ohjelmat kirjoitetaan esimerkiksi tekstimuotoisin komennoin ja sovellus kääntää tämän konekieleksi eli ykkösistä ja nollista koostuviksi numerojonoiksi. [4.]

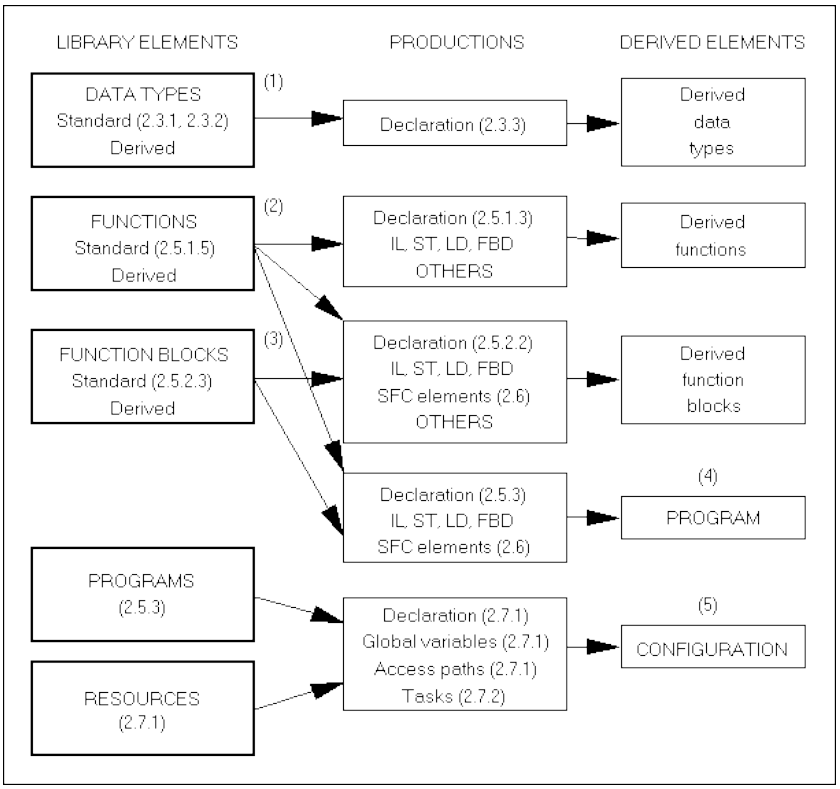
Ohjelmointikieliä on monen tasoisia. Siten ohjelmoijan käyttämää kieltä ei välttämättä käännetäkään suoraan konekielelle vaan välissä voidaan käyttää joskus useampaakin alemman tason kieltä. Tällöin ei sovelluksen tarvitse itse kääntää sille uudella ohjelmointikielellä tehtyä ohjelmaa konekielelle, vaan se voidaan kääntää aikaisemmin tehdylle kielelle, jolle käännös konekielelle on jo olemassa. Vaikeaa ja työlästä konekielelle kääntämistä voidaan näin helpottaa. [4.]

PLC tulee sanoista Programmable logical controller, suomeksi ohjelmoitava logiikka. PLC on tietokone, jota käytetään automaatiojärjestelmien ohjaukseen. Ohjelmoitavat logiikat kehitettiin teollisuuteen ohjaamaan tuotantoa, mutta ne soveltuvat yhtä hyvin rakennuksissa tapahtuvien prosessien ohjaukseen. Ohjelmoitavien logiikat ovat halventuneet käytön yleistyessä. PLC-ohjelmointiin on tehty oma standardi IEC 61131-3. Siinä kerrotaan tarkasti, miten ohjelmointikielet rakentuvat.

Ohjelmointikielissä käskyt ja muut ohjelmien peruskomponentit on koottu keskitetyiksi kirjastoiksi, joista niitä on helppo poimia sovellukseen tarvittavan ohjelman käyttöön. Ohjelmassa niitä yhdistellään toisiinsa vastaanottamaan ja käsittelemään niitä fyysisten

suureiden oloarvoista saatavia sähköisiä signaaleja, joiden perusteella prosessia ohjataan. Ohjelmien ollessa valmiita eli kun kaikki tarvittavat kirjaston komponentit on saatu kytkettyä halutulla tavalla toisiinsa, tarkastetaan ohjelman toimivuus kytkemällä ohjattavaa prosessia vastaavaan toimilaitekokonaisuuteen. Tämän jälkeen tarkistettu ohjelma tallennetaan uutena kokonaisuutena ohjelmakirjastoon. Kuvassa 2 esitetään tarkemmin ohjelmien muodostus ja tallennus kaikissa standardin 61131-3 mukaan toimivissa sovelluksissa. [5.]

Opinnäytetyön ohjelmakirjasto on tehty PLC:lle. PLC:iden ohjelmointiin on määritetty omat kielensä standardissa IEC 61131-3. Standardin sisältämät viisi ohjelmointikieltä esitellään myöhemmin tässä raportissa.



Kuva 2. Tiedostojen muodostuminen [5].

Olennainen osa PLC:n ohjelmointia on Function Blockien tekeminen. Ne tehdään yleensä kaikista usein käytettävistä ohjelmista. Niitä käyttämällä PLC on huomattavasti helpompi ohjelmoida kuin kirjoittamalla ohjelma joka kerralla uudestaan alusta alkaen. On ohjelmointikielestä ja sovelluksesta riippuvaa, miten Function Blockit lisätään ohjelmaan. Tekstipohjaisissa kielissä ne voi lisätä aina kirjoittamalla niiden kutsu. Useimmiten ne voidaan kuitenkin liittää myös ”raahaamalla” ne kirjastosta ohjelmointialustalle. Function Blockit luodaan samalla tavoin kuin normaalit ohjelmat tehdään. Tiedostoa luotaessa

valitaan ensin, luodaanko lähdekoodi, funktio vai Function Block. Function Blockeja tehdessä on huomioitava, että normaalien, ohjelmoitavaan Function Blockiin sisältyvien muuttujien lisäksi on luotava sen sisään- ja ulostulomuuttujat. Sisääntulomuuttujien arvot on luettava normaaleihin muuttujiin. Niitä ei siis voi käyttää sellaisenaan. Myös ulostulomuuttujat saavat normaaleista muuttujista arvonsa, joilla PLC säätää prosessia. Function Blockeja käytettäessä vain input- ja output -muuttujia voidaan muokata [6, s.66].

3.2 Instruction List

Instruction List (IL) on ensimmäinen standardin IEC 61131-3 määrittelemästä kahdesta tekstipohjaisesta kielestä. Se on kielenä helppokäyttöinen. Kieli koostuu siihen määritellyistä lyhyistä komennoista. Jokainen komento alkaa omalta riviltä [6, s.123]. Niitä yhdistetään muuttujiin ja siten saadaan muuttujien tilaa muutettua. Kielellä ohjelmoidakseen täytyy tuntea tarkasti, miten lyhyet komennot toimivat [6, s.127]. Siinä graafinen ulkoasu ei auta ymmärtämään, miten komennot toimivat. Instruction Listissä operaattorit ovat 1-3 kirjaimisia lyhenteitä. Komennot ovat samat kuin graafisissa kielissä. Ne on vain esitetty eri muodossa. Esimerkkikoodissa 1 on nähtävissä, miten kielellä annetaan komentoja.

```
START:      LD          %I11        (*Start button*)
            ANDN        %M15        (*Bit M15 Not on*)
            ST          %Q12        (*PUMP on*)
;
```

Esimerkkikoodi 1. Inrtuction List-kieli.

Lyhyet komennot ovat nopeita kirjoittaa, mikä mahdollistaa nopean ohjelmien tuottamisen. Lyhenteiden huono sovellettavuus kuitenkin sulkee pois monimutkaisten ohjelmien tekemisen tällä kielellä.

Function Blockien käyttämiseen Instruction Listillä on seuraavat kaksi tapaa. Function Blockia voidaan käyttää etsimällä se kirjastosta ja ”raahaamalla” ohjelmointialustalle. Tällöin ohjelma kysyy Function Blockille nimeä ja itse tuo alustalle Function Blockin sisääntulot ja ulostulot. Toinen tapa on kirjoittaa itse Function Blockin nimi ja sen sisään ja ulostulot. Tässä tavassa täytyy kuitenkin itse tietää, mitä sisään- ja ulostuloja Function Blockissa on. Myös Function Blockin nimen ja tyypin määrittäminen täytyy tällöin tehdä itse. [6, s.126.]

3.3 Structured Text

Structured Text (ST) on toinen IEC 61131-3 standardin määrittelemistä tekstipohjaisista kielistä. Se on kaikista standardin määrittelemistä kielistä vapain, eli Structured Text soveltuu hyvin erilaisten ohjelmien tekemiseen.

Kieli muistuttaa vahvasti C-kieltä. Siinä käytetään if- ja case-lauseita sekä while- ja for-silmukoita samoin kuin C-kielessä. Kielen vahvuudeksi voidaan lukea myös sen selkeys. Sen ymmärtämiseen tarvitaan englanninkielen taitoa ja vähän matemaattista ymmärrystä. Kieli koostuu niiden sekoituksesta. Silmukat ja if-lauseet annetaan englanninkielellä muuten käskyt ovat matemaattisia. Muuttujan arvoa kirjoitettaessa yhtäläisyysmerkin eteen pannaan muuttujan kaksoispiste osoittamaan, kummalle puolelle yhtäläisyysmerkkiä arvo kirjoitetaan [6, s.133], kuten alla olevasta esimerkkikoodi 2 osoittaa.

```
if MuuttujaA >= MuuttujaB then
 MuuttujaA := MuuttujaB;
elsif MuuttujaA <= MuuttujaB then
 MuuttujaB := MuuttujaA;
end_IF;
```

Esimerkkikoodi 2. Structured text-kielen if lause.

If-lauseet muodostetaan kirjoittamalla lauseen aloittavan rivin alkuun komento if, jotta sovellus tietää ehdon alkavan, jonka jälkeen kirjoitetaan ehto. Ehdon täyttyessä tehdään asiat, jotka ovat määritettynä then-komennon jälkeen. Ehdon jälkeen kirjoitetaan komento then, josta sovellus ymmärtää ehdon loppuneen. Suoritettavia asioita voi olla useita. Jokaisen asian jälkeen laitetaan puolipiste, jotta sovellus tietää asia päättyneen. Jos halutaan antaa toinen ehto, aloitetaan se lyhenteellä elsif. If-lauseissa usean ehdon toteutuessa samaan aikaan ohjelma suorittaa niistä ensimmäisenä annetun ehdon mukaisen toiminnon. Ehtolauseita voi antaa useita. If-lauseen loppuun voi antaa ehdon else. Tällä ehdolla suoritetaan sen jälkeen määritellyt asiat, jos yksikään aikaisemmin määritellyistä ehdoista ei täyty. Else -ehtoja voi antaa vain yhden kuhunkin if-lauseeseen. Siinä tapahtuvia asioita voi kuitenkin olla useita. If-lause päätetään komennolla end_if. Sanat erotetaan toisistaan alaviivalla. Näiden sanojen jälkeen kirjoitetaan puolipiste. [6, s.133.]

Case -lause muodostetaan samalla tavoin kuin if-lauseet. If- ja elsif -komentojen tilalla käytetään komentoa ”caseX”. Kirjain x korjataan juoksevalla numerolla. Tällä lauserakenteella saadaan ohjelmat selvemmiksi, sillä tämä lauserakenne on nimenomaan listan merkki. [6, s.133.]

While-silmukka aloitetaan vastaavasti kirjoittamalla silmukan alkavan rivin alkuun komento while. Tämän jälkeen kirjoitetaan ehto. While-silmukassa suoritetaan sen sisällä määritetty asia jokaisella ohjelmakierroksella, kun ehto täyttyy. Ehdon jälkeen kirjoitetaan komento then. Then-komennon jälkeen kirjoitetaan asiat, joiden halutaan tapahtuvan ehdon täyttyessä. Silmukka päätetään komennolla end_for. Komento päätetään puolipisteellä. [6, s.134.]

For-silmukka on toiminnaltaan vastaava kuin while-silmukka. For- ja while-silmukan toiminnan erottaa se, että for-silmukassa sen sisällä määritetty asia suoritetaan yhä uudestaan niin monta ohjelmakierrosta, kun lopetusehto on epätosi. While-silmukka edellyttää vastaavasti toteutusehdon täyttyessä suorittamaan ehdon jälkeen määritetyn toiminnon yhden kerran.

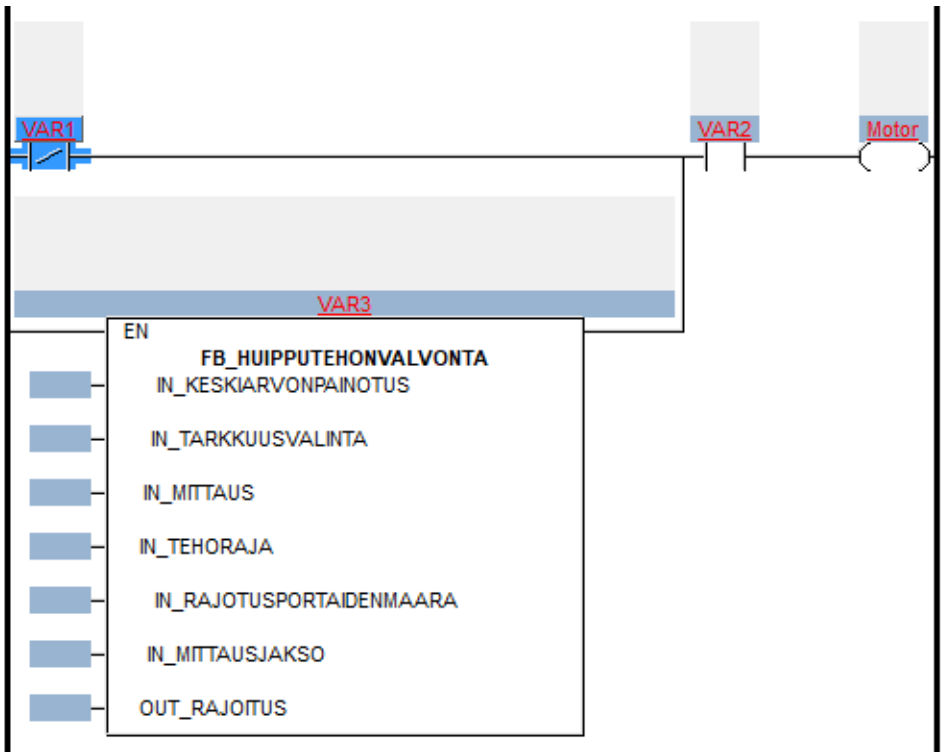
For-silmukan ensimmäisessä osassa esitetään muuttujan alkuarvo. Toisessa osassa esitetään, milloin silmukan suorittaminen lopetetaan. Kolmannessa, viimeisessä osassa kerrotaan, mitä jokaisella ohjelmointikierroksella muuttujalle tehdään.

Kun komennot järjestellään harkitusti, on ohjelmien lukeminen helppoa. Tämä vaatii kuitenkin ohjelman järjestelemistä selkeäksi. Esimerkiksi sisennysten käyttämättä jättäminen tekee ohjelmasta vaikeaselkoista ja raskasta lukea. Välilyöntienkin käyttö selkeyttää ohjelman luettavuutta.

Tällä ohjelmointikielellä voidaan helposti luoda Function Blockeja tekemällä tiedosto, jonka muodoksi määritellään Function Block. Function block -tiedosto eroaa normaalista ohjelmatiedostosta sen muuttujamäärityksissä olevilla Function Blockin sisään- ja ulostulomuuttujien määrityskohdilla. Myös niiden käyttäminen on Stuctured Text-kielellä helppoa. Niitä käytettäessä luodaan muuttuja, joka on tyypiltään kyseinen Function Block. Tämän jälkeen tätä muuttujaa täytyy kutsua ohjelmassa. Tämän muuttujan sisääntuloihin on merkittävä arvot sen mukaan, miten ne on Function Blockissa määritelty. [6, s.133.]

3.4 Ladder Diagram

Ladder diagram (LD) eli tikapuukaavio on standardin määrittelemistä graafisista kielistä ensimmäinen. Nimensä mukaisesti kielen ohjelmarakenne näyttää tikapuilta. Kieltä luotaessa on ajateltu sen soveltuvan erityisen hyvin henkilöille, jotka ovat tottuneet lukemaan sähkösuunnitelmien piirikaavioita. Kieli on suunniteltu näyttämään samalta ja toimimaan samalla periaatteella. On sanottu, että Ladder diagram on yhdeksänkymmentä astetta vasemmalle käännetty piirikaavio. Piirikaaviossa on syöttävä johto ylhäällä ja siitä alaspäin on komponentteja, kunnes viiva liittyy kuvan alareunassa olevaan nollajohtoon. LD:ssä tämä järjestys on vasemmalta oikealle [6, s.139]. Kuvassa 3 näkyy Ladder Diagramilla ohjelmointia.



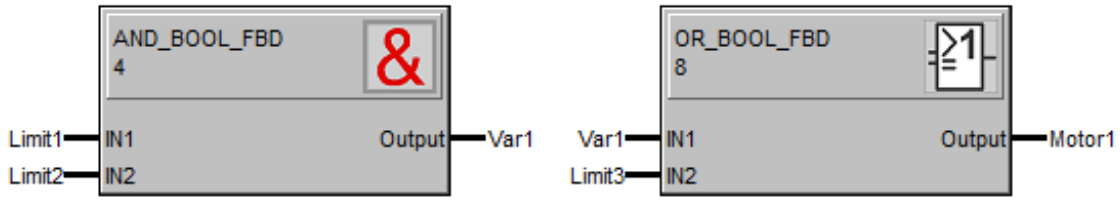
Kuva 3. Ladder Diagram.

Kuten kuvasta 3 ilmenee, Function Blockien käyttö on mahdollista myös tällä kielellä. Function Blockit näyttävät tässäkin kielessä ulkomuodoltaan melko samalta kuin Function Block Diagramissa. Vain liitynnät näyttävät erilaisilta. [6, s.141.]

3.5 Function Block Diagram

Funktio Block Diagram (FBD) on toinen standardin määrittämästä kahdesta graafisesta kielestä. Kielessä on määritelty laaja valikoima erilaisia, yksinkertaisia Function Blockeja. Näille Function Blockeille on määritelty tietty toiminto, jonka se tekee sille määritettyjen ehtojen täytyttyä. Halutut toiminnot saadaan toteutumaan yhdistämällä tarvittavat Function Blockit.

Kielen graafisessa ulkoasussa on suorakulmion muotoisia laatikoita, joissa on sisällä teksti tai merkki, joka ilmoittaa, miten laatikko eli Function Block toimii tai mitä se tekee. Laatikosta tulee ulos lyhyitä viivoja, joilla voidaan yhdistää blokkeja toisiinsa. Osassa sovelluksista voidaan Function Blockit yhdistää toisiinsa suoraan viivalla. Viivat laatikon vasemmalla puolella ovat Function Blockin sisääntuloja, laatikon oikean puoleiset viivat ovat Function Blockin ulostuloja. Sisääntulo tarkoittaa viestiä, joka tulee Function Blockin ulkopuolelta sähköisessä muodossa. Sisääntulot ovat yleensä anturitietoja. Näiden perusteella Function Block ohjaa ulostuloja sille määrätyllä tavalla. Näiden laatikosta tulevien viivojen kautta ohjelmaan yhdistetään fyysiset pisteet. Mittaukset, joiden perusteella ohjauksia halutaan tehdä, yhdistetään laatikon vasemmalla puolella oleviin viivoihin. Ohjattavat pisteet yhdistetään viivoihin, jotka ovat Function Blockin oikealla puolella. Kaikkia viivoja ei tarvitse yhdistää. Jos Function Blockilla on mahdollista toteuttaa useita eri toimintoja, voi tarpeettomiin toimintoihin liittyvät liitynnät jättää käyttämättä. Tällaisissa tapauksissa täytyy kuitenkin tietää tarkkaan kaikkien liityntöjen toiminnot. Tästä syystä on oltava erityisen huolellinen nimettäessä Function Blockin liityntöjä. Liitynnät nimetään Function Blockin luomisen yhteydessä. Kuvassa 4 on esitetty Function Block Diagram -ohjelmointi. [6, s.143]



Kuva 4. Function Block Diagram.

Function Block Diagram -kieli on yksinkertainen ja graafisena helppo ymmärtää. Function Block diagram -kielessä on monia yhtäläisyyksiä Instruction Listin kanssa. Komennot ovat usein samanlaisia, mutta ne on esitetty eri muodossa.

Tälläkin kielellä automaatio-ohjelman laatija voi luoda uusia Function Blockeja sellaisiin tarpeisiin, joihin ei valmista Function Blockia ole. Ne näkyvät samanlaisina laatikoina kuin ”tehdastekoiset” Function Blockit. Myös sen käyttäminen tapahtuu samalla tavalla kuin kaikkien muiden Function Blockien. [6, s.143]

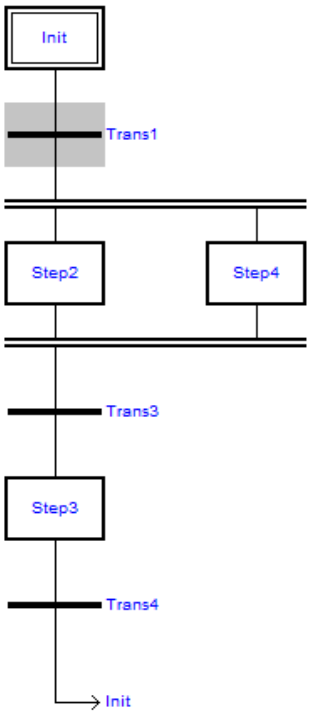
3.6 Sequential Function Chart

Sequential Function Chartia eli SFC-kieltä on vaikea määritellä graafiseksi tai tekstipohjaiseksi, koska se on sekoitus molempia. Siinä edetään sekvensseittäin ylhäältä alaspäin. Sekvenssit kuvataan laatikoina ja ohjelma viivana, jonka varrelle sekvenssit sijoitetaan. Yleensä sekvenssille määritellään myös siirtymisehto. Kielessä ehdot ja sekvenssin tapahtumat ohjelmoidaan jollakin muista standardin kielistä. [6, s.84]

Sekvenssiin siirtymisehdossa määritellään, mitä täytyy tapahtua ennen sekvenssin suorittamista. Jotta sekvenssi aloitetaan ehdon täyttyessä, täytyy ohjelman olla suorittamassa kyseistä kohtaa [6, s.84]. Kuvassa 5 näkyvät paksut mustat viivat esittävät sekvenssin siirtymisehtoja.

Sekvenssisssä määritellään, mitä siinä suoritetaan ja kuinka kauan. Sekvenssissä voidaan määrittää myös, mitä vaiheesta poistuessa tapahtuu. Poistuminen on joissain sovelluksissa jopa erotettu omaksi osakseen. [6, s.84.]

SFC:ssä voi Function Block:eja käyttää jokaisessa vaiheessa, siis sekvensseissä ja sekvenssien siirtymisehdoissa. Niinpä SFC-kielellä Function Blockeja luotaessa on otettava huomioon, että kaikki Function Blockissa olevat sisään- ja ulostuloja eivät näy kerralla, vaan ne vaikuttavat vain siihen vaiheeseen, jossa niitä on käytetty.




```
if VAR1 = VAR2 then
    Trans1 := true;
else
    Trans1 := false;
end:if;
```

Kuva 5. Sequential Function Chart.

Kuvassa 5 on määritetty siirryttävän suorittamaan sekvenssejä step2 ja step3, kun-muuttujat VAR1 ja VAR2 ovat yhtä suuret ohjelman ollessa suorittamassa Trans1:tä.

4 Ohjelmakirjasto

4.1 Ohjelmakirjaston tavoitteet ja hyödyt

Ohjelmakirjaston tekemisen tavoitteena oli koota kirjasto, joka kattaa yleisimmin käytetyt ohjelmat ja niistä tehdyt Function Blockit.

Ohjelmakirjastosta on paljon hyötyä ala-asemien ohjelmien tekemisessä. Valmiit Function Blockit nopeuttavat paljon ala-asemien ohjelmointia. Niiden liittämiseen ohjelmaan menee aikaa murto-osa verrattuna saman ohjelman uudelleen tekemiseen. Function

Blockin liittämisessä vain pistetunnukset liitetään sen sisään- ja ulostuloihin. Samat liittämiset jouduttaisiin muun koodaamisen lisäksi tekemään myös uuden ohjelman tapauksessa.

Kokeneille projektinhoitajille ja projektipäälliköille on kertynyt paljon ohjelmia aikaisemmista kohteista. Ohjelmista ei kuitenkaan aina ole tehty Function Blockeja. Aikaisemmin laaditun, käyttökelpoisen ohjelmapätkän löytäminen voi olla haastavaa, kun tehtyjä ohjelmia ei ole tallennettu systemaattisesti tiettyyn paikkaan. Osa kokeneista projektinhoitajista on koonnut itselleen kirjastoa käyttämistään ohjelmista ja Function Blockeista. Heillekin nyt luotavasta ohjelmakirjastosta voi olla hyötyä sen selkeän järjestyksen takia ja siksi, että ohjelmat on nimetty samalla nimeämisperiaatteella.

Ohjelmakirjastoa luotaessa otettiin mallia muun muassa Granlundin ja Optiplanin tapahtumaohjelmista.

Ohjelmien tekemiseen käytettävää sovellusta täytyi miettiä. Yrityksen kehittämän sovelluksen FX -Editorin uusimmassa versiossa on ohjelmointimahdollisuus. Ohjelmien tekemiseen se olisi ollut hyvä, mutta sillä testaaminen on haastavampaa kuin ohjelmakirjaston tekemiseen valitulla sovelluksella.

4.2 Ohjelmakirjaston laadintaan valittu sovellus

Ohjelmakirjasto luotiin infoteamin Open PCS sovelluksella versio 6.7. Sovellus tukee kaikkia viittä standardin IEC 61131-3 PLC:lle (Programmable logic controller) määrittämää kieltä. Sovelluksessa ohjelma luodaan jotakin näistä kielistä käyttäen ja sovellus kääntää tämän kielen PLC:n ymmärtämään muotoon. Kieliä voidaan käyttää toistensa kanssa samaan aikaan. Sovelluksessa on mahdollista, että tiedostoon sisältyy eri ohjelmointikielellä kirjoitettuja osia. [7, s.23.]

Sovellus tarkistaa ohjelman syntaksin. Syntaksi tarkoittaa ohjelmallista oikeinkirjoitusta. Sovellus huomaa seuraavat kolme virhettä:

- nollalla jakaminen
- käytetty muuttuja on käsiteltävään operaatioon soveltumatonta tyyppiä ja
- muuttujaan luettava arvo on muuttujatyyppiin sallitun skaalan ulkopuolella.

Jos ohjelmassa on suoritettu nollalla jakamista, ilmoittaa sovellus sen [7, s.156]. On myös mahdollista, että ohjelmassa jaetaan muuttujalla, joka voi saada arvokseen nollan. Tässä tapauksessa ohjelma ei ilmoita syntaksivirhettä. Jos ohjelman käydessä jakajana oleva muuttuja saa arvon nolla, ohjelma pysähtyy ja antaa ilmoituksen *nollalla jakaminen*. Jotta ohjelma käynnistyy uudelleen, on muuttujan saatava nollasta poikkeava arvo.

Jokaiseen operaatioon on määritetty tyyppi, jolla sen voi tehdä. Luonnollisesti muuttujien on oltava samaa tyyppiä, jotta niiden arvon vertaaminen on mielekästä. Yhtä luonnollisesti ajastimelle arvoja antavan muuttujan on oltava aikamuotoinen.

Sovellus tunnistaa, jos muuttujaan yritetään lukea siihen liian isoa tai liian pientä arvoa. Esimerkiksi INT-muuttujatyypin on sallittua saada vain kokonaislukuarvoja väliltä -32 768 ja +32 767. Tämä johtuu siitä, että INT-muuttuja on kooltaan 16-bittinen. Kuudellatoista bitillä saadaan 65 536 lukua, joista puolet ovat plus- ja puolet miinus-merkkisiä.

4.3 Ohjelmakirjaston tekeminen

Ohjelmakirjaston tekemisessä käytettiin kielenä structured textiä. Ohjelmointikielen valinta oli selvä, koska structured textillä on mahdollista tehdä muuttujien vertailut ja eri variaatiot saman pisteen ohjaukseen helpoiten. Silläkin oli merkitystä, että Fidelix Oy:ssä kaikki muutkin automaatio-ohjelmat tehdään tällä kielellä.

Ohjelmakirjastossa käytettiin paljon if-elsif-lauseita. Näiden lauserakenteiden käyttäminen on yksinkertaista. Apumuuttujia käyttämällä niillä saadaan aikaan myös samat asiat kuin for- ja while -silmukoilla. Kun ohjelmat tehdään tavalla, jolla voidaan suorittaa kaikki halutut toiminnot, saadaan ohjelmista selkeitä ja keskenään saman näköisiä. Tästä syystä opinnäytetyössä suosittiin if-lauseiden käyttämistä.

Ohjelmakirjastoon tehtiin suuri osa ohjelmista tämän opinnäytetyön osana. Osa function blockeista saatiin jo aikaisemmin tehdyistä projekteista, joista otettiin mallia uusien function blockien kommenttien muotoiluun. Muotoilussa oli huomioitu muun muassa ohjelmien kommentointityylit, ohjelman nimeämistapa sekä muita ohjelmoinnissa huomioitavia asioita. Ohjelmoinnissa huomioitavia asioita olivat sisentämisohjeet, pistetyyppien muuntamisohjeet ja se, että muuttujanimissä sanan vaihtuessa käytetään isoa kirjainta.

Function Blockeja tehtäessä huomioitavaa oli, että alkukommenteissa esiteltiin Function Blockin sisään menot ja ulostulot. Alkukommentteihin laitettiin myös käyttöesimerkit, jotta ne ovat helposti hyödynnettävissä kopioimalla siitä. Tällöin vain pistetunnukset on muutettava.

Ohjelmakirjaston luotaessa tehtiin ensin muut ohjelmat ja viimeiseksi Function Blockit, sillä muiden ohjelmien toiminta on helpompi koestaa. Niiden toimivuuden varmistuttua tehtiin niistä Function Blockit. Function Blockien toiminta testattiin vasta, kun ohjelmakirjasto oli muuten valmis. Näin menetellen voitiin toiminta testata nopeasti ja saatiin esiin, kirjoittaako useampi ohjelma samanaikaisesti samaan pisteeseen.

Liitteessä 1 on esimerkki Function Block, jonka alussa on kerrottu ohjelman toiminta.

Funktio Blockien pituus vaihteli paljon. Lyhin niistä oli noin 20 -rivinen. Sen pituudesta suurin osa oli kommentointia. Pisimmässä Function Blockissä oli yli 200 riviä. Function Blockien keskipituus kommentteineen oli noin 100 riviä. Alla olevassa kuvassa 6 on kirjaston lyhin Function Block, jossa itse komentoja on kolme riviä.

```
-------------------------------------------------------------------------------
      Lämmityksen asetusarvon valinta aikaohjelman mukaan V1.0
-------------------------------------------------------------------------------
Ohjelma lukee aikaohjelman. Tämän jälkeen se katsoo muunnostaulukosta aika-
ohjelman arvolla olevan asetusarvon ja kirjoittaa sen ohjelmaan.
-------------------------------------------------------------------------------
                        Blokin sisääntulot
-------------------------------------------------------------------------------
IN_Aikaohjelma          = Aikaohjelman tilatieto
IN_Muunnostaulukko      = Muunnostaulukko, josta luetaan aikaohjelman arvolla
                          asetusarvot
OUT_LampotilanAsetus    = Lämpötilan asetusarvo, joka kirjoitetaan grafiikalle

-------------------------------------------------------------------------------
                        Käyttöesimerkki
-------------------------------------------------------------------------------
Var
BlokiNimi   : FB_LammitusAsetusarvonValinta;
end_Var
-------------------------------------------------------------------------------
BlokiNimi(
IN_Aikaohjelma          := Tunnus_Aikaohjelma_T;
IN_Muunnostaulukko      := Tunnus_Muunnostaulukko_L
OUT_LampotilanAsetus    := Tunnus_LampotilanAsetus_C
);

Testattu

*)

(*-----------------------------------------------------------------------------*)
(*                          Aikaohjelman lukeminen                             *)
(*-----------------------------------------------------------------------------*)
Aikaohjelma := GetAnalogPointF(IN_Aikaohjelma);
(*-----------------------------------------------------------------------------*)
(*                          Asetusarvon hakeminen                              *)
(*-----------------------------------------------------------------------------*)
LampotilanAsetus := GetConversionTableF( Value := Aikaohjelma, Name := IN_Muunnostaulukko );
(*-----------------------------------------------------------------------------*)
(*                      Asetusarvon kirjoittaminen                             *)
(*-----------------------------------------------------------------------------*)
Tulos := SetAnalogPointF( LampotilanAsetus, 1, OUT_LampotilanAsetus);
;
```

Kuva 6. Lämmityksen asetusarvon valitseva Function Block.

Ohjelmakirjasto jakautuu neljään osa-alueeseen, jotka ovat

- IV-koneet
- lämmitysjärjestelmä
- erillispisteet
- hälytykset.

4.4 Ilmanvaihtokoneiden ohjelmat

Suurin osa ohjelmista liittyi ilmanvaihtokoneen toimintaan. Ala-asemien ohjelmoinnissa ilmanvaihtokoneiden tarvitsemien ohjelmien tekeminen vie yleensä suurimman osan työajasta. Niinpä siihen meni ohjelmakirjaston tekemisessäkin suurin osa ajasta. Lisäksi IV-koneissa on eniten vaihtelevuutta. Vaihtelevuus lisää työtä eri variaatiot kattavassa ohjelmakirjastossa. IV-koneiden ohjelmointia työllisti myös se, että ohjelmissa piti huomioida kaikki lämmön talteenoton tyypit. Erityyppisten lämmöntalteenottolaitteiden ohjaukset on toteutettava osittain eri tavalla, joten ohjelmaan on määritettävä LTO-tyyppi ja otettava siten käyttöön laitteen tyyppiä vastaavat toiminnot. Kuvassa 7 on IV-koneiden käynnistysohjelma.

```
(*                          Muuttujien lukeminen                              *)
(*----------------------------------------------------------------------------*)
Ulkolampotila := GetAnalogPointF(IN_Ulkolampotila);
Aikaohjelma := GetDigitalPointF(IN_Aikaohjelma);
KaynnistusViive := INT_TO_DINT(1000 * GetDigitalPointF(IN_KaynnistusViive));
(*----------------------------------------------------------------------------*)
(*                  Käynnistysohjelman ehtojen määritys                       *)
(*----------------------------------------------------------------------------*)
if Ulkolampotila < 5.0 and Aikaohjelma = 0 then
    Kaynnistus := 1;
else
    Kaynnistus := 0;
end_if;

(*----------------------------------------------------------------------------*)
(*                  Pakotetun käynnistyksen ajan määritys                     *)
(*----------------------------------------------------------------------------*)
KaynnistusKesto.IN := Kaynnistus;
KaynnistusKesto.PT := DINT_TO_TIME(60 * KaynnistusViive);
KaynnistusKesto();
(*----------------------------------------------------------------------------*)
(*                      Käynnistysohjelman toteutus                           *)
(*----------------------------------------------------------------------------*)
If Aikaohjelma > 0 and KaynnistusKesto.Q = 1 then
    LampotilaAsetus := 22.0;
    LTOOhjaus := 100.0;
elsif KaynnistusKesto.Q = 1 then
    LampotilaAsetus := 22.0;
end_if;
(*----------------------------------------------------------------------------*)
(*                          Muuttujien kirjoittaminen                         *)
(*----------------------------------------------------------------------------*)
if KaynnistusKesto.Q = 0 then
    Tulos := SetAnalogPointF(1.0, 0, OUT_LampotilaAsetus);
    Tulos := SetAnalogPointF(1.0, 0, OUT_LTOOhjaus);
else
    Tulos := SetAnalogPointF(LampotilaAsetus, 1, OUT_LampotilaAsetus);
    Tulos := SetAnalogPointF(LTOOhjaus, 1, OUT_LTOOhjaus);
end_if;
```

Kuva 7. Ilmanvaihtokoneiden käynnistysohjelma.

Ilmanvaihtokoneiden kirjastoon tuli seuraavat Function Blockit:

- huurtumisen esto-ohjelma
- ilmamäärän säätöohjelma
- IV-koneen käynnistysohjelma
- kuivatusohjelma
- kylmäkäynnistysohjelma
- kylmäntalteenotto-ohjelma
- LTO:n lukitusohjelma
- LTO:n hyötysuhdeohjelma
- LTO:n ohjausohjelma
- koneiden yhteinen aikaohjelma
- jäätymisvaaraohjelma
- taajuusmuuttajien lukitusohjauksienohjelma
- SFP-luvunohjelma
- tuloilman asetusarvo-ohjelma
- ylilämmön poisto-ohjelma
- yötuuletusohjelma.

4.5 Lämmitysjärjestelmän ohjelmat

Lämmitysjärjestelmään liittyviä ohjelmia oli vain muutama. Lämmitysjärjestelmän yleisimmin käytetty ja tärkein ohjelma on ilmanvaihtoverkoston häiriö. Lämmitysjärjestelmä on helposti säädettävä, eikä siinä ole poikkeustilanteita kuten ilmanvaihdossa. Lämmitysjärjestelmä ohjataan PI-säätimillä, jotka Fidelix Oy:n järjestelmässä tehdään pistemäärityksessä.

Lämmitysjärjestelmän osalta ohjelmakirjastoon tuli

- IV-verkoston häiriöohjelma
- Lämmitysasetusarvonohjelma
- Patteriventtiiliohjelma
- Pumppujen vuorotteluohjelma

4.6 Muut ohjelmat

Erillispisteisiin liittyviä ohjelmia on vain ulkovalo- ja erillispuhallinohjelma. Ulkovalojenohjelma huomio ulkovaloisuuden ja aikaohjelman. Erillispuhallinohjelma ottaa huomioon eri käynnistymistarpeet. Hälytysten toteuttamiseen yrityksessä on käytössä vain yksi Function Block, johon on ohjelmoitu kaikki hälytykset. Ohjelmakirjastoon tehtiin ohjelmia myös muista yleisesti käytetystä toiminnoista.

Muita ohjelmia ovat

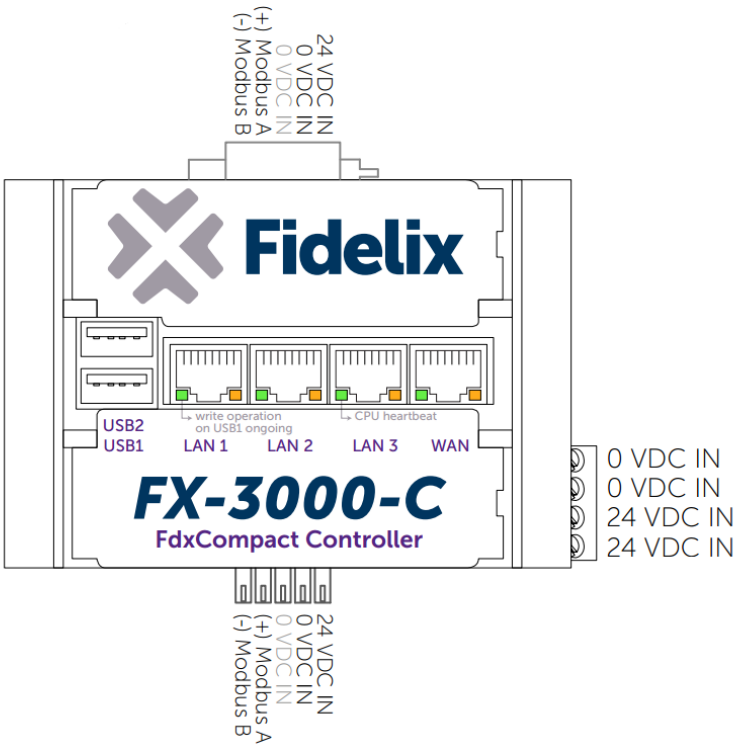
- Kiertoilmakoneohjelma
- Liukuvakeskiarvo-ohjelma
- Minimi- ja maksimiarvonlaskentaohjelma
- Porrastettu käynnistysohjelma

5 Laitteet

Kaikki grafiikat ja ohjelmat ovat tehty siten, että ne soveltuvat Fidelix Oy:n tuotteille. Ohjelmat on testattu ala-asemalla FX-3000-C. Grafiikat on optimoitu sopivan kokoisiksi Fidellix Oy:n Visio-15-C näytölle. Niitä voidaan käyttää myös kaikissa muissa Fidelix Oy:n laitteissa, mutta ne eivät ole niissä täsmälleen näytön kokoisia, mutta liikuttamalla niitä näytöllä voidaan kuitenkin saada kokonaisuus esiin.

5.1 FX-3000-C

FX-3000-C on vapaasti ohjelmoitava PLC, joka toimii samalla reitittimenä. PLC:hen on asennettu väyläliityntämahdollisuus Modbus-, M-Bus- ja BACnet -sarjaväylille. Kuvassa 8 on FX-3000-C ala-asema. [8, s.1.]



Kuva 8. Compact-sarjan ala-asema FX-3000-C [8].

Virran syöttö voidaan kytkeä laitteen sivusta tai pohjaan tulevan DIN-kiskoon asennettavan liittimen kautta. DIN-kiskoasennuksella on mahdollista saada useita laitteita saman virtalähteen syötettäväksi. Keskusyksikön käyttöjännite on 24 V:n vaihtojännite. Sen suurin sallittu virta on 7 A. [8, s.1.]

Väyläliityntä alakeskukseen voidaan toteuttaa kahdella tavalla. Ensimmäinen tapa on tuoda väylä perinteisesti RJ45-liittimen kautta. Toinen tapa on liittää laitteet pohjaan asennettavan liittimen kautta, josta saadaan myös virran syöttö. Tähän liittimeen voidaan kytkeä samanlaisia liittimiä, jotka taas sopivat mihin tahansa Compact-sarjan tuotteisiin. Liittimeen voidaan kytkeä myös FDX-Terminal-C-liitin, jolla väylä voidaan päättää tai sitä voidaan jatkaa muille laitteille. Alakeskukseen liitytään aina Modbus-väylällä. M-bus- ja BACnet-väyläliitynnät tapahtuvat Fidelixin multiLINK-mediamuuntimen avulla. [8, s.2.]

Ala-asema kytketään ulkoiseen verkkoon WAN-portin tai WiFi-yhteyden kautta. Liittimenä tässä portissa on RS485 Ethernet -liitin. Lisäksi ala-asemassa on kolme LAN-porttia, joiden kautta siihen voidaan kytkeä VISIO-15-C-näyttö, multiLINK-mediamuunnin tai niillä voidaan laajentaa sisäverkkoa. LAN-portit muodostavat ensimmäisen kahdesta reitittimien luomasta sisäverkosta. Toinen sisäverkko luodaan WLAN-yhteydelle. WLAN-yhteyden käyttämiseksi täytyy laitteeseen kiinnittää antenni. [8, s.2.]

Reitittimen hallinnointi tapahtuu laitteen päällä olevan ylemmän USB-portin kautta. Alemman USB-portin kautta voidaan muokata osaa keskusyksikön asetuksista. Ylempi portti on nimetty USB2:ksi ja alempi USB1:ksi. [8, s.2.]

Laitteessa on sisäänrakennettu web-palvelin. Palvelin siirtää tietonsa käyttäen FTP-protokollaa, joka varmistaa jokaisen lähetyksen onnistumiseen. Tämä hidastaa tiedonsiirtoa, mutta niin varmistetaan järjestelmän toiminta. Palvelimen kautta saadaan etäyhteydellä internet -selaimen kautta samanlainen näkymä kuin paikallisella näytöllä. [8, s.2.]

Laitteen ohjelmointiin käytetään Fidelix Oy:n kehittämää FX-Editor-sovellusta. Tällä sovelluksella on mahdollista ohjelmoida PLC:t. Sovelluksessa on myös grafiikkaeditori, jolla voidaan tehdä näytöllä ja etäyhteydellä näytettävät kuvat ohjattavista prosesseista. [8, s.2.]

Ala-asemassa on tallennustilana microSD -kortti, johon tallennetaan viikoittain järjestelmän tiedot. Tiedot tallentuvat muistikortille myös uudelleenkäynnistyksen yhteydessä. [8, s.2.]

Laitteessa on kaksiydinprosessori, 256 MB:n RAM-muisti ja Microsoft Windows Embedded CE 6 -käyttöjärjestelmä [8, s.2].

5.2 VISIO-15-C

VISIO-15-C on näyttö, joka yleensä asennetaan Fidelix-rakennusautomaatiojärjestelmien käyttöliittymäksi. Se ei kuitenkaan ole pakollinen osa järjestelmää, sillä ohjelma toimii ilman näyttöäkin. VISIO-15-C:llä on helppo tarkastella grafiikkakuvia, sillä sen kosketusnäyttö on hyvä ja sillä voidaan zoomata kuvia. Näyttö asennetaan tyypillisesti alakeskuksen oveen. Ovessa tulee olla tällöin 266 mm x 212 mm reikä, johon näyttö takapuoleltaan kiinnitetään. Kuvassa 9 on VISIO-15-C näyttö. [9, s.1.]

Kuva 9. VISIO-15-C-näyttö [9].

Näytön tekniset ominaisuudet ovat seuraavat [9, 2]:

- Prosessori: Rockchip RK3188 Quad Core Cortex-A9 1,6 GHz
- Muisti: 1 GB RAM, 8 GB SSD
- Virrankulutus: <650 mA @ 24 VDC
- Käyttölämpötila: 0 to +40 °C
- Verkko: TCP/IP Ethernetin tai Wi-fin kautta
- IP-luokka: 44

6 Grafiikkakirjasto

6.1 Grafiikkakirjaston tavoitteet ja hyödyt

Grafiikkakirjasto on yritykselle hyödyllinen, koska se säästää aikaa projektinhoidossa. Projektinhoidossa aikaa grafiikoiden tekoon menee tunneista useisiin päiviin. Kirjastolla voidaan tätä aikaa lyhentää merkittävästi. Jo ennen grafiikkakirjaston luomista automaatioprojektien grafiikkojen muotoilussa on käytetty hyväksi aikaisemmin toteutettujen projektien grafiikkoja työajan säästämiseksi. Vanhan grafiikan löytäminen vanhojen projektien joukosta voi olla vaikeaa. Tätä hankaluutta lievitetään nyt luotavan grafiikkakirjaston avulla. Vuositasolla työajan säästö voi olla hyvinkin merkittävä, kun vuotuinen automaatioprojektien määrä on satoja.

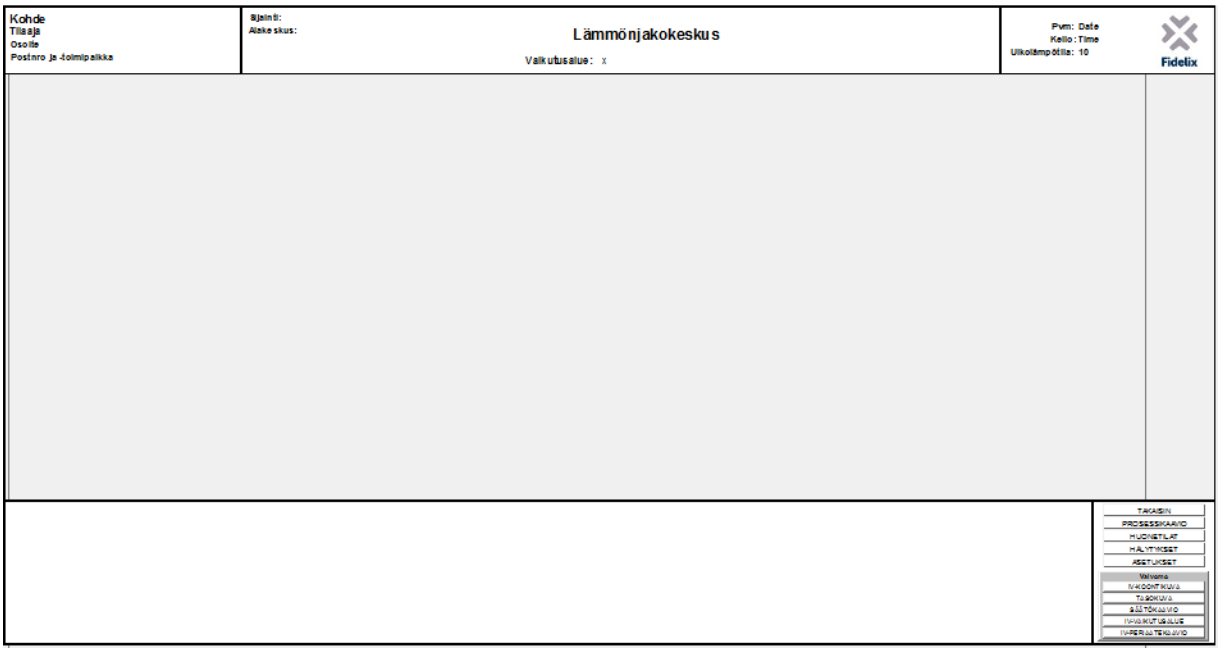
Grafiikkakirjastossa on kaikki grafiikat tehty samantyylisiksi. Yhdenmukaisuus helpottaa yrityksen strategiaan kuuluvaa palveluiden tuotteistamista. Kun grafiikoita on muokattu vanhoista projekteista, on yhden kohteen grafiikkakuvissa voinut olla erityylisiä grafiikkakuvia. Kun kohteen grafiikoista jokin on otettu vanhasta projektista, esimerkiksi kehykset ovat voineet jäädä erilaisiksi kuin muissa kuvissa.

Kuvissa on usein käytetty myös erilaisia symboleja. Symbolikirjastossa on monia erilaisia symboleja samoille asioille. Esimerkiksi ilmanvaihtojärjestelmän putkille on monia erisävyisiä keltaisia putkia, joista osa on reunoiltaan tummennettuja, jotta ne näyttäisivät enemmän putkilta. Näissäkin voi tulla erilaisuutta grafiikkakuvien välille. Eroavaisuudet kuvien välillä voivat aiheuttaa asiakkaassa tyytymättömyyttä järjestelmää kohtaan. Kirjastolla tätä voidaan vähentää.

Kehyksillä tarkoitetaan kuvan ylä- ja alapuolella olevia alueita, joissa on kuvaan liittyviä tietoja. Kehyksissä on myös usein kuvaan liittyviä hälytyspisteitä.

Liite 2 on sisältää parhaan varustelutason IV-koneiden grafiikkakuvat. Kuutio- ja nestetyyppisellä LTO-laitteella varustetut IV-koneet.

Kuvassa 10 esitetään kehykset, jollaisia on grafiikkakirjaston jokaisessa kuvassa. Osassa raporttiin otetuista kuvista ne ovat kuitenkin leikattu pois, jotta itse kuvasta saataisiin isompi ja selkeämpi.



Kuva 10. Grafiikkakirjastossa käytetyt kehykset.

Grafiikkakirjastosta pyrittiin tekemään sellainen, että yrityksen henkilöstö pitäisi sen ulkoasusta. Tällöin he mielellään aloittaisivat kirjaston käytön sen valmistuttua.

Kirjastoa tehtäessä mielipiteitä kysyttiin projektintoteutusosaston työntekijöiltä. Palaute oli pääosin positiivista. Myös palautteen osoittamat puutteet sekä muut mielipiteet ja näkemykset pyrittiin mahdollisuuksien mukaan huomioimaan. Kaikkia niitä ei kuitenkaan voitu täysin toteuttaa niiden ristiriitaisuuden vuoksi. Erilaisten mielipiteiden yhteen sovittamiseksi tehtiin kompromisseja.

Grafiikkakirjasto jaettiin kolmeen osaan:

- ilmanvaihtokoneisiin
- lämmitysjärjestelmään
- erillispisteisiin.

Näin saatiin projektin tätä osaa pienennettyä helpommin hallittaviksi kokonaisuuksiksi. Pienemmissä paloissa projektin etenemistä on helpompi seurata ja sen arvioida tehtyjen ratkaisujen toimivuutta.

6.2 Sovellus

Fidelix on kehittänyt grafiikkakuvien tekemiseen oman työkalun FX-Editorin. Se on keskitetty työkalu projektien hallinnointiin. Sillä voidaan luoda suurin osa projekteihin kuuluvista tiedostoista.

FX-Editorissa on neljä toimintoa:

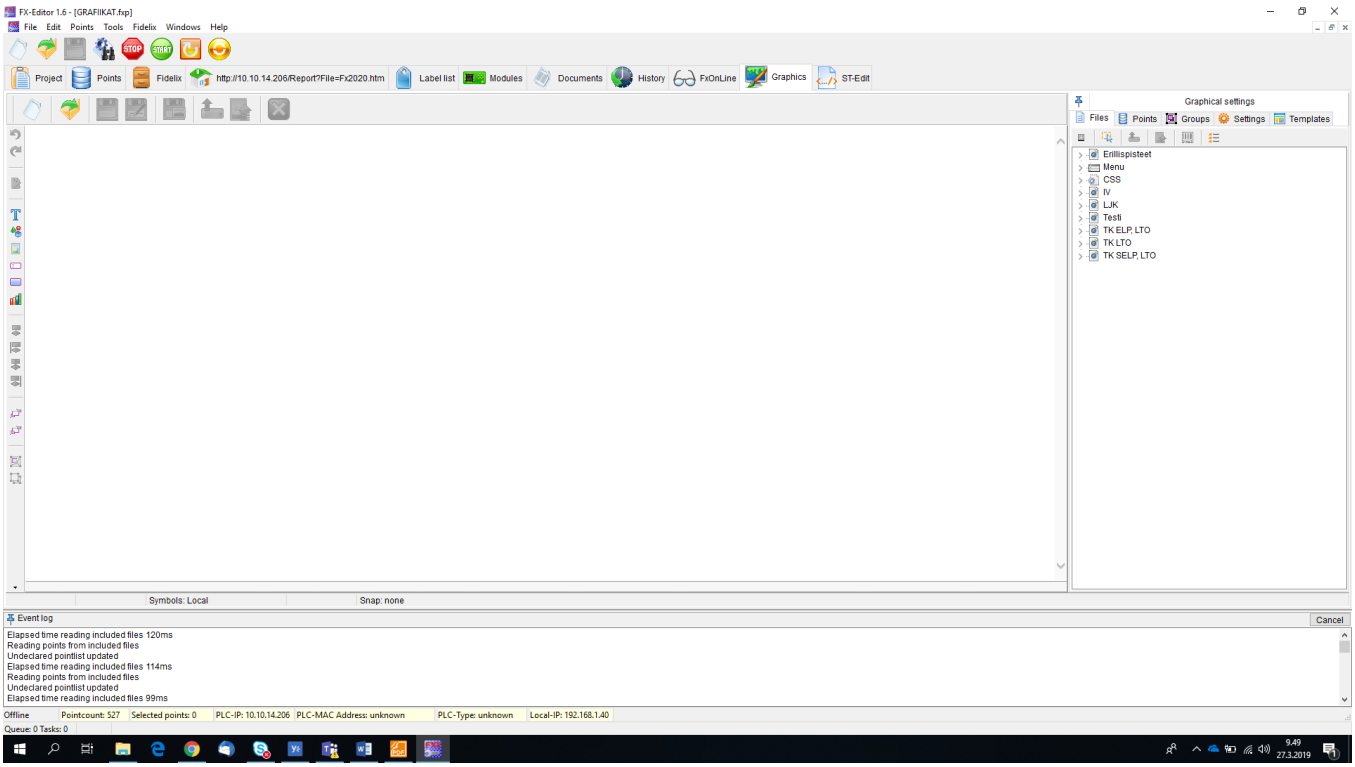
- Graphics-editor, grafiikkatyökalu
- OpenPCS, ohjelmointityökalu
- I/O-pisteiden kirjoitustyökalu
- FX-Connection, kytkentätöihin liittyvä työkalu.

Graphics-editor-sovellus on kehitetty Fidelixissä luomaan alakeskusten grafiikkakuvia. Se generoi grafiikat edelleen HTML-kieliseksi koodiksi tarpeen mukaan erilaiset pisteet, tekstit ja symbolit huomioon ottaen. [10.]

Grafiikkakuvat luodaan symboleista ja ohjelmapisteitä kuvaavista laatikoista sekä vapaasti kirjoitettavista teksteistä. Symbolit löytyvät symbolikirjastosta, ne ovat gif- tai png-muotoisia kuvia. Symboleita voidaan muokata kuvassa venyttämällä niitä. Täten esimerkiksi punaisesta neliöstä saadaan lämmitysputkea kuvaava paksu viiva. Symboleille voidaan antaa myös pistetunnus ja sen näkyvyys voidaan määritellä arvon perusteella. Symboleiden kopioiminen on sovelluksessa mahdollista. Kopioinnissa määritykset kopioituvat symbolin mukana. Symbolikirjastossa on kuvat tarvittavista elementeistä. Jos tarvittava symboli puuttuu, se voidaan piirtää piirtosovelluksella, tallentaa ja lisätä omaan symbolikirjastoon.

Pisteet luodaan grafiikalle vasemmalla laidalla olevalla painikkeella. Sovellus avaa pisteen luonnissa määritysikkunan, jossa määritetään pistetunnus, väri ja sijainti uudelle pisteelle. Pisteet voidaan kopioida symbolien tavoin, mutta niissäkin määritykset pysyvät samoina. Tekstit luodaan erillisestä napista samoin kuin pisteet. Tekstillekin voidaan liittää pistetunnus. Tekstille voidaan määritellä eri arvoilla eri tekstit. Täten saadaan teksti vaihtumaan pistetunnuksen vaihtaessa arvoaan. [10.]

Kuvassa 11 näkyy grafiikkakuvien piirtoon käytetty työkalu. Vasemmassa laidassa ovat toiminnot ja oikealla kuvan valinta sekä kuvan asetuksien muokkaus.



Kuva 11. Grafiikkakuvien luonti työkalu.

OpenPCS-sovelluksella voi ohjelmoida alakeskukset. FX-Editorin sisältämä OpenPCS-versio kykenee tuottamaan vain Structured Text -kielisiä ohjelmia, mutta FX-Editorista saa avattua myös Infoteamin OpenPCS-sovelluksen projektin määrittelyineen. Infoteamin sovelluksessa on mahdollista ohjelmoida kaikilla viidellä PLC ohjelmointiin luodulla kielellä. [10.]

I/O-pisteiden kirjoitukseen käytetyssä työkalussa on kaksi osaa. Toisessa luetaan grafiikassa ja ohjelmassa käytetyt pisteet. Pisteiden lukemisen jälkeen valitaan filtteri, jolla niille tehdään alkumuokkaukset. Luetut pisteet ajetaan filtterin läpi, jonka jälkeen niitä voi muokata vielä lisää. Lopuksi valitaan halutut pisteet ja ne kirjoitetaan alakeskuksen muistiin. [10.]

FX-Connection on työkalu, jolla voidaan luoda kytkentäluettelot, itselle luovutuslistat, laiteluettelot ja kaapeliluettelot. Sovelluksessa määritetään, mihin laitteisiin pisteet kytketään ja millä kaapelilla. Määritysten valmistuttua voidaan sovelluksella tulostaa kyseiset listat ja luettelot.

6.3 HTML-kieli

Grafiikkakuvien pohjalla toimii HTML-kieli. Lyhenne HTML tulee englanninkielen sanoista HyperText Markup Language. Se on hypertekstikieli, eli sillä voidaan kuvata hyperlinkkejä sisältävää tekstiä. Kielellä luodaan tekstimuotoinen ohjelma, jonka selainsovellus lukee ja tulostaa ohjelman määritykset graafiseen muotoon. Tätä graafista muotoa kutsutaan myös nimellä www-sivu. [11.]

HTML-kieli muodostuu peräkkäisistä ja sisäkkäisistä elementeistä. Elementit ovat kulmasulkein rajattujen tunnisteiden välinen alue tunnisteet mukaan luettuina. Elementin päättävä tunniste alkaa aina kenoviivalla. Esimerkkikoodista 3 voidaan nähdä, miltä elementit voivat näyttää. [11.]

```
<Tunniste> Elementin sisältö </Tunniste>
```

Esimerkkikoodi 3. Elementin ulkoasu.

HTML-kielellä luotavat tiedostot alkavat aina kulmasulkeilla, joissa kerrotaan ohjelmointikielen olevan HTML. Tiedoston viimeinen rivi käytetään vastaavasti kertomaan, että HTML -ohjelma loppuu.

Uusin versio on HTML5-kieli. Suuri muutos tässä kielessä edellisiin nähden on se, että tässä kieltä voidaan kuvata oliona. Tällöin kieltä voidaan käsitellä ohjelmallisesti. Elementtien hierarkia voidaan siten kuvata tekstimuotoisena. [11.]

Oliolla tarkoitetaan ohjelmoinnissa yksikköä, joka sisältää joukon toisiinsa liittäviä tietoja ja toimintoja. Tietoja kutsutaan ohjelmoinnissa nimellä attribuutti ja toimintoja nimellä metodi. Metodien avulla voidaan käsitellä attribuutteja, vaikka ne olisivat suojattuja olion ulkopuoliselta muokkaamiselta. Olioiden hyöty ohjelmoinnissa on se, että asioille voidaan helposti luoda vastine ohjelmassa. Tälle vastineelle voidaan antaa samat tiedot ja toiminnat, kuin asialla todellisuudessa on. [12.]

6.4 Ilmanvaihtokoneiden grafiikat

Ilmanvaihtokoneita on kolmen tyyppisellä lämmöntalteenottotavalla:

- Kiekko- LTO
- Kuutio- LTO
- Neste- LTO

Kiekko-LTO:ssa on kiekko, jossa on reikiä. Kiekko pyörii tulo- ja poistokanavienvälillä siirtäen näin lämmön poistokanavasta tulokanavaan. Kiekko-LTO:lla saavutetaan 70–80 %:n hyötysuhde, joka on LTO -tyypeistä paras. Tämän takia se on käytetyin tyyppi. Sen käyttö tietyissä paikoissa on kuitenkin kielletty. Sitä ei saa käyttää paikoissa, jossa poistoilmassa saattaa olla haitallisia hiukkasia ilmassa. Esimerkiksi WC:stä poistettavasta ilmasta ei saa ottaa lämpöä talteen kiekko- LTO:n avulla. [13, s.4.]

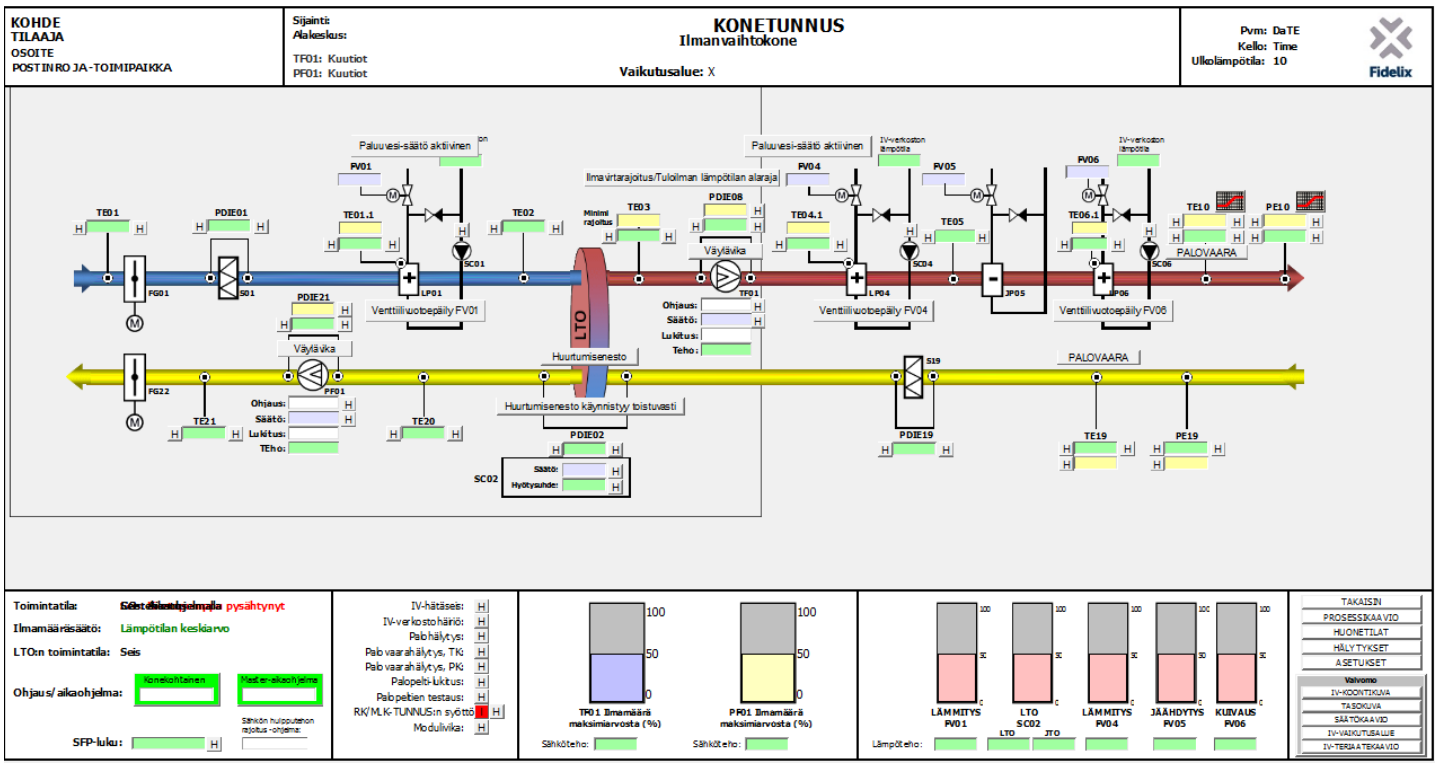
Kuutio-LTO:ta kutsutaan myös levy-LTO:ksi. Kuutio-LTO:ssa tulo- ja poistokanavat ohjataan LTO:n kohdalla ristiin. Tulo- ja poistoilmojen kanavat on ristiinmenokohdassa jaettu pieniin osiin ohuilla, usein alumiinisilla metallilevyillä, jotta lämmön siirtyminen olisi mahdollisimman tehokasta. Joka toisesta levyjen välistä kulkee tuloilma. Niistä väleistä kulkee poistoilma, joissa ei kulje tuloilma. Näin lämpö siirtyy metallilevyjen kautta poistoilmasta tuloilmaan. Kuutio-LTO:lla saadaan tyypeistä toisiksi parhaiten lämpö siirrettyä poistoilmasta tuloilmaan. Sen hyötysuhteet ovat 55–70 %. Tämän takia sitä käytetään paljon paikoissa, joissa kiekko- LTO:a ei saa käyttää. [13, s.5.]

Neste-LTO:ssa lämpö välittyy nesteen kautta, lämpö siirretään nesteeseen kanavissa olevien kennojen välityksellä. Nesteenä LTO:ssa käytetään yleensä glykolia sen hyvän lämpövaihtelukeston takia. Neste-LTO:n hyötysuhteet ovat 40–60 %, joka on LTO -tyypeistä heikoin. Se on kuitenkin helpoiten sovellettavissa oleva LTO -tyyppi. Sillä saadaan lämpöä kerättyä poistokanavasta, vaikkei se olisikaan tulokanavan vieressä. Neste-LTO:ssa nestettä kuljetetaan kennojen välissä putkella. Neste-LTO:lla on mahdollista kerätä lämpöä yhteen tulokoneeseen useammasta kuin yhdestä poistoputkesta. [13, s.6.]

Jokaisesta LTO-tyypistä tehtiin kuvat seuraavilla lämmitysvarusteluilla:

- esilämmitys-, LTO-, jälkilämmitys-, jäähdytys-, kuivauspatteri
- esilämmitys-, LTO-, jälkilämmitys-, jäähdytyspatteri
- esilämmitys-, LTO-, jälkilämmityspatteri
- LTO-, jälkilämmitys-, jäähdytys-, kuivauspatteri
- LTO-, jälkilämmitys-, jäähdytyspatteri
- LTO-, jälkilämmityspatteri

Kuvassa 12 on nähtävissä Ilmanvaihtokoneen grafiikkakuva, jossa LTO -tyyppinä on kiekko.



Kuva12. Ilmanvaihtokoneen grafiikka kehyksineen.

Lämmityspattereista tehtiin grafiikkakuvat nestelämmityksellä ja sähkölämmityksellä. Täten grafiikkakuvia tuli jokaisella LTO-tyypillä edellä mainittu lista kaksinkertaisena eli kolmekymmentäkuusi grafiikkakuvaa.

6.5 Lämmitysjärjestelmän grafiikat

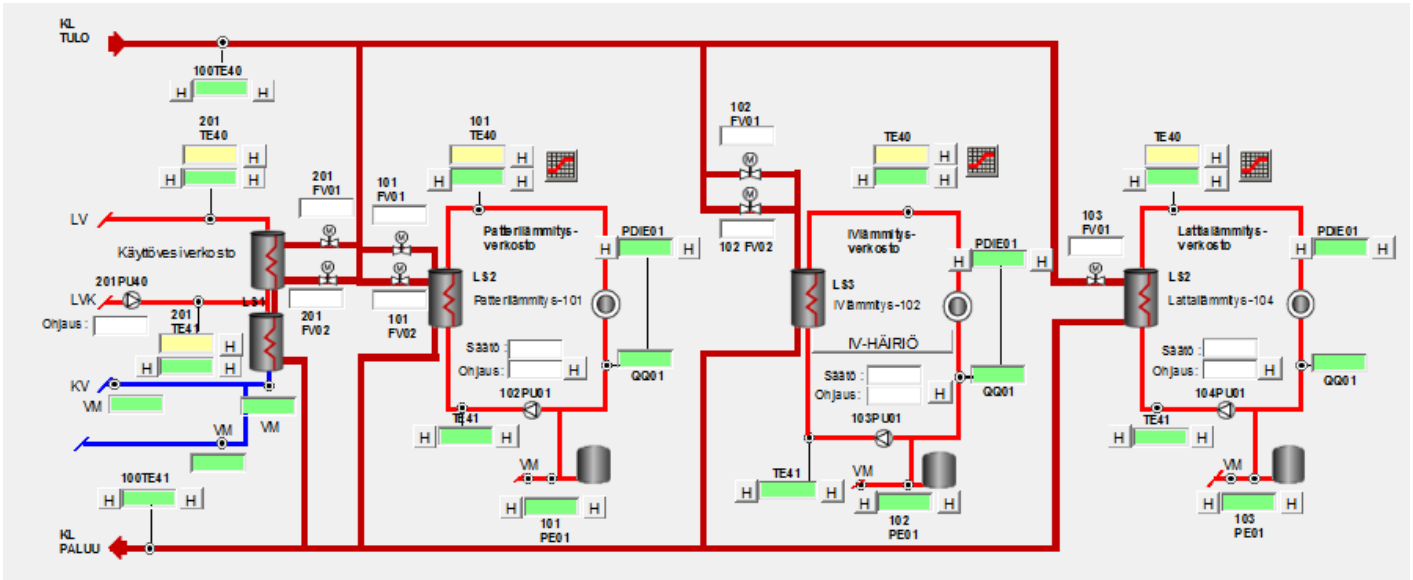
Lämmitysjärjestelmän grafiikkakuvia työssä tehtiin vain muutama, koska ne ovat joka kohteessa lähes samanlaisia. Isommissa kohteissa lämmityspiirejä on enemmän, mutta piirit ovat keskenään samanlaisia.

3- ja 4-piiristen lämmönjakokeskusten grafiikkakuvista tehtiin versiot, joissa hälytysten määritykset tehtiin samalla sivulla. Asetussivusta tehtiin myös versio, jossa hälytykset määritettiin.

Lämmitysjärjestelmästä tehtiin seuraavat grafiikkakuvat:

- 3- piirinen lämmönjakokeskus
- 4- piirinen lämmönjakokeskus
- 6- piirinen lämmönjakokeskus
- lämmönjakokeskuksen asetussivu
- lämmönjakokeskuksen hälytyssivu

Kuvassa 13 nähdään 4-piirinen lämmönjakokeskus.



Kuva 13. Lämmönjakokeskuksen grafiikka ilman kehyksiä.

6.6 Erillispisteiden grafiikka

Erillispisteillä tarkoittetaan pisteitä, jotka eivät liity mihinkään isompaan kokonaisuuteen ja täten niiden ohjaukset ovat muista prosesseista erillisiä. Tyypillisiä erillispisteitä ovat erilliset hälytykset, valojen ohjaukset, sulatukset ja sähköpisteiden ohjaukset.

Erillispisteisiin tehtiin vain yksi kuva. Koska erillispisteiden kuvat ovat hyvin erilaisia kohteittain, ei ollut järkevää yrittää tehdä niistä eri varaatioita. Erillispistegrafiikasta ei yritetty tehdä sellaisenaan kohteisiin sopivaa, vaan siitä pyrittiin tekemään pienellä työllä mahdollisimman moneen kohteeseen sopiva.

Erillispisteiden grafiikkaan pyrittiin tuomaan yleisesti erillispistekuvissa esiintyviä pisteitä ja laitteita.

Kuvassa 14 nähdään erillispisteistä tehty grafiikkakuva.

Kuva 14. Erillispisteiden grafiikkakuva ilman kehyksiä.

Erillispuhaltimet ja kiertoilmakoneet sekä itseohjautuvat IV-koneet tuodaan pienissä kohteissa osaksi erillispistekuvaa. Isoissa kohteissa niitä on paljon. Tällöin niille tehdään yksi tai useampi kuva kohteen tarpeen mukaan.

7 Yhteenveto

Opinnäytetyössä kehiteltiin projektintoteutuksen helpottamiseksi kirjastoja yleisistä projekteissa esiintyvistä ohjelmista sekä laitteiden grafiikoista. Tavoitteena oli kirjastojen avulla nopeuttaa projektinhoitajan projektikohtaista työtä, joka mahdollistaa yhdelle henkilölle useamman projektin samanaikaisen hallinnan.

Sovellusvalinnat kirjastojen luontiin onnistuivat. Kirjastoista saatiin odotusten mukaiset työkalut projektinhoitajien avuksi. Kirjastojen toiminnasta ja niiden kyvystä nopeuttaa projektin toteutusta saatiin näyttöä.

Kirjastot luotiin tämänhetkisen tarpeen mukaan. Tarpeet muuttuvat ohjelmien osalta, kun automaatio yleistyy. Tällöin tarvitaan uusien osa-alueiden ohjaukseen uusia ohjelmia. Grafiikoiden osalta tarpeet muuttuvat, kun laitteet päivittyvät. Isommalle näytölle tullaan tarvitsemaan uudet kuvat, joihin voidaan lisätä yksityiskohtia.

Tuloksena saatiin projektien toteuttamista projektin hoitajan osalta helpottavat kirjastot, joissa ovat yleisesti käytetyt grafiikat ja ohjelmat.

Työlle asetetut tavoitteet saavutettiin. Rajaus tapahtui työn aloituksen yhteydessä. Rajaus tehtiin opinnäytetyön ohjaajan kanssa. Rajauksessa käytiin läpi ohjelmat, jotka ohjelmakirjastoon tulevat. Ohjelmia lisättiin kuitenkin työn aikana, joten rajaus ei ollut aivan pitävä. Grafiikkakirjaston osalta rajauksessa päätettiin, mistä laitteista grafiikkakuvat piirretään.

Opinnäyteyössä tutkittiin myös alakeskuksen ohjelmointia ja pohdittiin sen ohjelmointiin käytettävien kielien hyviä ja huonoja puolia.

Grafiikkakirjaston osalta tutkittiin HTML-kielen toimintaa ja sovelluksen tapaa muuttaa graafisesti luodut tiedostot HTML-tiedostoiksi sekä kielen uusimpaan versioon siirtymisen hyötyjä. Siirtymisen hyötynä havaittiin mahdollisuus käsitellä HTML-kielen rakennetta oliona.

Ohjelmakirjaston tekeminen oli opinnäytetyön merkittävin yksittäinen osuus. Ohjelmakirjaston tekemiseen kului aikaa noin neljä viikkoa. Neljän viikon aikana siihen käytettiin noin sata tuntia, joista ilmanvaihtokoneiden ohjelmien tekemiseen aikaa kului noin seitsemänkymmentä tuntia.

Ohjelmakirjasto oli myös opinnäytetyön osuuksista haastavin. Haastavuus muodostui eri variaatioiden toimintakuntoon saattamisesta.

Toinen osa oli grafiikkakirjasto, jossa tehtiin laaja kirjasto ilmanvaihtokoneiden grafiikoista ja muutama variaatio eri varustelutason lämmitysjärjestelmistä. Kolmas osa oli loppuraportin kirjoittaminen, johon sisältyi työn teoreettinen osa. Teoreettisessa osassa etsittiin sekä ohjelmoinnin teoriaan että ohjelmoitavan laitteen toimintaan liittyvää tietoa.

Ohjelmakirjaston tavoite oli nopeuttaa projektinhoitajien ala-asemien ohjelmoimiseksi tekemää työtä. Yrityksen henkilöstö käyttää tuhansia tunteja vuodessa ohjelmien tekemiseen. Ohjelmointityöajasta suuri osa on samojen ohjelmien tekemistä tai niiden etsimistä aikaisemmin toteutettujen projektien aineistosta. Ohjelmakirjaston avulla tätä aikaa on mahdollista pienentää, jolloin projektiin kuluvat tunnit vähenevät vastaavasti ja ne voidaan käyttää toisiin työtehtäviin.

Aikaisemmin grafiikoiden tyylit ovat vaihdelleet paljon projektien kesken. Grafiikkakirjaston tavoite oli luoda yhtenäinen kirjasto, jossa grafiikoiden tyyli on kaikkialla sama. Samalla saadaan helposti muokattavat pohjat eri varustetasoisista ilmanvaihtokoneista, lämmönjakokeskuksista, sähköpisteistä ja erillisistä mittauspisteistä. Suunnitelmat ovat yleensä kohteissa hieman erilaiset. On siis vaikeaa tehdä grafiikkakirjastoa, joka kattaisi kaikki mahdolliset variaatiot eri järjestelmistä.

Grafiikkakirjasto on nyt erityisen ajankohtainen, sillä Fidelix Oy lanseerasi vuonna 2017 uuden näytön. Uuteen näyttöön ei ole tehty kaikkia yleisesti käytettyjä grafiikoita.

Lähteet

1 Fidelix Oy. 2019. Verkkoaineisto. Fidelix Oy <https://www.fidelix.fi/fidelix/>. Luettu 26.2.2019.

2 Fidelix Oy. 2019. Verkkoaineisto. Fonecta <https://www.finder.fi/Rakennusautomaatio/Fidelix+Oy/Vantaa/yhteystiedot/429903>. Luettu 26.2.2019.

3 Tanja, Peltoniemi. 2019. HR- ja markkinointipäällikkö. Fidelix Oy. Vantaa. Puhelinkeskustelu. 1.3.2019.

4 Ohjelmointi. 2018. Verkkoaineisto. Wikipedia< https://fi.wikipedia.org/wiki/Ohjelmointi>. Luettu 5.3.2019.

5 Introduction into IEC 61131-3 Programming Languages. 2019. Verkkoaineisto. PLCopen <http://www.plcopen.org/pages/tc1_standards/iec_61131_3/>. Luettu 27.2.2019.

6 International standard IEC 61131-3. 2003. Standardi. IEC <http://d1.amobbs.com/bbs_upload782111/files_31/ourdev_569653.pdf>. Luettu 27.2.2019.

7 Infoteam. 2008. OpenPCS User Manual. Bubenreuth Germany.

8 FX-3000-C. 2019. Verkkodokumentti. Fidelix Oy < https://www.fidelix.fi/wp-content/uploads/FX-3000-C_FI.pdf>. Luettu 8.3.2019.

9 VISIO-15-C. 2019. Verkkodokumentti. Fidelix Oy < https://www.fidelix.fi/wp-content/uploads/VISIO-15-C_FI.pdf>. Luettu 8.3.2019.

10 Fidelix FX-Editor. 2018. Käyttöopas. Fidelix Oy. Luettu 19.3.2019.

11 HTML. 2018. Verkkoaineisto. Wikipedia.< https://fi.wikipedia.org/wiki/HTML>. Luettu 13.3.2019.

12 Olio. 2018. Verkkoaineisto. Wikipedia.<https://fi.wikipedia.org/wiki/Olio_(ohjelmointi)>. Luettu 27.3.2019.

13 Ilmanvaihdon lämmöntalteenotto. 2017. Opinnäytetyö. Theseus. <https://www.theseus.fi/bitstream/handle/10024/127145/Bragge_Mikael.pdf?sequence=4&isAllowed=y>. Luettu 27.3.2019.

Esimerkkiohjelma

C:\Users\Joonas.Vaananen\Documents\Opinnäytetyö\Ohjelma- ja grafiikkakirjasto\Ohjelmakirjastot\Ohjelmat\FBt\lVmaanantai 18. maaliskuuta 2019 12.30

```
FUNCTION_BLOCK FB_Huurteenesto
VAR_EXTERNAL
END_VAR

VAR_INPUT
IN_Poistopuhallin_TT    : string(30);
IN_Huurteenesto_TT      : string(30);
IN_Poistoilmamaara      : string(30);
IN_OnRaja_Mtaulukko     : string(30);
IN_LTO_paine_ero        : string(30);
IN_Huurtumisraja        : string(30);
IN_Huurtumisraja_lim    : int;
IN_HP_hystereesi        : real;
IN_HP_paalle_viive      : string(30);
IN_HP_pois_viive        : string(30);
IN_Jateilma_LT          : string(30);
IN_JateIlmanRaja        : real;
IN_Halyraja             : string(30);
IN_Halytyspiste         : string(30);
END_VAR

VAR_OUTPUT

END_VAR

VAR
i_laskuri, i : int;
Systemtime : systemtimefb;
r_Huurteenpoisto_on : real;
i_OLD_MIN : int;
a_Halyhistoria : array[0..60] of int;
Huurteepoisto_Trig : r_trig;
r_Poistoilmamaara : real;
r_on_raja : real;
i_status_Mtaulukko : int;
i_HuurteenestoTilatieto : int;
i_PoistoP_TT : int;
r_LTO_PE : real;
r_TE21_M : real;
d_huurteenesto_paalle_viive : dint;
d_huurteenesto_pois_viive : dint;
paalle_ton, pois_ton : ton;
b_huurteenesto_paalle, b_ekaKerta : bool;
i_tulos : int;
END_VAR
(*
------------------------------------------------------------------------------
                                HUURTEENESTO V1.0
------------------------------------------------------------------------------
LTO-poistopatterin paine-eron noustessa huurtumisrajaan ja poistoilmalämpötilan
ollessa alle LTO:n jäätymisrajan, muuttaa blokki asetetun aloitusviiveen
jälkeen huurteeneston tilatiedon tilaan 1. Huurteenesto palaa
tilaan 0 kun paine-ero on laskenut eroalueen verran, aikaisintaan
```

```
poismenoviiveen kuluttua.
-------------------------------------------------------------------------------
                              Blokin sisääntulot
-------------------------------------------------------------------------------
    IN_Poistopuhallin_TT    =   Poistopuhaltimen tilatieto
    IN_Huurteenesto_TT      =   Huurteeneston tilatieto
    IN_Poistoilmamaara      =   Poistopuhaltimen ilmamäärä(mikäli taulukko
    käytössä)
    IN_OnRaja_Mtaulukko     =   Käynnistymisrajan muunnostaulukko (optio)
    IN_LTO_paine_ero        =   Paine-ero LTO:n yli
    IN_Huurtumisraja        =   Tunnus minkä limiteistä Huurteenpoiston PE raja
    löytyy
    IN_Huurtumisraja_lim    =   Limitin numero, jos nro. = 0, niin raja-arvo on
    tunnuksen arvo
    IN_HP_hystereesi        =   Huurteenpoiston hystereesi (real muuttuja)
    IN_HP_paalle_viive      =   Huurteenpoiston päällemeno viive (minuutteina)
    IN_HP_pois_viive        =   Huurteenpoiston sammutusviive (minuutteina)
    IN_Jateilma_LT          =   Jäteilman lämpötila
    IN_JateIlmanRaja        =   Jäteilman lämpötilan raja-arvo (real muuttuja)

    IN_Halyraja             =   Kuinka monta kertaa huurteenpoisto saa käynnistyä
                                tunnin sisällä, ettei tule hälytystä(optio)
    IN_Halytyspiste         =   em. hälytysrajan hälytyspiste
-------------------------------------------------------------------------------
                              Käyttöesimerkki
-------------------------------------------------------------------------------
VAR
    blokinNimi : FB_Huurteenesto;
END_VAR
-------------------------------------------------------------------------------
    blokinNimi.IN_Poistopuhallin_TT     := 'TUNNUS_SC21_FI';
    blokinNimi.IN_Huurteenesto_TT       := 'TUNNUS_HUURTEENESTO_FI';
    blokinNimi.IN_Poistoilmamaara       := 'TUNNUS_PDIE21_M';
    blokinNimi.IN_OnRaja_Mtaulukko      := 'TUNNUS_PDIE02_L';
    blokinNimi.IN_LTO_paine_ero         := 'TUNNUS_PDIE02_M';
    blokinNimi.IN_Huurtumisraja         := 'TUNNUS_PDIE02_M';
    blokinNimi.IN_Huurtumisraja_lim     := 5;
    blokinNimi.IN_HP_hystereesi         := getLimitF(limitnumber:=6, name:=
    'TUNNUS_PDIE02_M');
    blokinNimi.IN_HP_paalle_viive       := 'TUNNUS_LTO_HPOISTO_ON_VIIVE_FM';
    blokinNimi.IN_HP_pois_viive         := 'TUNNUS_LTO_HPOISTO_OFF_VIIVE_FM';
    blokinNimi.IN_Jateilma_LT           := 'TUNNUS_TE21_M';
    blokinNimi.IN_JateIlmanRaja         := getLimitF(limitnumber:= 1, name:=
    'TUNNUS_TE02_M');
    blokinNimi.IN_Halyraja              := 'TUNNUS_LTO_HPOISTO_HRAJA_FI';
    blokinNimi.IN_Halytyspiste          := 'TUNNUS_LTO_HPOISTO_FH';
    blokinNimi();
-------------------------------------------------------------------------------
    testattu 6.2.2018
*)

(*=============================================================================*)
(*                  Pistetunnuksien lukeminen muuttujiin                       *)
```

C:\Users\Joonas.Vaananen\Documents\Opinnäytetyö\Ohjelma- ja grafiikkakirjasto\Ohjelmakirjastot\Ohjelmat\FBt\Ivmaanantai 18. maaliskuuta 2019 12.30

```
(*================================================================================*)
i_PoistoP_TT := Getdigitalpointf(name := IN_Poistopuhallin_TT);
r_LTO_PE := Getanalogpointf(name := IN_LTO_paine_ero);
i_status_Mtaulukko := GetpointerrorF(name :=IN_OnRaja_Mtaulukko);
r_Poistoilmamaara := Getanalogpointf(name :=IN_Poistoilmamaara);
r_TE21_M := Getanalogpointf(name := IN_Jateilma_LT);
d_huurteenesto_paalle_viive := real_to_dint( Getanalogpointf(name :=
IN_HP_paalle_viive));
d_huurteenesto_pois_viive := real_to_dint( Getanalogpointf(name :=
IN_HP_pois_viive));

SYSTEMTIME();

(*================================================================================*)
(*                  Huurteenpoiston raja-arvon asettaminen                        *)
(*================================================================================*)
if i_status_Mtaulukko > -1 then
    r_on_raja := GetConversiontablef(value := r_Poistoilmamaara, name :=
    IN_OnRaja_Mtaulukko);
    if IN_Huurtumisraja_lim > 0 then
        i_tulos := SetLimitF(limitnumber := IN_Huurtumisraja_lim, value :=
        r_on_raja, lockstate := 1, name :=IN_Huurtumisraja);
    else
        i_tulos := setAnalogPointf(value := r_on_raja, lockstate := 1, name:=
        IN_Huurtumisraja);
    end_if;
end_if;

if IN_Huurtumisraja_lim > 0 then
    r_Huurteenpoisto_on := Getlimitf(limitnumber := IN_Huurtumisraja_lim, name :=
    IN_Huurtumisraja);
else
    r_Huurteenpoisto_on := getAnalogPointf(name:= IN_Huurtumisraja);
end_if;


(*================================================================================*)
(*                  Tarkistetaan huurteenpoiston tarve                            *)
(*================================================================================*)
if r_LTO_PE < r_Huurteenpoisto_on - IN_HP_hystereesi or i_PoistoP_TT = 0 then
    b_huurteenesto_paalle := false;
elsif r_LTO_PE > r_Huurteenpoisto_on and r_TE21_M < IN_JateIlmanRaja and
i_PoistoP_TT = 1 then
     b_huurteenesto_paalle := true;
end_if;

(*================================================================================*)
(*              Asetetaan huurteeneston tilatieto päälle/pois                     *)
(*================================================================================*)
if pois_ton.Q and b_huurteenesto_paalle = false then
    i_HuurteenestoTilatieto := 0;
    b_ekaKerta := false;
elsif paalle_ton.Q then
```

C:\Users\Joonas.Vaananen\Documents\Opinnäytetyö\Ohjelma- ja grafiikkakirjasto\Ohjelmakirjastot\Ohjelmat\FBt\Vmaanantai 18. maaliskuuta 2019 12.30

```
    i_HuurteenestoTilatieto := 1;
    if b_ekaKerta = false then
        a_Halyhistoria[SYSTEMTIME.MINUTE]:= a_Halyhistoria[SYSTEMTIME.MINUTE] + 1;
        b_ekaKerta := true;
    end_if;
end_if;

(*=============================================================================*)
(*                     Timerit päälle/pois ohjaukseen                          *)
(*=============================================================================*)
paalle_ton.in := b_huurteenesto_paalle;
paalle_ton.pt := dint_to_time( d_huurteenesto_paalle_viive * 1000 * 60);
paalle_ton();
pois_ton.in := (i_HuurteenestoTilatieto = 1);
pois_ton.pt := dint_to_time( d_huurteenesto_pois_viive * 1000 * 60);
pois_ton();

(*=============================================================================*)
(*              Huurteenesto liian usein päällä -Hälytys                       *)
(*=============================================================================*)

if i_OLD_MIN <> SYSTEMTIME.MINUTE then
    a_Halyhistoria[SYSTEMTIME.MINUTE] := 0;
    i_OLD_MIN := SYSTEMTIME.MINUTE;
end_if;

i_laskuri := 0;
for i := 0 to 59 do  (*Päällemenokertojen laskeminen *)
    if a_Halyhistoria[i] > 0 then
        i_laskuri := i_laskuri + a_Halyhistoria[i];
    end_if;
end_for;

if i_laskuri > getDigitalPointF(Name:=IN_Halyraja) then
    i_tulos := setDigitalPointF(Value:=1, Lockstate:=1, Name:=IN_Halytyspiste);
else
    i_tulos := setDigitalPointF(Value:=0, Lockstate:=1, Name:=IN_Halytyspiste);
end_if;

i_tulos := Setdigitalpointf(value := i_HuurteenestoTilatieto, lockstate := 1,
name :=IN_Huurteenesto_TT);
END_FUNCTION_BLOCK
```

Esimerkkigrafiikat