TAMPEREEN
AMMATTIKORKEAKOULU

# KOMPRESSORIYKSIKÖN SOVELLUSSUUNNITTELU JA OHJELMOINTI

Jussi Isotalo

Opinnäytetyö
Huhtikuu 2016
Sähkötekniikan koulutusohjelma
Automaatiotekniikka

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Sähkötekniikan koulutusohjelma
Automaatiotekniikka

ISOTALO, JUSSI:
Kompressoriyksikön sovellussuunnittelu ja ohjelmointi

Opinnäytetyö 77 sivua
Huhtikuu 2016

---

Tässä opinnäytetyössä suunniteltiin ja toteutettiin teollisuudessa käytettävälle kompressoriyksikölle automaatio-ohjelmisto sekä käyttöliittymä. Työn tarkoitus oli perehtyä ohjelmoitavien logiikoiden ohjelmointikielet määrittävään IEC 61131-3 -standardiin sekä sitä hyödyntäviin järjestelmiin. Erityisesti tutustuttiin standardin vuonna 2013 julkaistun päivityksen sisältämiin olio-ohjelmoinnista tuttuihin ominaisuuksiin, kuten luokkiin ja niiden periytymiseen. Lisäksi työn tarkoitus oli tutustua käyttöliittymän ohjelmointiin C#-ohjelmointikieltä käyttäen sekä teolliseen internetiin ja sen käyttöön työn kohteena olleessa kompressoriyksikössä.

Opinnäytetyössä käydään läpi yleistä tietoa IEC 61131-3 -standardista sekä standardin sisältämät ohjelmointikielet, ohjelmointikielien elementit ja niiden ominaisuudet. Elementeistä esitellään ohjelmien rakenneosat sekä tietotyypit ja tietorakenteet, jotka ovat erityisen tärkeitä ohjelmoitaessa IEC 61131-3 -standardia tukevassa ympäristössä. Lisäksi työssä tutustutaan olio-ohjelmoinnin perusperiaatteisiin ja mahdollisuuksiin.

Kompressoriyksikköön suunniteltiin ja ohjelmoitiin automaatio-ohjelmisto TwinCAT 3 -ympäristössä. Ohjelmistosta tehtiin versio asiakasyrityksen tutkimus- ja kehityslaboratorioon ja siihen suunniteltiin useita lisäominaisuuksia kehitystyötä ajatellen. Lisäominaisuuksia olivat muun muassa nopeasti käyttöönotettavat mittaukset sekä logiikkaohjelman muuttujien pakko-ohjaukset. Käyttöliittymä suunniteltiin ja ohjelmoitiin toimimaan sekä kompressoriyksikön että laboratorion kanssa. Käyttöliittymään tehtiin myös lisäominaisuuksia varten omia toimintoja.

Työssä kehitetty laboratorion automaatio-ohjelmisto ja käyttöliittymä ovat olleet jatkuvassa käytössä tutkimus- ja kehitystyössä. Toteutettujen lisäominaisuuksien avulla kehitystyö on helpottunut ja nopeutunut. Myös ohjelmiston kehityksen kohteena ollut kompressoriyksikkö on ollut testikäytössä teollisuudessa hyvin tuloksin. Testauksessa esiin tulleita parannus- ja kehitysehdotuksia tullaan lisäämään ohjelmistoon myös jatkossa.

---

Asiasanat: ohjelmoitava logiikka, iec 61131-3, oliokeskeisyys, teollinen internet, twincat, c#

**ABSTRACT**

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Electrical Engineering
Option of Automation Engineering

ISOTALO, JUSSI:
Software design and programming of a compressor unit

Bachelor's thesis 77 pages
April 2016

---

The subject of this thesis was to design and program automation software and a user interface for an industrial compressor unit. The purpose was to become familiar with IEC 61131-3 standard and automation programming environments. In particular the aim of this study was to explore object-oriented programming elements that were added to the standard in 2013. Additionally, the purpose was also to study programming of a user interface using C# programming language and also to abreast with concept of industrial internet and how to adapt it to the compressor system.

Information about IEC 61131-3 standard and its programming languages, common elements and their properties are presented in the study. The standard introduces program organization units, data types and structures that are specially focused since they play an important role in programming in IEC 61131-3 environment. The study also introduces the basics and opportunities of object-oriented programming methods including new features of the IEC 61131-3 standard.

Automation software of the compressor unit was designed and programmed using TwinCAT 3 environment. The software was adapted for the customer's research and development laboratory. The development of the laboratory version of the software included programming special and unique features for research purpose. Laboratory features included adjustable temporary measurements and an ability to force system variables. A user interface was designed and programmed to work with both the compressor unit and the laboratory system. Special laboratory features were developed for the user interface.

The developed laboratory's automation software and user interface have been used in customer's research and development with success. By using the specially designed features the customer has been able to develop its product with ease. In addition, the compressor unit that was designed and programmed in this study has been under testing in industrial environment with positive results. Improvements and ideas arising during testing will be implemented in the system in the future including other new planned features.

---

Key words: programmable logic, iec 61131-3, object orientation, industrial internet, twincat, c#

**SISÄLLYS**

**LYHENTEET JA TERMIT**

| | |
|---|---|
| ADS | Automation Device Specification, kommunikointiprotokolla |
| Assembler | Symbolinen konekieli, ensimmäisiä ohjelmointikieliä |
| Big Data | Käsite teollisen internetin aiheuttamalle suurelle tietomäärälle |
| C# | Microsoftin kehittämä luokkakeskeinen ohjelmointikieli |
| DIN | Deutsches Institut für Normung, yleinen keskieurooppalainen standardi |
| FBD | Function Block Diagram, lohkokaavioihin perustuva ohjelmointikieli |
| H | High, mittauksen alempi yläraja, yleensä varoitus |
| HH | High-High, mittauksen ylärajat, yleensä hälytys |
| HMI | Human-Machine Interface, käyttöliittymä |
| HTTP | Hypertext Transfer Protocol, kommunikointiprotokolla |
| I/O | Input/Output, siirräntä eli sisään- ja ulostulot |
| IEC | International Electrotechnical Commission, kansainvälinen sähköalan standardointiorganisaatio |
| IL | Instruction List, käskylistä, komentoihin perustuva ohjelmointikieli |
| IoT | Internet of Things, esineiden internet |
| JSON | JavaScript Object Notation, avoimen standardin tiedostomuoto |
| L | Low, mittauksen ylempi alaraja, yleensä varoitus |
| LD | Ladder Diagram, tikapuukaavioihin perustuva ohjelmointikieli |
| LINQ | Language Integrated Query, datakyselyiden mahdollistama komponentti .NET-ohjelmointiympäristössä |
| LL | Low-Low, mittauksen alempi alaraja, yleensä hälytys |
| Luokka | Olio-ohjelmoinnin käsite, määrittelee olion ominaisuudet ja menetelmät |

| | |
|---|---|
| Olio | Olio-ohjelmoinnin käsite, luokan perusteella tehty ilmentymä, joka toteuttaa luokan käytännössä |
| PC | Personal Computer, yleisluontoinen nimitys tietokoneelle |
| PLC | Programmable Logic Controller, ohjelmoitava logiikka |
| POU | Program Organization Unit, ohjelmien rakenneyksikkö IEC 61131-3 -standardissa |
| Pragma | Ohjelmakoodin kääntäjälle annettava komento tai viesti |
| SFC | Sequential Function Chart, sekvenssikaaviotyyppinen ohjelmointitapa |
| SIL | Safety Integrity Level, turvallisuus- ja eheystaso |
| ST | Structured Text, tekstipohjainen korkeamman tason ohjelmointikieli |
| Task | Ohjelmoitavassa logiikassa suoritettava tehtävä IEC 61131-3 -standardissa |
| TCP/IP | Transmission Control Protocol / Internet Protocol, Internetiin liittyvässä kommunikoinnissa käytetty yleinen protokolla |
| Teollinen internet | Käsite, jolla tarkoitetaan teollisten laitteistojen, kuten automaatiojärjestelmien, liittymistä internetiin |
| VFD | Variable-Frequency Drive, taajuusmuuttaja |
| XML | Extensible Markup Language, tiedonsiirrossa yleisesti käytetty merkintäkieli |

# 1 JOHDANTO

Tässä työssä kehitettiin asiakasprojektina teollisuuteen suunnattuun kompressoriyksikköön automaatio-ohjelmisto sekä käyttöliittymä. Kompressorin ohjauksen ohjelmistosta suunniteltiin lisäksi oma lisäominaisuuksia sisältävä versio tutkimus- ja kehitystyössä käytettävään laboratorioon. Ohjelmointi toteutettiin nykyaikaisella TwinCAT 3 -järjestelmällä sekä Beckhoffin ohjelmoitavina logiikkoina toimivilla teollisuus-PC:illä. Työn tilaaja oli tamperelainen insinööritoimisto InSolution Oy.

Ohjelmoitavien logiikoiden ohjelmointikielet ja -tekniikat kuvataan IEC:n standardissa 61131-3. Standardi on ollut olemassa jo yli 20 vuotta, mutta vasta viime vuosina sen asema on parantunut ja tuki laajentunut yhä useampaan valmistajaan. Saksalaiset CODESYS sekä Beckhoff ovat kasvattaneet jatkuvasti suosiotaan IEC 61131-3 -pohjaisina alustoina tehden samalla standardin tunnetuksi. Vuonna 2013 julkaistussa standardin päivityksessä esiteltiin logiikkaohjelmointiin olio-ohjelmoinnista tuttuja piirteitä, joiden ansiosta logiikkaohjelmointi on lähempänä nykyaikaista ohjelmistokehitystä kuin koskaan aiemmin. Olio-ohjelmoinnin mukanaan tuomat ominaisuudet vievät logiikkaohjelmoinnin uudelle tasolle mahdollistaen yhä monimutkaisempien järjestelmien hallinnan.

Paineilma on tärkeässä osassa missä tahansa teollisuuden alassa mahdollistaen instrumenttien sekä laitteiden toiminnan. Teollisuuden paineilman tuotannossa käytetyt kompressoriyksiköt vaativat lukuisia mittauksia toimiakseen oikein vaihtelevissa teollisuusympäristöissä taaten näin jatkuvan tuotannon. Työssä ohjelmoinnin kohteena oleva kompressoriyksikkö sisältää kymmeniä mittauksia ja useita säätöjä, joiden käsittelyä helpottaa huomattavasti IEC 61131-3 -standardin mahdollistamat monimutkaiset tietorakenteet sekä toimilohkot.

Työn lopussa tutustutaan myös 2010-luvulla pinnalle nousseen teollisuuden internetin tuomiin mahdollisuuksiin teollisuuden järjestelmissä. Kompressoriyksikkö liitetään teolliseen internetiin Inspector-tiedonkeruujärjestelmän avulla, jonka ansiosta kompressorin mitattavia suureita sekä käyttöä voidaan seurata mistä tahansa tavallisen internetselaimen avulla. Teollisen internetin kasvattaessa suosiotaan yhä useampi järjestelmä on kytketty internetiin mahdollistaen muun muassa etävalvonnan sekä järjestelmän toiminnan jatkuvan analysoinnin.

# 2 IEC 61131-3 -STANDARDI

Ohjelmoitavia logiikoita on käytetty teollisuudessa jo vuosikymmeniä. Nykyaikaisten logiikoiden kehitys alkoi 1970-luvulla mikrokontrollerien yleistymisen ansiosta. Ensimmäiset logiikat ohjelmoitiin käyttäen ainoastaan tikapuukaavioita, eli ladder-ohjelmointia, tai assembler-pohjaista yksinkertaista komentolistausta. Tikapuukaavio perustui relekaavioihin, joita käytettiin yleisesti sähköpiirustuksissa piirien toiminnan kuvaamiseen. Tämän ansiosta sähkökuvia lukemaan kykenevät henkilöt pystyivät ymmärtämään logiikkaohjelmien toiminnan sekä kehittämään niitä ilman tietoteknistä osaamista. (Hansen 2015, 4.)

Järjestelmiä kehittivät useat eri yritykset kukin omalla tavallaan ja omia tekniikoita hyödyntäen. Tämä johti siihen, että jokaisella logiikkavalmistajalla oli omat ohjelmointikielensä, kehitysympäristönsä sekä toimintaperiaatteensa ja jopa ohjelmointilaitteensa. Jos käytössä oli kahden eri valmistajan logiikat, täytyi niille olla myös omat ohjelmistot, kaapelit, laitteet sekä ohjelmointitietämys. Ohjelmakoodia ei voinut siirtää eri valmistajien laitteiden välillä vaan logiikkaohjelmat täytyi kirjoittaa uudelleen toisen valmistajan kehitysympäristöä käyttämällä. Valmistajien lukumäärän kasvaessa ja laitteiden lisääntyessä alkoi ohjelmointikielien ja -ympäristöjen kirjo olla niin suuri, että yhteisen standardin suunnittelu oli väistämätöntä. (Hansen 2015, 6–7.)

## 2.1 Standardin julkaisu

Vuonna 1979 International Electrotechnical Commission (IEC) perusti kansainvälisen logiikka-alan ammattilaisista koostuneen työryhmän, joka sai tehtäväksi suunnitella ohjelmoitaville logiikoille oman standardin. 1982 ilmestyi ensimmäinen ehdotus, jonka perusteella todettiin, ettei standardia voitu tiivistää järkevällä tavalla yhdeksi dokumentiksi. Standardin sisältö ja sitä kehittävä työryhmä jaettiin viiteen eri osaan, joiden työn perusteella standardin kehittämistä jatkettiin. Taulukossa 1 on lueteltu IEC-standardin osat, joista viisi ensimmäistä olivat mukana ensimmäisessä vaiheessa.

Vuonna 1993 julkaistiin standardin kolmas osa, ensimmäinen ohjelmoitavien logiikoiden ohjelmointikieliä koskeva standardi. Ohjelmoitavien logiikoiden standardi sai nimen IEC 1131, joka myöhemmin muuttui muotoon IEC 61131. Standardin kolmanteen osaan perehdytään tässä työssä tarkemmin ja myöhemmin standardiin viitatessa tarkoitetaan nimenomaan IEC-standardia 61131-3. (Hansen 2015, 134.)

TAULUKKO 1. IEC 61131 -standardi on jaettu useaan osaan (PLCopen 2011)

| # | Otsikko |
|---|---|
| 1 | General information |
| 2 | Hardware and requirements for testing |
| 3 | Programming languages |
| 4 | User guidelines |
| 5 | Communications |
| 6 | Functional safety for PLC |
| 7 | Fuzzy-control programming |
| 8 | Guidelines for the application and implementation of programming languages |
| 9 | Single-drop digital communication interface for small sensors and actuators |

IEC 611131 -standardiin on ilmestynyt päivityksiä useaan otteeseen alkuperäisen julkaisun jälkeen. Päivityksiä on julkaistu jokaiselle osalle erikseen, eikä koko standardia ole päivitetty kerralla. Osien lukumäärä on kasvanut alkuperäisestä viidestä yhdeksään. Uusia osia ovat turvalogiikoihin vaatimuksia määrittävä IEC 61131-6, sumean logiikan ohjelmointiin liittyvä IEC 61131-7, ohjelmointikielien toteutusta määrittävä IEC 61131-8 sekä kehittyneiden antureiden kommunikointirajapintaa määrittävä IEC 61131-9 (PLCopen 2011).

## 2.2 IEC 61131-3 -standardi

IEC 61131-3 on standardi, joka määrittää logiikkaohjelmien ohjelmointikielet sekä niiden syntaksin ja toiminnan. Vaikka IEC 61131-3 onkin standardi, se ei kuitenkaan velvoita valmistajia noudattamaan annettuja ohjeita. Sitä pidetään enemmänkin suunnan-

näyttäjänä, sillä liian tiukan standardin luominen olisi rajoittanut tutkimus- ja kehitystyötä sekä kilpailua (Hansen 2015, 137). Standardi on ollut osa IEC 61131 -standardia alusta alkaen ja siihen on julkaistu kaksi päivitystä. Vuonna 2002 julkaistiin toinen versio, joka korvasi alkuperäisen vuodesta 1993 käytössä olleen standardin. Kolmas, eli uusin, päivitys ilmestyi vuonna 2013 ja se toi mukanaan lukuisia uudistuksia pysyen kuitenkin yhteensopivana edellisen version kanssa (PLCopen 2011). Viimeisin päivitys toi uusia muista ohjelmointikielistä tuttuja ominaisuuksia logiikkaohjelmointiin esittelemällä erityisesti olio-ohjelmoinnista tuttuja elementtejä, joita hyödyntämällä voidaan jatkossa kehittää yhä monimutkaisempia järjestelmiä (PLCopen 2011). Olio-ohjelmointia käsitellään kappaleessa 3.3.

## 2.3 PLCopen

PLCopen on maailmanlaajuinen järjestö, jonka tehtävänä on edistää IEC 61131-3 -standardin käyttöä ja tietoisuutta maailmalla. Järjestö perustettiin vuonna 1992, jolloin standardin ensimmäinen versio hyväksyttiin valmiiksi. PLCopen on laite- ja tuoteriippumaton ja siihen kuuluu useita ohjelmoitavien logiikoiden valmistajia, ohjelmistoyrityksiä sekä järjestöjä. Järjestön tarkoitus ei ole kehittää itse standardia vaan enemmänkin edesauttaa sen käyttöä ja tunnettavuutta teollisuudessa. (John & Tiegelkamp 2010, 16–17.)

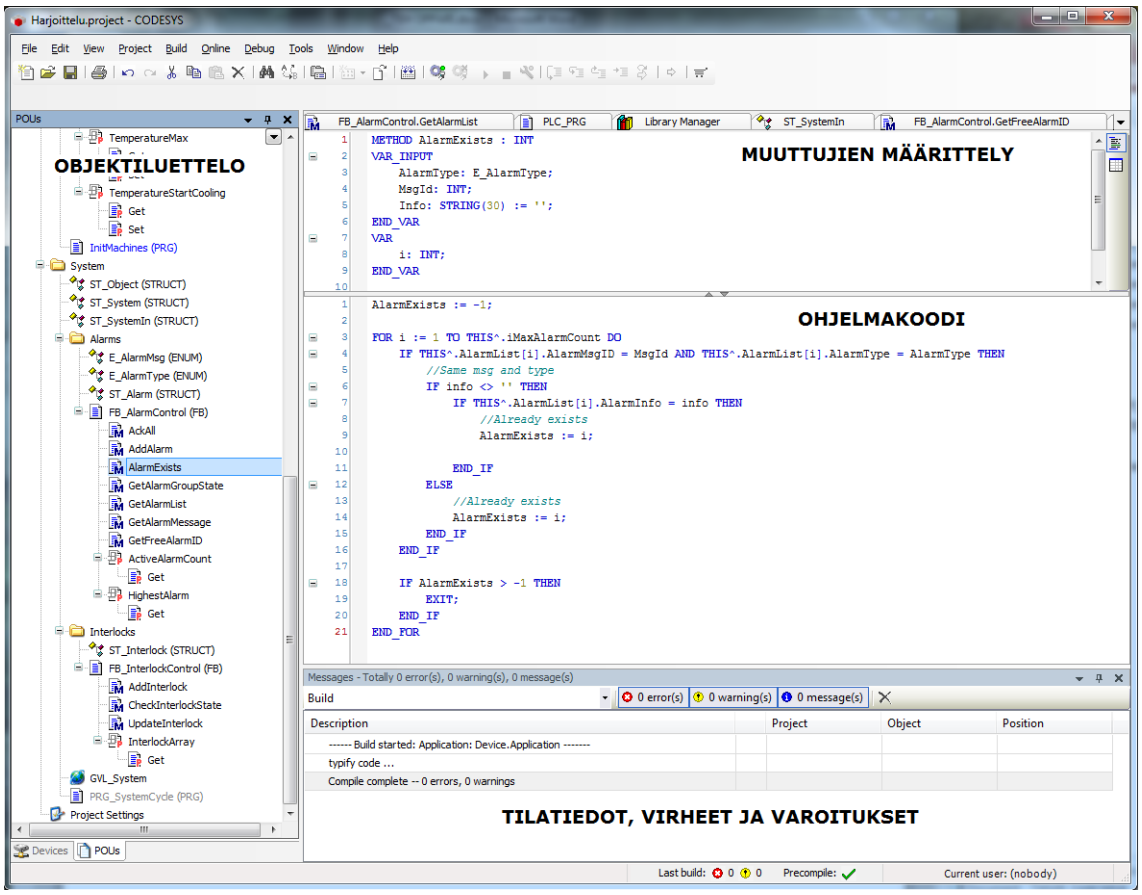
PLCopen tiedottaa IEC 61131-3 -standardin muutoksista sekä nykytilanteesta. Järjestö myös järjestää koulutuskursseja, jakaa materiaalia sekä toimii logiikkaohjelmistojen käyttäjien ja kehittäjien välisenä keskusteluympäristönä (John & Tiegelkamp 2010, 18). PLCopen on muun muassa julkaissut Motion Control -kirjastot, joiden sisältämien toiminnallisuuksien avulla servo- ja NC-ohjaukset voidaan toteuttaa IEC 61131-3 -ympäristöissä (Motion Control - An Introduction 2015). PLCopen on myös kehittänyt XML-pohjaisen tavan logiikkaohjelmien siirtämiseen eri valmistajien järjestelmien välillä, minkä avulla IEC 61131-3 -pohjaisia logiikkaohjelmia voidaan siirtää helposti ohjelmointiympäristöstä toiseen (John & Tiegelkamp 2010, 19). CODESYS-pohjaisissa järjestelmissä on mahdollista tallentaa ohjelmakoodi PLCopenXML-muodossa, joten esimerkiksi koodin siirto CODESYS- ja TwinCAT -järjestelmien välillä on helppoa.

PLCopen:in tehtävä on myös ohjelmoitavien logiikoiden sertifiointi. Järjestö on määrittänyt ehtoja, joiden perusteella voidaan tarkastaa onko valmistajan järjestelmä IEC 61131-3-standardia noudattava. Sertifikaatteja on kolme: Base Level, Reusability Level sekä Conformity Level. Base Level on tasoista alhaisin ja se myönnetään järjestelmille, jotka tukevat IEC 61131-3 -standardin mukaisia ohjelmointirakenteita. Reusability Level myönnetään järjestelmille, joiden toimilohkot ja funktiot ovat yhteensopivia muiden saman sertifikaatin omaavien järjestelmien kanssa. Conformity Level -sertifikaatti kertoo järjestelmän tukevan kaikkia standardin esittämiä tietorakenteita ja niiden toiminta on tarkastettu järjestön määrittämällä testillä. (John & Tiegelkamp 2010, 19.)

## 2.4 CODESYS

CODESYS on saksalaisen 3S-Smart Software Solutions:in kehittämä IEC 61131-3 -pohjainen automaatiojärjestelmä, joka on täysin laitteistoriippumaton. CODESYS:in avulla voidaan ohjelmoida ja konfiguroida usean eri valmistajan laitteita, toisin kuin perinteisillä valmistajien omilla ohjelmistoilla. Valmistajan tarvitsee vain ostaa lisenssi CODESYS:in valmistamaan Runtime System -kehitystyökaluun ja sovittaa se omaan laitteeseensa. Runtime voidaan joko sovittaa rautatason laitteeseen, kuten perinteiseen ohjelmoitavaan logiikkaan, tai esimerkiksi teollisuus-PC:lle CODESYS:in tarjoaman ohjelmiston avulla. Näin ollen mistä tahansa tietokoneesta voidaan tehdä tehokas ohjelmoitava logiikka eli PLC (Programmable Logic Controller). CODESYS-järjestelmien valtuutettu jälleenmyyjä Suomessa on SKS Control Oy. (3S 2014, 2015.)

CODESYS Engineering on ohjelmointiympäristö, jonka avulla voidaan ohjelmoida CODESYS-pohjaisia järjestelmiä. Ohjelmisto on lisenssivapaa eli sen käyttö on ilmaista. Uusin ohjelmiston versio on 3.5 ja se tukee täysin IEC 61131-3 -standardin viimeisimmän päivityksen ominaisuuksia, kuten olio-ohjelmointia. CODESYS tarjoaa myös työkalut käyttöliittymien (HMI, Human-Machine Interface) suunnitteluun paneeleille, tietokoneille sekä internet-selaimille. Myös turvaluokiteltujen järjestelmien (Safety) ohjelmointi on mahdollista SIL2- ja SIL3-luokkiin asti kehitysympäristöjen ja runtime-ohjelmiston integroitujen ominaisuuksien ansiosta. (3S 2014, 2015.)




KUVA 1. Ruudunkaappaus CODESYS-ohjelmointiympäristöstä ja näkymän eri osat

CODESYS-ohjelmointiympäristö vastaa käyttöliittymältään nykyajan standardeja. Ohjelmakoodi korostetaan eri väreillä, tiedostot avautuvat omiin välilehtiin, virhesanomat esitetään selkeästi ja projektin rakenne näytetään puurakenteen avulla (kuva 1). Ohjelmaan voidaan myös asettaa pysäytyspisteitä (breakpoint), joiden avulla virheiden etsiminen ja toiminnan varmistaminen helpottuu. Logiikkaan yhteydessä ollessa (online-tila), näytetään ohjelmakoodin yhteydessä muuttujien todelliset arvot suoritushetkellä. Kuvassa 2 nähdään kuinka ohjelmointiympäristö on korostanut ohjelmointikielen varatut sanat sinisellä ja kommentit vihreällä. Jokaisen muuttujan arvo näytetään muuttujan vieressä oranssilla värillä korostettuna. Kuvan viimeisellä rivillä ohjelmakoodin suoritus on keskeytetty pysäytyspisteen avulla, minkä merkkinä rivinumeron vieressä on punainen pallo.

```
1  AlarmExists[1] := -1;
2
3  FOR i[1] := 1 TO THIS^.iMaxAlarmCount[100] DO
4      IF THIS^.AlarmList[i[1]].AlarmMsgID[5001] = MsgId[5001] AND
5          THIS^.AlarmList[i[1]].AlarmType[AlarmTypeW] = AlarmType[AlarmTypeW] THEN
6
7          //Same message and type
8          IF info['Test alarm'] <> '' THEN
9              IF THIS^.AlarmList[i[1]].AlarmInfo['Test alarm'] = info['Test alarm'] THEN
10                 //Already exists
11                 AlarmExists[1] := i[1];
12             END_IF
13         ELSE
14             //Already exists
15             AlarmExists[1] := i[1];
16         END_IF
17
18     END_IF
19
20     IF AlarmExists[1] > -1 THEN
21         EXIT;
22     END_IF
23 END FORRETURN
```

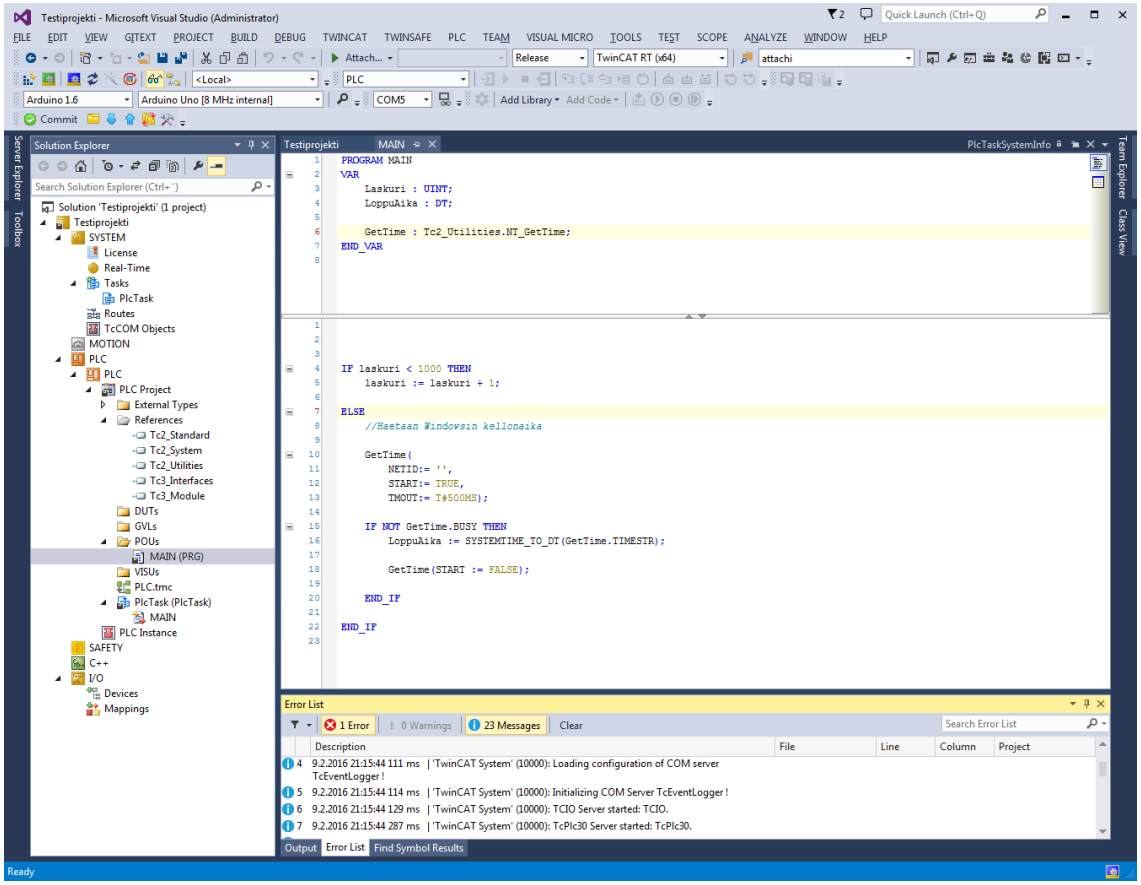
KUVA 2. Ohjelmakoodi korostetaan värein ja muuttujien arvot esitetään online-tilassa

## 2.5 TwinCAT

TwinCAT on saksalaisen Beckhoff Automation GmbH & Co. KG:n automaatio-ohjelmisto, joka noudattaa IEC 61131-3 -standardia. Beckhoff valmistaa ohjelmoitavia logiikoita, liikkeenohjaustuotteita sekä ohjauspaneeleita ja yritys on erityisesti erikoistunut PC-pohjaisiin ohjausjärjestelmiin. Beckhoff on voimakkaasti kasvava yhtiö ja sillä on edustus yli 75 eri maassa. Suomessa Beckhoff on toiminut omalla yhtiöllä vuodesta 2000 ja Suomen pääkonttori sekä komponenttivarasto sijaitsevat Hyvinkäällä, josta käsin tuotteita toimitetaan ympäri Suomea. (Beckhoff 2016.)

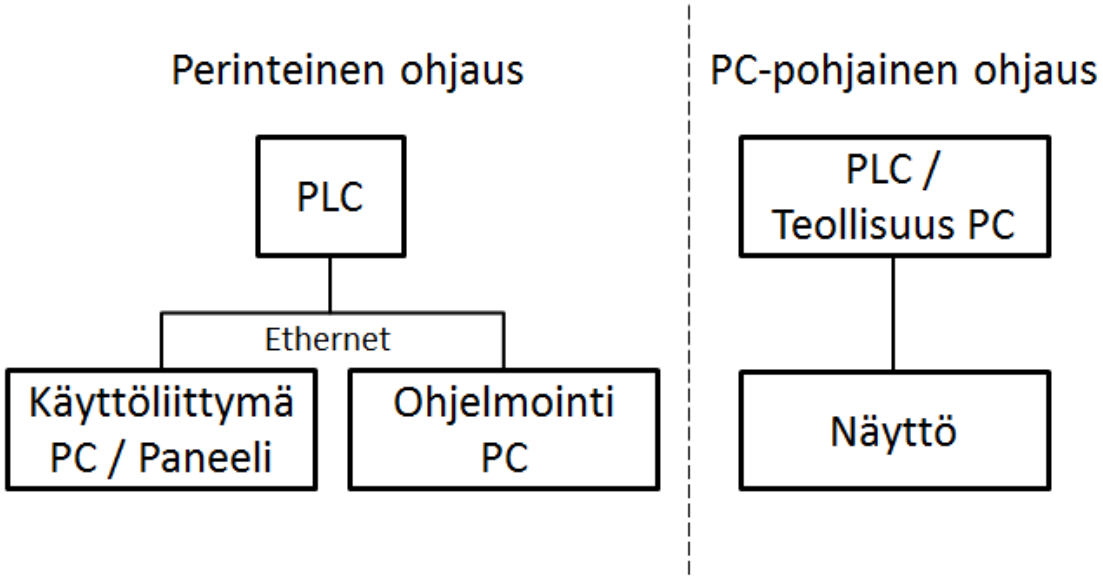
TwinCAT eXtended Automation Engineering (XAE) -ohjelmisto perustuu 3S-Smart Software Solutions:in kehittämään CODESYS-ohjelmointiympäristöön, minkä johdosta TwinCAT ja CODESYS ovat hyvin samankaltaisia (Hess 2014, 11). Beckhoffin TwinCAT rakentuu Microsoftin Visual Studio -ohjelmointiympäristön päälle, minkä ansiosta useita Visual Studion tuttuja ominaisuuksia voidaan hyödyntää logiikkaohjelmoinnissa (kuva 3). Uusin versio on TwinCAT 3, joka sisältää standardin uusimmat ominaisuudet, kuten mahdollisuuden käyttää oliopohjaisia ohjelmointitekniikoita. TwinCAT 3 -

ohjelmisto on saanut PLCopen-järjestön Base Level -hyväksynnän. (Beckhoff 2012, 5, 7, 23.)




KUVA 3. TwinCAT 3 -ohjelmointiympäristö toimii Visual Studiossa

TwinCAT 3 -pohjaiset järjestelmät suoritetaan Windows-käyttöjärjestelmässä, joten logiikkaohjelmaa suorittava järjestelmä voi toimia itsessään myös ohjelmointilaitteena. Näin ollen ei välttämättä tarvita erillistä ohjelmointitietokonetta vaan ohjelmointiympäristö voi sijaita samassa laitteessa. Samalla myös käyttöliittymä voi toimia samassa laitteessa kuin logiikkasovellus. Hyödyntämällä PC-pohjaisuutta ja järjestelmän kykyä käyttää Windowsin tiedostojärjestelmiä, ajureita sekä sovelluksia, on mahdollista korvata useita järjestelmiä yhdellä laitteella (kuva 4).




KUVA 4. PC-pohjainen järjestelmä voi korvata useita perinteisesti tarvittuja laitteita

# 3 LOGIIKKAOHJELMOINTI IEC 61131-3 -YMPÄRISTÖSSÄ

Tässä kappaleessa käydään läpi IEC 61131-3 -standardin määrittämät ohjelmointikielet, logiikkaohjelmissa käytetyt elementit sekä olio-ohjelmoinnista tuttuja uusia logiikkaohjelmointitekniikoita. Jokaisen ohjelmointikielen yhteydessä on esimerkkinä kyseisellä kielellä tehty ohjelma, jolla toteutetaan sama toiminnallisuus. Kappaleessa käsitellään myös olio-ohjelmointia yleisesti ja sen käyttöä logiikkaohjelmoinnissa.

## 3.1 Ohjelmointikielet

IEC 61131-3 -standardissa esitellään viisi eri ohjelmointikieltä, jotka ovat lueteltu taulukossa 2. Kielistä kaksi ovat tekstipohjaisia ja kolme graafisia ohjelmointikieliä. Tämän työn ohjelmoinnissa käytettiin Structured Text -ohjelmointikieltä.

TAULUKKO 2. IEC 61131-3 -standardin ohjelmointikielet ja niiden lyhenteet

| | |
|---|---|
| Structured Text | ST |
| Instruction List | IL |
| Function Block Diagram | FBD |
| Ladder Diagram | LD |
| Sequential Function Chart | SFC |

### 3.1.1 Structured Text (ST)

Structured Text (ST) on tekstipohjainen korkeamman tason ohjelmointikieli ja sen syntaksi perustuu Pascal-ohjelmointikieleen. Kielessä on myös samankaltaisuutta yleisesti käytettyyn C-kieleen. Kieli on kehitetty erityisesti aritmeettisiin laskuihin sekä numeroiden ja monimutkaisten tietorakenteiden käsittelyyn. Erityisesti perinteistä ohjelmointikokemusta omaaville kieli on nopeasti lähestyttävä ja vapaampi kuin yleisesti käytetty LD-ohjelmointikieli. (Hansen 2015, 278.)

```
IF Ehto1 AND Ehto2 THEN
    Summa := Luku1 + Luku2;
END_IF

OnSuurempi := Summa > 20;
```

KUVA 5. Esimerkkiohjelma Structured Text -ohjelmointikielellä

Kuvassa 5 on ST-kielellä toteutettu esimerkkiohjelma. Ohjelmassa asetetaan muuttujaan Summa muuttujien Luku1 ja Luku2 yhteenlaskun tulos, mikäli muuttujat Ehto1 ja Ehto2 ovat molemmat tosia. Lopuksi OnSuurempi-muuttujaan asetetaan arvo, joka kertoo onko Summa suurempi kuin 20. ST-ohjelmointikielellä voidaan myös toteuttaa kätevästi erilaisia silmukoita ilman IL-kielestä tuttua hyppykäskyä. Silmukoiden avulla voidaan toteuttaa monimutkaista tiedon käsittelyä (kuva 6).

```
PROGRAM Ohjelma
VAR
    Taulukko : ARRAY [1..5] OF REAL := [5.0, 10.55, 22.32, 2.35, 77.65];
    i: UINT;
    SuurinArvo : REAL;
END_VAR
//Käydään jokainen taulukon arvo läpi
FOR i := 1 TO 5 DO
    //Jos arvo on suurempi kuin nykyinen, päivitä
    IF Taulukko[i] > SuurinArvo THEN
        SuurinArvo := Taulukko[i];
    END_IF

END_FOR
```

KUVA 6. ST-ohjelmointikielellä voidaan toteuttaa helposti erilaisia silmukoita

### 3.1.2 Instruction List (IL)

Instruction List (IL) on tekstipohjainen ohjelmointikieli, joka perustuu yksinkertaisiin komentoihin. IL on assembler-tyylinen matalan tason ohjelmointikieli, jonka etuna on vähäinen laskentatehon tarve. Useita vanhoja logiikoita voidaan ohjelmoida vain IL- tai LD-kielillä, joten kieli on yhä tarpeellinen. (Hansen 2015, 139–140.) IEC 61131-3 -standardin vuonna 2013 julkaistussa päivityksessä IL-ohjelmointikieli julistettiin van-

hentuneeksi (deprecated) ja se ei enää kuulu standardiin seuraavassa julkaisussa (IEC 61131-3 2013, 195). Instruction List -ohjelmat jaetaan piireihin (network, rung) sekä nimikkeisiin (label), joihin voidaan hypätä hyppykäskyillä.

Kuvassa 7 on toteutettu esimerkkiohjelma IL-ohjelmointikielellä. Ohjelma on jaettu kolmeen eri piiriin ja lisäksi kahteen nimikkeeseen. Nimikkeisiin hypätään ehdollisella (JMPC) tai ehdottomalla (JMP) hyppykomennolla.

```
1   Piiri 1

    Jos Ehto1 ja Ehto2 ovat tosia, suorita piirissä 2 oleva yhteenlasku
    LD      Ehto1
    AND     Ehto2
    JMPC    Lasku       Lasketaan jos ehto tosi

    JMP     Vertailu    Vertaillaan joka tapauksessa

2   Piiri 2
    Suorita yhteenlasku

    Lasku:
    ld      Luku1
    ADD     Luku2
    ST      Summa

3   Piiri 3
    Suorita vertailu

    Vertailu:
    LD      Summa
    GT      20
    ST      OnSuurempi
```

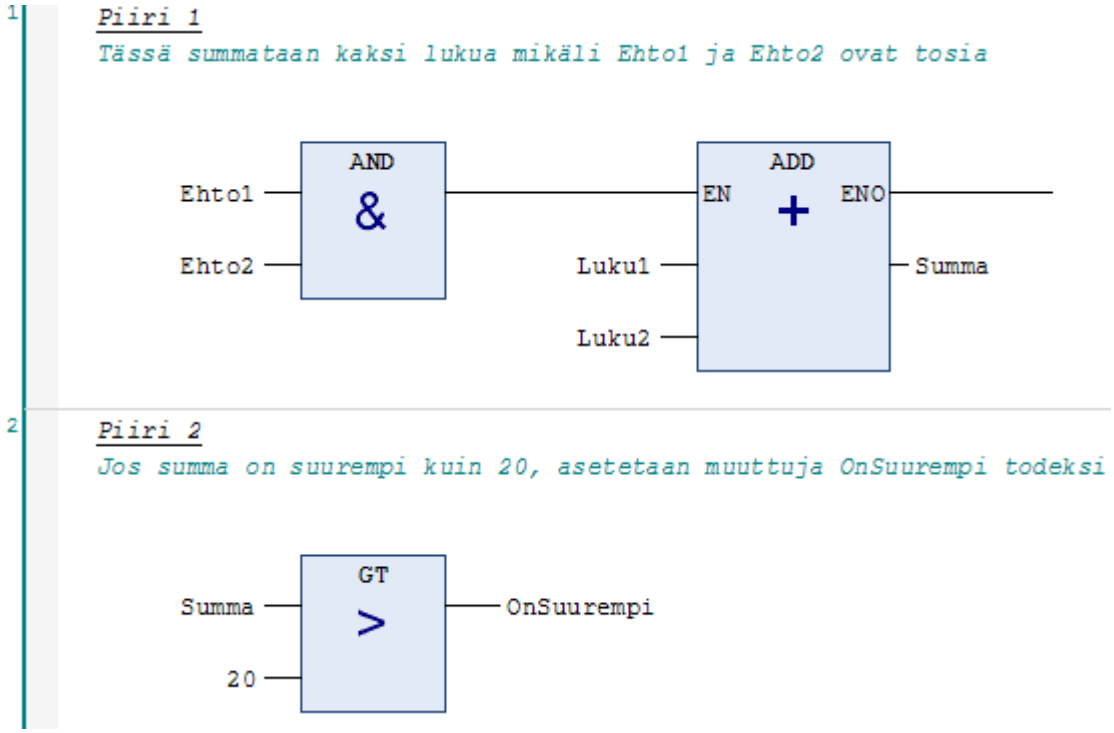
KUVA 7. Esimerkkiohjelma Instruction List -ohjelmointikielellä

### 3.1.3 Function Block Diagram (FBD)

Function Block Diagram (FBD) on ohjelmointikieli, joka perustuu laatikoina esitettyihin toimilohkoihin. Jokaisella toimilohkolla voi olla sisään- ja ulostuloja ja toiminta rakennetaan yhdistämällä eri lohkojen sisään- ja ulostuloliitäntöjä toisiinsa. Kielen hyvä ominaisuus on se, että ohjelmaa lukeva näkee nopealla silmäyksellä ohjelman tilan yksinkertaisen rakenteen ansiosta. FBD-ohjelmat jaetaan omiin päällekkäin aseteltaviin

piireihin vastaavasti kuin IL-ohjelmat ja piirit muodostavat ohjelmakokonaisuuden yhdessä. (Hansen 2015, 262–263.)

Kuvassa 8 on esimerkkiohjelma toteutettuna FBD-ohjelmointikielellä. Ohjelma on jaettu kahteen piiriin, joista ensimmäisessä suoritetaan ehtojen tarkastus sekä mahdollinen laskutoimitus ja toisessa yhteenlaskun tuloksen vertailu.

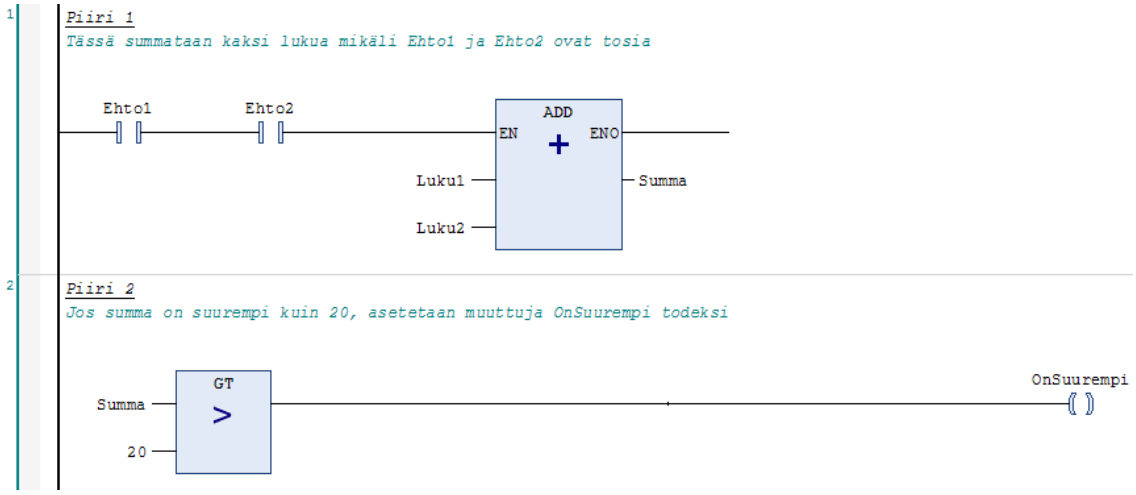



KUVA 8. Esimerkkiohjelma Function Block Diagram -ohjelmointikielellä

### 3.1.4 Ladder Diagram (LD)

Ladder Diagram (LD) eli tikapuukaavio on ohjelmointitapa, joka vastaa hyvin pitkälle sähködokumentaatiossa käytettyjä relekaavioita. Pääasiassa LD on suunniteltu binääritietojen käsittelyyn, mutta valmistajat ovat kehittäneet kieltä toimimaan yhdessä ST- ja FBD-kielillä tehtyjen ohjelmien kanssa. Kuten FBD:lla ohjelmoidut ohjelmat, myös LD-ohjelmat jaetaan omiin piireihin. Tikapuukaavioilla ohjelmoitujen ohjelmien hyvä puoli on se, että niistä saa nopeasti kokonaiskuvan toiminnasta ja tilasta myös ilman ohjelmointitaitoja. (Hansen 2015, 223–224.)

Kuvassa 9 on esimerkkiohjelma LD-ohjelmointikielellä. Ohjelma on lähes vastaava kuin FBD:lla, mutta ehtoja kuvaavat koskettimet ja sijoitusta kuvaava kela ovat erilaisia.



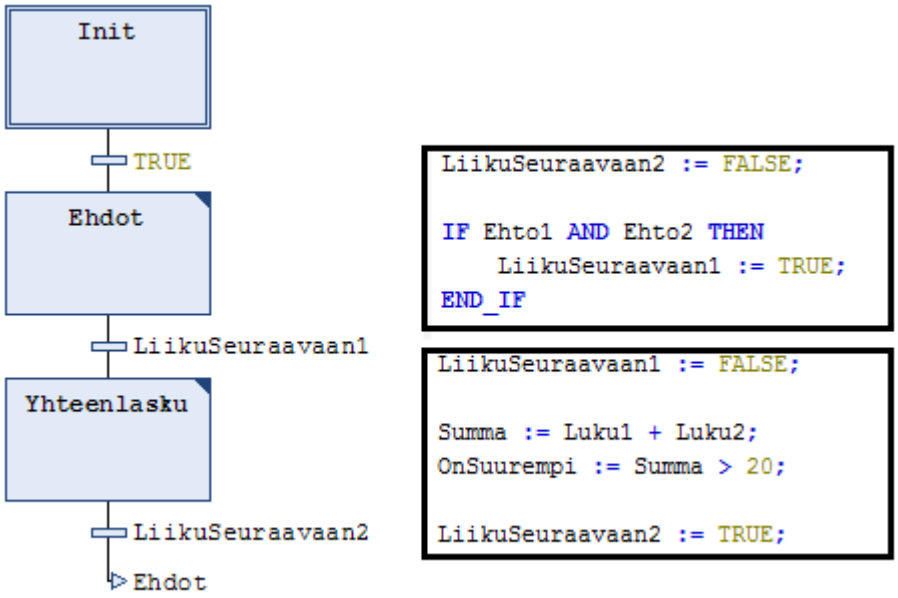

KUVA 9. Esimerkkiohjelma Ladder Diagram -ohjelmointikielellä

### 3.1.5 Sequential Function Chart (SFC)

Sequential Function Chart (SFC) eli sekvenssikaavio on ohjelmointitapa, joka perustuu peräkkäin tai rinnakkain tapahtuviin toimintoihin. SFC ei ole perinteinen ohjelmointikieli vaan enemmänkin graafinen tekniikka ohjelmakoodin organisointiin. SFC:n toiminta perustuu kolmeen erilaisiin elementteihin: askeleisiin (step), siirtymäehtoihin (transition) sekä toimintoihin (action). Askeleesta toiseen siirrytään siirtymäehdon ollessa tosi ja jokaisessa askeleessa suoritetaan määritetyt toiminnot. (Hansen 2015, 307.)

Esimerkkiohjelman toteuttaminen SFC-kielellä ei ole käytännöllistä, mutta kuitenkin mahdollista (kuva 10). Init-askel suoritetaan ainoastaan logiikan käynnistyksen yhteydessä. Seuraavassa Ehdot-askeleessa suoritetaan muuttujien Ehto1 ja Ehto2 vertailu ja mikäli molemmat ovat tosia, siirrytään seuraavaan askeleeseen. Yhteenlasku-askeleessa suoritetaan yhteenlasku sekä vertaillaan, onko Summa-muuttujan arvo suurempi kuin 20. Askelia kuvaavien laatikoiden oikeassa yläkulmassa oleva merkintä kertoo, että kyseiselle askeleelle on määritelty toimintoja eli ohjelmakoodia. Kuvassa toiminnot löyty-

vät laatikoiden vierestä korostettuina lukemisen helpottamiseksi. Sekvenssikaavion lopuksi siirrytään takaisin Ehdot-askeleeseen hyppykomennolla.

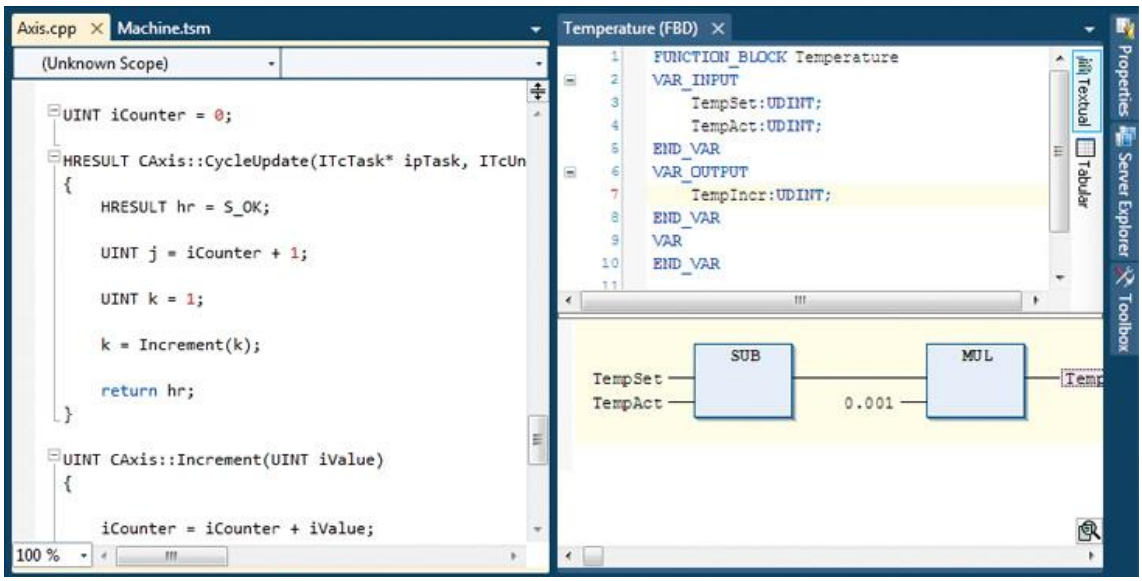



KUVA 10. Esimerkkiohjelma Sequential Function Chart -ohjelmointikielellä

### 3.1.6 Standardin ulkopuoliset kielet

CODESYS-pohjaisissa järjestelmissä on standardissa määritettyjen ohjelmointikielien lisäksi myös Continuous Function Chart (CFC) -ohjelmointikieli. CFC perustuu FBD-ohjelmointikieleen, mutta eroaa siinä, että ohjelmaa ei jaeta omiin piireihin vaan koko ohjelma on yhtä ja samaa piiriä. Lohkot voidaan sijoittaa täysin vapaasti ja takaisinkytkennät ovat mahdollisia, joten CFC on joustavampi ohjelmoida kuin FBD. (3S 2010, 33.)

CODESYS tukee myös C-ohjelmointikielellä toteutettujen lähdekoodien ja otsikkotiedostojen lisäämistä IEC 61131-3 -pohjaisiin projekteihin. Toiminnon avulla voidaan käyttää jo olemassa olevia algoritmeja PLC-sovelluksissa ilman koodin uudelleenkirjoitusta. (3S 2016.) Myös TwinCAT 3 -ympäristössä on mahdollista ohjelmoida C- ja C++ -ohjelmointikielillä (Beckhoff 2012, 8). Kuvassa 11 on ruudunkaappaus TwinCAT 3 -ohjelmasta, mistä nähdään kuinka C++ -koodin ja perinteisen logiikkaohjelmakoodin suunnittelu tapahtuu samassa ohjelmointiympäristössä.




KUVA 11. TwinCAT 3 tukee ohjelmointia myös C++ -kielellä (TwinCAT 3 – XA Language Support … 2011)

## 3.2 Ohjelmointikielten elementit

IEC 61131-3 -standardi määrittelee ohjelmoitavissa logiikoissa käytetyt tietotyypit, rakenteet ja ohjelmatyypit. Tässä kappaleessa käydään kielen elementit läpi pääpiireittäin keskittyen erityisesti tietorakenteisiin ja ohjelmien rakenneyksikköihin.

### 3.2.1 Tietotyypit

Standardissa määritetyt alkeistietotyypit (elementary data types) voidaan jakaa binäärisiin tyyppeihin, etumerkillisiin ja etumerkittömiin kokonaislukuihin, liukulukuihin sekä aikaa, päivämäärää ja merkkijonoja esittäviin tyyppeihin. Taulukossa 3 on luetteloitu standardin määrittämät alkeistietotyypit. (John & Tiegelkamp 2010, 84.)

TAULUKKO 3. Standardin määrittämät alkeistietotyypit

| | |
|---|---|
| Binääriset tyypit | BOOL, BYTE, WORD, DWORD, LWORD |
| Etumerkilliset kokonaisluvut | INT, SINT, DINT, LINT |
| Etumerkittömät kokonaisluvut | UINT, USINT, UDINT, ULINT |
| Liukuluvut | REAL, LREAL |
| Aika, päivämäärä ja merkkijono | TIME (T), DATE (D), TIME_OF_THE_DAY (TOD), DATE_AND_TIME (DT), CHAR, WCHAR, STRING, WSTRING |

Alkeistietotyypeistä voidaan muodostaa johdettuja tietotyyppejä (derived data types). Johdetuille tietotyypeille voidaan määrittää alkuarvo (initial value), sallittu arvoalue (range), taulukko-ominaisuus (array), lueteltu tietotyyppi (enumeration) sekä rakenne (structure). Johdetut tietotyypit esitellään ohjelmointiympäristössä TYPE- ja END TYPE -lohkojen välissä. (John & Tiegelkamp 2010, 87.)

Lueteltujen tietotyyppien avulla ohjelmakoodia voidaan yksinkertaistaa käyttämällä kokonaislukujen tilalla vakioituja merkkijonoja. Kuvassa 12 on määritelty johdettu tietotyyppi OhjelmanTila, joka on lueteltu tietotyyppi. Määrittely kertoo, että Stop vastaa lukua 0, Idle lukua 20 ja niin edelleen. Näin ollen muuttujaan, jonka tyyppi on OhjelmanTila, voidaan antaa ohjelmakoodia kirjoittaessa kyseiset merkkijonot eikä numeroita tarvitse käsitellä. Lueteltujen tietotyyppien avulla ohjelmakoodista saadaan paremmin ymmärrettävää, luettavampaa ja samalla ohjelmointivirheiden todennäköisyys pienenee. (John & Tiegelkamp 2010, 87.)

**MÄÄRITYS:**

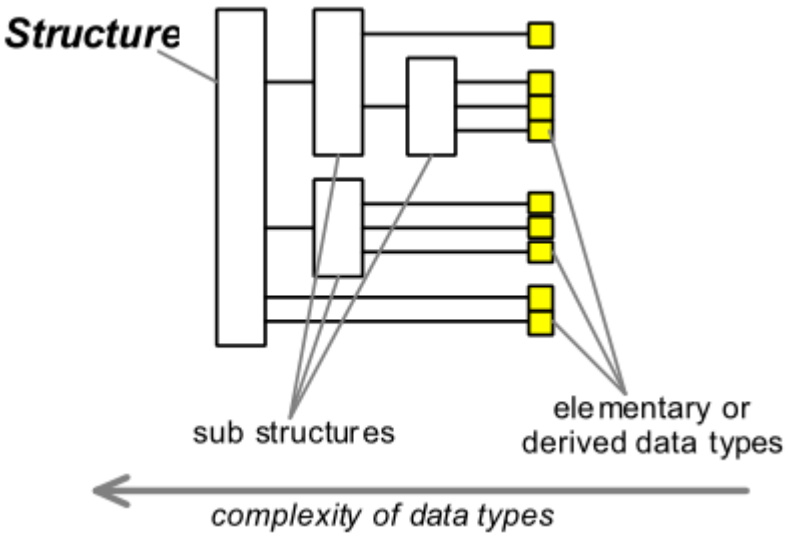
```
TYPE OhjelmanTila :
(
    Stop    := 0,
    Idle    := 20,
    Start   := 50,
    Run     := 100
);
END_TYPE
```

**KÄYTTÖ:**

```
VAR
    PaaOhjelma : OhjelmanTila;
END_VAR

PaaOhjelma := OhjelmanTila.Start;
```

KUVA 12. Luetellun tietotyypin OhjelmanTila määritys ja käyttö

Rakenteisten tietotyyppien avulla voidaan luoda korkean tason ohjelmointikielistä tuttuja tietorakenteita. Rakenteet voivat sisältää sekä alkeis- että johdettuja tietotyyppejä. Tämän ansiosta voidaan luoda monirakenteisia tietotyyppejä, eli rakenne voi sisältää useita muita rakenteita, jotka taas voivat sisältää lisää rakenteita (kuva 13). Rakenteisten tietotyyppien avulla on mahdollista hallita suuria määriä dataa ja pitää se selkeästi organisoituna. (John & Tiegelkamp 2010, 83.)




KUVA 13. Monirakenteisten tietotyyppien sisältö (John & Tiegelkamp 2010)

Kuvassa 14 on esimerkki tietorakenteen käytöstä. Aluksi on määritelty johdettu tietotyyppi Moottori, joka on rakennetyyppinen. Moottori-tyypin alle on määritelty erilaisia tietotyyppejä, kuten merkkijono Nimi, lueteltu tietotyyppi Tyyppi (tyyppiä Moottori-Tyyppi) ja rakenteinen tietotyyppi Vika (tyyppiä VikaTieto). Johdettu tietotyyppi VikaTieto taas sisältää kolme eri alkeistietotyyppiä, jotka ovat VirheKoodi, VikaPaalla ja Havaittu.

Moottori-tietotyyppiä käytetään ohjelmassa luomalla aluksi sen tyyppinen muuttuja KuljetinMoottori. Muuttujan esittelyn jälkeen kyseisen tietotyypin arvoihin päästään käsiksi helposti ohjelmakoodissa. Tietorakenteen arvoihin päästään käsiksi erottamalla rakenteen elementit toisistaan pisteen avulla. Näin Moottori-tietotyypin sisältämään

VikaTieto-tietotyypin Virhekoodi-arvoon päästään kirjoittamalla ohjelmakoodissa ”KuljetinMoottori.Vika.Virhekoodi”. Tietorakenteiden avulla ohjelmakoodista saadaan paremmin ymmärrettävä ja selkeämpi.

**MÄÄRITYS:**

```
TYPE Moottori :
STRUCT
    Nimi        : STRING(20);
    Tyyppi      : MoottoriTyyppi;
    MaxRPM      : INT := 1500;
    Ohjaus      : BOOL;
    KayntiTieto : BOOL;
    Vika        : VikaTieto;
END_STRUCT
END_TYPE
TYPE VikaTieto :
STRUCT
    VirheKoodi  : UINT;
    VikaPaalla  : BOOL;
    Havaittu    : DATE_AND_TIME;
END_STRUCT
END_TYPE
```

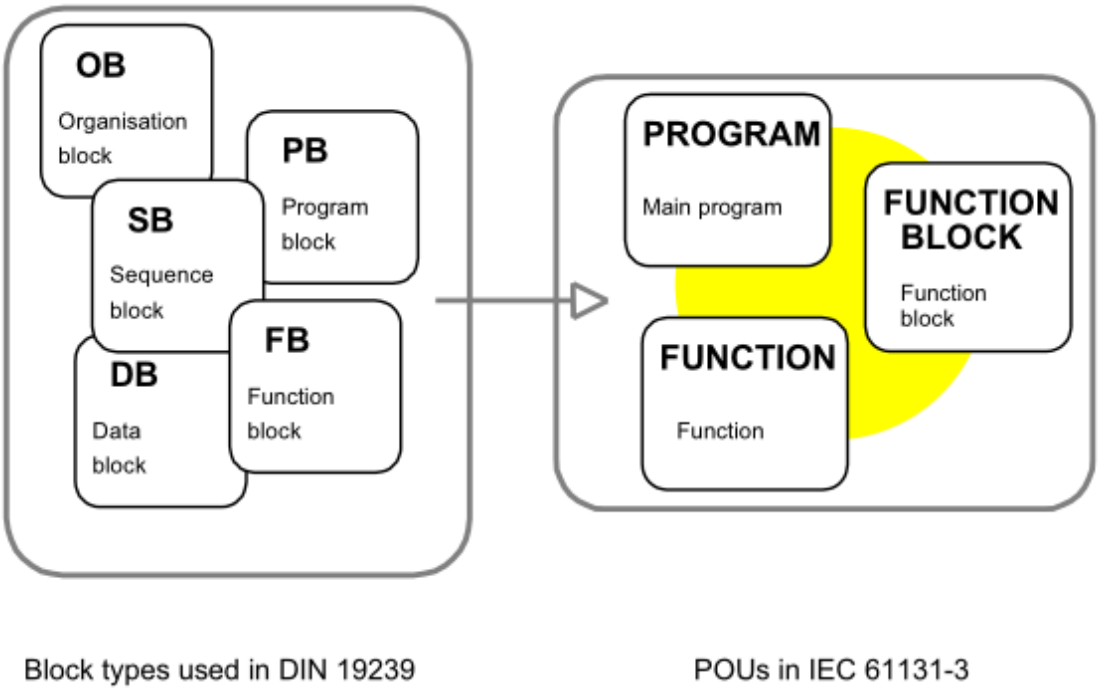
**KÄYTTÖ:**

```
VAR
    KuljetinMoottori : Moottori;
END_VAR

KuljetinMoottori.Nimi := 'Kuljetin 1';
KuljetinMoottori.Tyyppi := MoottoriTyyppi.Yksivaihe;

IF KuljetinMoottori.Vika.VikaPaalla THEN
    KuljetinMoottori.Ohjaus := FALSE;
END_IF
```

KUVA 14. Monirakenteisen tietotyypin määritys ja käyttö

### 3.2.2 Ohjelmien rakenneyksiköt

Standardissa toimintoja suorittavia ohjelmia kutsutaan rakenneyksiköiksi (Program Organization Unit, POU). IEC 61131-3 -standardin tarkoituksena on vähentää erilaisten ohjelmalohkojen lukumäärää järjestelmän yksinkertaistamiseksi. IEC-standardia edeltänyt saksalainen DIN 19239 -standardi sisälsi useita erilaisia ohjelmalohkoja. IEC-standardi on pyrkinyt vähentämään niiden lukumäärää ja tuloksena on ainoastaan kolme erilaista rakenneyksikköä (kuva 15). Standardissa määritetyt rakenneyksiköt ovat ohjelma (program), toimilohko (function block) sekä funktio (function). (John & Tiegelkamp 2010, 37.)




KUVA 15. IEC 61131-3 -standardissa on kolme rakenneyksikköä (John & Tiegelkamp 2010)

Jotta logiikkaohjelma toimii, sitä täytyy kutsua tietyn ajan välein. IEC 61131-3 -standardi määrittelee tehtävät (task), jotka määritellään ohjelmointiympäristössä. Tehtävä kutsuu määriteltyä rakenneyksikköä joko jonkin ehdon täyttyessä (event controlled), tietyn ajan välein (cyclically) tai jatkuvasti (freewheeling). Jokaisessa IEC 61131-3 -logiikassa on ainakin yksi tehtävä, joka kutsuu pääohjelmaa. Tehtäviä voi olla järjestelmästä riippuen lukuisa määrä ja jokaisella tehtävällä voi olla oma sykliaika tai ajoehto. Tehtäville voidaan antaa myös eri prioriteetit, joten korkeimman prioriteetin omaava tehtävä saa käyttää eniten prosessoria. (Hansen 2015, 144–145.) TwinCAT 3 -järjestelmissä tehtäviä voidaan jakaa moniydinprosessorien eri ytimille suorituskyvyn parantamiseksi, jolloin esimerkiksi ohjaus ajetaan 1. ytimessä, käyttöliittymä 2. ytimessä ja Windows-käyttöjärjestelmä 3. ja 4. ytimissä (Beckhoff 2012, 14).

### 3.2.3 Ohjelma (Program)

Program eli ohjelma on standardissa määritetyistä rakenneyksiköistä ylimmällä tasolla. Ohjelma kuvataan standardissa rakenneyksiköksi, joka yhdistää kaikki ohjelmointikielen elementit ja rakenteet siten, että tuloksena on halutun toiminnon hoitava yksikkö. Jokaisessa IEC 61131-3 -pohjaisessa projektissa on ainakin yksi ohjelma, joka toimii samalla järjestelmän pääohjelmana. (Hansen 2015, 217–218.)

Ohjelma voi kutsua muita ohjelmia, toimilohkoja sekä funktioita ja se myös säilyttää aina oman tilansa seuraavalle suorituskerralle, eli muuttujien ja ohjelman sisältämien toimilohkojen arvot pysyvät muistissa. Ohjelmia ei voida monistaa kuten toimilohkoja, vaan jokaisesta ohjelmasta on ainoastaan yksi ilmentymä eli instanssi. Muulta osin ohjelma vastaa toimilohkoja. (Hansen 2015, 218.)

### 3.2.4 Toimilohko (Function block)

Function block eli toimilohko on eniten käytetty rakenneyksikkö (John & Tiegelkamp 2010, 41). Toimilohkoja kutsutaan ohjelmista ja toimilohko voi kutsua muita toimilohkoja sekä funktioita. Toimilohkoilla on sisäinen muisti kuten ohjelmillakin, eli muuttujien ja toimilohkon sisältämien muiden toimilohkojen arvot säilyvät suorituskerrasta seuraavaan. Tämä tarkoittaa sitä, että toimilohko palauttaa samoilla sisääntuloarvoilla mahdollisesti eri ulostuloarvoja. Toimilohkoilla ei ole funktioille tyypillistä yksittäistä paluuarvoa, vaan sillä on tarvittava määrä ulostulomuuttujia. (Hansen 2015, 206.)

Toimilohkon suurin ero ohjelmiin ja funktioihin on se, että toimilohkoista voidaan tehdä useita ilmentymiä eli instansseja. Koska toimilohkolla on tiedot säilyttävä muisti, täytyy jokaiselle eri toimilohkolle varata muistia arvojen säilyttämiseksi. Jokainen toimilohkosta muodostettu instanssi suorittaa saman ohjelmakoodin mutta omilla muuttujilla, joten toimilohkot ovat monikäyttöisiä. Toimilohkojen instanssit määritetään kuten tietotyyppienkin joko rakenneyksiköiden muuttujamäärittelyissä tai globaaleissa muuttujissa. (Hansen 2015, 207.)

### 3.2.5 Funktio (Function)

Funktio on ohjelma, jonka ulostuloarvot ovat aina samat samoja sisääntuloarvoja käytettäessä. Funktioilla ei siis ole muistia ja käytettyjen muuttujien arvot palautuvat aina funktiokutsun jälkeen oletusarvoihin. Funktioista ei tehdä instansseja ja niitä voidaan kutsua mistä tahansa rakenneyksiköstä logiikkaohjelmassa eli ne ovat globaaleja. Funktioita käytetään pääasiassa suorittamaan erilaisia usein toistettavia tietotyyppeihin kohdistuvia toimintoja, kuten aritmeettisia laskutoimituksia. (Hansen 2015, 188.)

## 3.3 Olio-ohjelmointi

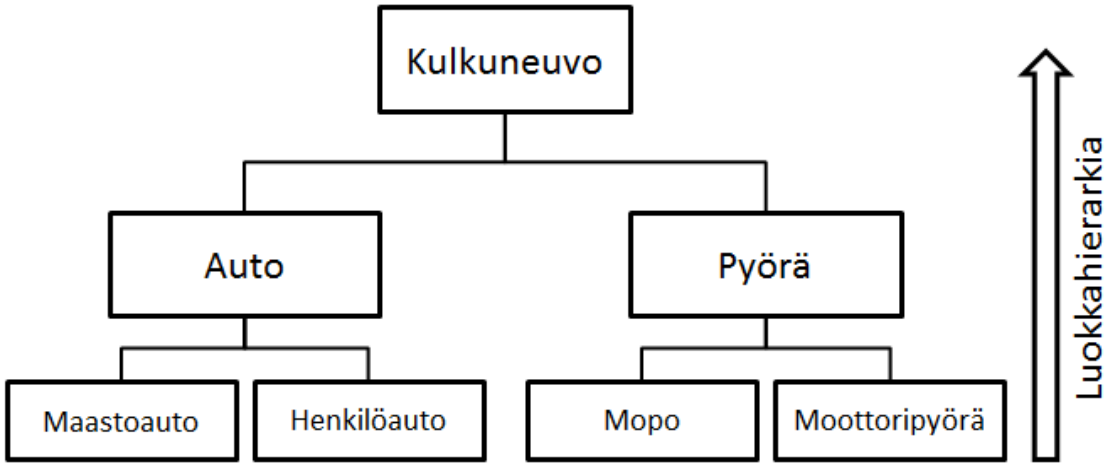
### 3.3.1 Yleistä olio-ohjelmoinnista

Olio-ohjelmointi on erityisesti Java ja C++ -ohjelmointikielten 1990-luvulla yleistämä lähestymistapa ohjelmointiin ja sen tärkeitä ominaisuuksia ovat ohjelmien osien uudelleenkäytettävyys sekä ohjelman jakaminen osiin vastuualueiden mukaan. Olio-ohjelmoinnissa ohjelman osat jaetaan luokkiin (class), joilla on omat tietonsa ja palvelunsa. Luokasta voidaan tehdä useita olioita eli luokan instansseja, jotka toteuttavat luokan palvelut. Olio-ohjelmoinnissa käytetään tiedon piilotuksen periaatetta, jonka perusteella jokainen olio voi käsitellä vain omia tietojaan. Ulkoisesti olion tietoja voidaan käsitellä ainoastaan olion tarjoaman rajapinnan kautta. Ohjelmoijan tarvitsee vain lisätä ohjelmakoodiin aiemmin luotu oliopohjainen ohjelmakoodi ja käyttää sen tarjoamaa rajapintaa. (Hietanen 1999, 6–7.)

Olio-ohjelmoinnissa luokkien uudelleenkäyttöä tukee luokkien periytyminen. Olemassa olemaa luokkaa voidaan käyttää uuden luokan pohjana, jolloin uusi luokka periytyy olemassa olevasta luokasta. Periytynyt luokka eli aliluokka (subclass) toteuttaa tällöin yliluokan (superclass) tiedot ja palvelut. Luokkia voidaan periyttää peräkkäin lukematttomasti, joten alaluokka voi toteuttaa useiden yläluokkien toiminnot. (Hietanen 1999, 11.)

Kuvassa 16 on esimerkki luokkahierarkiasta. Ylimpänä on pääluokka Kulkuneuvo, joka toimii pääluokkana kaikille alaluokille. Pääluokka voi sisältää yleisiä ja kaikkia luokkia

koskettavia tietoja kuten ajoneuvon valmistenumeron ja käyttöönottopäivän. Alaluokka Auto sisältää kaikki samat tiedot ja lisäksi omia autoille ominaisia tietoja, kuten esimerkiksi auton vetotavan. Auton alaluokkia on kaksi, jotka määräävät vielä omia tietoja ja palveluita. Näin ollen luokan Maastoauto toteuttava olio sisältää myös kaikki luokkien Auto ja Kulkuneuvo palvelut.




KUVA 16. Olio-ohjelmoinnissa luokkia voidaan periyttää

Olio-ohjelmoinnissa voidaan usein muodostaa rajapintaluokkia (interface class) sekä abstrakteja luokkia (abstract class). Rajapinta kuvaa luokan palvelut, muttei kuitenkaan sisällä niiden toteutusta. Jokaisen luokan, joka toteuttaa rajapinnan, täytyy sisältää kaikki rajapinnan ilmoittamat palvelut. Luokka voi myös toteuttaa useita eri rajapintoja. Tekemällä rajapinnan, jonka moni erilainen luokka toteuttaa, voidaan näitä luokkia käsitellä yhdessä välittämättä siitä, etteivät ne muuten vastaa toisiaan. Abstrakti luokka on yliluokka, joka voi sisältää tietoja ja palveluita, mutta siitä ei voida luoda oliota. Abstraktista luokasta voidaan luoda olioita ainoastaan periyttämällä. Näin kaikilla periytyville luokilla on abstraktin luokan palvelut. Ero rajapintojen ja abstraktien luokkien välillä on se, että rajapintoja toteuttavien luokkien ei tarvitse periytyä samasta luokasta. (Peltomäki & Malmirae 1999, 156–158.)

### 3.3.2 Olio-ohjelmointi logiikkaohjelmoinnissa

Vuonna 2013 julkaistu IEC 61131-3 -standardin versio esitteli olio-ohjelmoinnista tuttuja piirteitä logiikkaohjelmointiin. Standardissa oliopohjaiseksi luokaksi valittiin sekä ennestään tuttu toimilohko että uusi rakenneyksikkö luokka (Class) (IEC 61131-3 2013, 118, 146). Luokkaa eivät kuitenkaan CODESYS-pohjaiset järjestelmät tällä hetkellä tue (Hansen 2015, 187). Luokka ja toimilohko ovat muuten samanlaisia, mutta luokka ei itsessään sisällä ohjelmakoodia vaan sitä käsitellään ainoastaan metodien avulla (IEC 61131-3 2013, 120). Standardissa kaikki toimilohkoihin liittyvät oliopohjaiset ominaisuudet ovat määritelty valinnaisiksi, eli niiden käyttö on täysin valinnaista logiikkaohjelmoinnissa (IEC 61131-3 2013, 146). Standardin uusin versio esittelee periytymisen mahdollisuuden toimilohkoille ja luokille (IEC 61131-3 2013, 127, 150). Lisäksi CODESYS-pohjaiset järjestelmät tukevat STRUCT-tyyppisten johdettujen tietotyyppien periytymistä, joten tietorakenteet voivat myös olla monitasoisia.

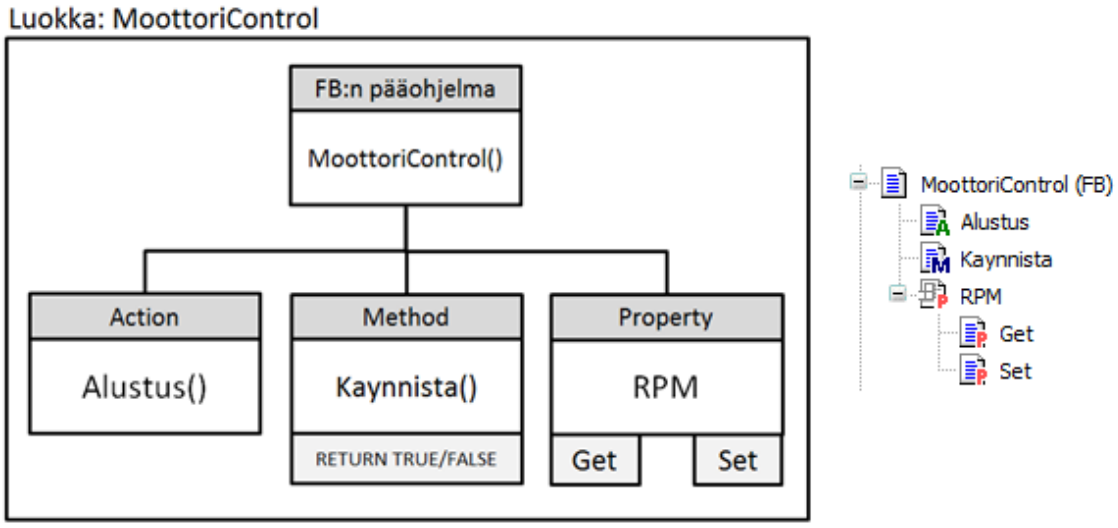
Ennen standardin kolmatta versiota toimilohkot suorittivat vain sisältämäänsä ohjelmakoodia. Uuden standardin myötä toimilohkoille voidaan lisätä metodeja (method), jotka ovat toimilohkon eli luokan sisältämiä funktioita. Metodeja voidaan kutsua toimilohkon ulkopuolelta ja ne voivat palauttaa arvoja sekä käsitellä toimilohkon arvoja (IEC 61131-3 2013, 149–150). Olio-ohjelmoinnin tiedon piilotuksen periaatteen mukaisesti toimilohkon muuttujiin ei pääse käsiksi ulkopuolelta ilman tietoa tarjoaa palvelua eli esimerkiksi metodia.

CODESYS-pohjaisissa järjestelmissä voidaan toimilohkoihin lisätä myös SFC-kielestä tuttuja toimintoja (action) sekä muista olio-ohjelmointikielistä tuttuja ominaisuuksia (property), joita IEC-standardissa ei ole määritelty. Toiminnot ovat vastaavia kuin metodit, mutta ne eivät palauta mitään vaan suorittavat ainoastaan toimilohkoon vaikuttavaa ohjelmakoodia. Lisäksi niiden avulla voidaan hajauttaa ohjelmakoodia lukuisiin eri osiin selventäen ja lyhentäen näin toimilohkon sisäistä ohjelmakoodia. Ominaisuuksien avulla luokan halutuista tiedoista voidaan tehdä julkisia niiden sisältämien get- ja set-metodien avulla.

Standardi mahdollistaa myös rajapintaluokkien käyttämisen ohjelmoinnissa. Rajapintaluokkia voidaan luoda INTERFACE-tyypillä ja rajapintaluokat voivat periytyä toisista rajapintaluokista (IEC 61131-3 2013, 137, 143). Rajapintaluokat voivat määritellä me-

todeja sekä tietoja, jotka rajapinnan toteuttavan luokan täytyy toteuttaa. Rajapintojen avulla erilaisia toimilohkoja voidaan käsitellä yhdessä sillä ne kaikki toteuttavat samat metodit ja sisältävät samat ominaisuudet. Luomalla esimerkiksi funktio, joka ottaa sisääntulona rajapintaluokan, voi funktio käsitellä mitä tahansa rajapinnan toteuttavaa luokkaa.

Kuvassa 17 on esimerkki logiikkaohjelmoinnissa käytettävästä luokasta. Kuvassa on luokan toteutus sekä hahmotelmana että CODESYS-ympäristön puurakenteessa. Luokka on toteutettu toimilohkolla ja sille on annettu toiminto Alustus, metodi Kaynnista sekä ominaisuus RPM. Toimilohkon pääohjelmassa voidaan suorittaa esimerkiksi lukitusten tarkastusta sekä moottorin tilakonetta, kun taas ulkopuolelta luokan toimintaa ohjataan määritetyillä toiminnoilla. Luokasta tehdään olio luomalla toimilohkosta oma instanssi, jolloin olion tietoihin päästään käsiksi edellä mainittujen liityntöjen avulla.




KUVA 17. Toimilohkolla toteutettu luokka hahmoteltuna ja CODESYS-ympäristössä

Aikaisemmin vastaavalle ohjaustoimilohkolle nopeuskäsky olisi annettu joko sisääntulomuuttujana tai globaalin muuttujan kautta. Nyt tieto voidaan antaa suoraan RPM-ominaisuuden avulla. Lisäksi get- ja set-metodeihin voidaan sisältää ohjelmakoodia, joka tarkistaa syötteen oikeellisuuden ja mahdolliset lukitukset (kuva 18). Oliopohjaisil-

la tekniikoilla toimilohkojen toiminnot saadaan pidettyä toimilohkon sisällä, jolloin myös ohjelmakoodia on helppo siirtää projektista toiseen.

```
//RPM-ominaisuuden Get-metodi

RPM := THIS^.nykyinenRPM;
```

```
//RPM-ominaisuuden Set-metodi

//Aseta RPM-pyyntö jos annettu luku on rajojen sisällä
IF RPM < THIS^.RPM_Max AND RPM > THIS^.RPM_Min THEN
    THIS^.haluttuRPM := RPM;
END_IF
```

KUVA 18. Ominaisuuksien sisältämät get- ja set-metodit voivat sisältää ohjelmakoodia

# 4 ETHERNET-POHJAINEN KENTTÄVÄYLÄ

Tässä kappaleessa käsitellään Ethernet-pohjaisia kenttäväyliä yleisesti sekä käydään läpi niiden hyötyjä. Lisäksi kappaleessa perehdytään tarkemmin työn kohteena olleessa järjestelmässä käytettyyn EtherCAT-kenttäväylän toimintaperiaatteeseen.

## 4.1 Kenttäväylät ja Ethernet

Digitaalinen tiedonsiirto on yleistynyt automaatiossa 1990-luvulta lähtien. Aikaisemmin käytettiin pääasiassa lähinnä analogisia signaaleja, kuten pneumaattisia viestejä ja milliampeerisignaaleja. Ensimmäiset digitaaliset väylät olivat hitaita eikä niillä siirretty suurta määrä tietoa. Kuitenkin kehityksen edetessä digitaalisista kenttäväylistä alkoi muodostua yleisimmin käytetty tekniikka. Digitaalisia kenttäväyliä käyttämällä järjestelmän osia voidaan hajauttaa helposti lähemmäksi laitteita, jolloin järjestelmää ei ohjata ainoastaan yhdestä paikasta. (Sundquist 2008, 28.)

Yhä useampi digitaalinen automaatioväylä perustuu Ethernet-verkkoon, joka on ollut yksityiskäytössä jo pitkään. Ethernetin käytön avulla on mahdollista yhdistää automaatiolaitteet TCP/IP-protokollaan, jota käytetään yleisesti tiedonsiirrossa. Näin automaatiojärjestelmään voidaan yhdistää www-liitäntä esimerkiksi konfigurointikäyttöä varten. Lisäksi Ethernetiä käyttämällä saavutetaan suuri hyöty kustannuksissa, sillä väylää voidaan käyttää perinteisillä Ethernet-kaapeleilla ja komponenteilla, jotka saattavat löytyä järjestelmästä jo valmiiksi. Ethernet-pohjaisia väyliä on markkinoilla lukuisia ja niiden takana on yleensä useita yrityksiä. Tunnetuimpia teollisuus-Ethernet-ratkaisuja ovat EtherCAT, Ethernet/IP sekä PROFINET. (Sundquist 2008, 61, 70.)

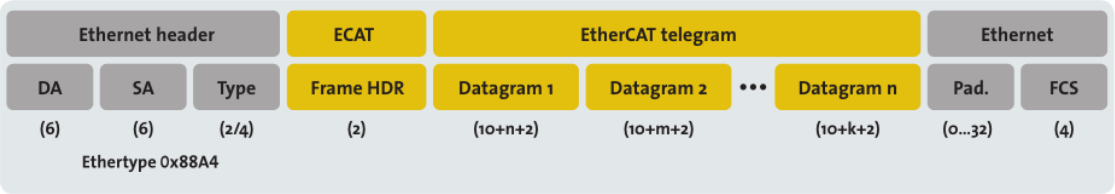
Koska Ethernetissä yleisesti käytetty TCP/IP-protokolla sisältää ja myös sallii pitkiäkin viiveitä, ei se suoraan sovi automaatiokäyttöön. Ethernet-pohjaisissa väylissä käytetäänkin useita erilaisia tekniikoita vasteaikojen pienentämiseen. Liikennettä voidaan esimerkiksi priorisoida siten, että tärkeät viestit ohjataan aina ensimmäiseksi. Muita tapoja ovat muun muassa EtherCAT-väylässä käytetty yhden Ethernet-kehyksen käyttö

tai virtuaalisten kanavien luonti reaaliaikaista liikennettä varten. Usein menetelmät vaativat laitteiden verkkokorteilta erityisominaisuuksia, joten laitteisto ei välttämättä ole täysin yhteensopiva perinteisen Ethernetin kanssa. (Sundquist 2008, 68–70.)

## 4.2 EtherCAT

EtherCAT (Ethernet for Control Automation Technology) on Beckhoffin kehittämä ja nykyisin EtherCAT Technology Group:in hallitsema Ethernet-pohjainen kenttäväylä. Reaaliaikaisessa Ethernetissä tiedon katoaminen ei ole sallittu jo pelkästään turvallisuussyistä, mutta EtherCAT-väylän vasteajat ovat hyvin pieniä. EtherCAT:in viiveet ovat yleensä alle 100µs kun taas Ethernetissä ne saattavat olla jopa 100ms (Sundquist 2008, 68, 71). Pienet mikrosekuntiluokan vasteajat ovat tärkeitä erityisesti teollisuuden liikeohjauksissa sekä silloin, kun mittausten näytteistyksen halutaan olevan tiuha. Työssä tehdyssä laboratoriolaitteistossa nopeasta vasteajasta on hyötyä lukuisten nopeiden mittausten takia.

EtherCAT-väylän tiedonsiirto perustuu fyysisellä tasolla Ethernetiin, mutta tieto siirretään Ethernet-kehyksen sisällä omalla protokollalla. Väylässä lähetetään ainoastaan yksi Ethernet-kehys, joka sisältää kaiken väylällä kulkevan tiedon EtherCAT-sähkeen (telegram) sisällä (kuva 19). EtherCAT-sähke koostuu yhdestä tai useammasta EtherCAT-paketista (datagram), jotka ovat osoitettu tietyille orjalaitteille. Paketti voi sisältää yhden tai useamman laitteen tiedot riippuen laitteistokokoonpanosta. (EtherCAT – the Ethernet Fieldbus 2012, 11–12.)




KUVA 19. EtherCAT-väylällä kulkevan paketin rakenne (EtherCAT – the Ethernet Fieldbus 2012)

EtherCAT-väylä koostuu yhdestä isännästä (master) ja väylän loput laitteet ovat orjia (slave). Isäntä lähettää EtherCAT-sähkeen, joka kulkee suljetussa segmentissä olevan verkon jokaisen orjan läpi. Jokainen orja lukee kehyksen sähkeen sisällöstä itselleen osoitetun paketin sisällön, kirjoittaa pakettiin omat tietonsa, kasvattaa laskuria sekä lähettää sen välittömästi eteenpäin seuraavalle orjalle. Näin viesti kulkee jokaisen laitteen läpi saavuttaen väylän viimeisimmän laitteen, joka käsittelee viestin vastaavasti ja lähettää sen takaisin samaa reittiä. Lopulta viestin saavuttaessa isännän pystytään laskurin avulla tarkistamaan viestin perillepääsy sekä saadaan väylälaitteiden lähettämä data isännän käyttöön. Yksi EtherCAT-väyläsykli kestää parhaimmillaan ainoastaan 30 µs. (EtherCAT – the Ethernet Fieldbus 2012, 11–12.)

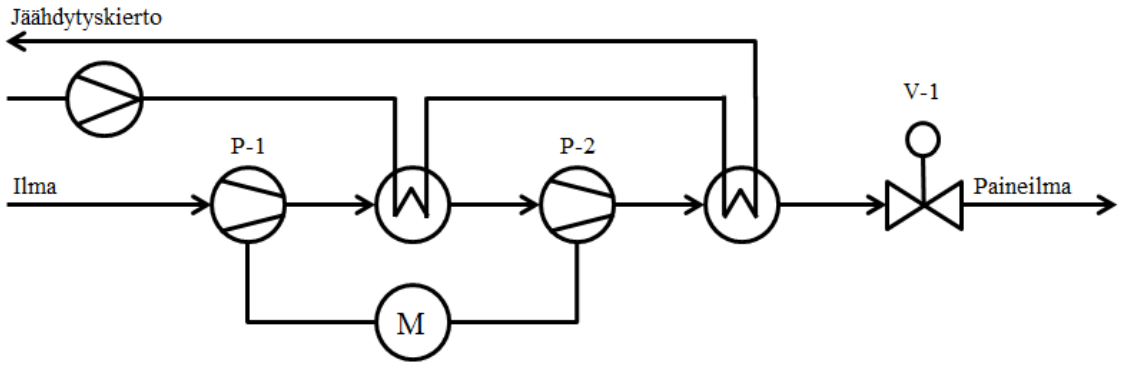
EtherCAT-isäntänä voi toimia periaatteessa mikä tahansa Ethernet-kortin omaava laite, kunhan valmistajan ajurit tukevat reaaliaikaisia toimintoja, sillä EtherCAT ei vaadi rautatasolta mitään ylimääräistä. Esimerkiksi Intelin Ethernet-piireissä on usein mahdollisuus kytkeä reaaliaikaiset ominaisuudet BIOS:in kautta käyttöön. Orjana toimivat laitteet vaativat puolestaan erityisen EtherCAT-väylään tarkoitetun EtherCAT Slave Controllerin (ESC), joka voi olla esimerkiksi mikrokontrolleri. ESC hoitaa nopeasti paketin lukemisen ja kirjoittamisen sekä tiedon siirtämisen joko laitteen keskusyksikölle tai suoraan I/O-rajapintaan. (EtherCAT – the Ethernet Fieldbus 2012, 32, 34.)

# 5 KOMPRESSORIYKSIKÖN AUTOMAATIO-OHJELMOINTI

Työssä kehitettiin teollisuuden paineilmakompressoriyksikköä ohjaavaa logiikkaohjelmaa sekä järjestelmän käyttöliittymää. Erityisesti työn tarkoituksena oli toteuttaa tutkimuslaboratoriossa käytettävä logiikkaohjelmisto sekä laboratorion ohjaukseen käytettävä käyttöliittymäohjelmisto. Tässä kappaleessa käydään läpi kompressoriyksikön rakenne ja käytetyt ohjelmoitavat logiikat sekä laboratoriossa käytössä olleen EtherCAT-kenttäväylän rakenne. Kappaleessa käydään läpi myös logiikkaohjelmiston suunnittelu ja toteutus sekä erityisesti laboratorioon suunniteltujen erityisominaisuuksien toiminta.

## 5.1 Järjestelmä

Työssä käytetty kompressoriyksikkö koostui suurnopeusmoottorista ja sen pyörittämästä kahdesta eri pumpusta. Molempien pumppujen jälkeen sijaitsivat jäähdyttimet, joiden avulla ilman lämpötilaa laskettiin pumppujen tuottaman lämmön takia. Kompressorin lähtevää painetta ohjattiin moottorin kierroksilla sekä venttiilillä, joka päästi painetta ulos tarvittaessa. Kuvassa 20 on järjestelmän rakenne esitetty PI-kaaviona, josta nähdään järjestelmän pääkomponentit. Koska moottori pyöri suurella kierrosnopeudella, oli järjestelmässä käytössä sähköisesti ohjattavat magneettilaakerit kitkahäviöiden vähentämiseksi ja huoltovälin pidentämiseksi.




KUVA 20. Järjestelmä koostui moottorista, pumpuista, venttiilistä ja jäähdytyskierrosta

Järjestelmä sisälsi lukuisia mittauksia. Ilman lämpötilaa, painetta ja virtausta mitattiin ennen pumppuja, pumppujen välillä sekä paineilman ulostulossa. Suurnopeusmoottoria ohjattiin taajuusmuttajalla ja lisäksi moottorin magneettilaakereille oli oma ohjainpiiri.

Laboratoriojärjestelmä vastasi muuten kompressoriyksikköä, mutta laboratoriossa oli käytössä enemmän mittauksia. Lisäksi laboratoriossa vaatimuksena oli helppo uusien väliaikaisten mittausten lisäys ja niiden näkyminen käyttöliittymässä sekä erilaisten logiikkaohjelman muuttujien pakottaminen käyttöliittymästä. Laboratoriossa oli käytössä erillinen mittakaappi, joka sisälsi I/O-kortit kaikille järjestelmään liittyville antureille, sekä erillinen ulkoisia ohjauksia varten oleva kaappi, joka sisälsi pistorasioiden ohjaamiseen tarkoitettuja releitä sekä niiden I/O-kortit.

## 5.2 Logiikkalaitteisto

Kompressoriyksikössä järjestelmää ohjasi Beckhoffin valmistama teollisuus-PC:n ja kosketusnäytön yhdistelmä CP2607. Laite on varustettu 1 GHz ARM Cortex -prosessorilla, 1 gigatavun RAM-muistilla sekä seitsemän tuuman monikosketusnäytöllä. Käyttöjärjestelmänä on Microsoft Windows Embedded Compact 7, jossa ajetaan TwinCAT 3 -runtime-sovellusta. Laitteessa on kaksi RJ45-liitäntää, joista toinen toimi EtherCAT-väyläliitäntänä ja toinen tavallisena Ethernet-liitäntänä. Kahden liitännän ansiosta laite oli samanaikaisesti yhteydessä lähiverkkoon ja internetiin sekä EtherCAT-väylän laitteisiin.

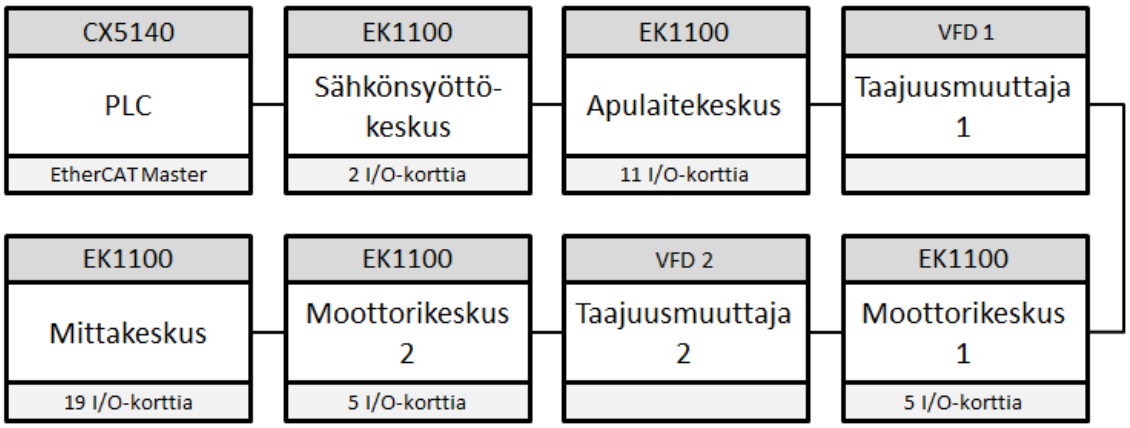

KUVA 21. Beckhoff CX5140 on neliytiminen sulautettu PC vaativiin ohjauksiin (CX51x0 Hardware Manual 2016)

Laboratoriossa järjestelmän ohjauksessa käytettiin ohjelmoitavana logiikkana Beckhoffin tehokasta sulautettua teollisuus-PC:tä CX5140 (kuva 21). CX5140 on varustettu Intel Atom E4856 1,91 GHz -neliydinprosessorilla, 4 gigatavun RAM-muistilla sekä Microsoft Windows Embedded 7 -käyttöjärjestelmällä. Laitteessa on neljä USB-liitäntää, kaksi RJ45-liitäntää sekä DVI-liitin näytön kytkemistä varten. Laite tukee TwinCAT 3 -järjestelmää, jonka avulla laitteen neljä ydintä voidaan ottaa hyötykäyttöön jakamalla PLC-tehtäviä omille ytimilleen. CX5140 tukee myös EtherCAT-väylää ja sen voi konfiguroida kumpaan tahansa verkkokorttiin. Lisäksi laitteeseen voidaan kytkeä EtherCAT-väylää käyttäviä Beckhoffin I/O-kortteja sekä erillistä adapteria käyttämällä K-väylässä toimivia I/O-kortteja. (CX51x0 Hardware Manual 2016, 9, 13.)

### 5.3 Kenttäväylä

Kompressoriyksikössä sekä laboratoriojärjestelmässä käytettiin kenttäväylänä EtherCAT-väylää. Laboratoriossa väylän isäntänä toimi CX5140, johon ei kytketty ollenkaan I/O-kortteja, vaan kaikki I/O:t olivat etä-I/O:na EtherCAT-väylällä. Väylässä topologiana käytettiin perinteistä lineaarista väylää, eli väylä kulki laitteelta toiselle päättyen viimeiseen laitteeseen. Järjestelmässä oli logiikkakeskuksen lisäksi viisi eri keskusta, joissa jokaisessa sijaitsi Beckhoffin EK1100-väylälaite sekä kahdessa keskuksessa väylään kytketty taajuusmuuttaja (kuva 22). EK1100:n avulla EtherCAT-väylään voidaan lisätä Beckhoffin E-väylän I/O-kortteja ja siinä on kaksi Ethernet-porttia väylän jatkamista varten.

EtherCAT-väylän ensimmäinen orjalaite oli sähkönsyöttökeskuksessa sijainnut EK1100, jossa sijaitsivat järjestelmän hätä-seis-toiminnoista ja turvaoven toiminnasta vastanneet turvareleet sekä pääkytkin. Releiden ja pääkytkimen tilatiedot olivat johdotettu väylälaitteeseen kytketyille I/O-korteille. Väylä jatkoi seuraavaksi apulaitekeskukselle, jossa sijaitsivat erilaisten apulaitteiden kytkemistä varten olevien 1- ja 3-vaihepistorasioiden ohjausreleet. Releiden kelat olivat johdotettu I/O-korteille ja releiden apukoskettimista johdotettiin myös tilatiedot digitaalisille sisääntulokorteille.



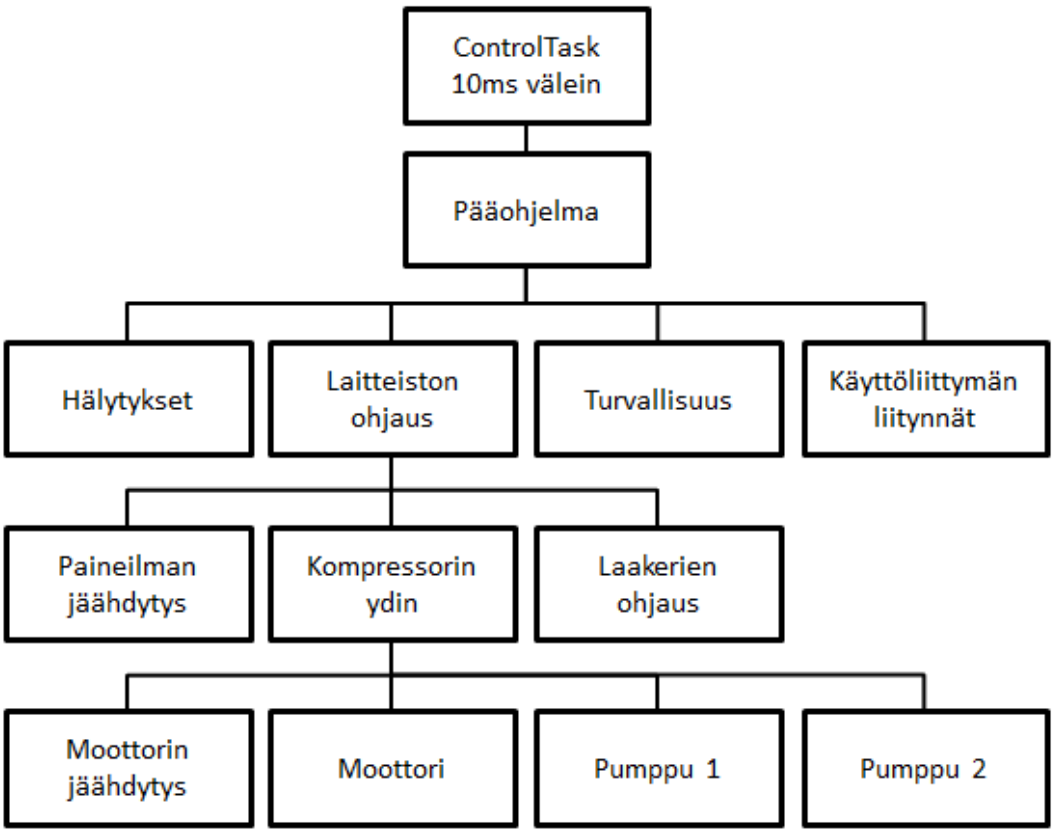

KUVA 22. Laboratorion EtherCAT-väylässä oli yhteensä kahdeksan laitetta

Apulaitekeskuksesta väylä jatkui kahdelle moottorikeskukselle, jotka olivat lähes identtisiä. Keskuksien taajuusmuuttajat (VFD, Variable-Frequency Drive) mahdollistivat eritehoisten moottorien ajamisen, joten jompikumpi oli käytössä tilanteen vaatimusten mukaisesti. EtherCAT-väylä kulki molemmissa kaapeissa taajuusmuuttajien sekä EK1100-väylälaitteiden läpi. Taajuusmuuttajien kommunikointi onnistui suoraan väylän avulla ja I/O:n tehtäväksi jäi laakereiden ohjaaminen sekä laakerien ohjausyksikön antamien tilatietojen välittäminen.

Moottorikeskusten jälkeen väylän viimeinen laite oli mittakeskuksessa sijainnut EK1100, jossa oli yhteensä 19 I/O-korttia. Mittakeskuksessa sijaitsivat kaikki laitteiston mittauksiin liittyvät kortit ja muuntimet sekä laitteen ohjaukseen vaadittavat analogia- ja digitaalilähdöt. Suurin osa korteista oli Beckhoffin EL3024 4..20 mA -analogiasisääntulokortteja, joihin kytkettiin paine- ja virtausanturit. PT100-lämpötila-anturit kytkettiin nelikanavaiseen Beckhoffin EL3204-resistanssinmittauskorttiin. Loput kortit olivat Beckhoffin digitaalisia sisään-ja ulostulokortteja sekä analogisia ulostulokortteja.

## 5.4 Logiikkaohjelma

Logiikkaohjelman kehitys aloitettiin jakamalla järjestelmä loogisiin osiin, joista jokaisesta tehtiin oma toimilohko ohjausta varten (kuva 23). Kompressorin ohjaukset jaettiin moottorin jäähdytykseen, moottoriin, pumppuihin, paineilman jäähdytykseen sekä laakerien ohjaukseen. Lisäksi kompressorin päätoiminnot eli moottori, moottorin jäähdytys sekä pumput koottiin yhden pääohjaustoimilohkon alle.




KUVA 23. Sovellussuunnittelu aloitettiin jakamalla järjestelmä loogisiin osiin

### 5.4.1 Logiikan tehtäväjaottelu

Logiikkaan määriteltiin TwinCAT 3 -ympäristössä kolme eri tehtävää eli taskia (kuva 24). Kompressorin ohjaukset koottiin yhden päätehtävän alle, jonka sykliajaksi asetettiin 10 ms. Järjestelmään kehitettiin lokitietojen tallennus, jonka tarkoitus oli tallentaa häiriötilanteessa järjestelmän tiedot ennen ja jälkeen häiriön. Ohjelmaa kutsuttiin omassa tehtävässään 100 ms sykliajan välein, jotta tiedostoon kirjoittaminen ei häiritsisi

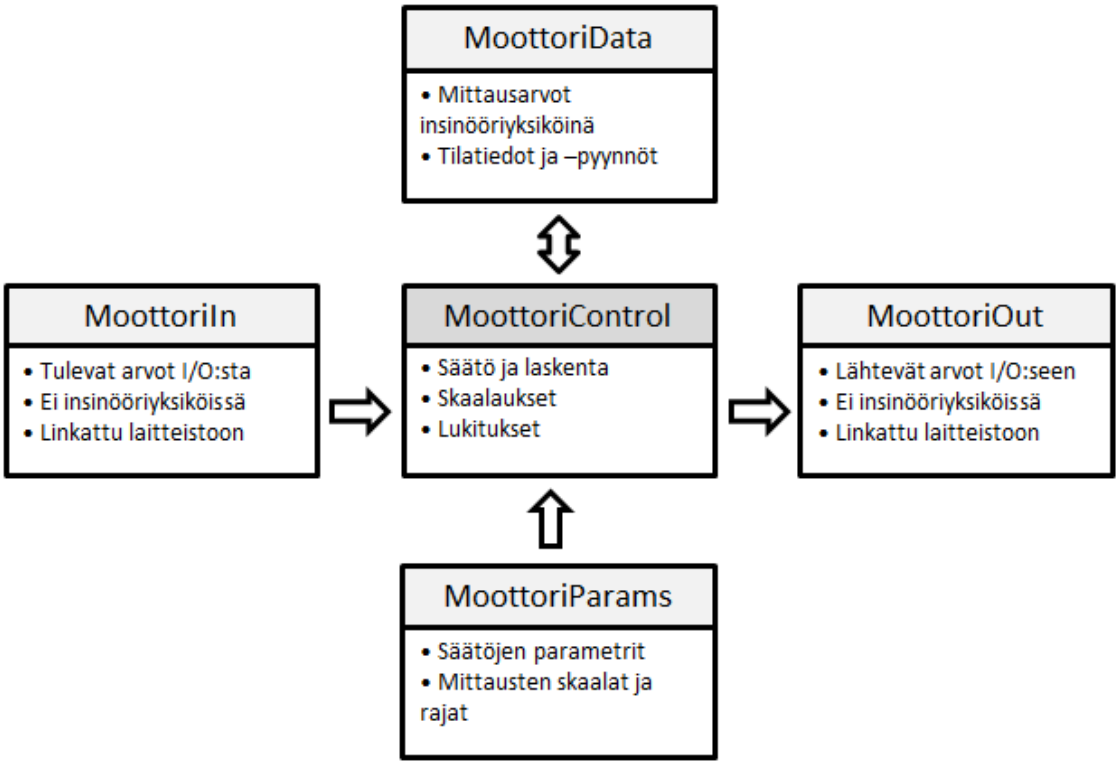
kompressorin ohjausta. Kolmas tehtävä määritettiin Inspector-tiedonkeruujärjestelmää varten, jotta tiedon kerääminen ja lähettäminen verkon yli eivät häiritsisi kompressorin ohjausta. Tiedonkeruutehtävän sykliajaksi asetettiin 100 ms. Jokaiselle tehtävälle määritettiin oma prioriteetti: kompressorin ohjaukselle 1, lokiin kirjoitukselle 2 ja Inspectorille 3. Näin kompressorin ohjausohjelma suoritettiin aina tärkeimpänä ennen muita.

| Object | RT-CPU | Base Time (ms) | Cycle Time (ms) | Cycle Ticks | Priority |
|---|---|---|---|---|---|
| ControlTask | Default (0) | 1 ms | 10 ms | 10 | 1 |
| LoggerTask | Default (0) | 1 ms | 100 ms | 100 | 2 |
| InspectorTask | Default (0) | 1 ms | 100 ms | 100 | 3 |

KUVA 24. Logiikkaan määriteltyjen tehtävien sykliajat sekä prioriteetit

### 5.4.2 Tietorakenteet ja ohjaustoimilohkot

Jokaiselle ohjaustoimilohkoa varten luotiin neljä erilaista tietorakennetta: sisääntulot, ulostulot, dynaaminen data sekä parametrit. Tietorakenteet toteutettiin standardin rakenteisen tietotyypin STRUCT avulla. Koska IEC-standardi ei salli toimilohkon sisäisten muuttujien käyttöä lohkon ulkopuolelta, määriteltiin tietorakenteista instanssit globaaleihin muuttujiin. Näin mistä tahansa ohjelman osasta oli mahdollista lukea ja tarvittaessa muuttaa minkä tahansa järjestelmän laitteen tietoja, kuten skaalattuja mittauksia sekä parametreja. Kolme eri tietorakenteen ansiosta raakoina arvoina esitettävät sisääntulot, skaalatut dynaamiset tiedot sekä raakoina arvoina esitettävät ulostuloarvot erotettiin selkeästi toisistaan (kuva 25).




KUVA 25. Järjestelmän moottoriohjauslohko ja siihen liittyvät tietorakenteet

Sisääntulotietorakenteeseen (kuvassa MoottoriIn) sisällytettiin kaikki kyseisen ohjaustoimilohkon tarvitsemat I/O:sta luettavat tiedot raakoina arvoina. Näin ollen digitaalinen diskreetti binääritieto esitettiin BOOL-tyyppisenä ja analogiatieto WORD-tyyppisenä muuttujana. Muuttujat linkitettiin I/O-kortteihin ja niiden arvot eivät olleet vielä skaalattuja insinööriyksiköihin vaan analogiasignaalit olivat pääasiassa välillä 0..32767.

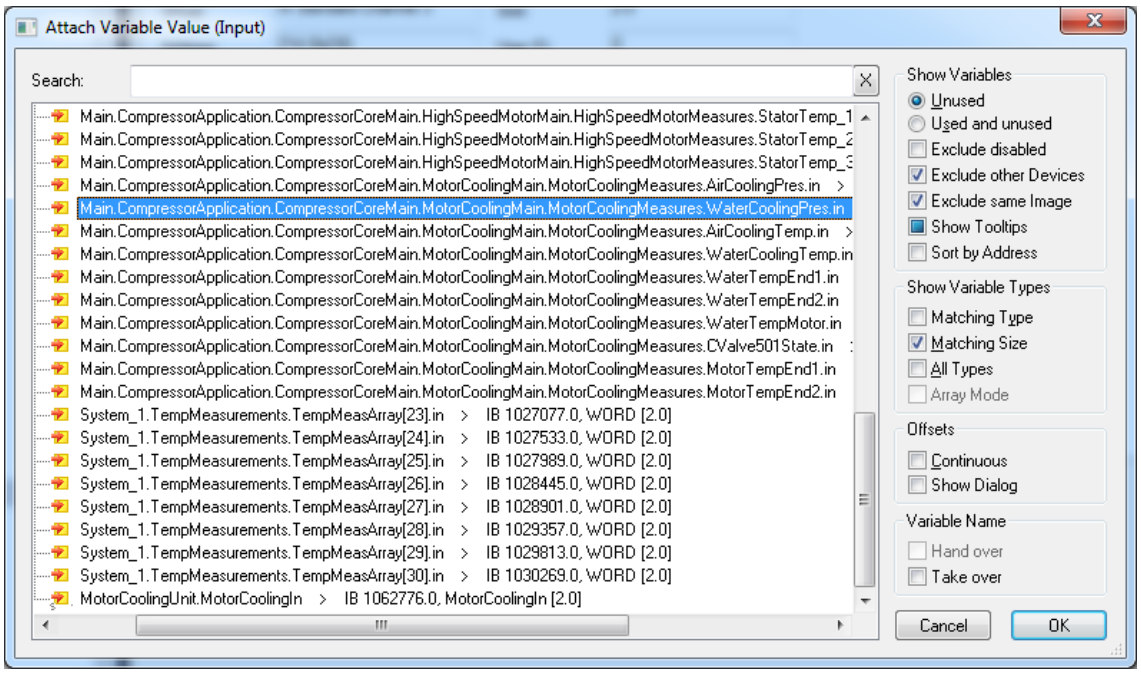
Dynaaminen tietorakenne (kuvassa MoottoriData) sisälsi ohjaustoimilohkon yleisiä tietoja, ohjauspyynnöt ja tilatiedot, mittaukset skaalattuina sekä muita yleisiä lohkoon liittyviä muuttujia. Dynaamisen tietorakenteen idea oli se, että sen sisältämät kaikki muuttujat olivat aina insinööriyksiköissä ja että ne kuvasivat järjestelmän osan senhetkistä tilaa. Ulostulotietorakenteeseen (kuvassa MoottoriOut) sijoitettiin kaikki kyseisen järjestelmän osan I/O-korteille lähtevät tiedot. Muuttujat olivat jälleen skaalaamattomia, kuten sisääntulorakenteessa. Jos lohko esimerkiksi ohjasi proportionaaliventtiiliä, oli ulostulotietorakenteessa lähtevän ohjauksen arvo välillä 0..32767 ja dynaamisessa tietorakenteessa ohjauksen arvoa käsiteltiin prosentteina välillä 0..100 %.

Jokaiseen dynaamiseen tietorakenteeseen lisättiin lohkon ohjaamista varten kaksi INT-tyyppistä muuttujaa. FunctionRequest-muuttujalla pyydettiin lohkolta jotain tiettyä toimintoa ja FunctionResponse-muuttuja kertoi järjestelmän senhetkisen tilan. FunctionRequestin avulla voitiin laite ohjata esimerkiksi päälle tai pois mistä tahansa logiikkaohjelman sisällä. Yleisesti laite ohjattiin pysäytystilaan arvolla 0 ja täyteen automaattiseen toimintatilaan arvolla 100. Loput arvot olivat pääasiassa pysäytyksen ja täyden toimintatilan välitiloja.

### 5.4.3 Mittaukset

Mittaukset toteutettiin luomalla mittaustoimilohko (Measure) sekä tietorakenteet mittauksen parametreille (MeasureParams) sekä mittaustuloksille (MeasureData). Measure-lohkon tehtävä oli lukea sisääntulosta annettu mittausarvo ja skaalata se parametrien mukaan insinööriyksikköön. Measure-lohkossa oli lisäksi mittausten ylärajojen (High, High-High) sekä alarajojen (Low, Low-Low) tarkastelu, jossa skaalattua mittausarvoa verrattiin parametreissa annettuihin rajoihin. Mikäli raja ylitettiin, asetettiin järjestelmään hälytys.

Mittauslohko yhdistettiin sisääntuloon lisäämällä lohkon sisäisiin muuttujiin tyyppiä WORD oleva muuttuja In, joka linkitettiin sisääntulokorttiin. Tämä tehtiin TwinCAT 3 -ympäristössä lisäämällä muuttujan määrittelyyn ”AT %I*”, joka kertoi kääntäjälle muuttujan saavan arvonsa sisääntuloprosessikuvasta linkityksen kautta. Kun mittaustoimilohkosta tehtiin instanssi jokaista mittausta varten, voitiin kyseisen mittauksen sisääntulo yhdistää EtherCAT-väylän analogiakortin kanavaan. Linkitys tapahtui kaksoisklikkaamalla puurakenteessa analogiakortin kanavan alla olevaa kohtaa Value ja valitsemalla haluttu muuttuja aukeavasti ikkunasta (kuva 26). Kuvasta nähdään myös, että linkattava muuttuja in sijaitsee moottorijäähdytyksen ohjaustoimilohkossa (MotorCoolingMain) sijaitsevan mittaukset sisältävän lohkon (MotorCoolingMeasures) sisällä ja mittauslohkon instanssin nimi on WaterCoolingPres eli jäähdytysveden paineenmittaus.




KUVA 26. Mittauksen muuttujan linkitys analogiakortin sisääntuloon TwinCAT 3 -järjestelmässä

Mittauksia varten jokaista järjestelmää ohjaavaa toimilohkoa kohti lisättiin oma toimilohko, joka hoiti kaikki mittaukset. Kyseisen lohkon alle luotiin jokaista mittausta varten oma instanssi Measure-lohkosta, jota kutsuttiin jokaisella PLC-syklillä. Jokaista mittausta varten luotiin MeasureParams-rakenteesta oma instanssi kyseisen järjestelmän osan parametreihin ja vastaavasti mittaustulokset sisältävistä MeasureData-rakenteista luotiin instanssit dynaamiseen tietorakenteeseen. Mittausten parametrit asetettiin halutuiksi logiikan ensimmäisellä syklillä ajettavassa ohjelmassa.

### 5.4.4 Säädöt

Järjestelmässä tarvittiin PID-säätölohkoa tuotetun paineilman paineen säätöön sekä moottorin jäähdytysvettä ohjaavan venttiilin säätämiseen. Säädössä käytettiin aikaisemmissa projekteissa luotua ohjelmallista PID-säädintä, joka tarvitsi toimiakseen PID-toimilohkon sekä säätimen parametrit sisältävän tietorakenteen. Säädintä kutsuttiin erillisen funktion kautta, joka välitti parametrit toimilohkolle ja hoiti virheentarkastelun.

Jäähdytysveden ohjauksessa säädin toimi ainoastaan PI-säätimenä ilman derivointiosaa, mutta paineenohjauksessa oli myös derivointi käytössä. Säätimien asetusarvot sekä vahvistus, integrointiaika ja derivointiaika asetettiin käyttöliittymässä halutuiksi ja ne säilytettiin logiikan PERSISTENT-muistissa. Näin ollen asetukset pysyivät tallessa myös virtojen katkaisun jälkeen. Säätimen laskennassa käytettävä näytteenottoväli saatiin ohjelman sykliajasta 10 ms.

Säätimiä käytettiin automaattitilassa ainoastaan tarvittaessa. Jäähdytysveden venttiili oli automaatilla aina, kun kompressori oli käynnissä. Kun kompressori ei käynyt, asetettiin manuaalitilaan ja venttiili asetettiin kiinniasentoon. Paineenohjaussäädin asetettiin automaattiasentoon ainoastaan kompressorin moottorin tavoittaessa asetetun kierrosnopeuden.

### 5.4.5 Hälytykset

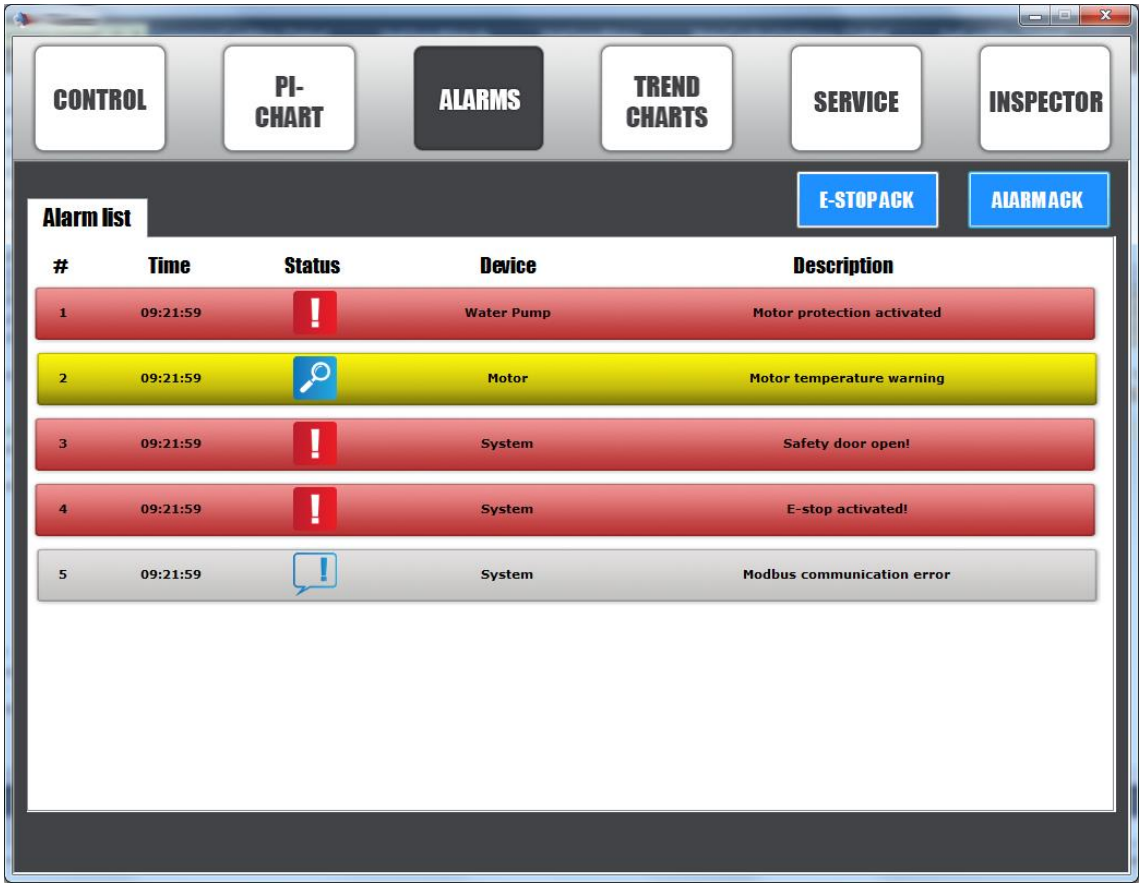
Järjestelmään määriteltiin kolme hälytystasoa: ilmoitus (note), varoitus (warning) sekä virhe (error). Virhetilaa korkeampi tila oli hätä-seis-tila (e-stop), joka asetettiin kun hätä-seis oli aktiivisena, mutta toiminta vastasi virhetilaa. Järjestelmän jokaisella osalle voitiin asettaa oma hälytysryhmä, jonka perusteella hälytysten toimintaan voitiin vaikuttaa. Jos järjestelmän osat olivat eri hälytysryhmissä, ei toiseen ryhmään kohdistunut virhetila vaikuttanut toiseen. Pääasiassa hälytykset vaikuttivat kuitenkin kaikki toisiinsa turvallisuuden takaamiseksi.

Ilmoitus oli tasoista alhaisin ja se ei vaikuttanut toimintaan millään tavalla eikä näkynyt korostetusti käyttöliittymässä. Ilmoitukset näkyivät kuitenkin hälytyslistalla harmaalla värillä ja niiden tarkoitus oli ilmoittaa jostain yleisestä tapahtumasta koneen käyttäjälle.

Varoitukset näkyivät käyttöliittymän tilapalkissa keltaisena huomion herättämiseksi. Varoituksen tarkoitus oli ilmoittaa lähestyvästä virheestä ja varoitukset aiheuttivat järjestelmän hallitun alasajon. Tämän työn kirjoitushetkellä varoituksien kuittaus vaati käyttäjän toimia, mutta mahdollisuus oli myös muuttaa ne kuittautumaan automaattisesti. Mittauksissa H- ja L-parametrien ylitykset aiheuttivat varoituksen, jossa kerrottiin

mittausarvon ylittäneen alemman asetetun rajan. Lisäksi väylässä mahdolliset hetkellistä pidemmät ongelmat aiheuttivat varoituksen.

Myös vakavimmassa virhetilassa järjestelmä ajettiin alas kuten varoituksen tapauksessa. Jos järjestelmään tuli virhe, vaati se aina käyttäjän kuittauksen käyttöliittymästä. Virhetila asetettiin mittauksen ylittäessä HH- tai LL-parametrit sekä erilaisten käynnistyksen aikaisten tarkistusten epäonnistuessa. Hätä-seis-tilassa toiminta oli samanlainen kuin virhetilassa ja sen tarkoitus oli eriyttää hätätoimintoihin liittyvä virhetila tavallisesta virheestä. Kuvassa 27 näkyy käyttöliittymään toteutettu hälytysikkuna, jossa on erityyppisiä ilmoituksia. Punaiset ovat virheitä, keltainen on varoitus ja harmaa ilmoitus.




KUVA 27. Käyttöliittymän hälytysikkuna ja erilaiset ilmoitukset

### 5.4.6 Lokiin kirjoittaminen

Toisessa logiikassa ajettavassa tehtävässä suoritettiin ohjelmaa, joka kirjoitti tarvittaessa lokia. Lokiin kirjoituksen idea on saada talteen järjestelmän muuttujien tilat ja sisältö tarvittaessa. Jos kompressoriyksikkö pysähtyi virheen takia, ei käyttäjällä olisi välttämättä mahdollista selvittää vikaa ilman kunnon lokitietoja. Kirjoitusohjelmaa ajettiin jatkuvasti 100 ms sykliajalla ja tallennettavia muuttujia oli noin 40 kappaletta.

Ohjelma keräsi jokaisella ohjelmasyklillä muuttujien arvoja puskuriin, johon mahtuvien mittausten määrä asetettiin 200 kappaleeseen eli 20 sekuntiin. Kun järjestelmän tila muuttui virhetilaksi, käynnistyi lokiin kirjoittaminen. Samanaikaisesti muuttujien arvojen tallennusta jatkettiin vielä 100 mittauksen eli 10 sekunnin verran. Ohjelma kirjoitti mittaukset csv-tiedostoon, jonka pystyi avaamaan esimerkiksi Microsoft Excel -taulukkolaskentaohjelmalla mittausten tarkastelua varten. Tiedostoon kirjoittaminen voitiin aloittaa myös käyttäjän toimesta käyttöliittymästä löytyvän painikkeen avulla. Lisäksi oli mahdollista kirjoittaa tämänhetkiset muuttujien arvot omaan tiedostoonsa käyttöliittymän painikkeen kautta ja näin ottaa mittaukset ylös aina tarvittaessa.

## 5.5 Laboratorion lisäominaisuudet

Tuotekehityslaboratoriossa operaattorilla on enemmän vapauksia kuin todellisessa käyttöympäristössä. Tästä syystä käyttöliittymään tarvittiin enemmän toimintoja, kuin tavallisen kompressoriyksikön käyttöliittymään. Työssä toteutettiin mahdollisuus ohjata mitä tahansa haluttua BOOL- tai INT-tyyppistä muuttujaa käyttöliittymän kautta sekä mahdollisuus lisätä väliaikaisia mittauksia pienellä ohjelmakoodin lisäyksellä. Kehitysinsinöörillä oli pääsy PLC-ohjelmakoodiin, mutta mittausten lisäämisestä haluttiin mahdollisimman yksinkertaista. Operaattorien työn helpottamiseksi muuttujien pakko-ohjaukset haluttiin myös käyttöliittymään, jotta PLC-muuttujia ei tarvinnut käsitellä TwinCAT-ympäristössä.

### 5.5.1 Ulkoiset ohjaukset

Laboratoriossa oli oma pistorasioita ja niitä ohjaavia kontaktoreita sisältävä sähkökeskus, jonka avulla voitiin ohjata erilaisia ulkoisia pumppuja, moottoreita ja muita laitteita. Laitteita voitiin ohjata käsin laboratorion käyttöliittymästä, mistä kerrotaan lisää kappaleessa 6.4.3. Logiikkaohjelman puolella toteutettiin ominaisuus, jolla ohjauksia saatiin kytkettyä päälle tiettyjen ehtojen täyttyessä. Idea oli se, että jos laitteen haluttiin käynnistyvän samanaikaisesti esimerkiksi jäähdytyspumpun käynnistyessä ohjaussekvenssissä, voitiin se toteuttaa helposti yhdellä riviä ohjelmakoodia. Jos käyttöliittymästä asetettiin ohjaus käsiajolle, voitiin sitä ohjata vapaasti linkityksestä huolimatta.

Ohjaukset toteutettiin luomalla tietorakenne ST_ExternalControl, joka sisälsi ohjaukseen tarvittavat muuttujat (kuva 28). Näitä olivat käyttöliittymän ohjausarvo, tilan valinta sekä lähtevä ohjaus ja paluutieto kontaktorilta. Lähtevä ohjaus ja paluutieto linkitettiin sisään- ja ulostulokorteille. Kaikki järjestelmän ohjaukset koottiin yhden tietorakenteen ST_ExternalControls alle. Ohjauksien hallinta tapahtui omassa ohjelmassa, jonka alussa yhdistettiin Output-muuttuja haluttuihin ehtoihin sekä lopussa tarkasteltiin oliko käyttöliittymän käsiohjaus käytössä.

```
{attribute 'pack_mode' := '1'}
TYPE ST_ExternalControl :
STRUCT
    GuiValue            :   BOOL;
    OverrideFromGui     :   BOOL;
    Output   AT%Q*      :   BOOL;
    Feedback AT%I*      :   BOOL;
END_STRUCT
END_TYPE
```

KUVA 28. Ulkoisen ohjauksen tietorakenne

### 5.5.2 Muuttujien pakko-ohjaus

Laboratorion kehityksen yhteydessä todettiin, että tiettyjen muuttujien käsiohjaus ja pakotus oli tarpeellista. Muuttujia olivat esimerkiksi päävirtakatkaisijan ohjaus sekä jäähdytyspumppujen kontaktorien ohjaukset. Tavallisesti ohjelmasekvenssi käynnisti

laitteet itse, mutta laboratorio-olosuhteissa oli välillä tarpeellista käyttää niitä järjestelmän ollessa muuten pysähdyksissä.

Pakko-ohjauksen kehitys aloitettiin luomalla tietorakenne ST_GuiForceControls, joka sisälsi kaikki mahdolliset pakko-ohjaukset taulukoissa sekä niiden tämänhetkiset lukumäärät, jotka laskettiin joka ohjelmasyklillä. Ohjauksia varten luotiin tietorakenne ST_GuiForceControl, joka sisälsi yhden pakko-ohjaukset tiedot (kuva 29). Tietorakenteessa määriteltiin käyttöliittymässä näytettävä muuttujan nimi, ohjattavan muuttujan osoite, käyttöliittymässä määritelty arvo sekä se, onko pakotus käytössä. Lisäksi tietorakenne sisälsi tiedon lähdön todellisesta arvosta ja mahdollisuuden estää muuttujan muokkaus käyttöliittymästä, jolloin kyseisen muuttujan arvoa ainoastaan monitoroitiin.

```
TYPE ST_GuiForceControlBOOL :
STRUCT
    Name            :   STRING(20);         //Name to show in gui
    pControlledBool :   POINTER TO BOOL;    //Pointer to forced boolean variable
    GuiValue        :   BOOL;               //Given value from gui
    OverrideFromGui :   BOOL;               //Gui wants to override value
    Output          :   BOOL;               //Real output value
    ReadOnly        :   BOOL := FALSE;      //Can only be read from gui
END_STRUCT
END_TYPE
```

KUVA 29. BOOL-tyyppisen muuttujan pakko-ohjauksen tietorakenne

Pakko-ohjaukset toteutettiin osoittimien (pointer) avulla. Osoitin on keskusmuistin muistipaikan osoite, jossa myös muuttujat säilytetään. CODESYS-pohjaisissa järjestelmissä muuttujan muistipaikan osoite saadaan ADR-funktion avulla ja osoitintyyppisen muuttujan voi luoda POINTER TO -tyyppisellä esittelyllä. Osoittimen osoittaman muistipaikan sisältämää arvoa voidaan käsitellä lisäämällä muuttujan perään ^-merkki. Pakko-ohjauksia voitiin lisätä PLC-ohjelmakoodissa kuvan 30 mukaisesti kahden rivin avulla, jonka jälkeen käyttöliittymältä saatiin tieto, pitikö muuttuja pakottaa johonkin tiettyyn arvoon. Itse pakotuksen hoiti PLC-koodissa kuvan mukainen ohjelmakoodi, jossa kaikki käytössä olevat pakko-ohjaukset käytiin läpi FOR-silmukan avulla.

```
GuiForceControls.Controls_BOOL[1].Name := 'Main power';
GuiForceControls.Controls_BOOL[1].pControlledBool := ADR(SystemOut.bMainPower);

FOR i := 1 TO GuiForceControls.ControlCount_BOOL DO
    //Check if pointer is > 0 (else there will be page fault)
    IF GuiForceControls.Controls_BOOL[i].pControlledBool > 0 THEN

        //Is GUI overriding?
        IF GuiForceControls.Controls_BOOL[i].OverrideFromGui THEN
            GuiForceControls.Controls_BOOL[i].pControlledBool^ := GuiForceControls.Controls_BOOL[i].GuiValue;
        END_IF

    //Copy real value to output for GUI
    GuiForceControls.Controls_BOOL[i].Output := GuiForceControls.Controls_BOOL[i].pControlledBool^;
    END_IF
END_FOR
```

KUVA 30. Pakko-ohjauksen määritys sekä niiden toteutus osoittimien avulla

### 5.5.3 Väliaikaiset mittaukset

Koska laboratorio-olosuhteissa antureita lisättiin, poistettiin ja vaihdettiin jatkuvasti, täytyi uuden anturin lisääminen ohjelmakoodiin olla mahdollisimman helppoa. Tämä toteutettiin luomalla tietorakenne, joka sisälsi taulukoissa kappaleessa 5.4.3 esitellyt mittauslohkot sekä tietorakenteet. Kuvassa 31 näkyy tietorakenteen määrittely, jossa vakion TempMeasurementMaxCount avulla voidaan määrittää mittausten maksimilukumäärä, joka asetettiin arvoon 30.

```
TYPE ST_TempMeasurement :
STRUCT
    TempMeasArray       :   ARRAY[1..TempMeasurementMaxCount] OF Measure;
    TempMeasDataArray   :   ARRAY[1..TempMeasurementMaxCount] OF MeasureData;
    TempMeasPrmsArray   :   ARRAY[1..TempMeasurementMaxCount] OF MeasurePrms;
END_STRUCT
END_TYPE
```

KUVA 31. Väliaikaisten mittausten tietorakenne koostui taulukoista

Väliaikaisten mittausten avulla käyttäjä pystyi lisäämään järjestelmään uuden anturin ja siihen liittyvän mittauksen helposti. Anturi liitettiin olemassa olevaan analogiatulokortin sisääntuloon ja kuvassa 32 näkyvä koodi kopioitiin ja liitettiin sekä siihen muokattiin kyseiseen mittaukseen liittyvät parametrit. Lopuksi kyseisen mittauksen sisääntulomuuttuja, esimerkiksi ”TempMeasArray[1].in”, linkitettiin I/O-korttin sisääntuloon ja mittaus oli käytössä.

```
//Measurement 1 - Motor temperature
TempMeasurements.TempMeasPrmsArray[MeasureCount].raw_offset := 0/20.0*32767.0;
TempMeasurements.TempMeasPrmsArray[MeasureCount].Sensor_range_L := 0;
TempMeasurements.TempMeasPrmsArray[MeasureCount].Sensor_range_H := 200;
TempMeasurements.TempMeasPrmsArray[MeasureCount].Sensor_range := 200;
TempMeasurements.TempMeasPrmsArray[MeasureCount].scaling_factor := 200 / 32767.0;
TempMeasurements.TempMeasPrmsArray[MeasureCount].HH := 60;
TempMeasurements.TempMeasPrmsArray[MeasureCount].LL := -1;
TempMeasurements.TempMeasPrmsArray[MeasureCount].H := 55;
TempMeasurements.TempMeasPrmsArray[MeasureCount].L := -1;
TempMeasurements.TempMeasPrmsArray[MeasureCount].lambda := 0.75;
TempMeasurements.TempMeasPrmsArray[MeasureCount].Diagnostic_ON := TRUE;
TempMeasurements.TempMeasPrmsArray[MeasureCount].description := 'Motor temperature';
TempMeasurements.TempMeasPrmsArray[MeasureCount].name := 'TI-102';
TempMeasurements.TempMeasPrmsArray[MeasureCount].group := '1.Motor';
MeasureCount := MeasureCount + 1;
```

KUVA 32. Uuden väliaikaisen mittauksen lisääminen ja parametrointi

Väliaikaisia mittauksia hoitavan toimilohkon lopussa kaikki asetetut mittaukset käytiin FOR-silmukassa läpi ja niiden Measure-lohkoja kutsuttiin mittausten, skaalausten ja hälytysten suorittamiseksi (kuva 33). Koska mittausten läpikäyminen perustui juoksevaan numeroon MeasureCount, huomasi järjestelmä myös poistetut mittaukset automaattisesti.

```
FOR i := 1 TO TempMeasurementMaxCount DO
    IF MeasureCount > i THEN
        //Measure exists, call function block
        TempMeasurements.TempMeasArray[i](
            MeasurePrms:= TempMeasurements.TempMeasPrmsArray[i],
            MeasureData => TempMeasurements.TempMeasDataArray[i]
        );
    ELSE
        EXIT;
    END_IF
END_FOR
```

KUVA 33. Väliaikaiset mittaukset suoritettiin silmukan avulla

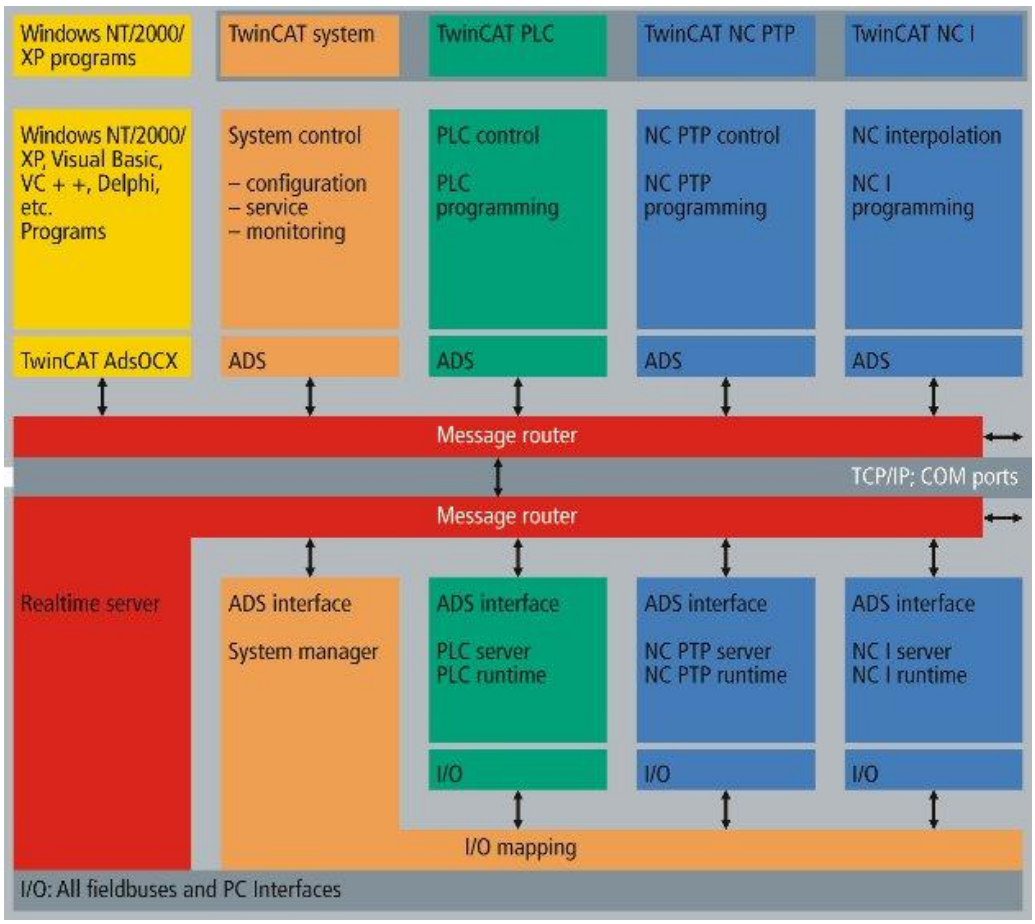
# 6 KÄYTTÖLIITTYMÄ

Järjestelmän käyttöliittymä toteutettiin Visual C# -ohjelmointikielellä, joka on oliopohjainen C++- ja Java-ohjelmointikieliin perustuva Microsoftin kehittämä ohjelmointikieli. Käyttöliittymä ulkoasu toteutettiin Microsoftin Windows Forms -luokkien avulla, jotka tulevat Visual Studion mukana. Windows Forms:in avulla käyttöliittymän graafinen puoli voitiin toteuttaa helposti pääasiassa Visual Studion suunnittelunäkymässä käyttämällä valmiita Windowsista tuttuja elementtejä. Tässä kappaleessa käydään läpi kommunikoinnissa käytetty Beckhoffin ADS-tekniikka, logiikan muuttujien käsittely C#-ohjelmointikielellä sekä laboratorion lisäominaisuuksien toteutus.

## 6.1 TwinCAT ADS

Beckhoff on kehittänyt Automation Device Specification (ADS) -protokollan logiikoiden, NC-ohjauslaitteiden, TwinCAT-ohjelmiston sekä muiden ohjelmistojen ja laitteiden väliseen kommunikointiin (kuva 34). ADS-protokolla on laiteriippumaton eikä myöskään välitä käytetystä väylätekniikasta, joten se voi toimia esimerkiksi TCP/IP-protokollan päällä. Näin samassa verkossa olevat laitteet voivat vaihtaa tietoja keskenään ADS-protokollan avulla. Tietojen välittämisestä vastaa reititin (ADS Router), joka hoitaa datan välittämisen laitteiden välillä. Windows-ympäristössä reititin toimii omana palveluna käyttöjärjestelmässä. (ADS Introduction.)

Jokaisella laitteella on ADS-kommunikointia varten oma yksilöllinen AmsNetId, joka on kuin Ethernetistä tuttu IP-osoite. AmsNetId on jokaisella laitteella oletuksena kyseisen laitteen IP-osoite sekä sen perässä ”1.1” eli IP-osoitteen 192.168.1.100 omaavan laitteen AmsNetId on oletuksena 192.168.1.100.1.1. AmsNetId ei kuitenkaan vaihdu välttämättä laitteen IP-osoitteen muuttuessa, joten se on syytä tarkastaa aina kokoonpanon muuttuessa. AmsNetId:tä käytetään laitteeseen yhdistäessä sekä TwinCAT-ympäristössä että esimerkiksi käyttöliittymän yhdistämisessä logiikkaan.




KUVA 34. ADS-protokollan avulla eri järjestelmät voivat keskustella keskenään (Beckhoff ADS Introduction)

Beckhoff tarjoaa useille eri ohjelmointiympäristöille kirjastoja ADS-kommunikointia varten. Tässä työssä käytettyyn .NET-ympäristöön löytyy luokkakirjasto ”TwinCAT.Ads”, jonka avulla esimerkiksi C#-ohjelmointikielellä voidaan ottaa yhteys ADS-protokollaa tukevaan laitteeseen.

## 6.2 Logiikan muuttujien käsittely C#:lla

ADS-yhteys luotiin C#-projektissa lisäämällä TwinCAT.Ads-kirjasto projektiin. Tämän jälkeen kirjaston nimiavaruus otettiin käyttöön lisäämällä ohjelmakoodin yläosaan ”Using TwinCAT.Ads”, jonka jälkeen kirjaston sisältö oli helposti käytössä. Ohjelman pääikkunaan luotiin olio kirjaston sisältämästä yhteysluokasta TcAdsClient, jonka kautta kaikki kommunikointi suoritettiin.

### 6.2.1 Yhteyden muodostaminen ja syklinen tiedonhaku

Käyttöliittymän käynnistyessä käynnistettiin yhteyden avaamista varten oma ajastin. Ajastin suoritti 400 ms välein yhteydenottoyrityksen logiikkaan luokan sisältämän Connect-metodin avulla. Yhteyden muodostamista yritettiin jatkuvasti, kunnes se saatiin luotua onnistuneesti tai käyttöliittymä suljettiin. Yhteyden muodostamisen jälkeen pääikkunassa sijainnut PlcOK-muuttuja asetettiin todeksi, joka kertoi yhteyden olevan kunnossa.

Käyttöliittymään tiedonhaku toteutettiin syklisesti ajastimilla siten, että jokaisessa ikkunassa oli oma ajastin. Suurimmalle osalle ajastimista asetettiin aikaväliksi 300 ms, joka riitti hyvin tietojen päivittämiseen. Ajastin kutsui funktiota, joka suoritti tietojen hakemisen ohjelmoitavalta logiikalta pääikkunassa olevan yhteysinstanssin avulla. Tärkeää oli tarkastella PlcOK-muuttujaa, sillä jos tietojen hakemista yritettiin yhteyden ollessa poikki, aiheutti se turhaa resurssien käyttöä.

Jokainen ikkuna hoiti omien tietojensa päivittämisen ja ainoastaan silloin, kun ikkuna oli aktiivisena. Tämä vähensi huomattavasti turhaa verkkoliikennettä, joka olisi muodostunut kaiken tiedon jatkuvasta hakemisesta. Pääikkunassa hoidettiin yleisien, koko järjestelmää koskevien tietojen päivittäminen. Näitä olivat esimerkiksi hälytykset sekä järjestelmän tila.

### 6.2.2 Tietojen luku logiikalta

Tietoja voitiin hakea logiikalta joko muuttuja kerrallaan tai kokonaisina tietorakenteina. Pääasiassa suurin osa tiedoista haettiin tietorakenteina siten, että logiikalle tehtiin käyttöliittymää varten omia STRUCT-tyyppisiä tietorakenteita, joiden avulla halutut tiedot saatiin luettua kerralla.

Aina tietoja lukiessa tai kirjoittaessa täytyi kyseiseen muuttujaan luoda oma kahva (handle). Kahvan avulla tietoja voitiin muokata ja hakea omien metodien avulla. Kahva luotiin TcAdsClient-luokan sisältämän CreateVariableHandle-metodin avulla, jolle annettiin parametrina logiikassa sijaitsevan muuttujan osoite. Jos kahvan luonti onnistui, palautti metodi kahvana toimivan int-tyyppisen luvun, joka tallennettiin muuttujaan. Lukemisen lopuksi täytyi kahva muistaa sulkea DeleteVariableHandle-metodilla, mikäli sitä ei enää tarvittu.

Yksittäinen tieto voitiin hakea logiikalta TcAdsClient-luokan sisältämällä ReadAny-metodilla. Metodilla annettiin luotu kahva sekä kyseisen muuttujan tietotyyppi. Jos kyseessä oli merkkijono tai taulukko, täytyi niiden pituus ja koko antaa kolmantena parametrina. Onnistuessaan metodi palautti kyseisen muuttujan arvon. Kuvassa 35 on esimerkki REAL-tyyppisen muuttujan lukemisesta logiikalta. Lukeminen on sijoitettu C#:n try-catch-lohkojen sisälle virheiden käsittelyä varten. Jos lukeminen epäonnistuu, eli esimerkiksi yhteys ei onnistu tai muuttujaa ei löytynyt, suoritetaan catch-lohkon sisältämä ohjelmakoodi.

```
try
{
    //Create handle to PLC variable
    int handle = parent.ads.getTcClient().CreateVariableHandle("MotorGVL.Motor.ActualSpeed");
    //Read current motor RPM
    actualRPM = ((Single)parent.ads.getTcClient().ReadAny(handle, typeof(Single)));
    //Close handle
    parent.ads.getTcClient().DeleteVariableHandle(handle);
}
catch (Exception ex)
{
    //Reading failed, notify user
    ConfirmationForm cf = new ConfirmationForm(this,
        "Error", "Command failed. Error information:" + Environment.NewLine + ex.Message, true);
    cf.ShowDialog(); cf.Dispose();
}
```

KUVA 35. REAL-tyyppisen muuttujan lukeminen ReadAny-metodilla

ReadAny-metodilla voidaan myös lukea tietorakenteita, mutta projektissa käytettiin toista tapaa joka perustui omaan kommunikointiluokkaan. Syy tähän oli se, että ReadAny-funktiolla ei voi lukea tietorakenteita jos käytössä on Microsoftin .NET Compact Framework, joka on Windows CE -tietokoneissa (Reading and writing of PLC variables of any type). Koska projekti aloitettiin aluperin Windows CE -käyttöjärjestelmän omaavalla Beckhoffin laitteella, käytettiin tietojen lukemiseen toisenlaista tapaa.

Jotta tietorakenne voitiin lukea tai kirjoittaa logiikalle, täytyi siitä luoda oma rakenne käyttöliittymän ohjelmakoodin. Tämä tapahtui käyttämällä C#-ohjelmointikielen struct-tyyppiä. Huomioitavaa oli, että tietorakenteiden täytyi olla täysin vastaavia bittitasolla jotta tiedot siirtyivät oikein (kuva 36). Tätä varten tarvittiin C#:n StructLayout- ja MarshalAs-määreitä. Ennen tietorakenteen esittelyä täytyi kääntäjälle kertoa StructLayout-määreen avulla, että tietorakenteen muuttujat tuli asetella keskusmuistiin peräjälkeen. Sama asia täytyi myös muistaa tehdä logiikan tietorakenteen esittelyssä asettamalla pragma eli esikääntäjäkomento ”pack-mode” arvoon 1. MarshalAs-määreen avulla C#-kääntäjää käskettiin käsittelemään muuttujaa tietyllä tavalla. Esimerkiksi bool-tyyppinen muuttuja voi viedä Windows-ympäristössä neljä tavua, kun taas logiikalla se on aina tavun kokoinen. Antamalla bool-muuttujalle määre ”[MarshalAs(UnmanagedType.I1)]” kerrotaan kääntäjälle, että kyseistä muuttujaa tulee käsitellä vain yhden tavun mittaisena.

**Käyttöliittymä:**

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct LaboratoryGuiStruct
{
    //State machine status
    public Int16 CurrentState;
    public Int16 PrevState;
    public Int16 ShutDownState;

    //Commands
    [MarshalAs(UnmanagedType.I1)]
    public bool openValve;

    //Measures
    public Single MotorSpeed;
    public Single OutletPressure;

    //PID
    [MarshalAs(UnmanagedType.I1)]
    public bool PID_in_automode;
}
```

**Logiikka:**

```
{attribute 'pack_mode' := '1'}
TYPE ST_LaboratoryGuiStruct :
STRUCT
    //State machine status
    CurrentState    :   ApplicationState;
    PrevState       :   ApplicationState;
    ShutDownState   :   ShutdownState;

    //Commands
    openValve       :   BOOL := FALSE;

    //Measurements and states
    MotorSpeed      :   REAL;
    OutletPress     :   REAL;

    //PID
    PID_in_automode :   BOOL;
END_STRUCT
END_TYPE
```

KUVA 36. Tietorakenteiden täytyy olla identtisiä lukemisen onnistumiseksi

Tietorakenteen lukeminen tapahtui oman luokan PLCObject sisältämällä metodilla ReadStruct. Kyseinen metodi haki annetusta logiikan osoitteesta tietorakenteen koon verran bittejä ja sijoitti ne annettuun tietorakenteeseen. Jos tietorakenteiden koot eivät olleet identtisiä, antoi luokan metodi myös virheilmoituksen, sillä tällöin luetut tiedot saattoivat olla mitä tahansa. Jokaista haluttua tietorakennetta kohti tehtiin oma luokka, jolla oli oma olio PLCObject-luokasta sekä kyseisen tietorakenteen osoite logiikassa. Tämän jälkeen kutsumalla luokan ReadStr-metodia voitiin tietorakenteen tiedot päivittää logiikan muistista. Kuvassa 37 on esimerkin vuoksi tietorakenteeseen yhdistävän luokan esittely, tietorakenteen esittely sekä tietojen luku logiikalta ReadStr-metodilla.

```
//Connection to struct
TwinCat.LaboratoryGui TcLaboratoryGui = new TwinCat.LaboratoryGui(ads.getTcClient());
//Data
TwinCat.LaboratoryGui.LaboratoryGuiStruct laboratoryGuiData;

//Read struct from PLC
laboratoryGuiData = TcLaboratoryGui.ReadStr();

if (laboratoryGuiData.MotorSpeed > 0)
{
    //Motor is running
}
```

KUVA 37. Tietorakenteen sisällön lukemisen jälkeen tieto oli käyttöliittymän käytettävissä

### 6.2.3 Tietojen kirjoitus logiikkaan

Tietojen kirjoittaminen logiikkaan tapahtui lähes vastaavasti kuin lukeminen. Kirjoittamisessa luotiin samalla tavalla kahva logiikan muuttujaan. Kirjoittaminen tehtiin käyttäen TcAdsClient-luokan WriteAny-metodia, joka toimi kuten lukemiseen käytetty metodi. WriteAny otti parametreina muuttujaan luodun kahvan sekä kirjoitettavan arvon. Lopuksi kahva poistettiin DeleteVariableHandle-metodilla. Kuvassa 38 on esimerkki BOOL-tyyppisen muuttujan arvon kirjoittamisesta logiikan osoitteeseen ”GUI.bRunRequest”.

```
try
{
    //Create handle to PLC variable
    int handle = parent.ads.getTcClient().CreateVariableHandle("GUI.bRunrequest");
    //Write TRUE to variable
    parent.ads.getTcClient().WriteAny(handle, true);
    //Close handle
    parent.ads.getTcClient().DeleteVariableHandle(handle);
}
catch (Exception ex)
{
    //Writing failed, notify user
    ConfirmationForm cf = new ConfirmationForm(this,
        "Error", "Command failed. Error information:" + Environment.NewLine + ex.Message, true);
    cf.ShowDialog(); cf.Dispose();
}
```

KUVA 38. Tietojen kirjoittaminen logiikkaan tapahtui WriteAny-metodilla

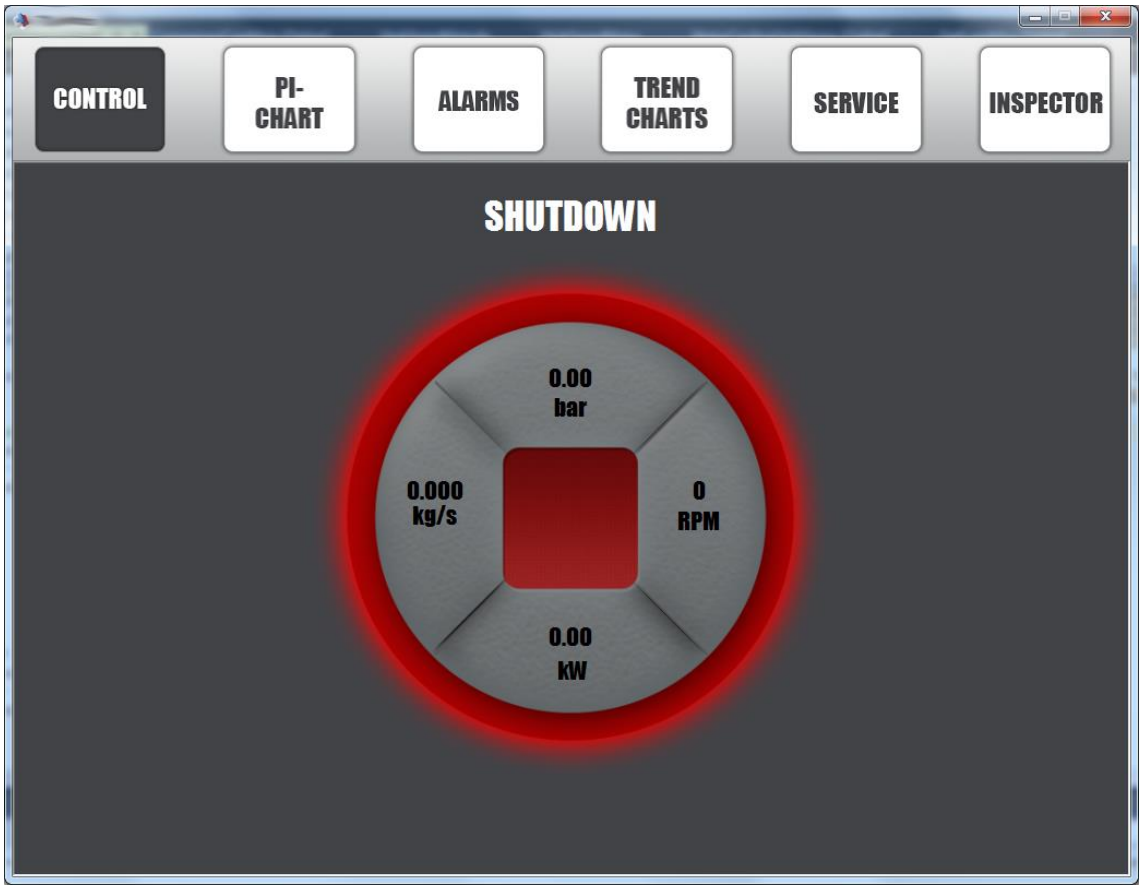
Tietorakenteiden kirjoitus olisi ollut myös mahdollista käyttäen joko WriteAny-metodia tai edellisessä kappaleessa esiteltyä omaa luokkaa. Projektissa ei kuitenkaan ollut tarvetta kirjoittaa suuria datamääriä logiikalle, joten kaikki kirjoitus suoritettiin yksitellen WriteAny-metodilla.

### 6.3 Peruskäyttöliittymä

Käyttöliittymä suunniteltiin alun perin ainoastaan Beckhoffin kosketusnäytölliselle CP2607-teollisuus PC:lle. Koska kosketusnäyttö oli sekä resoluutioltaan että fyysiseltä kooltaan pieni, täytyi painikkeiden olla tarpeeksi suuria sujuvan toiminnan takaamiseksi. Kuvassa 39 on kuvankaappaus peruskäyttöliittymän ohjaussivulta. Kuvassa käyttöliittymä on avattu laboratoriokoneessa, joten se on ikkunatilassa ja sisältää Windowsista tutun yläpalkin. Kosketusnäyttöpaneelissa ohjelma ajettiin koko näytön tilassa, jolloin palkki ei ollut näkyvissä.

Peruskäyttöliittymä koostui kuudesta eri ikkunasta. Pääikkunassa ohjattiin kompressoria, PI-kaavio-sivulla nähtiin prosessin mittauksia, hälytyssivulla lueteltiin ilmoitukset, kuvaajasivulla voitiin seurata mittauksia koordinaatiston avulla, huoltosivulla asetettiin parametreja ja Inspector-sivulla nähtiin laitteen käytettävyystiedot. Huoltosivua ei saa-

nut auki ilman salasanaa, jotta kompressorin parametreja ei pystynyt muuttamaan ilman oikeuksia.




KUVA 39. Peruskäyttöliittymä koostui selkeistä painikkeista kosketusnäyttöä varten

## 6.4 Laboratorion käyttöliittymä

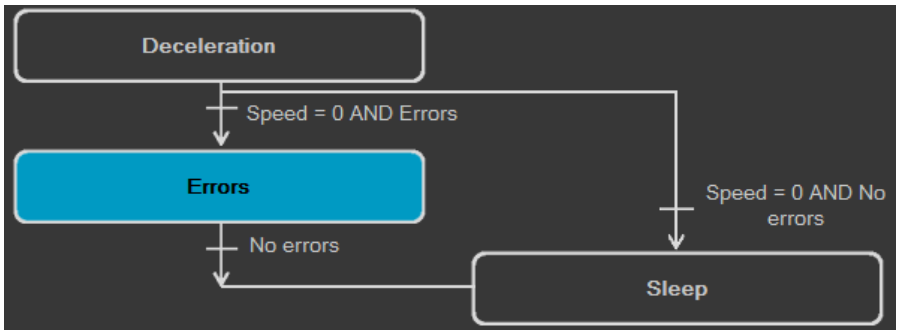
Laboratoriokäyttöliittymä päätettiin rakentaa peruskäyttöliittymän päälle. Koska laboratoriossa operointitietokoneessa oli kolme näyttöä, suunniteltiin käyttöliittymä toimimaan kaikilla kolmella. Koska laboratorion näyttöjen resoluutio oli 1920x1080, oli käyttöliittymällä käytössä paljon enemmän tilaa ja ikkunat suunniteltiin olemaan koko näytöllä. Ainoastaan peruskäyttöliittymä pidettiin pienen kokoisena yhteensopivuuden varmistamiseksi kosketusnäyttöpaneelin ja tavallisen tietokoneen välillä. Laboratorion käyttöliittymään toteutettiin yksi ikkuna järjestelmän mittauksille, yksi järjestelmän

ohjaukselle sekä yksi ikkuna ulkoisille ohjauksilla ja muuttujien ylikirjoittamiselle. Loput toiminnot, kuten hälytyslista, olivat käytössä peruskäyttöliittymän kautta.

Laboratorion käyttöliittymän suunnittelussa tärkeää oli se, että käyttöliittymästä tulisi hyvin dynaaminen. Käyttöliittymässä näkyvien mittausten sekä ohjattavien muuttujien määrän piti päivittyä automaattisesti vastaamaan todellisia, logiikassa olevia kohteita. Tällä haluttiin minimoida käyttöliittymän ohjelmistokehitykseen myöhemmin käytettävä aika ja parantaa laboratoriotyöskentelyä siten, että C#-ohjelmointia osaamattomien käyttäjien tarvitsi muuttaa ainoastaan tiettyjä asioita PLC-koodissa. Näin toimien laboratorion käyttöliittymää ei tarvinnut ollenkaan ohjelmoida mittausten lisääntyessä tai vähentyessä.

### 6.4.1 Laboratorion ohjaus

Laboratoriokäyttöliittymän toisena pääikkunana toimi ohjaussivu. Ohjaussivu sisälsi sekvenssikaavio kompressorin toiminnasta sisältäen käynnistyksen, käynnissäolon sekä pysäytyksen. Sekvenssikaavion jokaisessa siirtymäehdossa luki myös ehto tekstinä, jotta käyttäjä näki minkä takia sekvenssi ei siirtynyt seuraavaan askeleeseen. Aktiivisena ollut sekvenssikaavion lohko korostettiin sinisellä taustavärillä, jolloin käyttäjän oli mahdollista nähdä järjestelmän senhetkinen tila nopealla vilkaisulla. Kuvassa 40 on osa ohjaussivun vapaamuotoista sekvenssikaaviota, joka kertoo järjestelmän odottavan virheiden kuittausta.




KUVA 40. Ohjaussivulla järjestelmän tila esitettiin graafisesti sekvenssityyppisesti

Ohjaussivulta löytyivät painikkeet järjestelmän käynnistämiseksi sekä pysäyttämiseksi. Lisäksi käyttäjän oli mahdollista muuttaa yleisimmin muutettavia parametreja nopeasti. Ikkunan vasemmassa reunassa näytettiin lisäksi palkki, joka sisälsi järjestelmän tilatietoja kuten lähtevän paineen, virtauksen ja moottorin kierrosnopeuden asetusarvoineen. Ikkuna sisälsi myös toiminnot järjestelmän lokikirjoitusohjelman käyttämiseksi käsin sekä lokiin kirjoittamisen senhetkisen tilan.

### 6.4.2 Mittausikkuna

Laboratoriokäyttöliittymän tärkein ja monimutkaisin osa oli mittausikkuna. Mittausikkunan suunnittelu aloitettiin siitä, että ikkunaan tarvitsi saada kaikki järjestelmän mittaukset kerralla näkyviin. Koska mittauksia oli lähes 40 erilaista, täytyi ne saada järkevästi järjesteltyä. Mittausten esittämiseen päädyttiin käyttämään Windows Forms -alustan ListView-komponenttia, joka näyttää annetut tiedot Windowsin tiedostoselaimesta tutussa luettelossa. MeasurePrms-tietorakenteeseen lisättiin mittauksen järjestämistä varten nimi (name), kuvaus (description) sekä ryhmä (group). Näiden avulla mittaukset käyttöliittymässä voitiin jakaa käyttöliittymässä ryhmittäin omiin luetteloihinsa ja jokaisella mittauksella oli selkeät kuvaukset. Kuvassa 41 on laboratoriokäyttöliittymän mittausikkuna, jossa mittaukset ovat järjestelty viiteen eri ryhmään. Kuva on otettu kehitysympäristössä, joten mittaukset eivät ole kytketty antureihin.


KUVA 41. Mittausikkunassa mittaukset jaettiin ryhmittäin järjestettäviin taulukkoihin

Logiikkaan luotiin käyttöliittymässä näytettäviä mittauksia varten oma tietorakenne ST_GuiMeasurements, joka sisälsi mittausten lukumäärän, käyttöliittymälle kerrottavan päivityskäskyn sekä taulukon kaikista mittauksista (kuva 42). Päivityskäsky asetettiin todeksi silloin, kun mittausten lukumäärä muuttui, jotta käyttöliittymä voisi päivittää kaikki mittaukset. Muutoin ainoastaan mittausten arvot päivittyivät jatkuvasti. Mittaukset esitettiin oman tietorakenteen (ST_GuiMeasurement) avulla ja ne sisälsivät skaalatun arvon, mittauksen tilan, nimen, kuvauksen sekä ryhmän. Measure-toimilohkon lopussa mittauksen skaalattu arvo sekä tiedot kopioitiin käyttöliittymän lukemaan tietorakenteeseen. Mittaus voitiin myös piilottaa tarvittaessa käyttöliittymästä.

```
{attribute 'pack_mode' := '1'}
TYPE ST_GuiMeasurements :
STRUCT
    FreeMeasurementIndex    : INT := 1; //Next free index for GuiMeasurement array
    MeasurementTotalCount   : INT := 0; //How many measurements currently
    LastCycleMeasCount      : INT := 0; //How many measurements @last cycle
    ForceIndexUpdate        : BOOL; //Measures are deleted/added. Update all indexes

    GuiShouldUpdate         : BOOL; //GUI should update measurements.

    Measurements            : ARRAY[1..MaxGuiMeasurements] OF ST_GuiMeasurement;
END_STRUCT
END_TYPE

{attribute 'pack_mode' := '1'}
TYPE ST_GuiMeasurement :
STRUCT
    ScaledValue :   REAL;
    MeasState   :   WORD;
    Name        :   STRING(10); //ie. PI-205
    Desc        :   STRING(28); //ie. Inlet pressure
    Group       :   STRING(25); //ie. Motor 1
END_STRUCT
END_TYPE
```

KUVA 42. Mittausten siirtämisessä käyttöliittymään käytetyt tietorakenteet

Käyttöliittymässä mittausten lisäys ikkunaan tapahtui täysin ohjelmallisesti. Mittausikkunassa toimiva ajastin suoritti mittausten päivityksen 300 ms välein. Mittaukset sisältävä tietorakenne luettiin kokonaisena logiikasta, jonka jälkeen päivitysfunktio kävi kaikki mittaukset läpi. Mikäli mittauksia ei oltu vielä haettu tai logiikka komensi täyden

päivityksen, käytiin mittaukset läpi ryhmittäin ja jokaista ryhmää kohti lisättiin oma ListView-komponentti ikkunaan. Tämän jälkeen kaikki logiikalta haetut mittaukset käytiin läpi ja lisättiin kyseisen ryhmän listaan nimen, kuvauksen, tilan ja arvon kanssa. Kun mittaukset oli lisätty, eikä niiden määrä ollut muuttunut, päivitettiin jokaisella päivityssyklillä ainoastaan mittausten tilat ja mittausarvot. Näin ikkunassa ei tapahtunut minkäänlaista ylimääräistä vilkkumista ja mittaukset päivittyivät siististi.

Dynaamista mittausikkunaa tehtäessä huomasi Visual C#:n edun käyttöliittymäsuunnittelussa. Vaikkakin perinteisellä käyttöliittymäohjelmistolla on nopea tehdä yksinkertaisia käyttöliittymiä, pystyy C#:tä käyttämällä tekemään käytännössä mitä tahansa. C#:sta löytyvän LINQ-kirjaston avulla voidaan taulukoihin ja listoihin tehdä tietokantatyyppisiä kyselyjä, joiden avulla datan käsittelyä on helppoa. Mittauksista voitiin hakea kaikki eri ryhmät helposti LINQ-kyselyn avulla kuvan 43 mukaisesti. Kyseinen koodi palauttaa groups-muuttujaan taulukkona kaikki eri ryhmät aakkosjärjestyksessä, mitä käytettiin hyödyksi listoja luodessa.

```
var groups = GetTcGuiMeasurements().Measurements.Where(x => x.Desc != "" || x.Name != "").
                Select(x => x.Group).Distinct().OrderBy(x => x);
```

KUVA 43. LINQ-kirjaston avulla mittausryhmät saatiin haettua kyselymäisesti

Mittausten tilojen ja skaalattujen arvojen päivitys onnistui helposti C#:n foreach-komennolla, joka käy läpi kaikki annetun taulukon tai listan sisällön. Mittaukset päivitettiin käymällä kaikki ikkunaan luodut ListView-listat läpi. Jokaiseen ListView-listan riviin oli tallennettu myös mittauksen juokseva numero Tag-arvoon, jonka perusteella mittaus voitiin päivittää helposti (kuva 44).

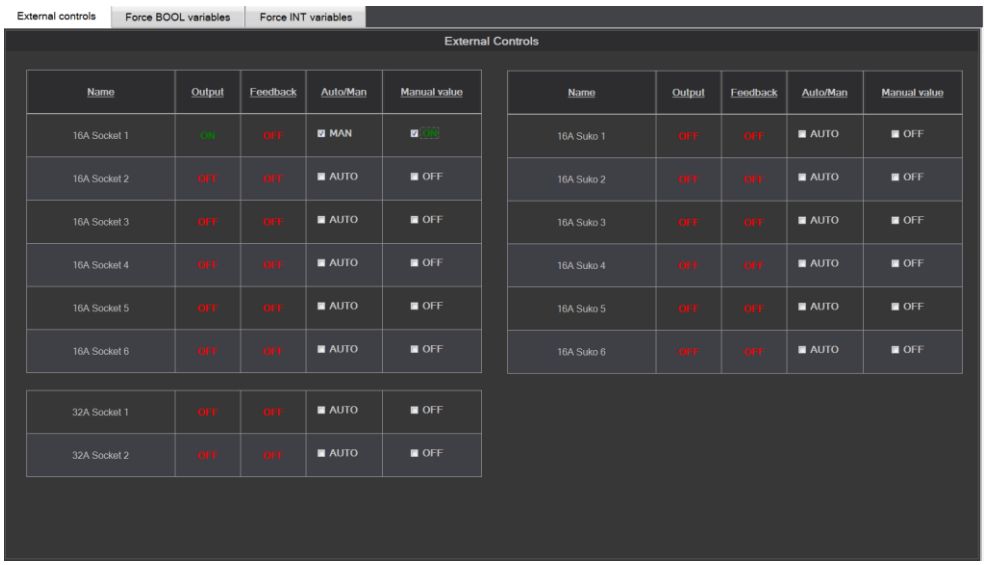
```
TwinCat.GuiMeasurements.GuiMeasurementStruct meas;

//Loop through each dynamically created list
foreach (KeyValuePair<string, doubleBufferedList> list in dictGroupLists)
{
    //Loop through each row in the list
    foreach (ListViewItem i in list.Value.Items)
    {
        //Get data of the measure from this row
        meas = GetTcGuiMeasurements().Measurements[(int)i.Tag];

        //Update scaled value
        i.SubItems[3].Text = meas.ScaledValue.ToString("F2");

        //Update state
        if (meas.MeasState == (ushort)0)
        {
            i.SubItems[2].Text = "";
        }
        else
        {
            i.SubItems[2].Text = meas.MeasState.ToString();
        }
    }
}
```

KUVA 44. Mittausten tilojen ja arvojen päivitys tapahtui foreach-komennolla.

### 6.4.3 Ulkoiset ohjaukset ja muuttujien pakko-ohjaukset

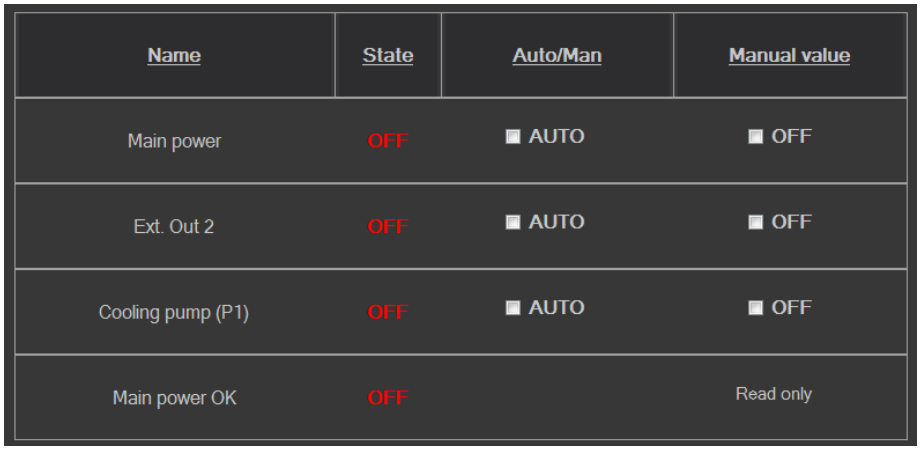
Koska laboratorio-olosuhteissa oli tarve ohjata ulkoisia pistorasioita sekä pakko-ohjata logiikan muuttujia tiettyihin arvoihin, tehtiin ohjauksille käyttöliittymään oma ikkuna. Ikkuna jaettiin kolmeen eri välilehteen .NET-ympäristön TabControl-komponentin avulla: ulkoisiin ohjauksiin, BOOL-muuttujien pakko-ohjauksiin sekä INT-tyyppisten muuttujien pakko-ohjauksiin.

Ulkoiset ohjaukset esitettiin taulukoissa riveittäin siten, että jokaista ohjausta kohti näytettiin sekä lähdön tila että kontaktoreilta saatu kytkintieto. Tilojen näyttämisessä käytettiin punaista ja vihreää fonttia käytön helpottamiseksi. Jokaista ohjausta kohti oli myös valinta, käytettiinkö käyttöliittymästä annettua vai logiikassa linkitettyä tietoa. Jos tila asetettiin käsiohjaukselle, voitiin ohjausta ohjata käyttöliittymästä käsin valintaruudun avulla. Kuvassa 45 on kuvankaappaus ohjaussivulta, jossa ensimmäisen ohjauksen lähtö on asetettu käsin päälle. Paluutiedosta kuitenkin nähdään, ettei kontaktori ole vetänyt eikä ohjaus näin ollen toteudu.




KUVA 45. Ulkoisia ohjauksia voitiin ohjata ja tarkastella käyttöliittymässä

Muuttujien pakko-ohjaukset toteutettiin vastaavalla tavalla. Ero ulkoisiin ohjauksiin oli se, että pakotukset määriteltiin PLC-koodissa dynaamisesti. Käyttöliittymä haki logiikalta tiedot ohjauksista ja näytti ne taulukkomuodossa. Jos logiikan ohjelmakoodiin lisättiin tai poistettiin pakko-ohjauksia, päivittyivät käyttöliittymän taulukot automaattisesti. Pakko-ohjattavia muuttujia voitiin konfiguroida myös vain-luku-tilaan, jolloin pakko-ohjaussivua voitiin käyttää muuttujien arvojen tarkasteluun. Kuvassa 46 on kuvankaappaus pakko-ohjauksia sisältävästä taulukosta. Kuvasta nähdään että päävirtojen (”Main Power OK”) tila oli ainoastaan tarkastelua varten eikä sitä voinut muuttaa, kun taas muita muuttujia voitiin pakko-ohjata käyttöliittymästä.




KUVA 46. Pakko-ohjauksien avulla voitiin käsitellä logiikan muuttujien arvoja

# 7 TIEDONKERUUJÄRJESTELMÄ

Tässä kappaleessa tutustutaan teolliseen internetiin sekä sen käyttökohteisiin. Lisäksi käydään läpi Inspector-tiedonkeruujärjestelmän idea ja toimintaperiaate sekä järjestelmän käyttöönotto kompressoriyksikköön.

## 7.1 Teollinen internet

Teollinen internet on 2010-luvulla yleistynyt termi, jolla tarkoitetaan teollisten laitteiden ja ohjausjärjestelmien kytkeytymistä internetiin. Yhä useammin teollisuuden järjestelmillä on pääsy verkkoon ja niihin on mahdollista päästä käsiksi myös ulkopuolelta. Laitteet lähettävät tietoja internetin kautta palvelimille, joiden avulla tietoa käsitellään ja käytetään hyödyksi. Teollinen internet liittyy myös vahvasti käsitteeseen esineiden internet (IoT, Internet of Things), jolla puolestaan tarkoitetaan erityisesti pienten laitteiden kuten langattomien antureiden kytkeytymistä verkkoon. (Ventä 2016.)

Ensimmäisiä teollisen internetin sovelluksia ovat olleet erilaiset kunnossapito- ja etävalvontasovellukset, jotka valvojat järjestelmän tilaa ja lähettävät tilatietoja sekä anturidataa palvelimelle. Tähän liittyy yleisesti teolliseen internetiin liittyvä termi Big Data, jolla tarkoitetaan järjestelmistä kerättyä suurta tiedon määrää. Yhä useamman anturin ja järjestelmän mittausdata tallennetaan tietokantoihin analysointia varten, jonka johdosta analysoitavan tiedon määrä kasvaa jatkuvasti. (Ventä 2016.)

Työssä ohjelmoitu kompressoriyksikkö liitettiin teolliseen internetiin mittausten etäseurantaa ja käyttöasteen mittaamista varten. Kompressori oli yhteydessä internetiin, jonka kautta logiikassa toimiva Inspector-ohjelmisto lähetti mittausdataa palvelimelle. Teollisen internetin ansiosta kompressoria voitiin valvoa mistä tahansa. Tulevaisuudessa kompressori voisi käyttää teollista internetiä hyödyksi esimerkiksi ilmoittamalla tulevista huolloista ja mahdollisista häiriöistä.

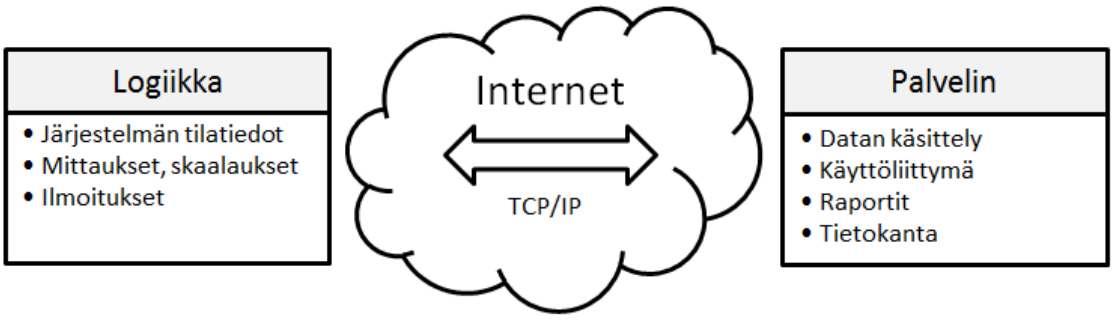
## 7.2 Inspector-tiedonkeruujärjestelmä

Inspector on tamperelaisen InSolution Oy:n kehittämä tuotannonseurantajärjestelmä. Järjestelmä on jaettu useaan osaan, joita ovat automaattinen tiedonkeruu (ADC), kunnonvalvonta (CM) sekä tuotantolaitteen kokonaistehokkuuden (OEE) laskenta. Inspectorin käyttöliittymä on selainpohjainen ja se toimii kaikilla laitteilla, kuten tietokoneilla, puhelimilla ja tableteilla.

Tiedonkeruu kerää järjestelmän tilatietoja, joita ovat käynti-, pysäytys- sekä vikatilat. Tilatietojen avulla voidaan laskea järjestelmän käytettävyys ja käyttöaste, joiden avulla laitteiden hyötysuhdetta voidaan seurata. Kunnonvalvonta seuraa järjestelmän toimintaa esimerkiksi värähtelyantureiden avulla ja mittausdataa voidaan seurata tarkasti käyttöliittymän kautta. OEE-laskennan avulla voidaan seurata järjestelmän todellista tehokkuutta sillä se ottaa huomioon käytettävyyden, tuotantonopeuden sekä laadun.

### 7.2.1 Toimintaperiaate

Inspector perustuu ohjelmoitavaan logiikkaan, joka kerää laitteiden tietoja, suorittaa analogiamittauksia sekä lähettää ja vastaanottaa dataa palvelimen kanssa (kuva 47). Logiikka hakee palvelimelta asetuksensa käynnistyessään sekä palvelimen ilmoittaessa niiden muuttuneen. Asetukset määrittävät logiikan I/O-asetukset, mitattavat signaalit sekä niiden skaalaukset ja suodatukset, käyntitietojen seurantaparametrit sekä erilaisia yhteysasetuksia. Yhteys palvelimen ja logiikan välillä on toteutettu TCP/IP-protokollalla.

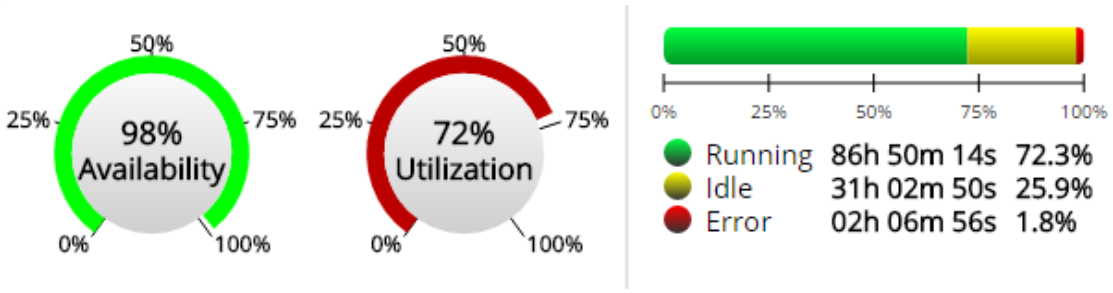



KUVA 47. Inspector toimii internetin välityksellä pilvipalveluna

Palvelimen päässä toimiva ohjelmisto käyttää asiakkaan valinnan mukaan joko MySQL- tai MsSQL-tietokantaa mittausdatan sekä asetusten tallentamiseen ja käsittelyyn. Palvelimen ohjelmisto sekä selainpohjainen käyttöliittymä on toteutettu Microsoft Server -käyttöjärjestelmässä suoritettavalla Asp.Net-kehitysympäristöllä.

### 7.2.2 Kompressoriyksikön seuranta

Inspector-seurantajärjestelmä otettiin käyttöön kompressoriyksikössä, jotta mittauksia sekä käyntitietoja voitiin valvoa ja seurata mistä tahansa kompressorin ollessa asiakkaan tiloissa. Lisäksi Inspector asennettiin laboratorioympäristöön paikallisesti tukemaan laboratoriossa tapahtuvaa kehitystyötä. Loppukäyttäjälle Inspectorista saatavia tärkeitä tietoja olivat kompressorin todellinen käytettävyys sekä käyttöaste, sillä näiden perusteella voidaan havainnoida laitteen käyttöä ja esimerkiksi takaisinmaksuaikaa. Käytettävyys ja käyttöaste esitettiin Inspectorin selainnäkymässä graafisesti päivä-, viikko- tai kuukausitasolla (kuva 48). Käyttöönotto- ja testausvaiheessa tärkeämpi ominaisuus oli kuitenkin kunnonvalvontajärjestelmän mittaukset, joiden avulla kompressorin mitattavia suureita voitiin seurata sekuntitarkkuudella. Mittausten avulla kompressorin toiminta tiedettiin tarkasti toimistolta käsin.

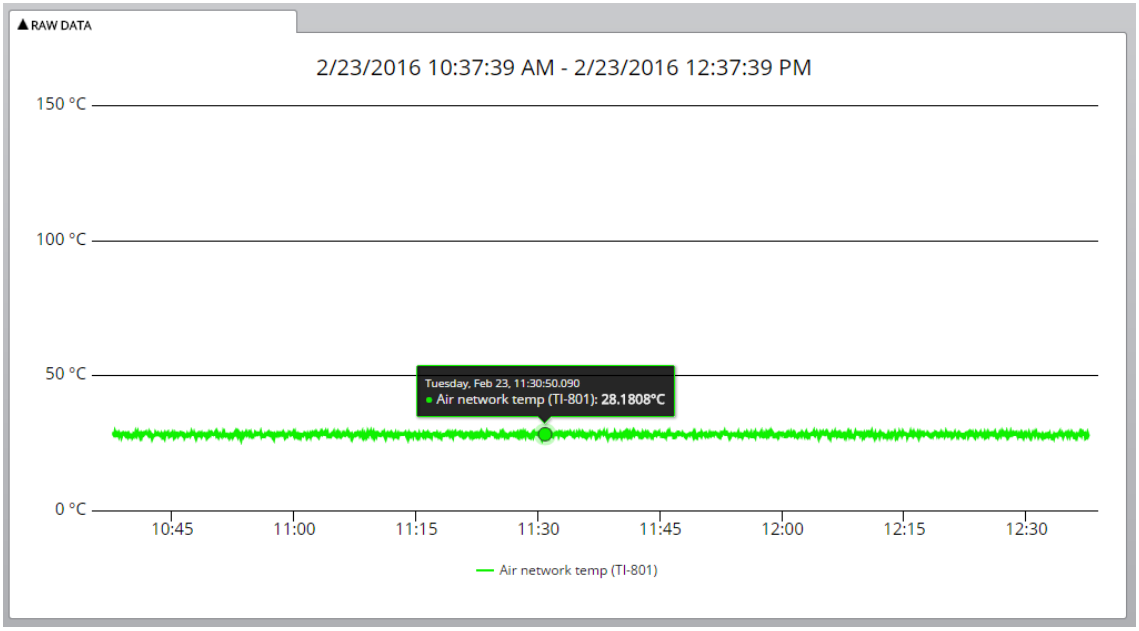



KUVA 48. Käytettävyys ja käyttöaste esitettiin Inspectorissa graafisesti

Inspector otettiin käyttöön tuomalla projektiin järjestelmään liittyvät ohjelmakoodit, tietorakenteet ja muuttujalistaukset TwinCAT 3 -ohjelmiston sisältämällä Import PLCopenXML -toiminnolla (kappale 2.3). Tämän jälkeen logiikkaohjelmaan lisättiin oma taski suorittamaan tiedonkeruun ohjelmakoodia. Inspectorin käyttämät asetukset

olivat järjestetty omaan muuttujalistaan, josta asetettiin yhteyden muodostamiseen tarvittavat kommunikointiasetukset. Koska logiikkaohjelmoinnissa taulukkojen koot eivät voineet olla dynaamisia, täytyi lisäksi asettaa mittausten enimmäislukumäärät taulukkojen alustamiseksi. Muita asetuksia ja määrityksiä ohjelmakoodiin ei tarvinnut tehdä, vaan järjestelmä sai kaikki asetuksensa palvelimelta logiikkaohjelman käynnistyttyä.

Kun logiikassa ajettava Inspector-ohjelma käynnistyy, hakee se palvelimelta kunnonvalvontamittausten parametrit. Parametreja ovat muun muassa mittausresoluutio, skaalaus sekä offset, erilaiset suodatukset ja aseteltavat ylä- ja alarajat. Mittavat signaalit skaalataan parametrien mukaisesti, puskuroidaan välimuistiin ja lähetetään palvelimelle tietyn ajan välein. Palvelin tallentaa vastaanotetut tiedot ja ne esitetään käyttäjälle kuvaajina valitulta aikaväliltä (kuva 49). Inspector mahdollistaa myös ilmoitusten lähettämisen palvelimelle. Ilmoitus voi olla esimerkiksi tieto järjestelmän käynnistymisestä, siinä sattuneesta virheestä tai tilan muutoksesta. Ilmoitus voidaan lähettää myös mitattavan signaalin ylittäessä asetetun ylä- tai alarajan, jolloin muuttuneeseen signaaliin voidaan reagoida.




KUVA 49. Kompressorin mittaama paineilmaverkon lämpötila Inspectorin selainnäkymässä aikatasossa

# 8 POHDINTA

Työn tavoite oli ohjelmoida kompressoriyksikköä ohjaava automaatio-ohjelmisto, toteuttaa laboratorioympäristöön oma versio ohjelmistosta sekä suunnitella ja ohjelmoida käyttöliittymä järjestelmän käyttöä varten. Työn tavoitteet saavutettiin ja ne onnistuivat hyvin.

Tutkimus- ja kehityslaboratorioon kehitetty lisäominaisuuksia sisältänyt automaatio-ohjelmisto on ollut aktiivisessa käytössä asiakkaan kehitystyössä. Erityisesti monipuolinen dynaamisesti toimiva laboratoriokäyttöliittymä on havaittu hyödylliseksi työskentelyn helpottumisen ansiosta. Laboratorion kehitystyö jatkuu tulevaisuudessa erityisesti erilaisilla väylärajapintojen kehittämisellä asiakkaiden liityntöjä varten.

Työssä ohjelmoitu kompressoriyksikkö on ollut raportin kirjoitushetkellä kaksi kuukautta testauksessa asiakkaalla teollisuusympäristössä. Järjestelmä on toiminut koko testausajan hyvin ilman mainittavia ongelmia ja testauksen aikana järjestelmään on myös suunniteltu ja kehitetty uusia ominaisuuksia. Järjestelmän mittauksia on myös seurattu etänä tiedonkeruujärjestelmän ansiosta kompressorin testauksen aikana.

Työssä tutustuttiin IEC 61131-3 -standardiin sekä sen uusiin oliopohjaisiin ominaisuuksiin siten, että työstä olisi hyötyä myös asiakasyritykselle. Työn aikana muodostunut tieto sekä kirjoitettu raportti tukevat asiakkaan dokumentointia kompressoriyksiköstä sekä lisäävät tietoa uusien ominaisuuksien mahdollisuuksista kehitystyössä. Työn aikana esille tulleet ohjelmointiympäristön ominaisuudet ja mahdollisuudet pyritään käyttämään hyödyksi myös muissa tulevissa asiakasprojekteissa.

# 9 JATKOKEHITYSEHDOTUKSET

Tässä kappaleessa käydään läpi työn aikana ilmi tulleita asioita, joita olisi voinut tehdä toisin esimerkiksi uusia ominaisuuksia hyödyntäen. Työtä tehdessä havaittiin myös muita ohjelmointia koskevia käytännön asioita, joita muuttamalla ohjelmakoodista olisi mahdollista tehdä selkeämpää.

## 9.1 Standardin uusien ominaisuuksien käyttö

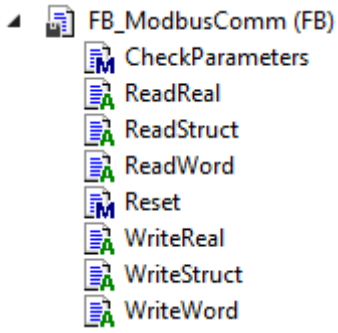
Projekti aloitettiin alun perin TwinCAT 2 -ympäristössä, joka ei mahdollistanut standardin viimeisimmän version uusien ominaisuuksien, kuten periyttämisen ja metodien, käyttöä. Myöhemmin työtä tehdessä TwinCAT 3 -ympäristössä tuli vastaan tilanteita, joissa huomasi uusien ominaisuuksien hyödyn.

Projektia voisi kehittää eteenpäin siten, että ohjelman eri osista saataisiin enemmän oliopohjaisia rakenteita, jotka sisältäisivät kaikki kyseisen ohjelman osaan liittyvät toiminnot ja tiedot. Tällä hetkellä ohjaustoimilohkon tietorakenteet, kuten parametrit, sisään- ja ulostulot sekä dynaaminen data, ovat kaikki julkisissa muuttujalistoissa näkyvyyden varmistamiseksi. Käyttämällä TwinCAT 3 -järjestelmästä löytyvää toimilohkoihin yhdistettävää ominaisuutta (property) olisi mahdollista siirtää tietorakenteet lohkon sisälle (kappale 3.3.2). Ominaisuuksien get-metodit palauttaisivat viittauksen kyseiseen tietorakenteeseen, jolloin niihin pääsisi käsiksi ulkopuolelta. Näin kaikki järjestelmän osaan liittyvät muuttujat ja tiedot olisivat olion sisällä. Jotta tämä toimisi, täytyisi kuitenkin toimilohko esitellä julkisessa muuttujalistassa näkyvyyden varmistamiseksi.

Standardin esittelemä rajapintaluokkia (interface) voisi hyödyntää projektissa oliopohjaisten järjestelmän osien käsittelyssä. Projektiin voisi esimerkiksi luoda rajapintaluokan MachinePart, joka sisältäisi jokaiselle järjestelmän osalle yhteiset metodit. Tällaisia metodeja voisivat olla pysäytys, käynnistys ja tilan asetus. Kun kaikki järjestelmän ohjaustoimilohkot toteuttaisivat kyseisen rajapinnan, voisi järjestelmään ohjelmoida funktioita, jotka ottaisivat vastaan rajapintaluokan toteuttavan lohkon ja käsittelisi sitä huo-

limatta sen todellisesta tyypistä. Rajapintaluokista voisi myös tehdä taulukon, jolloin kaikki järjestelmän osat voitaisiin esimerkiksi pysäyttää tai käynnistää käymällä taulukko läpi silmukan avulla.

Projektia kehittäessä TwinCAT 3 -ympäristössä otettiin oliopohjaisia ominaisuuksia käyttöön erityisesti toimilohkojen ja ohjelmien sisältämien metodien osalta. Aikaisemmin tiettyyn ohjelman osaan liittyvät funktiot olivat erillisiä, mutta kehityksen edetessä uusien lohkojen funktiot sisällytettiin kyseisen lohkon alle. Lohkojen sisältämää koodia selvennettiin jakamalla sitä osiin ja tekemällä osista omia toimintoja (action) tai metodeja lohkon alle. Näin ohjelmakoodista saatiin selkeämpää ja oliomaisempaa. Kuvassa 50 on esitetty Modbus-väylässä tapahtuvaan kommunikointiin ohjelmoitu toimilohkon rakenne. Toimilohko ei itsessään sisällä yhtään ohjelmakoodia, vaan kaikki sen toiminnot ovat eritelty metodeihin sekä toimintoihin. Näin yhden toimilohkon avulla saatiin luotua kommunikointiin käytettävä luokka, jonka eri toimintoja voitiin kutsua ulkopuolelta tarvittaessa.




KUVA 50. Standardin uusien ominaisuuksien käyttö toimilohkon yhteydessä

## 9.2 Luetellut tietotyypit

Järjestelmän perustuessa järjestelmän eri osia ohjaaviin toimilohkoihin ja niille annettaviin komentoihin oli ohjelmoijalla paljon muistettavaa. Jokainen komento annettiin numerona, joita yleisimmin olivat pysähdystilaa vastaava 0 ja käyntitilaa vastaava 100. Ohjelmakoodia voisi selventää käyttämällä hyödyksi lueteltuja tietototyyppejä eli enumeraatioita. Luomalla jokaiselle koneelle oma enumeraatio eri vaiheesta, ohjelmakoodista tulisi selkeämpää. Tällöin eri tilat esitettäisiin kuvaavina merkkijonoina numeroiden sijasta eli esimerkiksi moottorin pyyntö voisi olla kokonaisluvun 100 sijaan enumeraatio ”RunAutoMode”.

## 9.3 Käyttöliittymä

Käyttöliittymä kommunikoi ADS-tekniikan avulla siten, että jokainen muuttuja luettiin tietyn ajan välein. Beckhoffin ADS-tekniikka tukee myös ilmoituksia (notification), joita käyttämällä olisi mahdollista vähentää tiedonsiirtoa ja yksinkertaistaa ohjelmakoodia. Ilmoitukset toimivat siten, että ohjelman alussa asetetaan haluttuja logiikkaohjelman muuttujia valvovat ilmoitukset. Ilmoitus annetaan TcAdsClient-luokan metodilla AddDeviceNotification, joka ottaa vastaan muun muassa muuttujan nimen sekä sykliajan, jonka välein logiikka tarkastaa onko muuttujan arvo muuttunut. Jos muuttujan arvo muuttuu, lähettää logiikka ilmoituksen, joka voidaan käsitellä käyttöliittymässä halutulla tavalla. Ilmoituksia voisi käyttää harvoin muuttuvien ja ei-kriittisten arvojen lukemisessa logiikalta. Näin kaikkia tietoja ei luettaisi jatkuvasti vaan ainoastaan tarpeen vaatiessa.

**LÄHTEET**

3S-Smart Software Solutions GmbH. 2010. User Manual for PLC Programming with CoDeSys 2.3. Käyttöohje. Tulostettu 28.02.2016. http://www.wago.com/wagoweb/documentation/759/eng_manu/333/m07590333_00000000_1en.pdf.

3S-Smart Software Solutions GmbH. 2014. CODESYS Engineering. Professional Engineering of IEC 61131-3 Automation Projects. Esite. Tulostettu 27.09.2016. http://www.sks.fi/www/images/CODESYS_Engineering_online_en_1114.pdf/$FILE/CODESYS_Engineering_online_en_1114.pdf.

3S-Smart Software Solutions GmbH. 2015. Inspiring Automation Solutions. Esite. Tulostettu 27.09.2016. http://www.sks.fi/www/images/CODESYS_InspiringAutomationSolutions_en_0115.pdf/$FILE/CODESYS_InspiringAutomationSolutions_en_0115.pdf.

3S-Smart Software Solutions GmbH. 2016. CODESYS C-Integration. Www-sivu. Tulostettu 28.02.2016. https://www.codesys.com/products/codesys-engineering/c-integration.html.

Beckhoff. 2011. TwinCAT 3 – XA Language Support: C/C++. Www-sivu. Tulostettu 28.02.2016. https://www.beckhoff.com/english.asp?twincat/twincat-3-xa-language-support-c.htm?id=1893323818933256.

Beckhoff. 2012. TwinCAT 3 | eXtended Automation (XA). Esite. Tulostettu 28.02.2016. http://download.beckhoff.com/download/Document/catalog/Beckhoff_TwinCAT3_042012_e.pdf.

Beckhoff. 2016. CX51x0 Hardware Manual. Käyttöohje. Tulostettu 12.03.2016. https://download.beckhoff.com/download/document/ipc/embedded-pc/embedded-pc-cx/cx5100_hwen.pdf.

Beckhoff. 2016. Tietoa yrityksestä. Www-sivu. Päivitetty 04.01.2016. Tulostettu 27.09.2016. http://beckhoff.fi/fi/default.htm?beckhoff/default.htm.

Beckhoff. N.d. ADS Introduction. Www-sivu. Tulostettu 18.03.2016. https://infosys.beckhoff.com/english.php?content=../content/1033/tcadscommon/html/tcadscommon_intro.htm.

Beckhoff. N.d. Programming conventions for creating the IEC61131-3. Www-sivu. Tulostettu 28.3.2016. https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclibsoverview/html/tcplclibsprogrammingconventions.htm.

Beckhoff. N.d. Reading and writing of PLC variables of any type. Www-sivu. Tulostettu 24.3.2016. https://infosys.beckhoff.com/english.php?content=../content/1033/tcsample_net/html/twincat.ads.sample07.htm&id=23803.

Diater H. 2014. 3S-Smart Software Solutions GmbH and CODESYS. CODESYS Users' Conference 2014. Esitelmä. Tulostettu 28.02.2016. http://prolog-plc.ru/docs/conf14/UC_2014_CODESYS_Bussiness_en.pdf.

EtherCAT Technology Group. 2012. EtherCAT – the Ethernet Fieldbus. Esite. Tulostettu 12.03.2016. www.ethercat.org/pdf/english/ETG_Brochure_EN.pdf.

Hanssen, D. 2015. Programmable Logic Controllers. A Practical Approach To IEC 61131-3 Using CODESYS. 1. painos. Englanti: John Wiley & Sons, Ltd.

Hietanen, P. 1999. C++ ja olio-ohjelmointi. 4. painos. Porvoo: Teknolit

IEC 61131-3. 2013. Programmable controllers – Part 3: Programming languages. Sveitsi: International Electrotechnical Commission (IEC).

John, K.-H. Tiegelkamp, M. 2010. IEC 61131-3: Programming Industrial Automation Systems. 2. painos. Saksa: Sprigner.

Peltomäki, J. Malmirae, P. 199. Java-ohjelmoinnin perukirja. 1. painos. Jyväskylä: Teknolit.

PLCopen. 2011. The 3rd Edition of IEC 61131-3. Esitelmä. Julkaistu 27.09.2011. Tulostettu 25.02.2016.
http://www.iestcfa.org/events/id_plcopen_2011/IEC_3rd_Ed.pdf.

PLCopen. 2015. Motion Control - An Introduction. Esitelmä. Tulostettu 27.02.2016.
http://www.plcopen.org/pages/tc2_motion_control/downloads/plcopen_motioncontrol_feb2015.pptx.

Sundquist, M. (toim.) 2008. Teollisuusautomaation tiedonsiirtoliikenne - Turvaväylät. 1.painos. Espoo: Inspecta Koulutus Oy.

Ventä, O. 2016. Teollisen internetin ja digitalisaation nousu. Automaatioväylä 01/2016, 7.