Johanna Ratavaara

IEC 61499 -standardi ja sen käyttösovellukset

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

6.5.2024

Tiivistelmä

Tekijä:	Johanna Ratavaara
Otsikko:	IEC 61499 -standardi ja sen käyttösovellukset
Sivumäärä:	37 sivua + 1 liite
Aika:	22.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine:	Automaatiotekniikka
Ohjaajat:	Lehtori Reijo Leinonen

Tässä opinnäytetyössä tutustuttiin IEC 61499 -ohjelmointistandardiin. Kyseinen ohjelmointistandardi on julkaistu vuonna 2005 ja se on rakennettu vuoden 1992 vanhemman standardin IEC 61131-3 pohjalle. Vanhemmasta standardissa oli tiettyjä puutteita, joita on pyritty korjaamaan uudistetussa standardissa. Työssä pohdittiin erilaisten ohjelmointiympäristöjen yhteensopivuusongelmia ja sitä, riittääkö IEC 61499 ratkaisemaan näitä haasteita.

Opinnäytetyössä käytiin läpi pääpiirteittäin toimilohko-ohjelmoinnin perusperiaatteita. Työssä esiteltiin kolmen eri valmistajan kaupalliset ratkaisut ja tutustuttiin niihin liittyviin markkinointimateriaaleihin. Myös kaksi avoimen lähdekoodin ohjelmointiympäristöä esiteltiin yleisellä tasolla.

Lopuksi testattiin avoimen lähdekoodin ohjelmointialustaa Eclipse 4diacia. Sillä toteutettiin yksinkertainen ohjelma, joka vaihtoi toimilohkon ulostulon tilaa sekunnin intervallissa. Ohjelman toimivuus todettiin Eclipsen FORTE-virtuaalikoneessa ohjelmointiympäristön sisäänrakennetun monitorointiominaisuuden kanssa.

Yhteenvedossa todettiin, että vaikka IEC 61499 -standardi on merkittävä parannus käyttäjien valinnanvapauteen eri alustojen välillä, tiettyjä haasteita on edelleen olemassa.

Avainsanat: IEC 61499, IEC 61331-3, avoin lähdekoodi, 4diac

Abstract

Author:	Johanna Ratavaara
Title:	IEC 61499-Standard and Applications
Number of Pages:	37 pages + 1 appendix
Date:	22 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Electrical and Automation Engineering
Professional Major:	Automation Engineering
Supervisors:	Reijo Leinonen, Senior Lecturer

This bachelor's thesis investigates IEC 61499-standard on general level. Standard in focus was published in 2005 and it is based on older standard IEC 61131-3, published in 1992. Older standards have had some deficits that have been improved in updated standard. Thesis screens certain programming environment compatibility issues, and studies if new standard IEC 61499 is comprehensive enough to cover these challenges.

Thesis goes through the basics of function-block programming. Three different IEC 61499 compatible commercial solutions for programming environments are presented within the documentation. Two of available open-source environments get an overview display.

The final step of this thesis was testing IEC 61499 compatible open-source platform Eclipse 4diac on basic software. This software produces on/off outfeed for a blinking light, changing in 1 second intervals. The functionality of this program was tested within Eclipse FORTE-virtual machine in tandem with built-in monitoring functionality of 4diac.

The result of this thesis is that while IEC 61499 is a considerable improvement for users' free migration between platforms and manufacturers, there still are some considerable hurdles to overcome.

Keywords: IEC 61499, IEC 61331-3, Opensource, 4diac

Sisällys

Lyhenteet

IDE: *Integrated development environment.* Ohjelmistokokonaisuus, jossa koodia voidaan luoda, editoida ja testata ilman fyysistä laitteistoa.

IEC: *The International Electrotechnical Commission.* Kansainvälinen standardoimisjärjestö, joka valmistelee ja julkaisee kansainvälisiä standardeja koskien elektroniikka- ja sähkötekniikkaa.

IIoT: *Industrial Internet of Things.* Tällä viitataan erilaisiin kokonaisuuksiin sensoreita, toimilaitteita ja automatisoituja laitteistoja, jotka on kytketty toisiinsa ja teollisiin sovelluksiin internetin välityksellä.

OEM: *Original Equipment Manufacturer.* Alkuperäisvalmistaja, viittaa laitteen, laitteiston tai ohjelmiston alkuperäiseen valmistajaan.

PLC: *Programmable Logic Controller.* Ohjelmoitava logiikka. Pieni tietokone, jolla ohjataan automaatiolaitteita.

1 Johdanto

Tämän insinöörityön tarkoituksena on tutustua vuonna 2005 julkaistuun IEC 61499 -ohjelmointistandardiin. Standardi pohjautuu vanhempaan standardiin IEC 61131-3. Molemmat standardit määrittelevät PLC-logiikoiden ja niihin yhdistettävien laitteiden sekä laitteistojen ohjelmointiin käytettäviä kieliä sekä termistöä. Vanhemman, vuoden 1992, standardin tarkoitus oli määritellä ja tunnistaa PLC-laitteistoihin liittyvät perustoiminnot sekä kehittää yhtenevä termistö. Kun termit ovat yhteneviä, kommunikaatio käyttäjien ja kehittäjien välillä helpottuu merkittävästi. Standardissa määriteltiin vähimmäisvaatimukset laitteistojen testaamiseksi.

IEC 61131-3 määritteli myös hyväksytyt ohjelmointikielet, joita PLC-laitteiston tuli tukea, jotta laitetta voitaisiin markkinoida standardin mukaisena. Tosin standardissa on myös kirjoitettu seuraavaa:

> – – defines, for each of the most commonly used programming languages, major fields of application, syntactic and semantic rules, simple but complete basic sets of programming elements, applicable test and means by which manufacturers **may expand or adapt those basic sets to their own programmable controller implementations** [1, s. 5].

Käytännössä standardi määritteli siis vain minimivaatimukset, joita valmistajat saivat laajentaa tai soveltaa haluamallaan tavalla. Tämä johti kohtuullisiin yhteensopivuusongelmiin eri valmistajien laitteistojen kanssa. Osa näistä haasteista oli tahattomia, mutta todennäköisesti kaupallisten yritysten omissa intresseissä ei välttämättä ollut helpottaa loppukäyttäjien migraatiota toisille alustoille.

Mikäli vain joku tietty osuus laitekokonaisuudesta vikaantui, käyttäjän täytyi saada uusi identtinen komponentti korvaamaan sitä, koska loppulaitteisto ei todennäköisesti olisi yhteensopiva eri valmistajan uuden komponentin kanssa. Vaikka komponentin fyysiset ominaisuudet olisivat identtiset, ei se välttämättä ymmärtäisi sensoreiden tuomaa dataa tai käytössä olevaa ohjelmaa.

Ennen kuin ohjelmointikieliä oli standardoitu, jokaisella laitevalmistajalla oli oma näkemyksensä siitä, miten ohjelmointi tulee tehdä. Koska erilaisia laitevalmistajia on markkinoilla kymmeniä, ellei satoja, oli käyttäjien ja laitekehittäjien osaamisen ylläpitäminen hyvin kallista sekä ajallisesti että rahallisesti. [2; 3, s. 12.]

Uudistetun standardin, IEC 61499, tavoitteena on ollut alusta lähtien vapauttaa automaatio-ohjelmointi valmistajakohtaisista rajoitteista ja tähdätä *Open Source* -henkisiin avoimen lähdekoodin ratkaisuihin, joilla pystyttäisiin jatkossa yksinkertaistamaan ja standardoimaan eri laitteiden välistä kommunikaatiota.

Vaikka avoimen lähdekoodin ohjelmointiympäristöt ovat kohtuullisen helppokäyttöisiä ja ladattavissa verkkosivuilta ilmaiseksi, eivät laitteistovalmistajien omat ohjelmointiympäristöt ole jäämässä historiaan vielä vuosikymmeniin. Erityisesti turvallisuuskriittisissä sovelluksissa, tai niihin komponentteja valmistavissa prosesseissa, painotetaan raskaasti niin kutsuttua *Tried and proven* -menetelmää. Se tarkoittaa, että ennen kuin laite tai laitteiston osa hyväksytään käyttöön turvallisuuskriittisessä ratkaisussa, vastaavien kokonaisuuksien on täytynyt olla käytössä jossain muualla jo pidempään. Tällä pyritään varmistamaan, ettei mikään laitteiston osa toimi yllättävällä tavalla, vaan kaikki mahdolliset piilevät viat ja ominaisuudet on jo löydetty.

Tällä hetkellä markkinoilla on useita erilaisia toimilohkokirjastoja, joista käyttäjä voi käydä hakemassa omaan sovellukseensa tarpeelliset toimilohkot, *Function-block,* ja muodostaa niistä toimivan kokonaisuuden yksinkertaisesti raahaamalla toimilohkoja ohjelmointialustalla. [4.]

2 Toimilohko-ohjelmointi yleisellä tasolla

IEC 61499 -standardi julkaistiin ensimmäistä kertaa jo vuonna 2005, jolloin sitä pidettiin seuraavan sukupolven parannuksena PLC-ohjelmoinnin saralla. Viimeisin päivitys on tehty vuonna 2012. Perusajatuksena oli tuoda markkinoille tapahtumiin perustuvat toimilohkot, jotka aktivoituvat vain silloin, kun yksi tai

useampi toimilohkon tapahtumadiskreeteistä sisääntuloista aktivoituu. Tällä säästetään merkittävästi energiaa, kommunikaatiokaistaa ja laskutehoa verrattuna aikaisempiin vanhemman standardin mukaisiin aikadiskreetteihin toimilohkoihin. Tapahtumariippuvainen toimilohko ei jatkuvasti kysele uutta tietoa edeltäviltä sensoreilta tai toimilohkoilta eikä myöskään lähetä tyhjää signaalia eteenpäin tukkimaan kommunikaatioväyliä. [2.]

IEC 61499 -standardissa järjestelmät mallinnetaan kolmessa eri kerroksessa, sovellukset *Application layer,* resurssit *Resource layer* ja laitteet *Devices*. Sovellukset linkitetään resursseihin, jotka toimivat laitteilla. Lopputuloksena saadaan kokonaisuus, joka pitää sisällään sekä laitteiston toimintalogiikan että topologian. [5, s. 11.] Näiden välisiä suhteita on selitetty luvussa 3.2.

Sovellustasolla viitataan systeemin loogisiin toimintoihin. Tällä voidaan tarkoittaa hyvin erilaisia asioita suorittaviin ohjelmakokonaisuuksiin, aina vilkkuvasta valosta (esimerkki luvussa 6) monimutkaisiin kokonaisuuksiin, kuten hissin toimintaan ja sen turvallisuusominaisuuksiin. Sovellus voi toimia joko yhdellä laitteella tai se voi olla hajautettu useille laitteille erilaisten verkkoyhteyksien kautta. Sovellus voidaan luoda valmiiksi ilman varmaa tietoa siitä, miten käytännön toteutus hajautetaan resursseille käyttökohteessa. Se on siis vain kokoelma toimilohkoja, jotka suorittavat halutun toimintokokonaisuuden. Näiden toimilohkojen kapseloitujen funktioiden ohjelmointiin voidaan käyttää mitä tahansa IEC 61131-3 -standardissa määritellyistä kielistä, kunhan toimilohkojen sisään- ja ulosliikenne on yhteensopiva IEC 61499 -standardin kanssa. [5, s. 11–12.]

Resurssitasolla viitataan laitteisiin, joilla sovellustason ohjelmat toimivat. Resurssien tarkoitus on hallita ja suorittaa toimilohkojen funktioita sekä siirtää tietoa eri toimilohkojen välillä. Sovellukset voidaan linkittää eri resurssien välillä aliverkkojen *Subnet* välityksellä. [5, s. 17.]

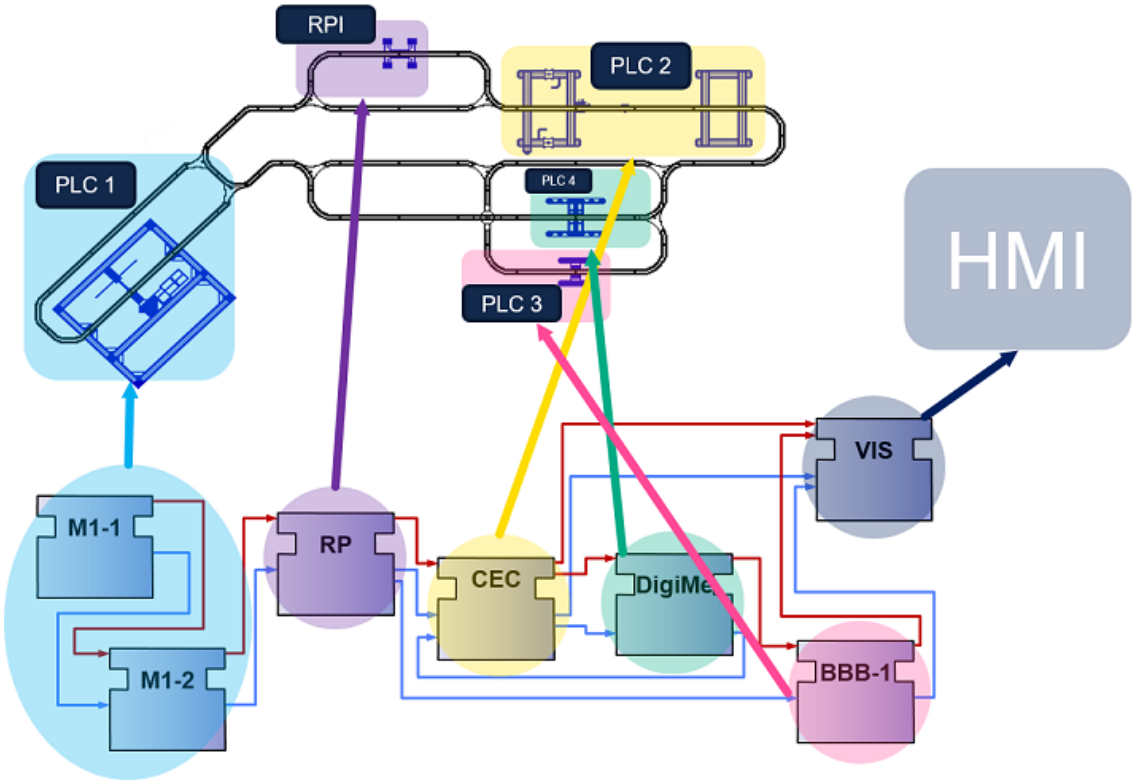
Merkittävä ero IEC 61499- ja IEC 61131-3 -standardien mukaisten resurssimallien välillä on, että vaikka eri resurssit ovat toisistaan riippumattomia, ne eivät välttämättä edusta yhtä fyysistä laitetta topologisesti. Eli IEC 61499 -standardin

mukaisesti yhdellä laitteella voi olla useita samanaikaisia sovelluksia käynnissä, kunhan ne eivät ole ristiriidassa keskenään. [5, s. 17.]

Laitetasolla, josta joskus käytetään myös termiä ylätasonäkymä, tarkoitetaan koko järjestelmän laitteita. Laitteet voivat olla joko virtuaalilaitteita tai fyysisiä laitteita, kuten esimerkiksi PC tai PLC. Laitetaso huolehtii kommunikaatiosta sovelluksien ja fyysisen maailman välillä, esimerkiksi I/O-kytkennöin. IEC 61499 -standardi ei ota erityisen tarkkaa kantaa laitetason ratkaisuihin, vaan standardi asettaa vaatimuksia lähinnä *runtime*-ympäristön kommunikaatioon. Sovellukset ladataan laitteille IDE:n välityksellä. Jos laitteella on käynnissä IEC 61499 -yhteensopiva *runtime*, sovellus lähetetään suoraan laitteelle ja sovellus käynnistyy. [5, s. 18.]

Toimilohkokaavio-ohjelmointi on visuaalinen ohjelmointikieli, jossa erilaiset prosessit on eroteltu omiksi lohkoikseen. Eri lohkot voivat suorittaa omia tehtäviään joko itsenäisesti tai riippuen muista ympäröivistä toiminnoista tarpeen mukaan. Vanhemman standardin mukaisissa kokonaisuuksissa lohkot yleensä suorittivat omaa ohjelmaansa jatkuvasti, riippumatta siitä oliko sille juuri sillä hetkellä tarvetta. IEC 61499:ssä on pyritty tehostamaan tätä prosessia ja toimilohkot aktivoituvat vasta, kun niiden tapahtumasisääntuloon tulee aktivoiva signaali. [6.]

Kun ohjelmoinnissa käytetään toimilohkokaaviota, ylätasonäkymässä on helposti nähtävissä koko järjestelmä yhdellä silmäyksellä – kaikki laitteet, kommunikaatioyhteydet ja aktiivisena olevat ohjelmat. Kuvassa 1 on esitettynä kuvitteellisen järjestelmän ylätason *Device layer* -näkymä. Jokainen eri värillä korostettu laatikko sisältää erilaisia ohjelmia tai laitteita, joiden yksityiskohtaista toimintaa ei ole järkevää esittää suurissa kokonaisuuksissa koko aikaa. Kokonaisuuksien luettavuus kärsii, mikäli käyttäjälle annetaan liikaa informaatiota liian pienessä tilassa. [2.]



Kuva 1. Esimerkki toimilohkokaavio ylätason näkymästä [6].

Kuvan 1 kuvitteellisessa järjestelmässä on nähtävissä ainakin neljä erillistä PLC-laitetta ja RPI, jolla yleensä viitataan Raspberry Pi -minitietokoneeseen, sekä HMI *Human-Machine-Interface,* eli käyttöliittymä. Käyttöliittymällä tarkoitetaan kokoelmaa painikkeita tai kosketusnäyttöä, jolla laitekokonaisuutta ohjataan käyttäjän toimesta. Näiden lisäksi kuvassa on kuusi erilaista prosessia, joista yhdessä, sinisellä korostetussa ellipsissä, on nähtävissä kaksi erillistä yhdessä toimivaa laitetta. Eriväriset nuolet kuvaavat erilaisia tietoliikenneyhteyksiä.

Kuvassa 2 on esitetty esimerkki kuljettimen toimilohko-ohjelmasta ja siitä, mitä näiden lohkojen alla tapahtuu, sekä niiden välisistä kommunikaatioista. Kyseinen kokonaisuus voisi toimia kuvan 1 tapauksessa esimerkiksi lohkon RP sisällä. Kuva 2 ei välttämättä ole kovin selkeä yhdellä silmäyksellä, joten kuvan

sisältöä on pyritty selittämään auki seuraavien kappaleiden aikana. Kuva 2 on nähtävissä liitteessä 1 isompana.

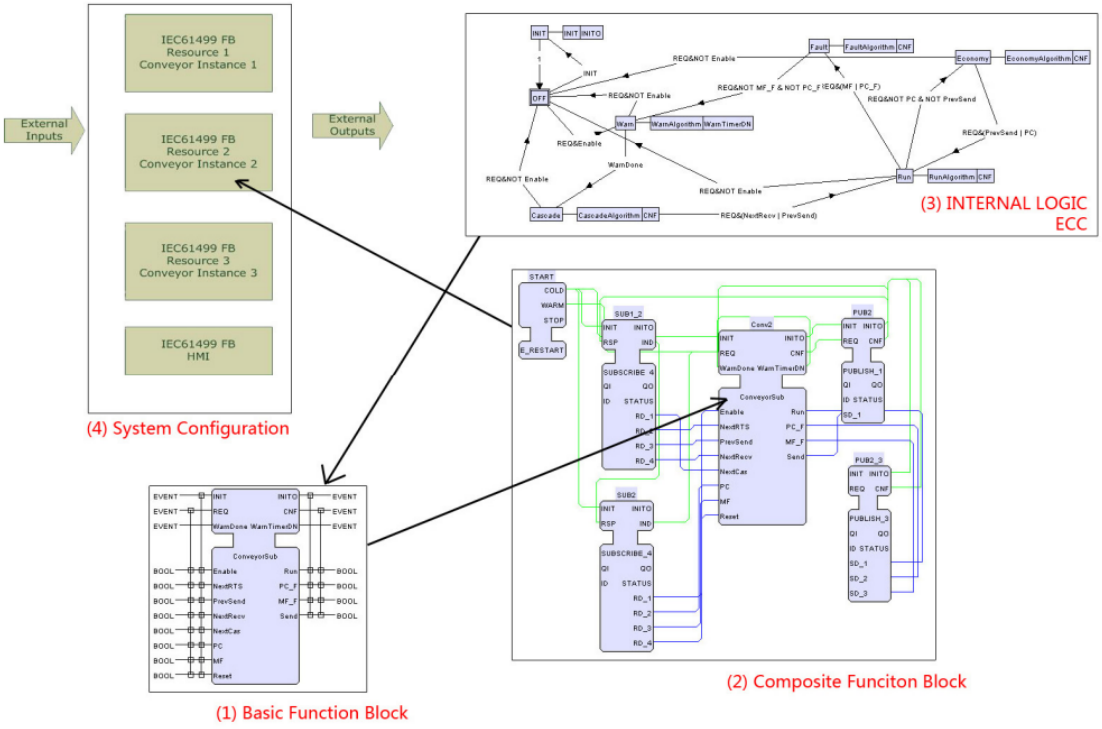
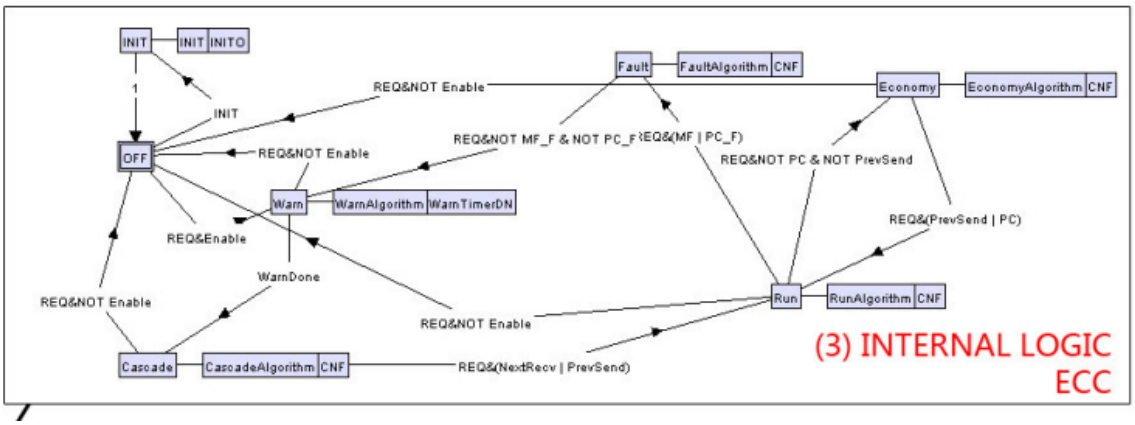


Fig.5. IEC 61499 FB implementation of Control Conveyor System.

Kuva 2. Toimilohko-ohjelmoinnin perusperiaate kaaviona [2].

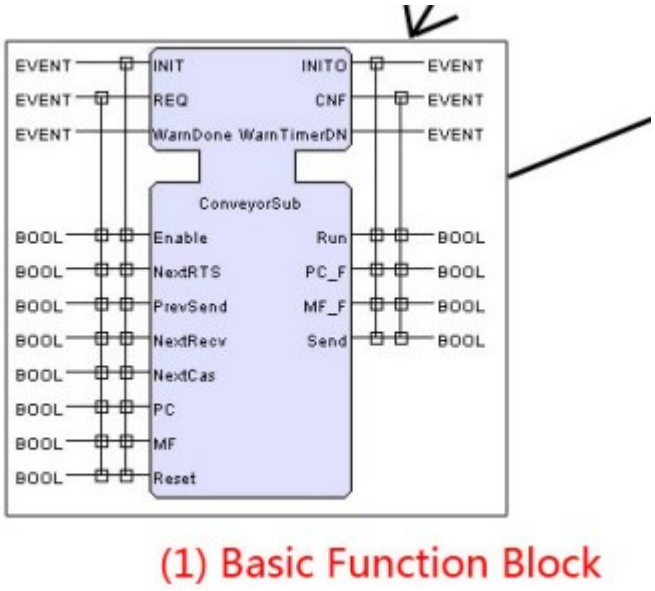
Kuvan 2 oikeassa yläkulmassa on kuvattuna *Internal Logic ECC,* eli kaavio tilakoneen toimintaperiaatteesta ja siitä, miten eri sisääntulot vaikuttavat toimilohkon aktivoitumiseen. Tämä osuus kuvasta on esitetty suurennettuna kuvassa 3.

Tilakoneen ja komposiittitoimilohkon toimintaa on selitetty tarkemmin luvuissa 3.1 ja 3.2.



Kuva 3. Tilakoneen sisäinen toimintakaavio [2].

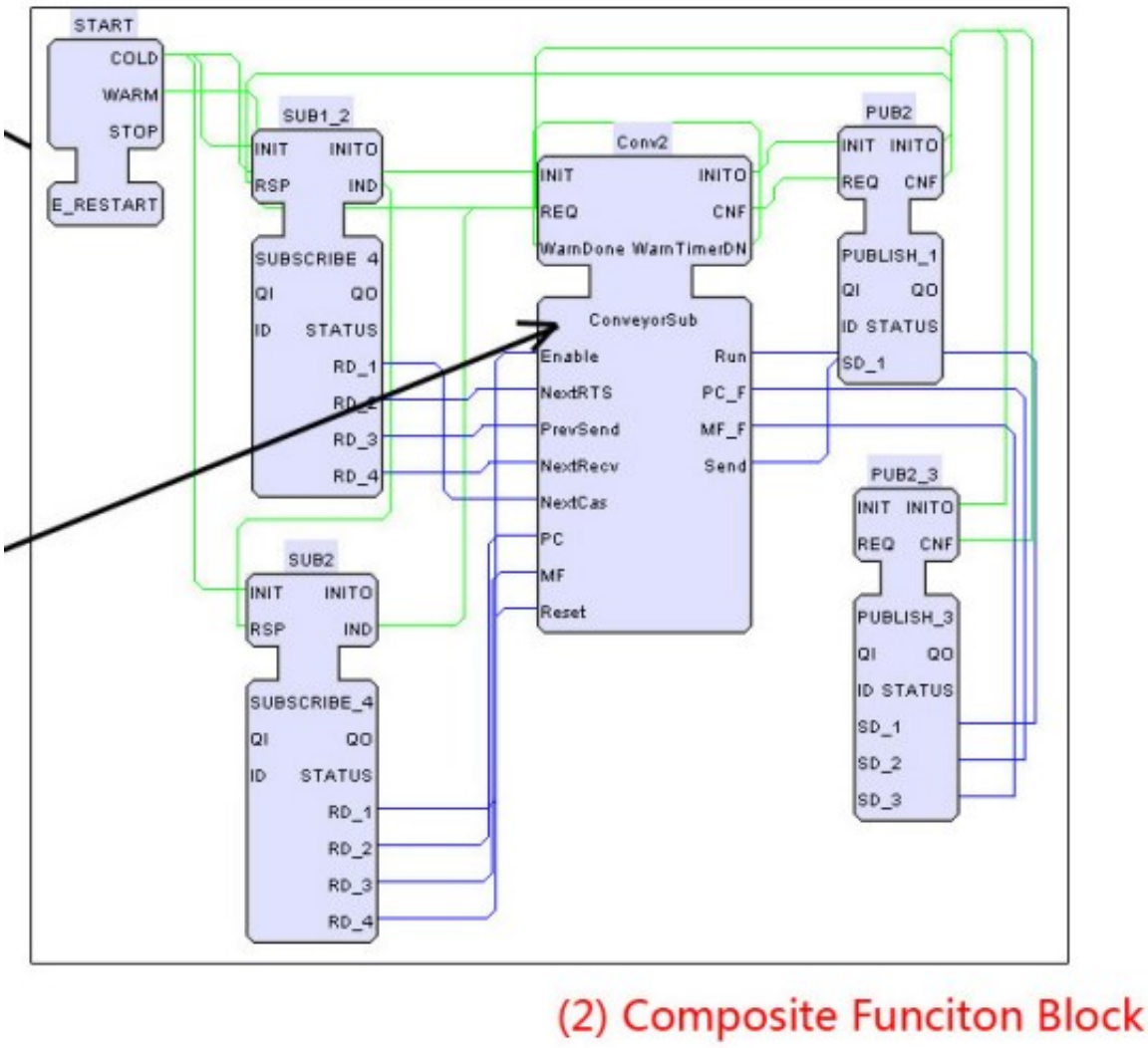
Kuvan 2 vasemmassa alareunassa on esitetty *Basic Function Block* -perustoimilohko, tämä suurempana kuvassa 4. Kaaviosta nähdään, että dataportteihin voidaan tämän toimilohkon tapauksessa syöttää ainoastaan *BOOL*-tyyppisiä muuttujia, eli signaali on joko 1 tai 0, on tai off. Tämä ei tarkoita, että toimilohkot yleisellä tasolla käsittelisivät vain boolean-arvoja, vaan kyseessä on vain esimerkki. Yksinkertaisuudessaan perustoimilohko ottaa sisään tietoa, suorittaa sille loogisen prosessin ja toimittaa päivitetyn tiedon eteenpäin. [2; 5, s. 12.]



Kuva 4. Perustoimilohko [2].

Tämä perustoimilohko on osa isompaa kokonaisuutta, kuvan 2 oikeassa alareunassa esitettyä *Composite Function Block* -komposiittoimilohkoa, josta joissain yhteyksissä käytetään myös termiä kapseloitu toimilohko. Tämä on esitetty isompana kuvassa 5. Komposiittitoimilohkon sisällä on itse kuljetinta ohjaavan toimilohkon lisäksi toimintaindikaattori, hätäseis, vikatilaa indikoiva toimilohko sekä käyttöliittymän toimilohko. Nämä voidaan löytää myös kuvassa 3 esitetystä tilakoneen kaaviosta. [2.]

Tämän esimerkin tapauksessa kaikkien muiden paitsi itse kuljettimen toimilohkon ohjelmointiin on käytetty IEC 61131-3:n mukaista tikapuukaaviota ja itse kuljettimen toimilohko on tehty IEC 61499:n mukaisesti. Koska kuljettimen toimilohko toimii tilakoneena, voidaan kapseloidusta toimilohkosta aktivoida kerrallaan vain osa tai suorittaa kaikki sen sisälle kapseloidut toiminnot tapahtumasisääntuloon tulevan signaalin perusteella. [2.]



Kuva 5. Komposiittitoimilohko eli kapseloitu toimilohko [2].

Edellä kuvattujen ominaisuuksien lisäksi kuvassa 2 on vielä esitetty *System Configuration,* jossa on kuvattu kolmehihnaisen kuljettimen resurssit (kolme kuljetinta) ja HMI-toimilohko, käytännössä start- ja stop-painikkeet. Kaikki nämä ominaisuudet ja toiminnot kuvataan ylätason näkymässä yhdellä yksinkertaisella laatikolla *Conveyor t*ai kuvan 1 esimerkissä RP. [2.]

Mikäli kaikkien toimilohkojen kaikki toiminnot olisi esitetty suurien järjestelmien ylätason näkymissä, olisi kokonaisuuden hahmottaminen hyvin haastavaa, ellei jopa mahdotonta. Kun ylemmältä tasolta voidaan siirtyä alemmas, lähemmäs

perustoimintoja, klikkailemalla lohkoja auki vain tarvittaessa, säilyy kokonaisuuden luettavuus parempana.

Tämäntyyppinen toimintojen kapseloiminen helpottaa myös niiden siirtämistä kohteesta toiseen. Mikäli tietty osa kokonaisuudesta vikaantuu tai laitteisto liitetään osaksi suurempaa kokonaisuutta, voidaan kapseloidut toiminnot linkittää toisiinsa merkittävästi pienemmällä vaivalla verrattuna vanhan standardin mukaisiin toteutuksiin. Hajautetussa järjestelmässä sen eri osilla on jokaisella omaa älyä ja ne kommunikoivat keskenään, jotta kokonaisuus pysyy yhtenäisenä. Tämän vuoksi standardoidun kommunikaation merkitystä on korostettu uudistetussa standardissa.

Toimilohkojen suorittamien funktioiden ohjelmointiin voidaan käyttää kaikkia vanhemmassa IEC 61131-3 -standardissa määriteltyjä kieliä, jotka käydään pääpiirteittäin läpi luvussa 2.1. Niiden lisäksi uudempi standardi kattaa myös korkeamman tason Java- ja C-ohjelmointikielet. [2.]

2.1 Korvattavan IEC 61131-3 -standardin haasteet

Opinnäytetyön pääkohteena olevaa standardia edeltävässä versiossa, joka oli julkaistu 1992, on määritelty viisi pääkieltä:

- *Structured text* (ST) – rakenteellinen teksti
- *Ladder Diagram* (LD) – tikapuukaavio
- *Instruction List* (IL) – käskylista
- *Sequential Function* Chart (SFC) – sekvenssikaavio
- *Function Block Diagram* (FBD) – toimilohkokaavio.

Vaikka PLC-laitevalmistajat käyttivät standardissa määriteltyjä kieliä, niitä ei ollut rajoitettu vain näihin. Tämä aiheutti jatkuvia yhteensopivuusongelmia eri valmistajien tuotteiden välillä. Vaikka yhden valmistajan käyttämä tikapuukaaviolla kirjoitettu ohjelma näyttäisi päällisin puolin täysin samalta, ei sitä voitu siirtää kopioimalla toisen valmistajan PLC-laitteeseen, koska muistipaikkojen käyttöä ei

ollut standardoitu. Globaalit ja päällekkäiset muuttujat tekivät saman ohjelmanpätkän käyttämisen eri laitteissa käytännössä mahdottomaksi. [2.]

IEC 61131-3 -standardiin liittyy 62 erilaista ominaisuustaulukkoa, joihin laitevalmistaja kirjaa kommenttien kera, miltä osin standardin mukaisia ominaisuuksia ja toimintoja on noudatettu. Taulukkojen perusteella pystyi esittämään arvauksia, olisiko mahdollista saada kahden eri valmistajan PLC:t kommunikoimaan keskenään, mutta varmuuden asiasta sai vasta, kun laitteita yritti saada toimimaan samassa järjestelmässä. [3, s. 12.]

2.2 IEC 61499:n edut verrattuna IEC 61131-3:een

Vaikka vanhempaa standardia IEC 61131-3 käytettiin jo hyvin laajasti eri valmistajien ja loppukäyttäjien kesken, jonkinasteisia yhteensopivuusongelmia havaittiin eri PLC-valmistajien välillä. Jos käyttäjä halusi siirtyä tietyn valmistajan PLC-laitteesta toisen valmistajan tuotteeseen, ohjelmistoa ei välttämättä saatu siirrettyä suoraan PLC:stä toiseen, vaan ohjelma jouduttiin kirjoittamaan kokonaan uudelleen. [2.]

Hajautettujen järjestelmien tarve teollisuudessa kasvaa jatkuvasti laitekokonaisuuksien monimutkaistuessa ja digitalisoituessa. Sen lisäksi, jos ohjausyksikkö vikaantuu keskitetyssä järjestelmässä, koko järjestelmä on toimintakyvytön. Jos hajautetussa järjestelmässä yritettiin käyttää eri valmistajien PLC:itä, niiden välinen kommunikaatio hyvin harvoin onnistui vanhemman IEC 61131-3 -standardin mukaan valmistetuilla tuotteilla. Tämän takia modulaarisuus ja uudelleenkäytettävyys on moderneissa järjestelmissä ja standardeissa erityisen tärkeää varsinkin pienille ja keskisuurille yrityksille. [2; 7.]

IEC 61499 -standardi laajentaa IEC 61131-3:n määriteltyjä ominaisuuksia parantamalla toimilohkojen kapselointia, jotta yksittäiset ohjelmakomponentit olisivat paremmin uudelleenkäytettävissä valmistajasta riippumattomassa formaatissa, sekä yksinkertaistamalla kontrollereiden välistä kommunikaatiota. Hajautettu järjestelmä tukee luonnollisesti dynaamista uudelleenkonfigurointia ja siten

tarjoaa tarvittavan infrastruktuurin erilaisille teollisuuden digitaalisointiin liittyville toteutuksille. [6.]

3 IEC 61499 -standardin mukaisten toimilohkojen toimintaperiaate

Luvussa 3 käydään läpi standardin IEC 61499 asettamat perusominaisuuksien vaatimukset. Dokumentoinnissa esitetään toimilohkojen toimintaperiaatteet ja tiedon siirtyminen lohkojen ja tiedonkäsittelynoodien välillä.

3.1 Toimilohkon perustoiminnot

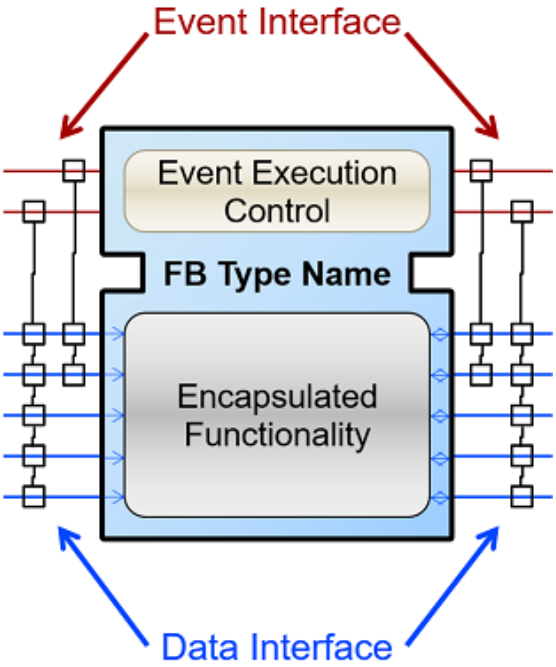
Kuvassa 6 on nähtävissä IEC 61499:n mukainen perustoimilohko. Toimilohkossa on kapseloituna haluttu funktio tai toimintokokonaisuus, eli komposiittitoimilohko, on kuvattu aiemmin luvun 2 kuvassa 5.

Kuvassa 6 lohkon sisääntulot on esitetty vasemmalla ja ulostulot oikealla. Tärkeänä erona verrattuna IEC 61131-3:n mukaisiin toimilohkoihin on, että IEC 61499 erottelee tapahtumat *Event,* punaiset merkinnät ja datan *Data,* siniset merkinnät. Jotta toimilohko aktivoituu suorittamaan haluttua toimintoa, täytyy *Event*-sisääntuloon tulla signaali jostain sitä ohjaavasta lähteestä, esimerkiksi toisesta toimilohkosta, sensorista tai käyttäjän pyynnöstä. Pelkkä uuden datan saapuminen toimilohkolle ei käynnistä lohkon toimintaa. [6.]

Ohjelma suoritetaan IEC 61499 -standardin mukaisessa ohjelmoinnissa tapahtumaperusteisesti *Event*, kun vanhemmassa standardissa ohjelma suoritetaan syklisesti – aina kokonaan ja ylhäältä alaspäin. Tapahtumasisääntulo on yhdistetty useisiin datasisääntuloihin, kuten kuvasta 6 nähdään. Sama on havaittavissa ulostuloissa. Mustat viivat määrittelevät, mitkä datan sisään- ja ulostulot päivitetään, kun kyseinen tapahtumasisääntulo vastaanottaa signaalin. [6.]

Tämä toimilohkon aktivoituminen riippuu ECC:stä *Event Control Chart*. ECC on tilakone, joka vastaanottaa signaalit tapahtumasisääntuloista. Riippuen lohkon

sen hetkisestä tilasta ECC suorittaa joko haluttuja osia kapseloidusta *Composite* funktiosta tai tarvittaessa koko funktion. [6.]



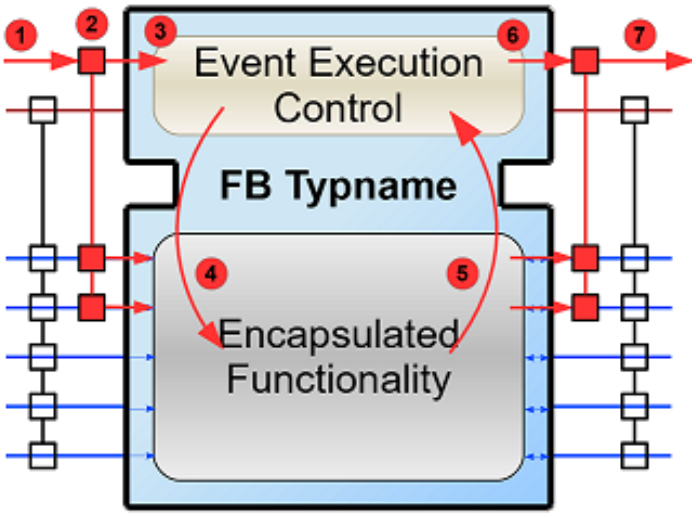
Kuva 6. IEC 61499 -standardin mukainen toimilohko [6].

Tapahtuma- ja datakytkennät eivät ole suoraan yhteensopivia, joten portteja ei voi yhdistellä täysin mielivaltaisesti. Luvussa 2 olevassa kuvassa 2 on kuvattu toimilohkon kytkentöjen toimintaa tarkemmin.

Dataulostuloja voidaan jakaa. Jakaminen tarkoittaa sitä, että yksittäinen ulostulo voidaan yhdistää useisiin sisääntuloihin seuraavassa vaiheessa. Sitä vastoin datan sisääntuloja ei voida hakea useasta lähteistä, koska silloin toimilohkon pitäisi kyetä valitsemaan tilanteeseen sopiva sisääntulo ja siihen käytössä olevat lohkot eivät kykene. Tapahtumien sisään- tai ulostuloja voidaan puolestaan hakea ja lähettää useista lähteistä tarpeen mukaan. [4.]

Kuvassa 7 on esitetty järjestys, jossa asiat tapahtuvat, kun tapahtumadiskreetin toimilohkon tapahtumasisääntulo aktivoituu.

1. Tapahtumasisääntulo aktivoituu.
2. Tapahtumasisääntuloon liitetyt datasisääntulot päivitetään.
3. Tapahtuma toimitetaan ECC:lle.
4. Riippuen toimilohkon tyypistä ja ECC:stä sisäiset funktiot suoritetaan.
5. Kun funktio on suoritettu, toimilohko toimittaa uuden datan dataulostuloihin.
6. Dataulostulot päivitetään.
7. Ulostulotapahtuma lähetetään eteenpäin.



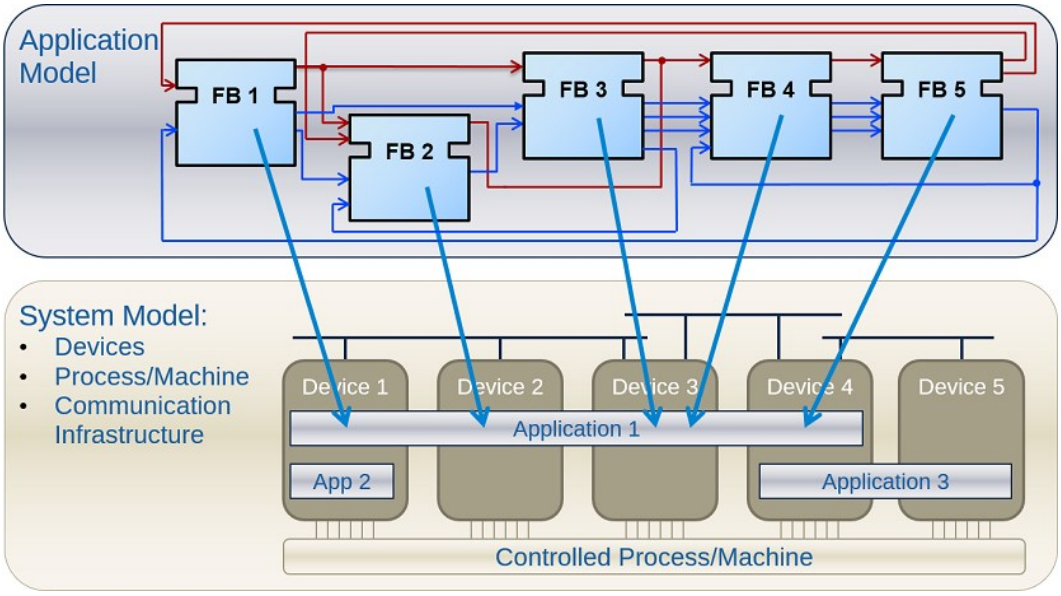
Kuva 7. Tapahtumasisääntulon aktivoituminen [6].

Riippuen kapseloidusta funktiosta kohtia 4–7 voidaan toistaa tarvittaessa useita kertoja. Tapahtumaulostulo voidaan antaa myös useita kertoja tarvittaessa ilman, että sisään tulevaan tapahtumaliittymään saapuu uutta signaalia. Kapseloitu funktio voi esimerkiksi jäädä pyörittämään kuljettimen hihnaa siihen saakka, kunnes kuljettimen loppupään sensori tunnistaa kappaleen ja pyytää toimilohkoa pysäyttämään kuljettimen.

3.2 Toimilohkojen välinen kommunikaatio

Luvussa 3.1 esiteltiin toimilohkojen sisäistä toimintaa. Tässä luvussa käydään läpi toimilohkojen välistä toimintaa. Toimilohkolla tarkoitetaan tässä yhteydessä toimintokokonaisuuksia, joiden alla saattaa olla pienempiä kapseloituja toimintoja.

Kuvassa 8 nähtävästä kaaviosta nähdään, kuinka toimilohkot kommunikoivat keskenään. IEC 61499 -standardi kattaa hajautettujen systeemien käytön, eli sovelluksen suoritus ei tapahdu välttämättä vain yhdellä keskuskoneella. Sen sijaan kokonaisuus voi olla jaettuna usealle laitteelle, joko PLC:ille, PC:ille tai virtuaalikoneille. Käytössä voi olla myös useita eri sovelluksia, jotka on jaettu usealle eri laitteelle. Kuvassa 8 esitetty systeemin mallinnuskaavin auttaa käyttäjää hahmottamaan ja suunnittelemaan tätä jakoa. [6.]



Kuva 8. Esimerkki toimilohkojen *FB*, laitteiden *Device* ja sovelluksien *Application* välisistä suhteista [6].

Kuvasta 8 voidaan siis nähdä, etteivät kaikki sovelluksen toimilohkot välttämättä ole samalla laiteella. Tämän lisäksi yksittäisellä laitteella saattaa olla käynnissä useita sovelluksia tai sovelluksen osia. On syytä huomioida, että yksittäinen

toimilohko kuitenkin toimii vain yhdellä laitteella, eikä sitä voida jakaa useampaan kohteeseen. [6.]

4 Markkinoilla olevat kaupalliset ohjelmointiympäristöt

Tässä luvussa esitellään muutamia markkinoilla tarjolla olevia ohjelmistoratkaisuja, joiden ohjelmointiin on käytetty IEC 61499 -standardin mukaisia toteutuksia. Koska oppilaitoksella tai työpaikalla ei ole missään käytössä näitä järjestelmiä, ominaisuudet ja niiden käyttömahdollisuuksien esittelyt perustuvat täysin yritysten omaan markkinointimateriaaliin.

4.1 Schneider EcoStruxure™ Automation Expert

Schneider Electricin IEC 61499 -yhteensopiva kehitysalusta on nimeltään EcoStruxure™ Automation Expert. Laitevalmistajan mukaan nopeasti muuttuvat tilanteet maailmanmarkkinoilla vaativat uusilta järjestelmiltä nopeaa mukautuvuutta. Teollisuus ja palveluntarjoajat joutuvat usein miettimään lähestymistä olemassa oleviin ratkaisuihinsa täysin uusista lähtökohdista, minkä vuoksi Schneiderin tarjoama EcoStruxure on merkittävästi kilpailijoiden tarjoamia vaihtoehtoja parempi. [8.]

Nykyään käytössä olevat automaatiojärjestelmät ovat peräisin pitkälti 1970- ja 1980-luvuilta ja toimivat vahvasti reaaliaikaisten syötteiden perusteella. Tietojenkäsittelyn kehittyminen, tekoäly, IIoT ja *Industry 4.0* kuitenkin mahdollistaisivat datan käsittelyn ja ennakoivat reagoinnit erilaisiin tilanteisiin. [8.]

Tämän takia EcoStruxture on laitevalmistajan mukaan soveltuva kaikkeen mahdolliseen automaatioon tuotannosta aina taloautomaatioon ja datakeskuksien tiedonkeruuseen. Järjestelmä kerää tiedon ja toimittaa sen automaattisesti pilvipalveluun, josta se on helposti saavutettavissa myös toimipisteen ulkopuolelta. Täten tiedonkäsittely voidaan toteuttaa jossain kauempana kohteissa, joihin datankäsittelypisteiden rakentaminen keruupisteiden välittömään läheisyyteen olisi epäkäytännöllistä. [8.]

Alustalla on useita sisäänrakennettuja visualisointivaihtoehtoja, joilla data saadaan helposti hahmotettavaan muotoon ilman suurta viivettä tapahtumien ja visualisointien välillä. Koska järjestelmä on yhteensopiva Microsoft Azuren kanssa, erilaisia digitaalisia palveluita voidaan kytkeä automaatiojärjestelmään helposti. [6.]

Vaikka tietoturva on yleiselläkin tasolla hyvin tärkeää, digitalisoituvassa teollisuudessa sen merkitys korostuu. Tämän vuoksi Schneider tarjoaa verkkosivuillaan useita erilaisia koulutusmateriaaleja, joissa käyttäjää opastetaan luomaan turvallinen verkko eri laitteiden ja laitekokonaisuuksien välille. Tietoturva on vain niin vahva, kuin sen heikoiten suojattu kohde. [9.]

4.2 ISaGRAF, Rockwell Automation, SoftPLC

ISaGRAF Workbench on Rockwellin ohjelmointiympäristö, jonka luvataan olevan intuitiivinen ja kaikenkattava alusta, jolla on helppo toteuttaa erilaisia ohjelmia sekä siirtää niitä laitteelta toiselle. ISaGRAF on ohjelmistovalmistajan mukaan yhteensopiva sekä IEC 61131-3- että IEC 61499 -standardien kanssa. Koska alusta tukee molempia standardeja, teoreettisesti käyttäjälle tarjotaan enemmän vaihtoehtoja käytettävän ohjelmointikielen käytössä. Workbench toimii Microsoft Visual Studion päällä, Windows-käyttöjärjestelmässä. [10.]

ISaGRAFilla toteutetut ohjelmat toimivat SoftPLC:llä, joten virallista PLC-yksikköä ei tämän ratkaisun yhteydessä tarvita. SoftPLC tarkoittaa käytännössä ohjelmoitavan logiigan ohjelmistopohjaista versiota, joka toimii tavallisella PC:llä. [7.]

Merkittävin ero ISaGRAFin ja muiden markkinoilla olevien IEC 61499 -yhteensopivien ratkaisujen välillä on, että ISaGRAF käyttää omaa yksilöllistä koodausta kirjastoissa. Standardin mukaan XML-koodausta tulisi käyttää kirjastoelementeissä, mutta ISaGRAF ei näin tee. Tämän takia käyttäjien keskuudessa on kyseenalaistettu ISaGRAFin yhteensopivuutta IEC 61499 -standardin kanssa ja pohdittu, onko kyseessä vain markkinointilupaus. [11, s. 39.]

4.3 NxtStudio

NxtStudio on itävaltalaisen nxtControl GmbH:n tarjoama IEC 61499- ja IEC 61131-3 -yhteensopiva ympäristö. Alusta on optimoitu hajautettujen järjestelmien ohjelmointiin ja tarjoaa hyviä visualisointivaihtoehtoja erilaisten laitteistokokonaisuuksien valvontaan. Ohjelmisto on maksullinen, mutta siitä on ollut ilmainen demoversio ladattavissa verkkosivuilla. Tämän opinnäytetyön kirjoittamishetkellä kyseistä linkkiä tai sivua ei kuitenkaan löytynyt.

Yritys kuitenkin lupaa, että nxtControl olisi ”yhden pysähdyksen ratkaisu kaikkeen automaatioon” pitäen sisällään kaiken mahdollisen tuotantoautomaatiosta taloautomaatioon ja varastotasojen seurantaan. Kaikki monimutkaisuus on piilotettu käyttäjältä ja virheiden sattuminen on tehty mahdottomaksi. [13.]

Yrityksen tärkein markkinointivaltti vaikuttaa olevan ohjelmistojen ja laitteiden elinkaaren erottaminen toisistaan. Ohjelmistot ovat käytännössä lähes ikuisia niin kauan kuin yhteensopivia laitteita on saatavilla, ja nxtControl lähestyy tätä ongelmaa jonkin tyyppisellä virtuaalikoneella. Tätä ei ole verkkosivuilla selitetty tai avattu mitenkään, joten syvällisempää analyysiä varten täytyisi olla yhteydessä yritykseen. [11.]

5 Avoimen lähdekoodin ohjelmointiympäristöt

Vaikka eri valmistajien tuottamat PLC-ratkaisut yhdenmukaistuvat jatkuvasti sekä ohjelmisto- että laitteistoteknisiltä ominaisuuksiltaan, niissä on edelleen merkittäviä eroja. Opiskelijan ja kehittäjän kannalta OEM:n asettamat rajoitteet aiheuttavat siksi usein ongelmia ja kohtuuttomia kustannuksia projektien kertaluonteisuuden huomioon ottaen.

Runsas harjoittelu ja käytännön kokemus ovat tärkeässä avainasemassa sujuvan ohjelmoinnin oppimisessa. Ongelmien ratkominen itsenäisesti ja virheiden korjaaminen on yleensä tehokkainta oppimista. Usein laitevalmistajien omat ohjelmointiympäristöt vaativat kalliita lisenssejä, eikä niiden hankinta ole

varsinkaan opiskelijalle järkevää. Jos tietyn alustan käyttö on mahdollista vain kampuksella ja sielläkin vain muutamalla työpisteellä, riittävä harjoittelu on lähes mahdotonta. Markkinoilla olevat avoimet ohjelmointiympäristöt pyrkivät ratkaisemaan ongelmaa tarjoamalla kokonaan ilmaisia ympäristöjä sekä niihin sopivia toimilohkokirjastoja.

5.1 HOLOBLOC FBDK

FBDK *Function Block Developement Kit* on yhdysvaltalaisen HOLOBLOC-yrityksen IEC 61499 -standardin kanssa yhteensopiva Java-pohjainen ympäristö, joka toimii sekä Linux-, Raspbian-, että Windows-käyttöjärjestelmissä. Alustan käyttö vaatii sekä itse ympäristön *FBEditor* että *runtime* osuuden *FBRT*. Molemmat komponentit ovat ladattavissa vapaasti verkkosivuilta. FBDK oli yksi ensimmäisistä IEC 61499 -yhteensopivista työkaluista, ja se edisti merkittävästi standardin leviämistä käyttäjien keskuuteen. [12.]

Ympäristön asentaminen ja käynnistäminen on tehty hyvin helpoksi, sillä molemmat komponentit ovat ladattavissa yhdessä .zip-tiedostossa. Kun pakkaus on purettu halutulle kovalevylle, ympäristö käynnistyy yksinkertaisesti klikkaamalla .exe-tiedosto auki. Koska ympäristö toimii Java-pohjaisesti, on muistettava varmistaa, että käytettävällä PC:llä on toimiva Java ennen ympäristön käynnistämistä. [12.]

5.2 Eclipse 4diac

Eclipse 4diac -ympäristö on täysin ilmainen avoimen lähdekoodin ohjelmointialusta. Ympäristöön kuuluu IDE-kehitysympäristö ja FORTE *runtime environment*. Jotta ympäristöä pääsee kokeilemaan, täytyy verkkosivulta ladata vähintään IDE-paketti, ja jos tehtyä ohjelmaa haluaa testata paikallisesti, myös FORTE-paketti täytyy ladata. Koska IDE-kehitysympäristö on kirjoitettu Javalla ja perustuu Eclipse *frameworkiin,* kannattaa myös varmistaa, että käytössä olevan tietokoneen Java on ajan tasalla. [14.]

Alustan graafinen käyttöliittymä on melko intuitiivinen ja ohjelman sisäänrakennetut lisäominaisuudet, kuten monitorointi ja *debuggaus,.* helpottavat tehdyn ohjelman toiminnan testaamista merkittävästi.

Koska kyseessä on avoimeen lähdekoodiin perustuva ohjelmointiympäristö, käyttäjät jakavat mielellään kokemuksiaan ja etsivät ratkaisuja ongelmiin ohjelmointiympäristön virallisilla keskustelufoorumeilla. Tietty yhteisöllisyys helpottaa merkittävästi esiin tulevien ongelmien ratkaisemisessa, eikä vastausta tarvitse aina odottaa (ja maksaa) OEM:ltä.

Käytännön projektina luvussa 6 on tutustuttu tämän ohjelmointiympäristön perustoimintoihin ympäristön sisäänrakennetun Help-valikon tutoriaalin mukaan. Sama tutoriaali on löydettävissä myös verkkodokumenttina ympäristön omilta verkkosivuilta. [15.]

6 Esimerkkiohjelma Eclipse 4diac -ohjelmointiympäristössä

Tässä luvussa esiteltävän 4diac -ohjelmointiympäristön sisäänrakennetun tutoriaalin avulla saa kohtuullisen hyvän käsityksen ohjelman perustoiminnoista. Ohjelmoinnissa tämän tyyppisiä kieleen johdattelevia perusohjelmia kutsutaan yleensä ”Hello World” -koodiksi niiden yksinkertaisuuden vuoksi. [15.]

Tutoriaalin tarkoituksena on johdatella käyttäjä ohjelmointiympäristön toimintoihin. Käyttäjältä odotetaan perustason tietämystä tietotekniikasta ja ohjelmoinnista, mutta mikäli käyttäjällä on edes vähäistä harrastuneisuutta alalta, projekti on melko helppo saada onnistumaan. Oman kokeilun suurimmaksi kompastuskiveksi muodostui toimivan Javan asentaminen. Alussa käytössä olleeseen vanhempaan kannettavaan tietokoneeseen ei löytynyt yhteensopivaa Javaa ja aikaa tuli tuhlattua ongelman kanssa painimiseen turhan paljon.

Luvussa 6.1 rakennetaan hyvin yksinkertainen toimilohko-ohjelma, jonka tarkoitus on tuottaa yhden sekunnin intervalleilla vaihtuvaa on ja off -tyyppistä ulostuloa. Ohjelman luomiseen käytetään ohjelmointiympäristön mukana tulevaa

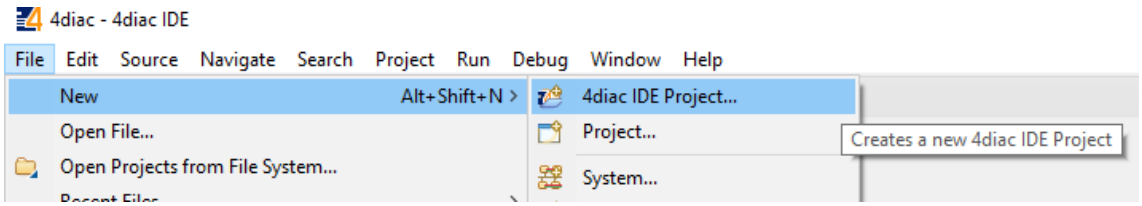
valmista toimilohkokirjastoa. Tässä luvussa käydään myös läpi ympäristön tärkeimpiä toimintoja ja navigointivaihtoehtoja.

Luvussa 6.2 tutustutaan ohjelman laitteistoteknisiin ominaisuuksiin, joissa ohjelmalle määritellään käytettävät resurssit. Luvussa 6.3 tehty ohjelma, eli toimilohkot, linkitetään tälle määritellylle resurssille. Luvuissa 6.4 ja 6.5 testataan kokonaisuuden toiminta FORTE-virtuaalikoneessa.

Tätä sovellusta voitaisiin käyttää esimerkiksi merkkivalon vilkkumiseen laitteen piilossa olevan kuljettimen lähellä, näkyvällä paikalla, jolla voitaisiin päätellä kyseisen kuljettimen aktiivisuus visuaalisesti.

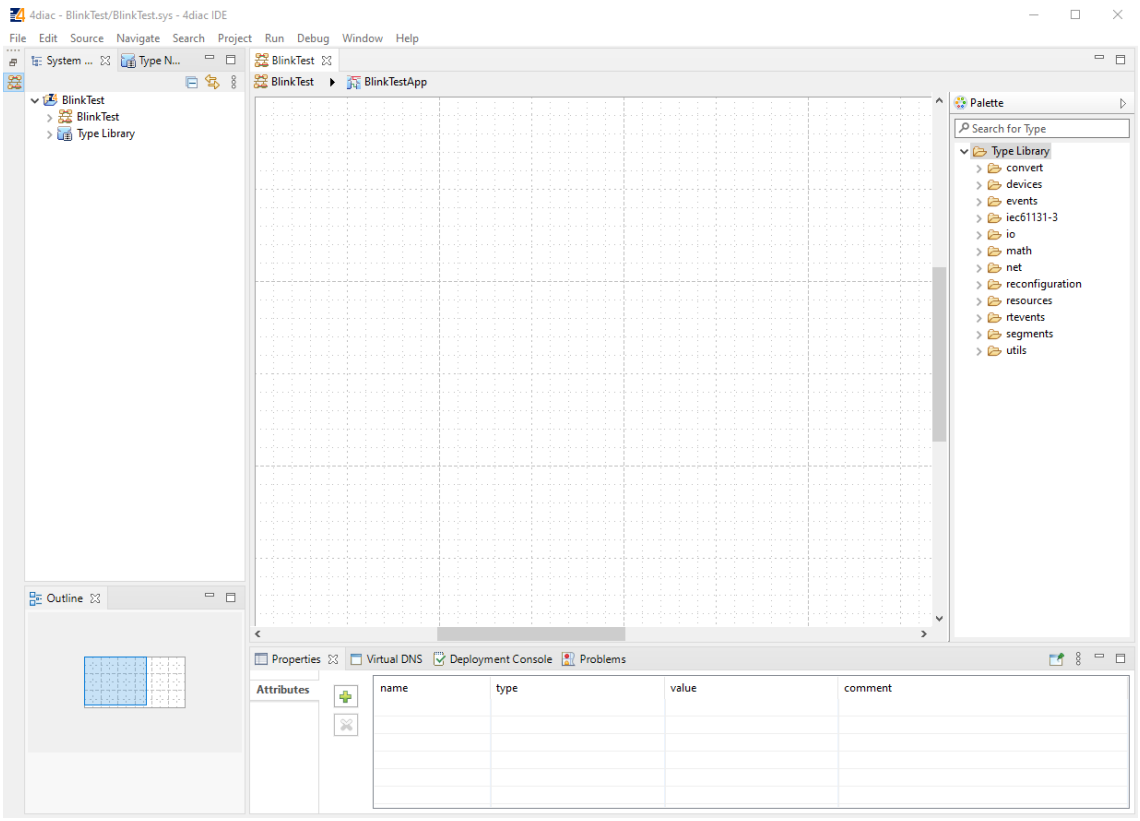
6.1 Yksinkertaisen ohjelman luominen

Työ aloitetaan luomalla uusi projekti valitsemalla *4diac IDE Project*, kuten kuvassa 9 on esitetty. Mikäli valitaan pelkästään *Project*, luodun ohjelman testaaminen vaikeutuu myöhemmin merkittävästi, sillä ohjelmointiympäristö jättää luomatta yhteyksiä linkityksiä varten. Linkkien luominen manuaalisesti myöhemmin on mahdollista, mutta mikäli käyttäjä haluaa välttää ajanhukkaa, kannattaa tässä vaiheessa klikata oikeaa painiketta.



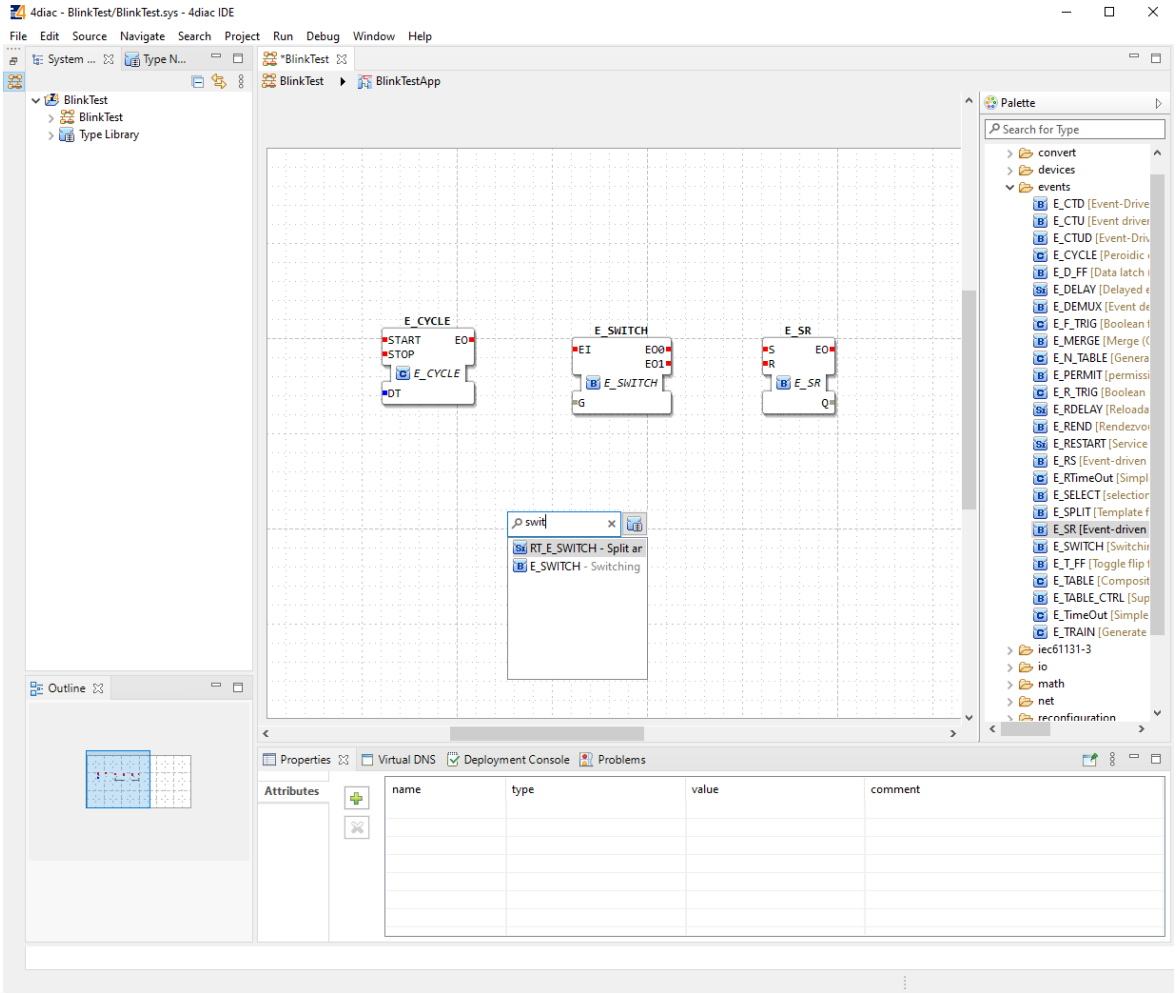
Kuva 9. Uuden projektin luominen [16].

Tutoriaalia jatketaan esittelemällä työtilan, eli W*orkspacen,* valikoita kuvan 10 mukaisesti. Valikot ovat selkeitä ja haluttujen toimintojen löytäminen ei ainakaan harjoituksen perusohjelmoinnissa ollut haastavaa.



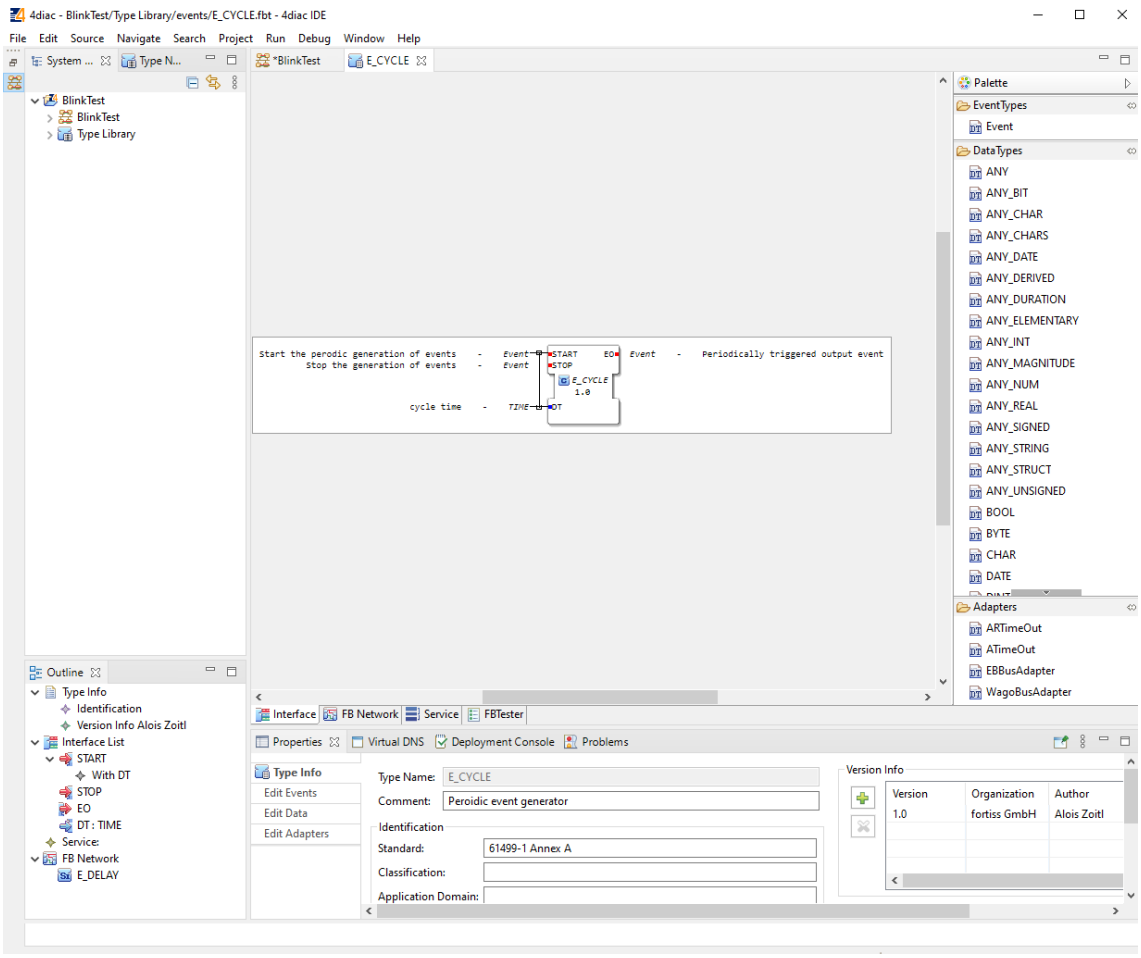
Kuva 10. Työtila, eli *Workspace* [16].

Ruudun vasemmassa reunassa on ohjelman perusrakenne pudotusvalikoissa, ja oikeassa reunassa näkyy asennuspaketin mukana tulleet perustoimilohkot. Tästä kirjastosta toimilohkot saa ohjelmaan helposti raahaamalla lohkot editointialueelle, isoimpaan ruutuun keskellä ikkunaa. Toinen vaihtoehto on kaksoisklikata editointialueella tyhjässä kohdassa hiiren vasemmalla painikkeella ja syöttää avautuvaan hakupalkkiin hakusana ja valita tuloksista haluttu toimilohko. Nämä valikot näkyvät kuvassa 11.



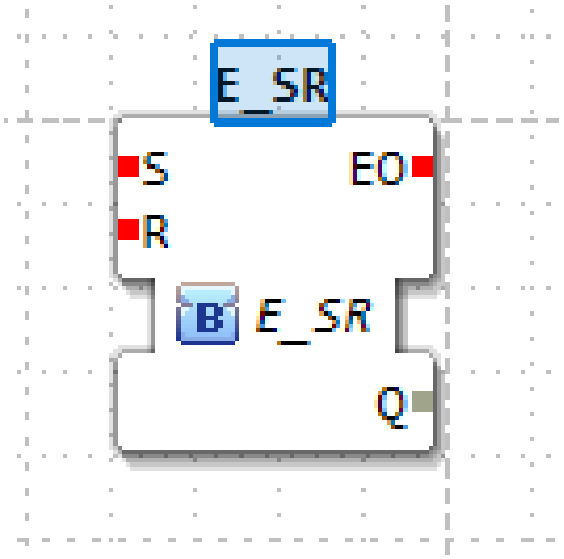
Kuva 11. Toimilohkot ruudulla [16].

Mikäli käyttäjä ei ole täysin varma toimilohkon operaatioista, *CTRL*+klikkaus tuo ruudulle informaatioruudun, joka näkyy kuvan 12 alemmassa osassa, jossa selitetään yksinkertaisesti lohkon toiminnot. Kyseisen valikon alla voidaan myös editoida toimilohkoa tarpeen vaatiessa. Käyttäjän ei siis ole pakko käyttää vain toimilohkokirjastossa olevia funktioita, vaan joskus voi olla tehokkaampaa rakentaa tiettyyn tarkoitukseen kokonaan oma toimilohko. Näin rakennetun toimilohkon voi tallentaa omaan kirjastoon tulevaisuutta varten tai jopa jakaa muille käyttäjille, esimerkiksi ohjelmointiympäristön omilla virallisilla foorumeilla. Tämä on yksi avoimeen lähdekoodiin pohjautuvien ympäristöjen vahvuuksista, sillä yksittäisen käyttäjän luoma toteutus saattaa olla hyödyllinen muillekin.



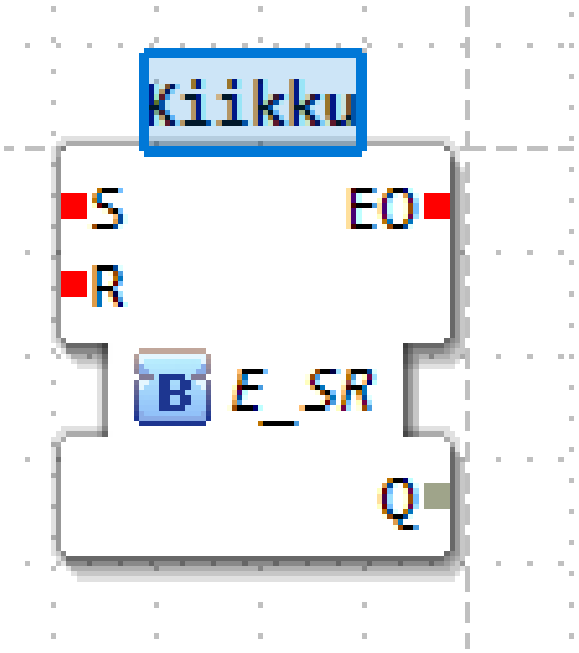
Kuva 12. Tietoja toimilohkosta ja toimilohkon editointi-ikkuna [16].

Toimilohkon yläpuolella oleva teksti on kyseisen toimilohkon nimi, joka on esitetty kuvassa 13. Yhdessä ohjelmassa ei voida käyttää samaa nimeä kahdessa paikassa, mutta onneksi sen editoiminen on helppoa – klikkaamalla nimeä aukeaa editointivalikko, jossa lohkon voi nimetä haluamallaan tavalla.



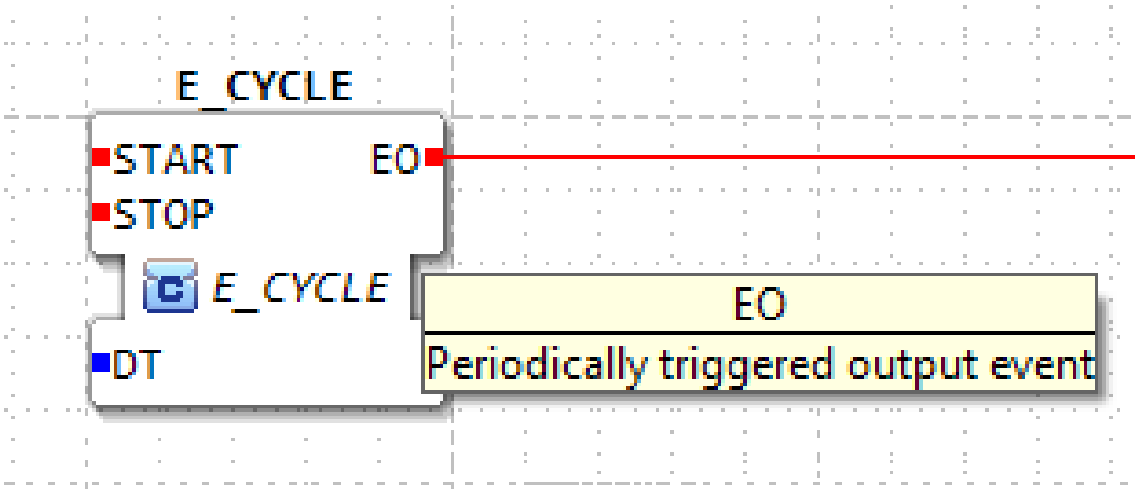
Kuva 13. Oletusnimi [16].

Kuvasta 14 nähdään miltä editoitu toimilohkon nimi näyttää editointinäkymässä. Vaikka editoiminen on melko helppoa, kannattaa kuitenkin muistaa nimetä toimilohkot loogisesti. Mikäli lohkon nimi on kuvaava, virheellisen toimivan toimilohkon paikallistaminen kokonaisuudesta helpottuu merkittävästi.



1. Kuva 14. Yksilöity nimi [16].

Hiiren vieminen erilaisten toimintojen, porttien, nimien tai viivojen päälle antaa tiivistetyn kuvauksen kyseisen kohdan toiminnasta, esimerkki tästä kuvassa 15. Tämä on hyvin tervetullut ominaisuus ohjelman käyttöä vasta opettelevalle käyttäjälle. Apuvalikoista ei välttämättä aina löydy kovin helposti tietoa siitä, mitä jokin tietty painike tai kytkentä tekee, joten tämän tyyppisestä lisätiedosta on usein paljon hyötyä.



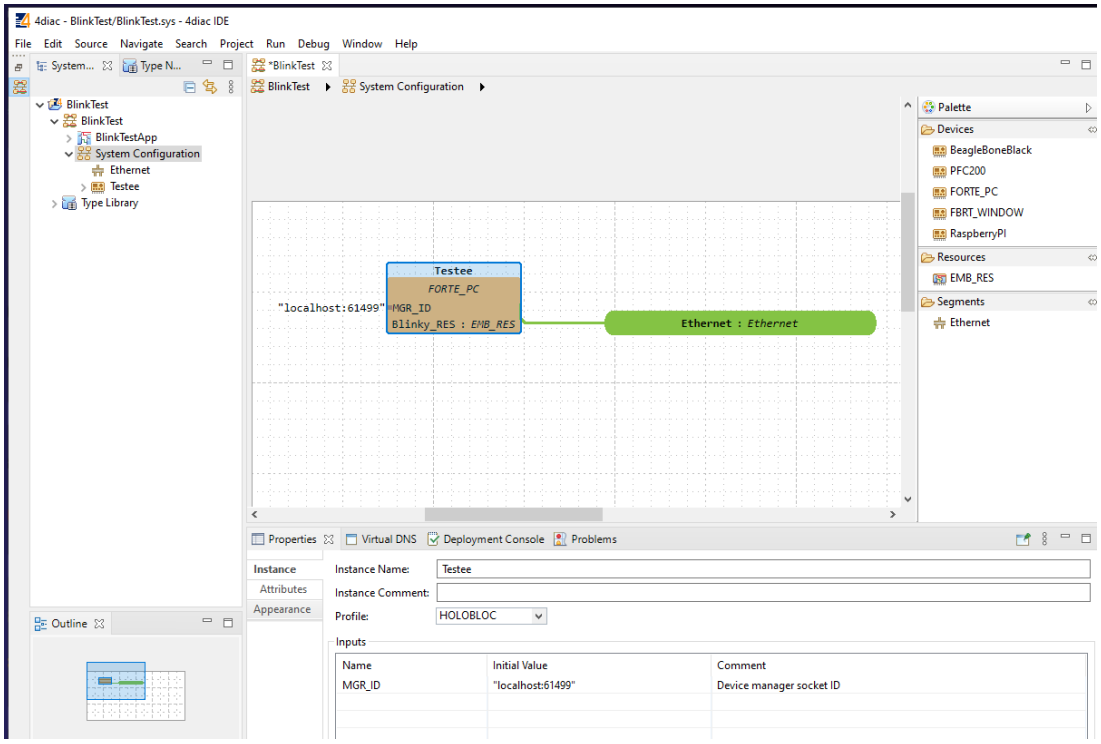
Kuva 15. Tiivistetty kuvaus [16].

Käyttäjän kannalta erilaisten yhteyksien viivojen värien muuttaminen voi selkeyttää ohjelman toiminnan hahmottamista sekä helpottaa toimimattomien yhteyksien paikallistamista. 4diac antaa kyllä ilmoituksen, mikäli käyttäjä yrittää yhdistää väärän tyyppistä dataa tai portteja toisiinsa, mutta pienehkö huutomerkki ruudun alareunassa saattaa jäädä huomaamatta.

6.2 Laitteiston konfigurointi

Laitteiston konfiguraatio onnistuu myös kohtuullisen helposti, ainakin demossa. Ruudun vasemmasta reunasta löytyy klikkailemalla *System Configuration* -välilehti, joka on esitetty kuvassa 16. Tässäkin editointinäkymässä toimintojen lisääminen ohjelmaan onnistuu raahaamalla valmiita laatikoita oikeassa reunassa olevasta kirjastosta. Mikäli haluttua toimintoa ei löydy, sen voi joko ohjelmoida itse tai etsiä internetistä kirjaston joka sisältää kyseisen toimilohkon.

Myös yksittäisiä lohkoja on mahdollista löytää keskustelupalstoilta, mikäli haluttu toiminto on tarkkaan määritelty.



Kuva 16. *System Configuration* [16].

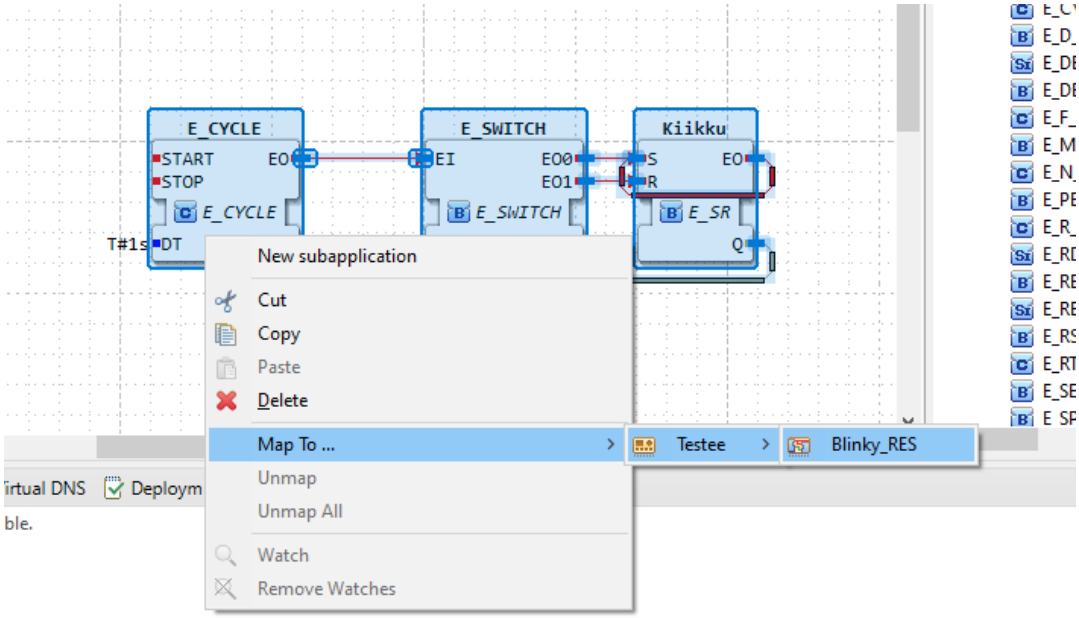
Resurssien uudelleennimeäminen ohjelman kannalta loogisiksi onnistuu samalla tavalla kuin toimilohkojen tapauksessa, eli kaksoisklikkaamalla nimeä ja muokkaamalla se halutuksi. Ohjelman luettavuuden kannalta on suotavaa nimetä ohjelman eri osat loogisesti. Mikäli laitteistokokonaisuudessa on esimerkiksi useita tietokoneita, niiden IP-osoitteiden pitäminen järjestyksessä helpottuu merkittävästi, jos ne on nimetty ohjelmointivaiheessa loogisesti.

Eri valmistajien laitteet tukevat erityyppisiä latausmekanismeja, mikä käytännössä tarkoittaa tapaa, jolla ohjelma siirretään ohjelmointiympäristöstä suorittavalle laitteelle. Tämän vuoksi ohjelmalle tulee kertoa, mitä laiteprofiilia halutaan käytettävän. 4diac-IDE tukee toistaiseksi kahta erilaista laiteprofiilia: HOLOBLOC, joka toimii laitteiden kanssa, jotka valmistajan mukaan ovat yhteensopivia IEC 61499 -standardin kanssa, ja FBDK laitteet, jotka on valmistettu ennen vuotta 2009. FBDK2-profiili taas on yhteensopiva sitä uudempien laitteiden

kanssa. Tämä ei toki tarkoita sitä, että kaikki laitteet ymmärtävät jompaakumpaa näistä. Käytännössä yhteensopivuus varmistuu edelleen vain kokeilemalla.

6.3 Laitteiden ja toimilohkojen linkittäminen

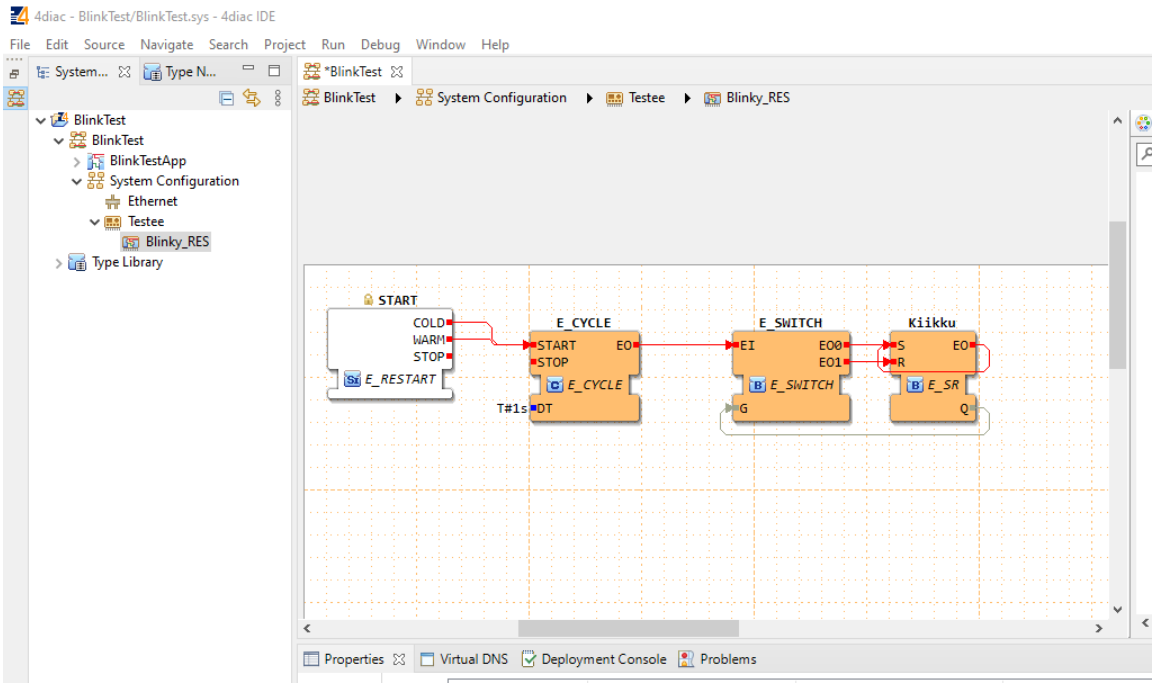
Niin kutsuttu toimilohkojen mappaus, eli linkittäminen, tehdään seuraavaksi, kuva 17. Tällä tarkoitetaan sitä, että laitteille kerrotaan, mitkä funktiot kuuluvat sille. Sitä varten palataan työtilan vasemmassa laidassa olevan *System-tree*:n alta takaisin sovellusnäkymään, valitaan haluttu toimilohko tai -lohkot ja klikataan valintaa hiiren oikealla. Valikon alta päästään klikkaamaan *Map To* ja listasta valitaan haluttu laite, eli resurssi.



Kuva 17. Toimilohkojen linkitys laitteelle [16].

Kun toimilohkot ovat linkitettyinä laitteelle, voidaan vasemmasta reunasta klikata auki kyseinen laite ja varmistaa, että linkitys on mennyt läpi. Valikkoa

kutsutaan resurssieditoriksi. Mikäli linkitys on onnistunut, editorissa pitäisi näkyä applikaatiovälilehdellä käytössä olevat toimilohkot, kuva 18.

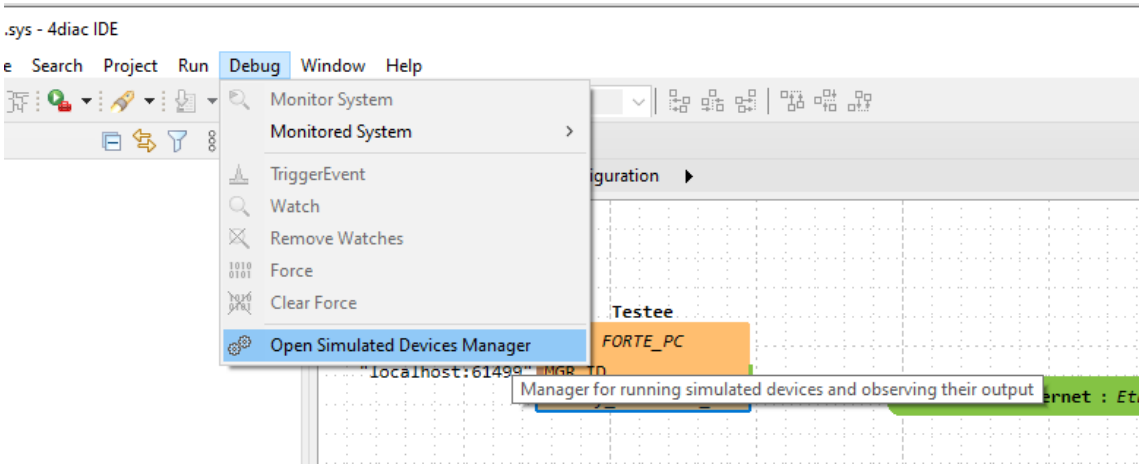


Kuva 18. Onnistunut linkitys [16].

6.4 Tehdyn demon testaaminen paikallisesti

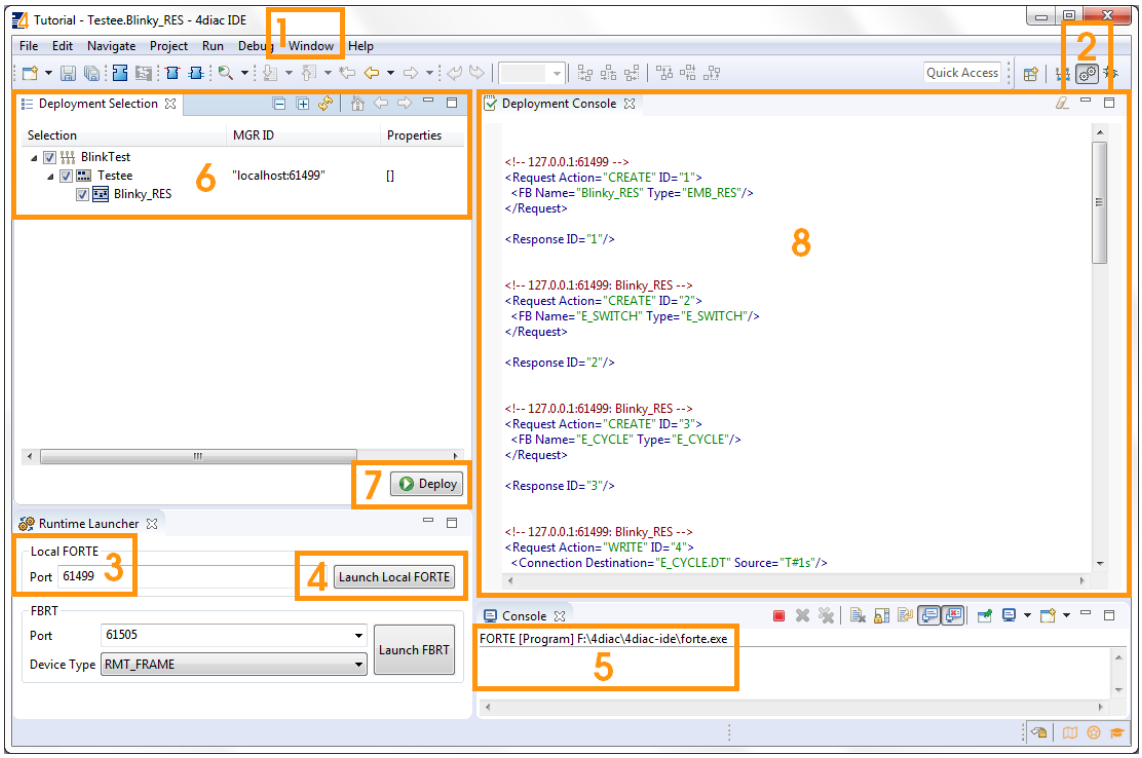
Ohjelman testaamiseksi paikallisesti ilman fyysistä laitteistoa tietokoneella täytyy olla 4diac FORTE, joka on käytännössä virtuaalikone. 4diac-kotisivuilta löytyy valmis paketti tätä varten, mutta se on mahdollista konfiguroida myös itse manuaalisesti. Aikarajoitteiden vuoksi sitä ei tässä opinnäytetyössä kokeiltu, mutta mikäli ohjeet ovat yhtä selkeät kuin tällä vilkkuvalo-ohjelmalla, konfigurointi onnistunee muutaman päivän harjoittelulla.

Verkkosivuilta ladattu .zip -tiedosto puretaan ja sen sijainti kovalevyllä kerrotaan ohjelmalle asetuksissa: *Window* > *Preferences* > *4diac IDE* > *FORTE Precferences*, *Apply and Close*. Virtuaalikoneen saa käyntiin valitsemalla yläpalkista *Debug* > *Open Simulated Devices Manager,* kuva 19. Avautuvasta ikkunasta voidaan klikata *Launch Local FORTE* ja ikkuna jätetään auki.



Kuva 19. Virtuaalikoneen käynnistys [16].

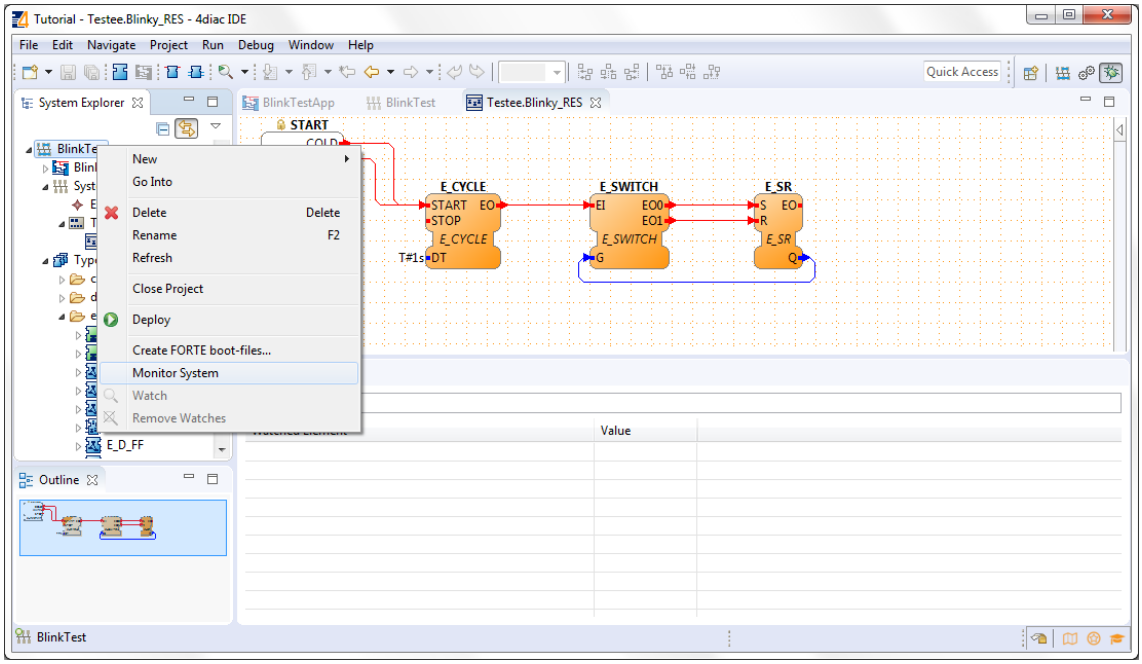
Kun virtuaalikone on käynnissä, voidaan ohjelma ladata virtuaaliresurssiin *Deploy*-painikkeella ylävalikosta. Tarkistetaan, että *Deployment Console* -ikkunaan tulee tekstiä eikä mihinkään ilmesty punaisia huutomerkkejä, kuvassa 20 on esitetty onnistunut lataus.



Kuva 20. Ohjelma ladattuna virtuaalikoneeseen [15].

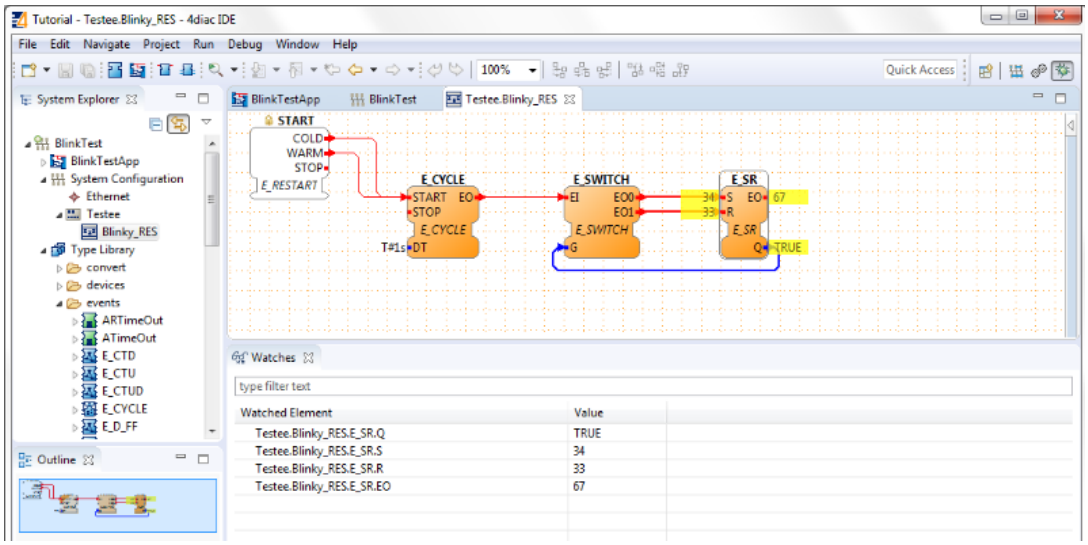
6.5 Sovelluksen toiminnan varmistaminen *debug*-moodissa

Kun tässä vaiheessa kaikki näyttää hyvältä, voidaan toiminta varmistaa *Debug*-moodissa. Moodi käynnistetään pienestä ötökästä ikkunan oikeassa reunassa. Klikkaamalla ohjelmaa hiiren oikealla painikkeella *System Explorer* -ikkunassa voidaan aktivoida systeemin monitorointi, kuva 21.



Kuva 21. Systeemin monitorointi [16].

Monitoroinnin ollessa päällä, täytyy vielä aktivoida *Watch*-moodi klikkaamalla hiiren oikealla painikkeella tyhjässä kohdassa editointi-ikkunaa. Seurantamoodissa voidaan tarkkailla sisääntulojen vaikutusta toimilohkojen toimintaan, kuva 22.



Kuva 22. Toimilohkojen monitorointi *Watch*-moodissa [16].

Tutoriaalin jälkeen käyttäjällä on käsitys, miten ohjelmointiympäristön perustoiminnot löytyvät. Sisäänrakennettujen testiominaisuuksien *(watch, debug)* merkitystä tämäntyyppisissä johdantoprojekteissa täytyy korostaa. Mikäli ruudulla vain siirreltäisiin eri lohkoja eikä niiden toimintaa havainnollistettaisi tai testattaisi millään tavalla, jäisi oppimisprosessi melko tyngäksi. Vaikkei tämä vilkkuvalo-projekti itsessään ole erityisen haastava toteutus, on tärkeää varmistaa ohjelman toimivuus.

7 Yhteenveto

Tässä opinnäytetyössä tutustuttiin uudehkoon IEC 61499 -standardiin ja siihen liittyviin kirjallisuuslähteisiin sekä markkinoilla oleviin standardin mukaisiin ohjelmointiympäristöihin. Uudempaa standardia verrattiin sen edeltäjään, IEC 61331-3 -standardiin, ja pohdittiin uudistetun standardin tarjoamia mahdollisuuksia avoimeen lähdekoodiin pohjautuvien toteutuksien käytössä nykyaikaisessa digitalisoituvassa automaatiossa. Uudistettu standardi helpottaa käyttäjien siirtymistä alustojen ja laitevalmistajien välillä, mutta täysin ongelmatonta eri laitteistojen välillä siirtyminen ei ole vieläkään.

Uudempi IEC 61499 -standardi pyrkii yhdenmukaistamaan eri valmistajien ja palveluntarjoajien ohjelmistoja. Pyrkimys on hyvä, mutta vielä toistaiseksi yhteensopivuusongelmia on selvästi havaittavissa, koska eri valmistajien ohjelmointiympäristöt toimivat eri ohjelmistopohjilla (Java, XML, HTML), kuten luvuissa 4 ja 5 on todettu.

Myöskään eri ohjelmointialustoilla tehdyt ohjelmat eivät välttämättä avaudu toisella alustalla suoraan, vaan ohjelmia joudutaan kopioimaan manuaalisesti tai vaihtoehtoehtoisesti rakentamaan erillinen kääntäjäohjelma, joka siirtää valmiit ohjelmat alustalta toiselle.

Ymmärrettävää on, että valmistajat haluavat varmistaa käyttäjien pysymisen tietyllä alustalla. Käyttäjän kannalta olisi kuitenkin suotavaa, että varsinkin tuotteen elinkaaren loppupuolella uuteen PLC-yksikköön tai ohjelmistoon siirtyminen ei vaatisi pitkällistä ja kallista prosessia. Pahimmassa tapauksessa olemassa oleva laite tai laitteisto joudutaan romuttamaan yksittäisen valmistajan tuotteen saatavuusongelmien vuoksi.

Ohjelmoijien ja insinöörien kannalta olisi myös helpompaa, mikäli ei tarvitsisi opetella käyttämään jokaisen valmistajan omaa ympäristöä, vaan ohjelmoinnin voisi tehdä itselleen tutulla käyttöliittymällä. Päivityksien hankkiminen yhteen ohjelmointiympäristöön on edullisempaa ja tehokkaampaa ajallisesti.

Teollisuudessa uudet standardit ja toimintatavat otetaan käyttöön pitkällä viiveellä. IEC 61499 -standardi on julkaistu jo vuonna 2005, eikä se vieläkään ole teollisuudessa erityisen vahvasti esillä, ainakaan vakiintuneissa tuotantomalleissa. Tämä johtuu teollisuuden tarpeesta nojautua testattuihin ja turvallisiin kokonaisuuksiin, joiden ominaisuudet sekä omituisuudet ovat jo selvillä.

Ilmaiset avoimen lähdekoodin ympäristöt tuovat erityisesti opiskelijoiden ohjelmointiin vapautta, koska kalliit yksittäisen valmistajan alustat ovat hyvin epäkäytännöllisiä hankkia, eikä juuri tietyn alustan hallitsemisesta ole välttämättä merkittävää etua työllistymisen suhteen. Kuitenkin ohjelmointiosaaminen on yleisellä tasolla kohtuullisen helposti siirrettävissä graafisten käyttöliittymien välillä alustalta toiselle.

Tietyllä tavalla standardoinnin mukanaan tuoma yhteneväinen termistö edesauttaa eri ohjelmointiympäristöjen käyttöä ristiin ilman kokonaisvaltaista uudelleenkouluttautumista. Kokenut käyttäjä suoriutuu varmasti annetusta tehtävästä nopeammin ja suoraviivaisemmin kuin jossain toisessa ohjelmointiympäristössä aiemmin toiminut käyttäjä, mutta kohtuullisella vaivalla on mahdollista löytää halutut toiminnot päättelemällä.

Oman standardeihin suhtautumiseni kannalta tämä opinnäytetyöprosessi on ollut hyvin silmiä avaava. Vaikka on olemassa standardi, sen noudattaminen vaikuttaa olevan lähinnä vahva suositus selkeiden sääntöjen sijaan. Mielestäni on kummallista, että täsmälleen saman näköinen ohjelma ei voi toimia täsmälleen saman näköisellä PLC-laitteella, jossa on eri valmistajan tarra. Harmillista on myös se, ettei tämä yhteensopivuusongelma välttämättä selviä ennen kuin on liian myöhäistä. Korvaava komponentti on tilattu vikaantuneen tilalle, ja laitekokonaisuus kieltäytyy kommunikoimasta tämän korvaavan komponentin kanssa.

Työelämässä tähän asti kohtaamani standardit ovat olleet vahvoja sääntöjä, joista poikkeaminen aiheuttaa suuria haasteita tuotannon ylläpitämiseksi ja tasalaatuisten tuotteiden valmistamiseen asiakkaalle. Tämän opinnäytetyöprosessin aikana opin suhtautumaan melko varovaisesti valmistajien tarjoamiin

markkinointilauseisiin. Internetin keskustelupalstoilla olevista ongelmanratkontaketjuistuista löytyi hyvinkin luovia ratkaisuja, joilla työelämässä olevat insinöörit ja ohjelmoijat ovat pyrkineet kiertämään OEM-pohjaisia haasteita avoimen lähdekoodin ohjelmointiympäristöissä. Tulen todennäköisesti jatkossa tutkimaan asiaa itsenäisesti vielä hieman syvällisemmin, koska omassa työssäni on tälläkin hetkellä laitekokonaisuuksia, jotka ovat käyttökelvottomia alkuperäisen laitevalmistajan poistuttua markkinoilta. Mekaanisesti laitteistokokonaisuudet olisivat kuitenkin edelleen tuotantokelpoisia.

Jo pidempään teknisellä alalla työskennelleenä jonkinasteista harmistusta opinnäytetyöprosessin aikana aiheutti käytettävä sanasto. Työkieleen asettuu usein erilaisia lainasanoja, joita käyttää tekstissä huomaamattaan ja olettaa lukijan ymmärtävän, mitä sillä tarkoitetaan. Ajoittain suomenkielisten termien etsiminen työpaikan kahvipöydässä aiheutti hilpeyttä kollegoissa. Toivon hartaasti, että tulevaisuudessa Suomeenkin saadaan oma standardoimisjärjestö, joka kerää ja julkaisee hyväksytyn ammattitermistön insinööreille.

Lähteet

1 IEC 61131-1. 2012. Programmable controllers – part 1: General Information. Second edition 2003-05. International Electrotechnical Commission.

2 Dai, William Wenbin & Vyatkin, Valeriy. A Case Study on Migration from IEC 61131 PLC to IEC 61499 Function Block Control. Verkkoaineisto. University of Auckland. <https://www.vyatkin.org/publ/indin_1%205_wd_vv.pdf>. Luettu 1.2.2024.

3 Johan, Karl-Heinz & Tiegelkamp, Michael. 2010. IEC 61131-3: Programming Industrial Automation Systems. E-kirja. Springer eBooks.

4 Function Block Libraries. Verkkoaineisto. drive.web automation. <https://driveweb.com/fbl-overview/>. Luettu 6.2.2024.

5 Kajola, Paavo. 2023. IEC 61499 based distributed data collection framework for multivariate time series data. Master of Science in Tehchnology Thesis. Aalto university. School of Electrical Engineering. Aaltodoc-tietokanta.

6 Zoitl, Alois. IEC 61499. The new standard in automation. Verkkoaineisto. IEC 61499.com <https://iec61499.com/>. Luettu 27.4.2024.

7 Mustard, Steve. 2022. There are misconceptions about what SoftPLCs can and can't do. Verkkoaineisto. International Society of Automation. <https://www.isa.org/intech-home/2022/february-2022/departments/what-you-may-not-know-about-softplcs>. 2/2022. Luettu 27.4.2024.

8 What is EcoStruxure Platform. Verkkoaineisto. Schneider Electric. <https://www.se.com/in/en/work/campaign/innovation/platform.jsp>. Luettu 27.4.2024.

9 Conway, John. 2020. IEC 61499: The Industrial Automation Standard for Portability that Unleashes Industry 4.0. Verkkoaineisto. Schneider Electric. <https://download.schneider-electric.com/files?p_Doc_Ref=998-21041914&p_enDocType=White+Paper&p_File_Name=998-21041914_GMA_WhitePaper.pdf>. 2020. Luettu 27.4.2024.

10 ISaGRAF Technology, Create Synergy. Verkkoaineisto. Rockwell Automation. <https://www.rockwellautomation.com/en-us/support/documentation/technical-data/isagraf_20190326-0743.html>. Luettu 27.4.2024.

11 Hopsu, Alexander. 2019. Portability of IEC 61499 compliant software. Master's thesis. Aalto university School of Electrical Engineering. Aalto-doc-tietokanta.

12 Recources for the New Generation of Automation and Control Software. 2021. Verkkoaineisto. HOLOBLOC Inc. <https://www.holobloc.com/>. Luettu 27.4.2024.

13 It's time for more intelligence. Verkkoaineisto. nxtControl GmbH. <https://www.nxtcontrol.com/en/>. Luettu 3.5.2024.

14 Open-Source PLC Framework for Industrial Automation & Control. Verkkoaineisto. Eclipse Foundation. <https://eclipse.dev/4diac/index.php>. Luettu 27.4.2024.

15 Step 1 – Use 4diac Locally (Blinking Light). Verkkoaineisto. Eclipse Foundation. <https://eclipse.dev/4diac/en_help.php?helppage=html/4diacIDE/use4diacLocally.html>. Luettu 27.4.2024.

16 Eclipse 4diac IDE. Versio 2.0.1. 2024. Eclipse Foundation.

Toimilohko-ohjelmoinnin perusperiaatteet kaaviona

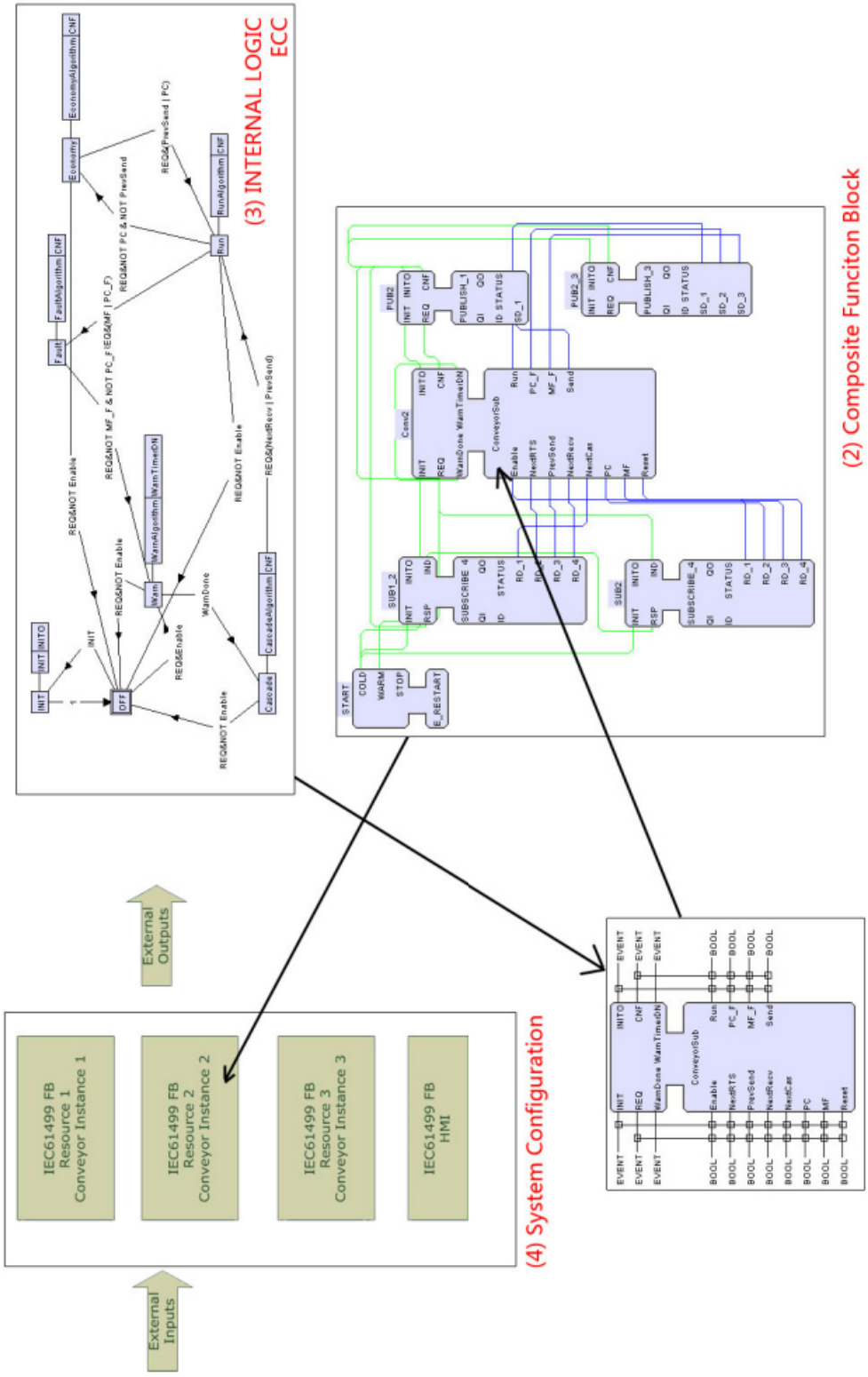


Fig.5. IEC 61499 FB implementation of Control Convevor System.