Simo Huhtanen

TIA Portal Openness – Suunnitteluprosessien automatisointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Konetekniikka

Insinöörityö

27.3.2024

Tiivistelmä

Tekijä:	Simo Huhtanen
Otsikko:	TIA Portal Openness – Suunnitteluprosessien automatisointi
Sivumäärä:	63 sivua + 4 liitettä
Aika:	27.3.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Konetekniikka
Ammatillinen pääaine:	Koneautomaatio
Ohjaajat:	Project Manager, Jaakko Tolonen Lehtori, Maria Sjöholm

---

Logiikkaohjelmoinnin toistuvat manuaaliset työvaiheet altistavat mahdollisille ohjelmointivirheille ja viivästyksille järjestelmän käyttöönottovaiheessa. Manuaalisten työprosessien automatisointi vapauttaa resursseja muuhun työhön. Tämä opinnäytetyö käsittelee TIA Openness -rajapinnan tarjoamia mahdollisuuksia automatisoida automaatiosuunnittelun manuaalisia työprosesseja liittyen TIA Portal -ohjelmointiympäristöön ennalta määritetyn tiedon pohjalta. Työn tilaajana toimi Insta Automation Oy. Rajapinnan ominaisuuksien esittelyssä pääpainona on Step7 -sovellukseen liittyvät toiminnot ja logiikkaohjelmoinnin osat kuten lohkojen vienti ja tuonti XML-tiedostomuodossa.

Tutkimustyön pohjalta kehitettiin .NET pohjainen työkalu C# kielellä. Työskentelyssä sovellettiin ketterän kehityksen periaatteita. Opinnäytetyössä käsitellään myös aiheeseen sekä toteutukseen liittyvät taustatiedot, menetelmät sekä työkalut.

Pääasiallinen tavoite on selvittää rajapinnan tarjoamat mahdollisuudet sekä rajoitteet ja korostaa sen tuomia mahdollisuuksia vähentämällä manuaalista työtaakkaa. Tämän lisäksi analysoidaan Siemensin demosovelluksia, jotka liittyvät aiheeseen. Huomioitavaa on työssä pääasiallisesti käytettyjen TIA Portal -versioiden V16- ja V17- toiminallisuudet saattavat poiketa muista sovellusversioista.

Työkalun jatkokehityksenä on mahdollista lisätä toimintoja liittyen laitteiston, HMI-käyttöliittymien ja moottoriohjainten määrittelyyn sekä laajentaa nykyisiä ominaisuuksia lohkojen generoinnissa laajemmiksi, että modulaarisemmiksi.

Avainsanat: TIA Openness, TIA Portal, XML, Ohjelmointi, PLC

Abstract

Author:	Simo Huhtanen
Title:	TIA Portal Openness – Automation of Design Processes
Number of Pages:	63 pages + 4 appendices
Date:	27 March 2024
Degree:	Bachelor of Engineering
Degree Programme:	Mechanical Engineering
Professional Major:	Machine Automation
Supervisors:	Jaakko Tolonen, Project Manager
	Maria Sjöholm, Senior Lecturer

This thesis explores the possibilities provided by the TIA Openness interface for automating manual processes in TIA Portal programming based on predefined data. Insta Automation Oy commissioned the work, which focused on Step7 application functions and logic programming aspects like block export and import in XML format.

A .NET-based tool was developed using C# following agile development principles. The thesis covers background information, methods, and tools related to the topic and implementation. Based on background research, an application was developed significantly reducing repetitive manual work.

The main goal was to explore the interface's capabilities and limitations, highlighting its potential to reduce manual workload. Additionally, Siemens demo applications related to the topic are analyzed, noting that functionalities in TIA Portal versions v16 and v17 may differ from other versions.

Future enhancements could include features related to hardware, HMI interfaces, and controller definitions, and expand current block generation features to be more comprehensive and modular.

Keywords: TIA Openness, TIA Portal, PLC, programming, XML

Sisällys

Lyhenteet

API: *Application Programming Interface*. Ohjelmointirajapinta, joka mahdollistaa eri ohjelmistojen välisen kommunikoinnin.

Compile: Kääntäminen tarkoittaa prosessia, jossa lähdekoodi muunnetaan konekieleksi suorittamista varten.

CPU: *Central Processing Unit*. Tietokoneen tai muun digitaalisen laitteen pääsuoritin, joka suorittaa ohjelmien käskyjä ja hallinnoi laitteen toimintaa.

DB: *Database*. Tietokanta, jossa säilytetään muuttujia ja niiden arvoja prosessin ohjaukseen sekä seurantaa.

Debug: Prosessi virheiden etsimiseen ja korjaamiseen.

Demo: Esittelyversio tuotteen tai ohjelmiston toiminnallisuuksista.

DLL: *Dynamic Link Library*. Windows-käyttöjärjestelmissä käytettävä tiedostomuoto, joka sisältää ohjelmakoodia ja tietoja, joita useat ohjelmat voivat yhteisesti hyödyntää suoritusaikana.

FB: *Function Block*. Logiikkaohjelmoinnin uudelleen käytettävä ohjelmointilohko muistilla.

FBD: *Function Block Diagram*. Graafinen ohjelmointikieli.

FC: *Function*. Logiikkaohjelmoinnin uudelleen käytettävä ohjelmointilohko ilman muistia.

IEC: *International Electrotechnical Comission*. Kansainvälinen standardointijärjestö, joka kehittää ja julkaisee standardeja sähkötekniikan, elektroniikan ja niihin liittyvien teknologioiden alalla.

IDE: *Integrated Development Enviroment*. Integroitu kehitysympäristö, joka tarjoaa kattavat ohjelmistokehityksen työkalut samassa käyttöliittymässä.

I/O: *Input/Output*. Tulojen ja lähtöjen kokonaisuus, joka liittää anturit ja toimilaitteet osaksi ohjausjärjestelmää.

LAD: *Ladder Diagram*. Graafinen ohjelmointikieli, joka muistuttaa ulkoasultaan relekaaviota.

Lohko: Ohjelmointielementti logiikkaohjelmoinnissa, joka sisältää koodia tai dataa tietylle toiminnolle tai tehtävälle osana järjestelmää.

OB: *Organization Block*. Logiikkaohjelmoinnin organisaatiotason lohko, joka määrittää ohjelman suoritusjärjestyksen.

SCL: *Structured Control Language*. Korkean tason tekstipohjainen ohjelmointikieli, jonka syntaksi muistuttaa C- tai Pascal-kieltä.

STL: *Statement List*. Matalan tason tekstipohjainen ohjelmointikieli, jonka syntaksi muistuttaa assembly-konekieltä.

Turva: Automaation erityisala, joka keskittyy ihmisten, laitteiden ja ympäristön suojeluun prosesseissa sekä järjestelmissä.

UI: *User Interface*, Järjestelmän käyttöliittymä, jonka kautta käyttäjä vuorovaikuttaa ohjelmiston tai laitteen kanssa.

UX: *User Experience*, Kokemus, jonka käyttäjä saa käyttäessään tuotetta tai palvelua.

XML: *eXtensible Markup Language*. Joustava merkintäkieli, jota käytetään tietojen rakenteen kuvaamiseen ja tiedon vaihtoon eri järjestelmien välillä.

1 Johdanto

Tämä työ tutkii Siemensin TIA Openness -rajapinnan ja TIA Portal -ohjelmiston välistä yhteistyötä, keskittyen erityisesti automaatioprosessien tehostamiseen ennalta määritetyn tiedon pohjalta ja demosovellusten analyysin kautta.

Taustatutkimuksen pohjalta kehitettiin sovellus, joka vähentää merkittävästi toistuvaa manuaalista työtä. Sovelluksen kehityksessä sovellettiin ketterän ohjelmistokehityksen menetelmiä. Tutkimuksen ydin on korostaa TIA Openness -rajapinnan teknisiä mahdollisuuksia ja kuinka se voi vähentää manuaalista työtaakkaa. Huomioitavaa on, että työssä käytettiin TIA Portalin versioita V16 ja V17. Ominaisuudet ja toiminnot saattavat muuttua uudemmissa versioissa.

1.1 Toimeksiantajan esittely

Työ on toteutettu Insta Automation Oy:n toimesta, joka kuuluu Insta Group Oy -konserniin. Konserni on tunnettu asiantuntemuksestaan teollisuusautomaation, ohjelmistokonsultoinnin, kyberturvallisuuden ja puolustusteknologian saralla, tarjoten asiakkailleen parannuksia operatiiviseen tehokkuuteen ja turvallisuuteen. [1.]

Insta Groupin suomalainen arvopohja, kansainvälinen suuntautuminen, asiakaslähtöisyys sekä sitoutuminen vastuullisuuteen, osaamiseen ja ihmisiin heijastavat sen keskeisiä vahvuuksia. Tämän tamperelaisen perheyrityksen juuret ja toimintatavat korostavat vahvaa paikallista identiteettiä samalla kun ne tukevat sen globaaleja pyrkimyksiä. [1.]

1.2 Toimeksiantajan tavoitteet

Toimeksiantajan tärkein päämäärä oli kehittää sovelluspohjainen työkalu, joka mahdollistaa autonomisen ohjelmointilohkojen generoinnin TIA Portal -ohjelmointiympäristössä pohjautuen Excel-tiedoston dataan.

Tutkimuksen keskeiset kysymykset keskittyivät TIA Openness -rajapinnan tarjoamiin mahdollisuuksiin ja rajoituksiin, sisältäen lohkojen ja turvalogiikan lohkojen generoinnin toteutettavuuden sekä sovelluksen mahdollisen kehittämisen eri ohjelmointikielillä.

1.3 Henkilökohtaiset tavoitteet

Henkilökohtaisina tavoitteina oli laajentaa ohjelmointitaitoja, syventää ymmärrystä kyseisten ohjelmointilohkojen toimintaperiaatteista ja kehittää Siemensin ohjelmointiympäristössä olevaa osaamista.

- Ensiksi halusin vahvistaa ja monipuolistaa ohjelmointiosaamistani.
- Toiseksi pyrkimyksenä oli saada perusteellisempi käsitys toimeksiantajan ohjauslohkojen toimintalogiikasta.
- Kolmanneksi tavoitteena oli parantaa kykyäni käyttää Siemensin ohjelmointialustaa tehokkaasti, avaten uusia innovatiivisia mahdollisuuksia.

Näiden tavoitteiden saavuttaminen tähtää ammattitaitoni syventämiseen ja valmiuksieni lisäämiseen automaation sekä siihen liittyvän ohjelmoinnin saralla.

2 Työskentelymenetelmät

Tässä kappaleessa on tarkoitus perehtyä tarkemmin työkaluihin ja menetelmiin, jotka olivat käytössä työskentelyssä.

Tutkimustyössä pyrittiin hyödyntämään ketterän kehityksen menetelmiä kuten iteratiivista lähestymistapaa mikä tarjosi mahdollisuuden joustavaan tiedonkeruuseen ja kehitysprosessin mukauttamiseen tarpeiden mukaan.

Tiedonkeruun osalta tavoitteena oli käyttää laadukkaita lähteitä kuten standardeja ja niihin liittyvää kirjallisuutta. Sovelluskehityksessä sovellettiin DSDM-menetelmää**,** joka on myös osa ketterää kehitysfilosofiaa. Projekti jaettiin hallittaviin osiin käyttäen Kanban-taulua tehtävien visualisointiin ja priorisointiin, johon Jira tarjosi kattavat työkalut projektinhallintaan.

Sovelluskehityksen versionhallinta ja dokumentointi liittyen työhön toteutettiin käyttäen GIT- ja Github-työkaluja (Kappale 4.4 Versionhallinta).

2.1 Ketterä kehitys

Ketterä kehitys on ohjelmistokehityksen filosofia, jonka alta löytyy useita eri menetelmiä ja viitekehyksiä kuten Kanban, DSDM (Dynamic Systems Development Method), Scrum ja Lean. [2.]

Yhteistä kaikille eri menetelmille on iteratiivinen lähestymistapa projektia kohtaan ja sen pilkkominen pienempiin kokonaisuuksiin. Perusperiaatteena toimii ajatus siitä, että projektin vaatimukset ja ratkaisut kehittyvät dynaamisesti hyvän viestinnän ja tiheän julkaisutahdin seurauksena. Filosofiassa korostetaan muutoksiin reagoimisen tärkeyttä alkuperäisen suunnitelman noudattamisen sijaan, sekä ohjelmiston toimivuutta yli dokumentaation. Hyötyinä menetelmän käytöstä voidaan pitää:

- Tyytyväisemmät asiakkaat
- Lopputuote vastaa paremmin asiakkaan tarpeita
- Lisääntynyt joustavuus
- Riskien väheneminen ja nopeampi takaisin maksuaika (ROI)

Vastakohdaksi menetelmän käytöstä sen huonoina puolina voidaan pitää:

- Puutteellinen dokumentointi
- Pirstaloitunut lopputuote
- Laajuuden hallinta
- Kustannusten ja aikataulun ennustus

Haasteiden vuoksi jatkokehitys voi olla eri tiimin toimesta hankalaa sekä jatkuvan muutoksen takia lopullinen tavoite voi hämärtyä. Verrattuna kiinteillä aikatauluilla ja resursseilla toteutettuun projektiin on hankala ennustaa projektin kustannusten ja aikataulun edistymistä. [3.]

2.1.1 Kanban

Kanban ei itsessään ole ketterän kehityksen menetelmä vaan työkalu. Sen periaatteita ovat työprosessien visualisointi, keskeneräisen työn määrän hallinta ja työprosessien tehostaminen. Keskeisin periaate on työtehtävien vaiheittainen visualisointi kaaviolle. Tätä kaaviota kutsutaan yleisesti Kanban-tauluksi. Taululla työtehtävät on jaettu vähintään kolmeen eri kategoriaan: tekemättä, työn alla ja tehty. [4.]

Kuva 1. Esimerkki Kanban-taulusta viidellä työvaiheella.

Rajoittamalla työn alla olevien tehtävien määrää pidetään projektin hallinta kontrollissa. Tehtävien keskimääräistä läpimenoaikaa seurataan, jonka perusteella voidaan arvioida prosessin tehokkuutta. Sitoutuminen taulun käyttöön on olennaista, jotta pullonkaulat voidaan tunnistaa ajoissa. [2.]

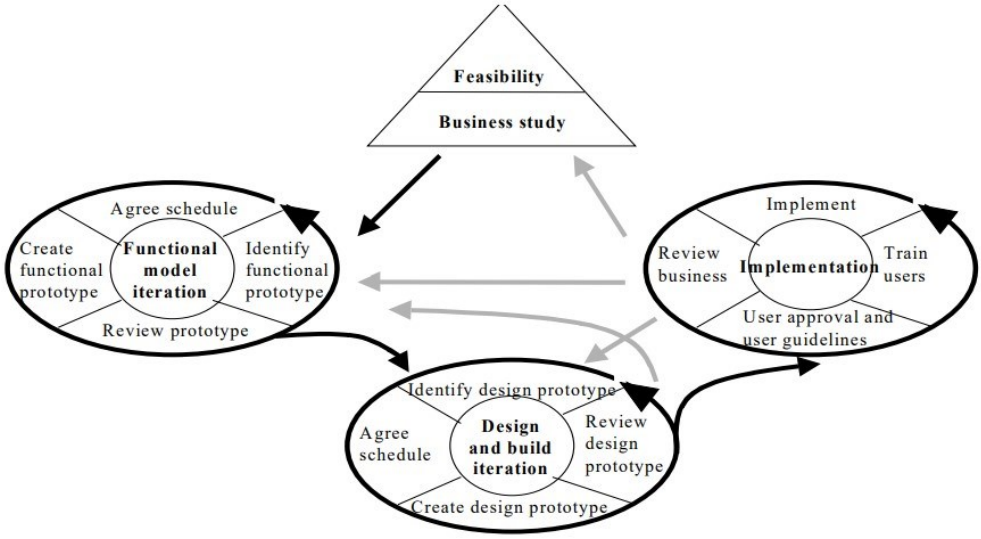
2.1.2 DSDM

Dynaamisen järjestelmän kehitystekniikkafilosofia (DSDM) perustuu muokattuun versioon sosiologian periaatteesta (Pareto-periaate) jonka mukaan 80 % sovelluksesta voidaan toimittaa 20 % ajasta, joka tarvittaisiin koko sovelluksen

toimittamiseen. Menetelmässä työ etenee vaiheittain siten, että jokaisessa vaiheessa keskitytään tekemään vain minimi (80 %) tarvittavasta työstä, joka on tarpeen edetäkseen seuraavaan vaiheeseen. Tämä tarkoittaa, että aluksi on tavoitteena saada aikaan vain toimiva perusversio. Yksityiskohtia sekä lisäominaisuuksia hiotaan ja täydennetään vasta myöhemmin, kun lisää tietoa on saatavilla tai muutoksia tarvitaan. [5;6.]

DSDM-elinkaari (kuva 2) koostuu kolmesta työvaiheesta, jotka toteutetaan sykleissä ja kahdesta edeltävästä tutkimusvaiheesta:

- **Toteutettavuus** (tutkimusvaihe). Arviointi sovelluksen sopivuudelle DSDM-prosessin käyttöä varten.
- **Liiketoiminta** (tutkimusvaihe). Määritellään käyttö- ja tietovaatimukset sekä sovelluksen arkkitehtuuri ja ylläpito.
- **Toiminnallinen** (työvaihe). Useiden prototyyppien luonti, mitkä esittelevät toiminnallisuutta ja keräävät palautetta.
- **Suunnittelu- ja rakennus** (työvaihe). Prototyyppien tarkistus varmistaen niiden kyky tuottaa liiketoiminta-arvoa.
- **Toteutus** (työvaihe). Prototyyppien siirto toimintaympäristöön ja tästä paluu toiminnalliseen vaiheeseen. [6.]



Kuva 2. DSDM-prosessi diagrammina [7].

2.2 Jira

Jira on Atlassianin kehittämä ketterän kehityksen tarpeisiin soveltuva projektinhallintatyökalu. Työkalu tarjoaa monipuolisia toimintoja suunnitteluun, tehtävien jakamiseen, edistymisen seurantaan ja raportointiin. Sovelluksesta löytyy visualisointia varten Scrum- ja Kanban-taulut ja aikajana. [8.]

Sovelluksella pystytään automatisoida monia manuaalisia töitä käyttämällä automaatiosääntöjä. Sääntöjen avulla voidaan konfiguroida sovellusta lähettämään ilmoituksia tietyistä tapahtumista, kuten tehtävän valmistumisesta, tai synkronoimaan tehtäviä ulkoisten työkalujen kanssa. Jiran kanssa voidaan integroida yli kolme tuhatta eri sovellusta, mikä mahdollistaa saumattoman työskentelyn ja tiedonvaihdon eri työkalujen välillä. [8.]

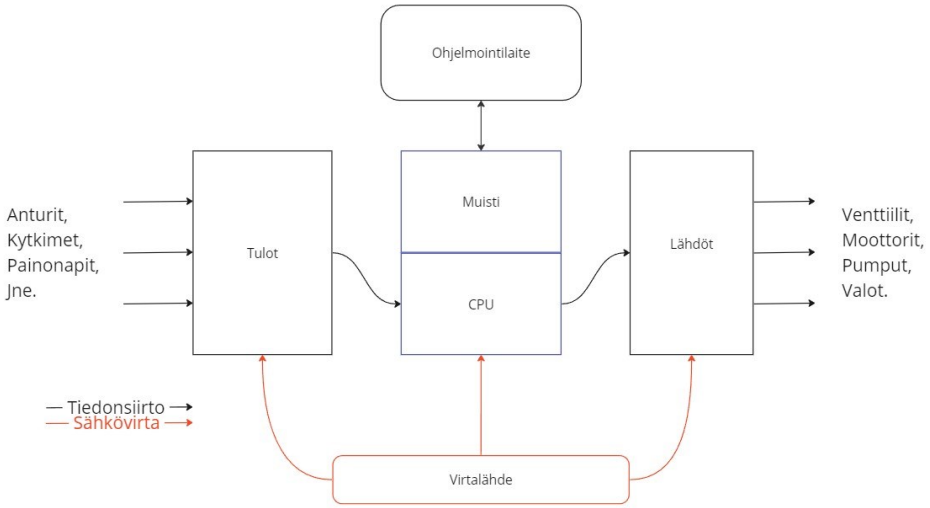
3 Ohjelmoitava logiikka

Tämän luvun tarkoituksena on luoda perusymmärrys ohjelmoitavasta logiikasta ja Siemensin TIA (Totally Integrated Automation) Portal -ohjelmointiympäristöstä.

3.1 Yleistä

PLC eli ohjelmoitava logiikka on tietokone, jota käytetään teollisuuden prosesseissa kuten tuotantolinjojen, robotiikan ja muun automaation ohjaamiseen. Käyttötarkoituksen mukaan sitä voidaan käyttää itsenäisenä järjestelmänä tai se voi olla osa suurempaa automaatiojärjestelmää, kuten SCADA-järjestelmää (Supervisory Control and Data Acquisition) tai DCS-järjestelmää (Distributed Control System). Verrattuna perinteisiin tietokoneisiin ohjelmoitavat logiikat on suunniteltu kestämään vaativia olosuhteita, kuten korkeita lämpötiloja, kosteutta, mekaanista rasitusta ja sähköisiä häiriöitä. [9.]

Laitteiston pääkomponentteja ovat: prosessori (CPU), muisti, tulo-/lähtömoduuli(I/O), virtalähde ja viestintämoduuli (kuva 3). Fyysisesti niiden rakenne vaihtelee integroiduista yksiköistä modulaarisin yksiköihin valmistajan ja käyttöympäristön vaatimusten mukaan. [9.]



Kuva 3. PLC-järjestelmän rakenne [9].

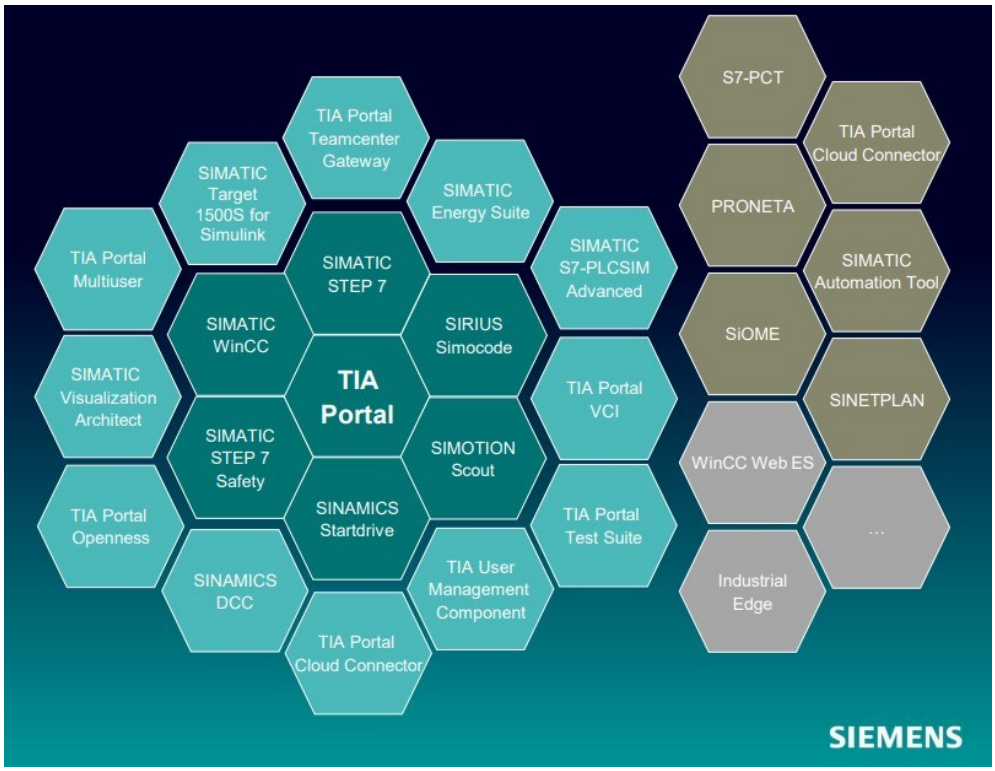
PLC-laitteistoille on kehitetty IEC 61131 -standardi, joka määrittelee järjestelmän toiminnalle välttämättömät toiminnot. Standardi koostuu useista osista, jotka käsittelevät järjestelmän toimintavaatimuksia teollisuusympäristössä. Standardin mukaan laitteistosta tulee löytyä nämä toiminnot [10]:

- **Viestintätoiminnot** mahdollistavat tiedonvaihdon eri järjestelmien välillä. Esimerkiksi toisen PLC-järjestelmän, tietokoneen tai kenttäväylälaitteen kanssa.
- **Virtalähteen** toiminnot varmistavat oikean jännitteensyötön järjestelmälle sekä eristyksen PLC-järjestelmän ja päävirtalähteen välille.
- **Ohjelmoinnin, vianetsinnän, testauksen ja dokumentoinnin toiminnot**. Työkalut ohjelmien lataukseen, testaukseen sekä dokumentointiin.
- **Ihmisen ja koneen välinen käyttöliittymä (HMI) -toiminto.** Operaattorin ja laitteiston välinen kommunikointi järjestelmän kanssa.

- **Anturien ja toimilaitteiden välinen liitäntätoiminto.** viittaa tulo- ja lähtömoduulien signaalien muuntamisen asianmukaisille signaalitasoille. Esimerkiksi analogiset signaalit pitää muuntaa digitaalisen muotoon, jotta PLC pystyy prosessoimaan tietoa.

3.2 Ohjelmisto

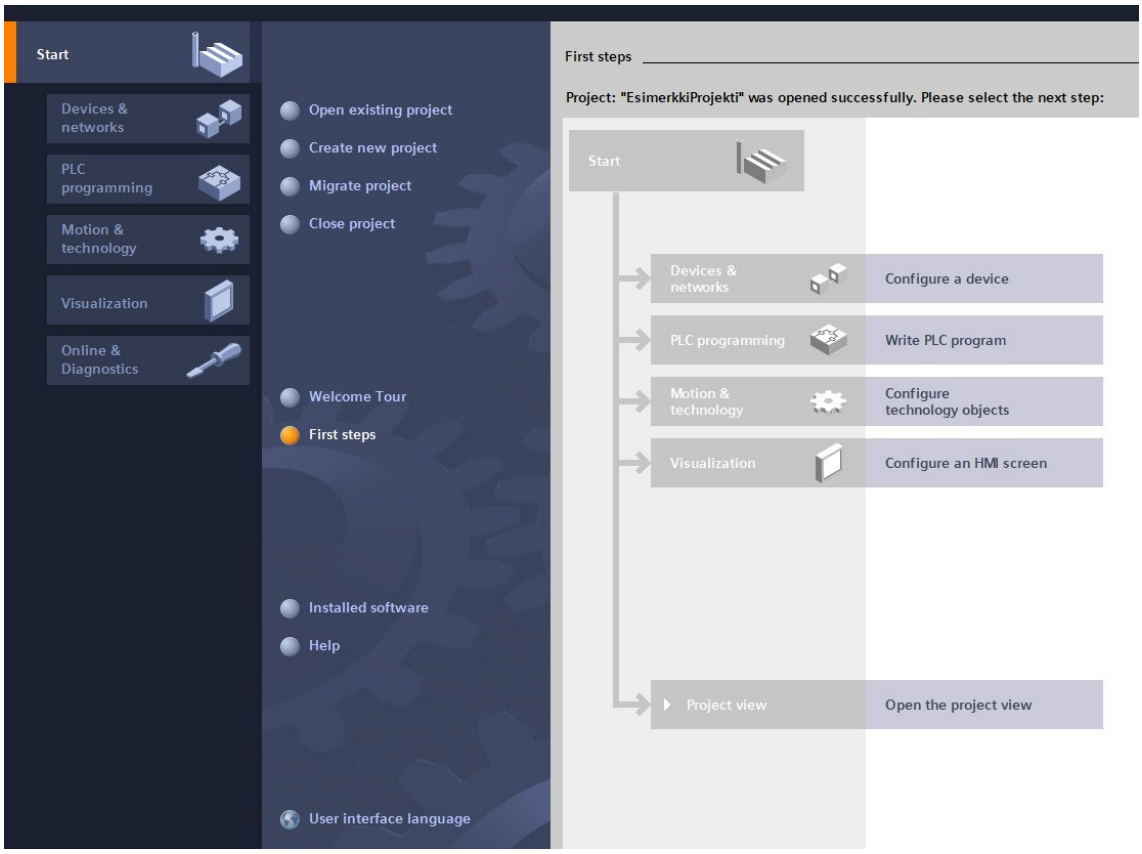
TIA Portal on Siemensin kehittämä ohjelmistoalusta, joka tarjoaa monipuoliset työkalut automaatiojärjestelmän suunnitteluun, simulointiin, ohjelmointiin, testaamiseen ja käyttöönottoon yrityksen valmistamille komponenteille. Simatic Step 7 on osa tätä kokonaisuutta ja se keskittyy logiikkaohjainten ohjelmointiin. Usein automaatiojärjestelmän osana on myös HMI-käyttöliittymä. Sen konfigurointi ja ohjelmointi toimii WinCC-työkalulla saman alustan sisällä saumattomasti, kuten myös Startdrive-työkalu, joka on tarkoitettu moottorienohjainten konfigurointiin. Alusta ja siihen liittyvät sovellukset tarvitset erillisen maksullisen lisenssin. Saatavilla on myös 21-päivän maksuttomia kokeiluversioita. [11.]



Kuva 4. TIA Portal Ohjelmointialusta ja eri sovellukset [12].

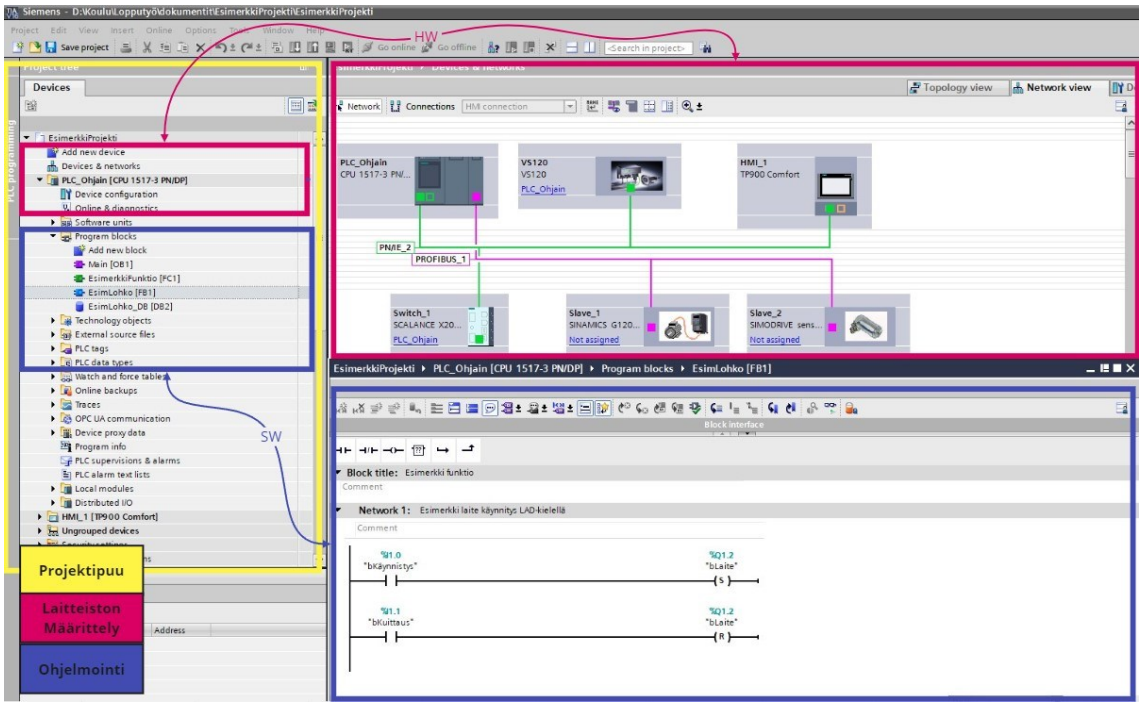
3.2.1 Aloitus

Sovelluksen käynnistettyä avautuu Portal-näkymä, jossa vaihtoehtoina on luoda, avata, siirtää tai sulkea projekti. Tämän jälkeen aukeavat vaihtoehdot siirtyä määrittämään laitteistoa, visualisointia, ja teknologiaobjekteja sekä ohjelmointi- ja vianselvitystyökaluja tai yksinkertaisesti avata projektinäkymä, jossa nämä kaikki edellä mainitut toiminnot ovat saatavilla.



Kuva 5. Portal view -projekti avattuna.

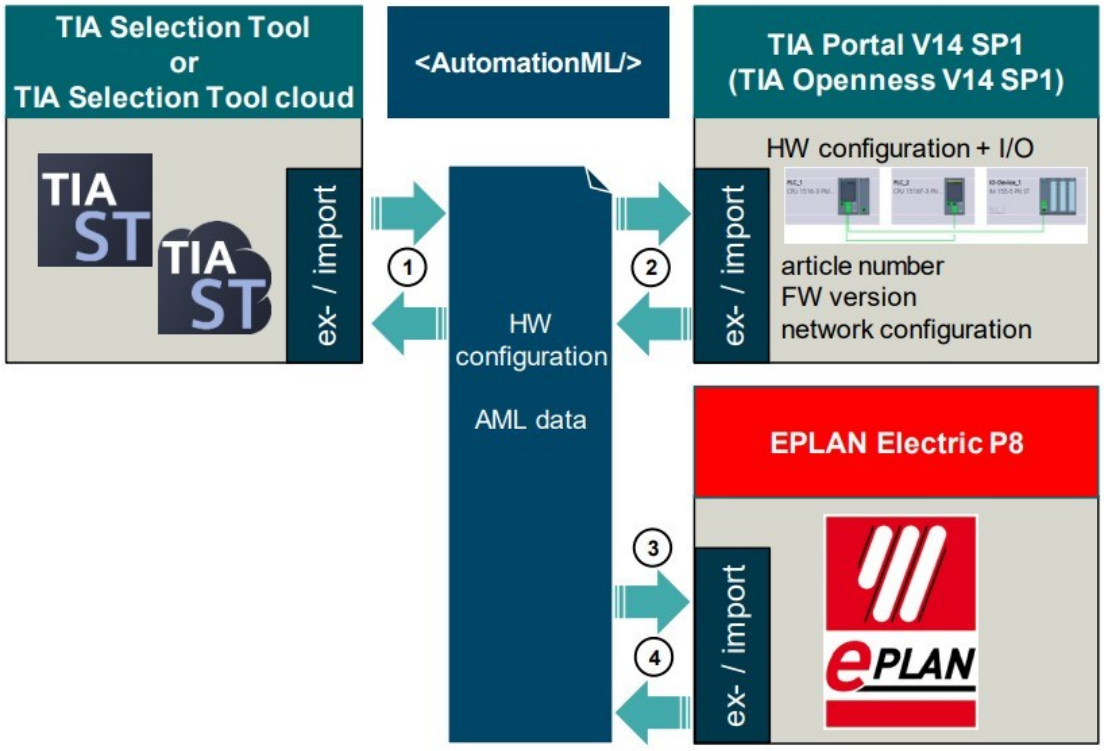
Projektinäkymästä voidaan suorittaa kaikki tarpeelliset toimet kuten laitteiston konfigurointi, ohjelmointi, simulointi, vianselvitys ja itse ohjelman lataus fyysiseen laitteeseen sekä sen toiminnan tarkastelu käyntitilassa. Myös muiden alustaan integroitujen työkalujen kuten WinCC:n käyttäminen tapahtuu saman näkymän kautta. [11.]



Kuva 6. Project view -näkymä, jossa on esillä Laitteiston määrittelyyn ja Ohjelmointiin liittyvät toiminnot.

3.2.2 Laitteiston määrittely

Laitteiston määrittelyllä tarkoitetaan itse fyysisen laitteiston malli- ja versio kohtaista lisäämistä projektiin ja sen asetusten kuten I/O-osoitteiden ja verkkoasetusten konfigurointia. TIA Portalista löytyy jo valmiina reilusti Siemensin valmistamia komponentteja. [11] Tämän lisäksi sovellukseen on mahdollista tuoda muiden valmistajien komponentteja GSD- (General Station Description) tiedosto muodossa. Valmiiksi määriteltyjä laitteita on mahdollista tuoda projektiin myös AML- (Automation Markup Language) tiedoistoina. AML-tiedostoja voidaan tuoda suoraan TST- (TIA Selection Tool) työkalusta tai Eplan CAD-ohjelmistosta. [12.]



Kuva 7. AML-tiedostojen siirto [12].

3.2.3 Ohjelmointi

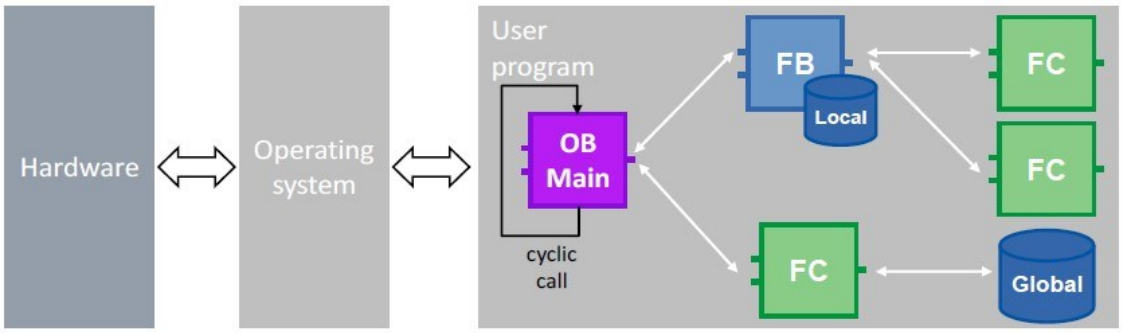
Logiikan ohjelmointi kuten yleisesti muu ohjelmointi on rakenteeltaan jaettu pääohjelmaan ja useampaan aliohjelmaan sekä funktioihin, joita kutsutaan pääohjelmaan suoritettavaksi. Mahdollista on myös toteuttaa koko ohjelma pääohjelmassa, mutta luettavuuden ja vianselvityksen kannalta on viisaampaa pilkkoa ohjelma pienempiin palasiin. Näitä ohjelman eri palasia kutsutaan lohkoiksi (POU). [14;15]

Taulukko 1. Eri lohkotyypit ja kuvaus niiden toiminnoista.

Lohkotyyppi	**Lyhyt kuvaus toiminnoista**
Organisaatio lohkot (OB)	Määrittää ohjelman suoritus järjestyksen

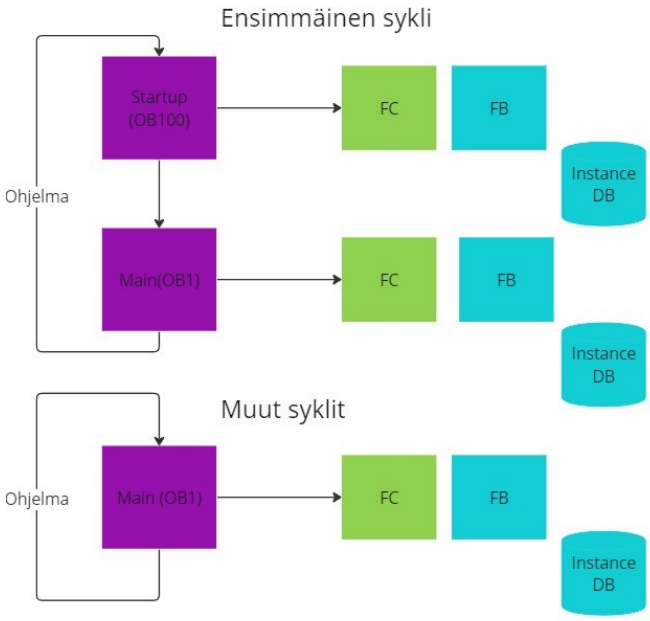
Funktiolohkot (FB)	Modulaarisia ohjelmointilohkoja muistilla
Funktiot (FC)	Modulaarisia ohjelmointilohkoja ilman muistia
Instanssi datalohkot (iDB)	FB:n tiedot tallennetaan instanssidatalohkoihin, lohkojen luonti automaattista, kun ne kutsutaan ohjelmaan.
Datalohkot (DB)	Koko ohjelman laajuinen tiedon tallennus alue mihin kaikilla lohkoilla on pääsy mukaan lukien funktiot.

Alemman tason lohkoja (FB, FC ja DB) käytetään itse ohjelmointiin ja datan tallennukseen, kun ylemmät tason lohkot (OB) taas määrittelivät lähinnä suoritusjärjestystä tai siitä poikkeamista. Organisointitason lohkoja löytyy useita, mutta jokaisesta ohjelmasta löytyy pakollinen lohko OB1. OB1 eli main suoritetaan syklisesti, kun CPU on käyntitilassa. [11;15]



Kuva 8. Ohjelmakierron toimintaperiaate [11].

Muita organisointitason lohkoja ovat tapahtumapohjaiset, aikapohjaiset ja keskeytyspohjaiset lohkot. Esimerkiksi OB100, joka suoritetaan vain kerran ensimmäisen syklin aikana eli käynnistyksessä. [16.]



Kuva 9. Ohjelman suoritus useammalla OB-lohkolla [11].

Funktiolohkoja käytetään suorittamaan tehtäviä, jotka vaativat tilatiedon säilyttämistä suorituskertojen välillä, kun taas funktioita käytetään tilattomiin laskentoihin ja datan käsittelyyn, joissa tilatietoja ei tarvita. Sovelluksesta löytyy valmiina standardin IEC 6113-3 mukaisia ohjelmointilohkoja kuten ajastimia, laskureita, loogisia operaatioita, jotka tehostavat ohjelmointia. Aliohjelmia voidaan sijoittaa sisäkkäin tiettyyn pisteeseen asti, mutta useiden sisäkkäisten ohjelmien käyttö voi huomattavasti hidastaa syklin suoritusnopeutta, kuluttaa paljon muistia ja heikentää ohjelman luettavuutta. Pahimmillaan se voi aiheuttaa ohjelmointivirheen, joka korruptoi ohjelman. Sisäkkäisyyksien maksimimäärä vaihtelee CPU-mallikohtaisesti. [14;15.]

Kun funktiolohko kutsutaan ohjelmaan, luodaan aina instanssidatalohko minne tilatiedot tallentuvat. Näitä on kolmenlaisia [15]:

- Instanssidatalohko yksittäisen lohkon käyttöön.
- Multi-instanssidatalohko useamman samanlaisen lohkon käyttöön
- Parametroitu instanssidatalohko käytetään, kun funktiolohkoja on sisäkkäin

Tämän lisäksi on vielä teknologiaobjekteja (TO) jotka ovat ennalta määriteltyjä monimutkaisempia komponentteja liikkeenhallintaan tai prosessien säätöön. Teknologiaobjekteille löytyy sovelluksesta omat työkalut konfigurointia sekä käyttöönottoa varten. [17.]

3.2.4 Ohjelmointikielet

Tyypillisesti teollisuusautomaatiossa käytetään IEC 61131-3 -standardin mukaisia ohjelmointikieliä, jotka voidaan luokitella graafisiin ja tekstipohjaisiin kieliin [9;19]:

- Instruction List (IL), Tekstipohjainen kieli.
- Structured Text (ST), Tekstipohjainen kieli.
- Ladder Diagram (LD), Graafinen kieli.
- Functional Block Diagram (FBD), Graafinen kieli.
- Sequential Function Chart (SFC), Graafinen kieli

Siemensin TIA Portal -ohjelmointiympäristössä kielet on nimetty hieman poikkeavasti standardista, mutta niiden toiminta on täysin standardin mukaista (Taulukko 2.). [18.]

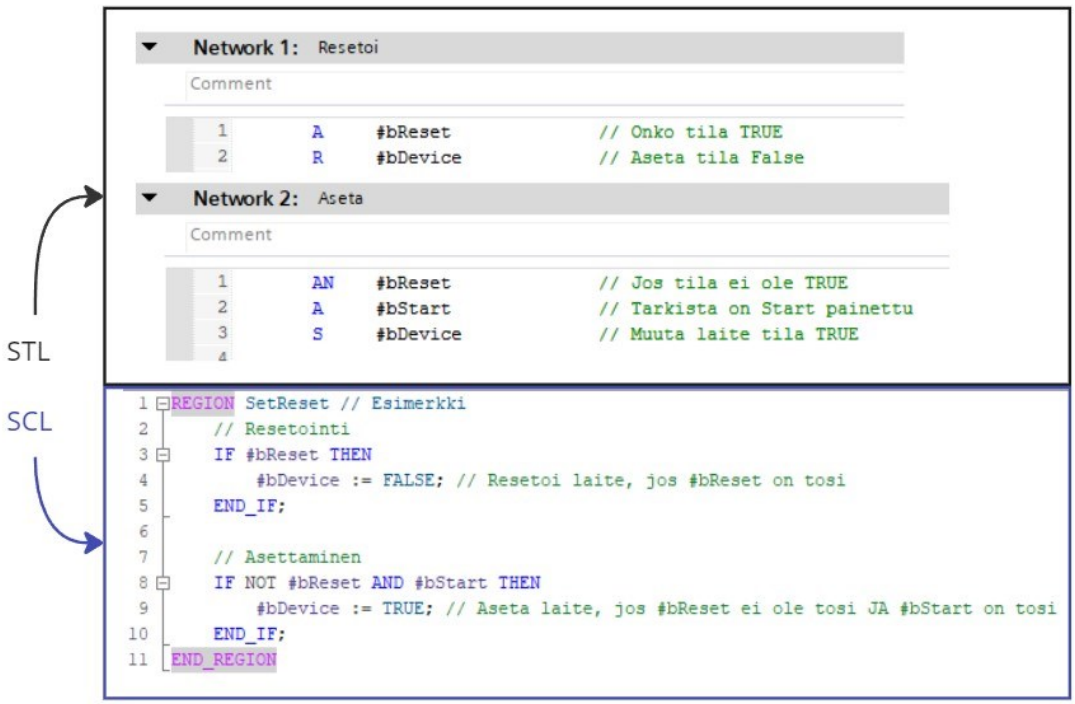
Taulukko 2. Nimeämiserot

IEC-61131-3	**Siemens**
Ladder Diagram (LD)	Ladder Logic (LAD)
Function Block Diagram (FBD)	Function Block Diagram (FBD)
Sequential Function Chart (SFC)	S7-Graph

Structured Text (ST)	Structured Control Language (SCL)
Instruction List (IL)	Statement List (STL)

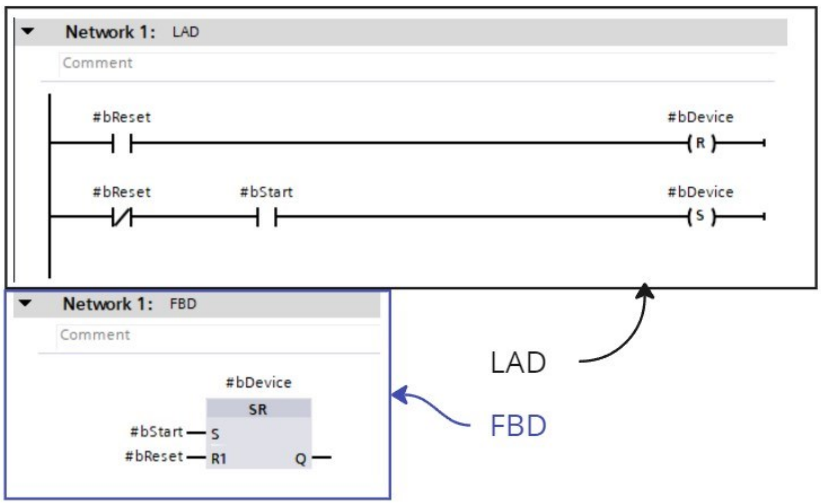
Lisäksi versiosta V17 eteenpäin tietyt CPU-mallit tukevat ohjelmointia graafisilla kielillä CEM (Cause Effect Matrix) ja CFC (Continuous Function Chart). Nämä kielet eivät ole mukana 61131–3-standardissa. [12.]

Tekstipohjaiset kielet, jotka ovat tehokkaita matemaattisten operaatioiden ja laskentojen suorittamisessa, tarjoavat kehittäjille suuren määrän joustavuutta ja tarkkuutta. [15] Seuraavaksi on esiteltynä yksinkertainen esimerkkiohjelma (kuva 10) Boolean tyyppisen lähdön tilan muuttamisesta SCL- ja STL-kielillä.



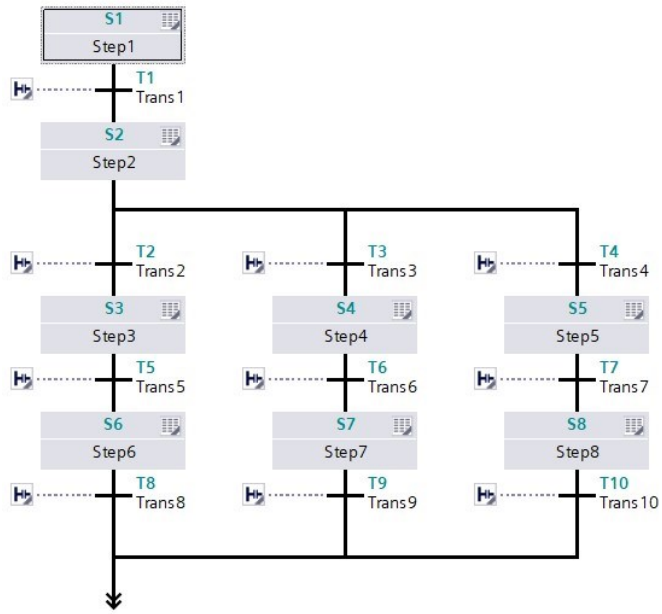
Kuva 10. Esimerkkiohjelma SCL- ja STL- kielillä.

Graafiset kielet tarjoavat intuitiivisen ja helposti ymmärrettävän lähestymistavan ohjelmointiin, mikä tekee vianselvityksestä suoraviivaisempaa. [15] Seuraavaksi (kuva 11) on aikaisemmin esitelty esimerkkiohjelma (kuva 10) toteutettuna LAD- ja FBD-kielillä.



Kuva 11. Esimerkkiohjelma LAD- ja FBD-kielillä.

SFC (Sequential Function Chart) ei varsinaisesti ole ohjelmointikieli vaan työkalu, joka keskittyy ohjelman loogisten toimintojen suoritusjärjestykseen ja visualisointiin (kuva 12) kuten vaikka tilakoneen. [15.]



Kuva 12. Esimerkkikoodi SFC

PLC-ohjelmoinnissa on tapana hyödyntää usein sekä graafisten että tekstipohjaisten kielten vahvuuksia. Esimerkiksi aliohjelmat tai funktiot voidaan toteuttaa tekstipohjaisella kielellä niiden tehokkuuden vuoksi, ja nämä komponentit voidaan sitten integroida pääohjelmaan, joka on luotu käyttäen graafista kieltä. Tällainen yhdistelmä mahdollistaa ohjelmien selkeyden ja hallittavuuden sekä helpottaa mahdollista vianselvitystä. [20;15.]

3.2.5 Datatyypit ja muisti

Sovelluksesta löytyy perinteiset tietotyypit tiedon käsittelyyn kuten: BOOL (totuusarvo), INT (kokonaisluku), REAL (liukuluku), sekä monia muita (liite 1). Tämän lisäksi on olemassa johdettuja tietotyyppejä kuten esimerkiksi [11]:

- **TIME**: Aikaa edustava tietotyyppi, jota käytetään aikaviiveiden, ajastimien määrittämiseen ohjelmassa.
- **DT** (Date_AND_TIME): Päivämäärän ja ajan yhdistävä tietotyyppi tarkoitettu prosessien aikaleimaamiseen.
- **STRUCT**: Tietorakenne, johon voidaan yhdistää eri tietotyyppejä yhdeksi loogiseksi kokonaisuudeksi. Esimerkiksi moottorin tilaan

liittyviä tyyppejä, kuten tila (BOOL), nopeus (REAL) jne, voidaan laittaa STRUCT-Moottori sisälle.

- **ARRAY**: Tietorakenne, joka koostuu samantyyppisistä tietotyypeistä eli taulukko.
- **UDT** (User Defined Type): Modulaarinen versio STRUCT-tietorakenteesta. Mahdollistaa samankaltaisten tietorakenteiden tehokkaan luonnin mallipohjan perusteella.

Tietotyypit sijaitsevat sovelluksessa tag-taulukoissa. Tagit ovat symbolisia nimiä, jotka on liitetty tietyntyyppisiin tietoihin. Tagit voivat viitata suoraan muistialueisiin, I/O-signaaleihin tai tietorakenteisiin. Tuloilla, lähdöillä ja muistilla on omat muistialueensa I, Q, M. Tag-taulut helpottavat tiedon hallintaa ja koodin luettavuutta. Muuttujien eli tagien toimintaa voidaan myös seurata ja pakottaa Watch- ja Force-taulukoissa. Tämä ominaisuus on hyödyllinen testaus- ja käyttöönottovaiheessa. [11.]

3.2.6 Kirjastot

Projektin palasia voidaan tallentaa kirjastoihin. Kirjastot ovat kokoelma uudelleen käytettäviä komponentteja ja ne jaetaan kahteen päätyyppiin:

- Projektikirjastot ovat projektin sisäisiä kirjastoja, jotka ovat saatavilla vain kyseisen projektin sisällä.
- Globaalit kirjastot ovat saatavilla kaikissa projekteissa.

Kirjastoissa voidaan käyttää mastercopy-versioita, jotka toimivat pohjana, kun uusia versioita luodaan komponenteista projektiin. Kirjastoihin voidaan tallentaa useita eri elementtejä, kuten:

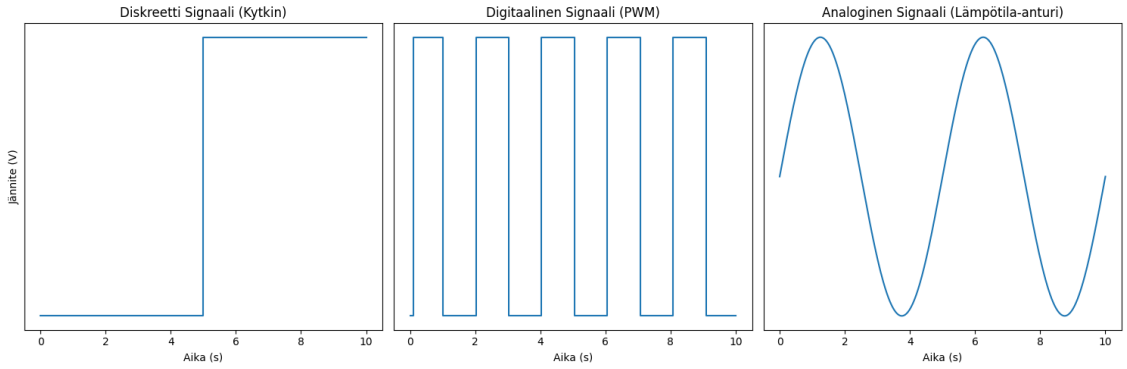
- Ohjelmointilohkoja (FB, FC)
- Tag-taulukoita (Tag tables)
- Laitteistomäärittelyjä (HW)
- Data- ja tyyppikirjastoja (UDT)
- Visualisointielementit (HMI)

Näiden käyttäminen osana suunnittelu- ja toteutusprosessia yhdenmukaistaa ja tehostaa työskentelyä. [11.]

3.3 Tiedonsiirto

Tiedonsiirto järjestelmän ja toimilaitteiden sekä anturien välillä tapahtuu signaaleilla. Signaalit ovat digitaalisia tai analogisia, mutta ne voidaan luokitella kolmeen eri luokkaan (kuva 13): [9]

- **Diskreetit signaalit** ovat binäärisiä eli ne voivat esiintyä vain yhdessä kahdesta tilasta: päällä tai pois päältä (0 tai 1). Käytännössä diskreetti signaali laitteesta, kuten kytkimestä, ilmaisee jonkin tilan läsnäolon tai puuttumisen.
- **Digitaaliset signaalit** ovat myös binäärisiä ja luonteeltaan diskreettejä mutta ne viittaavat usein sekvenssiin tai joukkoon diskreettejä signaaleita, jotka välittävät tietoa. Esimerkiksi tasavirtamoottorin PWM-ohjauspulssit ovat digitaalisia signaaleja, jotka säätävät tehoa pulssin pituuden muutoksilla.
- **Analogiset signaalit** edustavat jatkuvaa tietoa, kuten lämpötilaa tai painetta, ja voivat saada äärettömän monta eri arvoa tietyllä välillä. Esimerkiksi lämpötila-anturi tuottaa analogisen signaalin, joka muuttuu lämpötilan mukaan, mahdollistaen tarkan mittauksen.

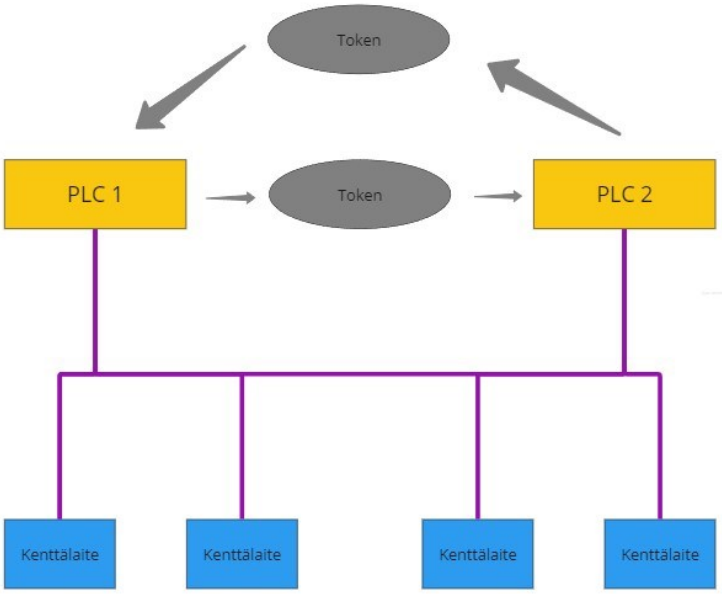


Kuva 13. Eri signaalityypit ajan funktiona.

3.4 Tietoliikenneverkot

Perinteisesti automaatiojärjestelmissä tiedonsiirto PLC:n ja kenttälaitteiden välillä on suoritettu käyttämällä analogista 4-20mA virtaviestiä tai 0-5V DC ja 0-10V DC jänniteviestiä [21]. Digitaalisen muodon viestintä laitteiden kanssa on yhä yleistymässä, ja tähän tarkoitukseen käytetään erilaisia kenttäväyläprotokollia. Sopivan kenttäväyläprotokollan valinta riippuu laitteiden määrästä, vaaditusta tiedonsiirtonopeudesta, komponenttien yhteensopivuudesta, etäisyyksistä ja käyttöympäristöstä. Kenttäväyliä teollisessa ympäristössä käsitellään tarkemmin IEC:n standardikokoelmassa 61158 [22]. Sen soveltamista varten on vielä kehitetty standardi IEC 61784, joka luokittelee kenttäväyläprotokollat niiden käyttöympäristön mukaan [23].

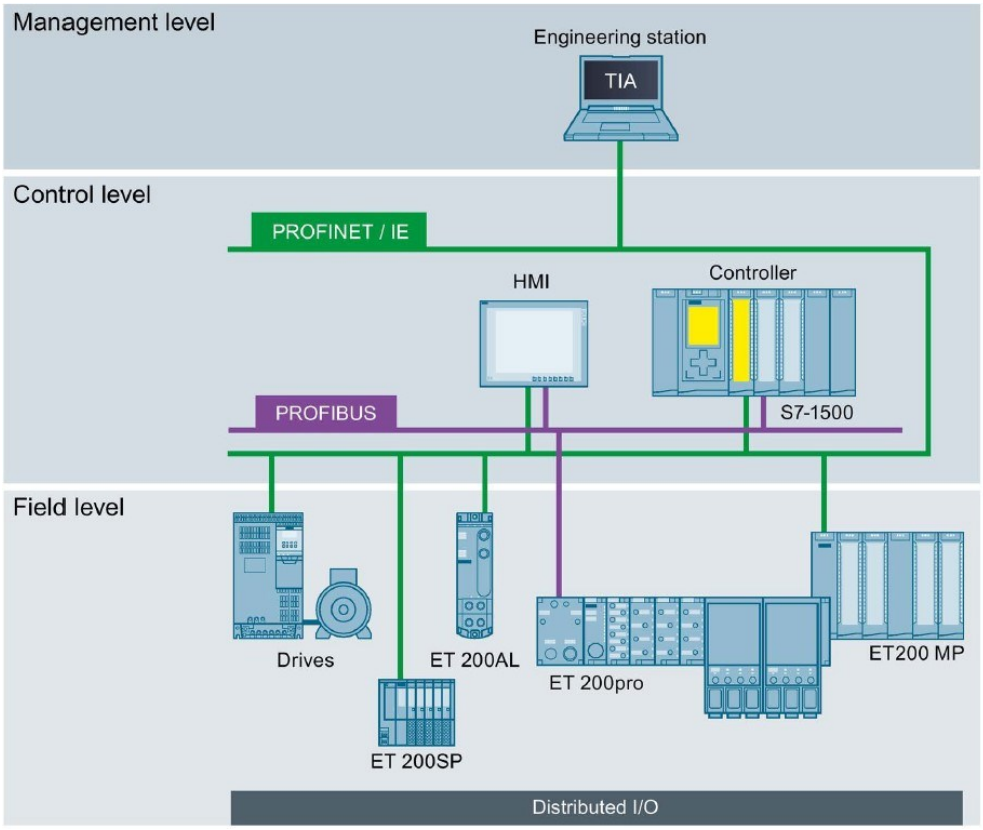
Kenttäväylässä laitteet jaetaan yksinkertaisesti master-(isäntä) ja slave-(orja) laitteisiin. Isäntä eli PLC ohjaa tiedonsiirtoa lähettämällä käskyjä ja vastaanottaen tietoja. Orjana toimiva kenttälaite vastaanottaa pyynnöt ja lähettää tietoa takaisin pyydettäessä. Jokaisella kenttälaiteella on uniikki väyläosoite tunnistamista varten. Tietyissä väyläprotokollissa, kuten Profibus DP:ssä, voidaan myös käyttää useita isäntälaitteita, joissa hallinnan vaihto suoritetaan tokenilla. Järjestelmän kytkentätapa eli topologia määräytyy käyttökohteen mukaan, joihin vaikuttavat tekijät kuten kustannukset, tiedonsiirron suunta ja nopeus, käyttövarmuus sekä laitteiden määrä. [24;25.]



Kuva 14. Esimerkki Profibus DP -väylästä ja tiedonsiirrosta [26].

Verrattuna perinteiseen järjestelmään, jossa jokainen laite tarvitsee oman johtimen, väylässä useampi laite voi hyödyntää samaa johdinta. Väylässä on myös mahdollista lähettää useampia parametrejä samanaikaisesti. Kenttäväylällä toteutettu järjestelmä vaatii huomattavasti vähemmän kaapelointia, mikä mahdollisesti johtaa kaapeloinnin osalta madaltuneisiin kustannuksiin.

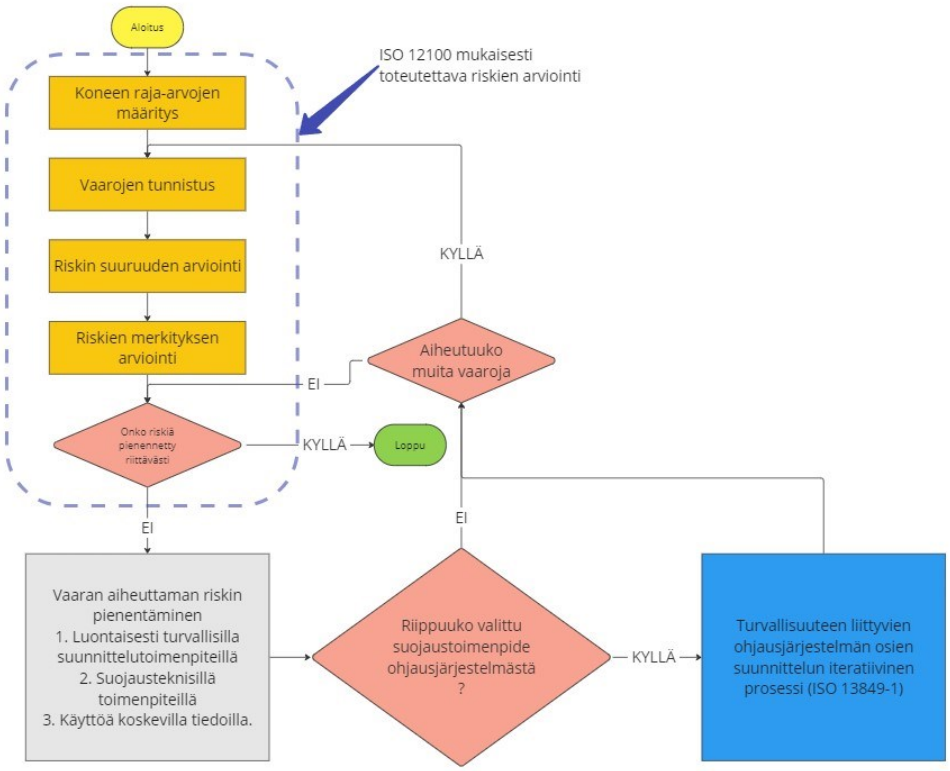
Kenttäväylä ei käsitä yksistään automaatiojärjestelmän tietoverkkoa vaan se on osa isompaa kokonaisuutta, joka voidaan jakaa kolmeen pääkategoriaan: kenttäväylät, langattomat ja teollinen internet (IIoT), jossa jokaisella on hyvin erilaiset ominaisuudet ja käyttötarkoituksensa [27]. Tämän takia isompien automaatiojärjestelmien tietoliikenneverkko kokonaisuutena muodostuu näiden yhdistelmistä. [24;25.]



Kuva 15. Profinet ja Profibus samassa automaatiojärjestelmässä [28].

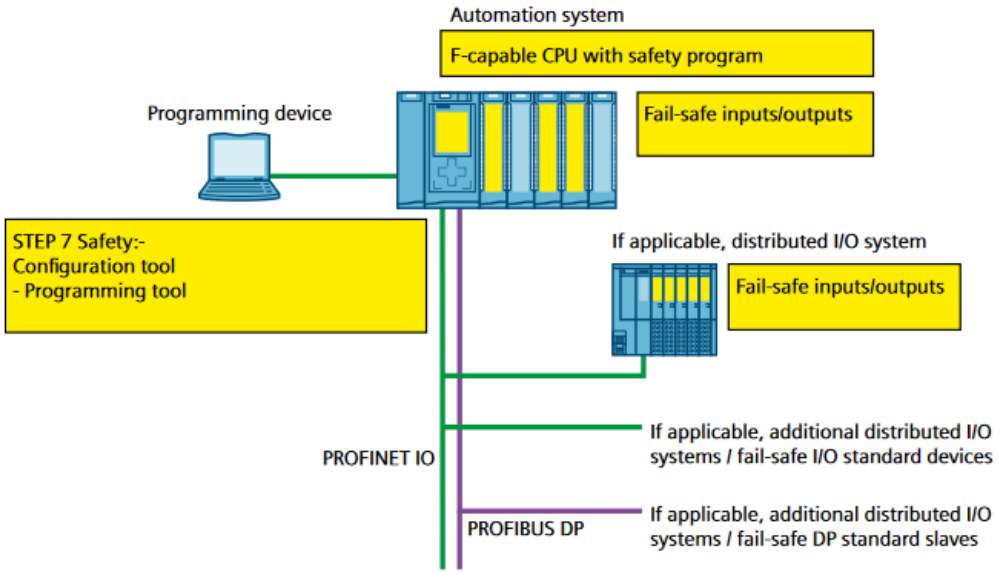
3.5 Toiminnallinen turvallisuus

Ohjausjärjestelmien käyttöympäristö asettaa turvalliseen toimintaan liittyviä haasteita, joita kartoitetaan koneturvallisuus standardien riskienarvioinnin perusteella. Riskienarvioinnin tulokset asettavat vaaditun suoritustason niin komponenteille kuin järjestelmälle, miten niiden tulee kestää ja reagoida vikaantumistilanteissa ja miten turvallinen toiminta on varmistettu. [29.]



Kuva 16. Yleinen riskienarviointi [29].

Turvallisuuteen liittyvät toiminnot toteutetaan PLC-ohjelmassa erillisinä organisaatiotason lohkoina, joille on tyypillistä rajallisemmat ominaisuudet ohjelmointikielten, funktioiden ja datatyyppien suhteen. Väyläliikenne näiden toimintojen osalta kulkee erillisessä verkossa verrattuna normaalisti suoritettavaan ohjelmaan. Turvallisuustoimintojen suorittamiseen voi olla käytössä turvaluokan PLC, joka kykenee suorittamaan sekä normaalia ohjelmaa että turvatoimintoja, ja eri ohjelmilla on omat I/O-korttinsa normaali- ja turvaluokkaa varten. Vaihtoehtoisena toteutuksena turvatoimintoja varten järjestelmässä voi olla oma PLC, joka täyttää vaaditut luokitukset. [10;29;30.]



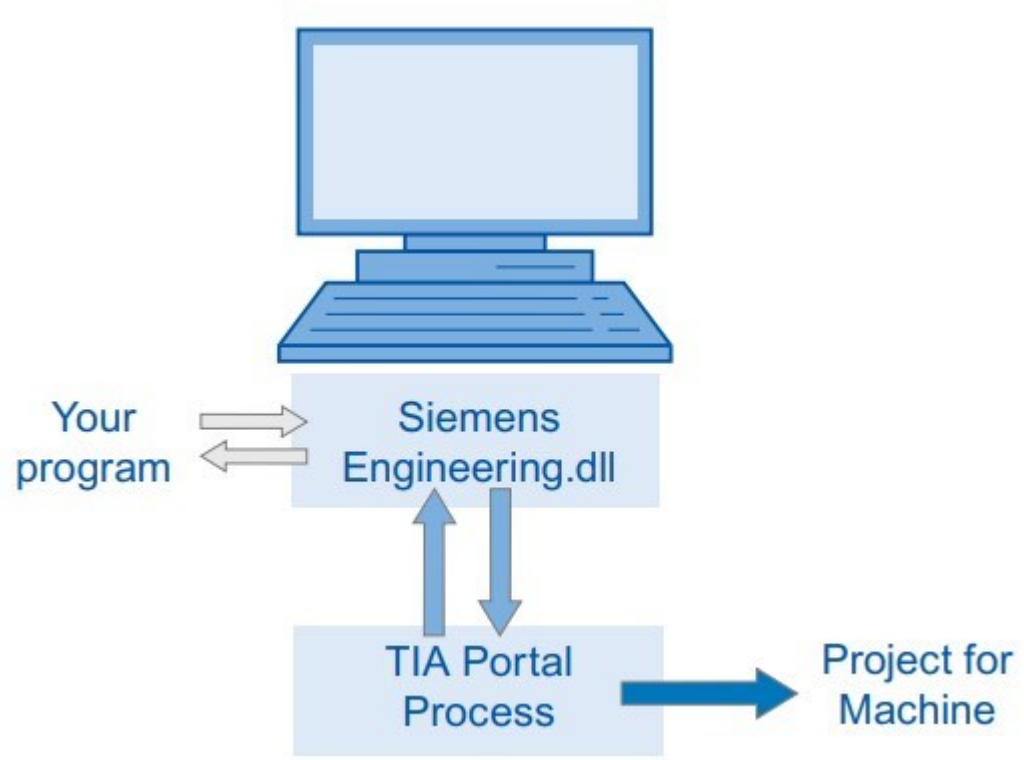
Kuva 17. Yleiskuva turvaluokan laitteistosta ja siihen liittyvästä ohjelmistosta [30].

4 Ohjelmointi

Tässä osiossa keskitytään työkaluihin ja teknologioihin, jotka ovat keskeisiä lohkojen generoinnin mahdollistavan työkalun kehittämisessä.

4.1 TIA Openness

TIA Openness on Siemensin sovellusrajapinta (API) joka mahdollistaa sovelluksien kommunikoinnin TIA Portal -ohjelmointiympäristön kanssa. Rajapinta tarjoaa mahdollisuuden tiedon vientiin ja tuontiin, kuin myös TIA Portal -prosessien etäohjaukseen. [31.]



Kuva 18. Toimintaperiaate [31].

Rajapinta koostuu kahdesta .dll-tiedostosta, jotka on kehitetty .NET-ohjelmisto-kehyksellä. Tiedostoja voidaan suoraan hyödyntää ohjelmointikielillä, jotka ovat yhteensopivia .NET-ympäristön kanssa, kuten C# ja VB.NET. [32.]

Muita ohjelmointikieliä kuten Python voi hyödyntää käyttämällä kolmannen osa-puolen kirjastoa, kuten IronPython, mikä mahdollistaa .NET-yhteensopivuuden.

TIA Portal -sovellukset, jotka tukevat rajapinnan käyttöä:

- Step7, Step7 Safety
- WinCC Unified, WinCC Professional (Basic/Comfort/Mobile ja RT Advanced)
- SIMATIC Visualization Architect (SiVArc), Startdrive
- SIMOTION SCOUT TIA, SINUMERIK, VCI

Step 7 -sovellukseen liittyen on mahdollista tuoda/viedä esimerkiksi: ohjelmointi- ja datalohkoja, tag-taulukoita, teknologiaobjekteja ja laitteistomäärittelyjä. Myös turvalohkoja (Step 7 Safety) on mahdollista siirtää rajapinnan kautta.

Taulukko 3. Objektien tuonti- ja vienti mahdollisuudet.

Objekti	**Vienti**	**Tuonti**
Lohko	x	x
Turvalohko	x	x
Tag-taulut / Tagit	x	x
Teknologia objektit	x	x
UDT (User Data Type)	x	x
Datalohkot	x	-

Vaikka datalohkojen tuominen ei ole mahdollista, uusien luominen on silti mahdollista, esimerkiksi lohkokutsuja luotaessa. Lohkokutsujen luonti ei automaattisesti luo tarvittavia datalohkoja, minkä vuoksi uusien datalohkojen luominen on tarpeellista kyseisessä yhteydessä. Taulukossa esiteltyjen objektien lisäksi on mahdollista viedä ja tuoda paljon muita objekteja kuten kokonaisia laitteistomäärittelyjä ja HMI-käyttöliittymiä. Näitä ominaisuuksia ei esitellä tässä työssä tarkemmin, mutta niissäkin löytyy omat rajoitteensa ja mahdollisuudet. Tämän lisäksi on mahdollista oman ulkoisen sovelluksen rakentamisen sijaan vaihtoehtona rakentaa laajennus TIA Portal -alustan sisälle. [33;31.]

Rajapinta on ollut mukana TIA Portal V13 -versiosta ja sitä on päivitetty jokaisen julkaisun yhteydessä lisäten uusia ominaisuuksia [31]. Tästä huolimatta rajapinta ei ole saavuttanut suurempaa suosiota tai tunnettavuutta. Automaatiosuunnittelijan näkökulmasta suurimpana syynä voidaan pitää jyrkkää oppimiskäyrää, joka liittyy rajapinnan käytön, uuden ohjelmointikielen ja XML-tiedostorakenteen opetteluun. Objektien siirto rajapinnan kautta tapahtuu XML- (Extensible Markup Language) tiedostomuodossa. Nämä tiedostot kasvavat hyvin

nopeasti yli tuhat rivisiksi ja hankaliksi lukea ja muokata ilman aikaisempaa kokemusta tiedostomuodosta. [34.]

Rajapinnan käyttämisen ydin on automatisoida manuaalisia työprosesseja kuten laitteistomäärittelyä, ohjelmalohkojen luontia ja HMI-käyttöliittymän rakentamista. Poistamalla manuaalisia työvaiheita vähennetään myös virheiden riskiä, joka nopeuttaa FAT- (Factory Acceptance Test) - sekä SAT- (Site Acceptance Test) työvaiheita.

Esimerkkinä yllä mainitusta prosessista voidaan kuvitella tilannetta, jossa meillä on useita kymmeniä tai vaikka satoja samankaltaisia laitteita, joiden aikaisemmasta toteutuksesta on olemassa testattu ja turvallisesti toimiva ratkaisu. Rajapinnan avulla pystyttäisiin lähes nappia painamalla luomaan koko projekti laitteistomäärittelystä ohjelmointiin juuri halutulla laitemäärällä.

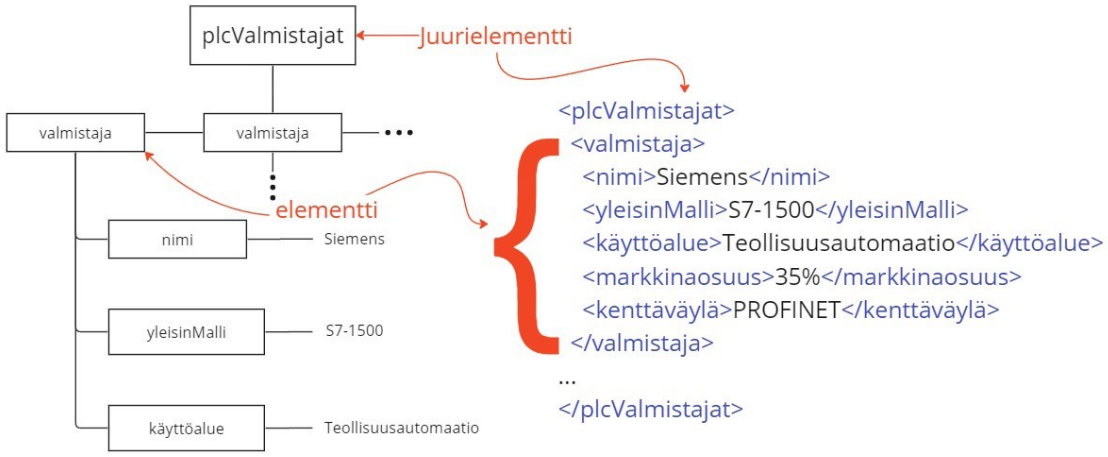
4.2 XML- ja XML-skeemat

XML (Extensible Markup Language) on merkintäkieli, joka on kehitetty joustavaksi välineeksi tiedon tallennukseen, jakamiseen ja siirtoon eri sovellusten kesken. XML-dokumenttien rakenne määritellään XML Schema Definition (XSD) -määrityksellä. Tämän kielen syntaksi on luotu ihmisen ja koneen ymmärrettäväksi, eikä se sisällä valmiiksi määriteltyjä tageja, vaan ne luodaan käyttäjän toimesta tai määritellään XML-skeemalla. [35;36.]

4.2.1 Tiedostorakenne

Kielen hierarkia esitetään puumuotoisena tietorakenteena (kuva 19). Dokumentin rakenne alkaa juurielementistä (root) ja haarautumalla lapsielementteihin (child), ja jokainen elementti voi sisältää alielementtejä (sub-child). Dokumentit koostuvat elementeistä, jotka kirjataan kulmasulkujen väliin. Elementit

muodostuvat aina alku- ja loppuosasta, joiden välille elementin tekstisisältö sijoitetaan. [37.]



Kuva 19. Esitettynä on yksinkertaisen XML-rakennepuun ja vastaavan XML-dokumentin rakennetta PLC-valmistajia käsittelevän tiedon avulla [37].

Tietoa voidaan myös lapsielementtien sijaan sijoittaa attribuutteihin, mutta sen lukeminen on huomattavasti selkeämpää, kun käytetään lapsielementtejä.

```
<valmistaja nimi="Siemens" yleisinMalli="S7-1500" käyttöalue="Teolli-
suusautomaatio" markkinaosuus="35%" kenttäväylä="Profinet" />
```

Esimerkkikoodi 1. Attribuuteilla muotoiltu tieto.

```
<valmistaja>
  <nimi>Siemens</nimi>
  <yleisinMalli>S7-1500</yleisinMalli>
  <käyttöalue>Teollisuusautomaatio</käyttöalue>
  <markkinaosuus>35%</markkinaosuus>
  <kenttäväylä>Profinet</kenttäväylä>
</valmistaja>
```

Esimerkkikoodi 2. Lapsielementeillä muotoiltu.

Suurin ero elementtien ja attribuuttien käytössä on, ettei attribuutti voi sisältää useita arvoja kuten elementit. [36.]

```
<valmistaja nimi="Siemens" malli="S7-1500" />
```

Esimerkkikoodi 3. Elementillä valmistaja on attribuutteja nimi ja malli, joista kumpikin voi sisältää vain yhden arvon.

Attribuutteja on suositeltavaa käyttää, jos käytössä on useita samankaltaisia elementtejä kuten vaikka PLC-valmistajia. Elementtien yksilöllinen numerointi tarjoaa tehokkaan tavan prosessoida tietoa käsittelyvaiheessa. [36.]

```
<valmistajat>
  <valmistaja id="001">
    <nimi>Siemens</nimi>
    <mallit>
      <malli>S7-1500</malli>
      <malli>S7-1200</malli>
      <malli>ET 200SP</malli>
    </mallit>
  </valmistaja>
  <valmistaja id="002">
    <nimi>Beckhoff</nimi>
    <mallit>
      <malli>CX5130</malli>
      <malli>CX2030</malli>
      <malli>CX9020</malli>
    </mallit>
  </valmistaja>
</valmistajat>
```

Esimerkkikoodi 4. Tämänkaltainen rakenne mahdollistaa tiedon tehokkaan prosessin ja pitää hierarkian rakenteen luettavana. Esimerkissä on myös demonstroitu miten lapsielementillä <mallit> on useampia arvoja mikä ei ole mahdollista attribuuteilla.

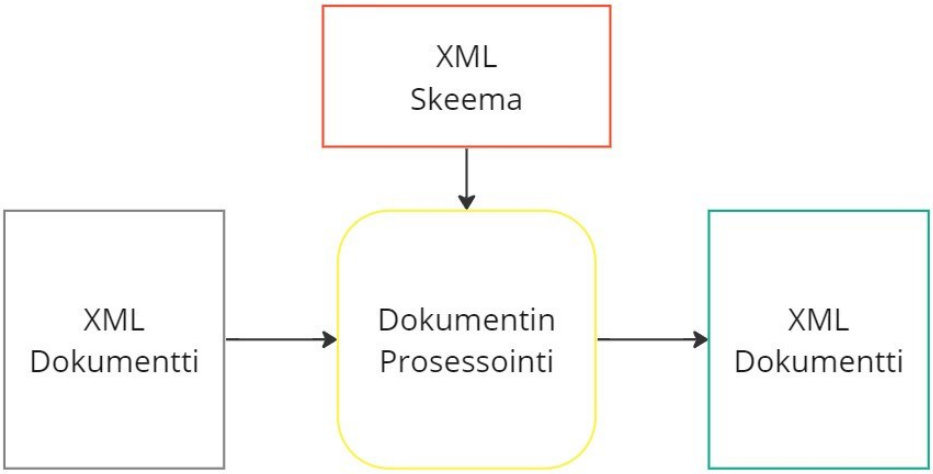
Elementtien ja attribuuttien kirjainkoko vaikuttaa myös tunnistamiseen, mikä tekee kielestä kirjainriippuvaisen. [35.]

```
<Valmistajat>
  <Nimi>Siemens</Nimi>
  <Malli>S7-1500</Malli>
</Valmistajat >
<valmistaja>
  <nimi>Siemens<nimi>
  <malli>30</malli>
</valmistaja>
```

Esimerkkikoodi 5. Isolla ja pienellä kirjoitetulla elementit ovat eri elementtejä. Kirjainkoon kanssa pitää olla tarkkana, kun käytössä on skeema.

4.2.2 XML-skeemat PLC-ympäristössä

XML-tiedostoilla käytetään myös skeemoja. Ne määrittelevät dokumentin sallitun rakenteen, elementit ja ominaisuudet sekä datatyypit. Tämä mahdollistaa dokumenttien tarkastuksen, että ne noudattavat määrättyä rakennetta ja datatyyppiä (kuva 20). Tästä syystä merkintäkieltä pidetään luotettavana tiedon siirto- ja tallennusformaattina. [38.]



Kuva 20. Kaavio XML-tiedoston muodon tarkastamisesta XML-skeemalla [38].

Standardin IEC 61131-10 osio käsittelee PLC-järjestelmien tiedonvaihtoa, XML-skeemoja, PLC-ohjelmien siirrettävyyttä sekä automaatiojärjestelmien dokumentointia. Sen tavoitteena on helpottaa tiedonsiirtoa ja yhteentoimivuutta eri laitteiden ja ohjelmistojen välillä valmistajasta riippumatta. OpenPLC-organisaation kehittämä skeema, joka määrittelee standardoidut rakenteet ja säännöt konfiguraatio- ja ohjelmointitietojen esittämiseen XML-formaatissa, on osa tätä standardia. [39;40.]

Osiossa kuitenkin esitellään vaihtoehtoinen menetelmä omien XML-skeemojen käytölle [40], kuten Siemens on toimintatavakseen valinnut. Siemensin kehittämä oma formaatti on suunniteltu yhteensopivaksi vain TIA Portalin kanssa.

Tämän seurauksena tiedostojen siirrettävyys ja yhteistoimivuus muiden valmistajien ja niiden ohjelmointiympäristöjen rajoittuu. Tällä saadaan tehokkaasti estettyä käyttäjän projektin siirtäminen toisen valmistajan alustalle nopeasti ja tehokkaasti jos, vaikka toimitusvaikeuksien, ominaisuuksien tai kilpailukykyisemmän hinnan takia haluttaisiin vaihtaa käyttämään vaihtoehtoista alustaa kuten Codesys tai Twincat.

4.2.3 Tiedostojen käsittely

Tiedostot ovat rivimäärältään usein hyvin suuria, joten manuaalisesti ei ole tehokasta käsitellä niitä. Sovellukset käyttävät yleensä jonkinlaista ohjelmistokirjastoa tiedostojen prosessointiin. Prosessi on yleisesti kaksivaiheinen:

- **Tiedoston lukeminen** (Parsing). Aluksi parseri lukee tiedoston ja jäsentelee dokumentin rakenteen (elementit, attribuutit, tekstisisällöt jne.) mukaan ja muodostaa siitä rakenteellisen muodon kuten DOM-puun (Document Object Model). Lukuvaiheessa tiedostoa myös verrataan skeemaan ja tarkastetaan dokumentin muotoilun oikeellisuus. Tarkoituksena siis on erottaa data sen esitysmuodosta käsittelyä varten.
- **Käsittelyvaihe** (Serialization). Lukemisen jälkeen data on käytössä ohjelmiston sisäisessä muodossa, ohjelmisto voi käyttää tiedostoa haluamallaan tavalla. Tämä voi olla datan visualisointia, sen tallentamista tietokantaan, laskutoimitusten suorittamista tai uuden tiedon luontia ja kirjoitusta tiedostoon. Kirjoitusvaiheen jälkeen tiedosto muunnetaan takaisin alkuperäiseen XML-muotoon.

Tiedon prosessointia varten löytyy valmiita kirjastoja, joiden käyttö tehostaa tiedostojen manipulaatiota. [36;38.]

4.3 Visual Studio

Visual Studio on Microsoftin kehittämä integroitu kehitysympäristö (IDE), joka on tarkoitettu monipuoliseen sovellusten ja palveluiden kehittämiseen eri alustoille. Käyttäjille tarjotaan monipuolisia työkaluja koodin kirjoittamiseen, kääntämiseen, virheenjäljitykseen ja ohjelmistojen julkaisemiseen. Ympäristöstä löytyy valmiina tuki kielille: C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML, ja CSS. Markkinoiden johtaviin

käyttöympäristöihin kuuluminen perustuu sen kattaviin ominaisuuksiin, laajaan kieli- ja alustatukeen sekä integroituihin kehitystyökaluihin. [41.]

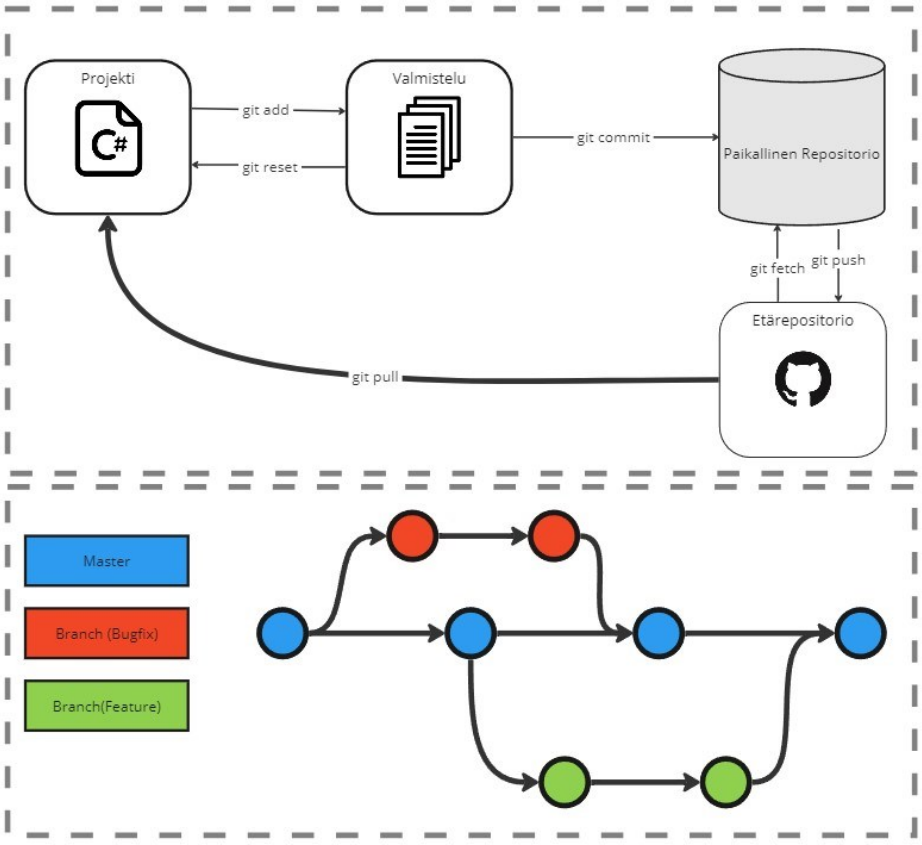
Laajennuksia, jotka tehostavat työskentelyä ja mahdollistavat monien muiden kielien käytön, on saatavilla kattavasti Visual Studioon. Esimerkiksi PLC-ohjelmoinnissa käytettävä Structured Text -kieli voidaan lisätä kehitysympäristöön laajennuksen avulla. Laajennuksena on saatavilla myös useita eri AI-työkaluja tehostamaan ohjelmointia kuten Tabnine, Github Copilot ja IntelliSense. [41.]

4.4 Versionhallinta

Git on hajautettu versionhallintajärjestelmä, se ei ole pelkästään sovellus. Työkalu mahdollistaa useiden kehittäjien samanaikaisen työskentelyn samassa projektissa. Hajautetun luonteen ansiosta kehitystyö voidaan toteuttaa omalla laitteistolla ilman verkkoyhteyttä. Muutokset synkronoidaan säilytyspaikkaan. Versionhallinnan avulla projektin nykyisten ja menneiden versioiden varmuuskopiot voidaan säilyttää, muutoksia kommentoida, projektin edistymistä seurata ja muutosten tekijät tunnistaa. Git on myös hyödyllinen työkalu virheiden etsimisessä ohjelmasta. Projekteja voidaan tallentaa ja jakaa erilaisissa palveluissa kuten Github. Github -projektissa tiedostot on organisoitu repositorioihin(repos) ja haaroiksi(branches). Seuraavaksi esitellään tärkeimmät ominaisuudet lyhyesti. [42;43.]

- **Repositorio** on projektin säilytyspaikka, joka sisältää kaikki tiedostot kuten koodin, dokumentaation ja konfiguraatiotiedostot. Se voi olla julkinen tai yksityinen.
- **Haara** on repositorion sisällä oleva tallennussijainti. Päähaara, johon lopullinen ja tuotantovalmis koodi yhdistetään. Se nimetään yleisesti terminä Main tai Master.
- **Kehityshaarat** (branches) käytetään uusien ominaisuuksien kehittämiseen, bugien korjaamisen tai testaukseen. Kehittäjät luovat usein uuden haaran (feature branch) jokaiselle uudelle ominaisuudelle. Valmis ominaisuus tuodaan takaisin päähaaraan.
- **Pull Request** (PR): Kun kehittäjä haluaa yhdistää kehityshaaransa muutokset päähaaraan (tai toiseen haaraan), hän luo vetopyynnön.

- **Fork:** Projektin kopio. Fork antaa käyttäjälle vapaat kädet tehdä muutoksia kopioituun projektiin ilman, että se vaikuttaa alkuperäiseen repositorioon. Käyttäjä voi myöhemmin ehdottaa muutoksiaan alkuperäiseen projektiin vetopyyntöjen kautta.



Kuva 21. Github -työnkulku sijaintien ja versioiden välillä.

4.5 C# ja .NET

C# on Microsoftin ohjelmointikieli, joka julkaistiin osana .NET-alustaa vuonna 2000. Kieli on saanut vaikutteita C++ ja Javasta mikä on nähtävissä sen syntaksissa ja ominaisuuksissa. [44.]

.NET-alusta tarjoaa laajan kirjaston ja runtime-ympäristön eri tarkoituksiin soveltuvan sovelluskehityksen tueksi, mukaan lukien web-, mobiili-, työpöytä- ja pilvisovellusten kehittämisen [45]. Tämän alustan merkitystä kehittäjäyhteisölle korosti entisestään Microsoftin entinen toimitusjohtaja Steve Ballmer, joka

tunnetusti korosti "Developers, developers, developers!" -huudolla ohjelmistokehittäjien tärkeyttä Microsoftin strategiassa ja tuotteiden menestyksessä [46].

Alustalta on saatavilla useita työkaluja työpöytäsovellusten kehittämiseen, mukaan lukien Windows Forms. Tämä työkalu tarjoaa yksinkertaisia ja tehokkaita menetelmiä Windows-pohjaisten sovellusten luomiseen. Työkalu sisältää valmiita käyttöliittymäelementtejä kuten painikkeita ja tekstikenttiä, joita voidaan drag&drop -menetelmällä lisätä projektiin. [47.]

Ympäristön muita oleellisia osia ovat kolmannen osapuolen tekemät valmiit kirjastot, joita voidaan tuoda projektiin esimerkiksi Visual Studiossa Nuget-paketinhallintajärjestelmän kautta osaksi C#-projektia [45][41]. Myös DLL (Dynamic Link Library) -kirjastot ovat keskeisiä komponentteja ympäristössä. DLL-tiedosto on kirjastotiedosto, joka sisältää koodia ja resursseja, joita eri ohjelmat voivat yhteisesti käyttää Windows-käyttöjärjestelmässä. [47.]

Olio- ja komponenttipohjainen ohjelmointi on oleellinen osa C#-kieltä [44]. Komponenttipohjaisuudella tarkoitetaan, että ohjelma koostuu pienistä uudelleen käytettävistä palasista. Olio-ohjelmointi on menetelmä, joka keskittyy olioiden käyttämiseen ohjelmistojen rakentamisessa. Tärkeimpiä käsitteitä ovat:

- **Luokat**: Määrittelevät olioiden rakenteen ja toiminnallisuuden.
- **Metodit**: Luokkaan kuuluvat funktiot, jotka suorittavat toimintoja luokan olioiden kanssa tai näille olioille.
- **Konstruktorit**: Erityisiä metodeja, jotka suoritetaan automaattisesti, kun luokan instanssi (olio) luodaan.
- **Polymorfismi**: Mahdollistaa aliluokkien olioiden käsittelyn yliluokan olioina, mahdollistaen metodien ylikirjoituksen.
- **Kapselointi**: Piilottaa luokan sisäisen tilan ja suojaa sitä suorilta muutoksilta ulkopuolelta.

Hyötynä menetelmällä on koodin uudelleen käytettävyys sekä oikein toteutettuna se parantaa koodin luettavuutta ja ymmärrettävyyttä ulkopuolisille. [49.]

4.6 UI/UX-perusteet

UI (User Interface) ja UX (User Experience) eli käyttöliittymä ja käyttökokemus ovat keskeisiä käsitteitä ohjelmoinnissa. Termi UI tarkoittaa lähinnä sovelluksen visuaalisia elementtejä ja käytettävyyttä, kun taas UX käsittää laajemman kokonaisuuden missä otetaan huomioon kuinka helppokäyttöinen ja intuitiivinen ohjelmisto on. [50.]

Millerin laki perustuu kognitiivisen psykologian periaatteeseen, jonka mukaan ihminen pystyy pitämään työmuistissaan keskimäärin 7 erillistä yksikköä kerrallaan [51]. Hickin laki toteaa, että valinnan tekemiseen kuluva aika kasvaa logaritmisesti, kun valittavissa olevien vaihtoehtojen määrä kasvaa [52]. 60–30–10 värimaailma on suunnitteluperiaate, joka auttaa luomaan tasapainoisen ja esteettisesti miellyttävän värimaailman. Tässä säännössä 60 % tilasta dominoi hallitseva väri, 30 % toissijainen väri ja 10 % korostusväri [53].

Yhteenvetona näiden lakien ja sääntöjen perusteella voidaan sanoa, että rajoittamalla näkyvillä olevaa informaation määrää ja pitämällä sisällön selkeänä. Saadaan aikaisiksi sovelluksen käyttöliittymä, joka on tehokas ja miellyttävä käyttää.

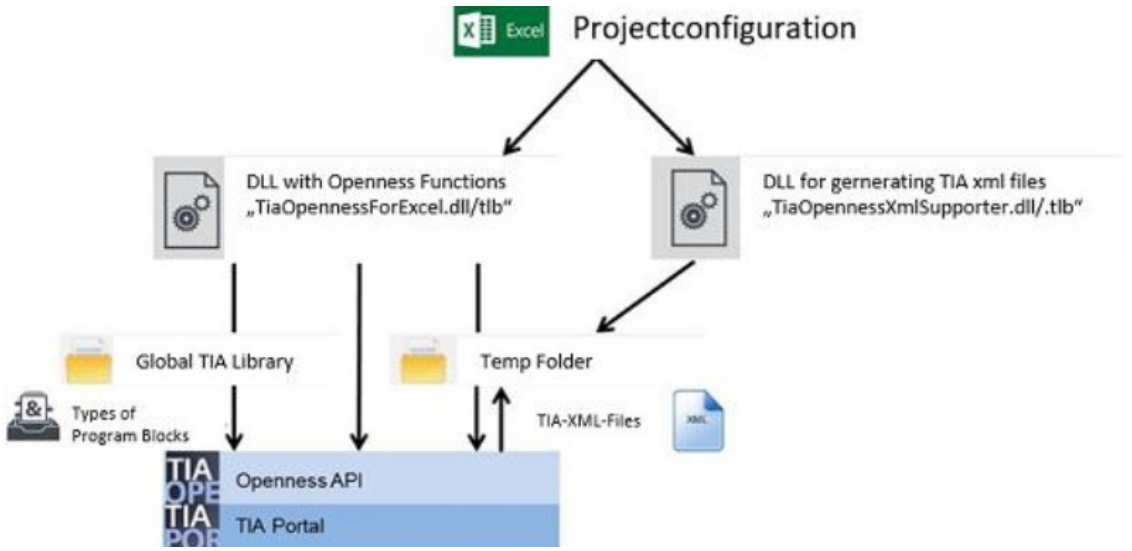
5 Demot

Tässä kappaleessa käymme läpi ja analysoimme Siemensin demosovelluksia, jotka esittelevät Openness-rajapinnan mahdollisuuksia. Yleisesti voidaan olettaa, että kaikki demot ja seuraavassa luvussa esitelty oma työkalut vaativat toimiakseen Windows– pohjaisen tietokoneen, jossa on asennettuna .NET Framework 4.6.1 Runtime tai uudempi ja TIA Portal -versio 16 sekä TIA Openness

API. Sekä käyttäjä on määritellyt tarpeelliset käyttäjäasetukset Windows -ympäristössä

5.1 Excel code generator

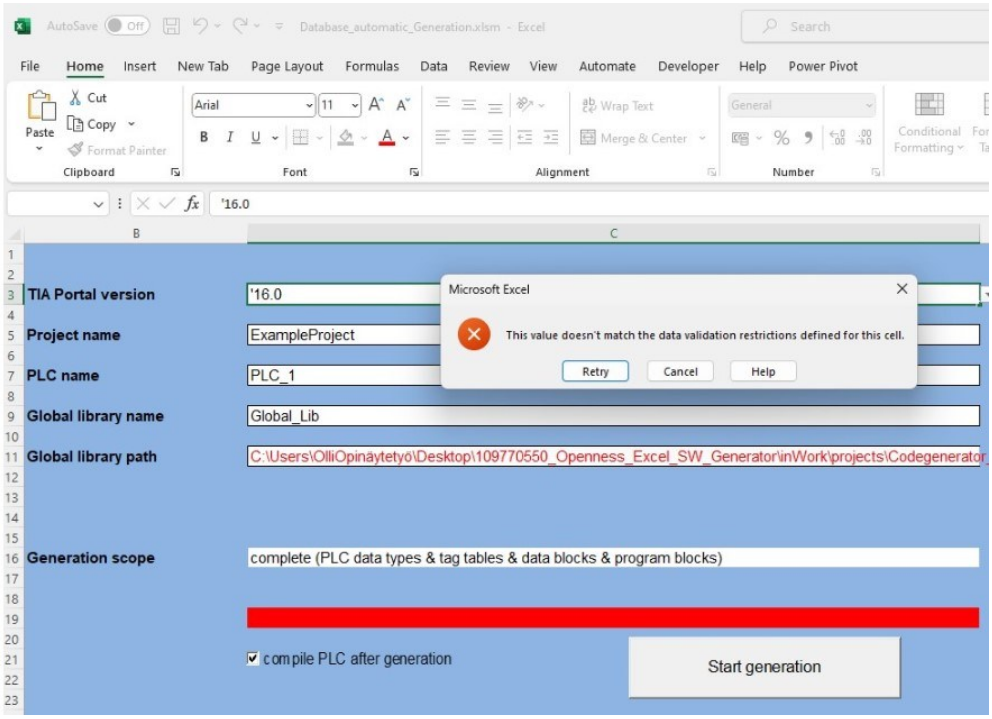
Kyseessä on Excel -pohjainen työkalu, jonka avulla on mahdollista generoida tehokkaasti objekteja TIA Portal -projektiin. Openness-rajapinnan .dll -kirjastot eivät ole COM (Component Object Model) -näkyviä eli Excel ei voi suoraan kommunikoida rajapinnan kanssa. Ratkaisuna Siemens on kehittänyt .dll kirjaston, joka hoitaa kommunikoinnin Excel-tiedoston ja rajapinnan välillä. Mukana tulee myös toinen .dll kirjasto mikä muuntaa Excel-välilehtien sisällä olevan tiedon XML-muotoon ennen tiedonsiirtoa sovellusten välillä. [54.]



Kuva 22. Excel-tiedoston ja rajapinnan välinen tiedonsiirto [54].

Työkalulla on mahdollista luoda: Datatyyppejä, Tag-muuttujia, Datalohkoja, Ohjelmalohkoja sekä asettamaan generoidut muuttujat generoituihin ohjelmalohkoihin. Työkalu rakentaa nämä globaalin kirjaston pohjalta, jonka tiedostopolku tulee olla määritettynä oikein ennen generointia. [54.]

Työkalun pitäisi tukea versioita V15, V15.1 ja V16, mutta tämän opinnäytetyön aikana sitä ei saatu toimimaan versiolla V16(Kuva 23).

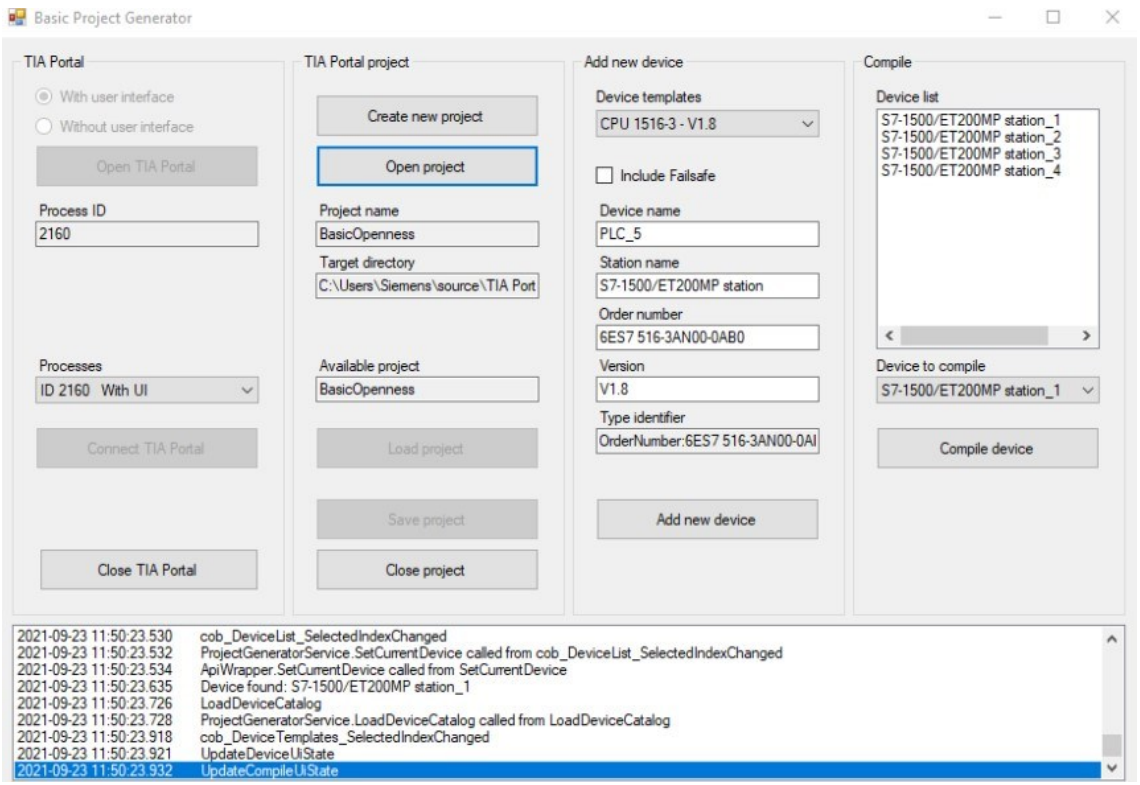


Kuva 23. Excel-työkalun näkymä ja yhteensopivuusongelma.

Mahdollista olisi tietenkin ollut joko ladata TIA Portalin versio V15 demo ja koittaa toimiiko työkalu sillä tai etsiä bugia lähdekoodista, mutta sitä testatessa seuraavaksi tuli yhteensopivuusongelmia Visual Studion version 2022 kanssa, joten tiukan aikataulun vuoksi, tutkimuksessa siirryttiin eteenpäin muiden vaihtoehtojen kanssa.

5.2 Basic Project Generator

Kyseessä on Siemensin kehittämä demosovellus, joka lähinnä demonstroi mahdollisuutta toteuttaa TIA Portalin perustoimintoja rajapinnan kautta. Työkalusta löytyy jo (kuva 24) useita yleisiä perustoimintoja kuten TIA Portalin käynnistys, sulkeminen, projektien luonti, tallennus, avaus, ja yhdistäminen avoimiin projekteihin. Tämän lisäksi sillä voidaan generoida laitteistoa yhdistettyyn projektiin ja kääntää (compile) haluttu laitteisto. Vaihtoehtoina generointiin löytyy muutama eri PLC-malli, joille voidaan asettaa sovelluksessa muutamia parametrejä kuten laitenimi tai salasana. [55.]



Kuva 24. Sovelluksen käyttöliittymä [55].

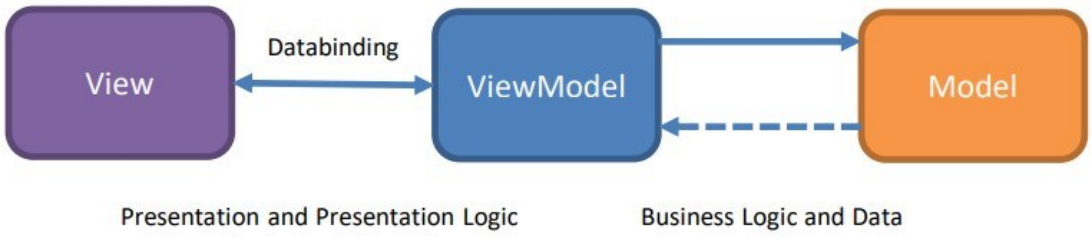
Demosovelluksen mukana toimitetaan sovelluksen lähdekoodi ja kattava dokumentointi liittyen sovelluksen toimintoihin. Basic Project Generator on lähtökohtaisesti suunniteltu oman sovelluksen rakentamiseen sen lähdekoodin pohjalta. Sovelluksesta löytyykin jo valmiiksi kehitetty tapahtumaloki, jota voidaan hyödyntää, kun koodia testataan. [55.]

5.3 Openness Demo

Openness Demo kuten Basic Project Generator on suunniteltu rajapinnan toimintojen esittelyyn ja käytettäväksi lähtökohtana oman projektin rakentamiseen. Kyseinen sovellus on luotu käyttäen Model-View-ViewModel (MVVM) ohjelmistoarkkitehtuuria. Rakenne tarkoittaa, että sen rakenne on jaettu kolmeen luokkaan [55]:

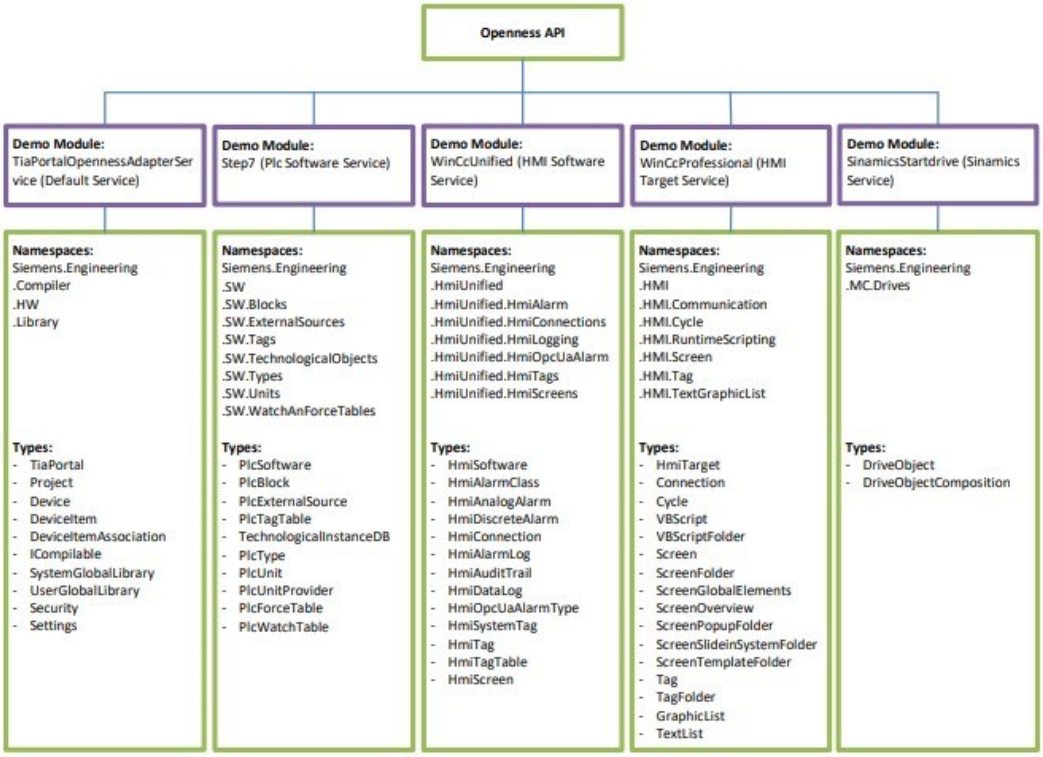
- View (näkymä) – Käyttöliittymä ja siihen liittyvät toiminnot

- ViewModel (näkymämalli) – Välittää tietoja näkymän ja mallin välillä sisältäen logiikan ja tilanhallinnan.
- Model (malli) – Edustaa sovelluksen tietoja ja liiketoimintalogiikkaa, vastaa tietojen hakuun, tallennukseen ja käsittelyyn liittyvistä toiminnoista.



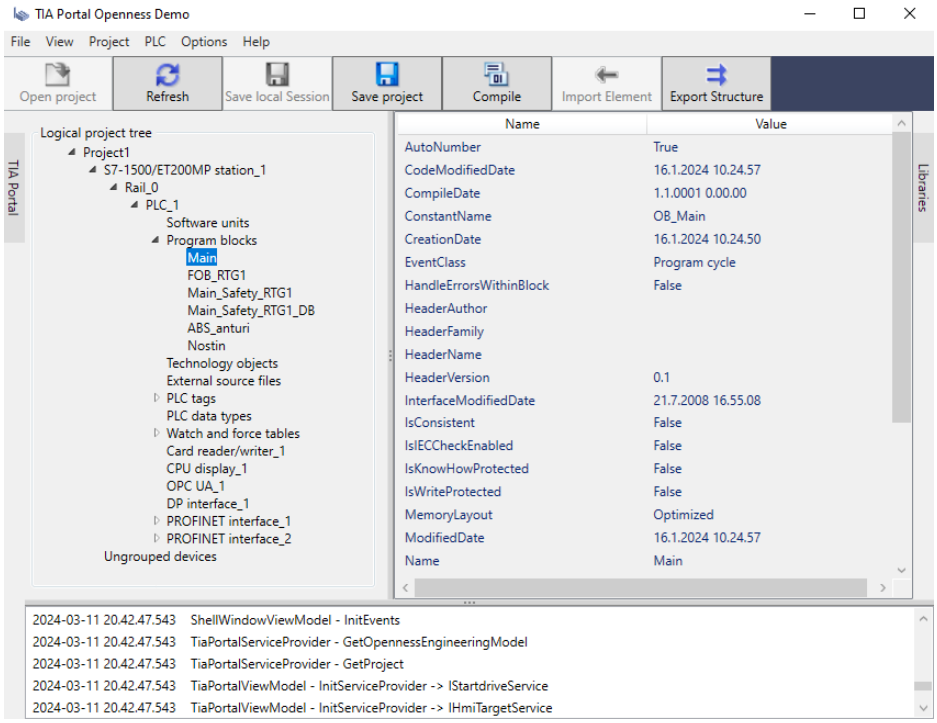
Kuva 25. Sovelluksen arkkitehtuuri [55].

Verrattuna edellisessä osiossa esiteltyyn sovellukseen tämä demo esittelee edistyneempiä toimintoja ja miten niitä on mahdollista toteuttaa. Kuten miten sovellus saadaan toimimaan riippumatta käytössä olevan TIA Portalin versiosta. Lähdekoodista ja dokumentaatiosta tulee myös esille, miten ohjelmiston rakenne on pilkottu pieniksi kokonaisuuksiksi eri moduuleihin (kuva 26). [55.]



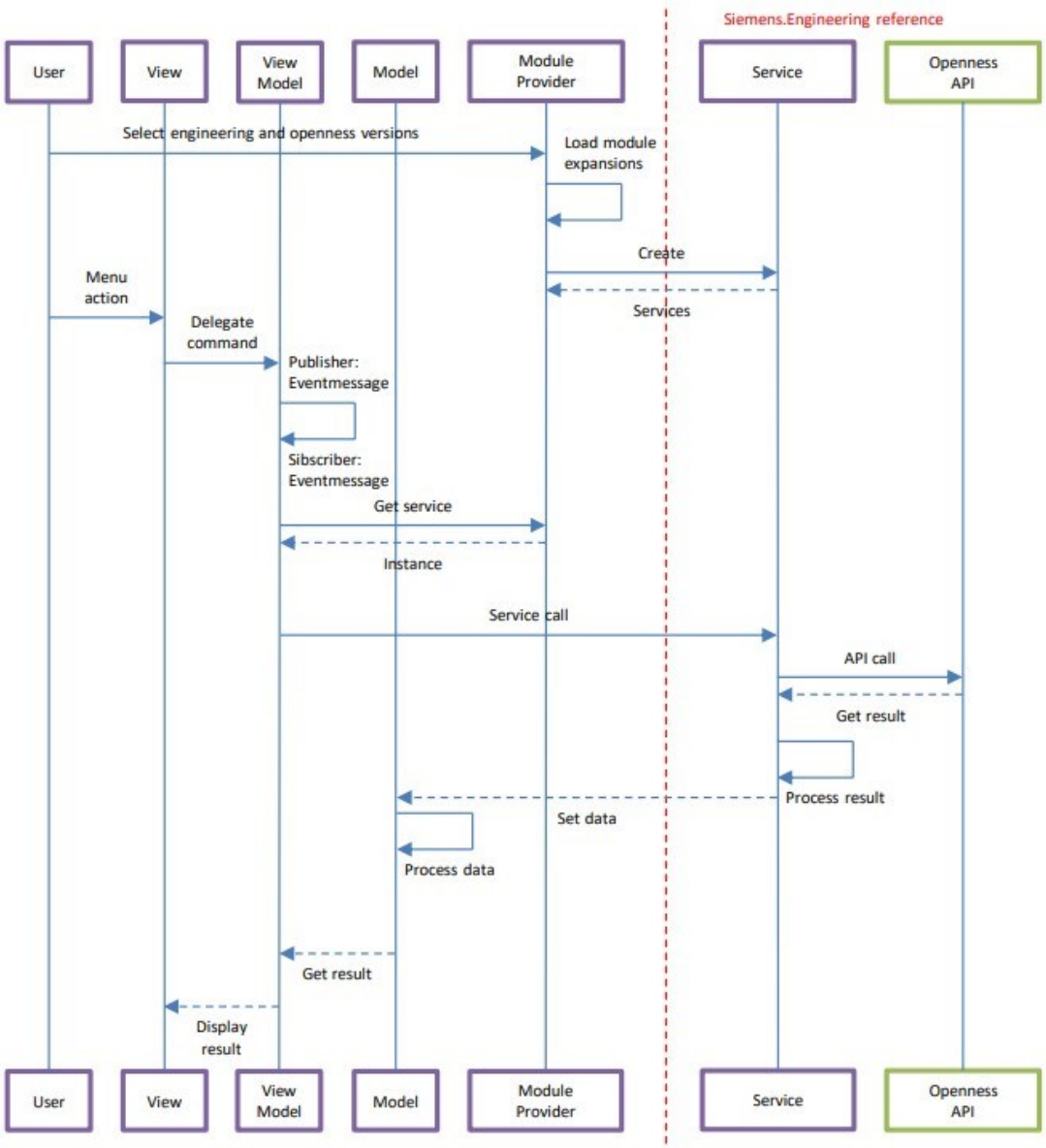
Kuva 26. Demosovelluksen moduulit [55].

Demosovelluksen avulla voidaan suorittaa lähes kaikki samat toiminnot kuin Siemensin ohjelmointiympäristössä PLC:n ollessa offline-tilassa. Lisäksi sovellus tarjoaa visualisoidun ja interaktiivisen yhdistetyn projektin hierarkianäkymän (kuva 27), joka tukee yleisiä hiiren toimintoja.



Kuva 27. Sovelluksen käyttöliittymä [55].

Käytännössä sovellus on TIA Portalin etäohjain. Laaja dokumentaatio ohjelman rakenteesta ja rajapinnan välisestä vuorovaikutuksesta (kuva 28) tarjoaa vankan perustan oman työkalun kehittämiseen.

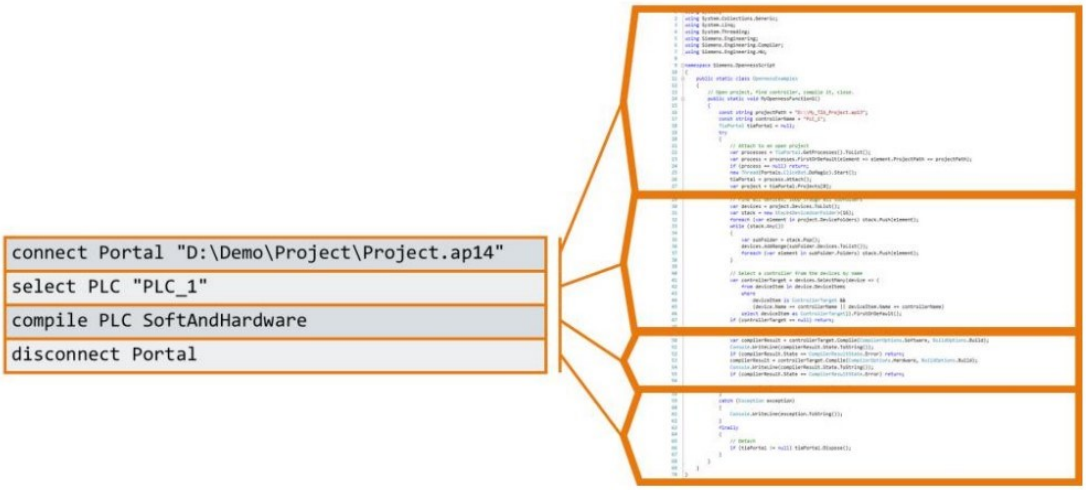


Kuva 28. Sekvenssikaavio sovelluksen ja rajapinnan välisestä vuorovaikutuksesta [55].

5.4 Openness Scripter

Sovellus on tarkoitettu lähinnä yksinkertaisten komentojen suorittamiseen kuten yksittäisten objektien vientiin ja tuontiin TIA Portalin kanssa. Se on suunnattu aloittelijoille eikä edellytä aikaisempaa ohjelmointikokemusta, tai syvällisempää TIA Portal osaamista. Sovelluksesta (kuva 29) löytyy myös esimerkkejä ja Siemens on tehnyt selkeät ohjeet yksinkertaisten toimintojen toteuttamiseen. [56.]

Skriptien suorittamisen syntaksi on sovelluksessa yksinkertaistettu verrattuna vastaavan toiminnon toteuttamiseen C#-kielellä (kuva x). Esimerkiksi yksittäisen datalohkon vienti TIA Portalista saadaan toteutettua muutamalla komennolla, kun taas C# kielellä se vaatisi lähemmäs sataa riviä koodia. [56.]



Kuva 29. Vasemmalla yksinkertaisen toiminnon toteutus Openness Scripterillä ja oikealla sama toteutettuna C#-kielellä [57].

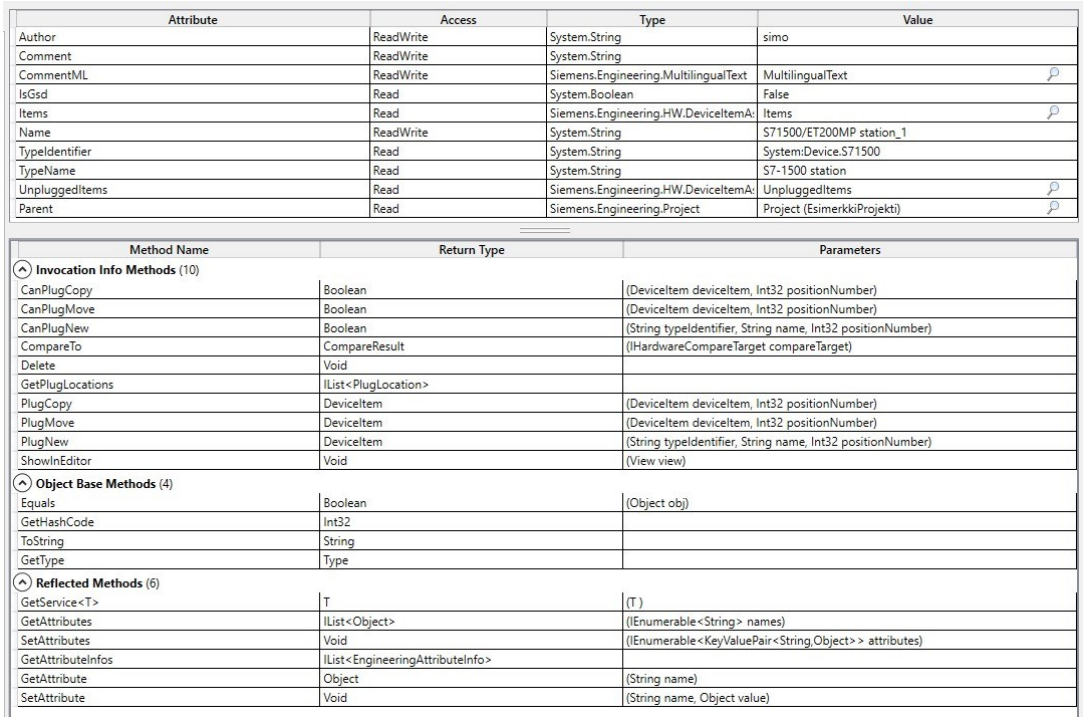
Ainoastaan Openness API:n perustoiminnot on tarjottu OpennessScripterin toimesta, ja suurimpana puutteena on havaittu, että syntaksista puuttuu kokonaan mahdollisuus ohjelmien suorittamiseen silmukkarakenteisena FOR- tai WHILE-rakennetta käyttäen. Ohjelman toistuvan suorituksen voisi mahdollisesti toteuttaa käyttämällä batch-tiedostoa (.bat), joka ohjaa OpennessScripter-sovelluksen toimintaa. Sovelluksen hyvänä puolena on, että se mahdollistaa toimintojen nopean testaamisen ilman tarvetta kokonaisen monimutkaisen ohjelman rakentamiseen, kuten C#-kielellä tehtäessä. [57.]

5.5 Openness Explorer

Openness Explorer on suunnattu sovelluskehittäjille ja se antaa kattavan yleiskatsauksen rajapinnan toiminnoista. Työkalun avulla voidaan tutkia rajapinnan käyttöä ilman oman sovelluksen luomista, ja voidaan tarkistaa, onko tietty

ominaisuus tai tieto saavutettavissa TIA Openness API:n kautta. Työkalu ei siis ole korvike Openness-sovellukselle. [58.]

Sovelluksessa tyypillinen työn kulku on TIA Portalin käynnistäminen ja halutun projektin avaaminen, minkä jälkeen navigoidaan halutun objektin kohdalle rakennepuussa. Sovelluksen oikealla puolella esitetään yleiskatsaus objektin attribuuteista ja metodeista (Kuva 30).

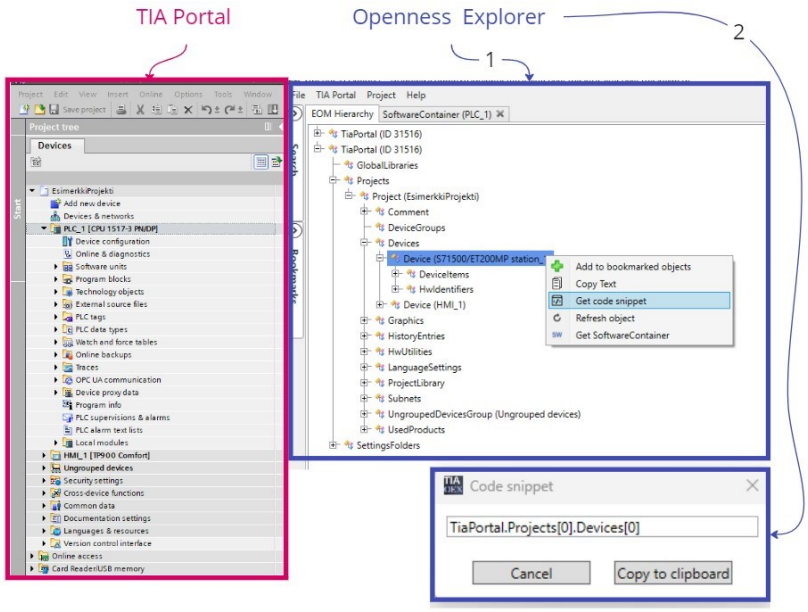


Attribute	Access	Type	Value
Author	ReadWrite	System.String	simo
Comment	ReadWrite	System.String	
CommentML	ReadWrite	Siemens.Engineering.MultilingualText	MultilingualText
IsGsd	Read	System.Boolean	False
Items	Read	Siemens.Engineering.HW.DeviceItemA	Items
Name	ReadWrite	System.String	S71500/ET200MP station_1
TypeIdentifier	Read	System.String	System:Device.S71500
TypeName	Read	System.String	S7-1500 station
UnpluggedItems	Read	Siemens.Engineering.HW.DeviceItemA	UnpluggedItems
Parent	Read	Siemens.Engineering.Project	Project (EsimerkkiProjekti)

Method Name	Return Type	Parameters
Invocation Info Methods (10)		
CanPlugCopy	Boolean	(DeviceItem deviceItem, Int32 positionNumber)
CanPlugMove	Boolean	(DeviceItem deviceItem, Int32 positionNumber)
CanPlugNew	Boolean	(String typeIdentifier, String name, Int32 positionNumber)
CompareTo	CompareResult	(IHardwareCompareTarget compareTarget)
Delete	Void	
GetPlugLocations	IList<PlugLocation>	
PlugCopy	DeviceItem	(DeviceItem deviceItem, Int32 positionNumber)
PlugMove	DeviceItem	(DeviceItem deviceItem, Int32 positionNumber)
PlugNew	DeviceItem	(String typeIdentifier, String name, Int32 positionNumber)
ShowInEditor	Void	(View view)
Object Base Methods (4)		
Equals	Boolean	(Object obj)
GetHashCode	Int32	
ToString	String	
GetType	Type	
Reflected Methods (6)		
GetService<T>	T	(T)
GetAttributes	IList<Object>	(IEnumerable<String> names)
SetAttributes	Void	(IEnumerable<KeyValuePair<String,Object>> attributes)
GetAttributeInfos	IList<EngineeringAttributeInfo>	
GetAttribute	Object	(String name)
SetAttribute	Void	(String name, Object value)

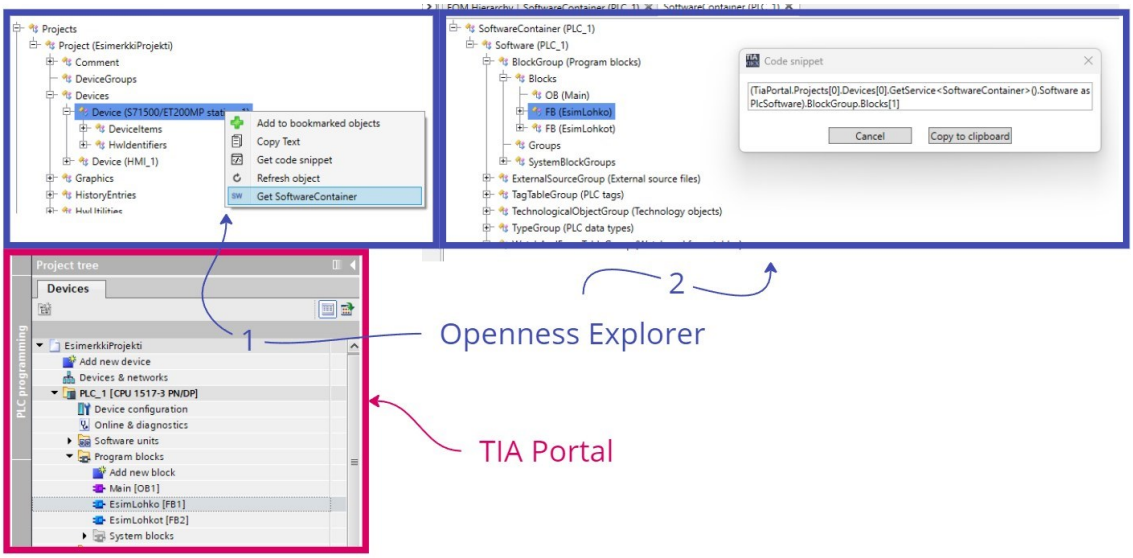
Kuva 30. PLC-Objektin attribuutit ja metodit.

Projektissa voi olla useita PLC-ohjaimia, ja kun halutaan kohdentaa komento koskemaan tietyn PLC:n yksittäistä objektia, on tehokasta navigoida kyseisellä sovelluksella sen kohdalle ja kopioida sovelluksesta polku käyttämällä työkalua code snippet (kuva 31).



Kuva 31. Projektinäkymä TIA Portalissa ja Openness Explorerissa.

Sovelluksessa lohkoihin liittyvät toiminnot tuodaan näkyville valitsemalla projektinpuusta toiminnolla GetSoftwareContainer, mikä avaa uuden välilehden, jossa ohjelmointiin liittyviä toimintoja esitellään (Kuva 32).

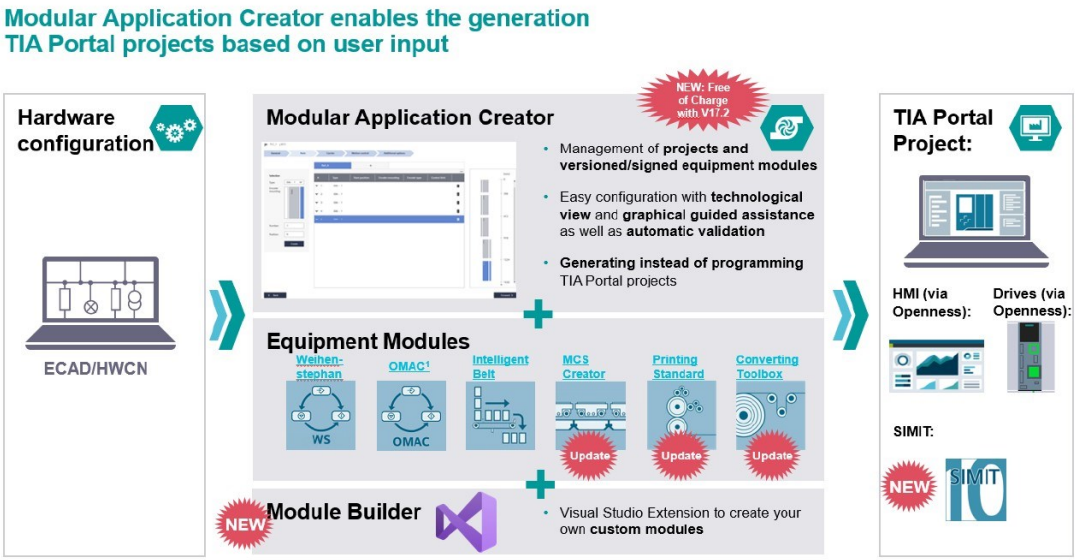


Kuva 32. Esimerkki miten komentoja voidaan kohdistaa tiettyyn lohkoon.

Sovelluksen avulla voidaan siis manuaalisesti suorittaa lähes kaikki samat toiminnot kuin TIA Portalissa, kun laitteisto on offline-tilassa. Sen käytön suurimpana hyötynä pidetään rajapintojen toimintojen ja niiden syntaksin tutkimista sekä niiden myöhempää soveltamista omassa sovelluksessa. [58.]

5.6 Modular Application Creator

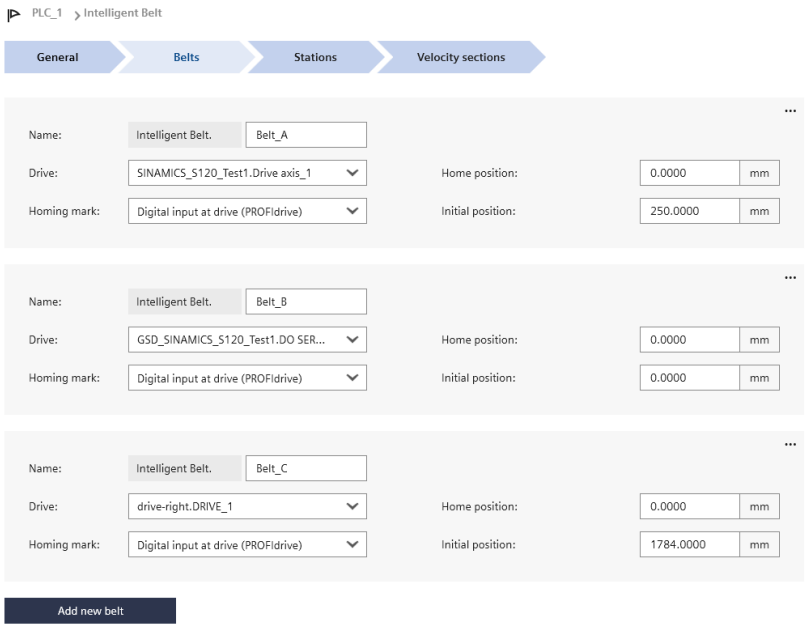
Modular Application Creator (MAC) -sovelluksella pystytään luomaan kokonaisia TIA Portal -projekteja, HMI-käyttöliittymiä, SIMIT-simulointimalleja sekä konfiguroituja ohjaimia. Valmiit projektit luodaan automaattisesti käyttäen ennalta määritettyjä ja testattuja ohjelmistomoduuleja (kuva 33). Sovellukselle löytyy Siemensin kehittämiä valmiita moduuleita liittyen robotiikkaan, kappaleautomaatioon, liikkeenohjaukseen ja teknologiaobjekteihin. [59.]



Kuva 33. MAC-sovelluksen työnkulku [59].

Työskentely sovelluksella alkaa luomalla uusi MAC-projekti. Seuraavaksi valitaan käytettävä ohjelmistomoduuli ja hardware(laitteisto). Esimerkiksi moduulilla IntelligentBelt, joka on tehty kappaleautomaation tarpeisiin, voidaan määritellä haluttu määrä hihnoja, minkä tyyppisiä ne ovat (ketju- vai hihnakäyttö). Jokaisen

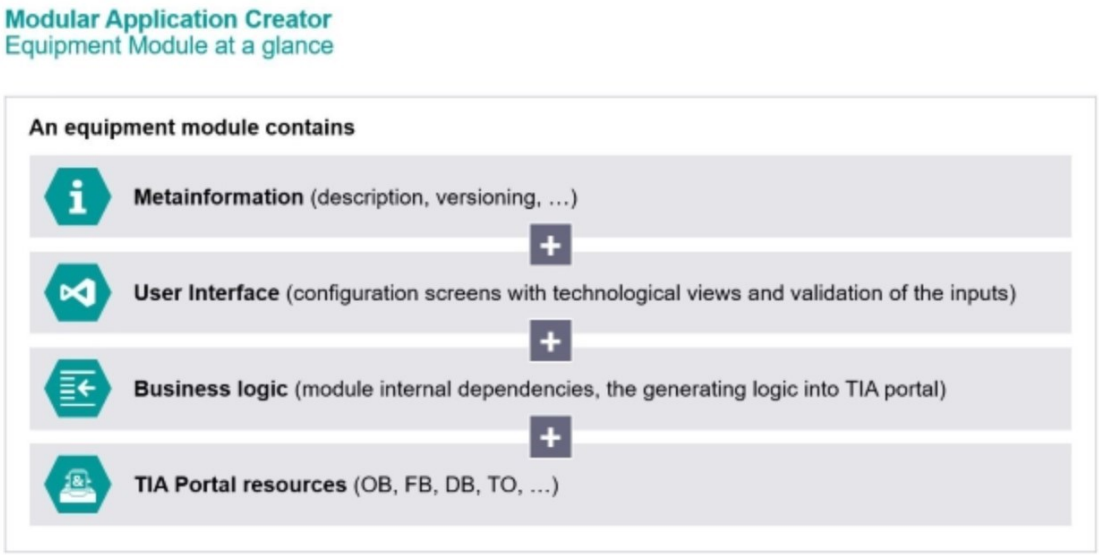
hihnan parametrit voidaan vielä määritellä tarkemmin moottorinohjauksen sekä paikkatiedon osalta (kuva 34).



Kuva 34. Hihnan parametrien määrittely.

Seuraavaksi voidaan määritellä työpisteiden toiminnot sen perusteella, toimivatko ne lastaus-, purku- tai kasaustyöpisteinä. Kun kaikki parametrit ovat määritelty, työkalu generoi annettujen arvojen pohjalta TIA Portal -projektin, HMI-käyttöliittymän ja Simit -simulointimallin. Nyt projektin pitäisi olla valmis suoraan ladattavaksi oikealla PLC:lle testausta varten tai vaihtoehtoisesti simulointia varten. [60.]

Sovellukseen on myös mahdollista rakentaa omia moduuleja Visual Studio -ohjelmointiympäristössä käyttäen Siemensin kehittämää laajennusta Modular Application Creator Module Builder -työkalua. Työkalulla voidaan rakentaa yrityksen omaan liikealueeseen soveltuva moduuli käyttäen toimiviksi testattuja TIA Portal -kirjastoja (kuva 35). Kun moduuli on saatu rakennettua Visual Studiolla, Siemensin laajennus kääntää sen kirjastoksi, joka integroidaan MAC-sovellukseen.



Kuva 35. Ohjelmistomoduulin sisältö [59].

Builderin käyttöön liittyen tarjolla on heikosti dokumentaatiota. Käytännössä yksi webinaari, jonka pohjalta pystyy rakentamaan oman moduuliin, mutta valitettavasti tämän työkalun tämänhetkisellä versiolla (V17.2) kirjastojen tuonti MAC-Builderiin aiheuttaa koko Visual Studion kaatumisen ja käytännössä tekee työkalusta käyttökelvottoman omien moduulien rakentamiseen. Tämä ongelma on Siemensin tiedossa, mutta korjauksesta ei ole kirjoitushetkellä tietoa. Tämä on erityisen harmillista, sillä työkalulla on suuri potentiaali tehostaa työskentelyä merkittävästi. Esimerkiksi, työkalun avulla on aiemmin onnistuttu vähentämään projektin ohjelmointiin kuluvaa aikaa kolmesta viikosta puoleen päivään, mikä osoittaa sen mahdollisuudet työskentelyn tehostamisessa [61]. [59.]

5.7 Yhteenveto

Yhteenvetona voidaan todeta, että ratkaisuna kehittää useita demoprojekteja, jotka tähtäävät lähes aina samaan päämäärään (suunnitteluprosessien automatisointi), saadaan lopputulokseksi kasa epävakaita ja puutteellisia sovelluksia. Tosin tähän termi "demo" (demonstration) viittaakin. Ehkä parempana ratkaisuna olisi ollut kehittää yhtä projektia eteenpäin panostamalla sen ominaisuuksien ja ohjelmiston laadun parantamiseen. Kun tutkitaan vielä aikaisempia

tutkimustöitä Openness-rajapintaan liittyen tai luetaan Siemensin foorumia, huomataan, että usein projektissa on sama tavoite: laitteiston ja ohjelmiston generointi jonkin aikaisemmin määritellyn tiedon pohjalta. Silti Siemens ei ole tarjonnut yksinkertaista ja toimivaa ratkaisua tähän ongelmaan valmiina. Lopuksi Taulukossa 4 on listattu demojen tärkeimmät ominaisuudet, yhteensopivuudet eri TIA Portal -versioiden kanssa sekä tieto siitä, onko lähdekoodi saatavilla.

Taulukko 4. Testattujen sovellusten eroavaisuudet.

Nimi	**Ominaisuudet**	**Yhteensopivuus**	**Lähdekoodi**
Excel Code Generator	Lohkojen luonti	V15, V15.1, V16?	Kyllä (C# ja VB.NET)
Openness Scripter	Manuaalinen objektien luonti / vienti	V13 Eteenpäin	Ei
Openness Explorer	Kaikki rajapinnan toiminnot	V13 Eteenpäin	Ei
MAC (Modular Application Creator)	Kokonaisten projektien luonti moduuleista.	V17 Eteenpäin	Ei (MAC-Builder Saatavilla C#-kiellä)
Openness Demo	Yleiset TIA Portal -toiminnot manuaalisesti. Projektin visualisointi,	V15 Eteenpäin	Kyllä (C# ja VB.NET)
Basic Project Generator	Hardware-generointi	V15 Eteenpäin	Kyllä (C# ja VB.NET)

6 Toteutus

Oman työkalun rakentaminen käynnistyi, kun aiheeseen liittyvään tietopohjaan oli riittävästi perehdytty. Kehitysprosessi aloitettiin määrittelemällä välttämättömät toiminnot sekä mahdolliset ylimääräiset laajennukset jatkokehitystä tai kehitystyölle mahdollisesti jäävää lisäaikaa varten.

Tekijälle uusien teknologioiden ja kielien, kuten Git, C#, XML ja TIA Openness, käyttö oli merkittävässä roolissa. Tämän takia tekeminen ei aina ollut tehokasta ja osaamista jouduttiin täydentämään ongelmatilanteissa. Sovelluksen kehitys eteni järjestelmällisesti, uusia toimintoja testattiin lisäyksen yhteydessä, ja muutokset tallennettiin versionhallintajärjestelmään.

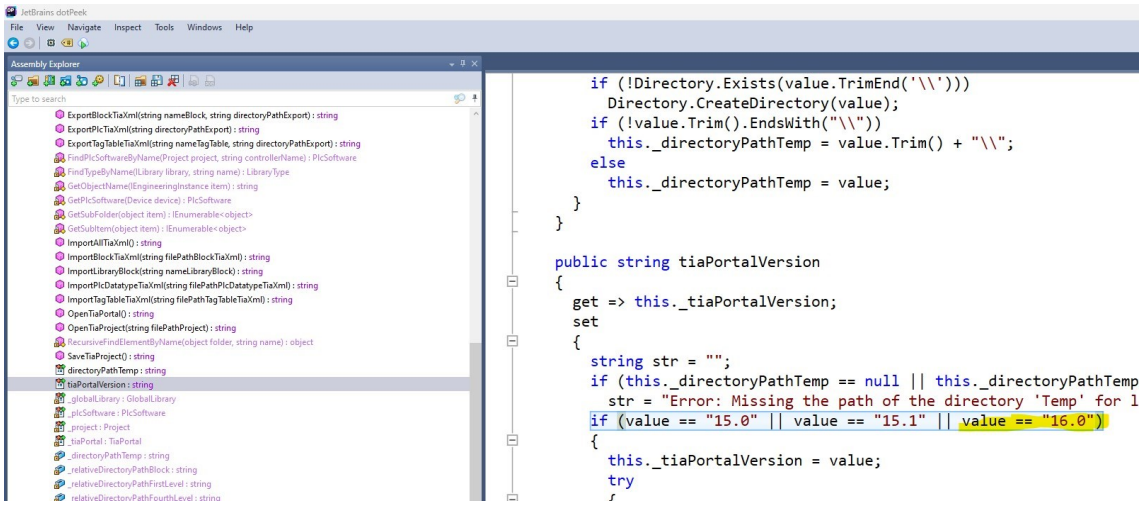
Työkalulle alustavasti suunnitellut toiminnot olivat selvästi rajattu jo aloituspalaverissa, ja tietopohjan perusteella oli selvää mitä oli mahdollista toteuttaa rajapinnan puitteissa ja mitä ei. Ainoa jäljellä oleva vaihe oli toteutus ja testaus.

6.1 Lähtökohta

Tavoitteena oli työkalu, jolla pystyttäisiin generoimaan lohkoja Excel tiedoston pohjalta ja demoihin perehtymisen jälkeen voidaan todeta, että jatkokehitystä varten oman työkalun rakennukseen parhaiten olisi soveltunut tähän tarkoitukseen Excel Code Generator tai Modular Application Creator. Valitettavasti nämä demot olivat epävakaita eikä niiden ominaisuuksiin päässyt tutustumaan tarpeeksi kattavasti, että olisi järkevä edetä.

Vaihtoehdoksi jäi rakentaa oma työkalu Openness Basic Project Generatorin ja Openness Demon lähdekoodin pohjalta. Tukena oman työkalun kehityksessä voidaan kuitenkin käyttää Excel Code Generatorin lähdekoodia ja sen mukana tulleet .dll-tiedostot saadaan avattua luettavaan muotoon lähempää tutkiskelua

varten decompile-työkalulla kuten JetBrainsin kehittämällä dotPeekillä (kuva 36).



Kuva 36. TiaOpennessExcel.dll avattuna decompile-työkalulla.

6.2 Ohjelmointikielen valinta

Lähes kaikki aiheeseen liittyvät aiemmat demot ja projektit oli toteutettu C#- tai Visual Basic.NET -ohjelmointikielellä, muutamaa Pythonilla tehtyä projektia lukuun ottamatta. C# ja VB.NET olivat suosituimpia kieliä, sillä rajapinnan kanssa tapahtuva kommunikointi oli toteutettu .dll-kirjastojen avulla. Ohjelmointiympäristön version mukaan .dll-kirjastot ovat yhteensopivia .NET Frameworkin version 4.6.1 tai sitä uudempien versioiden kanssa.

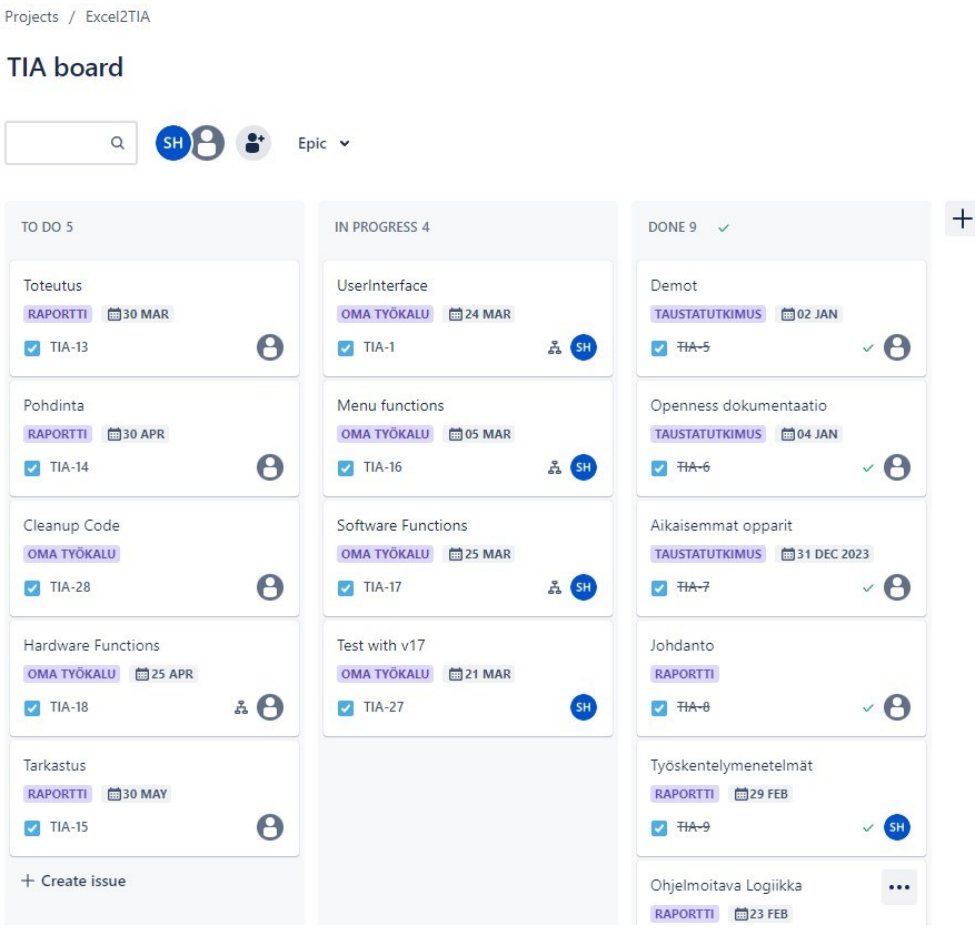
Eli C#- ja VB.net -kielten käyttämisen suosio johtui siitä, että ne voivat suoraan hyödyntää .dll-tiedostoja. Nämä kielet soveltuvat myös paremmin työpöytäsovellusten kehittämiseen kuin Python. Ne tarjoavat tehokkaat sekä monipuoliset työkalut UI:n (User Interface) kehitykselle, kuten Windows Forms ja WPF (Windows Presentation Foundation).

Kehitystyö Pythonia käyttäen olisi ollut mahdollista IronPythonin kaltaisten kirjastojen avulla, jotka mahdollistavat .NET-pohjaisten .dll-kirjastojen käyttämisen. Työpöytäsovelluksen käyttöliittymän rakentamiseen olisi vaadittu jonkin kolmannen osapuolen kirjaston käyttöä.

Aikataulullisten haasteiden vuoksi sovelluksen rakentaminen Pythonilla ei katsottu järkeväksi, kun demojen lähdekoodit olivat saatavilla C#- ja VB.Net -kielillä. C# valikoitui käytettäväksi kielenä, sillä sen syntaksi oli helpommin omaksuttavissa kuin VB.Net, johtuen aikaisemmasta kokemuksesta C- ja C++-kielten kanssa, joiden syntaksia C# muistuttaa. Lisäksi kielelle löytyy useita tehokkaita kirjastoja suurien XML-tiedostojen käsittelyyn.

6.3 Suunnittelu

Sovelluksen toiminnallisuuden suunnittelu käynnistyi listaten välttämättömät toiminnot ja mahdolliset laajennukset (liite 2). Toiminnot luokiteltiin alakategorioihin ja niiden puitteissa yksittäisiin toimintoihin. Pienempiin osiin jaettujen toimintojen edistymistä seurattiin Jirassa (kuva 37). Tämä menetelmä paransi huomattavasti työskentelyn tehoa.



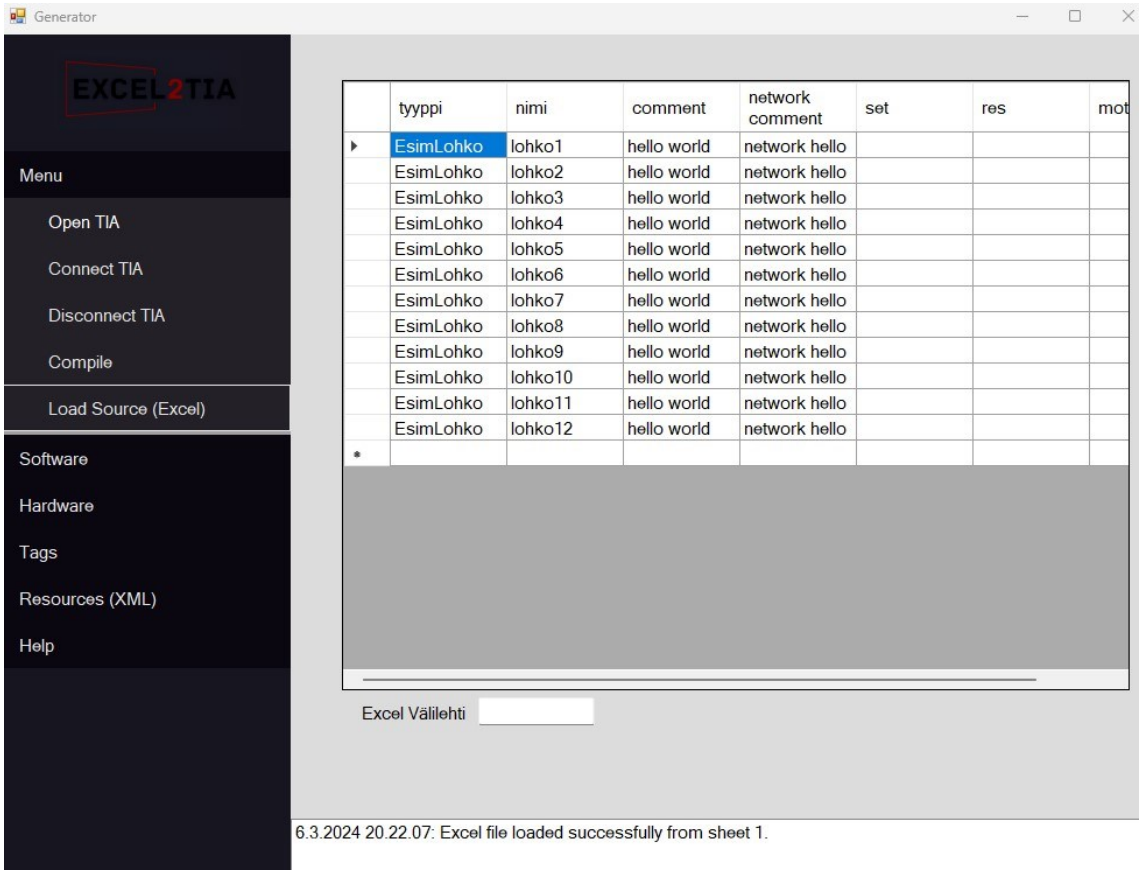
Kuva 37. Projektin tehtäviä Kanban taululla.

Työkalu rakennettiin alusta alkaen puhtaalta pöydältä sen sijaan, että olisi muokattu saatavilla olevien demojen lähdekoodeja. Tämän valinta tehtiin, jotta vianselvitys sekä testaus olisivat mahdollisimman tehokkaita. Lisäksi tämän ratkaisun avulla saavutettiin mahdollisimman paljon osaamista ohjelmoinnista. Saatavilla olevat demojen lähdekoodit tarjosivat kuitenkin arvokasta tietoa siitä, miten toiminnot voitaisiin toteuttaa.

6.4 Ulkoasu

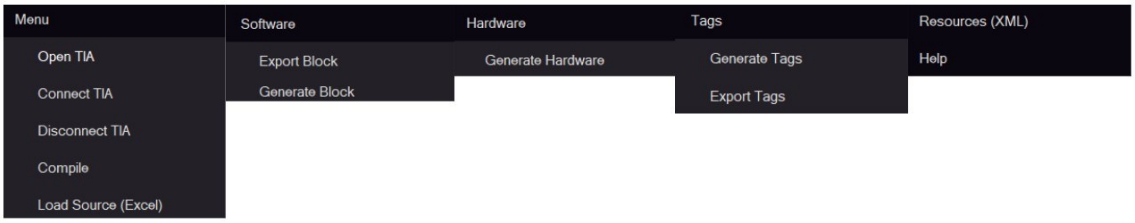
Layout-näkymän järjestely oli ensimmäinen vaihe uuden projektin luomisen jälkeen Visual Studiossa. Ohjelman ulkoasun ja valikoiden rakentamisessa otettiin

käyttökokemus (UX) huomioon, ja suunnittelu perustui minimalistisiin periaatteisiin. Visuaalisessa ulkoasussa tavoitteena oli noudattaa 60–30–10 väriskaalaa (kuva 38). Ohjelman selkeys ja helppokäyttöisyys olivat pääperiaatteita.



Kuva 38. Sovelluksen ulkoasu kehitysvaiheessa.

Toiminnot sijoitettiin alavalikoihin, jotka avautuvat päävalikon painalluksella (kuva 39). Lisäksi tietyt toiminnot laukaisivat alaikkunoita tarjoten esimerkiksi lisätietoa tuotavan lohkon tyypistä tai nimestä tai mahdollistivat tallennus- tai lähdetiedoston sijainnin valinnan. Tämä lähestymistapa pyrki vähentämään käyttäjän kognitiivista kuormitusta ja parantamaan käyttäjäkokemusta.



Kuva 39. Valikot ja niiden alavalikot avattuina kehitysvaiheessa.

6.5 Rajapinta ja Toiminnallisuus

Ohjelmistokehityksen seuraavana vaiheena oli ulkoasun toteutuksen jälkeen lisätä sovellukseen toiminnallisuudet. Sovelluksen kehitysvaiheessa yhdeksi suurimmaksi haasteeksi muodostui oikeiden komentojen etsiminen haluttuihin toimintoihin rajapinnan dokumentaatiosta. Lisäksi pohdittiin, miten varmistetaan ohjelmiston toimivuus riippumatta siitä, mille koneelle se on asennettu, sekä millä TIA Portal -versiolla ohjelman tulisi toimia.

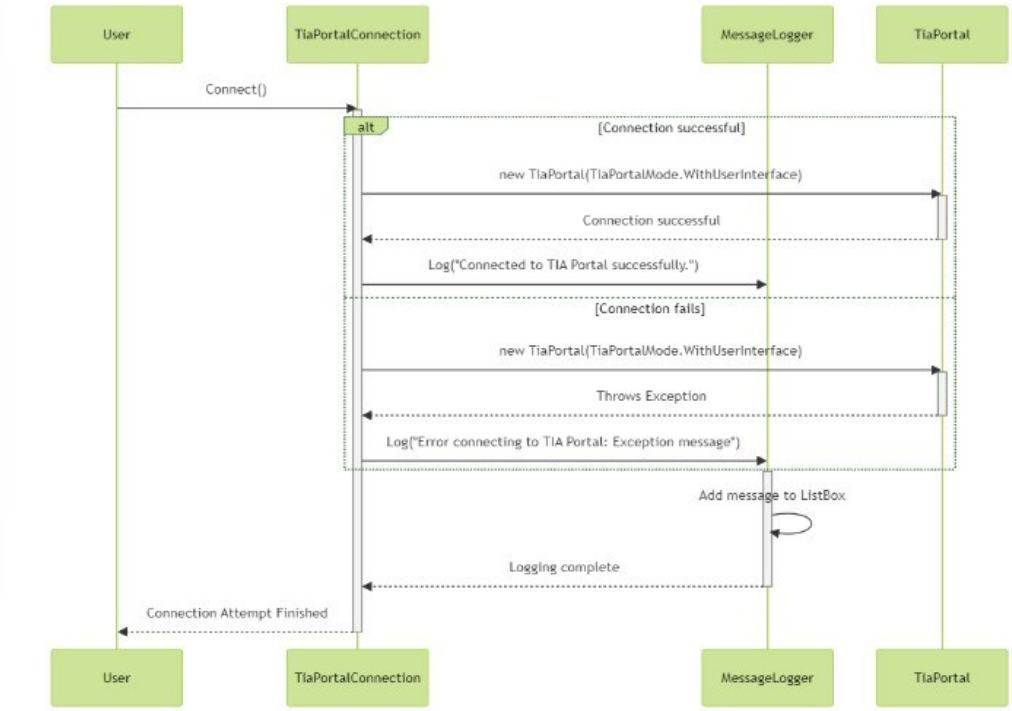
Versio V17 valittiin pääasialliseksi käyttöympäristöksi, sillä työkalua suunniteltiin käytettäväksi tuotannossa kyseisellä versiolla tehtyihin projekteihin. Rajapinnan dokumentaatiosta löydettiin lopulta valitulle versiolle soveltuvat komennot, ja dokumentaatiosta kävi ilmi, että rajapinnan .dll -kirjastoihin voidaan viitata useammalla eri menetelmällä. Tämä mahdollisti ratkaisun, jossa yhteensopivuus on taattu aina versioon V20 asti.

6.6 Ohjelmointi

Ohjelman rakenne muodostui pääohjelmasta ja useista aliohjelmista. Aliohjelmat eli luokat sisälsivät objektien rakenteet ja toiminnot. Kutsumalla luokkia eri ohjelman osissa ja luomalla niistä olioita ohjelman rakenne säilyi selkeänä sekä modulaarisena.

Esimerkiksi käyttäjän tekemä jokainen toiminto tulosti tapahtumalokiin jonkinlaisen ilmoituksen (kuva 40). Oliopohjaisella rakenteella koodi on tiivistetty yhteen

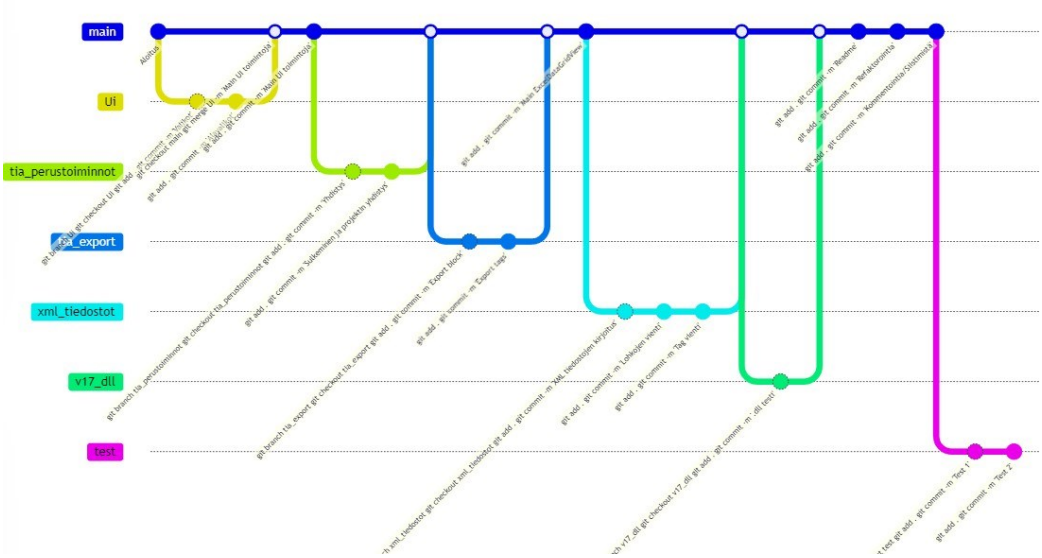
riviin, kun taas ilman oliopohjaista rakennetta vastaava toiminto olisi noin 30 riviä koodia toimintoa kohden.



Kuva 40. Sekvenssikaavio, joka kuvaa käyttäjän yhteydenottoyritystä TIA Portal -sovellukseen ja siihen liittyvien tapahtumien kirjausta.

Ohjelmistokehyksenä toimi Windows Forms, joka mahdollisti tehokkaan käyttöliittymän rakentamisen valmiilla työkalupakilla. Työkalupakista oli mahdollista vetää objekteja käyttöliittymään drag & drop -menetelmällä ja nopeasti määrittää näiden objektien toiminnot.

Uusien ominaisuuksien rakentamisen yhteydessä projektiin luotiin usein uusi haara, jolloin ohjelmasta säilyi toimiva versio, johon oli helppo palata tarvittaessa. Kun uudet ominaisuudet oli testattu toimiviksi, haara yhdistettiin takaisin päärepositorioon.

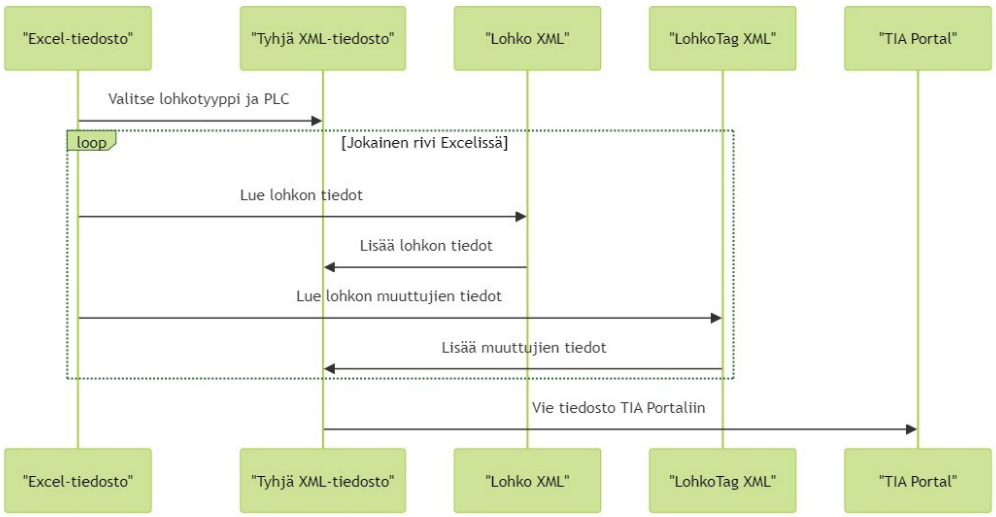


Kuva 41. Git-kaavio projektin versionhallinnasta kehitysvaiheessa.

6.7 Lohkojen generointi

Työläin vaihe ohjelmistokehityksessä oli itse lohkojen vienti takaisin TIA Portal -kehitysympäristöön. Ongelmaa koitettiin lähestyä ratkaisulla, jossa lohkoja luotaisiin suoraan kirjastosta. Tässä olisi ollut etuna, että yhteys kirjaston kanssa olisi säilynyt. Hyvin nopeasti kävi ilmi, ettei monimutkaisia ohjelmalohkoja ole mahdollista luoda tällä metodilla vaan ainoaksi ratkaisuksi jäi xml-tiedostojen generointi.

Yksittäisen lohkon takaisin vienti oli erittäin simppeliä, mutta useamman lohkon vienti järkevästi toteutettuna vaati monimutkaisempia toimenpiteitä. Ensimmäiseksi piti olla mallipohja tyhjälle ohjelmointilohkolle xml-tiedostona. Tämän tiedoston sisälle tuli kirjoittaa toisen xml-tiedoston sisältö mikä sisälsi halutun lohkon tiedot. Nämä tuli kirjoittaa niin monesti kuin Excel-tiedostossa oli määritelty. Tämän lisäksi lohkoille piti pystyä luomaan multi-instance datalohko tiedon tallennusta varten ja asettamaan oikeat muuttujat Excelin pohjalta myös XML-tiedostoon (kuva 42).

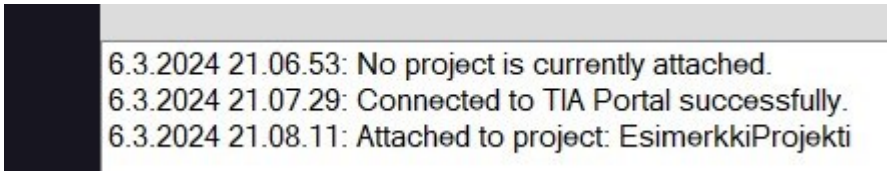


Kuva 42 Sekvenssikaavio lohkojen generoinnista.

Ylläkuvattu prosessi piti toistaa eri malliselle tyhjälle ohjelmointilohkolle, jos ohjelmointikieli tai ohjelmointilohkon tyyppi poikkesi aikaisemmasta prosessista.

6.8 Testaus

Uuden toiminnon lisäyksen jälkeen ohjelma rakennettiin Visual Studiossa ja käynnistettiin debug-tilassa, jolloin toiminnot testattiin TIA Portal -ohjelmointiympäristön kanssa. Sovelluksen käyttöliittymässä oleva tapahtumaloki (kuva 43) mahdollisti ominaisuuksien toimivuuden seuraamisen virhe- ja statusviestien avulla.

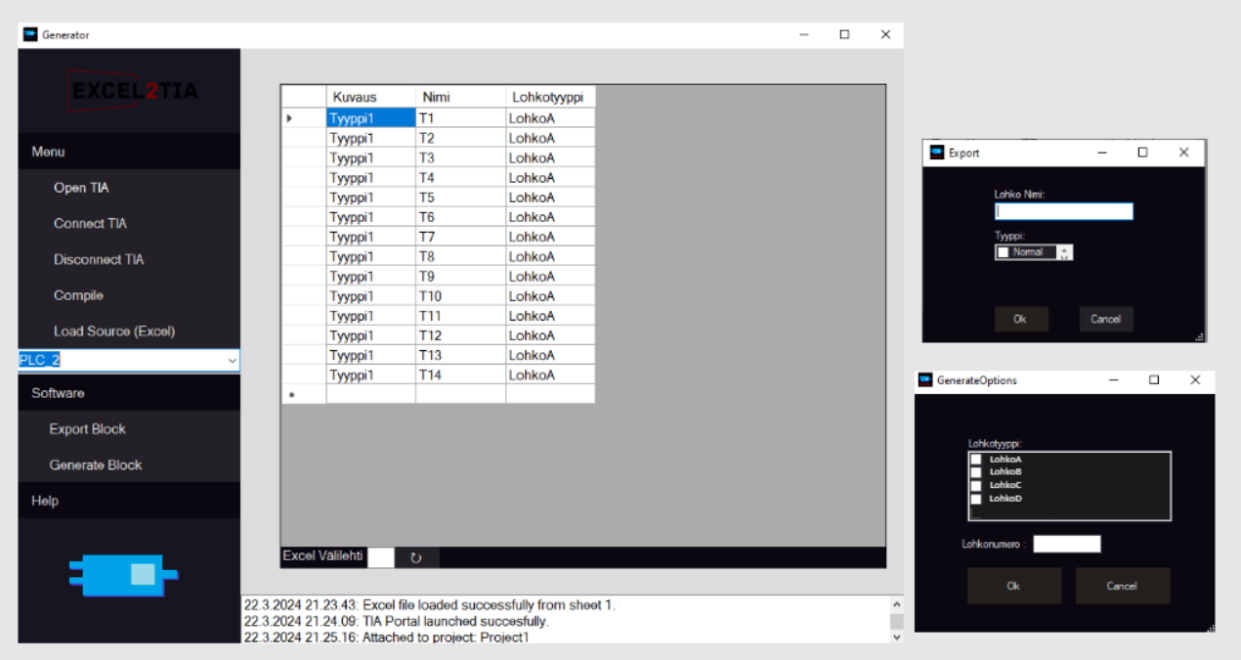


Kuva 43. Sovelluksen tapahtumaloki.

Haasteena oli, että lopullinen tuotantoympäristö sijaitsi virtuaalikoneella, mikä edellytti sovelluksen julkaisua testausta varten, koska Visual Studio ei ollut saatavilla samassa ympäristössä. Kehitystä nopeuttaakseen käytössä oli toinen laite, jossa oli paikallisesti asennettu sovellusversio V16. Kun kehitystyö oli edennyt tarpeeksi pitkälle ja siirryttiin testaamaan tuotantoympäristössä käyttäen sovellusversiota V17, sovellus jouduttiin aina julkaisemaan ja asentamaan se virtuaalikoneella. Tämä oli huomattavasti hitaampi testausmenetelmä kuin aikaisemmin käytetty.

6.9 Julkaisu

Lopputestausvaiheessa ohjelmistosta tehtiin useita kymmeniä julkaisuja, ja kun ensimmäiset generointiominaisuudet sekä sovelluksen toimintaan liittyvät yleiset toiminnot olivat valmiita, ensimmäinen demo julkaistiin toimeksiantajalle testausta ja kommentointia varten. Lopullinen julkaisu tehtiin, kun sovelluksen toiminnallisuus oli saavuttanut aloituspalaverissa määritellyn tason. Julkaisuversiosta karsittiin pois kaikki ylimääräiset kehitysvaiheen toiminnot. Jäljelle jäivät vain tarpeelliset toiminnot lohkojen generointia ja tuontia varten (liite 3).



Kuva 44. Sovelluksen lopullinen ulkoasu.

6.10 Käyttöohjeet

Lopulliselle sovellukselle laadittiin PDF-muotoinen ohjekirja. Ohjekirjassa käsiteltiin ohjelman toiminnallisuutta, laitteistovaatimuksia sekä TIA Openness -rajapintaan liittyvien Windows-asetusten tarvittavia konfiguraatioita. Ohjeet toimitettiin osana lopullista julkaisua yhdessä muiden materiaalien, kuten lähdekoodin, kanssa. Käyttöliittymässä oli lisäksi ominaisuus, joka mahdollisti käyttöohjeiden avaamisen suoraan sovelluksesta.

Kuva 45. Ohjekirjan sisällysluettelo.

7 Pohdinta

7.1 Tulokset

Toimeksiantajan näkökulmasta lopputulos on sovellus, joka vastaa aloituspalaverissa määriteltyjä toimintoja ja tarjoaa tietoa rajapinnan mahdollisuuksista sekä rajoituksista. Lisäksi työkalu tarjoaa merkittävän ajallisen edun manuaaliseen työhön verrattuna kyseisten lohkojen luomisprosessissa.

Henkilökohtaisen osaamisen ohjelmoinnin ja automaatiosuunnittelun alueilla havaittiin kasvaneen huomattavasti. Insinöörityön sekä sovelluskehityksen osalta voidaan todeta, että ketterän kehityksen periaatteita pystyttiin soveltamaan laajasti, ja ne edesauttoivat hyvän lopputuloksen saavuttamisessa.

7.2 Teoreettinen hyöty

Sovelluksen hyötyä voidaan arvioida laskemalla sen teoreettinen hyöty vapautuneella ajalla. Vapautunut aika muihin työtehtäviin voidaan laskea manuaalisen työn ja automatisoidun prosessin erotuksella. (kaava 1).

$$Vapautunut\ aika = Manuaalinen\ työ - Automatisoitu\ työ \quad (1)$$

Laskelmissa (liite 4) on käytetty manuaalisen työn arvioitua kestoa lohkojen luonnissa, mukaan lukien ohjelmointivirheistä johtuvat viivästykset. Arvioimalla sovelluksen kehitykseen ja ylläpitoon menevää aikaa sekä lohkojen generoinnin nopeutta verrattuna manuaaliseen työhön, voidaan määrittää ajallinen hyöty työajan vapautuminen muuhun käyttöön. Vaikka tilanne on täysin teoreettinen, se tuo hyvin esille työkalun potentiaalin, jos sitä käytetään laajasti.

7.3 Jatkokehitys

Rajapinta tarjoaa monia jatkokehitykseen soveltuvia ominaisuuksia kuten ennalta määritellyn tiedon pohjalta HMI-Paneelin (WinCC) generoimisen tai vaikka taajuusmuuttajien parametroinnin ja generoinnin projektiin.

Sovellukseen olisi mahdollista ehkä lisätä toiminto, joka muuntaisi XML-tiedostot Siemensin käyttämästä skeemasta standardin IEC-61131-10 mukaiseen XML-skeemaan. Kokonaisten projektien tai lohkojen siirtäminen toisen valmistajan ohjelmointiympäristöön voitaisiin mahdollistaa vaivattomasti. Eri valmistajan laitteiston käyttöönotto, esimerkiksi komponenttien saatavuusongelmien vuoksi, olisi tämän ansiosta nopeaa siirryttäessä toiselle kehitysalustalle.

Nykyisten XML-tiedostojen generointiominaisuuksien mahdollinen jatkokehityskohde olisi muokata toiminnosta modulaarisempi. Esimerkiksi käyttäjä voisi valita suoraan TIA Portalista minkä tahansa lohkon. Kyseinen lohko tuotaisiin suoraan muokkausta varten ja vietäisiin takaisin ilman manuaalista tiedoston käsittelyä. Modulaarisempi lähestymistapa generointiin vaatisi hyvin tarkat raamit lohkojen nimeämiselle ja lohkojen rakenteille, jotta tämä olisi mahdollista. Muita toteutuskelpoisia kehityskohteita on listattuna liitteessä kaksi ehdollisina toimintoina.

Lähteet

1 Insta Group. Tietoa meistä. Verkkoaineisto. https://www.insta.fi/tietoa-meista. Luettu 8.2.2024.

2 Roy S. 2022. Agile Development Methodologies: An Essential Guide. Verkkoaineisto. Browser Stack.https://www.browserstack.com/guide/agile-development-methodologies. 11.11.2022. Luettu 20.02.2024.

3 Beck, K. et al. 2001. Ketterän ohjelmistokehityksen manifesti. Verkkoaineisto. Agile Manifesto.https://agilemanifesto.org/iso/fi/manifesto.html. Luettu 20.02.2024

4 Lehtonen T, et al. 2014. Sulautettujen järjestelmien ketterä käsikirja. Verkkoaineisto. University of Turku.https://www.utupub.fi/handle/10024/99142. 30.09.2014. Luettu 20.02.2024.

5 GfG. 2019. "Dynamic Systems Development Method (DSDM)." Verkkoaineisto. GeeksforGeeks.https://www.geeksforgeeks.org/dynamic-systems-development-method-dsdm/. 18.07.2019. Luettu 19.02.2024.

6 Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. 2002. "Agile Software Development Methods: Review and Analysis." VTT Publications, nro 478, s. 61–64. Verkkoaineisto.https://www.vttresearch.com/sites/default/files/pdf/publications/2002/P478.pdf. Luettu 22.02.2024

7 Stapleton, J. 1997. Dynamic Systems Development Method – The Method in Practice. Addison Wesley.

8 Atlassian. "What is Jira Software?" Atlassian. Verkkoaineisto.https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software. Luettu 25.02.2024

9 Bolton W. Programmable Logic Controllers. 6th ed. Kidlington, Oxford, UK; Waltham, MA: Newnes; 2015 International Electrotechnical Commission, IEC 61131: Programmable controllers.

10 International Electrotechnical Commission, IEC 61131: Programmable controllers.

11 Siemens. 2021. "STEP 7 and WinCC Engineering V17 System Manual" Siemens. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109798671/simatic-step-7-basic-professional-v17-and-simatic-wincc-v17?dti=0&lc=en-CZ05.2021. Luettu 15.01.2024.

12 Siemens. 2021. "TIA Portal V17: Introduction - Your gateway to automation in the Digital Enterprise." Siemens. Verkkoaineisto.https://assets.new.siemens.com/siemens/assets/api/uuid:806dba5d-ef7d-465c-a033-1cb1c4890609/v17-launch-webinar-tia-portal-v17-introduction-1.pdf. Luettu 15.01.2024

13 Siemens. 2017. “Digitalization with TIA Portal: Integration of planning data from TIA Selection Tool to TIA Portal or EPLAN Electric P8” Siemens. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109748223/digitalization-with-tia-portal-integration-of-planning-data-from-tia-selection-tool-to-tia-portal?dti=0&lc=en-FI. 11.07.2017 Luettu 21.01.2024

14 John KH, Tiegelkamp M. IEC 61131–3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools. Springer Science & Business Media; 2013

15 Siemens. 2023. “Programming Guidelines for S7-1200 and S7-1500” Siemens. Verkkoaineisto.https://support.industry.siemens.com/cs/ww/en/view/81318674. 03.08.2024 Luettu 02.02.2024

16 Siemens. 2013. “Which organization blocks can you use in STEP 7 (TIA Portal)?” Siemens. Verkkoaineisto.https://support.industry.siemens.com/cs/document/40654862/which-organization-blocks-can-you-use-in-step-7-(tia-portal)-?dti=0&lc=en-FI. 29.04.2013 Luettu 04.03.2024

17 Siemens. 2013. “Configuring Technology Objects with SIMATIC S7 1500 and SINAMICS S210 in TIA-Portal” Siemens. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109749795/configuring-technology-objects-with-simatic-s7-1500-and-sinamics-s210-in-tia-portal?dti=0&lc=en-FI.Luettu 04.03.2024

18 Siemens. Standards Compliance according to IEC 61131-3. Verkkoaineisto.https://cache.industry.siemens.com/dl/files/932/8790932/att_82258/v1/norm_tbl.pdf. Luettu 12.2.2024

19 International Electrotechnical Commission. 2013. IEC 61131-3: Programmable controllers - Part 3: Programming languages. Geneva: IEC.

20 Hanssen, D. H. 2015. Programmable Logic Controllers: A Practical Approach to IEC 61131-3 using CoDeSys. Part Four: Programming.

21 PR Electronics. 4...20 mA:n virtapiirien perusteet. Verkkoaineisto.https://www.prelectronics.com/fi/4-20-man-virtapiirien-perusteet/. Luettu 04.03.2024

22 Fieldbus inc. IEC61158 Technology Comparison: State of the Bus. Verkkoaineistohttps://www.fieldbusinc.com/downloads/fieldbus_comparison.pdf. Luettu 04.03.2024

23 Pyyskänen, Seppo. 2013. Teollisuuden automaatio- ja ohjausjärjestelmät: Standardien valinta ja käyttö. Suomen Automaatioseura ry. Verkkoaineisto.https://www.automaatioseura.fi/site/assets/files/1426/standardikirja.pdf. Luettu 04.03.2024

24 Zurawski, Richard. 2017. Industrial Communication Technology Handbook, 2nd Edition. Kappale 1: Fieldbus System Fundamentals.

25 Anderson, Mondi. 2019. RealPars. Fielbus. Verkkoaineisto.https://www.realpars.com/blog/fieldbus. Luettu 05.03.2024

26 Felser, Max. 2017, PROFIBUS Manual: The token passing principle. Verkkoaineisto.https://www.felser.ch/profibus-manual/prinzip_des_token-passing.html. Luettu 05.03.2024

27 Carlsson, Thomas. 2023. Industrial network market shares 2023. Verkkoaineisto.https://www.hms-networks.com/news-and-insights/news-from-hms/2023/05/05/industrial-network-market-shares-2023. 05.05.2024. Luettu 05.03.2024

28 Siemens. 2016. SIMATIC S7-1500 / ET 200MP Automation system: In a Nutshell. Verkkoaineisto.https://files.realpars.com/siemens/manuals/s71500_et200mp_in_a_nutshell_en-US.pdf. Luettu 05.04.2024

29 SFS-EN ISO 13849-1 Koneturvallisuus. Turvallisuuteen liittyvät ohjausjärjestelmien osat. Osa 1: Yleiset suunnitteluperiaatteet.

30 Siemens. 2023. SIMATIC Safety - Configuring and Programming: Programming and Operating Manual. Verkkoaineisto.https://support.industry.siemens.com/cs/document/54110126/simatic-industrial-software-simatic-safety-configuring-and-programming?dti=0&lc=en-FI. 27.10.2024. Luettu 05.03.2024

31 Siemens. 2021.TIA Portal Openness: Automation of engineering workflows. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109792902/tia-portal-openness-automation-of-engineering-workflows?dti=0&lc=en-US. Luettu 05.03.2024

32 Siemens. 2023 TIA Portal Openness: Introduction and Demo Application. Verkkoaineisto.https://support.industry.siemens.com/cs/document/108716692/tia-portal-openness-introduction-and-demo-application?dti=0&lc=en-US. 01.03.2023. Luettu 05.03.2024

33 Siemens. 2022. SIMATIC TIA Portal Openness: API for automation of engineering workflows. Verkkoaineisto.https://support.industry.siemens.com/cs/mdm/109815199?c=163468723851&lc=en-FI. 05.12.2022. Luettu 05.03.2024

34 Smith, Patrick. DMC. Advantages & Disadvantages of Siemens' TIA Openness. Verkkoaineisto.https://www.dmcinfo.com/latest-thinking/blog/id/9877/advantages-disadvantages-of-siemens-tia-openness. 13.05.2019. Luettu 05.03.2024

35 Microsoft. XML:n perusteet. Verkkoaineisto.https://support.microsoft.com/fi-fi/office/xml-n-perusteet-a87d234d-4c2e-4409-9cbc-45e4eb857d44. Luettu 05.03.2024

36 W3Schools. Introduction to XML. Verkkoaineisto.https://www.w3schools.com/xml/xml_whatis.asp. Luettu 05.03.2024

37 Aarnio, Pekka. 2019. ELEC-C8203 - Automaatiojärjestelmät 2: XML-Merkintäkieli luento. Aalto Yliopisto.

38 Aarnio, Pekka. 2019. ELEC-C8203 - Automaatiojärjestelmät 2: XML Schema luento. Aalto Yliopisto.

39 IEC 61131-10:2019. Programmable controllers - Part 10: PLC Open XML exchange format.

40 PLC Open. 2009. XML Formats for IEC 61131-3. PLCopen XML Technical Document, (PDF).

41 Warren, Genevieve et al. 2024. What is Visual Studio? Microsoft. Verkkoaineisto.https://learn.microsoft.com/fi-fi/visualstudio/get-started/visual-studio-ide?view=vs-2022. Luettu 05.03.2024

42 Github. About Git. Verkkoaineisto.https://docs.github.com/en/get-started/using-git/about-git. Luettu 05.03.2024

43 Uotila, Terho. Tietokone työvälineenä - Lapio. Helsingin yliopisto. Verkkoaineisto.https://tkt-lapio.github.io/git/. Luettu 05.03.2024

44 Wagner, Bill et al. 2023 A Tour of the C# Language. Microsoft. Verkkoainesto.https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. Luettu 05.03.2024

45 Warren, Genevieve et al. 2024. Introduction to .NET. Microsoft. Verkkoaineisto.https://learn.microsoft.com/en-us/dotnet/core/introduction. Luettu 05.03.2024

46 Stone, Madeline. 2015. A Brief History of Steve Ballmer's Epic Freak-Outs. Business Insider. Verkkoaineisto.https://www.businessinsider.com/steve-ballmers-epic-freak-outs-2015-5?r=US&IR=T. Luettu 05.03.2024

47 De George, Andy. 2023. Windows Forms Overview. Microsoft. Verkkoaineisto.https://learn.microsoft.com/en-us/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8. Luettu 05.03.2024

48 Liang, Han. 2024. What is a DLL. Microsoft. Verkkoaineisto.https://learn.microsoft.com/en-us/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library. Luettu 05.03.2024

49 Wagner, Bill et al. 2023. Object-Oriented Programming (C#). Microsoft. Verkkoaineisto. https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop . Luettu 05.03.2024

50 GeeksforGeeks. 2024. Difference Between UI and UX Design. Verkkoaineisto.https://www.geeksforgeeks.org/difference-between-ui-and-ux-design/. Luettu 05.03.2024

51 Miller, G. A. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, 63(2), 81–97.

52 Hick, W. E. (1952). On the rate of gain of information. Quarterly Journal of Experimental Psychology, 4(1), 11–26.

53 Lopez, Judith. 2023. The 60–30–10 Rule: A Foolproof Way to Choose Colors for Your UI Design. UX Planet. Verkkoaineisto.https://uxplanet.org/the-60-30-10-rule-a-foolproof-way-to-choose-colors-for-your-ui-design-d15625e56d25. Luettu 05.03.2024

54 Siemens. 2021. Excel code generator for TIA Portal Openness. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109770550/excel-code-generator-for-tia-portal-openness?dti=0&lc=en-FI. Luettu 05.03.2024

55 Siemens. 2023. TIA Portal Openness: Introduction and Demo Application. Verkkoaineisto.https://support.industry.siemens.com/cs/document/108716692/tia-portal-openness-introduction-and-demo-application?dti=0&lc=en-US. Luettu 05.03.2024

56 Siemens. 2023. Openness Scripter: Introduction. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109742322/tool-for-easier-use-of-the-tia-portal-openness-interface-(openness-scripter)?dti=0&lc=en-FI. Luettu 05.03.2024

57 Siemens. 2023. Openness Scripter: Detailed Documentation. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109742322/tool-for-easier-use-of-the-tia-portal-openness-interface-(openness-scripter)?dti=0&lc=en-FI. Luettu 05.03.2024

58 Siemens. 2021. TIA Portal Openness Explorer. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109760816/tia-portal-openness-explorer?dti=0&lc=en-KW. Luettu 05.03.2024

59 Siemens. 2024. SIMATIC Modular Application Creator. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109762852/simatic-modular-application-creator?dti=0&lc=en-FI. Luettu 05.03.2024

60 Siemens. 2022. Getting Started of Modular Application Creator. Verkkoaineisto.https://support.industry.siemens.com/cs/document/109762852/simatic-modular-application-creator?dti=0&lc=en-FI. Luettu 05.03.2024

61 Siemens. Three weeks of programming in only half a day using Modular Application Builder. Verkkoaneisto.https://www.siemens.com/global/en/company/stories/industry/factory-automation/modular-application-creator-ab-electric.html. Luettu 05.03.2024

Datatyypit Step7

Table: Binary numbers

Binary numbers	S7-300/400	S7-1200	S7-1500
BOOL	X	X	X
Bit strings			
BYTE	X	X	X
WORD	X	X	X
DWORD	X	X	X
LWORD	-	-	X

Table: Integers

Integers	S7-300/400	S7-1200	S7-1500
SINT	-	X	X
INT	X	X	X
DINT	X	X	X
USINT	-	X	X
UINT	-	X	X
UDINT	-	X	X
LINT	-	-	X
ULINT	-	-	X

Table: Floating-point numbers

Floating-point numbers	S7-300/400	S7-1200	S7-1500
REAL	X	X	X
LREAL	-	X	X

Table: Timers

Timers	S7-300/400	S7-1200	S7-1500
S5TIME	X	-	X
TIME	X	X	X
LTIME	-	-	X

Table: Date and time

Date and time	S7-300/400	S7-1200	S7-1500
DATE	X	X	X

TIME_OF_DAY (TOD)	X	X	X
LTOD (LTIME_OF_DAY)	-	-	X
DT (DATE_AND_TIME)	X	-	X
LDT	-	-	X
DTL	-	X	X

Table: Character strings

Character strings	**S7-300/400**	**S7-1200**	**S7-1500**
CHAR	X	X	X
WCHAR	-	X	X
STRING	X	X	X
WSTRING	-	X	X

Table: PLC data types (UDT)

PLC data types (UDT)	**S7-300/400**	**S7-1200**	**S7-1500**
PLC data type (UDT)	X	X	X

Table: Anonymous structures

Anonymous structures	**S7-300/400**	**S7-1200**	**S7-1500**
STRUCT	X	X	X

Table: ARRAY

ARRAY	**S7-300/400**	**S7-1200**	**S7-1500**
ARRAY [....] of <data type>	X	X	X

Table: Pointer

Pointer	**S7-300/400**	**S7-1200**	**S7-1500**
References	-	-	X
VARIANT	-	X	X
POINTER	X	-	X
ANY	X	-	X

Table: Parameter types

Parameter types	**S7-300/400**	**S7-1200**	**S7-1500**

TIMER	X	-	X
COUNTER	X	-	X
BLOCK_FC	X	-	X
BLOCK_FB	X	-	X
BLOCK_DB	X	-	-
BLOCK_SDB	X	-	-
VOID	X	X	X
PARAMETER	-	X	X

Table: System data types

System data types	**S7-300/400**	**S7-1200**	**S7-1500**
IEC_TIMER	X[1)]	X	X
IEC_LTIMER	-	-	X
IEC_SCOUNTER	-	X	X
IEC_USCOUNTER	-	X	X
IEC_COUNTER	X[2)]	X	X
IEC_UCOUNTER	-	X	X
IEC_DCOUNTER	-	X	X
IEC_UDCOUNTER	-	X	X
IEC_LCOUNTER	-	-	X
IEC_ULCOUNTER	-	-	X
ERROR_STRUCT	-	X	X
NREF	-	X	X
CREF	-	X	X
VREF	-	X	X
SSL_HEADER	X	-	-
CONDITIONS	-	X	-
TADDR_Param	-	X	X
TCON_Param	-	X	X
HSC_Period	-	X	-
AssocValues	-	X	X

[1)] For S7-300/400 CPUs, the data type is represented by TP, TON and TOF.

[2)] For S7-300/400 CPUs, the data type is represented by CTU, CTD and CTUD.

Table: Hardware data types

Hardware data types	**S7-300/400**	**S7-1200**	**S7-1500**
REMOTE	-	X	X
HW_ANY	-	X	X

HW_DEVICE	-	X	X
HW_DPMASTER	-	-	X
HW_DPSLAVE	-	X	X
HW_IO	-	X	X
HW_IOSYSTEM	-	X	X
HW_SUBMODULE	-	X	X
HW_MODULE	-	-	X
HW_INTERFACE	-	X	X
HW_IEPORT	-	X	X
HW_HSC	-	X	X
HW_PWM	-	X	X
HW_PTO	-	X	X
EVENT_ANY	-	X	X
EVENT_ATT	-	X	X
EVENT_HWINT	-	X	X
OB_ANY	-	X	X
OB_DELAY	-	X	X
OB_TOD	-	X	X
OB_CYCLIC	-	X	X
OB_ATT	-	X	X
OB_PCYCLE	-	X	X
OB_HWINT	-	X	X
OB_DIAG	-	X	X
OB_TIMEERROR	-	X	X
OB_STARTUP	-	X	X
PORT	-	X	X
RTM	-	X	X
PIP	-	-	X
CONN_ANY	-	X	X
CONN_PRG	-	X	X
CONN_OUC	-	X	X
CONN_R_ID	-	-	X
DB_ANY	-	X	X
DB_WWW	-	X	X
DB_DYN	-	X	X

Sovelluksen alustavat toiminnot

Kategoria	Alakategoria	Välttämätön	Ehdollinen
Tagit	Tuo		x
	Vie		x
Hardware	Tuo		x
	Vie		x
	IP-asetukset		x
	Network luonti		x
	Muu konfigurointi		x
Software			
-Safety			
	LAD / FBD		x
	DATABASE		x
-Normal			
	LAD / FBD	x	
	SCL		x
	DATABASE	x	
	Technology object	x	
-Kirjastot			
	Tuonti		x
	Vienti		x
Yleiset toiminnot	Käynnistys	x	
	Yhdistys	x	
	Yhteyden katkaisu	x	
	Projektin luonti		x
	Projektin tallennus		x
	Compile	x	
	Projektipuun visualisointi		x
	.dll-kirjastoihin viittaaminen	x	
	Excel tiedostojen luku	x	
	Excel tiedostojen visualisointi	x	
	XML-tiedostojen käsittely	x	

Sovelluksen toiminnot

Kategoria	Alakategoria
Generointi	
	Lohko A + iDB
	Lohko B + iDB
	Lohko C + iDB
	Lohko D + iDB + TO
Perustoiminnot API	
	Käynnistys
	Yhdistys
	Yhteyden katkaisu
	Compile
	PLC kohteen valinta
Sovellus	
	Lohkotyypin valinta
	Lohkon numerointi
	.dll-kirjastoihin viittaus
	Excel tiedostojen luku
	Excel tiedoston visualisointi
	XML-tiedoston käsittely
	Tapahtumaloki
	Käyttöohjeet
	Valikot ja alavalikot

Teoreettinen hyöty

Työkalun kehittämiseen kuluu kolme kuukautta ja sen ylläpitoon menee vuodessa 37,5 tuntia. Manuaalisena työhön kuluu 30 tuntia per projekti ja siitä aiheutuu vianselvitystä 4 h. Automatisoituna samaan työhön menee 10min.

- **Skenaario A**: Työkalua käyttää yksi insinööri, hänellä on neljä saman kokoista projektia vuodessa.
- **Skenaario B**: Työkalua käyttää kolme insinööriä, joilla on neljä saman kokoista projektia vuodessa.
- **Skenaario C**: Työkalua käyttää viisi insinööriä, jotka tekevät kaksi saman kokoista projektia vuodessa.

Ylläpito	37.50	h
Kehitysaika	162.38	h
Automatisoitu työ	0.17	h

Skenaario:	A	B	C
Manuaalinen työ (h)	30.00	90.00	150.00
Vianselvitys(h)	16.00	48.00	40.00
Projekti(kpl)	4.00	4.00	2.00
Insinööri(kpl)	1.00	3.00	5.00
Summa (h)	46.00	138.00	190.00
Vapautunut aika (h)	8.33	100.33	152.33