Tampereen yliopisto

Hermanni Korkeamäki

SIMULOINTI AUTOMAATIOSOVELLUKSEN TESTAUKSESSA

Tekniikan ja luonnontieteiden tiedekunta
Diplomityö
Maaliskuu 2019

TIIVISTELMÄ

Hermanni Korkeamäki: Simulointi automaatiosovelluksen testauksessa
Diplomityö
Tampereen yliopisto
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Maaliskuu 2019

Simulointisovellus testauksen apuvälineenä tehostaa ja selkeyttää automaatiosovelluksen testausvaiheita ja järjestelmän tehdastestausta. Automaatiojärjestelmien tehdastestauksissa sekä toimittajan että asiakkaan tulee olla ajan tasalla toteutettavista testeistä ja tapahtumista. Ulkoinen simulointisovellus selkeällä informatiivisella käyttöliittymällä antaa asiakkaalle huomattavasti selkeämmän kuvan tapahtumista, kuin se että testaaja simuloi rajapintojen informaatiota suoraan ohjelmoitavalle logiikalle.

Tämän diplomityön tavoitteena on luoda laitteistoriippumaton simulointisovellus automaatiojärjestelmän I/O-rajapinnan muuttujien simulointiin. Laitteistoriippumattomuudella tarkoitetaan universaalia, abstraktia tiedonsiirtomenetelmää simulointisovelluksen ja logiikan välillä. Tämän lisäksi muita vaatimuksia ovat simulointisovelluksen helppokäyttöisyys, käyttöönotettavuus ja mukautuvuus, joiden avulla se on käytettävissä tehokkaasti koko automaatioprojektin elinkaaren ajan.

Työn teoriaosassa tutkitaan kirjallisuuteen pohjautuen metodeita ongelman ratkaisemiseksi. Aihealueita simuloinnin toteuttamiseen ovat: tehdasprosessien toiminta ja kuvaukset, automaatiojärjestelmän horisontaalinen ja vertikaalinen integraatio ISA-95 mallin mukaisesti, ohjelmoitavan logiikan toiminta laitteisto- ja sovellustasolla, laitteiden välinen tiedonsiirto automaatiojärjestelmissä, ohjelmistotuotannon menetelmät sovelluksen kehittämiseksi, ohjelmistotestauksen menetelmät ja tehdastestaus sekä testausten dokumentointi.

Työssä päädyttiin käyttämään OPC UA (OPC Unified Architecture) -standardin mukaista tiedonsiirtoa simulointisovelluksen ja logiikan välillä. Tämä tarjoaa mahdollisuuden laitteistoriippumattomaan tiedonsiirtoon palvelimen ja asiakkaan välillä, tarjoten abstraktin palvelupohjaisen mallin. Suunniteltu simulointiohjelmisto on Microsoft Excel OPC UA asiakas, joka rakentuu C#/.Net pinon päälle. Ohjelmisto suunniteltiin annetuista lähtötiedoista käyttäen ohjelmistotuotannon menetelmiä aina toteutusvaiheeseen saakka.

Toteutettu simulointiohjelmiston prototyyppi kykenee muodostamaan simulointiprojektin olemassa olevista automaatiosuunnittelun lähtötiedoista, jolloin simuloinnin käyttöönottaminen vaatii ainoastaan ohjelmoitavan logiikan OPC UA -rajapinnan käyttöönottamisen palvelimena ja simulointisovelluksen yhdistämisen palvelimeen. Menetelmällä onnistuttiin simuloimaan Siemens S7-1500 –sarjan logiikan I/O -rajapinnan muuttujia vaatimusten mukaisesti.

Avainsanat: automaatiojärjestelmä, simulointi, FAT, PLC, ohjelmistotestaus, OPC UA

ABSTRACT

Hermanni Korkeamäki: Simulation in automation software testing
Master's thesis
Tampere University
Master's Degree Programme in Automation Technology
March 2019

The simulation software as a testing tool improves and clarifies testing phases of an automation software and factory acceptance testing. In the factory acceptance testing of automation systems, both the supplier and the customer must be up to date on the tests and events. External I/O -interface simulation application provided by the supplier engineer with a clear informative user interface gives the customer a better picture of the events, unlike a simulation directly to the programmable logic controller I/O -variables.

The aim of this master's thesis is to create a hardware-independent simulation software for simulating the I/O -interface of the automation system. Hardware independence refers to a universal, abstract data transfer method between the simulation software and the programmable logic controller. Other requirements for simulation systems include user-friendly interface, initialization and adaptability, which make the simulation system available and efficient throughout the lifetime of the automation project.

In the theoretical part of this thesis, a literature review is executed to present different methods to solve the problem. The main topics on creating a simulation software are operation and descriptions of factory processes, horizontal and vertical integration of ISA-95 model, hardware and software level of programmable logic controllers (PLC), software production methods for developing simulation software, software testing, factory acceptance testing and documentation of testing.

Based on the literary review, the OPC UA (OCP Unified Architecture) communication standard was chosen for data transfer between the simulation software and PLC. This provides hardware-independent communication between the PLC -server and the simulation client, providing an abstract service-based model. The designed simulation software was a Microsoft Excel OPC UA client, built on the C # / .Net stack. The software was designed from the given requirements using software production methods.

The implemented prototype of the simulation software is capable of creating a simulation project from existing automation design data, where the usage of simulation requires only the hardware configuration of the PLC as OPC UA server and connecting the simulation client application to the server. The methods given in the literary review succeeded in simulating the I/O variables of the Siemens S7-1500 series logic controller according to the requirements.

Keywords: automation, simulation, FAT, PLC, software testing, OPC UA

ALKUSANAT

Tämä diplomityö on toteutettu yhteistyössä Insta Automation Oy:n kanssa.

Työn ohjaajana ja hyvänä neuvonantajana toimi Insta Automation Oy:n puolesta Arttu Hanhela ja TTY:n osalta professori Matti Vilkko.

Työn kotiohjauksesta, tukemisesta ja hyvinvoinnin ylläpitämisestä haluan kiittää rakasta, ymmärtäväistä ja avuliasta tyttöystävääni DI Jannika Paulamäkeä. Lisäksi haluan kiittää isääni suuresta tuesta ja kannustuksesta pitkän opintopolun loppuun saattamiseen sekä äitiäni, sisaruksiani ja muita läheisiä tuesta niin työssä kuin elämässä.

Diplomi-insinöörin tutkinnon suorittamisesta haluan kiittää myös teekkariystäviäni Elisaa, Jannea, Teppoa ja Tonia.

Tampereella, 13.3.2019

Hermanni Korkeamäki

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

AI	Analog Input, analogiatulo
API	Application Programming Interface. Ohjelmointirajapinta
AO	Analog Output, analogialähtö
CAPE	Computer Aided Production Engineering
COM	Component Object Model, Microsoftin ohjelmarajapinnan standardi
CPU	Central Prosessin Unit, logiikan keskusyksikkö
DCOM	Distributed Component Object Model, Microsoftin hajautettu ohjelmarajapinnan standardi
DCS	Distributed Control System, hajautettu automaatiojärjestelmä
DI	Digital input, digitaalitulo
DO	Digital Output, digitaalilähtö
ERP	Enterprise Resource Planning, Tominnanohjaus
FAT	Factory acceptance test, järjestelmän tehdastestaus
IEC	International Electrotechnical Comission, kansainvälinen sähköalan standardointiorganisaatio
I/O	Input Output, tulot ja lähdöt
IP	Internet Protocol, Internet-protokolla
ISA	International Society of Automation, kansainvälinen automaatiotekniikkaan keskittynyt järjestö
ISO	International Organization for Standardization, kansainvälinen standardointijärjestö
MES	Manufacturing Execution System, Tuotannonohjaus
OPC UA	OPC Unified Architecture.
OSI	Open Systems Interconnection, viitemalli avointen järjestelmien välisien yhteyksien määrittelemiseksi
PCS	Process control system, prosessin ohjausjärjestelmä
PLC	Programmable Logic Controller, ohjelmoitava logiikka
POU	Program Organization Unit, IEC 61131-1 rakenneyksikkö
RAM	Random Access Memory, prosessorin kekusmuisti
ROI	Return of invest
ROM	Read Only Memory, lukumuisti
SAT	Site acceptance test, järjestelmän käyttöönottotestaus
SCADA	Supervisory control and data acquisition, laitoksen ohjausjärjestelmä
UML	Unified Modeling Language

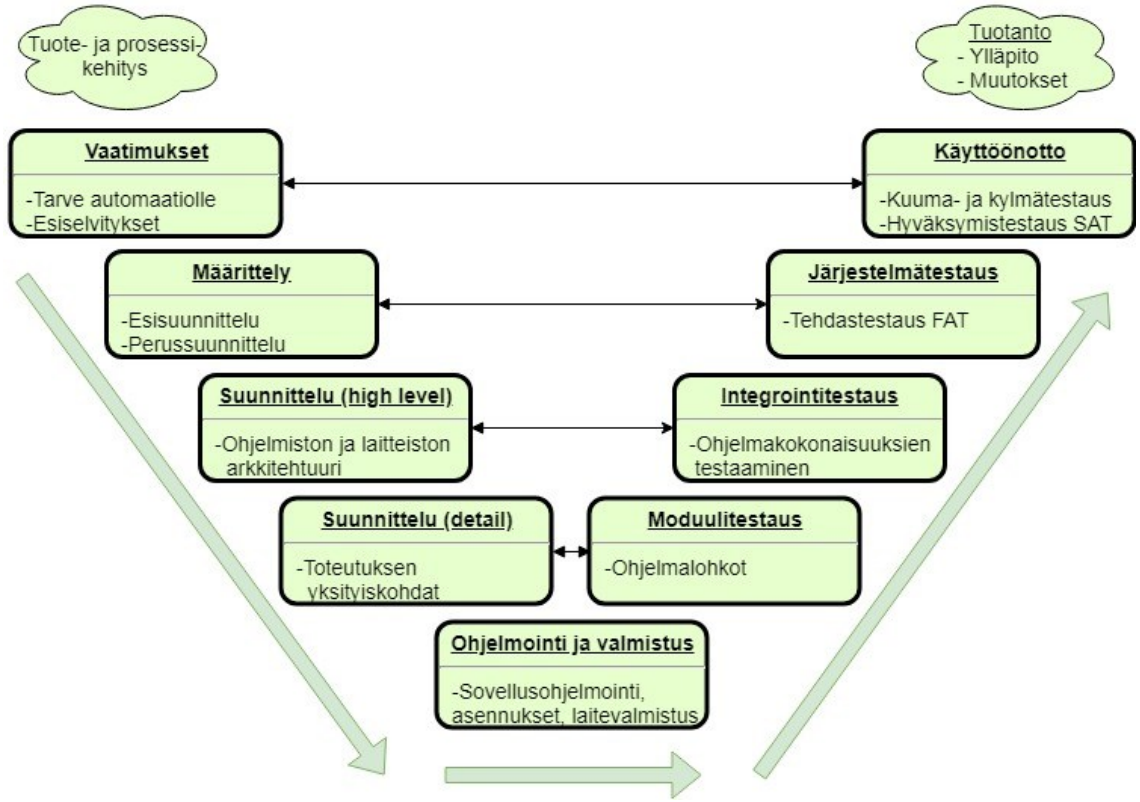
1. JOHDANTO

Automaatiosovelluksen toiminnallinen testaaminen vasta todellisessa käyttöprosessissa saattaa aiheuttaa vaaratilanteita ja on kustannuksiltaan tehotonta [1]. Sen sijaan, automaatiojärjestelmän suunnitelmallisella toteutusvaiheen kattavalla ohjelmistotestauksella pyritään varmistamaan automaatiosovelluksen toiminnallisuus ennen käyttöönottoa. Testaaminen käyttäen simulointia tehostaa ajallisesti ja laadullisesti automaatiosovelluksen toiminnallista testaamista sekä automaatiojärjestelmän tehdastestausta (Factory Acceptance Test, FAT).

1.1 Tausta ja ympäristö

Automaatiojärjestelmien toteutus koostuu määrittelystä, suunnittelusta, toteutuksesta, testauksesta ja käyttöönotosta. Testatun ja toimivaksi todetun järjestelmän toimitus sujuu tehokkaammin kuin sellaisen, jonka pelkästään oletetaan toimivan. Käyttöönoton aikana tehtävät testaukset ja virheiden korjaukset, jotka olisi pystytty tekemään toteutusvaiheessa, lisäävät kustannuksia ja mikäli ylimääräistä työtä ei ole aikataulutettu käyttöönottoon, sotkevat aikataulun.

Automaatioprojektin elinkaari voidaan esittää kuvan 1 mukaisen V-mallin avulla. Projektilla on aina alku, joka syntyy tuotannon ja prosessin tarpeista. Tarpeiden ja esiselvitysten myötä projektille muodostuvat vaatimukset. Määrittelyvaiheessa vaatimukset tarkentuvat toimintakuvauksilla ja käyttäjävaatimuksilla. Suunnitteluvaihe jakaantuu osiin alkaen laajasta järjestelmätason suunnittelusta jatkuen pienempiin paloiteltuihin ohjelmakokonaisuuksiin ja yksityiskohtiin. Kun projektin kuvaukset ja suunnitelmat ovat selvillä, alkaa toteutusvaihe paloiteltujen moduulien ohjelmoinnilla, laitevalmistuksella, asennuksilla ja lopulta moduulien yhteen liittämisellä. Projektin toteutusvaiheessa ohjelmistotestaukset sisältävät moduulitestauksia ja integrointitestauksia, johtaen koko järjestelmän kattavaan asiakkaan kanssa toteutettavaan tehdastestaukseen toimittajan tiloissa. Toteutusvaiheen jälkeen järjestelmä siirretään kohteeseen ja aloitetaan käyttöönottotestaukset sekä hyväksymistestaukset (Site Acceptance Test, SAT) ja luovutetaan projekti asiakkaalle. Luovuttamisen jälkeen järjestelmää ylläpidetään sen elinkaaren loppuun saakka. [2] Automaatioprojektin elinkaaren vaiheisiin liittyy myös erilaisia tukitoimintoja, kuten dokumentointi, laadunvarmistus, konfiguraatiohallinta ja vaatimustenhallinta. [3]



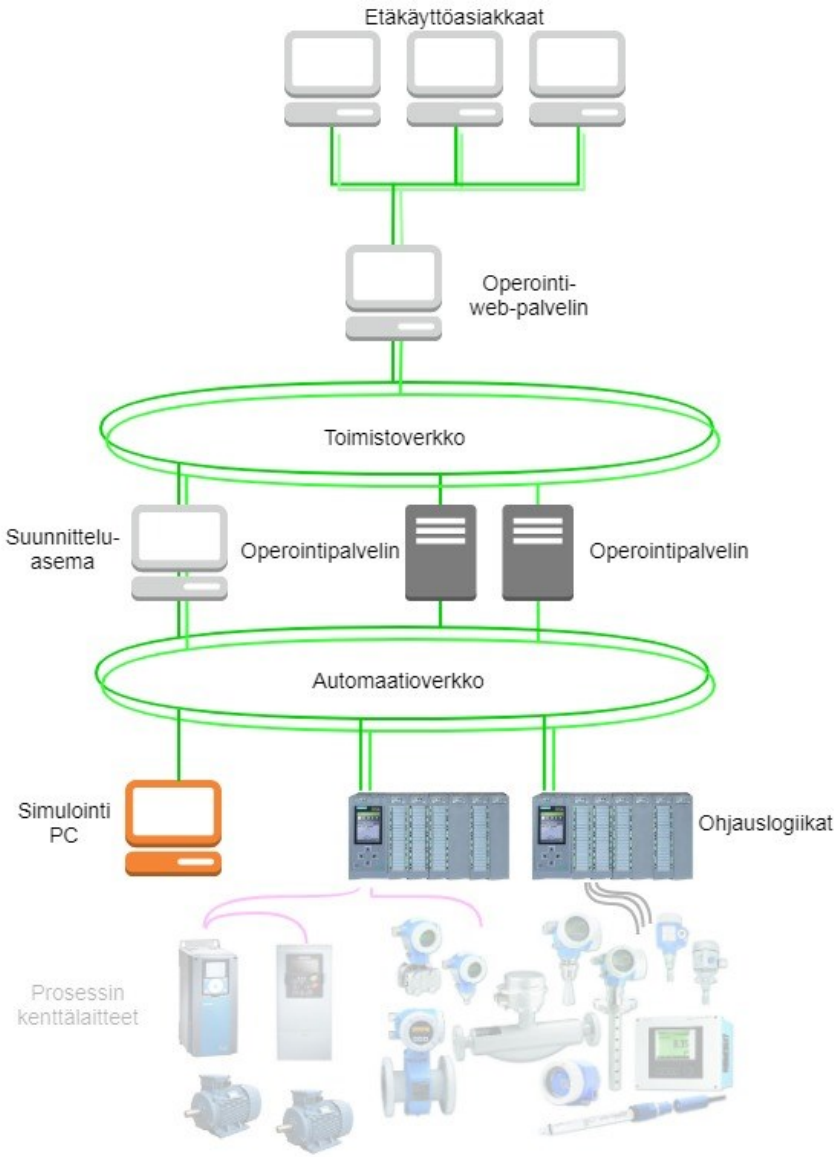
Kuva 1. *V-malli automaatiojärjestelmän toimitusprosessista. Muokattu [2].*

Automaatioprojektin elinkaaren aikana automaatiojärjestelmän komponenteille tehdään katselmuksia, tarkastuksia ja testejä projektin eri vaiheissa. Tarkastuksien tavoitteena on löytää virheet mahdollisimman varhaisessa vaiheessa, jotta seuraavan vaiheen alkaessa virheet eivät vaikuttaisi projektiin.

Projektin aikataulun toteutumista seurataan ennalta määrättyjen virstanpylväiden avulla, jolloin asiakkaalle tulee esittää tositteita toteutuksen toimivuudesta. Katselmusten ja tarkastusten dokumentointi mahdollistavat teknisen toteutuksen konkretisoimisen asiakkaalle esitettävään muotoon. [4] Lisäksi, ennalta määrätyt, yhteneväiset testaustoimenpiteet ja -dokumentaatiot toteutettaville automaatiosovelluksille luovat pohjaa laadunhallinnalle ja toimivat kokonaistoimituksen yhtenä laatuparametrina.

Automaatiosovelluksen testaaminen toimisto-olosuhteissa on haasteellista, sillä sovelluksen toiminnallisuus on voimakkaassa vuorovaikutuksessa tehdasprosessin tilan kanssa. Tehdastestauksissa automaatiojärjestelmä testataan ilman prosessia, joten mittausdataa ei ole saatavilla. Jotta automaatiosovelluksen toiminnallisuutta voidaan testata, tulee prosessin tilaa simuloida suoraan automaatiojärjestelmästä joko manipuloimalla logiikan muuttujia tai ulkoisen ohjelmoidun sovelluksen avulla. Automaatiosovellusta tulisi ihanteellisessa tilanteessa testata sen koko kehityskaaren ajan, jolloin prosessitilan simulointi on tehokas työkalu automaatiosovelluksen suunnitteluun ja kehittämiseen.

Koska automaatiojärjestelmän tehdastestaus on niin merkittävä osa automaatioprojektia, sen hyväksytty testaus on yksi isoista virstanpylväistä koko projektissa. Tehdastestauksissa koko automaatiojärjestelmä käydään asiakkaan kanssa läpi niin laitteiston kuin ohjelmistojen osalta ja projektien laajuudesta riippuen saattavat kestää tunneista jopa kuukausiin. Tehdastestauksessa automaatiojärjestelmä pyritään rakentamaan todellista kohdetta vastaavaksi, jotta koko järjestelmän toimivuus voidaan todentaa ennen siirtämistä asiakkaalle. Todenmukainen automaatiojärjestelmän testaaminen vaatisi kuitenkin prosessilaitteiston kaikkine instrumentteineen. Suunnittelijan tiloihin ei tätä prosessia ole kannattavaa tai mahdollista rakentaa, joten prosessin instrumenttien antama informaatio esitetään simuloimalla. Tavanomaisessa tehdastestauksessa automaatiojärjestelmä sisältää usein kuvan 2 mukaisesti kaikki komponentit ohjelmoitavista logiikoista ylöspäin.



Kuva 2. *Simulointisovellus liitettynä tehtaan automaatiojärjestelmään. Automaatiojärjestelmän arkkitehtuuri mukailtu lähteestä [5]*

Tehdastestauksessa pyritään todentamaan asiakkaalle koko järjestelmän vaatimuksenmukaisuus. Testaamisella pyritään esittämään logiikoiden ohjelmien toiminnallisuus, tiedonsiirto automaatioverkossa sekä valvomoiden näkymät ja toiminta. [6] Jotta automaatiosovellus tekisi haluttuja toimenpiteitä, tarvitsee se informaatiota prosessin tilasta. Tätä voidaan simuloida asettamalla mittausten arvoja logiikan muuttujiin manuaalisesti. Haasteena ilmenee kuitenkin prosessissa tapahtuvien toisistaan riippuvien tapahtumien simulointi, jolloin ohjelman suoritus ohjautuu virhetilanteeseen tai lukitukseen. Näiden vasteiden sekä mittauksien muuttamisen kankeus on tehdastestauksen sujuvan etenemisen suurin ongelma.

Prosessin vasteen ja tilan simulointi erillisen ohjelman avulla toimisi ratkaisuna testauksen sujuvaan etenemiseen. Lisäksi testauksen tehokkuus nousee ja painopiste siirtyy järjestelmän toimivuuden todentamiseen. Tehokas ja selkeä automaatiojärjestelmän tehdastestaus vahvistaisi myös asiakkaan luottamuksen tunnetta automaatiojärjestelmän toimittamisesta, sillä onnistunut testi on näyttö vaaditun laadun toteutumisesta.

On tärkeää huomioida, että prosessien vasteet ja automaatiojärjestelmien vaatimukset muuttuvat toimintaympäristön mukaan. Tämä tulee huomioida myös simulointiympäristön järjestelmää laadittaessa. Tässä työssä simulointiympäristö toteutetaan tukemaan vesi- ja ympäristöhuollon automaatiojärjestelmiä, jotka toteutetaan pääosin käyttämällä Siemens TIA portal tai PCS7 -järjestelmiä.

1.2 Tutkimuskysymykset ja tavoitteet

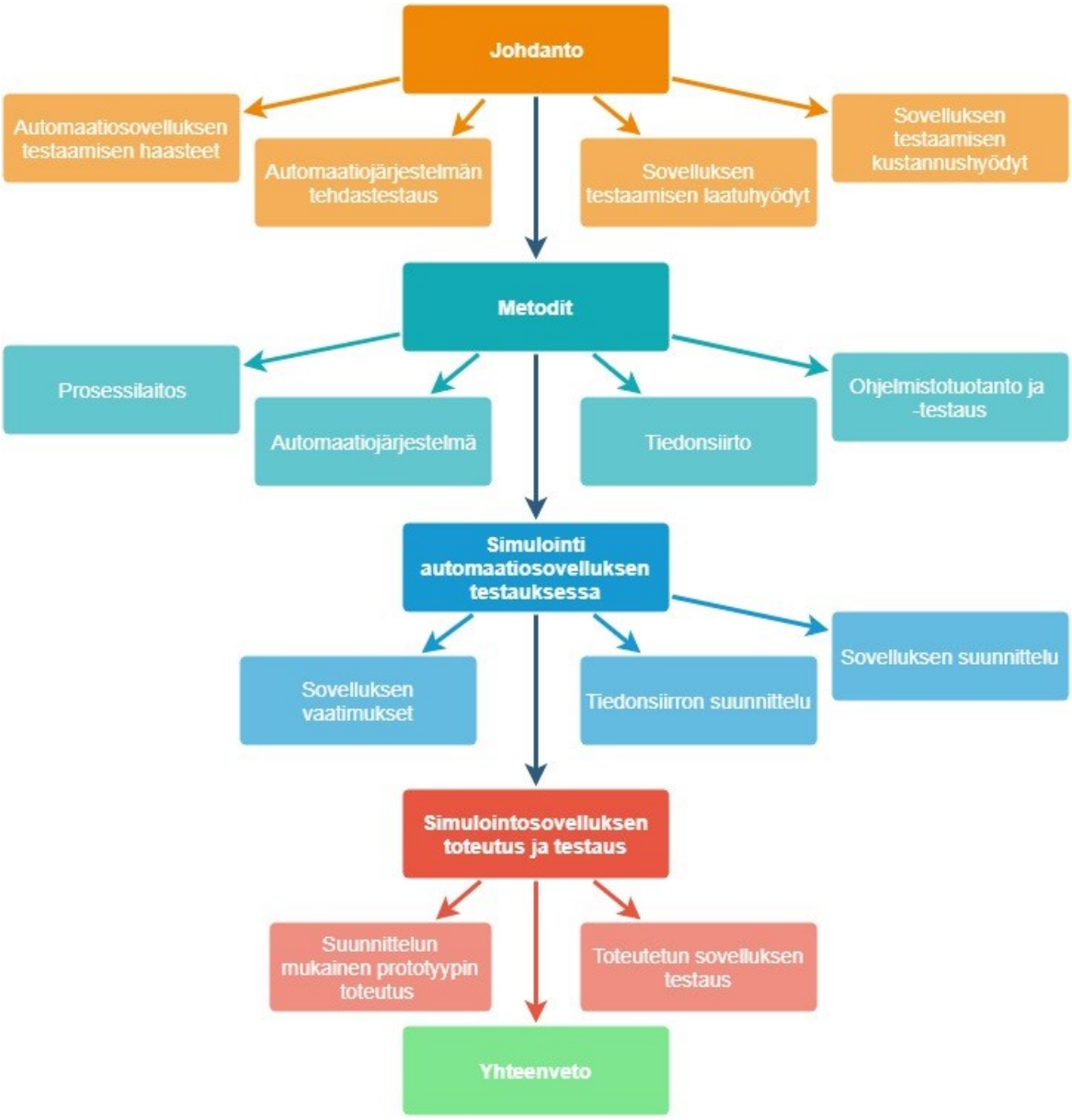
Diplomityössä selvitetään mahdollisuuksia tehostaa laadullisesti sekä ajallisesti automaatiosovelluksen testaamista simuloimalla prosessin tiloja. Työssä tutkittavia aiheita ovat:

1. Minkälaisin tiedonsiirtomenetelmin ohjelmoidun logiikan muuttujia voidaan manipuloida?
2. Miten voidaan tehokkaasti luoda sovelluskohtaiset muuttujat simulointiohjelmaan?
3. Miten erillinen testausohjelmisto voidaan käyttäjäystävällisesti yksilöidä projektikohtaiseksi, jotta testaaminen olisi vaivatonta ja tehokasta?

Diplomityön tavoitteena on automaatiosovelluksen testaamisen tehokkuuden parantaminen. Automaatiosovelluksen testaaminen saadaan yhtenäistettyä ja luotua yksikköä koskeva laatuparametri toimitettaville automaatiojärjestelmille. Simulointiohjelman käyttö selkeyttää ja tehostaa automaatiojärjestelmän tehdastestausta sekä toimittajalle että tilaajalle, sillä prosessin tilaan vaikuttavat tekijät ovat jatkuvasti nähtävillä.

1.3 Työn kulku

Diplomityö on jaettu viiteen osioon sisällön mukaan (Kuva 3). Johdannossa esitetään automaatiosovelluksen sekä -järjestelmän suunnittelu- ja toteutusvaiheisiin liittyviä toimia ja haasteita, jotta ymmärretään, mikä on varsinainen ongelma ja minkälaisin tutkimusmetodein ongelma voidaan ratkaista. Työn toisessa osassa tutkitaan työn toteuttamiseen johtavat metodit aihepiireittäin. Teoreettinen katsaus antaa pohjan ohjelmiston toteuttamiseen ongelman ratkaisemiseksi. Kolmannessa osassa esitetään valituin tutkimusmenetelmin ja edellä löydetyin metodein ongelmaan ratkaisu. Neljännessä osassa todistetaan ratkaisun paikkansapitävyys testaamalla toteutettua simulointisovellusta FAT-ympäristössä ja saadaan vertailtavia tuloksia. Viidennessä kappaleessa esitetään yhteenveto diplomityön tuloksista ja sen tuottamista hyödyistä.



Kuva 3. *Kaavio diplomityön rakenteesta.*

2. METODIT

Toimivan testausympäristön rakentamiseen tarvitaan tietoa prosessilaitoksesta, automaatiojärjestelmän eri komponenttien toiminnasta sekä niiden keskinäisistä yhteyksistä. Luvussa esitetään teoreettinen katsaus järjestelmän komponenteista ja tiedonsiirrosta testaussovelluksen ja automaatiojärjestelmän välillä. Automaatiosovelluksen testaaminen toteutetaan ohjelmistotestauksen keskeisillä periaatteilla, joita luvussa tutkitaan. Testaussovelluksen rakentamiseen käytetään luvussa tutkittuja ohjelmistosuunnittelun metodeja.

2.1 Prosessilaitos

Prosessilaitoksella tarkoitetaan hyödykkeiden tuottamiseen tarvittavaa kokonaisuutta, joka koostuu tiloista, laitteista ja palveluista [7]. Automaatiojärjestelmä on usein kytköksissä kaikkiin prosessilaitoksen osiin, mutta automaatiosovelluksen testauksen kannalta on oleellista esitellä tarkemmin laitteistoa ja niiden toimintoja.

Automaatioalalla prosessilla tarkoitetaan usein kenttälaitteiden ohjaamista prosessista saatavien mittaussuureiden avulla ja toiminnan vaikutusten arvioimista [2]. Standardin ISO 10628 mukaan prosessi on sarja toimintoja, joiden tarkoituksena on muokata, siirtää tai varastoida materiaalia tai energiaa. Prosessissa tuotteen jalostamiseen voidaan käyttää kemiallisia, biologisia tai fysikaalisia operaatioita. [2, 7, 8] Prosesseissa tuotteen jalostaminen voi olla yhden tai useamman syötteen, eli raaka-aineen operoimista. Prosessit jaetaan kahteen kategoriaan niiden ominaisuuksien mukaan: jatkuvat ja epäjatkuvat prosessit.

Jatkuvissa prosesseissa prosessiin annetaan jatkuvavirtainen syöte, jota eri toiminnoin käsitellään johtaen jatkuvavirtaiseen lopputuotteeseen. Tuotteen valmistus on usein monen eri toiminnon sarja, joista yhtä kuvataan prosessiaskeleella. Prosessiaskel muodostuu yksikköoperaatiosta, joka on pienin toiminto, joka voidaan esittää prosessin määrittelyllä. Jatkuvissa prosesseissa on usein vahvasti mukana erilaisia säätimiä, joiden avulla vaikutetaan toimintojen kvantitatiivisiin ominaisuuksiin, esimerkiksi virtausten suhteita säätämällä aikaansaadaan haluttu seossuhde. [7, 8]

Epäjatkuvissa prosesseissa tuotteet valmistuvat yksi kerrallaan tai ei-jatkuvissa erissä. Valmistuksen aikana tuotteet liikkuvat valmistusyksiköiden tai ryhmien välillä ja tuotteille tehdään kussakin yksikössä prosessivaiheen määräämiä toimenpiteitä. Epäjatkuvat prosessit ovat yleensä panosprosesseja, mutta mukana voi olla myös jatkuvia osia. Panosprosesseille ominaisia piirteitä on valmistus tietyn reseptin mukaan, jolloin tuotteen valmistuksen alkusyötteeksi ladataan tietty määrä eri raaka-aineita, joita käsitellään tietyssä järjestyksessä ja lopulta syntyy tuote-erä. Panosprosesseja esiintyy useimmiten lääke-,

elintarvike ja kemianteollisuudessa. Epäjatkuvien prosessien alaluokka on sekvenssi, jossa valmistusvaiheet etenevät määritellyssä kronologisessa vaiheessa. [7-9]

Prosessityypit vaikuttavat automaatiosovelluksen toteutustapaan ja luonteeseen. Automaatiosuunnittelijan täytyy ymmärtää prosessin toiminta pääpiirteittäin, jotta sovellus toteuttaa sille asetetut vaatimukset. Automaatiojärjestelmien tehdastestauksissa simuloidaan prosessin vastetta, jotta ohjelman toiminta voidaan todentaa. Prosessin erilaiset tilat testataan ja nämä otetaan huomioon mahdollisimman laajasti. Isot ja monimutkaiset prosessit, varsinkin jatkuvat, voivat sisältää niin laajan määrän erilaisia tiloja, että täydellistä testausta on lähes mahdotonta suorittaa.

Prosessin toimintaa esitetään erilaisten standardoitujen kuvausten ja kaavioiden avulla. Näiden prosessidokumentaatioiden pohjalta suunnitellaan vaatimukset täyttävä automaatiojärjestelmä. Yleisimpiin automaatiosuunnittelijan prosessikaavioihin kuuluvat lohkokaavio, virtauskaavio sekä instrumentointikaavio.

Lohkokaaviossa kuvataan prosessin toimintoa yleisellä tasolla lohkojen ja niiden välisten yhteyksien avulla. Kaaviomalli helpottaa ison prosessin toiminnan ymmärtämistä. Kaaviossa on kuvattu prosessin osat sekä niiden toimintasuunnat. Lisäksi kaaviosta näkee minkälaiset materiaalivirtaukset (syötteet) tuottavat tietyn tuotoksen. [7] Lohkokaavioita voidaan käyttää automaatiojärjestelmän arkkitehtuurin suunnittelussa.

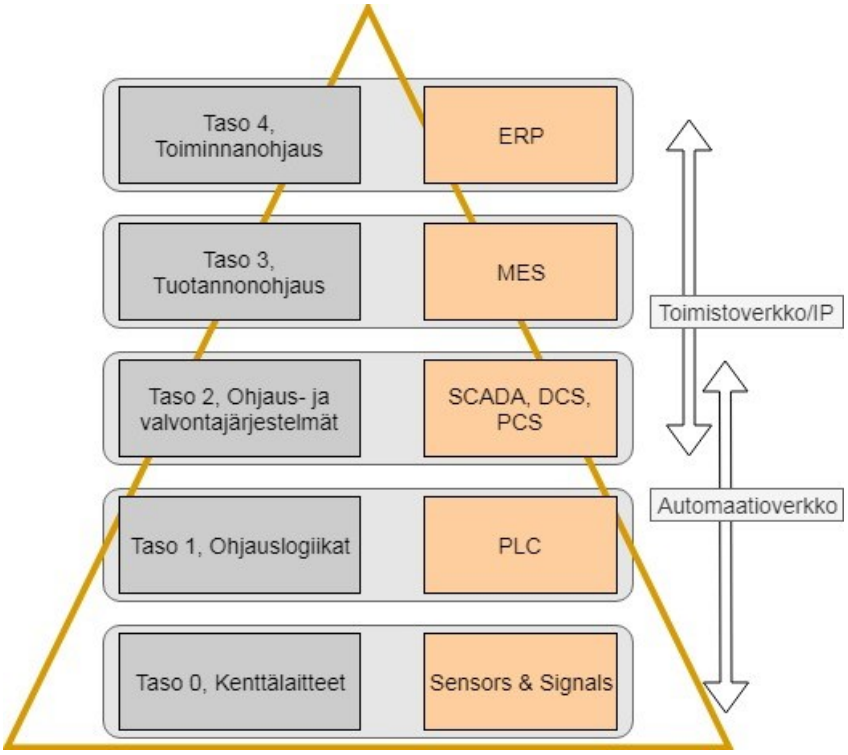
Virtauskaavio on graafisin symbolein kuvattu kaavio prosessista. Kaaviossa tulee näkyä prosessin suorittamiseen vaadittavat välttämättömät instrumentit ja putkistot. Lisäksi kaaviosta näkee tulevat ja lähtevät virrat, niiden suunnat, yksiköt ja määrät, sekä tyypilliset operaatioehdot. [7] Virtauskaavioita voidaan käyttää osana automaatiosovelluksen suunnittelua.

Instrumentointikaavio, tai toiselta nimeltään putkisto- ja instrumenttikaavio, eli PI-kaavio, pohjautuu virtauskaavioon, mutta sisältää kaikkien prosessin laitteiden tarkemmat määritteet, positiot sekä ohjaustavat. Kaaviossa näkyy myös putkistojen tekniset tiedot sekä identifiointinumero. [7] Instrumentointikaaviot ovat automaatiosuunnittelussa pohjana niin järjestelmän kuin sovelluksen suunnittelussa sekä testaamisessa.

2.2 Automaatiojärjestelmä

Automaatiojärjestelmä kytkee prosessilaitteiston, käyttäjän sekä yrityksen tietojärjestelmät yhteen. Automaatiojärjestelmän perusedellytyksenä on parantaa tuotannon laatua, tehokkuutta sekä turvallisuutta. Kappaleessa on esitetty automaatiojärjestelmän komponentteja sekä niiden toimintoja niin kutsutun ISA-95 automaatiopyramidin mukaisesti. Prosessilaitoksen toimintaa ja ohjaamista voidaan kuvata ISA-95 standardin avulla (Kuva 4). Standardin hierarkiamallia kutsutaan myös automaatiopyramidiksi. Hierarkiamalli koostuu viidestä tasosta, jotka läpileikkaavat yrityksen tuotantotoimet vertikaalisesti.

Vertikaalinen integraatio tähtää askelten ja monimutkaisuuden minimointiin ylimmän tason päätöksenteosta prosessinohjaukseen. Automaatiopyramidin kerrosten välinen tiedonsiirto pohjautuu useimmissa tapauksissa kahteen erilliseen tiedonsiirtoverkkoon, jonka solmukohtana toimii Taso 2 ohjaus- ja valvontajärjestelmät (Kuva 4). [10]



Kuva 4. *Laitoksen hierarkia ISA-95. Mukailtu [10, 11]*

Automaatiopyramidin kaksi ylintä kerrosta muodostuvat yrityksen tietojärjestelmiin kuuluvista toiminnanohjauksesta (Enterprise Resource Planning, ERP) sekä valmistuksen ohjauksesta (Manufacturing Execution System, MES). Yrityksen tietojärjestelmien alla toimii automaatiojärjestelmä, joka muodostuu kolmesta alimmaisesta tasosta. [10, 11]

ERP määrittelee liiketoiminnalliset aktiviteetit tuotantotoiminnan hallintaan. Tietojärjestelmä kerää ja ylläpitää tietoa raaka-aineiden, varaosien ja henkilöstön käytöstä, tilanteesta ja hankinnasta. ERP myös seuraa energiankäyttöä ja –hallintaa sekä kerää ja ylläpitää tietoa laadunhallinnasta asiakasvaatimusten täyttämiseksi. Näiden lisäksi, järjestelmä ylläpitää tietokantaa laitteistojen huoltoväleistä, huoltotarpeista ja luo niiden pohjalta ennakkohuoltosuunnitelman. Liiketoiminnan sisäisten toimintojen jatkeena, järjestelmä seuraa hyödykkeiden valmistuksen aikataulutusta, varastointia ja toimitusta. ERP:n toiminnan aikaikkuna riippuu liiketoiminnasta ja yrityksen koosta ja vaihtelee päivistä kuukausiin. [11]

MES määrittelee tuotannolliset aktiviteetit tuotteen valmistamiseen tuotannontekijöistä hyödykkeiksi. Tietojärjestelmä raportoi hyödykkeiden valmistuskustannuksia osastoittain. Sen tehtävänä on kerätä ja ylläpitää alueellista dataa tuotannosta, varastosta, raaka-

aineista, työvoimasta, varaosista sekä laadusta. Hierarkian Tasolla 3 määritetään tuotannon toteuttamiseen vaaditut laite, materiaali ja henkilöresurssit, sekä niiden varaaminen. Toisin kuin ERP:lle, MES:lle aikaikkuna tapahtumien käsittelylle on tiheämpää ja vaihtelee useimmiten sekunneista päiviin. [11]

Toiminnanohjauksen ja tuotannonohjauksen alle jäävät tasot kuuluvat automaatiojärjestelmään. Nämä voidaan jakaa kolmeen ryhmään: ohjaus- ja valvontajärjestelmät, ohjauslogiikat, sekä kenttälaitteet. Kenttälaitteiden tehtävänä on muuttaa prosessin mittalaitteiden fysikaaliset suureet järjestelmän ymmärrettäväksi tai järjestelmän ohjauksen fysikaaliseksi toiminnaksi. Seuraavaksi ohjauslogiikat määrittelevät fyysisen prosessin aktiviteetit kenttäinformaation tulkinnasta ja prosessin ohjaamisesta. [11] Viimeisenä, ohjaus- ja valvontajärjestelmät nimensä mukaisesti ohjaavat ja valvovat fyysisen prosessin aktiviteetteja, sekä toimivat näin automaatiojärjestelmän ylimmällä tasolla. [11] On huomioitava, että toisinkuin ERP ja MES järjestelmät, automaatiojärjestelmän aikaikkunat ovat huomattavasti tiheämpiä, vaihdellen millisekunneista tunteihin.

Seuraavissa kappaleissa esitetään kuvan 4 ISA-95 Tasojen 0-2 automaatiojärjestelmää koskevat komponentit tarkemmin sekä perehdytään tasojen horisontaaliseen integraatioon, joka sisältää saman tason eri laitteiden ja sovellusten yhteenliittymistä.

Ohjaus- ja valvontajärjestelmät

Nykyaikaiset tuotanto- ja valmistusprosessit pyritään toteuttamaan kustannustehokkaiksi, toimintavarmoiksi ja turvallisiksi järjestelmiksi. Tämä aikaansaadaan yhdistämällä prosessiosat samaan automaatiojärjestelmään, jolloin koko järjestelmän ohjaaminen toteutuu keskitetysti. Automaatiojärjestelmän kenttälaitteiden välimatkat ja laitemäärät voivat kasvaa suuriksi, jolloin kokonaisuuden hallintaan tarvitaan vaatimukset täyttävä ohjausjärjestelmä. Prosessiteollisuudessa automaatiojärjestelmät ovat tyypillisesti hajautettuja ohjaus- ja valvontajärjestelmiä. Näistä esimerkkeinä Supervisory control and data acquisition (SCADA), Distributed Control System (DCS) ja Process control system (PCS). [3]

SCADA –järjestelmää käytetään maantieteellisesti laajojen prosessien hallintaan. Näistä esimerkkinä kaupunkien vesilaitokset, joissa useiden ala-asemien informaatio siirtyy valvomoon pitkien etäisyyksien yli esimerkiksi radio- ja mobiiliverkkojen avulla. DCS- ja PCS-järjestelmät ovat saman kantaisia kuin SCADA, mutta usein tehtaiden sisäisiä hajautettuja järjestelmiä. Näissä useiden automaatioverkkoon kytkettyjen tietokoneiden avulla muodostetaan järjestelmä, joka sisältää prosessiohjauksen, valvonnan sekä hälytysten ja prosessidatan käsittelyn, esimerkiksi raportoinnin muodossa. DCS ja PCS pystyvät automaatioverkon suuren tiedonsiirtonopeuden ansiosta toimimaan reaaliaikaisena, kun taas SCADA-järjestelmä, jossa on pieni viive ala-asemille. [12]

Järjestelmän käyttöpäätteenä toimii yksi tai useampi prosessivalvomo, jonka kautta operaattori seuraa ja ohjaa laitoksen toimintaa. [3] Valvomo-ohjelmistolla seurataan prosessin etenemistä ja tarkkaillaan sen tietoja. Automaattiajolla valvomosta seurataan ja reagoidaan hälytyksiin ja poikkeuksiin. Operaattorilla on myös mahdollisuus ohjata prosessilaitosta, tai sen osia manuaalisesti. Valvomo-ohjelmistoon on usein integroitu myös historiatietokanta, karttapalvelu, optimointi ja muita lisäohjelmistoja[3].

Automaatiojärjestelmän tehtävä on laajentunut perinteisestä prosessinohjauksesta paljon myös ohjelmistoteknisiin toteutuksiin. Prosesseista halutaan enemmän dataa, jota kerätään ja käsitellään pitkällä aikavälillä mahdollistaen esimerkiksi resurssien tehokkaamman hyödyntämiseen. Prosessin ohjaaminen ja valvonta toteutetaan usein perinteisemmillä luotettavaksi todetuilla metodeilla ja lisäpalvelut erillisillä ohjelmistoilla. [2] Automaatiojärjestelmään kuuluu tänä päivänä myös olennaisesti etäohjaukset ja kunnossapitosovellukset. [3]

Ohjaus- ja valvontajärjestelmät keräävät informaationsa ohjauslogiikoilta (Programmable Logic Controller, PLC). Informaation keräämisen lisäksi ohjaus-ja valvontajärjestelmä antaa prosessi- ja laitekohtaisia ohjauksia ohjauslogiikoille. [3]

Ohjauslogiikat

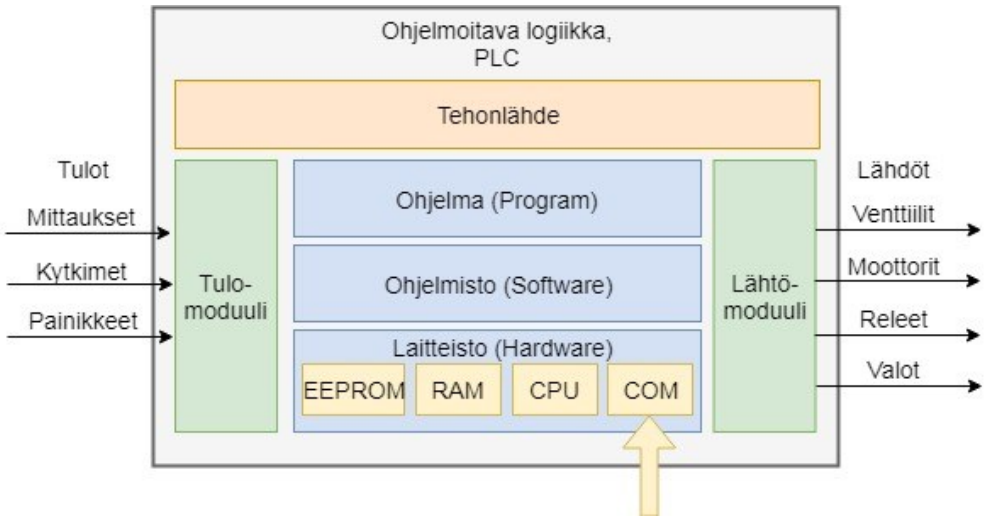
Prosessilaitosten ohjauksia hallitsevat dominoivasti tietokoneet. Prosessien ohjauksissa käytettävien tietokoneiden perusedellytyksiä ovat vakaus ja luotettavuus. Vaatimuksen saavuttamiseksi on kehitetty erityisellä arkkitehtuurilla varustettuja tietokoneita, kuten ohjelmoitavat logiikat. Ohjelmoitavien logiikoiden tulee olla rakenteeltaan robusteja, jotta ne kestävät teollisuuden vaihtelevia olosuhteita ja häiriöitä.[1]

Ohjelmoitavat logiikat eivät ainoastaan käsittele diskreettejä tapahtumia ja toimi SCADA-pohjana. Niiden toimintoja ovat myös analogia-, kone-, paikoitus- ja muut aikakriittiset ohjaukset. Prosessinohjausjärjestelmiä kutsutaan reaaliaikajärjestelmiksi, sillä prosessinohjauksissa vaaditaan nopeaa vasteaikaa. Mikäli vasteaika kasvaa hallitsemattomasti, aiheutuu vaaraa niin prosessille kuin kustannustekijöille. [4, 13]

Ohjelmoitavat logiikat käyttävät sisäistä ohjelmoitavaa muistia ohjeiden tallentamiseen ja funktioiden toteuttamiseen, näistä esimerkkinä loogiset ja aritmeettiset operaatiot, sekä ajastukset ja laskurit, joiden avulla ohjataan laitteita ja prosesseja. Ohjelmoitavat logiikat on optimoitu suorittamaan näitä operaatioita. PLC tarjoaa joustavuutta ja tehokkuutta, sillä prosessin ohjauksen muuttaminen onnistuu muokkaamalla logiikan ohjelmaa. [14]

Ohjelmoitavien logiikoiden toiminnallisuutta määritellään standardikokonaisuudessa IEC 61131, joskin valmistajakohtaisia muunnelmia kuitenkin on. Näistä toteutettavan simulointisovelluksen kannalta tärkeimpiä ovat standardin osat ”laitteisto 61131-1”, ”ohjelmisto 61131-3” ja ”kommunikaatio 61131-5”.

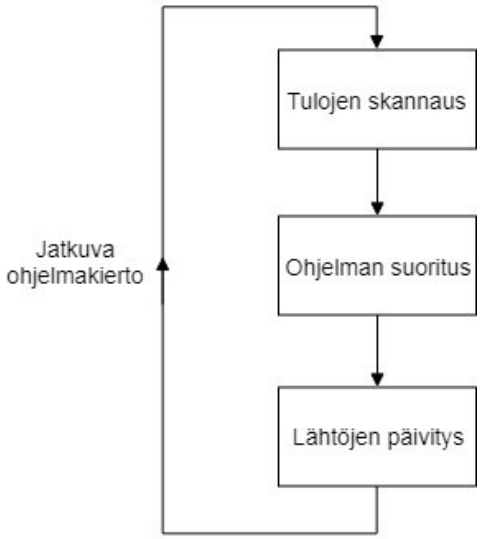
Laitteisto, eli logiikan fyysinen arkkitehtuuri on määritelty standardin IEC 61131-1 mukaan. Logiikan komponentit, rajapinnat sekä toiminnat kattavat standardoidut vaatimukset, jotka on esitetty kuvassa 5. Standardin mukaan arkkitehtuuri voidaan jakaa karkeasti kuuteen osaan: prosessori, ohjelmamuisti, teholähde, IO-rajapinta, kommunikointirajapinta sekä ohjelmointilaite.



Kuva 5. *PLC arkkitehtuuri. Muokattu [15]*

Prosessori (CPU) on mikroprosessori, joka tulkitsee sisään tulevat signaalit ja prosessoi niille tehtävät toimenpiteet ohjelmamuistiin (RAM) tallennetun ohjelman mukaisesti sekä antaa tilanteen mukaiset ohjeet ulostuloille. RAM sisältää myös IO-statukset sekä ohjelmassa käytettyjen ajastimien ja laskureiden tilat. IO-rajapinta on fyysinen rajapinta logiikan ja ulkomaailman välillä. Näissä sisääntulot voivat sisältää esimerkiksi kytkimiä, painemittauksia sekä virtausmittauksia, ja ulostulot esimerkiksi releiden ohjauksia, solenoidiventtiilejä ja taajuusmuuttajan ohjauksia. Kommunikointirajapinta (COM) mahdollistaa tiedonsiirron laitteiden välillä. Ohjelmointilaite on vastuussa logiikan ulkoisista toiminnallisuuksista, kuten esimerkiksi ohjelman debuggaus ja ohjelmointi. [16]

Standardin mukaan ohjelmoitavan logiikan toiminta perustuu jatkuvaan ohjelmakiertoon (cycle) (Kuva 6). Ohjelmakierto alkaa tulojen skannauksella IO-kortilta, jonka jälkeen tilat kopioidaan RAM-muistiin. Ohjelma suoritetaan askeleittain käyttäen RAM-muistissa olevia tilojen arvoja. Kun viimeinen ohjelman askel on suoritettu, kopioidaan lähtöjen tilanne RAM –muistista IO-kortille. Kun lähdöt on kirjoitettu, aloitetaan ohjelmakierto alusta. [14]



Kuva 6. *Ohjelmakierto ohjelmoitavalla logiikalla [14]*

Yhden ohjelmakierron aikaa kutsutaan kiertoajaksi (cycle time). Tyypillisesti ohjelman kiertoajaksi asetetaan 10-50 ms. Ohjelman kiertoaika riippuu ohjelman kompleksisuudesta, IO-määrästä ja käytettävästä CPU:sta.[14]

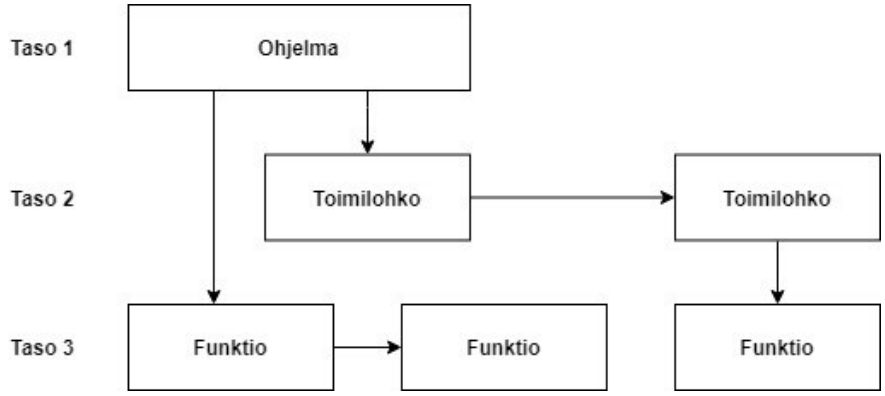
RAM-muistissa olevat I/O-pisteiden muistipaikat nimetään eri valmistajilla poikkeavasti. Taulukko 1 on esimerkkejä eri logiikkamallien I/O-rajapinnan muistipaikkojen eroavaisuuksia. RAM muistin osoite on usein riippuvainen myös konfiguroinnista. I/O-pisteiden arvoja voidaan muokata logiikan ulkopuolelta virtuaalisesti erilaisten mekanismien avulla. Valmistajilla on usein tarkoitukseen kehitettyjä omia sovelluksia, mutta standardoitujen tiedonsiirtoprotokollien avulla muistin käsittelyyn löytyy myös universaaleja mekanismeja, kuten myöhemmässä vaiheessa esiteltävä OPC UA.

Taulukko 1. *Esimerkki PLC IO-osoitteista (directly represented values)*

I/O-tyyppi	Beckhoff TC3	Omron NJ501	Siemens S7-1500
DI	IX0.00	J01_Ch1_In00	I0.00
DO	QX0.00	J02_Ch1_Out_00	Q0.00
AI	IW0	J03_Ch1_In	IW0
AO	QW0	J04_Ch1_Out	QW0

IEC 61131-3 määrittelee logiikan sisäisen ohjelmallisen hierarkian, jota kutsutaan rakenneyksiköksi (Program Organization Unit, POU). Standardin määrittelemän rakenneyksikön tarkoituksena on yksinkertaistaa järjestelmää vähentämällä erilaisten ohjelmalohkojen määrää. Rakenneyksikön ominaisuuksiin kuuluu sekä saman että alemman tason elementin kutsu, joka on esitetty kuvassa 7. Rakenneyksikkö muodostuu kolmesta osasta: ohjelma (program), toimilohko (function block) ja funktio (function).[17]

Ohjelma yhdistää IEC 61131-3 -ohjelmointikielen elementit ja toteuttaa halutun kokonaisuuden toiminnot. Suorituskerran päätyttyä sen sisältämät arvot säilyvät seuraavalle ohjelmakierrolle. [18] Järjestelmän fyysisen rajapinnan muuttujat ovat esitettynä ohjelman muistissa, jonka kautta myös muut rakenneyksikön osat pääset niihin käsiksi. Toimilohko voi sisältää rajattoman määrän liityntöjä sekä muuttujia, joihin kohdistetaan lohkon sisäisiä operaatioita ja ne säilyttävät tilansa ohjelmakiertojen välissä. [17] Funktio on toistettava ohjelma, jolla ei ole sisäistä muistia, toisin sanoen funktion palauttama arvo on samoilla syötteillä aina sama. [18]



Kuva 7. *Ohjelmoitavan logiikan rakenneyksikkö [17]*

IEC 61131-3 -standardi määrittelee viisi erilaista ohjelmointikieltä. Ohjelmointikielten syntaksit ja semantiikat ovat standardissa tarkasti määriteltyjä, mutta eri laitevalmistajien määrittelyissä niissä esiintyy kuitenkin poikkeavuuksia. Ohjelmointikielistä kolme ovat graafisia (tikapuukaavio, toimilohkokaavio ja sekvenssikaavio) ja kaksi tekstimuotoista (käskylista ja rakenteinen teksti). [19]

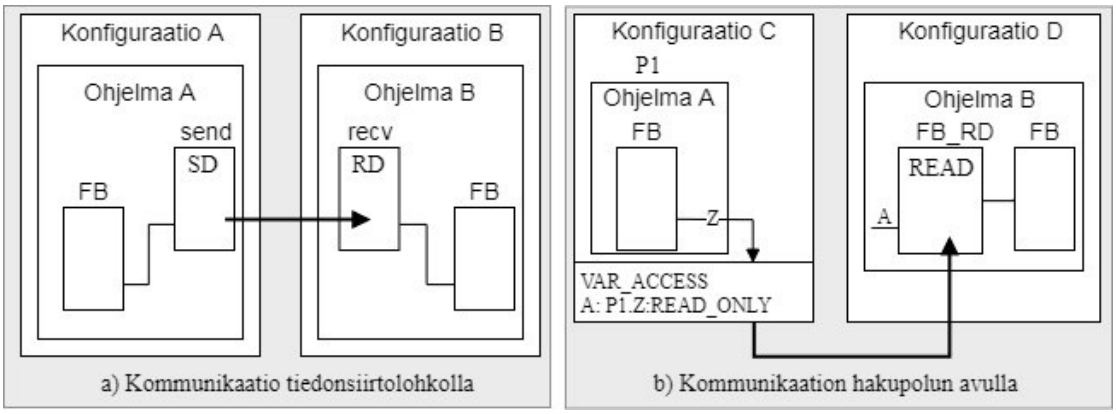
Ohjelmoitavissa logiikoissa käytettävät muuttujat tallennetaan logiikkaan käyttötarpeen ja yleisten vaatimusten mukaisesti eri datatyyppeinä (Taulukko 2). Kuten ohjelmoinnissa yleisesti, muuttujat esitellään ennen käyttöä ja niille määritetään datatyyppi. Muuttujat esitellään ohjelmoitavissa logiikoissa yleensä symbolilistoissa, joissa niille voidaan antaa osoite, symboli, kuvaus ja datatyyppi. Logiikan suorittaessa ohjelmaa, tulee datatyyppien olla operaatioita vastaavia, jotta ei synny suoritusvirheitä tai virheellisiä tuloksia. [20]

Taulukko 2. *IEC61131-1 datatyypit. Muokattu [20]*

Lyhenne	Datatyyppi	Pituus/bit	Alustus
BOOL	Totuusarvo	1	0
SINT	Lyhyt kokonaisluku	8	0
INT	Kokonaisluku	16	0
DINT	Kahdenkertainen kokonaisluku	32	0
LINT	Pitkä kokonaisluku	64	0
USINT	Etumerkitön lyhyt kokonaisluku	8	0
UINT	Etumerkitön kokonaisluku	16	0
UDINT	Etumerkitön kahdenkertainen kokonaisluku	32	0
UDLINT	Etumerkitön pitkä kokonaisluku	64	0
REAL	Reaaliluku	32	0.0
LREAL	Pitkä reaaliluku	64	0.0
TIME	Aika	*	T#0s
DATE	Päivämäärä	*	D#0001-01-01
TOD	Kellonaika	*	TOD#00:00:00
DT	Päivämäärä ja kellonaika	*	D#0 + TOD#0
STRING	Merkkijono muuttuva pituus	*	' '
BYTE	Tavu	8	' '
WORD	Sana, koostuu kahdesta tavusta	16	' '
DWORD	Kahdenkertainen sana	32	' '
LWORD	Pitkä sana	64	' '

Standardi IEC 61131 määrittelee mekanismeja ohjelmoitavan logiikan ja elektronisten laitteiden väliseen tiedonsiirtoon. Kommunikaatiomekanismeja ovat esimerkiksi konfiguraatioiden välinen tiedonsiirto IEC 61131-5 mukaisten kommunikaatiolohkojen avulla sekä tiedonsiirto eri laitteistojen välillä hakupolun avulla. Variaatiot on esitetty graafisesti kuvassa 8. [21]

Hakupolkuja (Access path) käytetään tiedonsiirtoon eri konfiguraatioiden sekä ohjelmien välillä, kuten esimerkiksi järjestelmässä kahden logiikan välisenä rajapintaana. Globaalit muuttujat (Global variables) voidaan esitellä ja niitä voidaan käyttää konfiguraatioissa, suorittimilla ja ohjelmissa. Toimilohkoilla pääsy muuttujiin ulkoisen esittelyn avulla. Funktioilla ei pääsyä. Tiedonsiirtolohkot (Communication block) ovat erityisiä toimilohkoja pakettien tiedonsiirtoon lähettäjältä vastaanottajalle. Tiedonsiirtolohkot ovat linkitetty yhteen ohjelmaan eivätkä ole näkyvissä konfiguraation ulkopuolelle. Kommunikointilohkot ovat määritetty IEC 61131-5 standardissa. [17] (Kuva 8) Monissa universaaleissa tiedonsiirtomenetelmissä ohjelmoitava logiikka käyttää kommunikointiolohkoa viestinnän käsittelyyn, kuten esimerkiksi Siemens S7-1500 OPC UA tiedonsiirtoon [22].



Kuva 8. *Standardin IEC 6113 mukainen kommunikointi [19]*

Automaation kenttälaitteet

Automaatiopyramidin alimmalla tasolla ovat prosessin kenttälaitteet (Kuva 4). Automaatiojärjestelmissä esiintyy tavanomaisesti mittauksia, koneita, venttiileitä ja säätimiä. Kenttälaitteiden tehtävänä on muuntaa automaatiojärjestelmän antamat ohjaussignaalit fysikaalisiksi toiminnoiksi ja prosessissa tapahtuvat fysikaaliset suureet mittaussignaaleiksi.

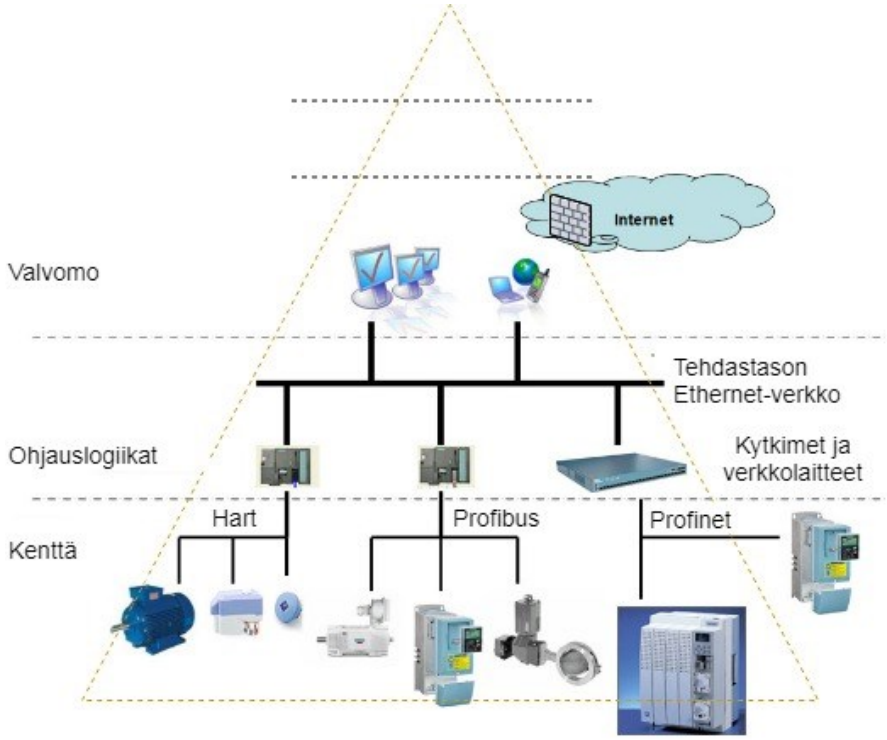
Prosessin kenttälaitteet kommunikoivat vertikaalisesti automaatiojärjestelmän ohjauslogiikoille eri tiedonsiirtomenetelmin, joita kutsutaan kenttäväyliksi. Automaatiojärjestelmän kenttäväylät ovat joko analogisia, digitaalisia tai niiden yhdistelmiä. Analogiset kenttäväylät sisältävät ainoastaan informaatiosignaalin, kun taas digitaaliset kenttäväylät mahdollistaa monipuolisemmat kommunikointiominaisuudet esimerkiksi diagnostiikkatiedot.

Automaatiojärjestelmän ohjauslogiikoilla kenttäväylät vaikuttavat usein ohjelman konfigurointiin ja kenttälaitteiden tiedon käsittelyyn. Prosessiteollisuudessa käytettyjä kenttäväyliä perinteisen 4-20 mA analogiaviestin lisäksi ovat AS-I, Interbus, Profibus PA, Profibus DP, DeviceNet, Control Net, Modbus, HART ja Foundation Fieldbus. [23]

2.3 Automaatiojärjestelmän tiedonsiirto

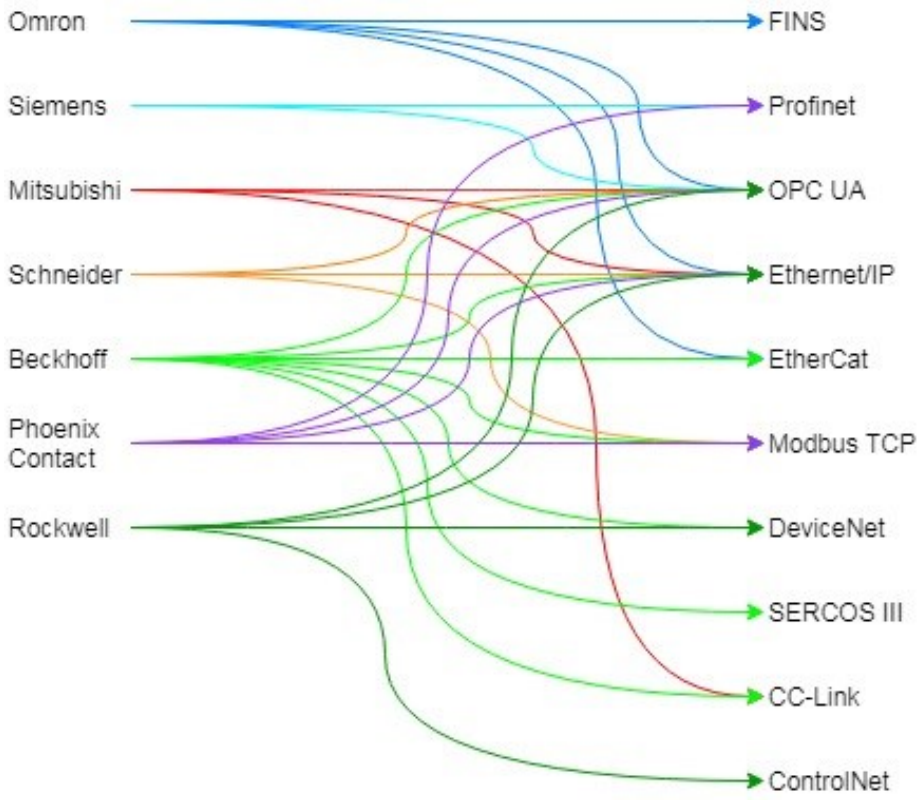
Automaatiojärjestelmän tiedonsiirrolla tarkoitetaan ISA-95 mallin vertikaalista ja horisontaalista laitteiden välistä kommunikaatiota. Tiedonsiirto automaatiojärjestelmissä jaetaan kenttätasolle sekä järjestelmätasolle (Kuva 9). Kenttälaitteiden tiedonsiirtoon käytetään pääosin väylätekniikkaa sekä ylempien tasojen järjestelmissä IP-pohjaisia verkkoja, joita kutsutaan teollisuuden lähiverkoksi (Industrial Ethernet). Automaatiojärjestelmissä tiedonsiirron reaaliaikaisuus ja vakaus ovat tärkeimpiä prioriteetteja. [24] Teollisuuden lähiverkon kommunikointiprotokollat, kuten esimerkiksi EtherCAT, EtherNet/IP, Modbus TCP ja PROFINET käyttävät muokattua Media Access Control, MAC –kerrosta, jonka avulla saavutetaan pienet viiveet ja deterministinen vaste. Riippuen käytettävästä

protokollasta, voidaan tyypillisesti saavuttaa jopa alle yhden millisekunnin vasteaika laitteiden välillä. [25]



Kuva 9. *Väyläkaavio tehtaan automaatiojärjestelmästä. Muokattu [26]*

Ohjelmoitavat logiikat tukevat useita standardoituja tiedonsiirtoprotokollia. Logiikat kommunikoivat niin horisontaalisesti saman tasoisten laitteiden kanssa, kuin myös vertikaalisesti SCADA-järjestelmiin sekä kenttälaitteille. Tiedonsiirto logiikoilta ylöspäin pohjautuu usein teollisuuden lähiverkkoratkaisuihin. Kuvassa 10 on esitetty yleisesti käytössä olevien ohjelmoitavien logiikoiden lähiverkkoprotokollat. Kuvan mukaan käytetyimpien ja samalla tärkeimpien ohjelmoitavien logiikoiden ainoaksi suoraan yhteensopivaksi tiedonsiirtostandardiksi valikoituu OPC UA (Unified Architecture). OPC UA jatkaa siitä, mihin edellinen malli, OPC Classic, jäi, joten on odotettavissa, että standardi säilyy alalla kattavimpana laitteistoriippumattomana tiedonsiirtomenetelmänä. Näin ollen on kannattavaa rakentaa sovellus tukemaan OPC UA tiedonsiirtoa.

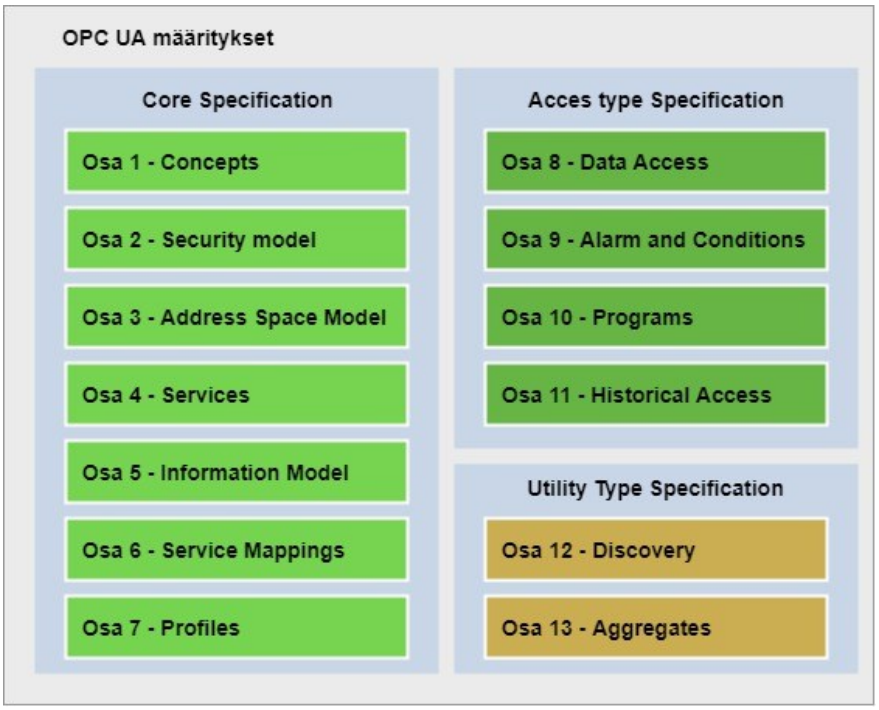


Kuva 10. *Laitteistojen tukema teollisuuden lähiverkkoprotokollat. [27-33]*

Automaatiojärjestelmien käytetyin universaali tiedonsiirtoprotokolla on vuosia ollut vuonna 1996 kehitetty Microsoftin COM/DCOM –rajapintaa hyödyntävä OPC Data Access. OPC DA yhteensopivia laitteita on tuhansia, mutta riippuvuus COM/DCOM alustasta rajoittaa laitekannan kasvua. [34] OPC on saavuttanut teollisuuden laitteiden välisessä tiedonsiirrossa ”de facto” aseman [13]. Vuonna 2006 lanseerattu täysin uusi OPC UA versio 1.0 on palvelukeskeinen arkkitehtuuri, joka sisältää aiemmat Classic OPC:n ominaisuudet, mutta tarjoaa lisäksi useita uusia ominaisuuksia irrottautuen Microsoftin alustasta. Uuden OPC version tärkeimpinä kriteereinä sen olivat skaalautuvuus, suorituskyky ja täysin laitteistoriippumaton alusta. Määrittelyvaiheessa kriteerit jaettiin kommunikointiin hajautettujen järjestelmien välillä sekä tiedon mallintamiseen. [34] OPC UA -säätiö kehittää protokollaa jatkuvasti kymmenien työryhmien toimesta. Uusin versio OPC UA versio 1.04 julkaistiin vuonna 2018. [35]

OPC UA standardi soveltuu järjestelmän vertikaaliseen ja horisontaaliseen kommunikointiin järjestelmäriippumattomasti. Asiakkaat ja palvelimet käyttävät API (Application Programming Interface) –rajapintaa tiedonvaihtoon. [36]

OPC UA tiedonsiirtostandardi IEC 62541 koostuu 13 osasta, jotka on esitetty kuvassa 11 [34]. Näistä tärkeimpiä simulointisovelluksen tiedonsiirron kannalta ovat Osa 3 osoiteavaruusmalli, Osa 4 palvelut ja Osa 5 informaatiomalli.



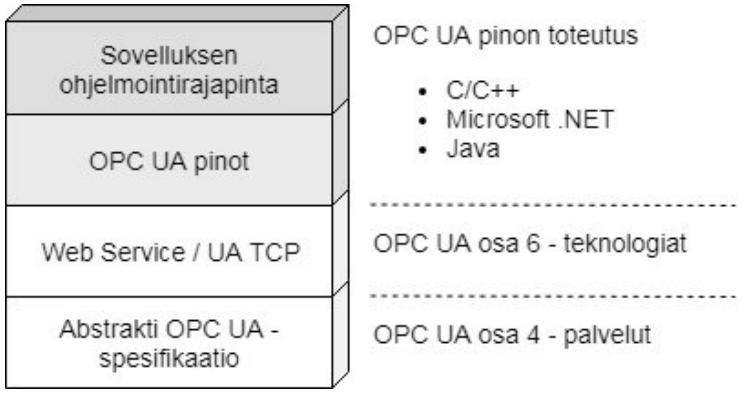
Kuva 11. *OPC UA määrittelyt. Muokattu* [37]

Osoiteavaruus mahdollistaa palvelimen olioiden (objects) esittämisen standardoidusti asiakkaalle. Se muodostuu ryhmästä tietopisteitä, jotka muodostavat verkkomaisen rakenteen sallien tietojen yhdistämisen. Osoiteavaruuden rakennuselementteinä ovat solmut (node) sekä solmujen väliset viittaukset. OPC UA määrittelee kahdeksan solmuluokkaa. Jokainen solmu avaruudessa on jonkin luokan instanssi. Luokat määrittelevät attribuutit ja viittaukset eri solmuille. Solmujen attribuutit ovat eräänlaisia määritteitä solmun dataelementeistä, kuten esimerkiksi solmun yksilöivä attribuutti *nodeid.* Asiakas pääsee käsiksi solmun attribuuttien arvoihin OPC UA:n määrittelemien palveluiden avulla, jotka ovat Read, Write, Query ja Subscription/MonitoredItem. Viittauksilla yhdistetään solmuja toisiinsa. Asiakas pääsee käsiksi viittauksien sisältöön browsing- ja querying-palveluilla. [34]

OPC Classicista tutut osamallit tiedonkäsittely (Data Access, DA), hälytykset ja tapahtumat (Alarm & Events, AE) ja historiatiedot (Historical Data Access, HDA) on OPC UA:ssa integroitu yhdeksi malliksi samaan osoiteavaruuteen (address space), jolloin niihin pääsee käsiksi palvelimen tarjoamien palvelujen kautta. [34]

OPC UA on palvelukeskeinen arkkitehtuuri, joka noudattaa palvelimen ja asiakkaan välistä palvelupyyntö/vastaus-mallia. Kaikki palvelimen ja asiakkaan välillä tapahtuva vuorovaikutus tapahtuu palvelujen avulla. [38] OPC UA määrittelyn Osassa 4 kuvatut palvelut ovat joukko abstrakteja etäkutsuja, joiden avulla tiedonsiirto palvelimen ja asiakkaan välillä tapahtuu, toisin sanoen palvelu määrittää tapahtuman tietosisällön ja operaatiot, ei toteutustapaa. Asiakas pääsee käsiksi palvelimen informaatiomallissa sijaitsevaan tietoon näiden etäkutsujen avulla. [34]

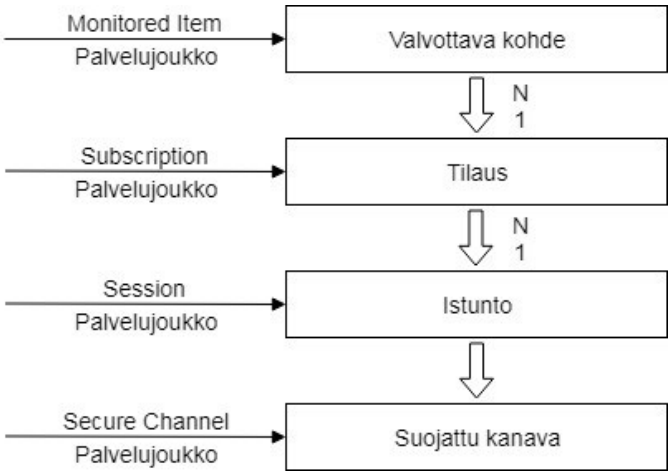
Verraten perinteiseen OPC classic tiedonsiirtoon, OPC UA irrottautuu COM/DCOM riippuvuudesta. OPC UA tarjoaa kaksi tiedonsiirtomekanismia, Web Service ja UA TCP, jotka ovat määritelty OPC UA spesifikaation Osassa 6. Lisäksi OPC UA mahdollistaa sovelluskohtaiset ohjelmointirajapinnat toteutettavaksi C/C++-, .NET- ja Java-ohjelmointikielillä. Ohjelmoitavat rajapinnat ovat määritelty OPC UA-pinossa (stack). Vapaus käytettävästä tiedonsiirrosta ja ohjelmointiympäristöstä vaatii palveluilta abstrakteja määrittelyitä, mutta mahdollistaa näiden soveltamisen ja yhdistämisen OPC UA kommunikaatiokerroksissa (Kuva 12). [34]



Kuva 12. *OPC UA kommunikaatiokerrokset. Muokattu [34]*

OPC UA:n palvelut eivät ole ainoastaan datan siirtoon, vaan iso osa palveluista on kommunikaatiorakenteen ja -kerrosten luomiseen, ylläpitämiseen ja muokkaamiseen liittyviä. OPC UA kommunikaation rakenneosat on esitetty kuvassa 13. [34]

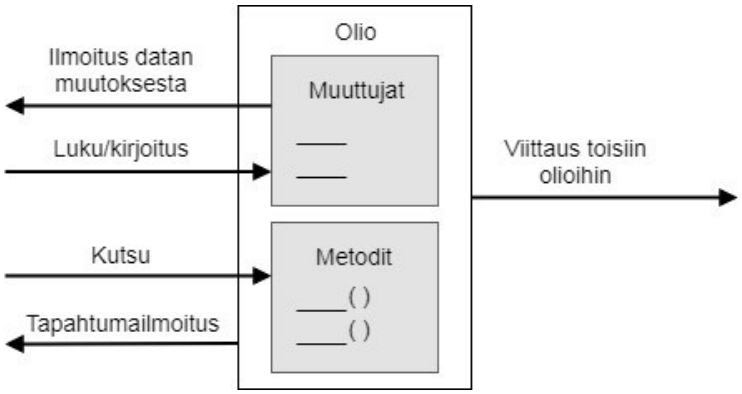
Matalan tason protokollariippuvainen tiedonsiirtokanava, Secure Channel, mahdollistaa asiakkaan ja palvelimen välisen suojatun kommunikoinnin. Suojatun yhteyden avaamisessa sille määritellään ennalta yhteyden voimassaoloaika. Yhteys uusitaan voimassaoloaikana tietoturvan ylläpitämiseksi. Suojatulle kanavalle muodostetaan istunto (Session), joka yhdistää kaksi suojatun kanavan sovellusta toisiinsa. Istunto varaa palvelimelta resursseja istunnon määrätyn voimassaolon ajan, mutta voi vapauttaa ne väliaikaisesti istunnon tultua timeout-tilaan. Istunnon aikana voidaan muodostaa useita tilauksia (Subscriptions). Tilauksilla tarkoitetaan asiakkaan ja palvelimen välistä datamuutosten ja tapahtumailmoitusten vaihtoa. Tilauksilla täytyy olla voimassa istunto, jotta tietoa voidaan välittää asiakkaalle, mutta tietoa voidaan kuitenkin välittää myös muille istunnoille. Viimeisenä, istunnon tilauksessa voidaan muodostaa yksittäisiä valvottavia kohteita (Monitored item), kuten yhden solmun yksittäisen attribuutin datamuutos tai tapahtumailmoitus. [34]



Kuva 13. *Kommunikaatiorakenne eri tasoilla [34]*

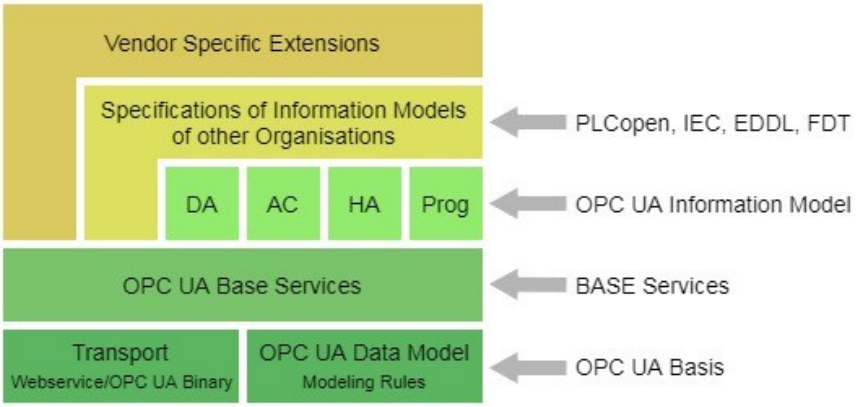
OPC UA palvelumääritykset ovat abstrakteja, tarjoten universaalin tavan laitteiden väliseen tiedon siirtoon. OPC UA määrittelyn Osa 4 palvelut, määrittelee palvelujoukkoja yhteyden muodostamiseen ja datan siirtämiseen. Protokollan määrittelemät palvelut löytyvät liitetiedosta (LIITE A) [34]

OPC UA tarjoaa palvelimille yhtenäisen menetelmän esittää informaatiota asiakkaalle alustariippumattomasti. Tarkoitusta varten on määritetty oliomalli, jonka avulla kuvataan palvelimella olevan osoiteavaruuden rakenne ja määritellään palvelimen dataoliot. [38] Dataoliot rakentuvat muuttujista (variables), menetelmistä (methods) ja tapahtumista (events). Muuttujat esittävät olion arvoa, jota voidaan lukea ja kirjoittaa sekä saada arvon muuttumisesta ilmoitus. Menetelmä on eräänlainen funktio, jota asiakas voi kutsua ja saada vastauksena paluuarvon. [34, 38]



Kuva 14. *OPC UA oliomalli* [37]

OPC UA:n informaatiomallin avulla voidaan tiedon semantiikka esittää tehokkaasti. Kanta-informaatiomalli tarjoaa ainoastaan infrastruktuurin tiedon mallintamiseen, mutta malli on laajennettavissa. Näin kolmannet osapuolet, kuten laitevalmistajat voivat luoda omia mallejaan osaksi OPC UA:ta. Informaatiomalli tukee olio-ohjelmoinnin periaatteita, mikä helpottaa tiedon esittämistä ja käsittelyä. Kuvassa 15 on esitetty informaatiomallin rakenne sekä täydentävät tietomallit. [34]

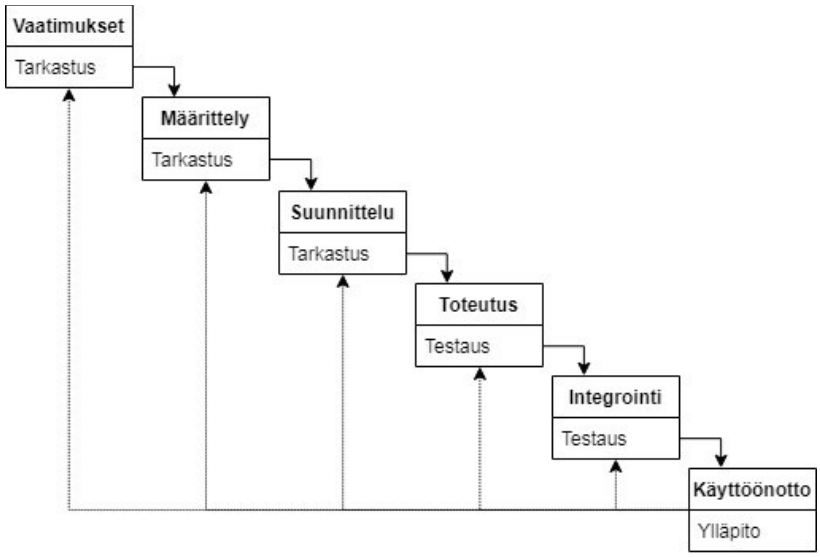


Kuva 15. *OPC UA informaatiomallin rakenne. Muokattu* [37]

2.4 Ohjelmistotuotanto

Ohjelmistotuotannolla tarkoitetaan ohjelman toteuttamista vaatimuksista valmiiksi, testatuksi sovellukseksi. Ohjelmistoprojektin vaiheita voidaan kuvata vesiputousmallin avulla, joka kuvaa projektin elinkaaren vaiheet: vaatimukset ja määrittelyt, suunnittelu, toteutus, integrointi, ja käyttöönotto. (Kuva 16) Lisäksi alalla yleisesti tunnetulla Unified Modeling Language- kaaviolla (UML) on ohjelmistotuotannossa suuri rooli ohjelmistoprojekteissa.

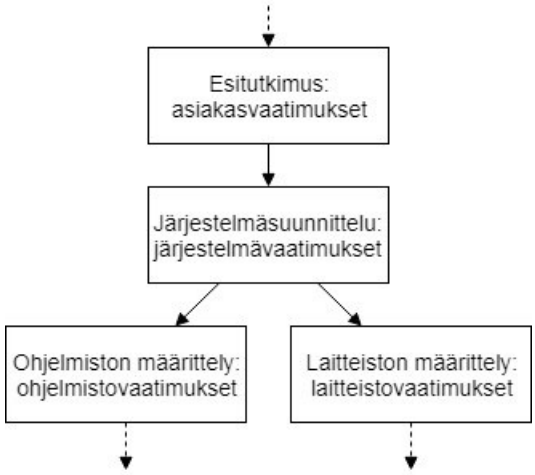
Projektien lähtökohtana asiakkaalla on usein tarve saada tuote, jonka tarkoitus on tuottaa arvoa asiakkaalle. Jotta tuote voisi tuottaa lisäarvoa asiakkaalle, tarvitaan havaitusta ongelmasta esitietoja. Monet automaatio- ja ohjelmistoalan projektit koostuvat sekä järjestelmästä että ohjelmistosta. Kuvassa 17 on havainnollistettu asiakasvaatimusten jakaantuminen ohjelmistoprojekteissa.



Kuva 16. *Ohjelmistosuunnittelun vesiputousmalli. Muokattu [4]*

Ohjelmistotuotanto alkaa toteutettavan ohjelmiston vaatimusten määrittelystä. Ohjelmiston tavoitteiden selvittämiseen käytetään niin kutsuttua käyttötapausmallia (use case). Määrittelyvaiheessa ei oteta kantaa ohjelman sisäiseen toimintaan, komponentteihin tai toteutukseen. Ohjelmiston määrittely voidaan kategorioida FURPS+ mallin mukaan, jolloin tärkeimmät ja yleisimmät ongelmat tulee selvitettyä. Selvitykseen kuuluu muun muassa tehokkuus, jossa määritellään vasteaika ja määritellään hyödykkeiden käyttö. Toiminnollisuus, jossa määritellään ominaisuudet sekä kapasiteetti. Toteutuksen vaatimukset, joissa selvitetään käytettävät standardit. Lisäksi malliin kuuluvat luotettavuus, käytettävyys, tuettavuus ja rajapinnat. [39]

Suunnitteluvaihe voidaan jakaa kahteen eri vaiheeseen: arkkitehtuurisuunnitteluun ja moduulisuunnitteluun. Arkkitehtuurisuunnittelussa järjestelmä pyritään osittamaan niin pieniin osiin, eli moduuleihin, että sen jokainen osa voidaan kuvata ohjelmistotason välineillä. Ohjelma ositetaan abstraktioiden mukaan: toiminto, tieto ja tyyppi. Rajapinta abstraktioon näkyy käyttäjälle, mutta sen sisään koteloitu toteutus ei. Yksittäisten moduulien toiminnallisuutta voidaan muokata ilman, että koko ohjelmisto kärsii. Moduulisuunnittelussa suunnitellaan moduulien sisäiset ominaisuudet ja toiminnallisuudet, joiden avulla moduuli toteuttaa sille määrätyt operaatiot. [4]



Kuva 17. *Asiakasvaatimukset ohjelmistoprojekteissa. Muokattu [4]*

UML -notaatioita käytetään tuotteen määrittelyyn, visualisointiin, rakentamiseen ja dokumentointiin. UML on laajasti käytössä ohjelmistoalalla, mutta myös liiketoiminnan ja muiden ei-ohjelmistojen suunnittelussa.[39] Notaatioiden eli kaavioiden avulla pyritään visuaalisesti helpottamaan osapuolten välistä kommunikaatiota, vaihtamaan ajatuksia sekä esittelemään suunnitelmia muille. Kaaviot ovat universaali työkalu esittämään kokonaisuutta asiaan perehtymättömälle aina ohjelmakoodin toteuttamisen yksityiskohtiin saakka. [40] UML kuvauskieltä on laajennettu ajan saatossa kattamaan yhä useampia käyttötarpeita sisältäen 14 erilaista kaaviomallia [41]. Alla olevissa kappaleissa esitetään diplomityössä käytetyt kaaviomallit.

Käyttötapauskaavioiden (use-case diagram) ensisijainen tarkoitus kartoittaa asiakasvaatimukset ja implementoida niistä ohjelmakuvaus. Järjestelmän toiminnan esittäminen joukkona käyttäjien suorittamia tapahtumaketjuja. Käyttötapauksessa käyttäjät esitetään käyttäjärooleina, jotka järjestelmän ulkopuolelta tietyillä toimillaan aiheuttavat käyttötapauksen, jonka tuloksen järjestelmä tuottaa käyttäjälleen lisäarvoa. Käyttötapauskaavioilla, kuten useilla muillakin UML-toteutuksilla on käytettävissä erinäisiä laajennuksia, mutta näiden kanssa tulee olla varovainen, jotta kaavioiden todellinen tarkoitusperä ei unohtuisi, eli tuottaa selkeyttävä ja ymmärrettävä kokonaisuus, jonka avulla vaatimukset täytetään. [4, 39]

Luokka (class) on yksi abstrakti kokonaisuus, joka sisältää attribuutteja ja metodeja. Attribuuteilla kuvataan luokan muuttujien nimiä, jotka sisältävät informaatiota. Metodeilla tarkoitetaan luokan toteuttamia operaatioita. Yhteys luokkien välillä esittää keskinäiset rajapinnat ja lukumääräsuhteilla kuvataan montako luokan ilmentymää, eli oliota, yhteyden vastapäässä kyseiseen luokkaan voi liittyä. Koostuminen; Olio on toisen olion komponentti, ts. olio koostuu sen oliosta (moottori on auton komponentti, työntekijä kuuluu osastoon ja osasto yritykseen). Yleistäminen; eräänlainen aliluokka oliosta. Attribuutit ja metodit kopioituvat ja aliluokka voi myös luoda ja muokata näitä itselleen lisää. Yleistetyn olion tyyppi ei voi muuttua enää. Luokkakaaviolla (class diagram) kuvataan sovelluksen luokkien keskeisiä käsitteitä ja niiden riippuvuuksia. [4]

Tapahtumasekvenssikaavioilla (sequence diagram) kuvataan erilaisia tapahtumaketjuja prosessin etenemisestä. Tapahtumaketjussa voi esiintyä myös toimintoja kuten algoritmikuvauksissa. Tapahtumasekvenssiin osallistuu usein useita olioita, joiden kautta prosessi kulkee alusta loppuun. Sekvenssikaavion tapahtumien väliset yhteydet voivat olla synkronisia tai asynkronisia. Synkronisessa kommunikoinnissa tapahtumalle tulee jokin vastaus tai paluuarvo, kuten funktiokutsussa. Asynkroninen kommunikointi ei ota kantaa ohjelman jatkosta. Tapahtumasekvenssikaaviossa voidaan esittää sen sisältämien olioiden tilakaavion tila, mikäli sellainen on laadittu. Tämä helpottaa sekvenssin tarkastamista ja selkeyttämistä. [4]

Suunnitteluvaiheen jälkeen vesiputousmallin mukaisesti (Kuva 16) ohjelmiston toteutuksessa ositetut moduulit toteutetaan itsenäisinä. Moduulin sisäiset tietorakenteet on koteloitu moduulin sisään ja niiden käyttäjille annetaan funktiorajapinta, jonka avulla niitä voidaan käsitellä. Ohjelmien muuttujat valitaan siten, että muutokset eivät vaikuta moduulien toimintaan tai niiden rajapintoihin. Elinkaaren seuraavassa vaiheessa integroidaan toteutetut komponentit osaksi suurempaa toiminnollisuutta, jonka jälkeen käyttöönottovaiheessa viimeistellään koko ohjelmiston toiminta kokonaisuutena. [4]

2.5 Ohjelmistojen testaus

Tietotekniikan lisääntyessä sovellukset ovat monimutkaistuneet, jolloin suunnittelun hallinta on tullut haasteellisemmaksi. Kova kilpailu ja tiukentuneet aikataulut ovat johtaneet

ohjelmistotuotannon tehostamiseen, jonka seurauksena laadun varmistaminen kireissä aikatauluissa on noussut tärkeäksi kysymykseksi. Ohjelmavirheet tuotantolaitoksen automaatiossa voivat aiheuttaa merkittäviä henkilö-, ympäristö- ja materiaaliriskejä. Ohjelmistojen laadun varmistaminen on haastavaa, sillä suunnitteluun liittyvät inhimilliset virheet, jotka eivät ole ennustettavissa. Ohjelmistot eivät tee virheitä, toisin kuin ihmiset, jotka ovat ohjelmistot suunnitelleet ja toteuttaneet. [2]

Kappaleessa esitetään automaatiosovellusten ohjelmistotestauksen keskeisiä periaatteita. Automaatioprojekteissa ohjelman testaaminen aloitetaan toteutusvaiheessa, ja jatkuu aina hyväksymistestaukseen (SAT) asti. Testaamisen tavoitteena on taata ohjelman tehokas ja turvallinen toiminnallisuus vastaten asiakkaan määrittelyitä. Testaaminen on osa automaatioprojektin laadunvarmistusta niin itsenäisen suunnittelijan omatoimisena työkaluna kuin asiakkaalle toimitettavana laatudokumentointina.

Ohjelman testaus voidaan jakaa eri vaiheisiin: moduulitestaus, integrointitestaus ja järjestelmätestaus. Moduulitestauksessa testataan automaatiosovelluksen yksittäisiä komponentteja, kuten esimerkiksi lohkoja. Integrointitestauksessa testataan yhteen liitettyjen moduulien rajapintojen toimintaa ja kokonaisia toiminnallisuuksia. Järjestelmätestaus kattaa automaatiojärjestelmän sekä ohjelman toiminnallisuuden ja suorituskyvyn. Järjestelmätestaus suoritetaan tästä syystä laitteistolla, joka projektiin toimitetaan. Yhteistä kaikille testaustoimille on, että testaukset suoritetaan aina ennalta määrätyn testaussuunnitelman mukaisesti. [2]

Ohjelmistotestaus jakaantuu lisäksi staattisiin analyyseihin ja dynaamisiin testauksiin. Staattisten analyysien avulla katselmoidaan ohjelmasovellus ilman ohjelman suorittamista. Analyysissä otetaan kantaa ohjelman rakenteeseen. Dynaamisissa testauksissa ohjelma suoritetaan kokonaisena tai osissa. Testauksissa annetaan ohjelmalle syötteitä ja tulkitaan, vastaako niiden toiminta haluttua lopputulosta. [42]

Automaatiosovelluksen ohjelmistotestauksessa ongelmia aiheuttavat reaaliprosessien tilojen jatkuvat ja nopeat muutokset. Prosessilta saatavat tilat saattavat muuttua jokaisella ohjelmakierrolla ja näin ollen koko järjestelmän testaaminen simuloimalla on haastavaa. Ulkoisella simulointisovelluksella pyritään helpottamaan koko järjestelmän testaamista.

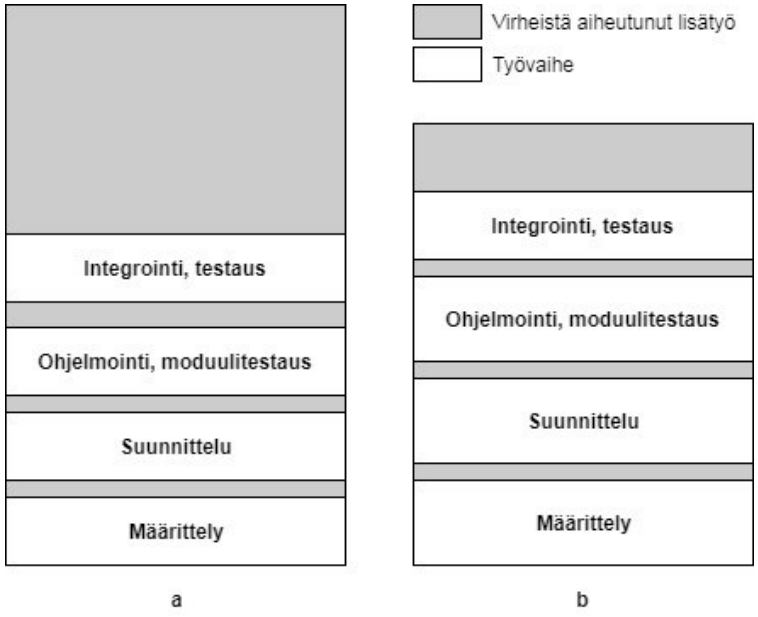
Vaativien ja monimutkaisten ohjelmien toteutus aiheuttaa haasteita ohjelmien suunnittelussa ja testaamisessa. Erityisesti testaaminen kuluttaa resursseista enemmän kuin 50%. [1] Ohjelmien testaamisessa täytyy kuitenkin muistaa, että monimutkaisissa ja isoissa ohjelmissa mahdollisia tiloja voi olla niin paljon, että täydellinen testaaminen voi olla mahdotonta. [40]

Ohjelmoitavien logiikoiden ohjelmistojen testaamiseen on kehitetty erilaisia työkaluja, mutta ongelmana on logiikkaohjelmien erilaiset metodologiat ja semantiikat. Logiikkaohjelmien eroavaisuuksista ja kehityksestä johtuen testaussovellukset tulevat aina hieman

jäljessä. Testaussovelluksien ominaisuuksissa on tärkeää niiden saavutettavuus ja käytettävyys, jotta testaaminen olisi helppoa ja tehokasta. [1]

Perinteisesti automaatioprojekteissa määrittely, laitteistosuunnittelu ja ohjelmasuunnittelu ovat edenneet perättäisissä järjestyksissä. Ohjelmasuunnittelija on voinut aloittaa prosessiohjelmoinnin, optimoinnin ja testaamisen vasta sitten, kun laitteistosuunnittelija on päässyt omassa työssään päätökseen. Tehokkaampaa hallintaa on suunnittelijoiden tapa toimia yhteistyössä. Laitteistosuunnittelija jakaa ohjelmistosuunnittelijan kanssa tarpeelliset tiedot, jolloin ohjelmisto voidaan toteuttaa testauskuntoon riippumatta laitteistosuunnittelusta. Tapaa kutsutaan termillä ”virtual comissioning”. [13]

Arviointi, katselmus ja tarkastus ovat osa yrityksen laatujärjestelmää. Arviointi tarkoittaa laadunvarmistusta (audit), jonka suorittaa kohteesta riippumaton taho. Katselmus tarkoittaa epävirallisempaa, projektin sisäistä arviointia. Katselmuksen tarkoituksena on saattaa vaihe päätökseen ja tuottaa etapin dokumentit. Tarkastuksilla pyritään todentamaan etapin tuotteen täyttävän laatukriteerit. Kun tuote läpäisee tarkastuksen, voidaan siirtyä seuraavaan projektivaiheeseen. Virheet jotka läpäisevät tarkastuksen, tai joihin ei puututa ovat ongelma jatkossa. Niiden korjaamiseen kuluu resursseja moninkertaisesti (Kuva 18). [4]



Kuva 18. *Virheiden aiheuttamat lisätyöt projektissa. a) projekti ilman tarkastuksia ja b) projektivaiheet tarkastusten kera. Muokattu [4]*

Testaustasot

Ohjelmistojen testaus jaetaan ohjelmistotuotannossa perinteisesti V-mallin mukaisiin kehitystasoihin; yksikkötestaukseen, integrointitestaukseen ja järjestelmätestaukseen. [40] Yksikkötestaus on ohjelmistojen tuotannossa yleisimmin käytetty testausmuoto, joka tun-

netaan myös moduulitestauksena. Yksikkötestauksella tarkoitetaan yksittäisen komponentin, kuten moduulin funktion lohkon tai olion testaamista. Testaustoimenpiteitä suoritetaan usein sitä mukaa, kun valmiita komponentteja saadaan ohjelmoitua. Testauksen päämääränä on varmistaa, että juuri toteutettu ohjelmakomponentti täyttää sille määrätyn toiminnallisuuden ja se voidaan suorittaa ilman virheitä. [40]

Yksittäisten komponenttien testaamisessa ongelmia aiheuttaa usein komponentin vaatiman rajapinnan tietojen puutos. Komponenttien testaamiseen voidaan käyttää apuna erillisiä testauskomponentteja ja testitynkiä, joiden tarkoitus on antaa testattavalle komponentille tarvittavat rajapinnan parametrit. Testikomponentteja käytetään avuksi myös, kun halutaan testata ison ohjelmiston yhtä komponenttia. Kokonaista prosessinohjausjärjestelmää ei välttämättä ole mahdollista suorittaa ohjelmointiympäristössä tai virtuaalikoneella, joten testikomponenteilla luodaan vastaava komponenttiin vaikuttava ympäristö. Lisäksi testikomponenteilla voidaan simuloida sellaisia häiriötilanteita, joita varsinaisille komponenteille on haastava toteuttaa todellisessa järjestelmässä, mutta jotka ovat kuitenkin mahdollisia. [40]

Integrointitestauksessa yksittäisten, jo testattujen ja toimivaksi todettujen, ohjelmakomponenttien jälkeen järjestelmää rakennetaan sovittamalla moduuleja yhteen. Tavoitteena on saada yhtenä toimiva kokonaisuus. Integrointitestausta suoritetaan pala kerrallaan, liittämällä moduuleja testattuun kokonaisuuteen. Kun toiminta lisätyn moduulin kanssa toimii, voidaan liittää uusi moduuli ja niin edelleen. Yhteen sovitettujen moduulien ja tehtyjen testausten lopputuloksena voidaan todeta laajojen ohjelmakokonaisuuksien toiminnallisuus ja käyttäytyminen erilaisissa virhetilanteissa. Kuten yksikkötestauksissa, myös integrointitestauksissa puuttuvat kytkennät ja rajapintatiedot aiheuttavat testiongelmia. Näitä varten tarvitaan rajapinnan simulointia, joka voidaan suorittaa esimerkiksi erilaisilla testiohjelmilla. Testiohjelmat toteutetaan ainoastaan testausta varten ja aiheuttavat usein isoja kustannuksia integrointitestauksessa. [40]

Osien integrointi voidaan toteuttaa kolmella eri tavalla. ”Alhaalta ylöspäin” -mallissa (bottom up) integrointi aloitetaan matalimman tason komponenteista, jotka kommunikoivat suoraan laitteiston rajapinnan kanssa, kuten esimerkiksi I/O:n, tietokannan ja verkkoyhteyden. Järjestelmää rakennetaan moduuli kerrallaan kohti järjestelmähierarkian huippua. Etuna on matalan tason virheiden havaitseminen. Päinvastaisesti ”ylhäältä alaspäin” -mallissa (top down) integrointi aloitetaan järjestelmähierarkian huipulta, eli hienoimmista toiminnallisuuksista, kohti alimman tason moduuleja. Etuna on paremman toiminnallisen yleiskuvan saaminen aikaisemmassa vaiheessa ja toimintojen lisääminen. Voileipätestauksena (sandwich) tunnetussa mallissa integrointi aloitetaan järjestelmähierarkian keskeltä ja komponentteja liitetään yhteen edeten hierarkkisesti molempiin suuntiin. Etuna on testimoduulien ja testitynkien vähäisempi tarve, mutta toisaalta haasteellisempi komponenttien yhteensovittaminen. [40]

Järjestelmätestauksessa aikaisempien kehitystasojen jälkeen ohjelmistot ja laitteistot on rakennettu toimimaan kokonaisuutena. Yksikkö- ja integrointitestauksen jäljiltä ohjelmista on poistettu kaikki testikomponentit ja tynkäsovellukset. Järjestelmätestauksessa koko järjestelmä kootaan yhteen ja testataan kokonaisuuden toimintaa. Järjestelmätestaus ei tarkoita mitään tiettyä testaustapaa eikä siinä tuoteta välttämättä mitään uutta, vaan käytännössä vaihe tarkoittaa kaikkea testausta mitä kokonaiselle järjestelmälle tehdään. Järjestelmätestauksessa testausmenetelmiä on useita, riippuen järjestelmästä ja tarpeesta. Yleisiä testauksia ovat ainakin musta- ja lasilaatikkotestit, joiden avulla pyritään todentamaan järjestelmän toiminnallisuus. Järjestelmälle voidaan tehdä myös käyttäjätestejä, kuormitustestejä, I/O-testejä sekä monia muita testauksia. Usein järjestelmätestauksessa löydetään virheitä vielä komponenttitasolta, eivätkä asiakkaan muutostarpeet ole kovinkaan epätavallisia. [40]

Järjestelmätestaus suoritetaan toimittajan tiloissa testiympäristössä. Suunnittelija voi suorittaa järjestelmätestauksia itsenäisesti tai asiakkaan kanssa, jolloin vaihetta kutsutaan usein tehdastestaukseksi. [40]

Testausmenetelmät

Musta laatikko -testaus (black box testing) on käytetyin testausmuoto, jossa kuvitteellisesti mustalle laatikolle annetaan testitapauksissa määriteltyjä syötteitä ja katsotaan minkälaisia tuloksia laatikko antaa. Testauksessa laatikon sisällä tapahtuvia ei-toimintoja ei noteerata, vaan arvioidaan ainoastaan ulkopuolelle nähtäviä tapahtumia. Testitapauksissa määritellään etukäteen toimintakuvauksen mukaan, miten laitteen tulisi reagoida tietynlaisiin syötteisiin ja tehtäväsarjoihin. [40] Testausmenetelmää voidaan käyttää kaikissa ohjelmiston elinkaaren kehitysvaiheissa. Testausmenetelmän yksinkertaisen käytön, mutta toisaalta monivaiheisen testaamisen vuoksi musta laatikko -testausmenetelmää automatisoidaan erilaisin testausautomaatioiden avulla. [43]

Lasilaatikkotestauksella (glass box/white box/clear box testing) voidaan testata vastaavilla testitapauksilla kuin musta laatikko -testauksessa, mutta lähestymistapana on tutkia myös, mitä laatikon sisällä tapahtuu kyseisillä hetkillä. Testauksessa pyritään selvittämään, miten syötettä käsitellään lähdekooditasolla laatikon sisällä, jotta haluttu lopputulos saavutetaan. Lasilaatikkotestaus vaatii testauksen tekijältä ohjelmointiymmärrystä, jotta voidaan päätellä ohjelman toimineen oikein, eikä lopputulos ollut vain sattumaa. Lasilaatikkotestauksella voidaan todeta, että ohjelma toimii lähdekooditasolla, kokeilematta kaikkia mahdollisia syötteitä. Etuina ovat myös testaajien tarkka perehtyminen sisäiseen toimintaan, jolloin mahdolliset virheet löytyvät. Tämä on lasilaatikon etu verrattuna mustaan laatikkoon. Lasilaatikkotestaus ei ota kuitenkaan kantaa ohjelman puutteellisesta vaatimusten määrittelystä, joista aiheutuu ominaisuuksien puutteita. Tästä johtuen lasilaatikkotestaus ei voi olla laadunvarmennustestauksessa ainoa testausmenetelmä. [40]

Staattisella testaamisella tarkoitetaan järjestelmän testaamista nimensä mukisesti staattisessa tilassa, toisin sanoen ohjelmaa ei suoriteta. Ohjelmaa tarkastetaan esimerkiksi ohjelma-arviointien, -katselmuksien tai arkkitehtuurisuunnittelun näkökulmasta. Staattinen testaus on perustason tarkastus, jonka tarkoituksena on poistaa ilmiselvät virheet, kuten esimerkiksi syntaksivirheet. Staattista testausta voidaan käyttää esimerkiksi lasilaatikkotestauksessa arvioiden laatikon sisäistä toimintaa. Suunnittelija suorittaa staattista testaamista jatkuvasti ohjelman edetessä tarkastellen ohjelman rakennetta ja toimintaa. [40]

Dynaaminen testaus taas tarkoittaa ohjelman testaamista ohjelmaa suorittaessa. Ohjelman toimintaa testataan käyttäen testattavaa järjestelmää ja arvioimalla sen reagointia annettuihin syötteisiin. Dynaamista testaamista voidaan näin ollen suorittaa vain ohjelman ollessa vaiheessa, jossa käännös onnistuu. Dynaamista testaamista käytetään usein yksikkö-, integrointi- ja järjestelmätestauksissa, jossa tarkastetaan ohjelman reaktiot käytännössä. [40]

Mallipohjaisessa testissä (Model based testing) testitapaukset ja suoritettavat testit valitaan siten että ne tarkastavat, onko ohjelma toteutettu määrittelyjen mukaisesti ja täyttääkö se esitetyt toiminnallisuudet. Ohjelmasta luodaan ensin suunnittelumalli, esimerkiksi UML kuvauksilla, jotka kuvaavat ohjelman toimintoja. Mallista luodaan tämän jälkeen testitapauksia, jotka todentavat ohjelman toiminnallisuuden. Ohjelma, testitapaukset ja suunnittelumalli vastaavat näin toisiaan. Tällaisia voi olla esimerkiksi kokonaisvaltaiset testisimulaattorit. [40]

Ohjelman toiminnallisten testien lisäksi voidaan toteuttaa säätöteoreettisia malleja, joiden avulla mallinnetaan systeemin dynamiikkaa. Säätöteoreettisten mallien avulla voidaan testata säätimien toimintaa ja viritystä sekä järjestelmän käyttäytymistä. Prosessin vastetta voidaan mallintaa diskreetin tilamallin avulla, jolloin ohjelmistojen testaus voidaan toteutusvaiheessa saattaa vielä pidemmälle ja testata pitkän aikavälin suorituskyky lyhyessä ajassa [13]. Diskreetit tilamallit voidaan jakaa erilaisiin tasoihin testauksen yksityiskohtien tarkkuuden mukaan, kuten esimerkiksi dynaamisiin ja ei-dynaamisiin ominaisuuksiin. [44]

Testausdokumentointi ja laatu

Ohjelmistoja voidaan testata suunnittelijan toimesta pitkin ohjelmiston kehityskaarta omana laadunvarmistuksena, mutta kehitysvaiheiden testauksista on saatava myös testausdokumentaatioita, jotta laatu ja kriteerien täyttymys voidaan osoittaa myös asiakkaalle. Testausraportoinnin optimaalista kattavuutta havainnollistaa Taulukko 3, jossa listataan dokumentointi järjestelmän jokaiselle vaiheelle.

Automaatioprojektin toimitus on pitkä projekti, jonka toteutunut lopputulos nähdään myöhään. Välikatselmukset ovat tärkeitä edistymisen seuraamiseksi ja virheiden korjaa-

miseksi. Ohjelmistojen testaaminen ja testausdokumentointi tärkeää, mutta osittain ongelmallista sen epäyhtenäisten menettelyiden takia. Näiden epäyhtenäisten menettelyiden vuoksi automaatiojärjestelmät eivät ole aina riittävän testattuja ja valmiita, kun käyttöönotto aloitetaan. Virheiden korjaamiseen ja aiheutuneisiin viivästyksiin kuluu rahaa. [2]

Taulukko 3. *Ohjelmistotuotannon testausraportit [2]*

Elinkaaren vaihe	**Dokumenttilaji**	**Toiminto/kohde**
Toiminnan määrittely	Testimäärittely	Kelpuuttaminen, tehdastesti
	Testimäärittely	Kelpuuttaminen, asennuspaikkatesti
Järjestelmäsuunnittelu/järjestelmätoteutus	Testimäärittely	Ohjelmistointegrointi
	Testimäärittely	Ohjelmisto-/laitteistointegrointi
Moduulisuunnittelu	Testimäärittely	Moduuli
	Testiraportti	Moduuli
Järjestelmäintegrointi	Testiraportti	Ohjelmistointegrointi
	Testiraportti	Ohjelmisto-/laitteistointegrointi
Kelpuuttaminen, tehtaalla	Testiraportti	Järjestelmän kelpuutus, tehdastesti
	Testipäiväkirja	Järjestelmän kelpuutus, tehdastesti
Kelpuuttaminen, asennuspaikalla	Testiraportti	Järjestelmän kelpuutus, asennuspaikkatesti
	Testipäiväkirja	Järjestelmän kelpuutus, asennuspaikkatesti

Laatu ei ole yksikäsitteinen termi. Laatu käsitteenä voi tarkoittaa useaa asiaa riippuen sen määrittelijästä. Kielellisesti laadun määritelmät kuvataan erinomaisuutena. Siirryttäessä enemmän konkreettiseen laatuun jaetaan se usein subjektiiviseen, objektiiviseen ja arvioimattomissa oleviin komponentteihin. Subjektiivinen laatu saadaan tutkimalla vaatimusten täyttymistä ja toteutumista, objektiivinen laatu asiakastyytyväisyydellä ja arvioimattomissa oleva laatu tulevaisuudessa tapahtuvista tekijöistä. [4] Automaatiosovelluksen testaamisessa pyritään vaikuttamaan tuotteen laatuun pääosin subjektiivisesti: sovellusta testataan, jotta se täyttäisi asiakkaan vaatimukset. Toisaalta, laaturaportoinnilla pyritään nostamaan objektiivista laatua, sillä asiakas näkee näin konkreettisesti, että laatuun on panostettu ja tuotteen kuva paranee asiakkaan näkökulmasta. Arvioimattomissa olevaan laatuun ei sovelluksen testaamisella ole juurikaan merkitystä. Arvioimaton laatu syntyy pääosin suunnitteluvaiheessa ja sen toteutuminen nähdään prosessin toiminnallisella vakaudella ja joustavuudella.

Ohjelmistot ovat monimutkaistuneet ja laadunhallinta ja -varmistus ovat entistä tärkeämpiä. Laatua tuotetaan ohjelmistojen toteutuksella hyviksi todetuilla ratkaisuilla, mutta myös testauksilla, katselmuksilla ja tarkastuksilla. Laadunvarmistuksella pyritään varmistamaan, että vaaditut laatutekijät toteutetaan. Projektin testaukset, katselmukset ja tarkastukset dokumentoidaan, jotta projektin ulkopuolinen taho voi toteuttaa laadunvarmistusta. [2]

Tehdastestaus

Automaatiojärjestelmän tehdastestaus pidetään toimittajan tiloissa asiakkaan kanssa yhteistyössä ja se kattaa kenttälaitteita lukuun ottamatta koko automaatiojärjestelmän kokoonpanon testaamisen. Toimittajan suorittaman tehdastestauksen tavoitteena on osoittaa ja esittää dokumentoituna, että laitteisto ja ohjelmisto täyttävät tilausvaiheessa sovitut vaatimukset. Tehdastestaus kattaa toiminnallisten vaatimusten viankäsittelyn ja toipumisen, sekä todistaa kommunikoinnin, yhteensopivuuden, järjestelmätuen ja käyttöliittymien toimivuuden. [6] Hyväksytyn tehdastestauksen jälkeen asiakas myöntää järjestelmälle toimitusluvan, jonka jälkeen järjestelmä on valmis toimitettavaksi. [2]

Tehdastestaus toteutetaan toimittajan ennalta laaditun, asiakkaan hyväksymän testaussuunnitelman mukaisesti. Tehdastestausta varten järjestelmä kootaan asennussuunnitelmien mukaiseen kokoonpanoon toimittajan tiloihin. Toimittajalla on oltava esittää asiakkaalle projektin vaadittu dokumentointi, joka sisältää laitteistojen tekniset-, asennus- ja testausdokumentit. Testauksen alussa tarkastetaan laitteiston vaatimuksenmukaisuus ja yhdenmukaisuus dokumentointiin. [2]

Laitteistojen tarkastusten lisäksi tehdastestauksessa tarkastetaan ohjelmistojen toimintakuvauksen mukainen toiminnallisuus. Ohjelmistojen testaukseen on laadittu asiakkaan kanssa testaussuunnitelma, jonka avulla voidaan todeta sovellusten täyttävän annetut kriteerit. Ohjelmistojen toiminnalliseen testaamiseen käytetään apuna manuaalista tai automaattista kenttädatan simulointia. Testauksessa tehdään myös poikkeamatestejä, jossa simuloidaan vaurioita tai ulkoisia häiriöitä. Poikkeamien testaaminen myöhemmissä vaiheissa saattaa olla vaikeaa ja aiheuttaa vaaratilanteita. [2]

Ohjelmistojen testaamisen yhteydessä tarkastetaan kaikki valvomonäkymät ja niiden toiminta. Tehdastestauksessa asiakas usein näkee toimitettavan järjestelmän ensi kertaa toiminnassa ja muutostarpeita saattaa syntyä. Tehdastesteihin tulee osallistua myös asiakkaan käyttöhenkilökuntaa (operaattoreita ja prosessisuunnittelijoita), jotta järjestelmän käytettävyyteen liittyvät puutteet voidaan korjata ennen siirtoa. Sekaannuksen minimoimiseksi koulutukset pidetään kuitenkin erillään varsinaisesta testauksesta. Mikäli rakennetaan ja toimitetaan pakettiyksikköä tai muuta kokonaisuutta, voidaan testit suorittaa toimittajan tiloissa myös pidemmälle. Kuten esimerkiksi kylmäajot asiakkaan raaka-aineilla. Useimmiten prosessilaitteistoa ei kuitenkaan ole käytettävissä. [2]

Tehdastestaus on usein haastava ja pitkä prosessi. Ennalta laaditut ja tarkat testaussuunnitelmat auttavat systemaattisen testaamisen hallinnassa ja organisoinnissa. Ennen tehdastestausta suoritetaan usein integrointitestit, jotka auttavat tehdastestauksen valmistelussa. [2]

Tehdastestaus vie paljon resursseja niin toimittajalta kuin asiakkaaltakin, mutta panostaminen kattavaan tehdastestaukseen tuo useimmiten säästöjä koko projektin

elinkaaressa. Vianmääritykset ja ongelmien korjaamiset ovat aina huomattavasti helpompi toteuttaa tehdastestauksen aikana kuin käyttöönottovaiheessa, jossa ympäristön paine on huomattavasti suurempi. [6]

3. SIMULOINTI AUTOMAATIOSOVELLUKSEN TESTAUKSESSA

Tämä on osio simulointiohjelmiston suunnittelusta. Tarkoituksena on suunnitella ohjelmisto, jonka avulla voidaan ohjelmoitavan logiikan muuttujia simuloida erillisen simulointisovelluksen avulla. Simulointisovellusta käytetään apuna automaation ohjelmistotestauksessa. Kappaleessa simulointiohjelmistolla tarkoitetaan simulointiin tarvittavien ohjelmistojen, kuten esimerkiksi käyttöliittymän, ohjelmoitavan rajapinnan ja OPC UA pinon muodostamaa kokonaisuutta. Simulointisovelluksella taas tarkoitetaan toteutuksen ylintä kerrosta, jossa sijaitsevat käytettävyyteen liittyvät ominaisuudet.

3.1 Simulointiohjelmiston lähtökohdat

Ohjelmoitavan logiikan simulointisovelluksen käytettävyyttä ja näin ollen hyötyjen saavuttamista edellyttää sovelluksen saavutettavuus ja helppokäyttöisyys. Automaatiojärjestelmät ja ohjelmoitavien logiikoiden sovellukset ovat usein projektikohtaisia, joten simulointisovelluksen täytyy mukautua yksilöllisesti jokaista projektia vastaavaksi. Testattavan automaatio-ohjelman ei tule sisältää testausta varten rakennettuja komponentteja.

Automaatio-ohjelmiston kenttärajapinnan simulointiohjelmisto suunnitellaan ohjelmistokehityksen kehitystasojen mukaisesti, jotka ovat esitetty kappaleessa 2.4. Seuraavissa kappaleissa esitetään simulointiohjelmiston suunnittelun vaiheet tarkemmin.

Määrittely

Simulointisovelluksen rakentaminen aloitetaan määrittelemällä sovellus kappaleessa 2.4 esitetyn FURPS+-osituksen mukaan. Luonnehdinta osituksen mukaisesti auttaa luomaan selkeät tavoitteet ja määrittelyt sovellukselle. Lisäksi erilliset tavoitteet auttavat muodostamaan kokonaiskuvan sovelluksen tarpeista.

Määrittely aloitetaan toiminnallisuudesta. Simulointisovelluksen avulla voidaan lukea sekä kirjoittaa ohjelmoitavalle logiikalle kenttärajapinnan arvoja. Lisäksi sovellus sisältää ennalta määriteltyjä vasteita, joiden avulla voidaan helpottaa logiikan ohjelmistotestausta. Simulointisovellukseen tuodaan tarvittava määrä kenttärajapinnan pisteitä, jotka on nimetty ja skaalattu projektin lähtötietojen mukaisesti.

Seuraavaksi määritellään käytettävyys. Ohjelmoitavan logiikan testauksen tehostamiseksi, simulointisovelluksen tulee olla helposti käyttöönotettava, tehokas käyttää sekä hyvin saavutettavissa. Tehokkaan käytön edellytyksenä on yksinkertaisuus, sekä käyttöä tukeva dokumentointi. Lisäksi sovelluksella tulee olla liitettävyys nykyaikaisiin Ethernet-

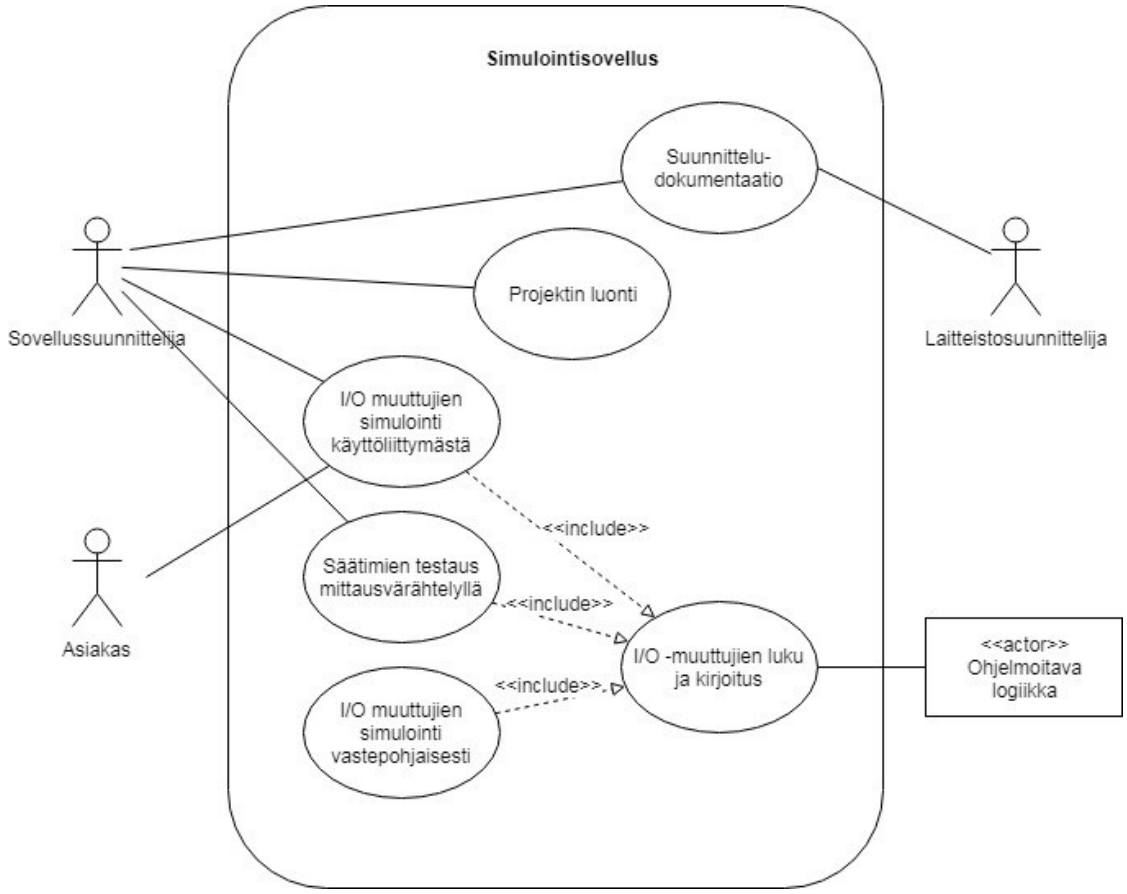
rajapintaa hyödyntäviin ohjelmoitaviin logiikkoihin. Simulointisovellus konfiguroidaan automaatiosovelluksen mukaan ja sovellus saa parametrinsa koontitaulukosta.

Sovelluksen luotettavuus varmistaa, että ohjelmisto ja sen vasteet ovat luotettavia, jotta testaamisen hyödyt voidaan valjastaa. Simulointisovelluksen tarkoitus on vähentää testaajan työtaakkaa ja inhimillisiä virheitä. Toisaalta luotettavuuden lisäksi sovelluksen tulee myös olla tehokas. Simulointisovelluksen tehokkuus pohjautuu yksinkertaiseen kenttärajapinnan simuloimiseen. Käyttöliittymän on oltava yksinkertainen, helposti luettava sekä hallittava. Arvojen muutosten on oltava vasteeltaan nopeita, jotta tehdyt toimenpiteet näkyvät selkeästi ja ovat tulkittavissa yksiselitteisesti.

Sovellukselle määrätään myös yksityiskohtaisempia tehtäviä. Sovelluksen rajoitukseksi asetetaan, että testaussovellus sisältää liitynnät ainoastaan I/O rajapintaan. Rajapintavaatimukseltaan testaussovelluksen on käytettävä tiedonsiirtostandardia, joka kykenee toimimaan kaikilla yleisimmillä uusilla ohjelmoitavilla logiikoilla. Viimeiseksi asetetaan ympäristövaatimus, jonka mukaan testaussovelluksen on toimittava suunnittelijan omalla tietokoneella.

Alla on esitetty sovellusta koskevat käyttötapaukset, jotka ovat havainnollistettu myös kuvan 19 käyttötapauskaaviossa.

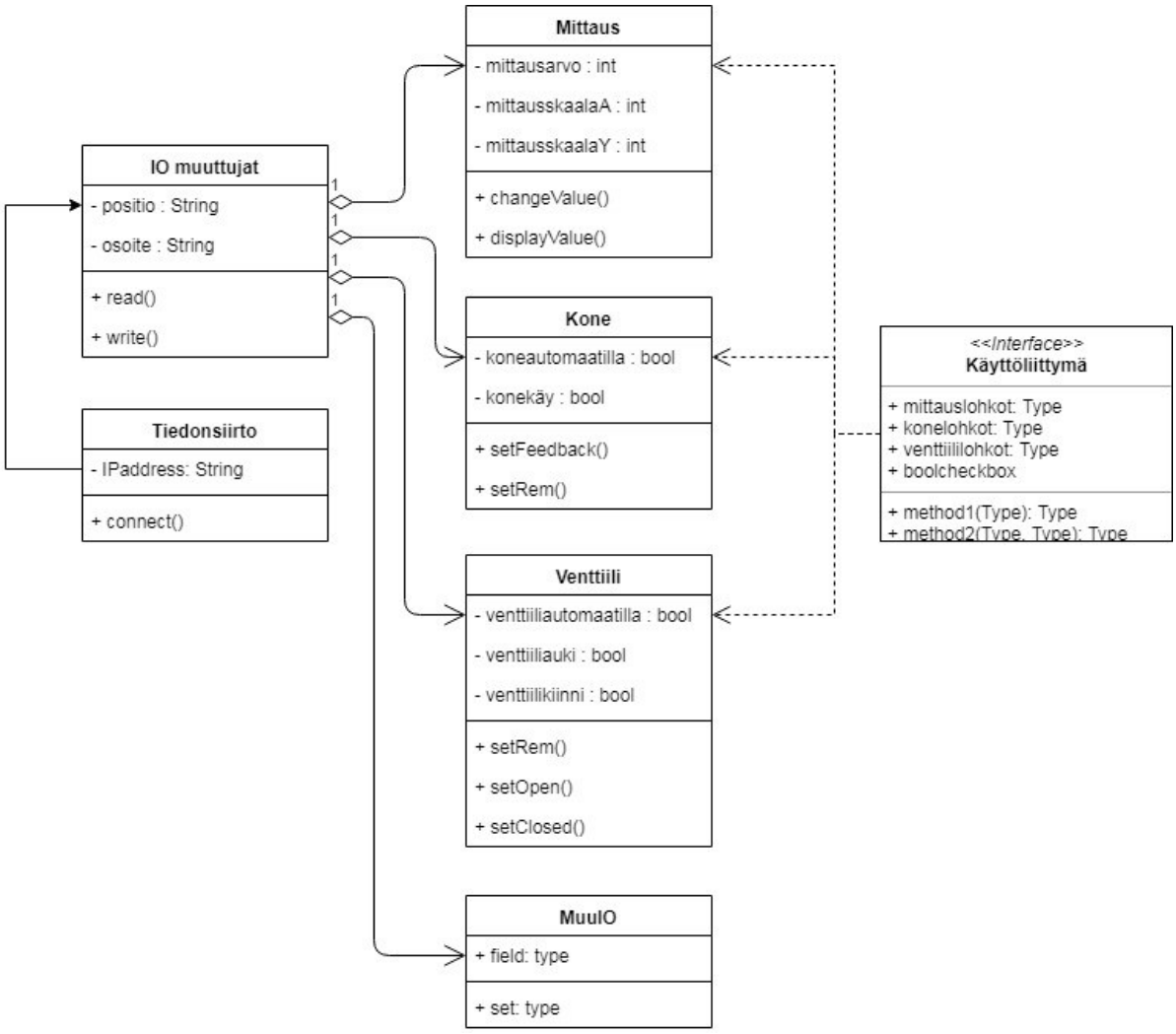
- *Käyttäjä*: Suunnittelija
- *Tavoite:* Simuloida prosessin käyttäytymistä erillisen sovelluksen avulla, jotta voidaan testata automaatiosovellusta eri kehitystasoilla.
- *Kuvaus:*
    1) Käyttäjä luo simulointiprojektin automaatiojärjestelmän suunnitteludokumentaation pohjalta, kuten esimerkiksi ohjelmoitavan logiikan generointitaulukko, sekä määrittää tiedonsiirtoa koskevat parametrit ohjelmistolle, jonka jälkeen simulointisovelluksella on edellytykset kommunikaatioon logiikan kanssa.
    2) Käyttäjä voi ohjelmiston avulla simuloida logiikan kenttärajapintaa käyttöliittymän graafisten kuvakkeiden avulla.
    3) Simulointisovellus antaa toimintojen edellyttämät tarvittavat vasteet lohkoille, jotta prosessi ei pysähdy esimerkiksi käyntitiedon puutteeseen.
    4) Käyttäjä voi testata säätimien toimisuunnat mittausvärähtelyjen avulla.
- *Poikkeukset:* Simulointi ei onnistu, mikäli ohjelmoitava logiikka ei ole käynnissä tai tiedonsiirto-ominaisuudet on poistettu käytöstä.
- *Muut vaatimukset:* Ohjelmiston toiminta ei saa aiheuttaa ohjelmoitavan logiikan normaalille ohjelmasuoritukselle muutoksia. Ohjelmiston kirjoitus ainoastaan I/O-rajapinnan muuttujille. Ohjelman pääkäyttäjä on suunnittelija, mutta ohjelmiston käyttöliittymän on oltava myös asiakkaalle ymmärrettävä. Tiedonsiirto on toteutettava laitteistoriippumattomalla menetelmällä.



Kuva 19. *Käyttötapauskaavio testaussovelluksen yleisistä käyttötapauksista*

Simulointisovelluksen on tarkoitus tukea automaatiosovelluksen kehitystä elinkaaren ajan. Automaatioprojekti etenee useissa projekteissa kutakuinkin samankaltaisesti, jolloin simulointisovelluksen liittämiselle voidaan antaa aikataululliset minimivaatimukset. Automaatioprojekteissa projektin dokumentaatio kehittyy suunnittelun edetessä. Simulointisovellus voidaan ottaa käyttöön, kun projektista on luotu I/O taulukko tai vastaava, josta löytyvät vähintään positiot ja skaalaukset, ja kun ohjelmoitava logiikka on konfiguroitu OPC UA palvelimeksi.

Ohjelman yleisten käyttötapausten sekä vaatimusten lisäksi simulointiohjelman rakennetta tarkennetaan luokkakaavion avulla. Luokkakaaviossa esitetään ohjelman tärkeimmät komponentit ja niiden ominaisuudet. Ohjelmoitavan logiikan testaamisessa simuloidaan kenttärajapintaa, joka koostuu mittauksista, koneista, venttiileistä ja yksittäisistä digitaalituloista. Luokat toteutetaan ominaisuuksien mukaan (Kuva 20).



Kuva 20. *Luokkakaavio ohjelmiston yleisistä ominaisuuksista*

Automaatiojärjestelmien simulointia varten on markkinoilla useita erilaisia valmiita ohjelmistoja, joita kutsutaan myös nimellä ”virtual engineering tool”. Yhtenä esimerkkinä annettakoon WinMOD, joka on ympäristö prosessin digitaaliseen suunnitteluun ja mallintamiseen. Ohjelmisto on valmis työkalu, jota voidaan käyttää ohjelmoitavan logiikan sovelluksen ohjelmistotestaukseen ja tehdastestauksen apuvälineenä. [36] Kaupalliset ohjelmistot sisältävät paljon ominaisuuksia, joista osa, kuten ohjelmoitavan logiikan sovelluksen generoiminen ja säätöjen optimointi ovat jo ennestään toteutettu Insta Automation Oy:n omilla työkaluilla. Toisaalta taas kaupalliset ohjelmistot sisältävät lukuisia määrittelyyn kuulumattomia ominaisuuksia, jotka hankaloittavat tässä tapauksessa tehokasta käytettävyyttä. Ohjelmisto toteutetaan näin ollen itsenäisenä, simulointia varten räätälöitynä, joka täyttää annetut vaatimukset mahdollisimman helppokäyttöisenä ja tehokkaana työkaluna.

3.2 Testaussovelluksen suunnittelu

Vaatimusten ja määrittelyiden myötä voidaan aloittaa simulointisovelluksen tarkempi suunnittelu. Tämän vaiheen tarkoituksena on kuvata, miten ohjelma toteutetaan, eli toisin sanoen tekninen määrittely, arkkitehtuurisuunnittelu. Kun ohjelmiston tekninen toteutus on suunniteltu ja toimintokokonaisuudet jaoteltu osiin, voidaan osien tarkka määrittely suunnitella moduulisuunnittelussa. Moduulisuunnittelussa komponenttien tarkka toimintatapa kuvataan ohjelmistotason menetelmin. Simulointisovellus toteutetaan lopulta moduulien kuvausten mukaisesti.

Arkkitehtuurisuunnittelu

Arkkitehtuurisuunnittelussa ohjelma ositetaan abstraktioiden, toiminto, tieto ja tyyppi, mukaan seuraavasti: tiedonsiirto OPC UA palvelimelle, projektikohtaisten muuttujien haku ja sovelluksen toiminnollisuus. Alla on esitetty ohjelman jaotellut osat.

Kappaleen *2.3* tiedonsiirto -selvityksen mukaan käytetyimpien ja samalla tärkeimpien ohjelmoitavien logiikoiden ainoaksi suoraan yhteensopivaksi tiedonsiirtostandardiksi valikoitui OPC UA. Näin ollen on kannattavaa rakentaa sovellus tukemaan OPC UA tiedonsiirtoa.

Useat ohjelmoitavat logiikat, kuten esimerkiksi Siemens S7-1500, tarjoavat sisäänrakennetun OPC UA palvelimen tiedonsiirtoa varten. Integroitujen palvelinten lisäksi valmistajilla on tarjolla lisäosia, jolla OPC UA palvelin voidaan toteuttaa ohjelmoitavalle logiikalle. Simulointisovelluksen kommunikointiin toteutetaan tietokoneelle OPC UA asiakas, joka palveluillaan kykenee yhteyden luomiseen, päättämiseen ja ylläpitämiseen OPC UA palvelimelle, tapahtumien hallintaan sekä informaatiotiedon lukemiseen ja kirjoittamiseen.

Insta Automation Oy:n käyttämiltä logiikkavalmistajilta löytyy tuotteistaan OPC UA palvelimen tarjoamia malleja. Valmistajien sivustoilta sekä datalehdiltä poimittuna voidaan todeta OPC UA –palvelimien kykenevän muodostamaan OPC UA protokollan Osan 3 mukaisen osoiteavaruuden sekä kykenemään Osan 5 informaatiomallin mukaiseen tiedonsiirtoon. [45-47]

PC-pohjainen simulointiohjelmisto toimii OPC UA asiakkaana, joten sen on kyettävä muodostamaan ja välittämän UA protokollan mukaiset kutsut palvelimelle. Kutsujen avulla muodostetaan asiakkaan ja palvelimen välille vaaditut kommunikaatiokerrokset, joita ovat suojattu kanava, istunto, tilaus sekä kohteen valvonta. Ohjelmointirajapinnan (API) pinon (stack) toteuttamiseen on mahdollista käyttää kappaleen 2.3 mukaisesti seuraavia menetelmiä: .NET Client, C++ Client, ANSI C Client ja Java Client.

Simulointisovellus on mahdollista alustaa samoista lähtötiedoista ja generointitaulukoista kuin automaatiosovelluksen perusversio. Toisin sanoen simulointisovelluksen käyttöönottamiseen riittävät lähtötiedot, joiden avulla toteutetaan ohjelmoitavan logiikan sovellus. Lähtötietoina ohjelmoitavan logiikan sovelluksen toteuttamiseen ovat useimmiten I/O-luettelo peruspiirien eli moduulien toteuttamiseen, kuten esimerkiksi mittaus-, kone-, venttiili- ja säätöpiirit sekä toiminnallisuuden toteuttamiseen prosessin toimintakuvaukset ja PI-kaavio. Simulointisovellus ilman prosessin dynamiikkaa tarvitsee lähtötietoina ainoastaan I/O-luettelon, joka sisältää kytkentäpisteiden ja symbolien lisäksi skaalaukset sekä yksiköt.

Simulointisovellus sisältää tässä vaiheessa mittausten, koneiden ja venttiilien tietojen käsittelyyn tarvitut tiedot. Ohjelmiston saavutettavuuden ja käytettävyyden vuoksi lähtökohtana on simulointisovelluksen käyttöönottaminen suoraan Excel-taulukoista tai Excel-taulukoista muodostetusta tekstitiedostosta (Comma-Separated Values, csv). Muuttujat on mahdollista viedä myös logiikalta erilliseksi tiedostoksi. Tiedostoa voidaan käsitellä ja karsia sieltä vain simulointisovellusta koskevat muuttujat/symbolit. Simulointiin tarvitut tiedot tuodaan sovellukseen ja yhdistetään oikeisiin toimintoihin lähtötietojen mukaisesti.

Insta Automation Oy:llä on käytössä ohjelmoitavien logiikoiden sovellusten generointi peruspiiritasolle saakka käyttäen sitä varten räätälöityä Excel-taulukkosovellusta, joten työn myöhemmissä osissa simulointisovelluksen muuttujien luomiseen käytetään generointiin tarkoitettuja projektikohtaisia taulukkoja. Ohjelmointitaulukko itsessään sisältää välilehdillään kaikki simulointiin tarvitut tiedot niin mittauksista, koneista kuin muista peruspiireistä, joten simulointisovelluksen muuttujat voidaan hakea suoraan taulukon soluista.

Simulointiohjelmiston tulee päivittää arvoja tapahtumapohjaisesti, toisin sanoen tilanteissa, kun muuttujan arvo muuttuu logiikalla tai simulointisovelluksella. OPC UA tarjoaa mekanismin, jolla saadaan ilmoitus palvelimen solmun arvon muutoksesta. Näin logiikan arvojen luku onnistuu tapahtumapohjaisesti turhaan kuormittamatta.

Simulointisovelluksessa hyödynnetään graafista käyttöliittymää. Kuvakkeet jaotellaan peruspiirien mukaisesti, jolloin toiminta on selkeää ja loogista. Kuvakkeet sisältävät generointipohjan mukaiset toiminnot, skaalaukset ja positiot. Graafisen käyttöliittymän toteuttamiseen on useita mahdollisuuksia, mutta edellytyksenä on helppo käyttöönotto sekä käyttö.

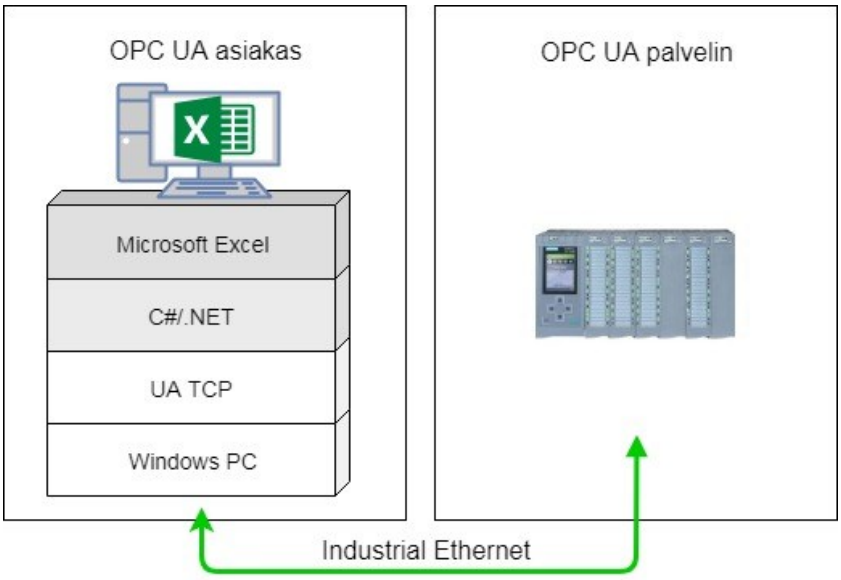
Automaatiosovelluksen testaamisessa suurimpia haasteita ovat reaaliprosessin samanaikaiset tietojen muutokset, eli on otettava huomioon lohkojen toiminnallisuudet ja vasteet samanaikaisesti. Muuttujilla on usein keskinäisiä yhteyksiä, joiden puuttuva reaktio saa aikaan toiminnon keskeytymisen. Simulointisovelluksen yhtenä tärkeänä toimintona on perustoimintojen vasteiden simulointi, joita ovat koneiden ja venttiileiden käyntitiedot.

Moduulisuunnittelu

Kun simulointisovellus on jaettu arkkitehtuurisuunnittelussa osiin, voidaan kunkin osan rakenne tarkentaa ja kuvata niin tarkasti, että toteutus on mahdollista.

Tiedonsiirto toteutetaan kappaleen 2.3 OPC UA tiedonsiirto periaatteiden mukaan. Toteutetaan siis kommunikaatiorakenne, jonka palvelujen ja kerrosten avulla on mahdollista luoda palvelimen ja asiakkaan välille kanava, istunto tilaus sekä kohteen valvonta.

Simulointisovelluksen alustaksi valitaan käytettävyyden, liitettävyyden ja ylläpidon vuoksi Microsoft Excel. Tiedonsiirron rakentamiseen täytyy tietokoneelle luoda OPC UA asiakasohjelmisto, jota käytetään Excelin avulla. Kuvassa 21 on havainnollistettu toteutus simulointijärjestelmän OPC UA kommunikaatiorakenteesta. [22]



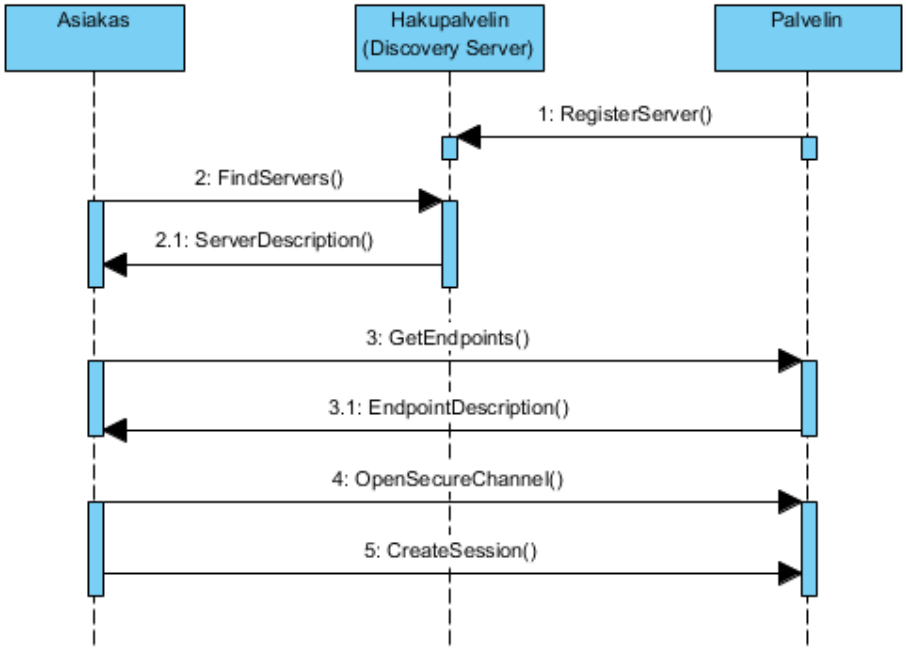
Kuva 21. *Excel-pohjainen OPC UA -asiakas liitettynä PLC –palvelimeen*

OPC UA asiakkaan ja palvelimen välinen tiedonsiirto perustuu palvelimen rajapinnan abstrakteihin palveluihin. Palvelupyyntöjen, eli kutsujen lähettäminen asiakkaalta palvelimelle suoritetaan OPC UA pinossa. Työn pinoksi valitaan OPC Foundationin tarjoama Microsoft C#/.NET, sillä simulointisovellusta suoritetaan myös Windows –pohjaisella tietokoneella ja yhteen liitettävyys on luontevaa. OPC .NET pino tarjoaa sovellukselle Taulukko 4 mukaiset pavelut.

Taulukko 4. *OPC Foundationin tarjoamat C#/.NET pohjaiset palvelut [36]*

Palvelun toiminto	**Palvelupaketti**	**Kutsunimi**
Palvelimien haku	Discovery Service Set	FindServers, GetEndpoints
Kanavan luominen	Secure Channel Service Set	OpenSecurechannel
Istuntojen aloitus ja lopetus	Sessions Service Set	CreateSession, CloseSession)
Navigointi osoiteavaruudessa	View Service Set	Browse, RegisterNodes, UnregisterNodes
Attribuuttien ja tunnisteiden luku ja kirjoitus	Attribute Service Set	Read, Write
Tilauksien valvonta	Subscription Service Set	CreateSubscription, DeleteSubscription
	MonitoredItem Service Set	CreateMonitoredItem, DeleteMonitoredItem
Metodien kutsut	Method Service Set	GetMethodArguments, Call-Method

OPC UA asiakkaan ja palvelimen välisen suojatun kanavan muodostaminen ja istunnon avaaminen esitetään kuvan 22 sekvenssikaaviossa. Kaaviossa esiintyvää ”Discovery Service Set” –palvelupakettia käytetään verkon OPC UA -palvelimien skannaamiseen hakupalvelimelta sekä päätepisteiden hakuun. Kanavan muodostamisessa käytetään ”Secure Channel Service Set” –palvelupakettia. Kanava tarvitsee lisäksi istunnon, jotta osoiteavaruuden käsittely on mahdollista.

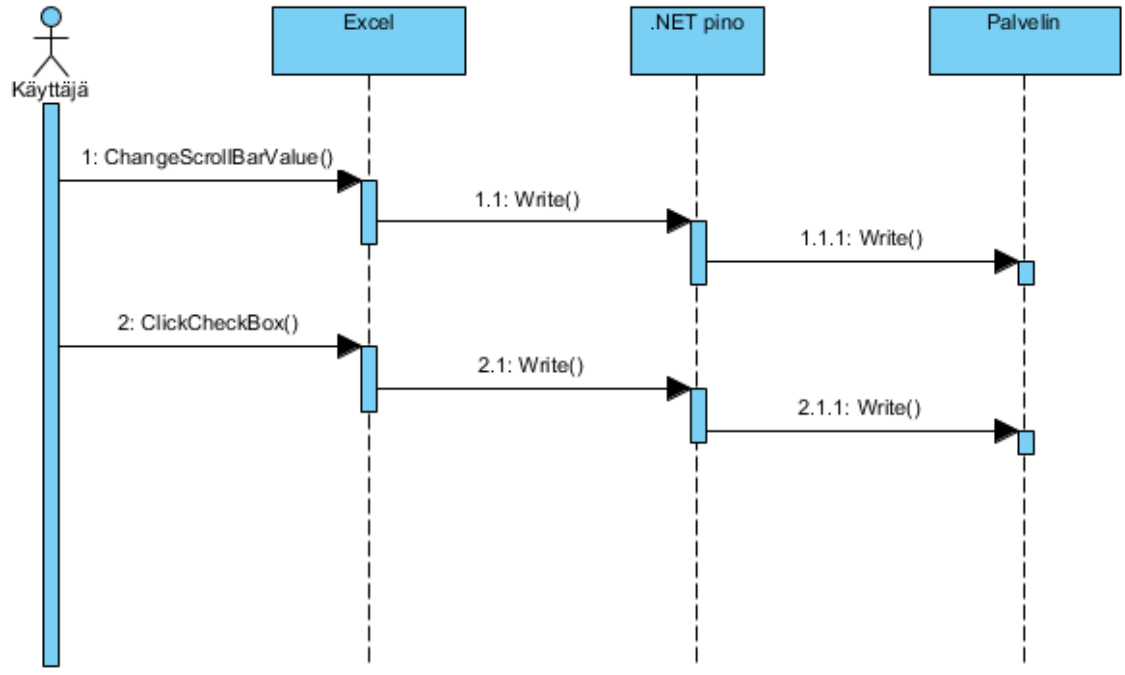


Kuva 22. *Palvelinten haku ja yhteyden muodostus*

Simulointisovellus integroidaan osaksi Insta Automation Oy:llä käytettävää automaatio-ohjelmistojen Excel –generointityökalua, joka sisältää projekteissa ylläpidettynä kaikki simulointiin vaaditut I/O-rajapinnan tiedot. Simulointisovelluksena tarkoitetaan tästä eteenpäin Excel taulukkolaskentaohjelman välilehteä, joka sisältää Microsoftin omalla ohjelmointikielellä (Visual Basic for Application, VBA) toteutettuja luokkia sekä funktioita. Simulointisovelluksen muuttujat haetaan logiikkaohjelman generointivälilehdiltä, jolloin muuttujien tiedot pysyvät keskitetysti ajan tasalla. Skaalauksien, positioiden ja yksiköiden osoiteukset ovat solujen välisiä viittauksia käyttöliittymässä. Keskitetty kanta ja integroitu simulointisovellus tarjoavat tehokkaan saavutettavuuden ja käytettävyyden.

Ohjelmiston avulla kenttärajapinnan simuloituja tietoja kirjoitetaan OPC UA yhteyden kautta logiikalle. UA tiedonsiirrossa palvelupaketti ”Attribute Service Set” mahdollistaa tietojen siirron haluttuun solmuun osoiteavaruudessa. Simulointisovelluksen käytettävyyden vuoksi lähetykset automatisoidaan käyttöliittymän arvon tai tilan muutoksesta, jolloin erillistä lähetä-painiketta ei tarvita.

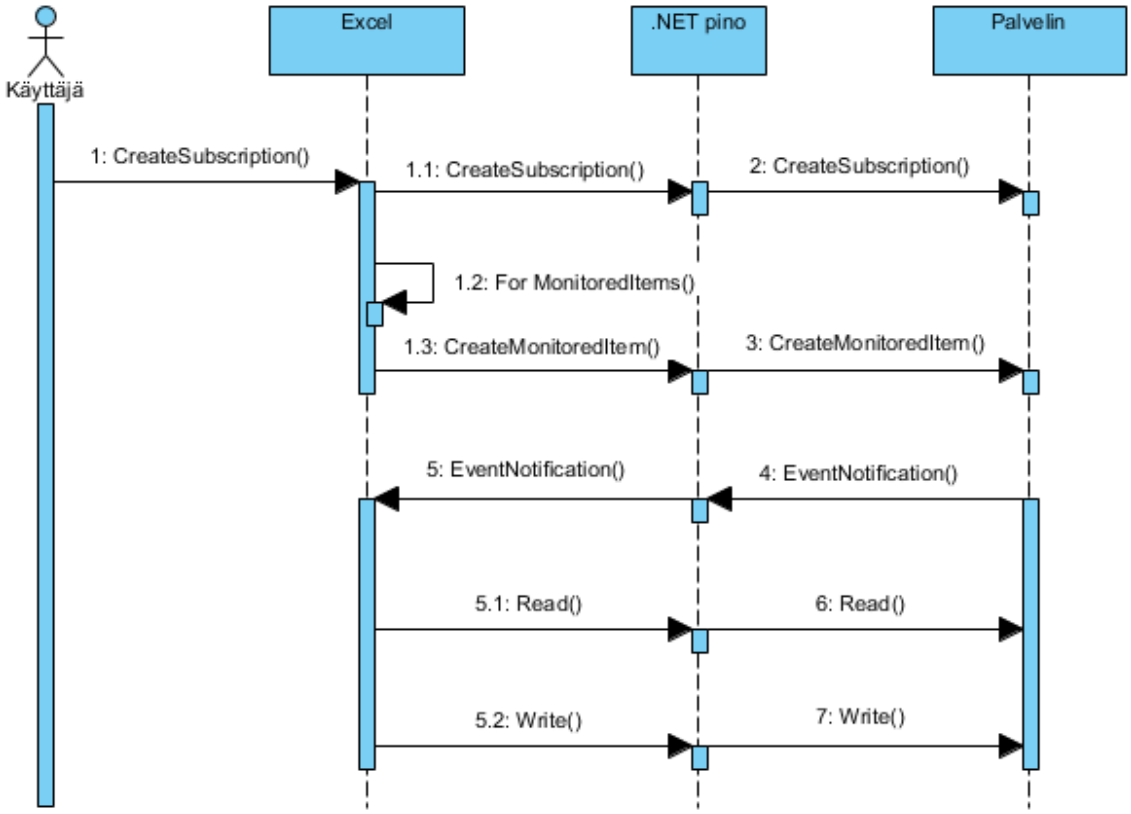
Simulointisovelluksen käyttöliittymä sisältää graafisia kuvakkeita, joihin on ohjelmoitu toiminnollisuus. Simulointi voidaan toteuttaa täysin käyttöliittymän grafiikasta. Painikkeiden painamisesta ja liukupalkkien muuttamisesta luodaan tapahtuma, joka kirjoittaa sen hetkisen tilan OPC UA palvelimelle. (Kuva 23)



Kuva 23. *Arvojen lähetys OPC UA asiakkaalta palvelimelle.*

Simulointiohjelmiston on kyettävä reagoimaan ohjelmoitavan logiikan lähdöissä tapahtuviin muutoksiin, jotta simuloitavat vasteet voidaan käsitellä ja kirjoittaa logiikalle. Logiikan lähdöissä tapahtuvia muutoksia voidaan lukea tilausten ja kohteenvalvonnan

avulla. Tapahtumapohjainen mekanismi tarjoaa mahdollisuuden käyttää simulointisovellusta binäärisistä takaisinkytkennöistä aina prosessin vasteiden simulointiin. Vasteiden hallinta tapahtuu simulointisovelluksessa. Simuloinnissa takaisinkytkentäarvojen lisäksi on erittäin tärkeää tiedon reaaliaikaisuus etenkin systeemin dynamiikan simuloinnissa. Riippuen OPC UA palvelimen fyysisistä ominaisuuksista ja seurattavien kohteiden määrästä, kykenee OPC UA alle 250 millisekunnin vasteeseen (latency) [48]. Ongelmana kuitenkin tiedon reaaliaikaisuuden hallintaan on verrattain suuri vasteen vaihteluväli (jitter), jota voidaan kuitenkin pienentää jopa alle 10 millisekuntiin käyttämällä OPC UA Time Sensitive Networks protokollaa [49]. Kuvan 24 sekvenssikaaviossa on esitetty tilauksen sekä kohteen valvonnan luominen. Kuvassa on esitetty myös simulointisovelluksen vasteiden käsittely seurattavan kohteen tapahtumailmoituksesta.



Kuva 24. *Sekvenssikaavio tapahtumapohjaisten vasteiden luomiseen.*

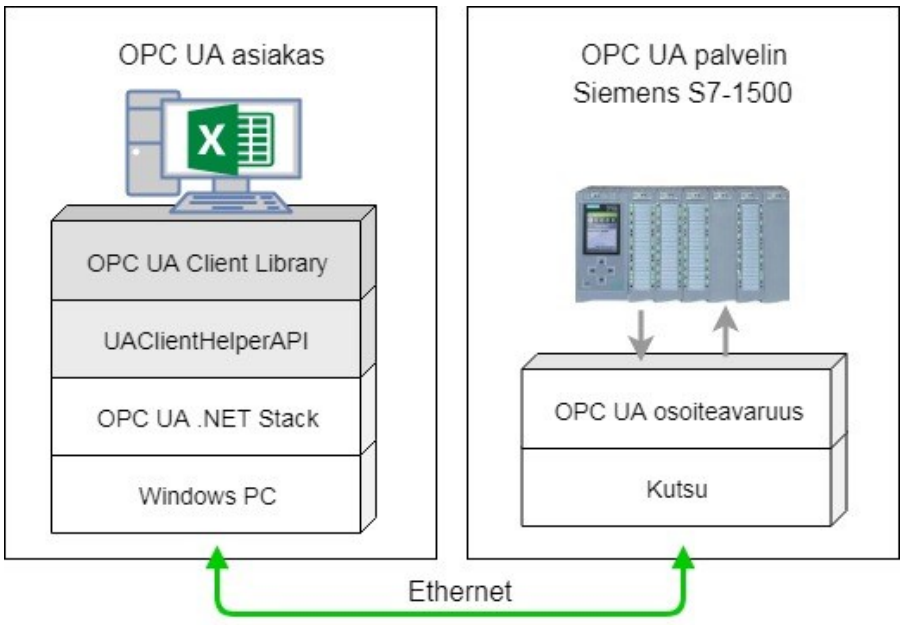
Suunnitelluilla ominaisuuksilla rakennettu simulointisovellus kykenee ohjauslogiikan kenttärajapinnan kattavaan simulointiin. Simulointisovellukselle voidaan ohjelmoida erilaisia tapahtumapohjaisia vasteita, jolloin voidaan suorittaa prosessiosien sekä säätöpiirien toiminnollisuutta. Ohjelmoitavan logiikan säätöpiirien toimisuuntia ja reagointia voidaan testata syöttäen simulointiohjelmalla hidasta mittausvärähtelyä, eli oskillointia mittauksen kenttärajapintaan.

4. SIMULOINTIOHJELMISTO EXCEL OPC UA

Simulointisovelluksen vaatimuksissa ja määrittelyissä on annettu reunaehdot toteutettavalle ohjelmistolle. Suunnittelussa ohjelmistoprojekti on saatettu pisteeseen, jonka jälkeen moduulit ohjelmoidaan ja integroidaan lopulta yhdeksi kokonaisuudeksi. Toteutettavassa versiossa ohjelmoitavan logiikan rajapintaa simuloidaan käytettävyyden ja integroituvuuden vuoksi Microsoft Excel 2010 –taulukkolaskentaohjelmalla. Seuraavaksi kerrotaan toteutukseen käytetyt laitteistot sekä ohjelmistot.

Excel toimii simuloinnin käyttöliittymänä sekä OPC UA -asiakasohjelmiston ylimpänä kerroksena. Taulukkolaskentaohjelmistoon on ladattu ”Siemens OPC UA Client Library” -lisäosa, jonka avulla se pystyy kommunikoimaan C# -pohjaisen ohjelmoitavan rajapinnan kanssa [50]. Ohjelmoitava rajapinta ”Siemens UAClientHelperAPI” tarjoaa rajapinnan sekä sisältää OPC UA palveluiden luokat, joita vaaditaan asiakas-palvelin yhteyteen [22]. OPC UA pinona toimii OPC Foundationin luoma avoin OPC UA .NET Stack [51]. OPC UA .NET pino toimii Windowsin alustalla. (Kuva 25)

Simuloinnin testauksessa OPC UA palvelimena toimii Siemens S7-1500 -sarjan ohjelmoitava logiikka, tarkemmalta malliltaan S7-1513-1PN. Logiikan ohjelmointialustana käytetään Siemens Totally Integrated Automation –portaalin (TIA) versiota V15.1. Ohjelmistoja suoritetaan Windows 10 pohjaisella virtuaalikoneella.



Kuva 25. *Simulointiohjelmiston prototyypin rakenne [50]*

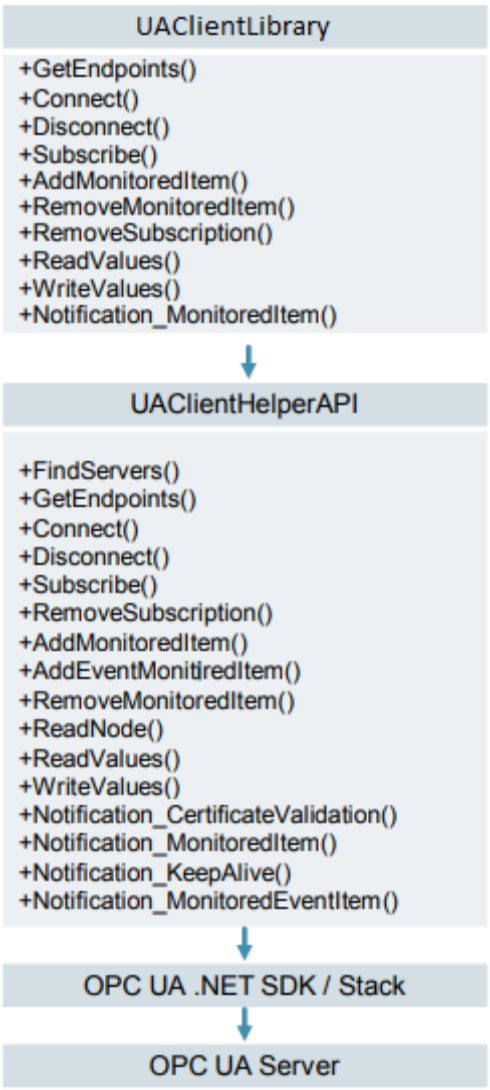
4.1 Siemens S7-1500 OPC UA -palvelimen konfigurointi

Siemensin S7-1500 –sarjan ohjauslogiikoita voidaan ohjelmoida ja konfiguroida Siemens TIA portal -ohjelmistolla. Ohjelmisto on kokonaisvaltainen automaatiojärjestelmän toteutus- ja ylläpito ohjelmisto niin logiikoille, verkoille kuin valvomoille. Siemens S7-1500 logiikan käyttöönottaminen OPC UA palvelimena vaatii laitteiston konfiguroinnin TIA-portaalilla. Seuraavassa on esitetty logiikan konfigurointi OPC UA –palvelimen käyttöönottamiseksi.

S7-1500 sarjan logiikat sisältävät OPC UA palvelimen sisäänrakennettuna, mutta sen käyttö on oletuksena estetty. OPC UA:n käyttö vaatii vähintään ”SIMATIC OPC UA S7-1500 small” –tasoisen lisenssin. Kun lisenssi on valittu, voidaan logiikan OPC UA –palvelin ominaisuudet aktivoida käyttöön. Palvelimelta määritellään yhteyttä koskevat yleiset asetukset, kuten esimerkiksi käytettävä porttinumero, istuntojen ”time out” sekä istuntojen ja solmujen suurin määrä. Lisäksi määritetään tilauksia koskevat asetukset kuten pienin päivitys- (sampling interval) ja julkaisuväli (publishing interval) sekä suurin seurattavien kohteiden määrä. Suojatun kanavan muodostamiseen täytyy valita käytettävät salausluokat, jotka ilmoitetaan asiakkaalle yhteyden päätepisteiden haussa. Jotta palvelin näkyy asiakkaalle ja kanava saadaan muodostettua, vaatii palvelin projektin suojaamisen ja ylläpitäjän asettamisen. Ohjelmoitava logiikka toimii annetuilla konfiguraatioilla palvelimena ja luo osoiteavaruuden solmuluokille. TIA -portaalista valitaan ne kenttärajapinnan muuttujat, joita asiakkaalla on oikeus lukea ja kirjoittaa. Palvelimen osoiteavaruudesta löytyvät solmut ja yhteydet voidaan skannata OPC UA asiakkaalta ”BrowseNode()”- ja ”BrowseRoot()” –palveluilla esimerkiksi kenttärajapinnan solmutunnisteiden määrittämiseen.

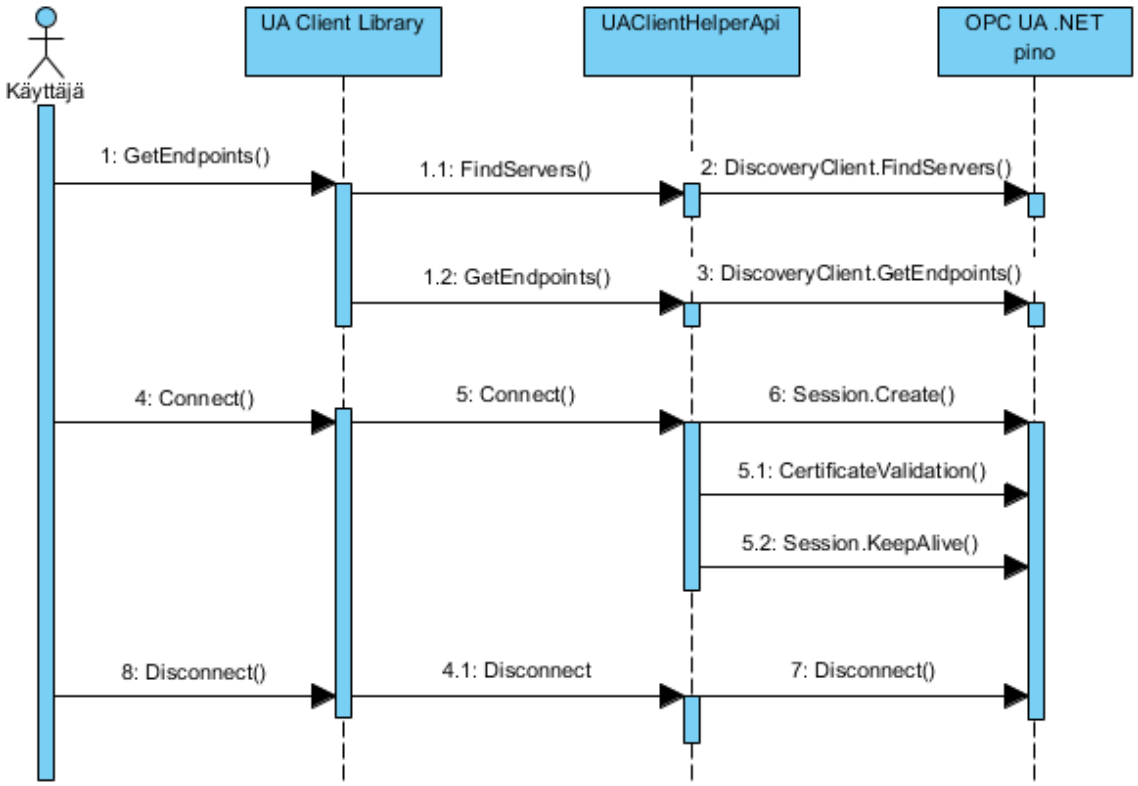
4.2 OPC UA asiakkaan prototyypin toteutus

Simulointisovelluksen prototyyppi rakennetaan Siemensin tarjoaman OPC UA Excel -asiakkaan päälle, sillä valmis rakenne sisältää simulointisovellukselta vaaditut OPC UA palvelut. Simulointisovellus tarvitsee toimiakseen vähintään seuraavat palvelut: FindServers(), GetEndpoints(), OpenSecureChannel(), CloseSecurityChannel(), CreateSession(), CloseSession(), Subscribe(), AddMonitoredItem(), RemoveSubscription(), Read() ja Write(). Kuvassa 26 on esitetty simulaatiosovelluksessa käytettyjen palvelupyyntöjen kulku Excel asiakkaalta “UAClientLibrary” S7-1500 OPC UA palvelimelle.



Kuva 26. *OPC UA asiakkaan palvelupyynnöt toteutetussa sovelluksessa*

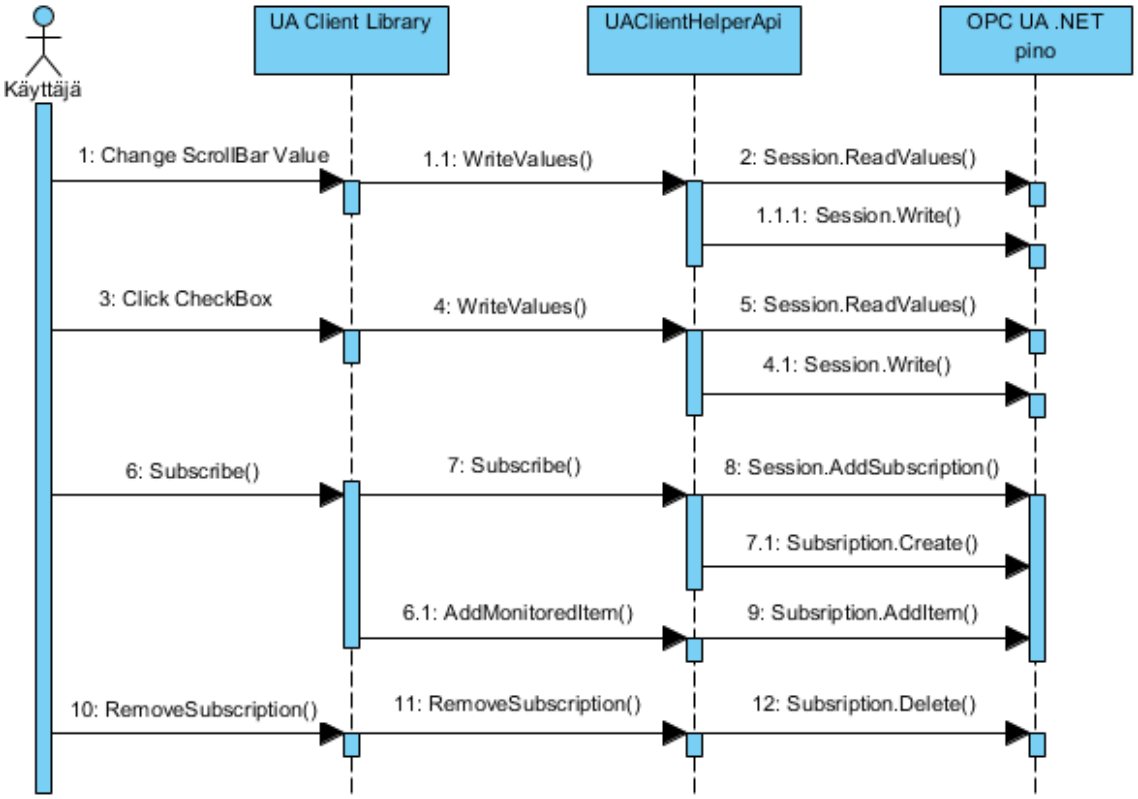
OPC UA asiakkaan toteutus toimii ”OPC UA .NET stack” –pinon päälle rakennettuna. Pino sisältää kaikki varsinaiseen OPC UA tiedonsiirtoon liittyvät luokat sekä oliot ja suorittaa näiden avulla OPC UA standardin mukaiset kutsut palvelimelle. Ohjelmoitava käyttäjärajapinta ”UAClientHelperAPI” on C# -luokka, jonka tarkoituksena on koota kaikki tärkeimmät OPC UA kommunikaatioon liittyvät kutsut sekä antaa .NET –pinon olioille vaadittavat tiedot. Ohjelmoitavan käyttäjärajapinnan ja varsinaisen Excel ohjelmiston välissä toimiva ”UAClientLibrary” –kirjasto on rekisteröity Microsoftin oman .NET Framework ohjelmistokomponenttikirjaston käytettäväksi. UA kirjasto sisältää oliot, joiden avulla Excelin ohjelma (Visual Basic for Application, VBA) välittää tapahtumapohjaisesti kutsut API rajapinnalle. Kuvassa 27 on esitetty kutsujen kulku turvallisen kanavan ja istunnon muodostamisessa OPC UA .NET pinolle. UA Client –kirjasto yhdistää palvelimien haun ja päätepisteiden haun kutsut käyttäjän samalle operaatiolle. API välittää kirjaston kutsut pinolle, mutta myös käsittelee istuntoon liittyvät sertifikaattien validoinnit sekä yhteyden ylläpidon.



Kuva 27. *Sekvenssikaavio turvallisen kanavan muodostamisesta*

Simulointiarvojen sekä tilatietojen kirjoitus on toteutettu Excelin ActiveX komponentin, kuten liukupalkin ja valintaruudun, arvon muutoksesta Microsoft VBA ohjelmoinnilla. Suora lähetys tapahtumasta helpottaa sovelluksen käyttöä, sillä erillisiä kirjoituspainikkeita ei tarvita. Liukupalkkien ja valintaruutujen lähetykset on toteutettu komponentteihin VBA ohjelmoinnilla. ”UAClientHelperAPI” -luokan sisäinen toiminta lukee tapahtumassa palvelimelta tiedon datatyypin ja lähettää valitun arvon palvelimelle samana data tyyppinä. (Kuva 28)

Koneiden ja venttiilien takaisinkytkentätietojen kirjoittaminen logiikalle vaatii logiikan ohjausten tapahtumien välittämisen UA kirjastolle. Tilatietojen saamiseen käytetään istunnolle luotavaa tilausta, joka sisältää kaikkien koneiden ja venttiilien ohjaustiedot. Ohjaustiedon aktivoituessa logiikalla OPC UA asiakas saa tapahtumailmoituksen kyseisestä solmutunnisteesta ja VBA:lle luotu ohjelma vastaa tilan muutokseen lähettämällä koneelle käyntitiedon ja venttiileille asentotiedon. UA kirjaston tehtävänä on luoda tilaus ja kohteen valvonta kullekin solmutunnisteelle. (Kuva 28)



Kuva 28. *Sekvenssikaavio luvusta ja tilauksen luomisesta*

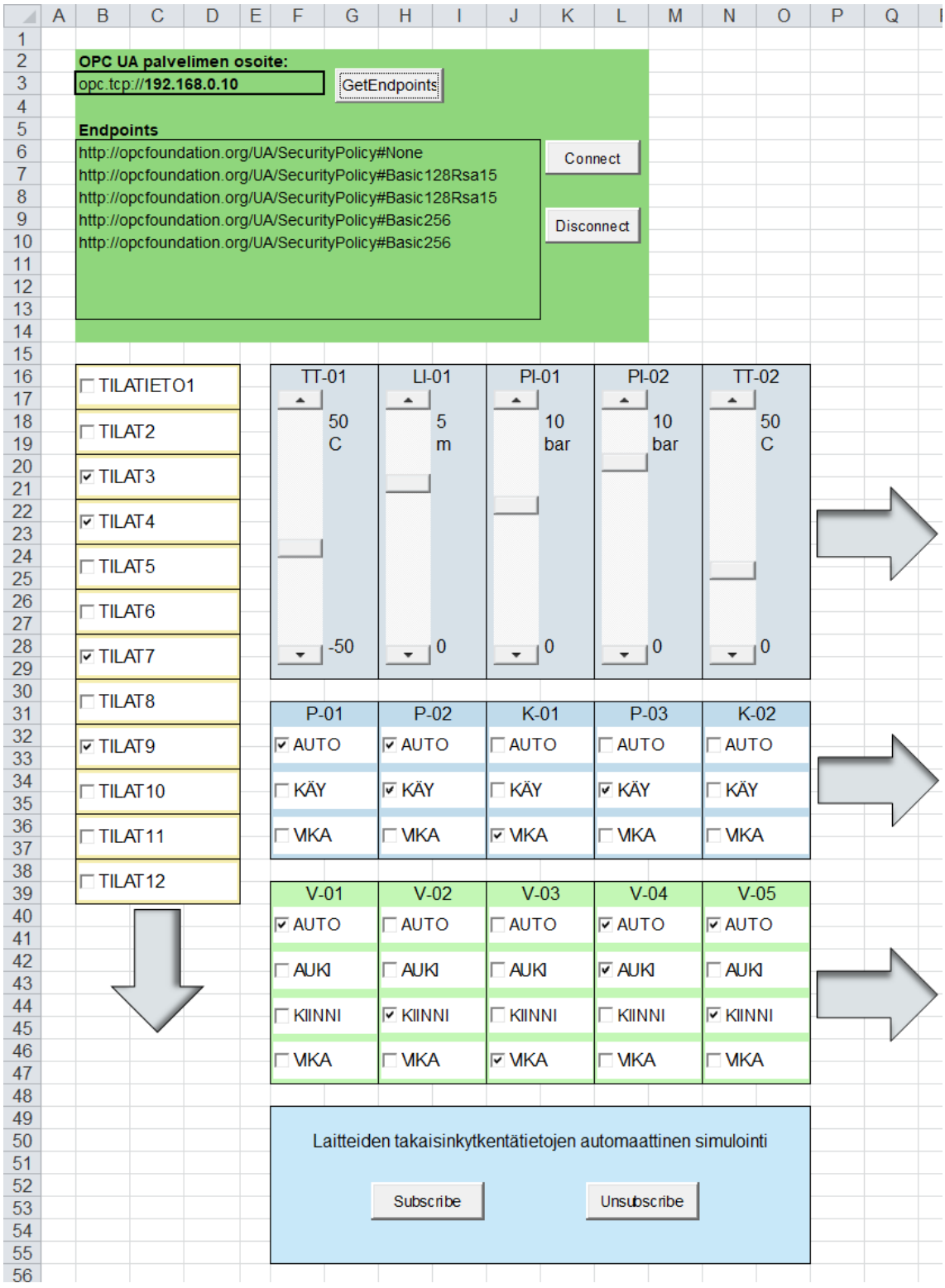
4.3 Ohjelmiston testaus simuloimalla I/O-rajapintaa

Simulointisovelluksen yksittäiset komponentit toteutettiin Excel käyttöliittymään ActiveX-komponenteilla. Komponentit käyttävät Microsoft VBA ohjelmointikieltä toiminnollisuuteen. VBA ohjelma voi kutsua UA asiakaskirjaston olioita määritellyin metodein, jonka jälkeen ohjelmointirajapintasovellus sekä .NET pino hoitavat kutsut palvelimelle. Sovellus testattiin yksittäisillä komponenteilla, jotta voitiin todeta metodien toiminta. Kokonaisuuden ohjelmointi Microsoft VBA ohjelmointikielellä osoittautui raskaaksi ja toteutetaan myöhemmässä vaiheessa tehokkaammilla ohjelmointimenetelmillä. Sovelluksesta rakennettiin havainnollistamisen vuoksi kokonainen käyttöliittymä, jossa kuitenkin käytössä ovat vain yksi mittaus, koneita tilatiedot sekä tilaus. Kuvassa 29 on esitetty kokonaisen simulointisovelluksen havainne kuva yksinkertaistetulla toiminnoilla.

Kuvan 29 mukaisen simulointisovelluksen toiminta: 1.”Get Endpoints” skannaa verkon palvelimet ja hakee päätepisteet valitulle OPC UA palvelimelle. Päätepisteiden avulla voidaan muodostaa suojattu kanava asiakkaan ja palvelimen välille. 2. Valitaan käytettävä tietoturvasertifikaatti ja yhdistetään ”Connect” –painikkeen painalluksella. 3. Mittauksien, koneiden, venttiilien ja tilatietojen informaatiot, kuten positiot, skaalaukset ja mittayksiköt haetaan PLC- ja valvomogeneroinnin taulukosta automaattisesti. 4. Palkkia liikuttamalla asiakas lähettää muuttuneen arvon palvelimelle käyttäen ”Write()” -palvelua ja ”Nodeid” osoitinta. 5. Näkymän tilatietojen valintaruudut lähettävät muuttuneen

tilan ”Write()” -palvelulla ”Nodeid” –tiedon perusteella. 6. Koneiden ja venttiilien takaisinkytkentätiedot voidaan ottaa käyttöön painamalla ”Subscribe” painiketta, jolloin sovellus luo tilauksen koneiden ja venttiilien ohjaustiloille. Simulointisovelluksen toiminnallisuus antaa venttiileille ja koneille määrätyn ajan kuluessa muuttuneen käynti- ja asentotiedot. Tilauksen ollessa aktiivinen ”GetMonitoredItem” –palvelu jää odottamaan palvelimelta saatavaa tapahtumailmoitusta. 7. Simulointisovellus skaalautuu nuolten suuntaan riippuen käytössä olevien mittausten, koneiden, venttiileiden ja tilatietojen määrästä. 8. Testauksen päättyessä yhteys katkaistaan ”Disconnect” -painikkeesta, jolloin istunto suljetaan. Sovelluksen sulkeutuessa katkaistaan suojattu kanava.

Simulointiohjelmiston toiminta testattiin Siemens S7-1500 sarjan logiikan kanssa edellisen kappaleen vaiheiden mukaisesti. Simulointisovelluksella kirjoitetut arvot siirtyivät logiikan kenttärajapinnan attribuutteihin vaatimusten mukaisesti. Logiikka säilytti OPC UA -asiakkaalta saamansa arvot eikä irti kytketty kenttälaite aiheuttanut arvon päälle kirjoitusta. I/O -rajapinnan lähdöistä saadulla tapahtumailmoituksella voitiin toteuttaa koneen käyntitiedon simulointi logiikalle. Vaste säilyi testitapauksissa alle sekunnin mittaisena. Simulointiohjelmiston testaamisen tuloksena voidaan todeta OPC UA tiedonsiirtoprotokollan toimivan S7-1500 sarjan logiikan kanssa tässä diplomityössä esitettyjen menetelmien mukaisesti.



Kuva 29. *Käyttöliittymä valmiista simulointisovelluksesta*

5. YHTEENVETO

Tämän diplomityön tavoitteena oli tutkia menetelmiä toteuttaa ja liittää simulointiohjelmisto automaatiojärjestelmään ohjelmoitavan logiikan sovelluksen testaamiseksi. Työssä selvitettiin myös simuloinnin ja testaamisen tuomia hyötyjä automaatioprojekteissa. Työssä käsiteltiin prosessien simuloinnin ja automaatiosovelluksen testaamisen kannalta oleellisia asioita, joiden avulla työn toteuttaminen on mahdollista. Tutkimuksen edetessä valittiin käytettävä tiedonsiirtomekanismi ja selvittiin sen ominaisuuksia tarkemmin.

Haasteita tutkimuksessa aiheutti aihealueen laajuus. Sovelluksien testaamiseen, automaatiojärjestelmän tiedonsiirtoon ja prosessien simulointiin löytyy informaatiota laajasti. Annettu päämäärä toteuttaa tehokas ja helppokäyttöinen sovellus sekä prosessin dynamiikan rajaaminen pois simuloinnin vaadituista vasteista helpotti tutkimuksen kohdistamista.

Työn alussa määritettiin tutkimuskysymykset, joihin tutkimustyön on tuotava ratkaisut. Alla on esitetty tutkimuskysymykset sekä ratkaisut työssä esitetyn materiaalin ja toteutuksesta saatujen tulosten mukaisesti.

> *1. Minkälaisin tiedonsiirtomenetelmin ohjelmoidun logiikan muuttujia voidaan manipuloida?*

Ohjelmoitavan logiikan muuttujia voidaan konfiguraation ulkopuolelta manipuloida käyttäen OPC UA tiedonsiirtoa. OPC UA määrittelee logiikan toiminnan UA -palvelimena, jolloin rajapinta kolmannen osapuolten laitteille on universaali. Logiikan sisäisällä UA –palvelimen ja logiikan muuttujan välillä toimii IEC61131-5 mukainen tiedonsiirtolohko, jonka avulla luetaan ja kirjoitetaan I/O -rajapinnan muuttujan arvoa.

> *2. Miten voidaan tehokkaasti luoda sovelluskohtaiset muuttujat simulointiohjelmaan?*

Vaihtoehtoja sovelluskohtaisten muuttujien luomiseen on useampi ja valinta riippuu käytettävistä työkaluista. Ensimmäinen vaihtoehto on hakea OPC UA osoiteavaruuden avulla palvelimella käytettävät I/O -muuttujat sekä niihin liittyvät tiedot, kuten arvo, positio, skaalaus ja mittayksikkö. Toinen vaihtoehto on käyttää pohjana taulukkoa, kuten esimerkiksi I/O –luetteloa, josta tarvitut tiedot voidaan viedä esimerkiksi csv-tiedostoon. Tiedosto voidaan avata ja lukea simulointisovelluksella. Kolmas vaihtoehto on käyttää vastaavaa taulukkoa kuten vaihtoehdossa kaksi, mutta simulointisovellus hakee tiedot suoraan taulukosta. Työn toteutuksessa käytettiin kolmatta vaihtoehtoa, joka todettiin Insta Automation Oy:n mallissa toimivaksi.

> *3. Miten erillinen testausohjelmisto voidaan käyttäjäystävällisesti yksilöidä projektikohtaiseksi, jotta testaaminen olisi vaivatonta ja tehokasta?*

Simulointisovelluksen käyttöliittymään tuodaan I/O –rajapinnan pisteet, jotka on koottu informaation ja toiminnon mukaisiin osioihin. Kussakin simuloinnissa käyttäjälle näkyy kyseisen projektin mukaiset datapisteet, positioituna ja skaalattuna. Projektin yksilöinti tapahtuu jo projektin luontivaiheessa, jolloin simulointisovellus hakee tiedot edellisen kohdan mukaisesti joko projektin suunnitteludokumentaatiosta tai UA osoiteavaruudesta.

Simulointisovelluksen avulla voidaan tukea Insta Automation Oy:n liiketoimintaa tehostamalla automaatiosovellusten toteutusvaihetta. Simulointisovelluksella voidaan saavuttaa laatuvaikutuksia sekä nostaa testausten tehokkuutta. Molemmat vaikuttavat suoraan projektin kustannustehokkuuteen ja aikatauluun. Pitemmällä ajanjaksolla laatuvaikutusten ja kustannussäästöjen vaikutukset voivat vaikuttaa myös asiakastyytyväisyyteen. Simulointisovelluksen merkittävimmät hyödyt konkretisoituvat tehdastestausten selkeytymisessä ja läpivientiajoissa

Simulointisovellus suunniteltiin ja toteutettiin annetut vaatimukset täyttäväksi. Käytetyt menetelmät todettiin prototyypin testauksessa toimiviksi. Haasteeksi kuitenkin muodostui Microsoft VBA ohjelmointikielen kankeus skaalautuvissa projekteissa. Työn aikana kerätyistä materiaaleista syntyi ideoita tuotteen jatkokehitykseen, joiden avulla voidaan tukea sovellusten testausta vieläkin tehokkaammin. Automaatiosovelluksen elinkaaren aikana tehdään testauksia, jotka voidaan integroida simulointisovellukseen. Testaustasojen toimenpiteet sekä dokumentaatiot voidaan yhtenäistää ja luoda automatisoidusti testausdokumentaatioita.

Simulointisovelluksen toiminnallisuutta voidaan laajentaa kattamaan todellisen prosessin vasteita erilaisten siirtofunktioiden ja ehtojen avulla. Siirtofunktioilla voidaan kuvata todellisen prosessin dynamiikkaa, jolloin automaatiojärjestelmän testaaminen vastaa enemmän todellista tilannetta. Testaaja voi lisäksi toteuttaa ennalta määrättyjä tapahtumaketjuja mukautettujen toiminnollisuuksien avulla. Prosessin dynamiikka ja tapahtumia voidaan mallintaa myös diskreetein tilamallein, joka sopii ainakin kyseiselle VBA toteutukselle.

LÄHTEET

[1] G. Hassapis, Soft-testing of industrial control systems programmed in IEC 1131-3 languages, ISA Transactions, Vol. 39, Iss. 3, 2000, pp. 345-355. Available (accessed ID: 271436): http://www.sciencedirect.com/science/article/pii/S001905780000029X.

[2] T. Tommila, Laatu automaatiossa - parhaat käytännöt, Suomen Automaatioseura, Suomi, 2001, 245 p.

[3] J. Hirvonen, K. Hukki, T. Tommila, M. Strömman, Automaatiosuunnittelun prosessimalli. Yhteiset käsitteet verkottuneen suunnittelun perustana, Suomen Automaatioseura ry Finnish Society of Automation, 2007, 43 p.

[4] I. Haikala, J. Märijärvi, Ohjelmistotuotanto, 10. uud. p. ed. Talentum Oyj, Helsinki, Suomi, 2004, 440 p.

[5] M. Iacob, G.-. Andreescu, N. Muntean, SCADA system for a central heating and power plant, 2009 5th International Symposium on Applied Computational Intelligence and Informatics, Timisoara, Romania, May 28-29, 2009, IEEE, pp. 159-164.

[6] Center For Chemical Process Safety, Guidelines for Safe Automation of Chemical Processes, Second edition. ed. Wiley-AIChE, US, 2017, 648 p.

[7] ISO 10628, Flow diagrams for process plants - General rules, International Organization for Standardization, Geneve, 1997, 64 p.

[8] M. Barker, J. Rawtani, Practical Batch Process Management, Elsevier Science & Technology, Oxford, 2004, 192 p.

[9] Panosprosessien automaatio, in: VTT Tiedotteita - Meddelanden - Research Notes 1487, VTT, 1993, pp. 175.

[10] Åkerman Magnus, Implementing Shop Floor IT for Industry 4.0, Chalmers University of Technology, Gothenburg, Sweden, 2018, 62 p. Available: https://research.chalmers.se/publication/502937/file/502937_Fulltext.pdf.

[11] ISA 95.00.01, Enterprise-Control System Integration − Part 1: Models and Terminology, International Society of Automation, 2010, 154 p.

[12] D. Bailey, E. Wright, Practical SCADA for Industry, Elsevier Science & Technology, Oxford, UNITED KINGDOM, 2003, 304 p.

[13] H. Carlsson, B. Svensson, F. Danielsson, B. Lennartson, Methods for Reliable Simulation-Based PLC Code Verification, IEEE Transactions on Industrial Informatics, Vol. 8, Iss. 2, 2012, pp. 267-278. https://ieeexplore.ieee.org/document/6121945.

[14] W. Bolton, Programmable Logic Controllers, Elsevier Science & Technology, Jordan Hill, United Kingdom, 2014, 304 p.

[15] I. Ahmed, S. Obermeier, S. Sudhakaran, V. Roussev, Programmable Logic Controller Forensics, IEEE Security & Privacy, Vol. 15, Iss. 6, 2017, pp. 18-24. http://ieeexplore.ieee.org/document/8123493.

[16] BS EN 61131-1:2003, Programmable controllers. General information, British Standards Institute, 2003, 232 p.

[17] K. John, M. Tiegelkamp, IEC 61131-3: programming industrial automation systems, 2. ed. ed. Springer, Berlin ; Heidelberg, 2010, 390 p.

[18] D.H. Hanssen, Programmable Logic Controllers : A Practical Approach to IEC 61131-3 using CoDeSys, John Wiley & Sons, Incorporated, New York, 2015, 416 p.

[19] IEC 61131-3, Programmable controllers Part 3: Programming languages, 2013, 234 p.

[20] F. Bonfatti, P.D. Monari, U. Sampieri, IEC 1131-3 programming methodology, CJ International, Seyssins, France, 1997, 336 p.

[21] Automaatio: Automation: Osa 2, Ohjelmointi ja teollisuusprosessien valvonta, Suomen standardisoimisliitto SFS ry, Helsinki, Finland, 2006, 514 p.

[22] OPC UA .NET Client for the SIMATIC S7-1500 OPC UA Server, Siemens Industry Online Support, 2018, 56 p.

[23] Automaatio 1, ELEC-C1210, Aalto yliopisto, 2014.

[24] H. Kühnle, Distributed Manufacturing, Springer, 2009, 205 p.

[25] An inside look at industrial Ethernet communication protocols, Texas Instruments, 2018, 7 p.

[26] R. Asp, T. Tuominen & H. Hyppönen, Kunnossapito - menestystekijä: 2. Automaatiojärjestelmä, http://www03.edu.fi/oppimateriaalit/kunnossapito/sahkotekniikka_a2_automaatiojarjestelma.html.

[27] Omron communications. Available (accessed 13.3.2019): http://www.ia.omron.com/products/family/3074/.

[28] Siemens communications. Available (accessed 13.3.2019): https://w3.siemens.com/mcms/programmable-logic-controller/en/advanced-controller/s7-1500/communication/pages/communication.aspx.

[29] Mitsubishi communications. Available (accessed 13.3.2019): https://www.mitsubishielectric.com/fa/products/cnt/plcr/pmerit/network/index.html.

[30] Schneider communications. Available (accessed 13.3.2019): https://www.schneider-electric.com/en/product-category/2400-industrial-communication/.

[31] Beckhoff communications. Available (accessed 13.3.2019): https://www.beckhoff.com/english.asp?twincat/tf6100.htm.

[32] Phoenix communications. Available (accessed 13.3.2019): https://www.phoenixcontact.com/online/portal/pi?1dmy&urile=wcm%3apath%3a/pien/web/main/products/subcategory_pages/Drivers_and_interfaces_P-19-07/1429bca8-7058-491d-b8fb-75827cff08b5.

[33] Rockwell communications. Available (accessed 13.3.2019): https://literature.rockwellautomation.com/idc/groups/literature/documents/pp/logix-pp002_-en-p.pdf.

[34] W. Mahnke, S. Leitner, M. Damm, OPC unified architecture, Springer, Berlin, 2009, 339 p.

[35] OPC Foundation, OPC Foundation, web page. Available (accessed 3.1.2019): www.opcfoundation.org.

[36] E. Moraes, H. Lepikson, S. Konstantinov, J. Wermann, A.W. Colombo, B. Ahmad, R. Harrison, Improving connectivity for runtime simulation of automation systems via OPC UA, 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), pp. 288-293.

[37] Unified Automation: Products, web page. Available (accessed 6.3.2019): https://www.unified-automation.com/products.html.

[38] OPC Unified Architecture e-book, CAS, web page. Available (accessed 25.02.2019): http://www.commsvr.com/UAModelDesigner/.

[39] C. Larman, Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process, Upper Saddle River, NJ, United States, 2002, 627 p.

[40] J.P. Kasurinen, Ohjelmistotestauksen käsikirja, Dodelco, Jyväskylä, 2013, 204 p.

[41] OMG - Object Management Group: About the Unified Modeling Language Specification Version 2.5, Object Management Group, web page. Available (accessed 26.02.2019): https://www.omg.org/spec/UML/2.5/About-UML/.

[42] M. Hurme, I. Turunen, S. Väyrynen, A. Vuori, Turvallisuus prosessien suunnittelussa ja käyttöönotossa, VTT, 2002, Available: http://virtual.vtt.fi/virtual/proj3/alarp/aineisto/luento-09-moduuli-02.pdf.

[43] Y. Hu, N. Lin, Automatic black-box method-level test case generation based on constraint logic programming, 2010 International Computer Symposium (ICS2010), pp. 977-982.

[44] P.P. Schmidt, A. Fay, Transformation of continuous simulation models of automated manufacturing systems into discrete event models on different levels of detail, 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, pp. 1-5.

[45] Programming Guide: Modicon M241 Logic Controller, Schneider Electric, 2018.

[46] Industry Online Support; Product Support: OPC UA methods for the SIMATIC S7-1500 OPC UA server, Siemens, web page. Available (accessed 03.03.2019): https://support.industry.siemens.com/cs/document/109756885/opc-ua-methods-for-the-simatic-s7-1500-opc-ua-server-?dti=0&lc=en-WW.

[47] Machine Automation Controller: CPU Unit OPC UA - User's Manual, OMRON, 2018.

[48] S. Cavalieri, F. Chiacchio, Analysis of OPC UA performances, Computer Standards & Interfaces, Vol. 36, Iss. 1, 2013, pp. 165-177. https://www.sciencedirect.com/science/article/pii/S0920548913000640.

[49] C. Pogacean, S. Broschei & G. Süss, Implementing Deterministic OPC UA Communication, https://www.automation.com/pdf_articles/softingna/OPCUAPublisherSubscriber_W_EN_160407_100.pdf?utm_source=bm23&utm_medium=email&utm_term=Image+-+Arrow&utm_content=White+Paper+Alert+-+Implementing+Deterministic+OPC+UA+Communication+%257C+Vision+Systems+for+Collaborative+Robots&utm_campaign=White+Paper+Alert+-+March+17,+2017.

[50] Industry Online Support; Product Support: OPC UA Client Library, Siemens, web page. Available (accessed 03.03.2019): https://support.industry.siemens.com/cs/document/109748892/opc-ua-client-library?dti=0&lc=de-WW.

[51] .NET Stack and Sample Applications. Available (accessed 03.03.2019): https://opcfoundation.org/developer-tools/samples-and-tools-unified-architecture/net-stack-and-sample-applications/.

LIITE A: OPC UA PALVELUIDEN KUVAUKSET

Method	Explanation
FindServers	Searches for OPC UA servers in the network. Requirement: A LDS (Local Discovery Server) or GDS (Global Discovery Server) has to be available.
GetEndpoints	Determines the available endpoints on a server via which a connection can be established.
Connect	Establishes a connection to a server and creates a secure channel and a session to the server.
Disconnet	Ends an existing session and disconnects the connection to the server.
BrowseRoot	Returns a collection of nodes that can be found in the root directory of the server.
BrowseNode	Returns a collection of nodes that can be hierarchically found in a specific node.
BrowseByReferenceType	Returns a collection of nodes that can be found in a specific node by a specific reference type.
GetNamespaceUri	Returns the Namespace Uri related to an submitted Namespace Index.
GetNamespaceIndex	Returns the Namespace Index related to an submitted Namespace Uri.
GetNamespaceArray	Returns the Namespace Array.
Subscribe	Creates a subscription on the server.
RemoveSubscription	Deletes a specific subscription from the server.
AddMonitoredItem	Adds a MonitoredItem for monitoring an existing subscription.
AddEventMonitoredItem	Adds a (Event)MonitoredItem for monitoring an existing subscription.
RemoveMonitoredItem	Deletes an existing MonitoredItem of a subscription.
ReadNode	Reads the metadata of a specific node.
ReadValues	Reads the values of a variables node.
WriteValues	Writes values to node variables.
ReadStructUdt	Reads values of user-defined structures and UDTs with the help of the "TypeDictionary" of the server.
WriteStructUdt	Writes values to user-defined structures and UDTs that were read before.
RegisterNodeIds	Registers node IDs at the server for an optimized access to the nodes.
UnregisterNodeIds	Deletes the registration of already registered node IDs.
GetMethodArguments	Determines the available input and output arguments of a method.
CallMethod	Calls a method on a server.
Notification_CertificateValidation	Event that is fired when a server certificate cannot be accepted.
Notification_MonitoredItem	Event that is fired when the value of a MonitoredItem is
Notification_KeepAlive	Event that is fired when the value of a KeepAliveNotification arrives.
Notification_MonitoredEventItem	Event that is fired when a OPC UA-Event arrived.