Aleksi Hänninen

Pakkauslinjan kommunikointijärjestelmä

Opinnäytetyö
Kevät 2022
Tietotekniikan insinööri (AMK)

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Aleksi Hänninen

Työn nimi: Pakkauslinjan kommunikointijärjestelmä

Ohjaaja: Jyri Lehto

Vuosi: 2022 Sivumäärä: 42 Liitteiden lukumäärä: 5

Tässä opinnäytetyössä toteutettiin yläjärjestelmäkommunikointi käyttäen Socket ja OPC UA -protokollia sekä SQL-tietokantayhteyttä. Asiakas oli tilannut uuden kelojen pakkauslinjan Pesmel Oy:ltä, tässä työssä keskityttiin itse kommunikointiinjärjestelmään eikä niinkään käytössä olevaan laitteistoon. Opinnäytetyö alkaa arkkitehtuurin ja järjestelmän esittelyllä ja kommunikointijärjestelmien teorialla. Sitten käydään läpi ohjelmointi- ja testausvaiheet ja lopuksi on tulokset ja pohdinta

Lopputuloksena oli toteutettu ohjelmistokokonaisuus joka keskusteli asiakkaan oman järjestelmän kanssa socket-viesteillä ja lukee ja kirjoittaa dataa laitetasolle käyttäen OPC UA -kirjastoja.

Tämä opinnäytetyö on osittain salattu.

Avainsanat: SQL, OPC UA, Java, C#, Socket, industry

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Aleksi Hänninen

Title of thesis: Packing line communication system

Supervisor: Jyri Lehto

Year: 2022 Number of pages: 42 Number of appendices: 5

The thesis was about upper level system communication made using socket and OPC UA protocols and SQL-database connections. A Customer had ordered a new wire-coil packing line from Pesmel Oy. The thesis focused on the actual communication system and not so much the equipment involved. The thesis was started by studying on the architecture of the system, the different communication protocols and programming languages that were going to be used. Next attention was paid to the programming of the software and the testing of the programs.

As the final result of the thesis there was a software package that is capable of communicating with the customers own computer systems through socket messaging and communication with the device layer using OPC UA program libraries.

The thesis is partly classified.

Keywords: SQL, OPC UA, Java, C#, Socket, industry

SISÄLTÖ

Kuva-, ja kuvioluettelo

Käytetyt termit ja lyhenteet

ACK	Lyhenne englannin kielen sanasta acknowledgement. Yläjärjestelmä lähettää kuittauksen, kun viesti on saapunut ja L2-järjestelmä lähettää kuittauksen, kun se on vastaanottanut viestin.
ANS	Lyhenne englannin kielen sanasta answer. Viesti, joka lähetetään vastauksena tiettyjen viestin saapumisen jälkeen
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange on 1960-luvulla kehitetty 7-bittinen merkistökoodausjärjestelmä, joka on laajasti käytössä vielä nykyään.
EBCDIC	**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode on 1960-luvulla kehitetty 8-bittinen merkistökoodausjärjestelmä, joka on käytössä vanhemmissa IBM:n suur- ja minitietokoneissa.
HEADER	Header eli suomeksi ”ylätunniste” on viestin alkuun lisätty tieto, joka kertoo viestin tyypin, viestin pituuden, lähettäjän ja vastaanottajan.
HIBERNATE SQL	Hibernate SQL on JPA-kirjaston toteutus, joka mahdollistaa oliopohjaisen relaatiokartoituksen tietokannan kanssa (Object Relational Mapping (ORM)).
HMI	**H**uman **M**achine **I**nterfacet ovat pakkauslinjoilla käytössä olevia käyttöliittymiä joilla ohjataan linjaston eri osuuksia.
IP	**I**nternet **P**rotocol on TCP/IP-mallin internetkerroksen protokolla.
JAVA	Java on Sun Microsystemsin vuonna 1995 julkaisema oliopohjainen ohjelmointikieli.

JPA	**J**ava **P**ersistance **A**PI on relaatiotietokantoihin perustuva malli, joka mahdollistaa tietokantakommunikoinnin Java-ohjelmointikielellä.
L1	Lyhenne sanoista Level 1, joka tarkoittaa PLC-tasoa eli laitetasoa
L2	Lyhenne sanoista Level 2, joka tarkoittaa tässä projektissa Pesmelin kommunikointijärjestelmää
L3	Lyhenne sanoista Level 3, joka tarkoittaa tässä projektissa asiakkaan yläjärjestelmää
NACK	Lyhenne englannin kielen sanoista Not acknowledged, joka tarkoittaa, että jokin on pielessä viestin saapumisessa tai sen datassa on jotain vikaa.
OPCUA	**OPC U**nified **A**rchitecture on tietokoneiden välinen kommunikointiprotokolla, joka on käytössä teollisessa automaatiossa. Teknologiaa kehittää OPC Foundation.
PLC	**P**rogrammable **L**ogic **C**ontroller eli ohjelmoitava logiikka on pieni tietokone, jota käytetään automaatioprosessien ohjauksessa.
PLC TAG	PLC:llä olevia merkkijonoja, jotka on linkitetty PLC:n data block -muistipaikkoihin. Näiden avulla Databridge-ohjelma saa yhteyden PLC:hen.
RW	**R**adial **W**rapping eli suomeksi radiaalinen käärintä on kelan ympärikäärintä johonkin materiaaliin.
SOAP	**S**imple **O**bject **A**ccess **P**rotocol on tietoliikenneprotokolla, joka mahdollistaa proseduurien etäkutsut. Se on toimintaperiaatteeltaan saman tyyppinen kuin muut RCP-protokollat, mutta pohjautuu XML-kieleen.

SOCKET	Rajapinta tiedon lähettämiseen ja vastaanottamiseen päätepisteiden välillä joko verkossa tai sovellusten välissä.
SPRING BOOT	Spring boot on avoimen lähdekoodin Java-pohjainen ohjelmistokehys, jolla voidaan luoda mikropalveluita.
SQL	**S**tructured **Q**uery **L**anguage on kyselykieli, jolla tietokantaan voidaan tehdä erilaisia hakuja, muutoksia ja lisäyksiä.
TARKISTUSSUMMA	Tarkistussumma lasketaan jokaiselle viestille. Tunnetaan englanninkielisellä nimellä Checksum. Jos L2- ja L3-tarkistussummat eroavat, tarkoittaa se, että viestin datassa on mennyt jokin pieleen.
TCP	**T**ransmission **C**ontrol **P**rotocol on yleinen internetin tietoliikenneprotokolla, joka mahdollistaa tavujen turvallisen lähetyksen.
TELEGRAM	Socket-ohjelmassa viestien rakenne määräytyy ns. sähkeiden eli Telegramien avulla. Riippumatta siitä, lähtevätkö viestit yläjärjestelmään vai tulevatko ne sieltä, tarvitaan sähke, joka määrittää viestin rakenteen, datatyypit ja viestien eri osien pituudet.
TEW	**T**rough **E**ye **W**rapping eli suomeksi silmästäkäärintä on prosessi, jossa kela kääritään kreppipaperiin ja/tai muoviseen kutistekalvoon. Kreppipaperi suojaa kelaa kosteudelta, ja kutistekalvo suojaa kreppipaperia ja kelaa.
XML	**Ex**tensible **M**arkup **L**anguage on hyvin yleisessä käytössä oleva merkintäkielien standardi, jolla tiedon merkitys on kuvattavissa tiedon sekaan.

1 Johdanto

1.1 Työn tausta

Yläjärjestelmäkommunikointi on tärkeä osa teollisuuden projekteja. Haasteet, joita voi syntyä, vaihtelevat riippuen asiakkaan omista tietokonejärjestelmistä, käytössä olevista teknologioista ja asiakkaan vaatimista ominaisuuksista.

Teräsfirma Japanissa tilasi Pesmel Oy:ltä uuden rautakelojen pakkauslinjan. Pesmel Oy on erikoistunut metalli- ja paperikelojen pakkauslaitteistoihin ja varastointiratkaisuihin. Tässä projektissa tarvitaan yläjärjestelmäkommunikointia ja se päätettiin toteuttaa käyttäen socket- ja OPC UA -protokollia.

1.2 Työn tavoite

Työn tavoitteena oli tehdä yläjärjestelmäkommunikointiohjelmisto, joka kommunikoi tehtaalla olevan yläjärjestelmän kanssa lähettäen ja vastaanottaen tarvittavaa tietoa pakkausprosessista sen eri vaiheissa.

Raportointi ja eri pakkausvaiheissa olevien kelojen tiedot tuli lähettää yläjärjestelmälle ja yhteyden tuli olla katkeamaton

1.3 Työn rakenne

Luvussa kaksi käydään läpi järjestelmää sen eri tasoja, käytössä olevia viestityyppejä ja järjestelmän arkkitehtuuria. Tämän jälkeen luvussa kolme käydään läpi käytetyt teknologiat, niiden ominaisuudet ja käytetyt ohjelmointikielet. Työvaihe ja siihen liittyvät ohjelmointitehtävät esitetään luvussa neljä. Luvussa viisi käydään läpi järjestelmän testaaminen ennen varsinaista käyttöönottoa. Lopulliset tulokset käydään läpi luvussa kuusi. Luvussa seitsemän käydään läpi yhteenveto ja pohdinta opinnäytetyöstä.

2 Järjestelmän arkkitehtuuri

2.1 Level 3 -taso

L3-taso on asiakkaan oma IBM-suurtietokonejärjestelmä, joka tunnetaan tässä projektissa myös nimellä Business Computer. Asiakas lähettää viestejä Ebcdic-muodossa L2-tasolle socket-yhteydellä. L2-taso käsittelee viestit, muuntaa ne ASCII-muotoon ja tallentaa tietokantaan myöhempää L1-tason prosessointia varten.

Viestejä on useita erilaisia ja niitä lähetetään pakkauslinjan prosessin eri vaiheissa mm:

- Aloitus (Startup)
- Sisääntulo (Infeed)
- Pakkaus (Packing)
- Ulostulo (Outfeed)
- Testaus (Testing).

Asiakkaan lähettämät viestit ja niiden tyyppitunnisteet

Ks. Liite 2, Asiakkaan lähettämät viestit ja niiden tyyppitunnisteet.

Asiakkaalle lähetetettävät viestit ja niiden tyyppitunnisteet

Ks. Liite 3, Asiakkaalle lähetettävät viestit ja niiden tyyppitunnisteet.

2.2 Level 2 -taso

L2-taso on Pesmelin järjestelmä, joka sisältää socket-ohjelman, joka on kirjoitettu C#-ohjelmointikielellä, OPC UA -ohjelman, joka on kirjoitettu Java-ohjelmointikielellä, ja SQL-tietokannan, joka on tässä projektissa Microsoft SQL-database 2018 -ohjelmisto.

Viestien käsittely alkaa viestin saapumisesta socket-ohjelmaan. Viesti muunnetaan XML-muotoon telegramin viestirakennetta käyttäen. Jokaiselle viestille on omat jäsentämismallinsa. Viestiin lisätään ylätunniste ja lasketaan tarkistussumma (Checksum), joka lisätään viestin loppuun. Tarkistussummaa käytetään varmistamaan, että data on oikeanlaista.

Seuraavaksi viesti tallennetaan SQL-tietokantaan ja sen tilaksi asetetaan ’2’, joka tarkoittaa, että viesti on käsitelty.

Viesti on nyt valmis luettavaksi Databridge-ohjelmassa, joka odottaa, että PLC asettaa viestin trigger bitin virittyneeseen tilaan ja ohjelma aloittaa tietokannan lukemisen. Tiedot lähetetään PLC:lle käyttäen OPC UA:n TCP-protokollaa.

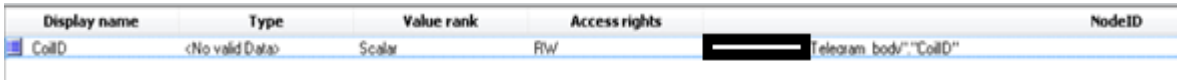
Tämän jälkeen Databridge asettaa viestin tilan SQL-tietokannassa käsitellyksi.

Viestin lähetys toiseen suuntaan toimii vastakkaisessa järjestyksessä: ensin PLC asettaa oman trigger bittinsä virittyneeseen tilaan, jonka jälkeen OPC UA lukee PLC:ltä tarvittavat tiedot, jotka on sidottu PLC tageihin. Tämän jälkeen muodostetaan viesti, joka tallennetaan tietokantaan. Socket-ohjelma lukee tietokantaa jatkuvasti ja havaitsee, että siellä on käsittelemätön viesti. Tässä vaiheessa lisätään viestiin ylätunniste ja luodaan tarkistussumma ja lisätään se viestin perään. Yläjärjestelmä vastaa omalla ACK-viestillään.

2.3 Level 1 -taso

Level 1 -taso on PLC-taso eli laitetaso. Tässä projektissa käytetään Siemens S4-700 -logiikkaa. Pääasiallinen PLC ohjaa suurinta osaa laitteista ja säilyttää muistipaikoissa viestin sisällön ja ns. trigger bitit, jotka virittyessään lähettävät

viestin eteenpäin. Data säilötään muistipaikkoihin, joista käytetään nimitystä DataBlock. Muistipaikkoihin pääsee käsiksi OPC UA:n kautta käyttämällä node-osoitteita.



Kuva 1. Esimerkki Node-ID:stä

Yläpuolella näkyvässä kuvassa 1 on esimerkki muistipaikasta, jossa säilytetään yhden kelan ID-tunnusta yläjärjestelmälle lähetettävää viestiä varten.

2.4 Socket-ohjelma

Socket-ohjelma kuuntelee yläjärjestelmästä tulevia viestejä, tallentaa niitä tietokantaan ja vastaa omilla ACK- ja ANS-viesteillään. Socket-ohjelma hoitaa viestien tallennuksen ASCII- ja Ebcdic-muodoissa. Viestit lähetetään ja vastaanotetaan Ebcdic-muodossa. Ne tulee muuntaa ASCII-muotoon ennen tietokantaan tallennusta. Socket-ohjemaan lisättiin myös testiviestien lähetysruutu helpottamaan asiakkaan kanssa testaamista.

Socket-ohjelman käyttöliittymä selitettynä

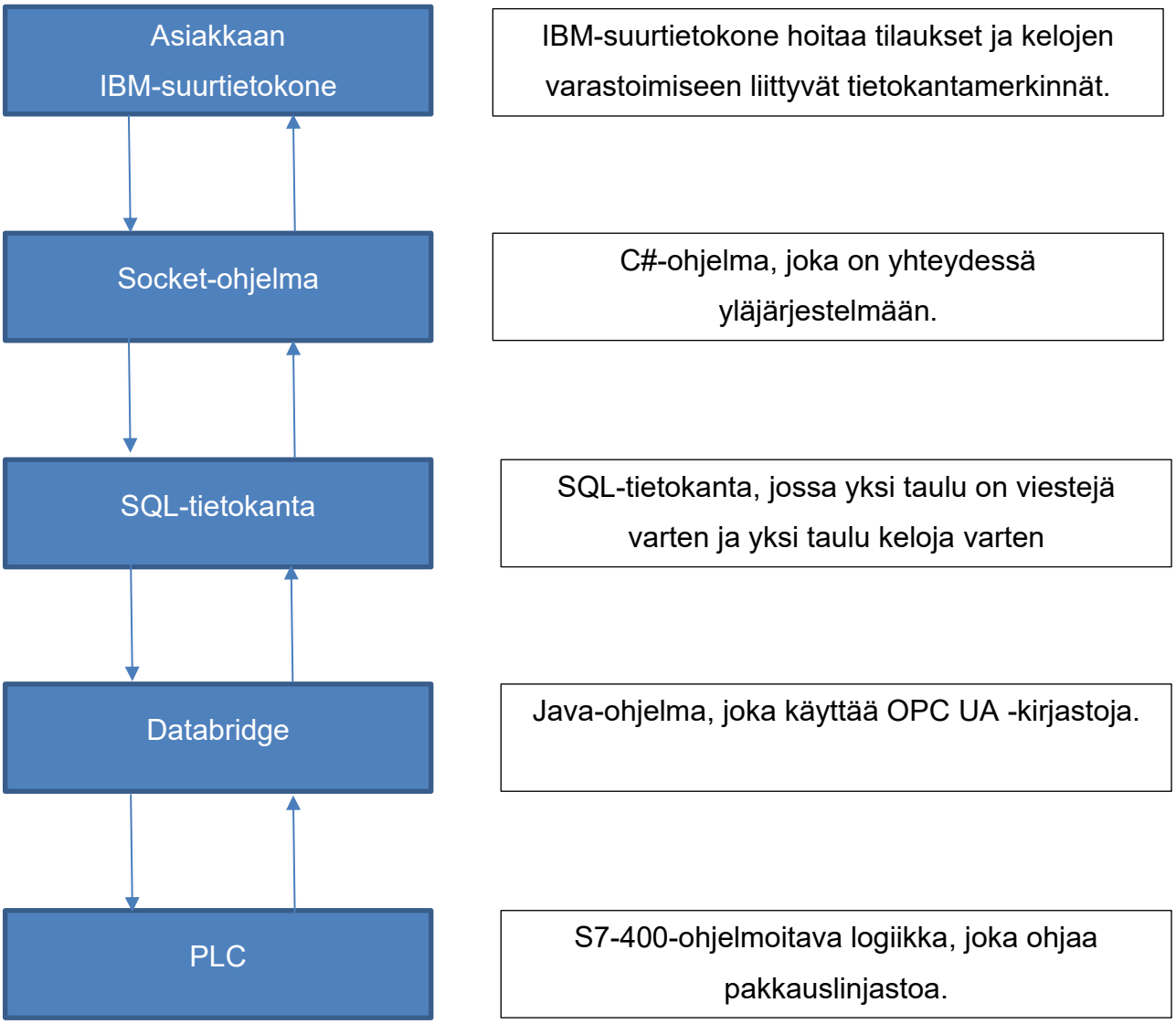
Ks. Liite 1, Socket-ohjelman käyttöliittymä selitettynä.

2.5 Databridge-ohjelma

Databridge-ohjelma toteutetaan spring boot -projektina. Ohjelma käyttää OPC UA -kirjastoja ja nämä mahdollistavat PLC:n muistipaikkojen kirjoittamisen ja lukemisen. Ohjelma toteutetaan ilman käyttöliittymää ns. konsolisovelluksena. Tämä sovellus ohjelmoidaan Java-ohjelmointikielellä. Databridge-ohjelmassa on myös tietokantayhteys.

Databridge käsittelee viestejä ainoastaan ASCII-muodossa ja jättää Ebcdic-muunnokset socket-ohjelman vastuulle.

2.6 Järjestelmän arkkitehtuuri kaaviona

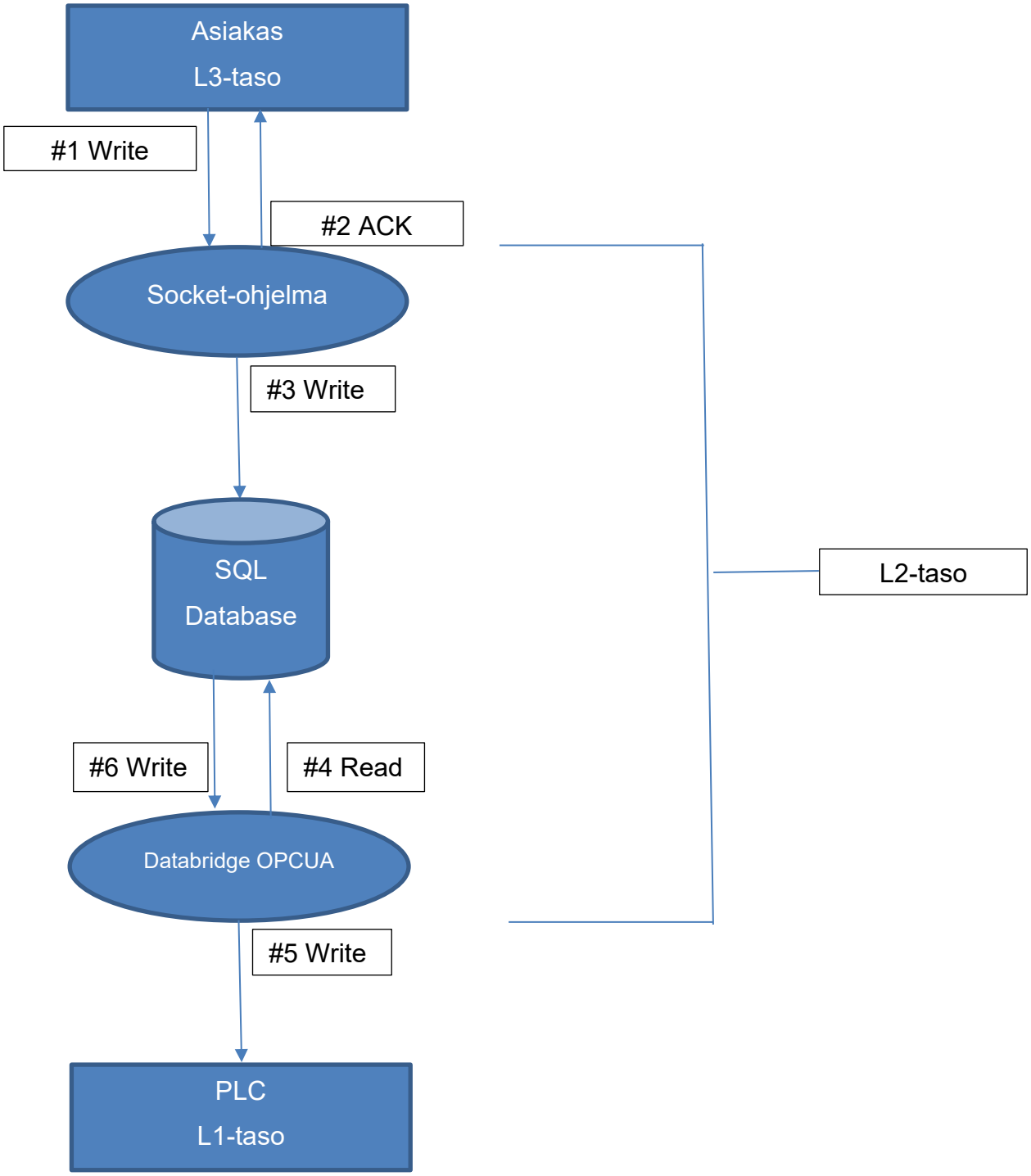


Kuvio 1. Järjestelmän arkkitehtuuri

Ylimmällä tasolla on asiakkaan IBM-suurtietokone, joka on lähettää alimpien tasojen tarvitsemia tietoja käyttäen socket-yhteyttä. Tästä ylimmästä tasosta käytetään nimitystä level 3.

Socket-yhteys muodostetaan socket-ohjelmaan, joka ottaa viestit vastaan, ja paloittelee ne viestityypin mukaan ennen tallennusta tietokantaan. Databridge-ohjelma lukee tietokantaa ja havaitsee uudet merkinnät ja siirtää datan alimmalle tasolle eli PLC-tasolle. Jos dataa halutaan lähettää level 3 -tasolle, aloitetaan datan lukeminen PLC-tasolta ja se siirtyy databridge-ohjelman kautta tietokantaan. Socket-ohjelma havaitsee uuden datan ja tekee tarvittavat toimenpiteet ennen viestin lähetystä level 3 -tasolle.

2.6.1 Kommunikointijärjestelmä: viestin vastaanottaminen

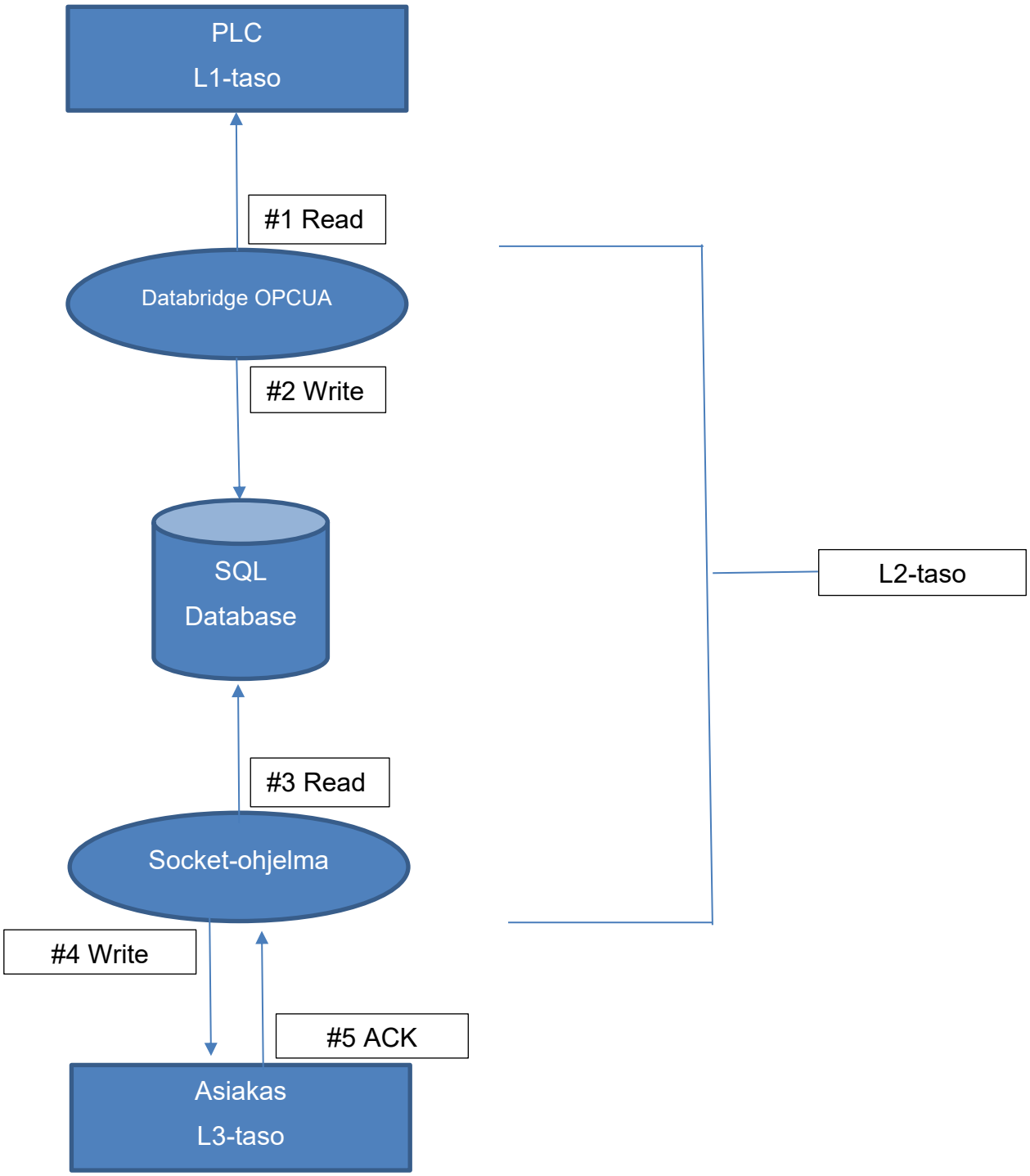


Kuvio 2. Viestin vastaanottaminen.

Kun level 3 -taso lähettää uuden viestin järjestelmään, prosessi on seuraavanlainen:

1. Level 3 -taso lähettää viestin käyttäen socket-yhteyttä socket-ohjelmalle.
2. Socket-ohjelma kuittaa viestin vastaanotetuksi ACK-viestillä.
3. Socket-ohjelma kirjoittaa SQL-tietokantaan uuden merkinnän.
4. Databridge-ohjelma havaitsee uuden merkinnän tietokannassa.
5. Databridge-ohjelma kirjoittaa PLC:n muistipaikkoihin tiedot riippuen viestityypistä.
6. Databridge-ohjelma kuittaa tietokanta-merkinnän käsitellyksi ja jää odottamaan uutta työtehtävää.

2.6.2 Kommunikointijärjestelmä: viestin lähettäminen



Kuvio 3. Viestin lähettäminen.

Kun level 1 -tasolla on level 3 -tasolle uusi viesti lähetettäväksi, prosessi on seuraavanlainen:

1. Databridge-ohjelma lukee PLC:n muistipaikkoja ja havaitsee, että jonkin viestin trigger bit on virittyneessä tilassa.
2. Databridge lukee PLC:ltä tarvittavat tiedot riippuen viestityypistä ja kirjoittaa uuden merkinnän SQL-tietokantaan.
3. Socket-ohjelma havaitsee, että SQL-tietokannassa on uusi merkintä ja lukee merkinnästä tarvittavat tiedot uutta viestiä varten.
4. Socket-ohjelma lähettää viestin level 3 -tasolle.
5. Level 3 -taso kuittaa viestin saapumisen ACK-viestillä.

3 Käytetyt teknologiat

3.1 OPC Unified Architecture

OPC UA on OPC Foundationin kehittämä yhteensopivuusstandardi, joka mahdollistaa turvallisen ja luotettavan datan vaihdon teollisen automaation laitteistoissa ja teollisuudessa. Se on alustasta riippumaton ja monikäyttöinen. Standardia on kehitetty yhdessä teollisuuden ammattilaisten ja ohjelmistokehittäjien kanssa ja se mahdollistaa nykyaikaiset ominaisuudet, kuten reaaliaikaisen datan tarkastelun, hälytysseurannan ja datan loggauksen. (OPC foundation 2019.)

3.1.1 Eritasoista suojausta

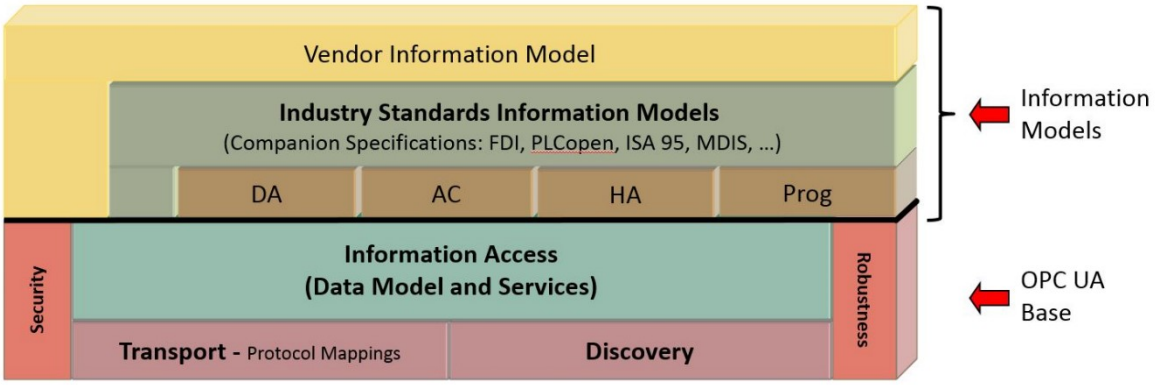
Yksi tärkeimmistä valintakriteereistä käytettävää teknologiaa valittaessa on turvallisuus. OPC UA on palomuuriystävällinen ja tarjoaa sarjan erilaisia turvallisuusominaisuuksia. (OPC foundation 2019.)

Datan liikkuvuus on turvattu erilaisilla protokollilla mm.

- **Istunnon salaus**: Viestit salataan 128- tai 256-bittisillä salaustasoilla
- **Viestien allekirjoitus:** Viestit saapuvat täsmälleen sellaisina kuin ne on lähetettykin.
- **Sekvenssipaketit:** Altistuminen paketin uudelleenlähetyshyökkäyksille on estetty sekvensoimalla.
- **Autentikointi:** Jokainen UA-käyttäjä ja -palvelin tunnistetaan käyttäen OpenSSL-sertifikaatteja, jotka kertovat, mitkä järjestelmät saavat yhdistää toisiinsa.
- **Käyttäjien seuranta:** Sovellukset voivat ottaa käyttöön käyttäjien seurannan, joka mahdollistaa käyttäjätunnusten ja salasanojen, sertifikaattien ja käyttäjätasojen käytön.
- **Auditointi:** Pitää kirjaa käyttäjien tekemistä muutoksista. (OPC foundation 2019.)

3.1.2 OPC-tietomalli

OPC UA -informaatiomallinnusrakenne muuntaa dataa informaatioksi. Oliopohjaisella järjestelmällä on mahdollista mallintaa monitasoisia järjestelmiä. (OPC foundation 2019.)



Kuva 2. OPC-tietomalli (OPC foundation 2019).

3.2 SQL

SQL mahdollistaa datan muokkauksen SQL-kommenoilla. Data on muotoiltu riveiksi, sarakkeiksi, soluiksi ja tauluiksi. Ohjelmoija voi saavuttaa paljon kokoelmapainotteisilla operaatioilla ja teknologia mahdollisti suuria suorituskykyparannuksia tiedonhakuun ja prosessointiin. (Chapple 2019.)

3.2.1 SQL-komennot

SQL-komennot voidaan jakaa kahteen eri osa-alueeseen. Data definition language (DDL) sisältää komennot, kuten CREATE, DROP ja USE, jotka liittyvät tietokantojen luontiin, poistoon ja käyttöön. Kun tietokanta ja sen rakenne on luotu käyttäen DDL-pohjaisia komentoja, voivat järjestelmänvalvojat ja käyttäjät käyttää Data manipulation languagea (DML), joka sisältää datan muokkaamiseen liittyvät komennot, kuten INSERT, UPDATE, SELECT ja DELETE. (Chapple 2019.)

3.2.2 Ominaisuudet

Tallennetut proseduurit. SQL mahdollistaa tallennettujen proseduurien käytön. Tallennetut proseduurit ovat SQL-komentoja, jotka voivat ottaa parametereja sisään ja palauttaa arvoja. Proseduurit säilyvät SQL-tietokannassa ja niiden ajamista voidaan kutsua muista ohjelmista normaalin SQL-tietokantayhteyden kautta. Proseduureille voidaan antaa alkuarvot ja ne tekevät kyselyitä ja muutoksia tietokantaan. (IBM 2021b.)

SQL-proseduurit mahdollistavat koteloidun logiikan, jota pystytään kutsumaan ns. alirutiinina (IBM 2021b).

Ajoitetut tehtävät. SQL Server Agent käyttää SQL-palvelinta varastoidessaan työtehtäviä (Job). Työtehtävät sisältävät yhden tai useamman askeleen (Step). Jokainen askel sisältää oman tehtävän (Task), kuten esimerkiksi tietokannan varmuuskopioinnin. (Microsoft 2021.)

3.3 EBCDIC-merkistökoodaus

EBCDIC on 1960-luvulla kehitetty 8-bittinen merkistökoodauskieli, jota käytetään joissain IBM-suur- ja keskiluokan tietokoneissa. EBCDIC perustuu vanhoissa reikäkorteissa käytettävään järjestelmään. (Pediaa 2022.)

Kuva 3. EBCDIC-reikäkortti (Nikevich 2011).

3.3.1 Yhteensopivuus ASCII-järjestelmän kanssa

EBCDIC eroaa ASCII-merkistökoodauksesta kokonaisvaltaisesti. EBCDIC-merkistössä alaindeksissä olevat kirjaimet ovat kooditaulukossa ennen yläindeksissä olevia kirjaimia ja kirjaimet ennen numeroita. Ne ovat täysin vastakkaisessa järjestyksessä ASCII-merkistöön verrattuna. (IBM, [Viitattu 10.05.2022].)

Tämä tekee koodin muuntamisesta ASCII:n ja EBCDIC:in välillä lähes mahdottomaksi varsinkin kun EBCDIC-koodistosta puuttuu myös aaltosulkumerkit (Pediaa 2022).

3.4 Socket-protokolla

Socket-protokollaa käytetään yleisimmin asiakasohjelman ja palvelimen välisessä viestinnässä. Yleinen käytäntö on asettaa palvelin yhteen tietokonejärjestelmään ja asiakasohjelmat toisiin järjestelmiin. Asiakasohjelma ottaa yhteyden palvelimeen, vaihtaa tietoa palvelimen kanssa ja katkaisee yhteyden. (IBM 2021a.)

Socket-yhteydessä on tyypillinen tapahtumakulku. Yhteyspohjaisessa asiakasohjelmasta palvelimeen -mallissa palvelin odottaa yhteyspyyntöä asiakasohjelmalta. Tämän tehdäkseen palvelin valjastaa jonkin osoitteen omaan käyttöönsä, minkä avulla asiakasohjelma pystyy löytämään palvelimen. Tämän jälkeen palvelin jää odottamaan mahdollisia pyyntöjä asiakasohjelmalta. Datansiirtoprosessi aloitetaan, kun asiakasohjelma ottaa yhteyden palvelimeen. Palvelin suorittaa asiakasohjelman pyynnöt ja lähettää vastauksen takaisin asiakasohjelmalle. (IBM 2021a.)

Socket-sovellukset on yleensä kirjoitettu C- tai C++-ohjelmointikielillä käyttäen alkuperäistä Berkeley Software Distributionin määrittelemää socket-ohjelmointirajapintaa (IBM 2021b).

3.5 Java

Java on oliopohjainen ohjelmointikieli, mikä tarkoittaa käytännössä sitä, että toisin kuin jotkin vanhemmat ohjelmointikielet, Java keskittyy enemmän dataan kuin itse komentoihin. Datasta luodaan luokkia ja näissä luokissa dataa säilytetään muuttujissa. Näistä luokista voidaan luoda instansseja, joita käytetään itse komentojen suorittamiseen. (Hartman 2022.)

3.5.1 Ominaisuudet

Java on yksi helpoimmista ohjelmointikielistä oppia, se on alustariippumaton, se käyttää automaattista muistinhallintaa ja mahdollistaa monisäikeisen ohjelmoinnin. Ohjelman ajaminen voidaan siis jakaa moneen pienempään toimintoon, jotka ajetaan samanaikaisesti pääohjelman kanssa. (Hartman 2022.)

3.6 C#

C# on moderni, oliopohjainen ja tyyppiturvallinen ohjelmointikieli. C#-kieli mahdollistaa kehittäjät tekemään turvallisia ja vankkoja sovelluksia, jotka suoritetaan .Net-ohjelmistokomponenttikirjastoissa. C#-kielen juuret ovat C-ohjelmointikielessä, ja ohjelmointikieli on helposti lähestyttävä C-, C++- ja Java-ohjelmoijille. (Microsoft 2022.)

4 Ohjelmointi

4.1 Socket-ohjelma

Socket-ohjelma on ollut käytössä vanhoissa projekteissa Pesmel Oy:llä. Ominaisuuksien ohjelmoiminen tähän projektiin aloitettiin viestirakenteiden muutoksilla. Socket-ohjelma lukee SQL-tietokantaa ja hakee viestityypin mukaisen telegram-viestipohjan.

4.1.1 Telegram-viestirakenne

Socket-yhteydestä tulevaa raakaa dataa paloitellaan XML-muotoon käyttäen viestikenttien pituutta. Viestikenttien pituudet ovat asiakkaan pyytämät vakiot. Viestikentille ohjelmoidaan myös tyyppi-kenttä, joka määrittelee, miten dataa käsitellään. Jotkut viestin osat tallennetaan sellaisenaan tietokantaan ilman data-muunnosta, kuten ylätunnisteosuus viestistä. User-data muunnetaan EBCDIC-muodosta ASCII-muotoon yläjärjestelmästä saapuville viesteille ja ASCII-muodosta EBCDIC-muotoon yläjärjestelmään lähteville viesteille. Viestikentästä otetaan talteen SEQ2-kenttä ja muunnetaan se viestinumeroksi. Jos tietokannassa on jo samalta lähettäjältä viesti, joka sisältää saman viestinumeron, viestiä ei tallenneta tietokantaan, vaan se hylätään tuplamerkintänä.

```
<!-- [redacted] Test information (B / C -> sequencer)-->
<Telegram Id="[redacted]">
  <[redacted] type="hexshort" length="4"></[redacted]>
  <[redacted] type="ebcstring" length="180"></[redacted]>
  <[redacted] type ="string" length ="4"></[redacted]>
</Telegram>
```

Kuva 4. Telegram-viestiesimerkki

Viestit käsitellään eri tavalla riippuen siitä, tuleeko viesti yläjärjestelmältä vai lähetetäänkö se sinne. ACK-viestien rakenne ja käsittely on myös erilainen. Järjestelmä lähettää ACK-viestin välittömästi yläjärjestelmälle silloin, kun viesti saapuu ja se muodostetaan sen mukaan, oliko data kunnossa. Yläjärjestelmälle

lähetettävissä viesteissä viesti lähetetään maksimissan kolme kertaa, jos ACK-viesti ei saavu perille.

Viestiä luodessa ja sen eri vaiheiden aikana pidetään kirjaa prosessin onnistumisesta. Jos jossain vaiheessa viestin luomista tai tietokantaan tallennusta tapahtuu virhe, asetetaan virhenumero virittyneeseen tilaan. Tämän avulla luodaan NACK-viesti, jossa on oma kenttä merkitsemässä virhetilaa.

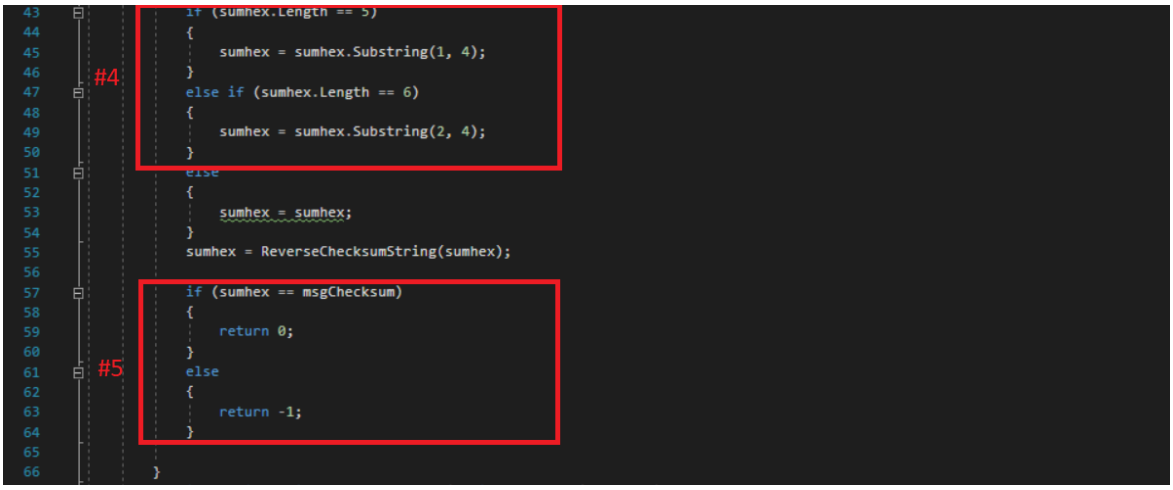
4.1.2 Tarkistussummaohjelman luominen

Asiakkaan C-ohjelmointikielellä toteutettu tarkistussummaohjelma päätettiin uudelleen koodata C#-ohjelmointikielelle, koska jatkokehityksen kannalta on parempi, että koko socket-ohjelma on ohjelmoitu samalla kielellä. Koodia on helpompi tarkastella ja muokata.

Tarkistussummalaskenta on C#-luokka, jota voidaan käyttää tarkistussumman tarkistukseen ja luomiseen.

```
11  public static int CheckChecksum(string receivedMessage)
12  {
13      string msg = receivedMessage.Substring(4, receivedMessage.Length - 8);
14  #1  string msgChecksum = receivedMessage.Substring(receivedMessage.Length - 4, 4);
15      string fullMessage = msg;
16
17      List<String> wordlist = new List<string>();
18      string word = string.Empty;
19      int j = 0;
20      string byte1 = string.Empty;
21      string byte2 = string.Empty;
22      for (int i = 0; i < fullMessage.Length / 4; i++)
23      {
24          word = fullMessage.Substring(j, 4);
25          byte1 = word.Substring(0, 2);
26          byte2 = word.Substring(2, 2);
27
28  #2      word = byte2 + byte1;
29
30          wordlist.Add(word);
31
32          j = j + 4;
33      }
34      long sum = 0;
35      int temp = 0;
36      foreach (string item in wordlist)
37      {
38  #3      temp = Convert.ToInt32(item, 16);
39          sum += temp;
40      }
41
42      string sumhex = sum.ToString("X");
```

Kuva 5. Tarkistussummaluokka 1-2




```
43          if (sumhex.Length == 5)
44          {
45              sumhex = sumhex.Substring(1, 4);
46          }
47          else if (sumhex.Length == 6)
48          {
49              sumhex = sumhex.Substring(2, 4);
50          }
51          else
52          {
53              sumhex = sumhex;
54          }
55          sumhex = ReverseChecksumString(sumhex);
56
57          if (sumhex == msgChecksum)
58          {
59              return 0;
60          }
61          else
62          {
63              return -1;
64          }
65
66      }
```

Kuva 6. Tarkistussummaluokka 2-2

1. Tarkastuksessa otetaan viestin alkuperäinen tarkistussumma talteen.

2. Uuden tarkistussumman luominen aloitetaan käyttäen rajattua osaa saapuvasta viestistä. Viesti paloitellaan ”sanoihin”, jotka koostuvat yhden tavun hex-numeroista. Sanat käännetään päinvastaiseen järjestykseen.

3. Sanat muunnetaan desimaaliseen muotoon ja lasketaan yhteen lopulliseen summaan.

4. Summa muunnetaan takaisin hex-muotoiseksi. Jos hex-summan pituus on yli 4, täytyy edestä poistaa kirjaimia niin monta että pituus on taas neljä. Tämä johtuu alkuperäisen C-ohjelman digit overflow -ominaisuudesta.

5. Seuraavaksi tehdään lopullinen tarkistussumman tarkistus, ohjelma palauttaa numeron 0, jos tarkistus onnistui, ja -1, jos tarkistussumma on väärä.

4.1.3 Tarkistussumman laskeminen käytännössä

Viestiä luodessa luodaan viestille ylätunniste, joka sisältää tietoa vietin lähettäjästä, vastaanottajasta, viestin pituudesta ja muutamista muista seikoista.

Viestin ylätunniste näyttää kuvan 7 mukaiselta.

Kuva 7. Header-esimerkki

Ylätunnisteeseen lisätään tämän jälkeen user-data, joka sisältää viestin oleellisen datan, eri viestityyppejä käytetään eri tarkoituksiin. User data on koodattu EBCDIC-merkistökoodausstandardilla.

Kun koko viesti on saatu koottua, voidaan aloittaa tarkistussumman laskenta. Ensin viesti paloitellaan kahden tavun ”sanoihin” ja sanat käännetään väärinpäin. Näistä tavuista lasketaan desimaalinen summa, joka lisätään kokonaissummaan.

Kun kokonaissumma on saatu laskettua headerin ja userdatan avulla, muunnetaan se taas hex-tyyppiseksi. Numeroylivuodon takia summan alusta tulee poistaa niin monta hex-merkkiä, että lopullinen tulos on neljä merkkiä pitkä. Lopuksi hex-arvo käännetään ympäri ja tässä on tarkistussumma.

4.1.4 Asiakkaalle lähetettävän viestin rakenne

Ks. Liite 4, Asiakkaalle lähetettävän viestin rakenne

4.1.5 Ajoitettu yhteyden tarkistus

Socket-yhteyden alkukantaisuuden vuoksi oli luotava ohjelmaan osuus, joka tarkastaa ajastettuna luotua yhteyttä. Ilman tarkastusta socket-ohjelma saattaa jäädä virittyneeseen tilaan ja lähettää viestejä, vaikka todellisuudessa yhteys on jo katkennut.

Toiminta

- Kun yhteys muodostetaan, luodaan uusi säie, joka tarkastelee yhteyttä. Säie aloittaa tarkastuksen.
- Seuraavaksi tarkistetaan, onko TCP-client vielä olemassa. Jos ei ole, yhteys suljetaan ja vapautetaan mahdollisuus uudelleen yhdistämiseen.

- Säie asetetaan nukkumistilaan viideksi sekunniksi. Jos tätä ei tehtäisi, yhteyttä tarkastettaisiin liian usein, mikä ei ole suotavaa.
- Tämän jälkeen tarkistetaan, onko verkko vielä saatavilla System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable( );-komennolla. Komento löytyy C#-verkkokirjastoista.
- Seuraavaksi aloitetaan ping-komento nykyisellä yhteydellä. Dataksi asetetaan yksi tavu, joka vastaa kirjainta ”1” ASCII-merkistökoodauksella, eli datan pituus on 1 tavu.
- Nyt tarkastetaan, menikö ping-komento läpi. Jos se meni, asetetaan muuttuja pingWentTrough virittyneeseen tilaan.
- Seuraavaksi tarkastetaan onko verkkoja saatavilla ja menikö ping-komento läpi. Jos se ei mennyt, yritetään vielä kaksi kertaa uudelleen ping-komentoa. Jos ping ei vieläkään mennyt lävitse, suljetaan yhteys ja vapautetaan mahdollisuus uudelleen yhdistämiseen.
- Viimeiseksi tehdään tarkastus, onko yhteys vielä sama kun aikaisemmin on todettu. Jos on, kirjoitetaan logiin, että yhteys on kunnossa.
- Säie aloittaa uuden tarkistuksen.

Kaikki tarkistukset on ympäröity try & catch -lohkoilla. Jos yritys eli try epäonnistuu, se tarkoittaa, että yhteys ei ole kunnossa ja suljetaan yhteys, suljetaan säie ja vapautetaan mahdollisuus uudelleen yhdistämiseen.

Järjestelmä toimi hyvin testeissä, mutta kaikista varmin ratkaisu on heartbeat-järjestelmä, joka vaatii molemmilta järjestelmiltä toimintaa. Heartbeat on järjestelmä, joka lähettää viestin, johon odotetaan vastausta. Jos vastaus ei saavu kohteeseen, yhteys katkaistaan ja muodostetaan uudelleen. (RabbitMQ, [Viitattu 10.05 2022].)

4.2 OPC UA, Databridge

Databridge on ohjelma, joka lukee ja kirjoittaa SQL-tietokantaan, sekä lukee ja kirjoittaa PLC:lle arvoja käyttäen OPCUA-kirjastoja. Kun socket-ohjelma kirjoittaa SQL-tietokantaan uusia merkintöjä, databridge-ohjelma havaitsee ne ja kirjoittaa arvot PLC:lle. Kun kirjoitus on valmis, databridge-ohjelma asettaa SQL-tietokannan arvon käsitellyksi.

Ohjelma myös hoitaa etikettien tulostukseen tarvittavan tiedonkäsittelyn, kun asiakkaalta tulee viestejä, jotka vastaavat uusia keloja linjastolla. Ne käsitellään uusiksi merkinnöiksi tietokantaan etikettitauluun, josta data haetaan tulostusvaiheessa kelan etiketille.

Ohjelma ajaa ajoitetusti työtehtäviä, jotka on jaoteltu erilaisille viestityypeille ja muutamalle erikoistapaukselle. Jos SQL-tietokannassa on käsittelemättömiä viestejä ja PLC on valmis käsittelemään viestin, ns. työtehtävä käsittelee viestin.

4.2.1 Ohjelman eri osuudet

Seuraavaksi esitellään ohjelman eri osuudet ja ohjelman tarvitsemat luokkatyypit. Ohjelma on jaoteltu erilaisiin työtehtäviin riippuen siitä, mikä työn tyyppi on kyseessä. Ohjelma sisältää myös oman luokan jokaiselle viestityypille.

Kirjoittajat:

- Kirjoittajat ovat työtyyppejä, jotka kirjoittavat uutta dataa PLC:lle

Lukijat:

- Lukijat ovat työtyyppejä, jotka lukevat dataa PLC:ltä, ja luovat uusia viestejä tietokantaan ja asettavat ne lähetysvalmiuteen.

Viestityypit:

- Jokaiselle viestille on oma työtehtävä, ja niille tehdään erilainen luku/kirjoitustoimenpide riippuen viestin rakenteesta. Esimerkiksi ”Kirjoittajat” lukevat SQL-tietokannasta yläjärjestelmän viestejä, kun taas ”Lukijat” lukevat PLC:tä ja kirjoittavat tietokantaan.

Etikettityö:

- Etikettityö kirjoittaa kovalevylle uuden parametritiedoston, jonka etiketintulostusohjelma havaitsee.

HMI-työ:

- HMI-työ kirjoittaa PLC:lle dataa. Tätä dataa näytetään HMI-paneeleissa silloin, kun HMI-paneelilta pyydetään jonkun kelan tietoja.

Heartbeat-työ:

- Asiakkaan pyynnöstä luotiin heartbeat-työ, joka lähettää viestin tietyn ajanjakson kuluttua. Tällä tavoin voidaan tarkistaa, että yhteys on vielä kunnossa PLC:lle.

Tagfactory:

- Tagfactory sisältää määrittelyt tagien nimille. Jos haluttu toiminto on PLC:ltä lukeminen, määritellään lista tagien nimiä, luetaan arvot ja luodaan tietokantamerkintä näiden arvojen perusteella.

- Jos taas haluttu toiminto on kirjoittaminen PLC:lle, määritellään lista tagien nimistä, haetaan tietokannasta viesti, paloitellaan viesti ja kirjoitetaan viestin eri palaset tagien arvoiksi.

OPC UA -kirjastot:

- Ohjelma sisältää OPC UA -kirjastot, jotka mahdollistavat PLC:lle kirjoittamisen ja PLC:ltä lukemisen.

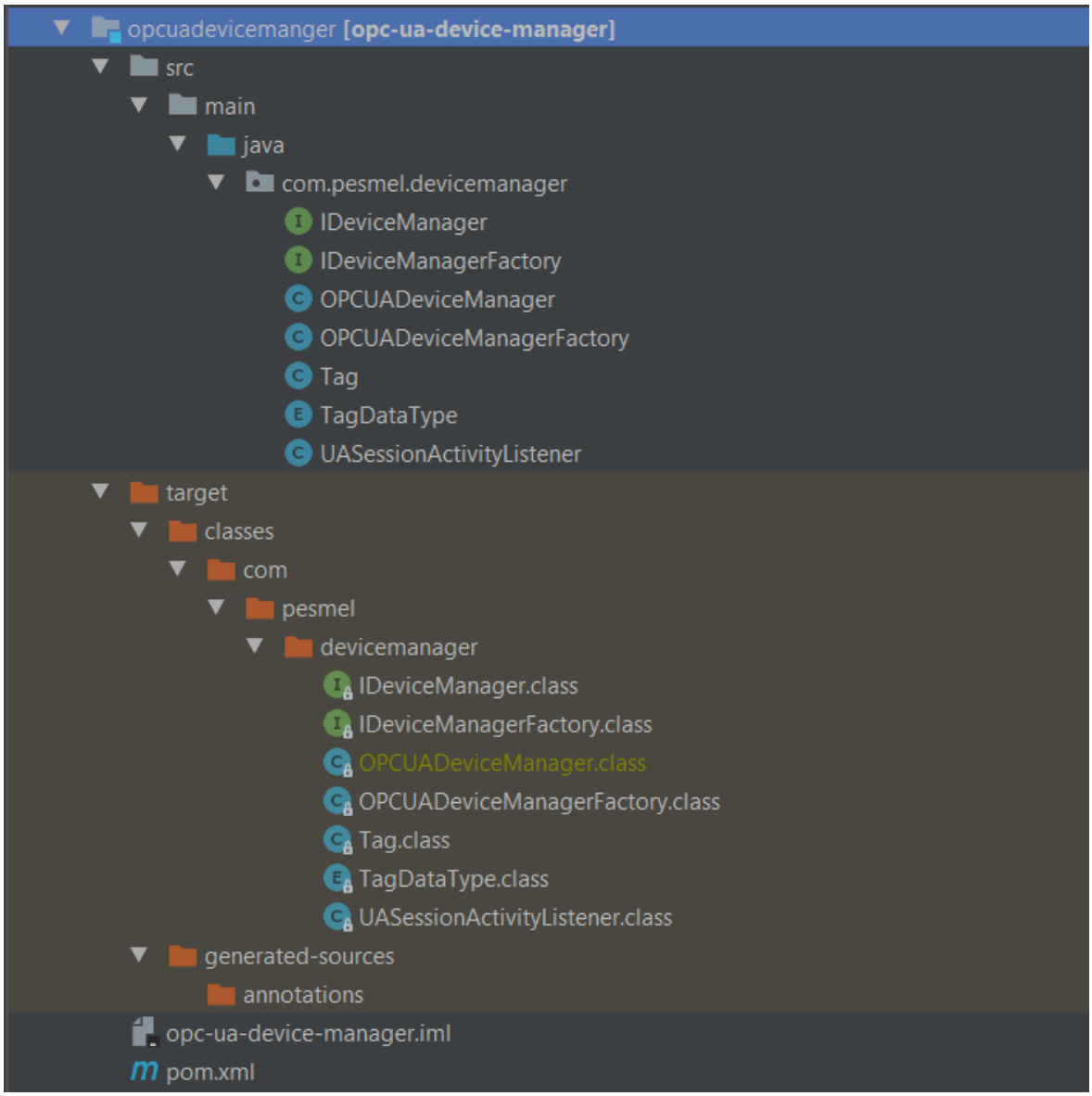
Hälytysviestit

- Yläjärjestelmä saattaa lähettää hälytysviestin esimerkiksi jos pakkauslinjalta lähtevän kelan data ei täsmää yläjärjestelmän datan kanssa. Tälläisessä tapauksessa järjestelmään muodostetaan hälytys ja virhetilanne täytyy kuitata linjaston HMI-paneelista.

Databridge, luokkien selitykset ja toimintaperiaatteet

Ks. Liite 5, Databridge, luokkien selitykset ja toimintaperiaatteet.

4.2.2 OPC UA -kirjastot



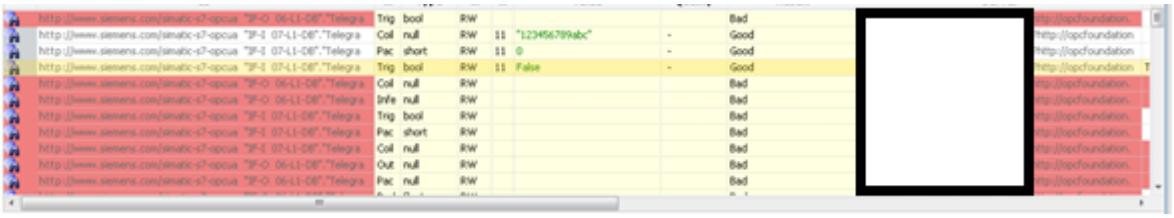
Kuva 8. OPC UA -Java-kirjastot

Kuvassa 8 on OPC UA:n Java-kirjastot, jotka mahdollistavat kommunikoinnin L1:n eli laitetason kanssa. OPC UA on vastuussa PLC:ltä lukemisesta ja PLC:lle kirjoittamisesta.

5 Testaus

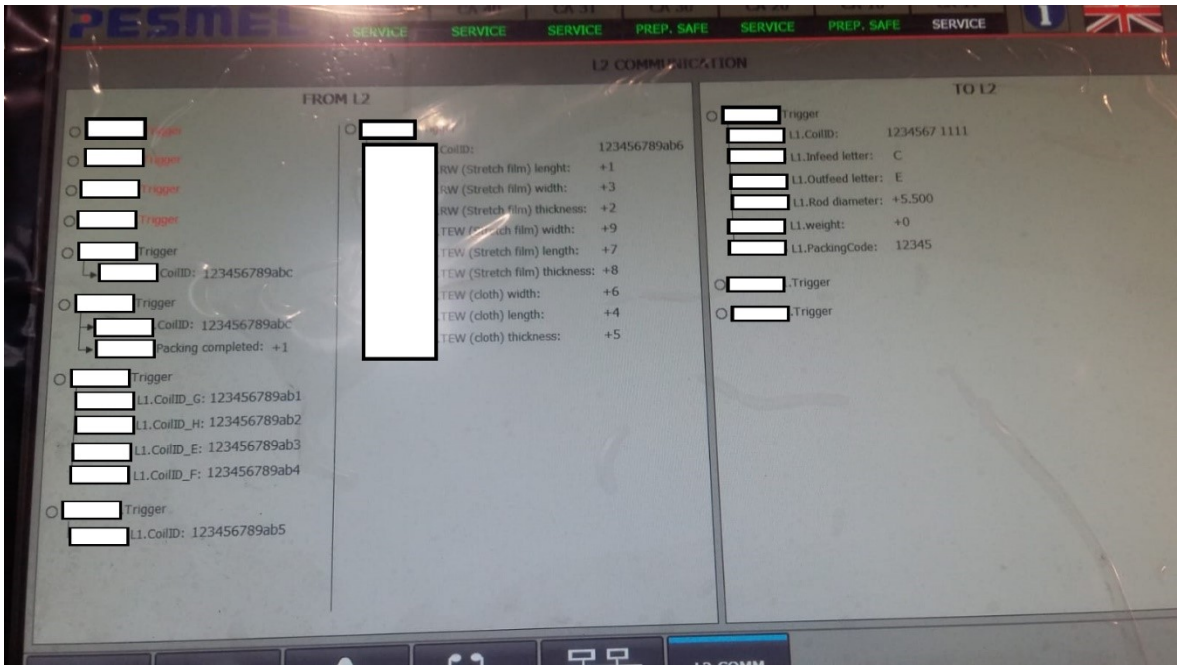
5.1 PLC-testit Virossa

PLC-yhteyttä testattiin Viron toimipisteelle 18.11.2018. Virossa on yksi Pesmelin toimipiste, jossa voidaan testata linjastoa ennen varsinaista käyttöönottoa. PLC-yhteys muodostettiin OPC UA:n kautta ja viestejä lisättiin onnistuneesti SQL-tietokantaan. IP-asetukset tuli uusia ja OPC-lisenssi tuli ladata PLC:lle. Testejä oli tehty jo Seinäjoen yksikössä ja tageja ei tarvinnut muuttaa.



Kuva 9. PLC:ltä luettuja tagien arvoja

Linjaston kokoamisvaiheessa aloitetaan viestien automaattisen lähetyksen testaus.



Kuva 10. Testidataa HMI-paneelilla testien aikana

Data lähetettiin onnistuneesti OPC UA:n kautta PLC:lle, myös viestien lukemista ja luomista testattiin onnistuneesti.

5.2 Protokolla- ja applikaatiotestaus Japanissa

Japanissa testattiin ohjelman toimivuutta erilaisissa virhetilanteissa. Testaus piti sisällään kaikki viestit, jotka liittyivät kelan pakkausprosessiin. Testaus kesti kaksi viikkoa ja piti sisällään protokolla- ja applikaatiotestit.

5.2.1 Protokollatestit

Protokollan testaus aloitettiin viestinnän testauksella. Pian ilmeni ongelmia, jotka liittyivät asiakkaan spesifikaatioihin. Ohjelman datan muunnostapaan ohjelmoitiin muutos, että datavirtaan ei kirjoiteta EBCDIC-hex-arvoja suoraan ASCII-muodossa, vaan datavirtaan kirjoitetaan tavuja, jotka vastaavat kaivattua EBCDIC-hex-arvoa.

Protokollatesteissä ilmeni myös, että ohjelma ei kunnolla ymmärtänyt koska yhteys on katkennut. Socket-yhteys on alkukantainen protokolla, tarvitaan ylimääräisiä tarkistuksia, että havaitaan yhteyden katkeaminen. Asiakkaan kanssa sovittiin mahdollisesta heartbeat-järjestelmästä. Paikan päällä ohjelmoitiin myös ping ja tarkastusjärjestelmä, jotta testit saataisiin menemään läpi.

5.2.2 Applikaatiotestit

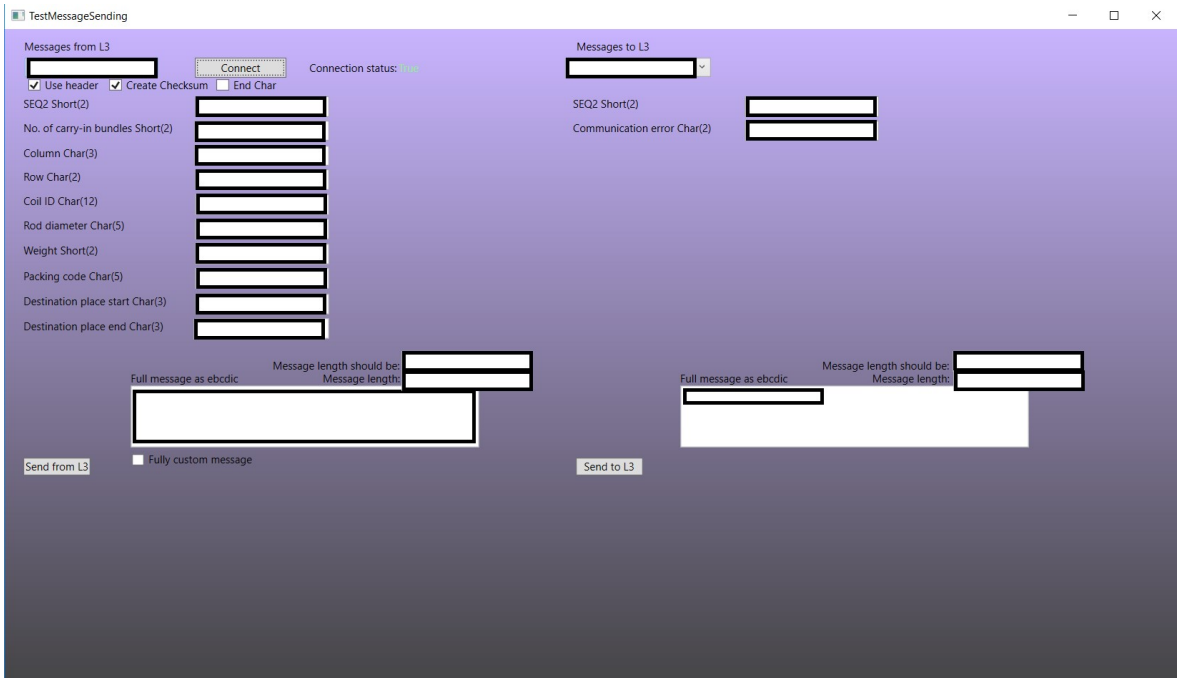
Applikaatiotestit aloitettiin testauksen toisella viikolla. Testien tarkoitus oli lähettää yläjärjestelmältä linjalle sen tarvitsemat viestit. L2-taso lähettää yläjärjestelmälle linjan saapumis-, pakkaus valmis- ja valmis poisvientiin -viestit. Testit sujuivat ilman että ohjelmaan tarvitsi tehdä muutoksia. Viestien kulku oli testattu aiemmin Virossa oikean PLC-järjestelmän kanssa.

5.3 Testiviestin lähetysohjelma

Testauksen helpottamiseksi koodattiin osaksi socket-ohjelmaa myös testisanomien lähetyspaneeli. Paneelista voi valita viestityypin ja kirjoittaa viestin sisältöön haluttua dataa. Tämä helpotti asiakkaan kanssa tehtäviä testauksia, sillä yläjärjestelmästä

tulevia viestejä voitiin jo etukäteen tarkistaa ennen varsinaisia automatisoituja viestinlähetystestejä.

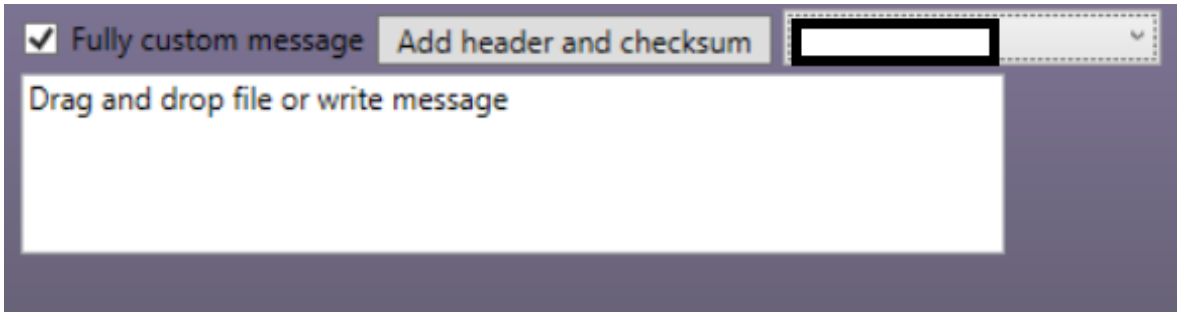
5.3.1 Käyttöliittymä



Kuva 11. Testiviestien lähetyksen käyttöliittymä

Viestien muodostaminen onnistuu käyttöliittymästä seuraavanlaisesti:

Ensin valitaan lähetettävän viestin tyyppi, sitten kirjoitetaan haluttu data alla oleviin kenttiin. Viesti muodostetaan oikean muotoiseksi ja käännetään EBCDIC-merkistökoodauksella.



Kuva 12. Mukautettu viestikenttä

Viestityypin valinnan jälkeen voidaan painiketta painamalla lisätä viestiin header ja tarkistussumma. Tämän jälkeen ’Send from L3’-nappia painamalla viesti lähetetään socket-yhteydellä socket-ohjelmalle.

Asiakkaan testisanomien lähettämisen helpottamiseksi ohjelmaan lisättiin myös mahdollisuus pudottaa tekstitiedostoja Custom-message-tekstilaatikon päälle. Ohjelma lukee binäärilukijalla tekstitiedossa olevien merkkien hex-arvot ja muuntaa viestin EBCDIC-muotoon.

6 Tulokset

Työn tuloksena kehitettiin kaksi ohjelmaa ja niihin liitännäinen tietokantayhteys. Yläjärjestelmälle viestejä lähettävä ja viestejä vastaanottava socket-ohjelma ja ohjelmoitavalle logiikalle dataa kirjoittava ja dataa lukeva ohjelma databridge.

Socket-ohjelma sisälsi ominaisuudet jotka mahdollistivat EBCDIC-merkistökoodauskielen purkamisen ja viestien datan paloittelemisen ja tietokantaan myöhempää prosessointia varten tallentamisen. Ohjelma oli myös vastuussa tietokannasta ASCII-muotoisten merkintöjen lukemista, viestien muodostamisesta ja EBCDIC-muunnoksen tekemisestä ennen lopullisen viestin lähettämistä yläjärjestelmälle. Ohjelma sisältää kaikkien käytössä olevien viestin viestimäärittelyt ja on vastuussa yläjärjestelmän kanssa kommunikoinnista.

Databridge-ohjelma sisälsi OPC UA -kirjastot, jotka mahdollistivat ohjelmoitavan logiikan kanssa keskustelun käyttäen node-osoitteita. Ohjelma sisältää listaukset logiikan node-osoitteista. Ohjelma on jatkuvassa yhteydessä laitetasoon, tarkastellen onko PLC:n node-osoitteissa uutta dataa, josta voidaan muodostaa yläjärjestelmälle lähetettäviä viestejä. Ohjelma on myös jatkuvassa yhteydessä tietokantaan, tarkastellen onko siellä käsittelemättömiä viestejä joiden data pitäisi kirjoittaa PLC:lle.

Tietokannassa on kaksi taulua, yksi viestejä varten ja yksi kelojen dataa varten. Kelojen dataa säilytettiin etikettien tulostukseen liittyvään prosessiin ja HMI-paneeleilla datan näyttämiseen. Tietokantantana käytettiin Mircosoft SQL Server 2018 -ohjelmistoa.

7 Yhteenveto ja pohdinta

Tämän opinnäytetyön tavoite oli suunnitella ja kehittää pakkauslinjaston kommunikointijärjestelmä Pesmel Oy:n tarpeisiin. Lähteinä käytettiin pääosin eri teknologioiden verkkosivuja ja joitain verkkojulkaisuja.

Kommunikointijärjestelmä toimi käytössä hyvin. Asiakas oli ohjelmistoon tyytyväinen, vaikka testausvaiheessa ilmeni joitain ongelmia varsinkin protokollan kanssa.

Opinnäytetyöprosessi sujui melko hyvin vaikka asiakkaan kanssa kommunikoinnin olisi voinut hoitaa paremmin ja nopeammalla aikataululla. Myös asiakkaan vaatimat ominaisuudet ja käytännöt olivat melko vanhanaikaisia, tästä johtuen oli opiskeltava vanhoja teknologioita, kuten EBCDIC merkistökoodausta ja IBM- suurtietokoneiden toimintaa.

Testiviestin lähetysohjelma vähensi merkittävästi työmäärää. Applikaatio- ja protokolla testaamiselle oli varattu vain rajattu määrä aikaa. Koska testiviestien lähetysohjelman ansiosta applikaatiotestit menivät ongelmitta läpi, aikaa jäi enemmän protokollaongelmien korjaamiseksi.

Opinnäytetyötä voidaan käyttää ohjenuorana vastaavanlaisen kommunikointijärjestelmän suunnitelutyötä tehdessä. Minkälaisia teknologioita voidaan käyttää ja minkälaisia ongelmia saattaa ilmaantua esimerkiksi socket-yhteyksien kanssa. Koska tässä työssä keskitytään pelkkään kommunikointijärjestelmään eikä itse laitteistoon, olisi mahdollista käyttää samanlaista arkkitehtuuria minkä tahansa kommunikointijärjestelmän kanssa, jossa on käytössä socket-yhteys ja PLC-logiikka.

LÄHTEET

Chapple, M. 2019. The Fundamentals of SQL. About SQL. [Verkkojulkaisu]. ThoughtCo. [Viitattu 19.11.2019]. Saatavilla https://www.lifewire.com/sql-fundamentals-1019780

Hartman, J. 2022. What is Java? Definition, Meaning & Features of Java Platforms. [Verkkojulkaisu]. Guru99. [Viitattu 10.05.2022]. Saatavilla: https://www.guru99.com/java-platform.html#1

IBM. 2021a. How sockets work. [Verkkosivu]. IBM. [Viitattu 25.04.2022]. Saatavilla: https://www.ibm.com/docs/en/i/7.1?topic=programming-how-sockets-work

IBM. 2021b. SQL procedures. [Verkkosivu]. IBM. [Viitattu 25.04.2022]. Saatavilla: https://www.ibm.com/docs/en/ias?topic=routines-sql-procedures

IBM. 2021c. What is a TCP/IP Socket Connection. [Verkkosivu]. IBM. [Viitattu 25.04.2022]. Saatavilla: https://www.ibm.com/docs/en/zvse/6.2?topic=SSB27H_6.2.0/fa2ti_what_is_socket_connection.html

IBM. Ei päiväystä. Application programming on z/OS. [Verkkojulkaisu]. IBM. [Viitattu 10.05.2022]. Saatavilla: https://www.ibm.com/docs/en/zos-basic-skills?topic=mainframe-ebcdic-character-set

Microsoft. 2021. SQL Server Agent. [Verkkosivu]. Microsoft. [Viitattu 25.04.2022]. Saatavilla: https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?redirectedfrom=MSDN&view=sql-server-ver15

Microsoft. 2022. A tour of the C# language. [Verkkosivu]. Microsoft. [Viitattu 26.04.2022]. Saatavilla: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Nikevich 2011. Keypunch. [Kuva]. Wikipedia. [Viitattu 19.11.2019]. Saatavilla: https://en.wikipedia.org/wiki/File:Blue-punch-card-front-horiz_top-char-contrast-stretched.png#file

OPC foundation. 2019. UA Companion Specifications. [Verkkosivu]. OPC foundation. [Viitattu 19.11.2019]. Saatavilla: https://opcfoundation.org/about/opc-technologies/opc-ua/ua-companion-specifications/

Pediaa. 2022. Difference Between ASCII and EBCDIC. [Verkkosivu]. Pediaa. [Vitattu 18.04.2022]. Saatavilla: https://pediaa.com/difference-between-ascii-and-ebcdic/

RabbitMQ. Ei päiväystä. Detecting Dead TCP Connections with Heartbeats and TCP Keepalives. [Verkkojulkaisu]. RabbitMQ. [Viitattu 10.05.2022]. Saatavilla: https://www.rabbitmq.com/heartbeats.html

LIITTEET

Liite 1. Socket ohjelman käyttöliittymä selitettynä

Liite 2. Asiakkaan lähettämät viestit ja niiden tyyppitunnisteet

Liite 3. Asiakkaalle lähetettävät viestit ja niiden tyyppitunnisteet

Liite 4. Asiakkaalle lähetettävän viestin rakenne

Liite 5. Databridge, luokkien selitykset ja toimintaperiaattee