jamk

Tehdasautomaationlinjaston digitaalinen mallinnus viestijonotekniikalla

Markku Puura

Opinnäytetyö, AMK
Toukokuu 2023
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Puura, Markku

Tehdasautomaatiolinjaston digitaalinen mallinnus viestijonotekniikalla

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2023, 43 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Tutkimuksellisessa kehittämistyössä oli tavoitteen kehittää digitaalisen tehdasautomaatiolinjaston prototyyppi, joka hyödyntäisi viestijonotekniikkaa osana automaatiolinjaston mallinnusta. Viestijonotekniikkaa hyödyntämällä oli tarkoitus parantaa suorituskykyä mallinnuksessa, sekä mahdollisuutta hajauttaa automaatiolinjaston digitaaliset linjastolaite-sovellukset eri palvelimille. Digitaalisen tehdasautomaatiolinjaston prototyypin oli tarkoitus toimia osana jalostajan tuotannon infrastruktuuria elintarvikeketjun kyberturvallisuus hankkeessa. Hankkeesta järjestettiin myös kyberturvallisuus pilottiharjoitus, jossa prototyyppi olisi käytössä.

Kehittämistyössä käytettiin TypeScript-ohjelmointikieltä, sekä Vue.js-ohjelmistokehystä ja Node.js-ohjelmointiympäristöä. Palvelinsovelluksen kehittämisessä käytettiin TypeScript ohjelmointikieltä, sekä Node.js-ohjelmointiympäristöä. Käyttöliittymän toteutuksessa käytettiin Vue.js-ohjelmistokehystä sekä TypeScript-ohjelmointikieltä. Käytetty viestijonotekniikka oli Bull.js-kirjasto, joka hoiti viestien lähettämisen ja vastaanottamisen, sekä ylläpiti itsenäisesti tietoliikenneyhteyksiä taustalla toimivan Redis-palvelimen kanssa.

Lopputuloksena saatiin kehitettyä viestijonotekniikkaa hyödyntävä digitaalinen tehdasautomaatiolinjaston prototyyppi, joka oli dynaamisesti konfiguroitavissa, sekä horisontaalisesti skaalautuvissa. Viestijonotekniikan käyttö toimi erinomaisesti linjastolaitteiden välisien kuljettimien sekä putkistojen mallinnuksessa. Prototyyppi sisälsi myös käyttöliittymän, josta voitiin seurata eri materiaalivirtojen, sekä tuotteiden liikkumista automaatiolinjastolla. Tulevaisuudessa prototyypin pohjalta pystytään kehittämään vielä tarkempi sekä laaja-alaisempi digitaalinen tehdasautomaatiolinjasto.

Avainsanat (asiasanat)

Node.js, Vue.js, TypeScript, Redis, Bull.js, Viestijonotekniikka, Automaatiolinjasto

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Puura, Markku

Digital modeling of automated factory production line with message queue technologies

Jyväskylä: JAMK University of Applied Sciences, May 2023, 43 pages.

Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

In research and development work, the goal was to develop a prototype of a digital factory automation line that would utilize message queue technology as part of the automation line modeling. By using message queue technology, the aim was to improve performance in modeling, as well as the possibility to distribute the automation line's digital line device applications to different servers. The prototype of the digital factory automation line was intended to be part of the processor's production infrastructure in the food supply chain cybersecurity project. A cybersecurity pilot exercise was also held for the project, in which the prototype would be used.

Node.js and Vue.js software frameworks, as well as TypeScript programming language, were used in the development work. Node.js programming environment and TypeScript were used on the server side. Vue.js and TypeScript were used on the user interface side. The message queue technology used was the Bull-Node.js library, which handled message sending and receiving, as well as independently maintained network connections with the Redis server running in the background.

The end result was a prototype of a digital factory automation line that utilized message queue technology, was dynamically configurable and horizontally scalable. The use of message queue technology worked exceptionally well in modeling of conveyors and pipelines between the line devices. The prototype also included a user interface that allowed for tracking the movement of different material flows and products on the automation line. In the future, based on the prototype, it will be possible to develop an even more precise and comprehensive digital factory automation line.

Keywords/tags (subjects)

Node.js, Vue.js TypeScript, Redis, Bull.js, Message queue, Automated production line

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet

API	Application Programming Interface
MES	Manufacturing Execution System
HMI	Human-Machine Interface
JSON	JavaScript Object Notation
JWT	JSON Web Token
ORM	Object-relational mapping
PLC	Programmable Logic Controller
REST	REpresentional State Transfer
RGCE	Realistic Global Cyber Environment
SCADA	Supervisory Control and Data Acquisition

1 Johdanto

1.1 Kyberturvallisuusriskit elintarviketeollisuudessa

Kyberturvallisuuden ylläpitäminen ja resilienssin kasvattaminen vaatii jatkuvaa työtä, erityisesti kriittisessä infrastruktuurissa toimivien yritysten parissa. Suomessa tarve kyberturvallisuuden kasvattamiseksi on noussut huimasti vuoden 2022 aikana ja tarvittaisiinkin paljon lisää säännöllisiä kyberturvallisuusharjoituksia. Kyberturvallisuusharjoituksissa esiin nousseet kehityskohteet ja opit viedään käytäntöön yrityksissä ja näin saadaan lisättyä resilienssiä haitalliseen kybervaikuttamiseen. Vuoden 2022 kyberturvallisuusselvityksen perusteella myös elintarviketeollisuuden alalla on tunnistettu toimialaan liittyviä kyberriskien heikkouksia. Yhtenä heikkoutena esille nousi kyberriskien hallinnan prosessien puutteet, myös lokien hyödyntämisessä tilannekuvan kannalta oli puutteita. (Kyberturvallisuus vaatii jatkuvaa työtä. 2022; Toimialojen kyberkypsyyden selvitys. 2022)

Elintarviketeollisuudessa käytössä olevien automaatiolaitteiden tietoliikenneyhteyksien siirtyminen suljetuista sisäverkoista avoimeen internettiin mm. lisääntyneen etäkäytön takia, nostaa haitallisen kybervaikutuksen riskiä. Myös tiedossa jo olevien automaatiolaitteiden haavoittuvuuksien hyödyntäminen helpottaa kyberhyökkäyksien toteutuksia. (Salvador, L. Dai, N. & Zoltán, R. 2023; Urooj, B. Ullah, U. Shah, M. Sikandar, H & Stanikzai A. 2022)

Jyväskylän ammattikorkeakoulun järjestämässä elintarvikeketjun kyberturvallisuus hankkeessa selvitettiin toimintatapoja sekä malleja kyberriskien hallintaan elintarvikealalla työskentelevien yrityksien käytettäväksi. Tässä opinnäytetyössä keskityttiin yhteen elintarvikeketjun kyberturvallisuus hankkeen osa-alueen suunnitteluun ja toteuttamiseen.

1.2 Toimeksiantaja

Opinnäytetyön toimeksiantajana toimi Suomen johtava riippumaton kyberturvallisuuden tutkimus-, kehitys- ja koulutuskeskus JYVSECTEC – Jyväskylä Security Technology. JYVSECTEC on osa Jyväskylän ammattikorkeakoulun IT-instituuttia. JYVSECTEC oli mukana elintarvikeketjun kyberturvallisuus hankkeen kehitystoiminnassa, sekä myös toteutti elintarvikeketjun kyberturvallisuus

hankkeen pilottiharjoituksen. Hankkeen kokonaisuudesta vastasi Jyväskylän ammattikorkeakoulun tutkimus, kehitys-, ja innovaatiotoiminta. (Elintarvikeketjun kyberturvallisuus. 2021)

1.3 Tavoite

Tämän opinnäytetyön tavoite oli suunnitella sekä toteuttaa elintarvikeketjun kyberturvallisuus hankkeelle digitaalinen tehdasautomaatiolinjaston prototyyppi, sekä samalla selvittää viestijonotekniikan soveltuvuutta automaatiolinjaston toiminnan mallintamiseen. Toteutettava prototyyppi tultiin myös liittämään osaksi digitaalisen jalostajan toiminnanohjausjärjestelmiä RGCE-ympäristössä. Prototyypillä piti myös pystyä tuottamaan RGCE:ssä järjestettävän pilottiharjoituksen aikana muun muassa digitaalisia elintarvikkeita, muiden elintarvikeketjun kyberturvallisuus hankkeen infrastruktuurissa olevien palvelujen käytettäväksi.

RGCE-ympäristö tarjosi realistisen vastineen oikealle internetille, mahdollistaen samalla turvallisen toimintaympäristön harjoituksessa toteutetuille kybervaikutusoperaatioille, koska ympäristö oli täysin eristetty internetistä, sekä muista ulkomaailman tietoverkoista. (Realistic Global Cyber Environment - Overview. 2022.)

1.4 Tutkimuskysymys ja rajaus

Tutkimuskysymys johdateltiin työssä käytetyn viestijonotekniikan soveltuvuudesta automaatiolinjaston digitaalisessa mallinnuksessa. Tutkimuskysymykseksi muodostui, onko mahdollista mallintaa tehdasautomaatiolinjaston toimintaa käyttäen viestijonotekniikkaa?

Tutkimusmenetelmä tässä työssä on tutkimuksellinen kehittämistyö. Kyseinen tutkimusmenetelmä valittiin tähän työhön, koska työn lopputuloksena syntyy konkreettinen prototyyppi, jonka toiminta tulee vastaamaan tutkimuskysymyksessä asetettuun ongelmaan. (Toikko, T. & Rantanen, T. 2009.)

Opinnäytetyössä suunnittelu- sekä toteutuskohteet rajattiin sisältämään vain linjastolaite-sovellus ja niitä ohjaava ohjainyksikkö-sovellus. Rajaus tehtiin, koska työn lopputuloksen kannalta oli tärkeää keskittyä vain itse digitaalisen tehdasautomaatiolinjaston toiminnan mallintamiseen sekä toteuttamiseen.

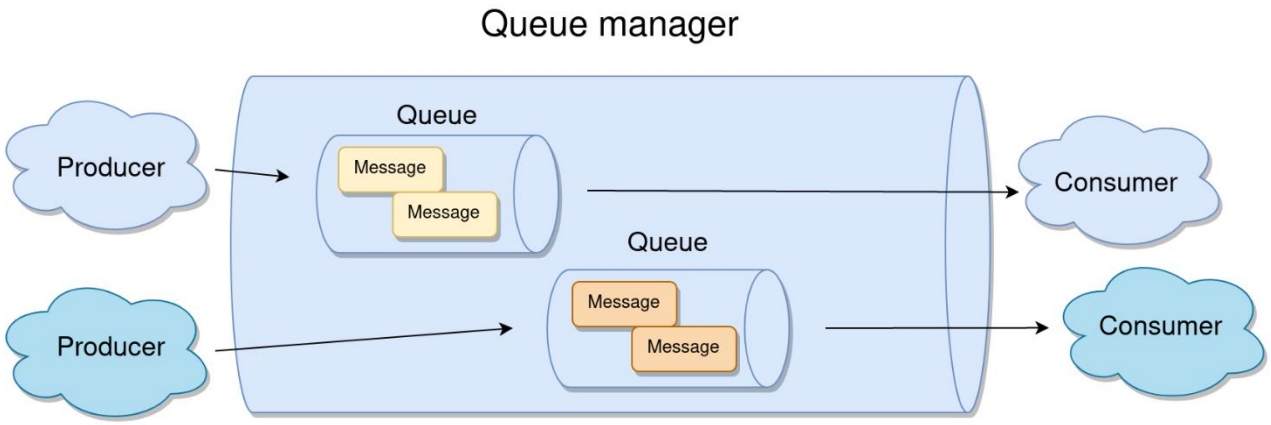
2 Teknologiat

2.1 Viestijonot

Viestijonotekniikkaa on käytetty jo pitkään tietojenkäsittelyssä ja se onkin yksi yleisimmistä tavoista lähettää ja vastaanottaa dataa eri palvelujen tai prosessien välillä. Erillinen palvelin tai ryhmä palvelimia toimii viestijonokeskuksena, joka vastaanottaa ja lähettää viestejä eteenpäin oman rajapintansa kautta. Itse jonot voidaan mieltää yksilöllisesti nimettyinä data puskureina, jotka säilyttävät viestin ja sen sisältämän datan. Jonoille määritellään tietyt kuuntelijat prosessit tai palvelut, jotka saavat tiedon viestin saapumisesta sekä itse viestin sisältämän datan. Kuuntelijat prosessoivat viestin sisältämän datan ja kuittaavat sen vastaanotetuksi viestijonopalvelimelle, jolloin viesti voidaan poistaa jonosta. Tämä prosessi mahdollistaa täysin asynkronisen toiminnan, mikä on tärkeä ominaisuus, kun isoja sekä hajautettuja järjestelmiä liitetään toimimaan yhdessä. Muita hyötyjä viestijonotekniikan käytössä on käyttöjärjestelmä riippumattomuus, skaalautuvuus, datan säilyvyys mahdollisten verkkokatkoksien tai viiveiden takia. (Introduction to message queuing. N.d; What’s a Message Queue. 2022.)

Digitaalista tehdasautomaatiolinjastoa mallintaessa viestijonotekniikan toimintatapa, sekä sen asynkroninen viestinvälitys ovat varsin tärkeitä ominaisuuksia mallinnuksen kannalta. Mallintamisessa lähdettiin liikkeelle ajatuksesta, että yksittäiset mallinnettavat digitaalisen tehdasautomaatio-linjaston laitteet kuuntelevat omaa viestijonoaan. Omaan jonoonsa saapuvat viestit laitteet prosessoivat itsenäisesti. Prosessoinnin mahdollisesti kestäessä pidempään viestijonoon kertyvät viestit ilmentävät oikealla tehdasautomaatio-linjastolla kerääntyviä tuotteita. Onnistuneen prosessoinnin jälkeen digitaaliset tehdasautomaatiolinjaston laitteet lähettävät viestin sekä prosessoidun datan eteenpäin seuraavalle ennalta määritetyn linjastonlaitteen viestijonoon ja viestin prosessointi jatkuu uudella linjastolaitteella. Näin saadaan toteutettua prosessointiketju, mikä mallintaa oikean tehdasautomaatiolinjaston toimintaa.

Kuviossa 1 on esitetty pelkistetty esimerkki viestijonotekniikan sekä viestijonojen toiminnasta. Vasemmalla kuviossa on viestintuottajia, jotka tuottavat viestejä viestijonoihin. Keskellä kuviossa on viestijonopalvelin, joka hallinnoi eri viestijonoja, sekä vastaanottaa ja lähettää viestejä eteenpäin. Oikealla kuviossa on viestin kuluttajat, jotka vastaanottavat viestit tietyiltä viestijonoilta.



Kuvio 1 Viestijonon toimintamalli

2.2 Käytetyt teknologiat ja kirjastot

2.2.1 TypeScript

TypeScript on avoimen lähdekoodin ohjelmointikieli, joka on kehitetty Microsoftin toimesta. Syntaksiltaan se on samalainen kuin JavaScript, mutta se sisältää tiettyjä ominaisuuksia, joilla koodin kirjoittamisesta tulee tehokkaampaa sekä virhealttiutta vähentävää. TypeScript on vahvasti tyypitetty ohjelmointikieli ja se on suosittu valinta modernissa web-sovelluskehityksessä, sekä palvelinpuolen sovelluskehityksessä. TypeScript koodi on myös helpommin ymmärrettävää ja ylläpidettävää tyypityksen ansiosta. (Why You Should Use Typescript for Your Next Project. 2022.)

2.2.2 Node.js

Node.js on palvelimelle tarkoitettu avoimen lähdekoodin JavaScript-ohjelmointiympäristö, jonka toiminta perustuu Chrome V8-JavaScript moottoriin. Node.js onkin suosittu valinta modernissa web-kehityksessä, koska sitä käyttämällä kehittäjät pystyvät käyttämään samoja ohjelmointitaitoa selain- ja palvelinpuolella, mikä nopeuttaa ja helpottaa kehittämistä. Node.js:llä pystyy myös toteuttamaan palvelimen komento rivillä toimivia työkaluohjelmistoja. (Introduction to Node.js. N.d.)

2.2.3 Vue.js

Vue.js on avoimen lähdekoodiin perustuva JavaScript-ohjelmistokehys. Se on tarkoitettu modernien yksisivuisten web-käyttöliittymien toteuttamiseen. Vue.js:llä kehittäessä käytetään komponenttipohjaista ohjelmointia. Yksittäinen komponenttitiedosto koostuu JavaScript, HTML ja CSS-koodista. Vue.js ominaisuuksiin kuuluu automaattinen JavaScript-koodin tilan ylläpitäminen ja seuraaminen. Tilassa tapahtuvat muutokset päivitetään tehokkaasti selaimen esitys- ja käyttöliittymäohjelmointirajapintaan. (Introduction - Vue.js. N.d.)

2.2.4 Hapi.js

Hapi.js on avoimen lähdekoodin JavaScript-ohjelmistokehys ja sen ominaisuuksiin kuuluu mm. HTTP-pyyntöjen käsittely, reititykset ja autentikointi. Hapi.js on tarkoitettu API-rajapintojen kehittämiseen ja se on yhteensopiva Node.js kanssa. Hapi.js tarjoaakin kehittäjille valmiiksi valittuja turvallisia ratkaisuja yleisimpiin web-sovelluksiin liittyviin haasteisiin. (Introduction to hapi.js Framework. 2020.)

2.2.5 Bull.js

Bull.js on Redis-pohjainen avoimen lähdekoodin viestijono JavaScript-kirjasto. Se tarjoaa kevyen ja helpon tavan luoda sekä hallita viestijonoja. Viestijonoja voidaan käyttää asynkronisten tehtävien suorittamiseen sovelluksien taustalla. Bull.js mahdollistaa toistuvien tehtävien, viiveellä suoritettavien ja priorisoitujen tehtäviä käytön. Bull.js käyttö auttaa sovelluksia skaalautumaan paremmin ja mahdollistaa monimutkaisten tehtävien tehokkaan hallinnan ja suorittamisen.(Bull. 2021.)

2.2.6 Redis

Redis on avoimen lähdekoodin in-memory-tietokanta, sitä käytetäänkin usein nopeasti muuttuvan datan käsittely tarpeissa. Se on suunniteltu olemaan nopea ja skaalautuva, mikä tekeekin siitä hyvä vaihtoehdon, kun nopeus ja suorituskyky ovat kriittisiä tekijöitä. Redis:in tarjoamia ominaisuuksiin kuuluu mm. transaktiot, julkaisu/tilaus -viestintä ja sille löytyykin valmiiksi moneen eri ohjelmointikieleen tarvittavat kirjastot. (Introduction to Redis. N.d.)

2.2.7 Docker

Docker on avoimen lähdekoodin ohjelma, joka tarjoaa kevyen tavan pakata, jakaa ja suorittaa sovelluksia konttien avulla. Docker-kontti on kuin eräänlainen kevyt virtuaalikone, joka on eristetty taustalla olevasta käyttöjärjestelmästä. Se sisältää myös kaikki tarvittavat riippuvuudet ja konfiguraatiot kontin sisällä olevalle sovellukselle. Docker tarjoaa myös kaikki työkalut konttien luomiseen, hallintaan ja orkestrointiin, helpottaen näin sovelluksen kehittämistä ja julkaisua. Docker-konttien käyttö on yksi suosituimmista tavoista hallinnoida sovelluksia palvelimilla. (What is Docker?. N.d.)

2.2.8 InfluxDB

InfluxDB on aikasarjatietokanta, joka on suunniteltu keräämään, käsittelemään ja myös visualisoimaan suuria määriä aikasarjadataa. InfluxDB käyttö hyödyttää erityisesti sovelluksia, jotka kerätään suuria määriä dataa mm. mittareista, antureista tai lokeista ja kyseisillä sovelluksilla on tarve visualisoida kerättyä dataa myöhemmin. InfluxDB käyttää SQL-pohjaista kyselykieltä, joka mahdollistaa monimutkaisten kyselyjen suorittamisen. InfluxDB tarjoaman HTTP-rajapinnan kautta päästää lukemaan ja kirjoittamaan dataa eri sovelluksilla. (What is InfluxDB. 2023.)

3 Suunnittelu

3.1 Toimintaympäristö

Toimintaympäristö toteutettavalle digitaalisen tehdasautomaatiolinjaston prototyypille oli RGCE. RGCE on JAMK:in ja JYVSECTEC:in kehittämä ainutlaatuinen kyberharjoitusympäristö, jolla mallinnetaan realistisesti globaalin digitaalisen yhteiskunnan tarvitsemia palveluita sekä organisaatioita. RGCE:ssä on järjestetty muuan muassa kansallisia kyberturvallisuusharjoituksia eli KYHA-harjoituksia jo vuodesta 2013. Kyberharjoituksien aikana RGCE-ympäristössä on mahdollista harjoitella toimintatapoja, erityyppisten hyökkäysvektoreiden varalta. Palvelinestohyökkäykset, tietojenkalasteluyritykset sekä kiristyshaittaohjelmat ovat vain muutamia mahdollisia esimerkkejä, mitä RGCE ympäristössä pystytään toteuttamaan. Koska kaikki tämä toiminta tapahtuu täysin suljetussa verkossa, harjoituksen aikaisista kyberhyökkäyksistä ei ole vaaraa internetin puolella oleville palveluille. (Realistic Global Cyber Environment - Technical details. 2022.; Tietoturvatunnustukset 2022.).

RGCE:ssä tietoliikenneyhteyksien toiminta vastaa kuitenkin oikean internetin tietoliikenneyhteyksien toimintaa. Ohjainyksikkö-sovellus ja kaikki linjastonlaite-sovellukset saivat omat yksilölliset verkko-osoitteensa käytettäväkseen ja sovelluksien välinen tietoliikenne kommunikointi tapahtui sovelluksien omien API-rajapintojen kautta. Ohjainyksikkö-sovellus, sekä linjastolaite-sovellukset kontitettiin omiksi itsenäisiksi konteiksi Docker-kontitusteknologiaa käyttämällä. Kontitusteknologialla sovelluksesta, sekä sen kaikista riippuvuuksista koostetaan oma itsenäinen paketti, joka voidaan suorittaa RGCE:n palvelimilla virtuaalikoneissa.

3.2 Yleiset vaatimukset sovelluksille

Yleisiä vaatimuksia kehitettäville sovelluksille määritteli tieto tulevasta toimintaympäristöstä RGCE:stä. RGCE:ssä toimivien digitaalisien tehdasautomaatiolinjastojen toiminta tuli olla itsenäistä, ilman erillistä käyttäjien vaatimaa toimintaa. Käyttäjille tuli kuitenkin olla tarvittaessa mahdollisuus vaikuttaa yksittäisien tuotantolinjastojen tiettyihin toimintoihin, esimerkiksi tuotantolinjaston toiminta piti pystyä pysäyttämään sekä palauttamaan käyttäjän toimesta.

Linjastonlaitteiden sekä ohjainyksikön sovelluksien tuli pysyä toiminnassa mahdollisten verkkoliikenne katkoksien tai viiveiden aikana. Sovelluksien piti myös pystyä jatkamaan toimintaansa uudelleen käynnistyksen jälkeen, sekä myös tuotantolinjastojen toiminta tuli jatkua tämä jälkeen itsenäisesti. Linjastonlaite-sovelluksien ohjaus tuli tapahtua keskitetysti ohjainyksikkö-sovelluksen kautta. Linjastolaite-sovelluksien toiminnan aikaista statistiikkaa piti myös voida seurata, sekä taltioida keskitettyyn lokiin. Loki järjestelmänä tuli olla erillinen palvelin, joka toteutettiin InfluxDB-teknologiaa käyttäen.

3.3 Ohjainyksikkö-sovelluksen suunnittelu

Ohjainyksikkö-sovelluksen suunnittelun alussa selvitettiin oikeita teollisuusprosessien ohjauslaitteita. Selvityksen perusteella valittiin kaksi mallinnukseen sopivaa järjestelmää SCADA ja HMI. SCADA on kehitetty jo 1970-luvulla ja se onkin nykyään yksi käytetyimmistä teollisuusprosessien ohjauslaitteista. SCADA-järjestelmän tehtävänä on kerätä, valvoa ja analysoida tietoja teollisuusprosesseista. Nykyiset SCADA-järjestelmät tarjoavatkin edistyneet etäkäyttöominaisuudet ja ne ilmoittavat käyttäjälle automaattisesti mahdollisista häiriöistä ja hälytyksistä. HMI puolestaan toi-

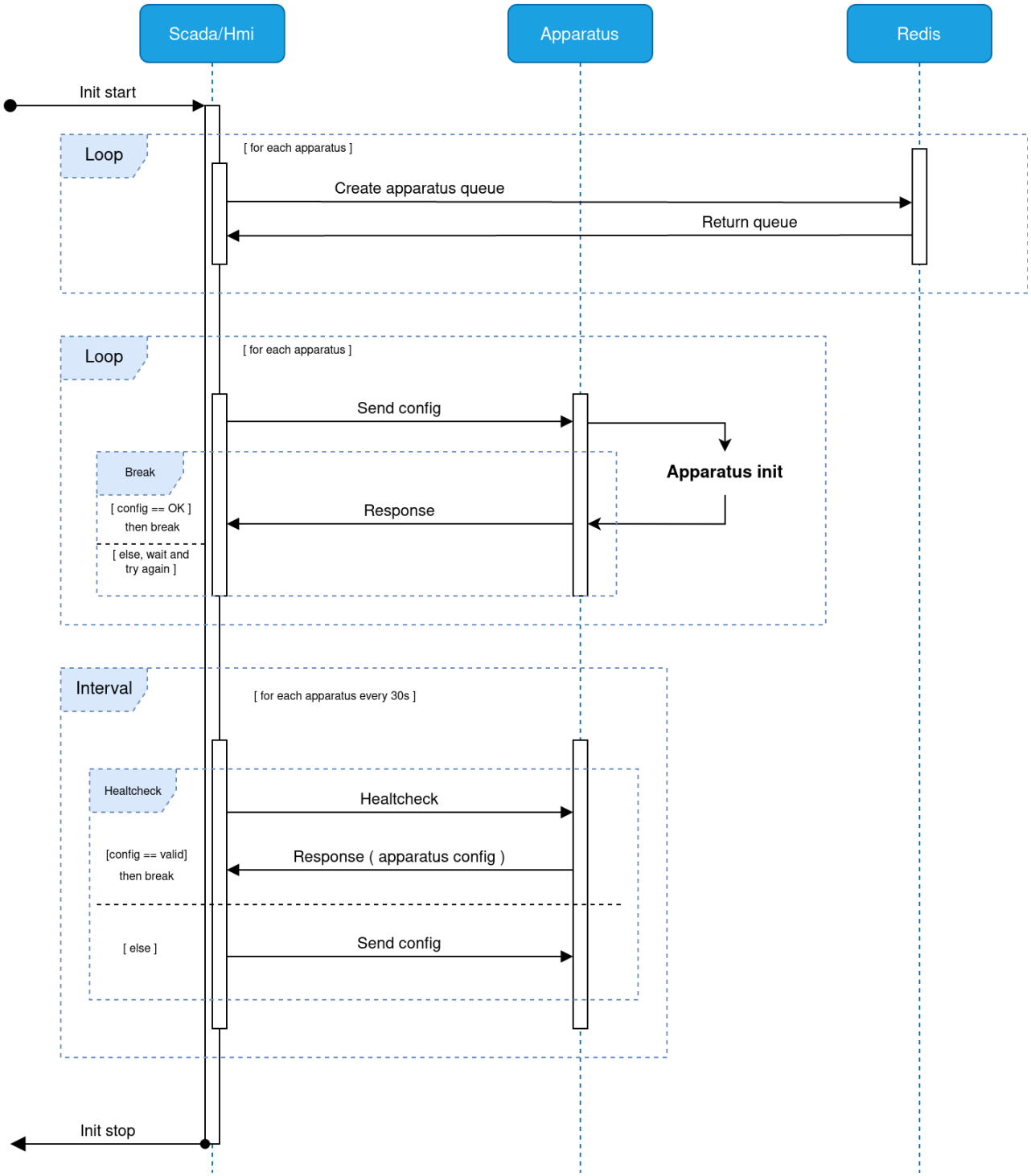
mii käyttöliittymänä laitteiden ja käyttäjän välillä. HMI avulla käyttäjä voi esimerkiksi muuttaa prosessin asetuksia, käynnistää tai pysäyttää prosessin. Tyypillisesti nämä järjestelmät toimivatkin rinnakkain ja mahdollistavat yksittäisten laitteiden hallinnan paikan päältä, että myös etänä. (How SCADA, HMI, and PLC Work Together. 2019.; What is a SCADA System. N.d.)

Ohjainyksikkö-sovellus nimettiinkin SCADA/HMI-sovellukseksi toteutustyön alkupuolella, koska ohjainyksikkö-sovellukseen mallinnettavat ja toteutettavat ominaisuudet vastasivatkin paljon näiden oikeiden laitteiden ominaisuuksia. Ohjainyksikkö-sovellus koostuu käyttäjän selaimessa avattavasta käyttöliittymästä sekä palvelimella suoritettavasta ohjainyksikkö-sovelluksesta.

Kuviossa 2 esitetään ohjainyksikkö-sovelluksen käynnistyksen jälkeen tapahtuvan alustuksen kulku. Ensimmäisenä sovellus määrittää ulkoisen scadaConfig.json tiedoston perusteella muodostettavat tuotantolinjastot. Kyseinen tiedosto sisältää resurssipoolin käytettävistä linjastolaite-sovelluksista, sekä niiden alustukseen tarvittavia tietoja. Ohjainyksikkö-sovellus alustaa viestijonopalvelimelle jokaiselle linjastolaite-sovellukselle omat viestijononsa, sekä liittää myös omat kuuntelijansa kyseisiin viestijonoihin. Näiden kuuntelijoiden avulla pystytään seuraamaan muun muassa valmistuneiden ja hylättyjen tuotteiden määrää tuotantolinjastolla.

Seuraavaksi ohjainyksikkö-sovellus lähettää jokaiselle linjastolaite-sovellukselle niiden tarvitsemat alustustietonsa. Alustustiedot sisältävät linjastolaite-sovellus kohtaisesti tarvittavia tietoja, joiden perustella laite pystyy toimimaan yhtenä osana tuotantolinjastoa. Yksittäisen linjastolaite-sovelluksen alustuksen epäonnistuessa ohjainyksikkö-sovellus lähettää alustustiedot uudestaan tietyin väliajoin kyseiselle laitteelle, niin pitkään kunnes alustus on saatu suoritettua onnistuneesti. Yksittäinen tuotantolinjasto on käyttökunnossa vasta kun kaikki siihen kuuluvat linjastolaite-sovellukset ovat alustettu. Ohjainyksikkö-sovellus pyytää myös taustalla tietyin väliajoin linjastolaite-sovelluksia lähettämään omat alustustietonsa. Mikäli vastaanotetut alustustiedot eivät täsmää ohjainyksikkö-sovelluksen aikaisemmin lähettämiä alustustietoja, alustetaan kyseinen linjastolaite-sovellus uudelleen ohjainyksikkö-sovelluksen lähettämillä alustustiedoilla.

Lopuksi kun kaikki linjastolaite-sovellukset ovat alustettu, ne jäävät odottamaan tuotettavan tuotteen reseptiikkaa, sekä niille kuuluvia viestejä viestijonoista. Myös ohjainyksikkö-sovellus jää odottamaan ulkoisesta MES-tuotannonohjausjärjestelmästä tulevia tuote-erä tilauksia, joita sitten eri tuotantolinjastot aloittavat valmistamaan.



Kuvio 2 Sekvenssikaavio ohjainyksikkö-sovelluksen alustuksesta

Käyttöliittymän suunnittelussa nousi muutamia tärkeitä ominaisuuksia, joita tarvittiin toteutukseen. Esimerkiksi käyttöliittymästä käyttäjän piti pystyä seuraamaan tuotantolinjastojen toimintaa, sekä näkemään reaaliajassa tuotteiden valmistuksen. Tuotantolinjaston rakenne tuli visualisoida dynaamisesti käyttäjälle. Visualisointia piti pystyä tarvittaessa loitontamaan sekä lähentämään, jolloin pystyttäisiin tarkkailemaan yksittäisien linjastolaitteiden toimintaa lähemmin.

Käyttöliittymän kautta käyttäjän piti myös pystyä vaikuttamaan linjastolaitteiden toimintaa. Mahdollisten häiriö tilanteiden sattuessa yksittäisiä linjastolaitteita piti pystyä pysäyttämään ja häiriön poistuessa linjastolaitteiden toimintaa piti pystyä jatkamaan. Tarvittaessa käyttöliittymään piti myös pystyä saamaan pakollinen käyttäjän kirjautumistoiminto päälle. Käyttäjän kirjautumistoiminnossa käytettäisiin erillistä Active Directory-kirjautumispalvelua ja sen tarjoamaa kirjautumisprosessia.

3.4 Linjastolaite-sovelluksen suunnittelu

Linjastolaite-sovelluksen suunnittelun alussa lähdettiin liikkeelle tutkimalla mahdollisuuksia mallintaa oikeita PLC-laitteita. PLC-laitteet ovat kuin pieniä tietokoneita oikeiden automaatiolaitteiden sisällä. PLC-laite myös mahdollistaa tietoliikenneyhteydet sen ja ohjainyksikön, yleensä SCADA:n välillä. (What is a SCADA System and How Does It Work?. 2022.)

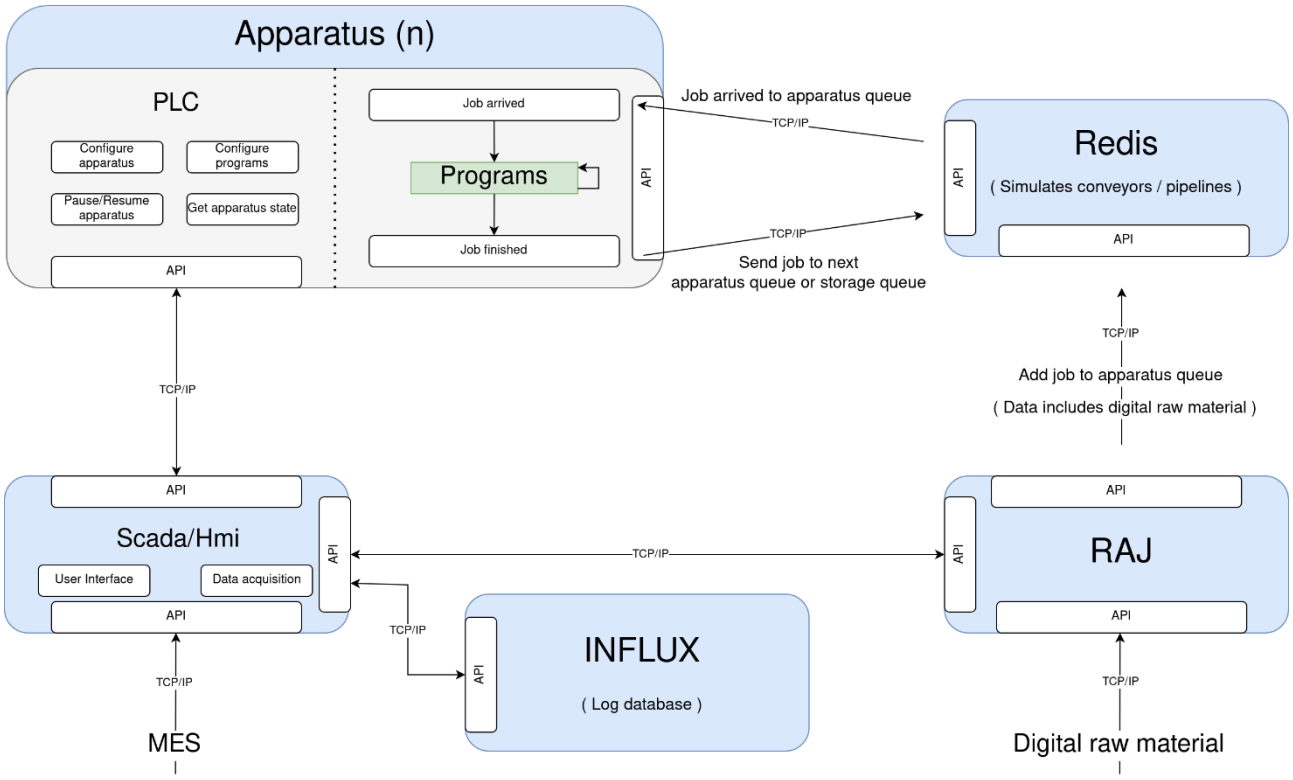
Linjastolaite-sovelluksen suunnittelua ohjasi tieto myös siitä, että tuotantolinjastoista oli tarve saada horisontaalisesti skaalautuvia, tämä tarkoitti käytännössä sitä, että kaikki linjastolaite-sovellukset tulisivat käyttämään samaa koodipohjaa, josta sitten Docker-kontitus teknologiaa käyttämällä tultaisiin skaalaaman tarvittava määrä linjastonlaite-sovelluksia palvelimille. Samaa koodipohjaa käytettäessä haasteeksi tuli saada tuotantolinjaston yksittäiset linjastolaite-sovellukset mallintamaan eri prosesseja tuotantolinjastolla, tämä ongelma ratkaistiin suunnittelemalla linjastolaite-sovellukselle konfiguroitavissa oleva yksilöllinen ohjelmasilmukka, johon pystyttiin asettamaan monia erilaisia simulointiohjelmia.

Kuviossa 3 esitetään linjastolaite-sovelluksen tekninen arkkitehtuuri suunnitelma, sekä myös sen toiminnan kannalta tärkeitä tietoliikenneyhteyksiä. Linjastolaite-sovellus on kuvioissa jaettu katkoviivalla PLC-ohjelma ja simulointiohjelma osiin. PLC-ohjelma osa on yhteydessä ohjainyksikkö-so-

vellukseen, josta se vastaanottaa ohjauskomentoja, sekä myös linjastolaite-sovelluksen omat yksilölliset asetukset vastaanotetaan tässä osassa. Linjastolaite-sovelluksen toiminnan aikaiset statistiikat siirtyvät myös PLC-ohjelma osan tietoliikenneyhteyksien kautta ohjainyksikkö-sovellukselle.

Simulointiohjelma osassa tapahtuu itse digitaalisen raaka-aine datan tai riippuen linjastolaite-sovelluksen sijainnista tuotantolinjastolla, prosessoidaan digitaalisen tuotteen sisältämää dataa, ennalta määriteltyjen simulointiohjelmien mukaan. Tämä simulointiohjelma osa on yhteydessä viestijonopalvelimeen, jonka kautta linjastolaite-sovellus on kytköksissä muihin tuotantolinjastolla oleviin linjastolaite-sovelluksiin.

Näistä kahdesta eri osasta koostuu yksi yksittäinen linjastolaite-sovellus, joka nimettiin toteutuksessa Apparatus-linjastolaitteeksi. Viestijonopalvelimen toiminta simuloi fyysisiä kuljettimia tai putkistoja, joilla digitaaliset raaka-aineet sekä tuotteet liikkuvat linjastolaite-sovelluksien välillä tuotantolinjastolla.

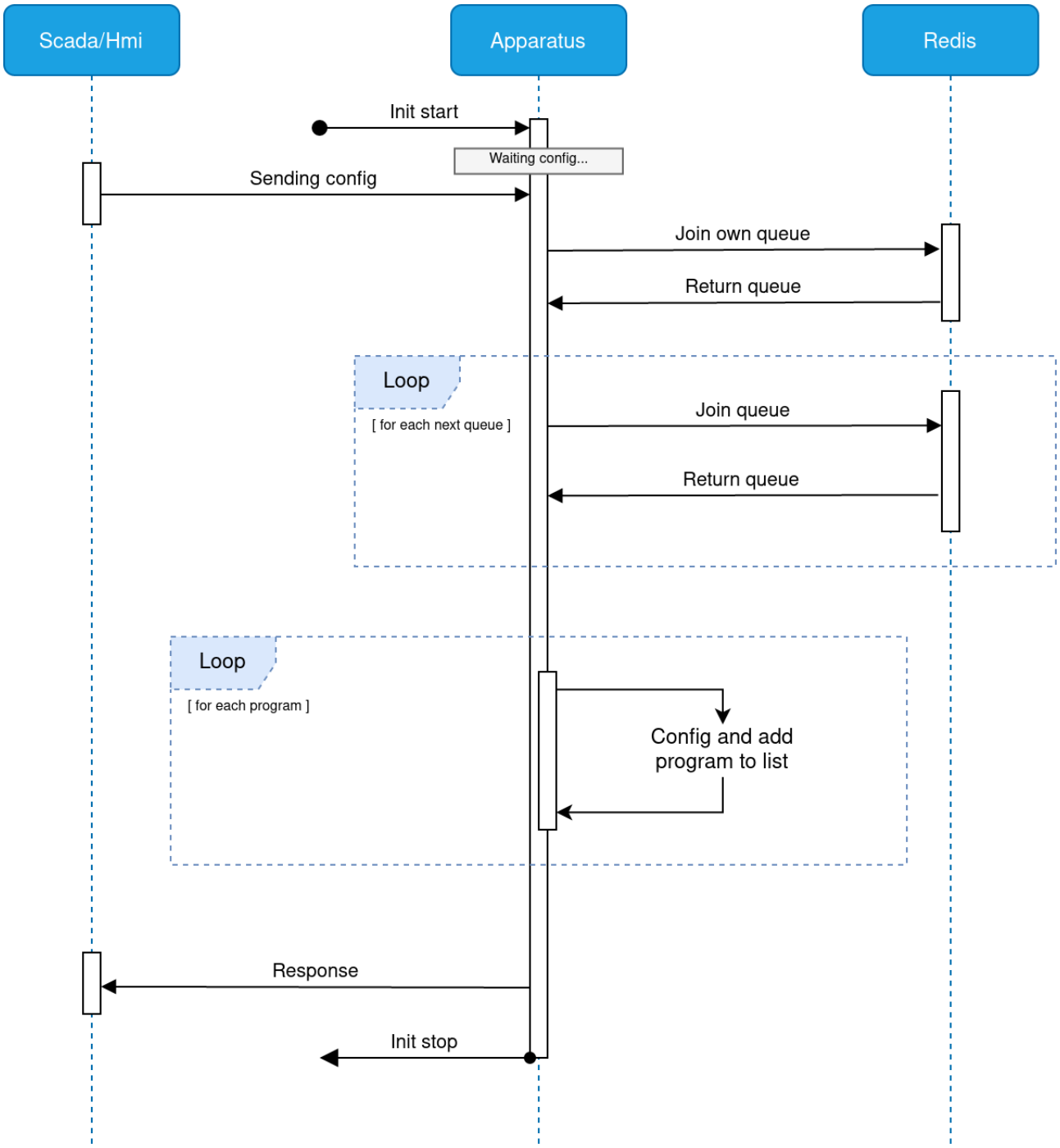


Kuvio 3 Linjastolaite-sovelluksen tekninen arkkitehtuuri

Yksittäinen linjastolaite-sovellus tarvitsee toimiakseen alustuksen, jolla määritellään sen perusasetukset, simulointiohjelmat sekä siihen liitettyjen muiden linjastolaite-sovelluksien tunnistetietoja. Alustustiedoilla linjastolaite-sovellus yksilöidään tekemään vain tiettyjä prosesseja tuotantolinjastolla. Kuviossa 4 esitetään alustuksen kulku. Linjastolaite-sovelluksen käynnistymisen jälkeen sovellus jää odottamaan alustustietoja ohjainyksikkö-sovellukselta. Alustustietojen saavuttua käynnistyy linjastolaite-sovelluksen alustusprosessi. Mahdollisista epäonnistuneista alustuksista kirjataan merkintä ohjainyksikkö-sovelluksen lokiin.

Ensimmäisenä linjastolaite-sovellus liittyy omaan viestijonoonsa viestijonopalvelimen kautta, jolloin se pystyy vastaanottamaan sille kohdistetut viestit tuotantolinjaston muilta linjastolaite-sovelluksilta. Seuraavaksi linjastolaite-sovellus käy läpi ohjelmasilmukassa siihen liitettyjen muiden linjastolaite-sovelluksien tietoja, jonka jälkeen se liittyy myös näiden linjastolaite-sovelluksien viestijonoihin. Tämä viestijonoihin liittyminen mahdollistaa viestien lähettämisen eteenpäin tuotantolinjastolla. Viestin lähettäminen tapahtuu aina kun linjastolaite-sovelluksen kaikki simulointiohjelmat ovat suorittaneet viestin prosessoinnin. Simulointiohjelmat määritellään seuraavaksi linjastolaite-sovellukselle, tämä tapahtuu myös ohjelmasilmukassa. Yksittäisien linjastolaite-sovelluksien toimintoihin voi kuulua monien eri simulointiohjelmien suorittaminen. Monia eri simulointiohjelmia kasaamalla pystytään mallintamaan monimutkaisiakin tuotannon prosesseja yhdellä linjastolaite-sovelluksella.

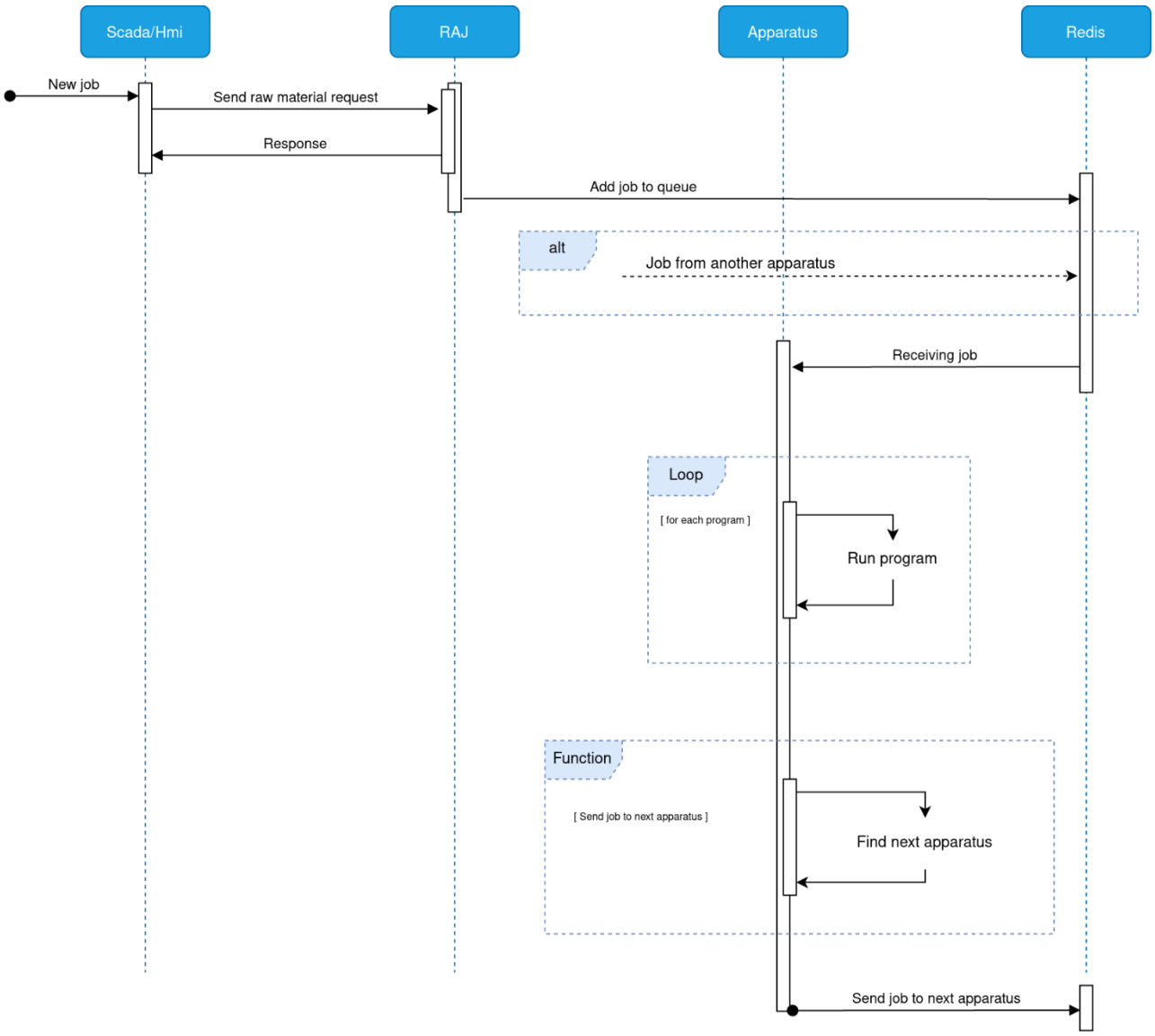
Lopuksi onnistuneen alustuksen jälkeen linjastolaite-sovellus lähettää kuittauksen onnistuneesta alustuksesta ohjainyksikkö-sovellukselle, tämän jälkeen linjastolaite-sovelluksen alustusprosessi on suoritettu. Linjastolaite-sovelluksen alustus tarvitsee tehdä vain kerran sen käynnistyksen yhteydessä. Alustuksen epäonnistuessa ohjainyksikkö-sovellus saa tiedon siitä ja yrittää sen jälkeen alustaa kyseistä linjastolaite-sovellusta uudelleen tietyin väliajoin. Tuotantolinjasto on toimintakunnossa vasta sitten, kun kaikki siihen kuuluvat linjastolaite-sovellukset ovat suorittaneet alustuksen loppuun onnistuneesti.



Kuvio 4 Sekvenssikaavio linjastolaite-sovelluksen alustuksesta

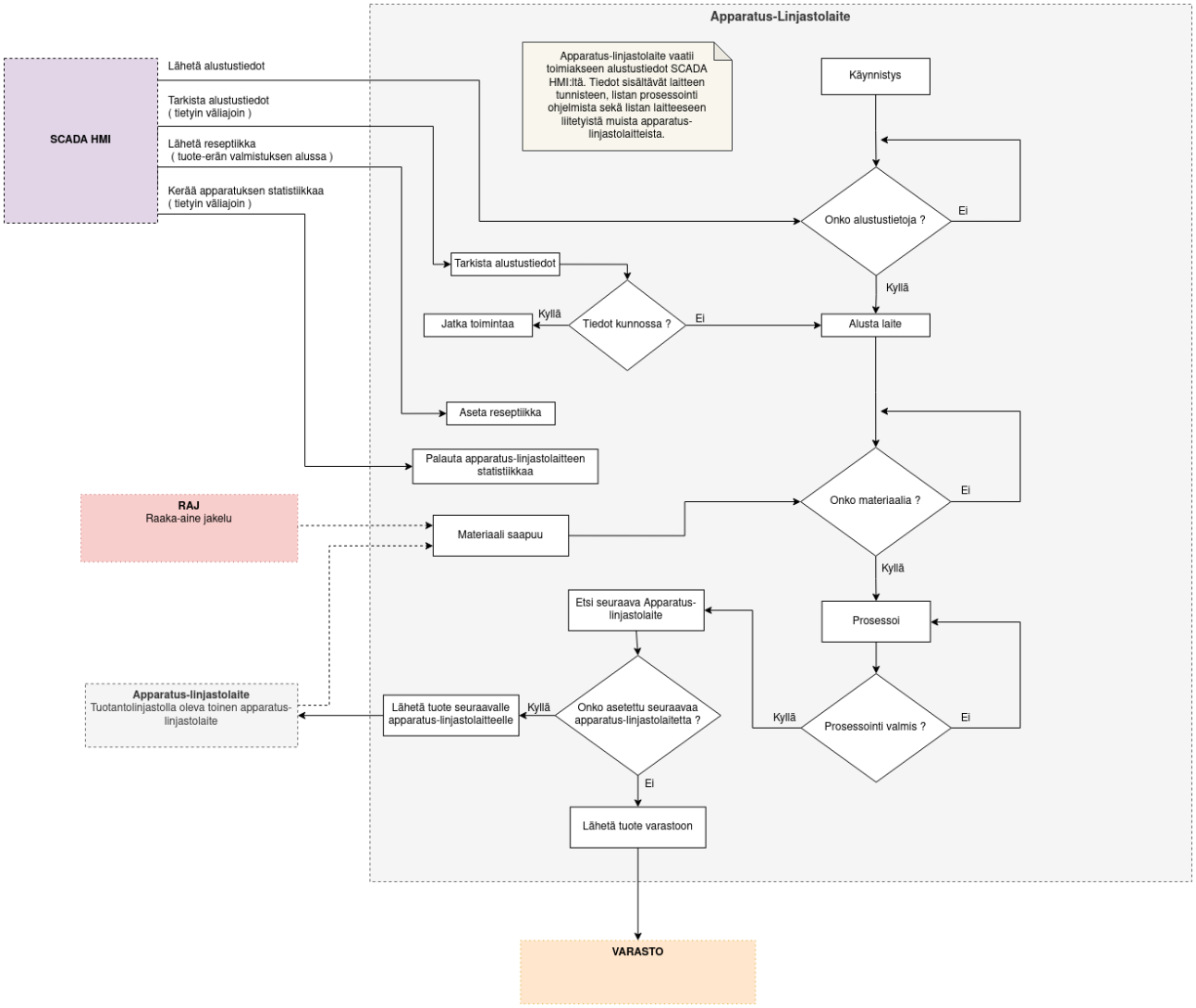
Kuviossa 5 esitetään linjastolaite-sovelluksen keskeinen toiminta. Yksittäisen tuote-erän valmistaminen alkaa ohjainyksikkö-sovelluksen vastaanottamasta ohjauskäskystä, jossa määritellään tuote-erässä valmistettava tuote. Seuraavaksi ohjainyksikkö-sovellus selvittää käytettävissä olevan tuotantolinjaston, millä kyseistä tuote-erää aloitetaan valmistamaan. Sen jälkeen raaka-aine jakelu

järjestelmä saa pyynnön lähettää digitaalista raaka-ainetta tuotantolinjastolla ensimmäisenä olevan linjastolaite-sovelluksen viestijonoon. Digitaalisen raaka-aine viestin saapuessa linjastolaite-sovellus aloittaa prosessoimaan viestiä, asetettujen simulointiohjelmien mukaan. Vaihtoehtoisesti linjastolaite-sovellus voi saada omaan viestijonoonsa viesti myös toiselta linjastolaite-sovellukselta, joka sijaitsee samalla tuotantolinjastolla.



Kuvio 5 Sekvenssikaavio linjastolaite-sovelluksen toiminnasta

Kuviossa 6 esitetään vuokaavio muodossa oleva suunnitelma linjastolaite-sovelluksen toiminnasta. Vuokaavio suunnitelman avulla varmistettiin prosessien toimivuus, ennen toteutustyön aloittamista.



Kuvio 6 Vuokaavio linjastolaite-sovelluksen toiminnasta

4 Toteutus

4.1 Toteutuksen teknologiat

Ohjainyksikkö-sovelluksen sekä linjastolaite-sovelluksen toteutuksissa käytettiin TypeScript-ohjelmointikieltä. Käyttöliittymässä käytössä oli Vue.js ohjelmistokehys. Palvelimella käytössä oli Node.js-ohjelmointiympäristö.

Toteutuksessa hyödynnettiin elintarvikeketjun kyberturvallisuus hankkeen alkupuolella kehitettyä valmista koodipohjaa. Kyseinen koodipohja sisälsi REST API-rajapinta valmiuden, joka oli toteutettu Hapi.js-ohjelmistokehyksellä. Koodipohjassa oli myös valmiiksi käyttäjän autentikointia varten ominaisuudet JWT-token:ia hyödyntäen. Koodipohjasta löytyi myös valmiiksi tietokanta ORM, joka oli tehty käyttäen Sequalize-Node.js -kirjastoa.

Kyseistä koodipohja hyödynnettiin ohjainyksikkö-sovelluksen sekä linjastolaite-sovelluksen toteuttamisessa, koska se sisälsi tarvittavat perustoiminnot, sekä sen käyttö nopeutti alkuvaiheen toteutustyötä, jolloin päästiinkin keskittymään itse varsinaisiin sovelluksien vaatimiin ominaisuuksien toteuttamiseen.

Viestijonotekniikaksi valittiin Node.js:lle tehty Bull.js-ohjelmistokirjasto. Bull.js hyödyntää Redis-palvelinta viestijonopalvelimena ja abstraktoi viestijonoliikenteen toiminnan sovelluksen kehittäjältä (Bull. 2021.). Kyseinen ohjelmistokirjasto integroitui hyvin käytettyyn koodipohjaan, sekä myös muiden valittujen teknologioiden kanssa.

4.2 Ohjainyksikkö-sovelluksen toteutus

Ohjainyksikkö-sovelluksen toteutuksen kokonaisuuteen kuului selaimen kautta toimiva käyttöliittymä, sekä myös palvelimella toimiva ohjainyksikkö-sovellus. Käyttöliittymässä käytössä oli TypeScript-ohjelmointikieli ja Vue.js-ohjelmistokehys. Palvelimella käytössä oli TypeScript-ohjelmointikieli ja Node.js-ohjelmointiympäristö.

Selaimen kautta toimiva käyttöliittymä haki kaikki tarvittavat tiedot näkymiinsä palvelinsovelluksen API-reittien kautta. Käyttöliittymän kautta pystyi myös suorittamaan tiettyjä toimintoja tuotantolinjaston linjastolaitteille. Itse tuotantolinjastojen toiminnan ohjaus tapahtui kuitenkin palvelinsovelluksen ohjaamana. Ohjainyksikkö-sovelluksen toteutus aloitettiin palvelimella toimivasta sovelluksesta. Toteutuksen edetessä pystyttiin siirtymään myös samanaikaisesti toteuttamaan selainpohjaista käyttöliittymää.

Ohjainyksikkö-sovelluksen käynnistyttyä muodostetaan kaikki tuotantolinjastot. Rinnakkaisten tuotantolinjastojen määrälle ei ole asetettu mitään ylärajaa, vaan tuotantolinjastot muodostuvat ulkoisen scadaConfig.json tiedoston tietojen perusteella. ScadaConfig.json tiedosto sisältää resurssipoolin kaikista käytettävistä olevista linjastolaitteista. Tiedostossa itsessään ei ole suoraan määritelty yksittäisien tuotantolinjastojen kokoonpanoa, vaan tämä tieto muodostetaan ohjainyksikkösovelluksen käydessä läpi linjastolaite resurssipoolia ja liittäen linjastolaitteet toisiinsa käyttäen yhdistävänä tekijänä linjastolaitteen tietoihin lisättyä liitostaulukkoa. Tästä taulukosta löytyy objekti, joka sisältää tiedon siitä mihin muihin linjastolaitteisiin yksittäinen linjastolaite on liitetty, sekä kyseisen liitoksen painoarvoluku. Painoarvoluvun avulla linjastolaite-sovellus pystyy selvittämään oikean kohteen, esimerkiksi joka toiselle laitteen läpi menevälle tuotteelle. Näin tuotantolinjaston rataa pystytään haaroittamaan eri linjastolaite-sovelluksien välillä.

Kuviossa 7 esitetään scadaConfig.json tiedoston sisältävän resurssipoolin läpikäyntiä, jonka perusteella yksittäiset tuotantolinjastot muodostetaan. Selvityksen suorittavaa funktiota kutsutaan rekursiivisesti, tämä siksi että selvitettävän liitospolun pituudelle ei ole asetettu ylärajaa, mahdollistaen näin todella monimutkaistenkin tuotantolinjastojen muodostamisen.

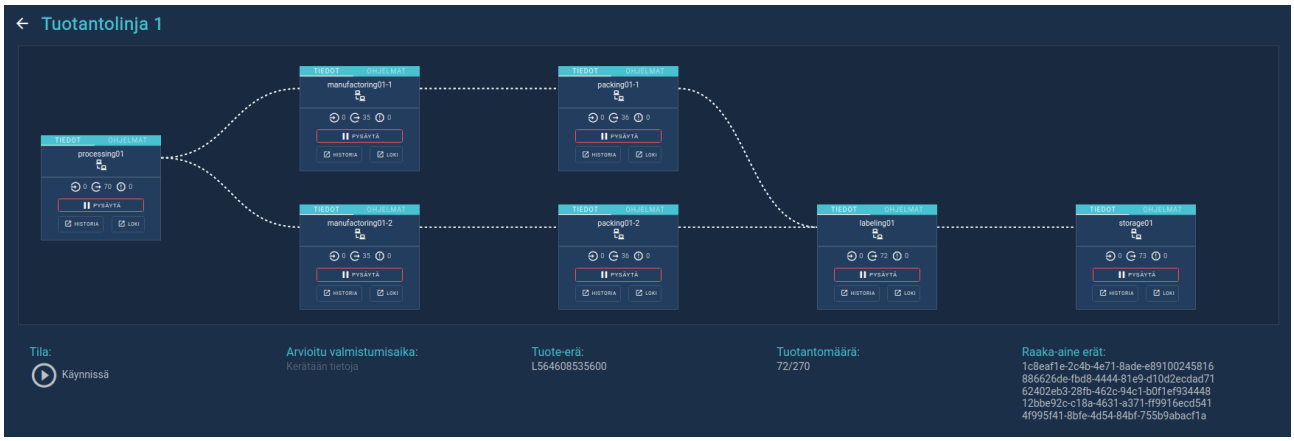
```
// recursive helper function to find all the children apparatus
const findChildren = (node: IApparatus) : Array<IRawApparatus2> => {
  return node.next.map((element: INext) => {
    const node = this.apparatus.find((apparatus: IApparatus) => apparatus.name === element.name)
    if (node === undefined) {
      throw new SyntaxError(`${element.name} not found at scada config`)
    }

    const nodeProperties = {
      ip: node.ip,
      name: node.name,
      program: node.program,
      weight: element.weight,
      disabled: element.disabled
    }

    if (node.next.length === 0) {
      // Last node
      endNodes.push(node.name)
      return {
        ...nodeProperties,
        children: []
      }
    } else {
      // Node has children
      return {
        ...nodeProperties,
        children: findChildren(node)
      }
    }
  })
}

// Each begin node is equivalent to one production line
for (const beginNode of beginNodes) {
  try {
    const node = this.apparatus.find((apparatus: IApparatus) => apparatus.name === beginNode)
    if (node !== undefined) {
      productionLines.push({
        ip: node.ip,
        name: node.name,
        program: node.program,
        children: findChildren(node)
      })
    }
  } catch (error) {
    if (error instanceof RangeError) {
      ScadaHmiLogger.fatal('Error in scada config, please check config and try again.')
    }
    throw error
  }
}
```

Kuvio 7 Tuotantolinjastojen muodostaminen

Tuotantolinjastojen muodostaminen aloitetaan keräämällä tiedot linjastolaitteista, joita ei ole asetettu minkään muun linjastolaitteen liitostaulukkoon. Nämä kerätyt linjastolaitteet ovat yksittäisien tuotantolinjastojen alussa oleva aloitus linjastolaitteita ja näiden lukumäärä on suoraan verrannollinen tuotantolinjastojen määrään. Puolestaan linjastolaitteet, joiden liitostaulukossa ei ole asetettu yhtään linjastolaitetta, ovat tuotantolinjaston viimeisiä laitteita. Tämän selvityksen

perustella ohjainyksikkö-sovellus osaa muodostaa kaikki eri tuotantolinjastot, sekä alustaa kaikki linjastolaite-sovellukset oikeilla tiedoilla.

Kuviossa 8 esitetään käyttöliittymässä visualisoitu tuotantolinjasto, tämä visualisointi on täysin dynaaminen ja se tehdään palvelinsovelluksen keräämien tietojen perusteella.



Kuvio 8 Näkymä yksittäisestä tuotantolinjastosta käyttöliittymässä

Kuviossa 9 esitetään linjastolaite-sovelluksien alustustietojen lähetysfunktio. Alustustietojen lähetys toteutetaan silmukassa. Mikäli linjastolaite-sovellukseen ei saada yhteyttä tietyn ajan kuluessa, yrittää ohjainyksikkö-sovellus hetken kuluttua uudelleen alustaa kyseistä linjastolaite-sovellusta. Muun virheen sattuessa alustusta yritetään myöhemmin uudelleen ohjainyksikkö-sovelluksen tekemien tarkastuksien yhteydessä. Palvelimen lokiin jää tieto tapahtuneista virheistä. Kyseinen alustusfunktio on asynkroninen, mikä mahdollistaa ohjainyksikkö-sovelluksen lähettää alustustiedot samanaikaisesti kaikille linjastolaite-sovelluksille. Tämä toimintatapa nopeuttaa tuotantolinjastojen käyttökuntoon saamisessa.

```
private sendApparatusInitConfig = async (apparatus: IApparatus) : Promise<void> => {
  /**
   * Post config to specific apparatus
   * - Try until config is successfully sended
   * - Break if some catastrophic error occur
   */
  while (true) {
    try {
      await this.modBusTCP.post(`${apparatus.ip}:${config.modBusTCP.port}/api/apparatus/config`, {
        id: apparatus.name,
        program: apparatus.program,
        next: apparatus.next
      })
      ScadaHmiLogger.info(`${apparatus.name} | Init config send succesfully`)
      break
    } catch (error) {
      if (error instanceof AxiosError) {
        ScadaHmiLogger.error(`${apparatus.name} | Init config error | ${error.message}`)
        await new Promise((resolve) => { setTimeout(resolve, 5000) }) // wait and try again
        continue
      }
      GeneralLogger.error(error)
      break
    }
  }
}
```

Kuvio 9 Lähetysfunktio linjastolaite-sovelluksien alustukseen

Uuden tuote-erä tilauksen saapuessa ohjainyksikkö-sovellukselle, suoritetaan tilauksen käsittelevä funktio. Kuviossa 10 esitetään kyseisen funktion toiminta. Ensimmäisenä tarkastetaan, löytyykö vapaata tuotantolinjastoa. Tähän tarkastukseen hyödynnetään kuviossa 11 esitettyä etsintäfunktiota. Etsintäfunktion tarkastaa ensin löytyykö kyseisellä tuote-erä tiedoilla jo varattua, mutta ei toiminnassa olevaa tuotantolinjastoa. Raaka-aineen riittämättömyys tai reseptiikan lähetyksessä tapahtunut virhe aiheuttaa tämän kaltaisen tilanteen. Mikäli tuote-erä tiedoilla ei löydy varattua tuotantolinjasto, tarkastetaan seuraavaksi mahdolliset vapaat sekä valmiit tuotantolinjastot. Funktion löytäessä tuotantolinjaston, palauttaa se löydetyn tuotantolinjasto-objektin. Mikäli vapaata tuotantolinjastoa ei tarkastuksessa löydy, keskeytyy uuden tuote-erä tilauksen käsittely. Keskeytyneitä tuote-erä tilauksia yritetään aloittaa uudelleen tietyn ajanvälein.

Vapaan tuotantolinjaston löydyttyä, varataan kyseinen tuotantolinjasto tuote-erän valmistukseen. Seuraavaksi lähetetään valmistettavan tuotteen reseptiikka tuotantolinjaston linjastolaite-sovelluksille. Lopuksi raaka-aine jakelu järjestelmälle lähetetään raaka-aine materiaali pyyntö. Jos raaka-ainetta on riittävästi tuote-erän valmistukseen, aloitetaan tuotanto tuotantolinjastolla.

```
export const ordersInit = async (orders : Queue, productionLines : Array<IProductionLine>, modBusTCP : Axios) : Promise<void> => {
  /**
   * Define queue process
   */
  orders.process('order', async (job: IJobOrder) => {
    let lineNumber : number | undefined
    try {
      /**
       * 1: Find a free production line ( if not found, throw error )
       * 2: Config production line apparatus ( if failed, throw error )
       * 3: Send order to RAJ ( If failed, throw error )
       */
      const productionLine = await findFreeProductionLine(job.data, true)
      lineNumber = productionLine.line
      await scadaHmi.configProductionLine(lineNumber, job.data)

      const response = await modBusTCP.post(`${config.raj.host}:${config.raj.port}/api/order`, {
        id: job.data.lotId,
        // calculate and add some extra amount to ensure smooth production
        amount: Math.ceil((job.data.product.content.batchAmount * job.data.productAmount) + (job.data.product.content.batchAmount * 10)),
        sendTo: productionLines[lineNumber].name
      })
      if (response.status === 200) {
        ScadaHmiLogger.info(`Order ${job.data.lotId} processing started | Product line: ${lineNumber}`)
        await job.log(`Order processing started | Product line: ${lineNumber} | ${new Date()}`)
        await changeProductionLineState(lineNumber, states.BUSY)
        await job.progress(100)
        return Promise.resolve(true)
      }
    } catch (error) {
      // Insufficient material ERROR
      if (error instanceof AxiosError && error.response?.status === 409) {
        await changeProductionLineState(lineNumber, states.ERROR, OrderFailReason.InsufficientMaterial)
        return Promise.reject(new Error(OrderFailReason.InsufficientMaterial))
      }

      // No free production line ERROR
      if (error instanceof NoFreeProductionLineError) {
        return Promise.reject(new Error(OrderFailReason.NoFreeProductionLine))
      }

      // Production line device ERROR
      if (error instanceof ProductionLineDeviceError) {
        await changeProductionLineState(lineNumber, states.ERROR, OrderFailReason.ProductionLineDevice)
        return Promise.reject(new Error(OrderFailReason.ProductionLineDevice))
      }

      // General ERROR
      ScadaHmiLogger.error(error)
      if (lineNumber !== undefined) {
        await changeProductionLineState(lineNumber, states.ERROR, OrderFailReason.General)
      }
      return Promise.reject(new Error(OrderFailReason.General))
    }
  })
```

Kuvio 10 Uuden tuote-erä tilauksen aloitus

```
/**
 * Finds one free production line and return it
 * - If reserveLine === true, then reserve production line by changing it state to RESERVED
 * - Throws NoFreeProductionLineError if no free production line found
 */
export const findFreeProductionLine = async (order: ILot, reserveLine = false) : Promise<ProductionLine> => {
  const transaction = await Database.transaction()
  try {
    /**
     * Priority order
     * 1: State = ERROR and lotId = order.lotId
     * 2: State = IDLE or READY
     */
    let productionLine = await ProductionLine.scope('productionLineResponse').findOne({
      where: {
        state: states.ERROR,
        lotId: order.lotId
      },
      rejectOnEmpty: false,
      transaction
    })

    if (productionLine === null) {
      productionLine = await ProductionLine.scope('productionLineResponse').findOne({
        where: {
          state: { [Op.or]: [states.IDLE, states.READY] }
        },
        rejectOnEmpty: true,
        transaction
      })
    }

    if (reserveLine) {
      await productionLine.update({
        state: states.RESERVED, // This will change to BUSY state, when the production line actual start
        lotId: order.lotId,
        product: order.product,
        amount: order.productAmount,
        amountReady: 0,
        amountFailed: 0,
        batches: null, // RAJ fills this field later
        errorMsg: null
      }, { transaction })
    }

    await transaction.commit()
    return productionLine
  } catch (error) {
    await transaction.rollback()
    if (error instanceof Sequelize.EmptyResultError) {
      throw new NoFreeProductionLineError()
    }
    throw error
  }
}
```

Kuvio 11 Vapaana olevan tuotantolinjaston etsintä logiikka

Ohjainyksikkö-sovelluksen käyttöliittymästä löytyy myös valvontanäkymä. Kuviossa 12 esitetään tuotantolinjaston valvontanäkymä. Tässä näkymässä tarkasteltavan tuotantolinjaston kaikki linjastolaitteet ovat erikseen visualisoituna.

Yläpuolella olevassa osassa on linjastolaitteen perustiedot ja toiminnot. Liitokset edellisiin ja seuraaviin linjastolaitteisiin on visualisoitu käyttäen viivaa. Liitoksen ollessa toiminnassa käytetään katkoviivaa, joka on animoitu liikkumaan. Jos edellisen tai seuraavan linjastolaitteen toiminta on pysäytetty, viivan animointi pysäytetään, sekä vaihdetaan käytettäväksi yhtenäistä viivaa. Tämän ansioista käyttäjä pystyy hahmottamaan muutokset tuotantolinjaston toiminnassa nopeasti.

Alapuolen osassa esitetään kuvaaja linjastolaitteen toiminnasta ajan funktiona. Kuvaajasta pystytään havainnoimaan prosessoitujen viestien lukumäärää. Hylättyjen viestien lukumäärä selviää myös kuvaajasta. Kuvaaja on täysin dynaaminen ja käyttäjä pystyy halutessaan rajaamaan kuvaajan esittämää aikaväliä. Kuvaajassa näkyvät tiedot kerätään InfluxDB-aikasarjatietokannasta.

Kuvio 12 Valvontanäkymä ohjainyksikkö-sovelluksen käyttöliittymässä

4.3 Linjastonlaite-sovelluksen toteutus

Linjastolaite-sovelluksen toteutukseen kuului pelkästään palvelimella toimiva sovellus. Käyttäjällä ei ole mahdollisuutta päästä vaikuttamaan suoraan linjastolaitteiden toimintaa, vaan linjastolaitteiden toimintaan liittyvät muutokset tehdään keskitetysti ohjainyksikkö-sovelluksen tarjoaman käyttöliittymän kautta. Linjastolaite-sovelluksen toteutuksessa käytettiin TypeScript-ohjelmointikieltä ja Node.js ohjelmointiympäristöä.

Sovelluksen toteuttamisessa hyödynnettiin pääosin samaa valmista koodipohjaa, kuin ohjainyksikkö-sovelluksessakin. Tiettyjä käyttöliittymään liittyviä ominaisuuksia kuitenkin poistettiin koodista. Koodinpohjan hyödyntäminen nopeutti paljon alkuvaiheen toteutustyötä ja melko nopeasti päästiinkin keskittymään itse sovelluksen kannalta tärkeiden ominaisuuksien toteuttamiseen.

Kuviossa 13 esitetään linjastolaite-sovelluksen alustuksen toteuttava asynkroninen funktion. Funktio vastaanottaa parametreinä objektin, mikä sisältää alustukseen tarvittavat tiedot. Ensimmäisenä linjastolaite-sovelluksen mahdollinen aikaisempi alustus nollataan, näin varmistutaan siitä, että aikaisemmin epäonnistuneen alustuksen mahdolliset tiedot poistetaan. Tämän jälkeen lisätään vastaanotetut alustustiedot linjastolaite-sovelluksen käyttöön. Sitten lajitellaan taulukko, joka sisältää tiedot muista linjastolaite-sovelluksista, jotka on liitetty kyseiseen linjastolaite-sovellukseen. Lajittelu tapahtuu liitoksen painoarvon mukaan laskevasti. Seuraavaksi lisätään käytettävät simulointiohjelmat omaan taulukkoon. Alustustoimenpiteen viimeinen vaihe on liittyminen omaan, sekä muiden alustustiedoissa olevien linjastolaite-sovelluksien viestijonoihin viestijonopalvelimen kautta.

```
public setApparatusConfig = async (apparatusConfig: IApparatusConfig) : Promise<IApparatusConfigResponse> => {
  try {
    await this.resetApparatus()
    this.apparatusConfig = apparatusConfig
    this.apparatusConfig.next.sort((a, b) => b.weight - a.weight) // Sort next apparatus queues by their weights ( descending order )

    /**
    * Set apparatus program list
    */
    for (const program of apparatusConfig.program) {
      const programFound = apparatusPrograms.find(obj => obj.name === program.name)
      if (programFound !== undefined) {
        this.programs.push({ name: program.name, handler: programFound, data: program.data ? { ...program.data } : {} })
        ApparatusLogger.info(`Added program: ${program.name} to program list ${program.data ? (`| data: ${JSON.stringify(program.data)}`) : ''}`)
        continue
      }
      ApparatusLogger.error(`Program: ${program.name} not found`)
    }
```

Kuvio 13 Linjastolaite-sovelluksen alustus sekä simulointiohjelmien lisääminen

Linjastolaite-sovellukseen ladattavat simulointiohjelmat ovat jaettu omiksi tiedostoiksi. Tämän tiedosto rakenteen ansiosta yksittäisien simulointiohjelmien lisääminen ja poistaminen ei vaadi isoja muutoksia linjastolaite-sovelluksen lähdekoodiin. Uusien simulointiohjelmien kehittäminen on myös selkeämpää, koska näiden ohjelmien lähdekoodit sijaitsevat omissa tiedostoissaan. Kuviossa 14 esitetään käytettävissä olevat simulointiohjelmat, sekä niiden lisääminen objektitaulukkoon.

Alustuksen yhteydessä kyseisestä taulukosta etsitään ohjainyksikkö-sovelluksen lähettämien alustustietojen perustella käytettävät simulointiohjelmat. Pelkästään löydetyt simulointiohjelmat lisätään linjastolaite-sovelluksen käyttöön.

```
/**
 *  Apparatus programs
 */
import { combiner } from './combiner'
import { collector } from './collector'
import { qualityControl } from './qualityControl'
import { sendToStorage } from './sendToStorage'
import { labelProduct } from './labelProduct'
import { addNutrients } from './addNutrients'
import { addVitamins } from './addVitamins'
import { labelSequence } from './labelSequence'
import { weighProduct } from './weighProduct'
import { dummyJobs } from './dummyJobs'

export const apparatusPrograms = [
  ...Object.values(collector),
  ...Object.values(combiner),
  ...Object.values(qualityControl),
  ...Object.values(sendToStorage),
  ...Object.values(labelProduct),
  ...Object.values(addNutrients),
  ...Object.values(addVitamins),
  ...Object.values(labelSequence),
  ...Object.values(weighProduct),

  // dummyJobs
  ...Object.values(dummyJobs)
]
```

Kuvio 14 Simulointiohjelmien lataus

Linjastolaite-sovelluksen päätoiminta koostuu kuviossa 15 esitetystä viestijonokuuntelijan prosessiin liitetystä anonyymistä asynkronisesta funktiosta. Tässä funktiossa suoritetaan kaikki alustuksessa asetetut simulointiohjelmat. Ensimmäisenä parametrinä funktio saa viestiobjektin, joka sisältää itse viestin datan, sekä viestinkäsittelyyn liittyviä muita funktioita. Toinen funktiolle tuleva parametri on viestinprosessoinnin jälkeen käytettävä kuittausfunktio, joka ilmoittaa viestijonopalvelimelle onnistuneesta viestin prosessoinnista. Simulointiohjelmien luku määrälle ei ole asetettu ylärajaa, vaan alustuksessa asetetut simulointiohjelmat suoritetaan silmukassa vuoron perään. Viestin saapuessa käynnistyy prosessointi ja ensimmäisenä lokeihin lisätään käynnistysaikaleima ja sen jälkeen aloitetaan suorittamaan asetettuja simulointiohjelmia. Yksittäisien simulointiohjelmien aloituksesta ja lopetuksesta lisätään myös aikaleimat lokeihin.

Yksittäinen simulointiohjelma pystyy palauttamaan viestin prosessoinnin jälkeen erilaisia palautusarvoja. Näillä palautusarvoilla pystytään vaikuttamaan yksittäisen viestin prosessoinnin kokonaisuuteen. Palautusarvona voi olla simulointiohjelman suorituksen aikana tapahtunut virhe, käsky keskeyttää simulointiohjelmia suorittava silmukka tai käsky ohittaa viestin lähetys seuraavan linjastolaite-sovelluksen viestijonoon. Palautusarvoja käyttämällä pystytään esimerkiksi hylkäämään yksittäinen valmistettava tuote, jonka painontarkistus simulointiohjelma on merkannut virheelliseksi. Tämän jälkeen kyseisen viestin jatkoprosessointi lopetetaan tuotantolinjastolla.

```
/**
* Set apparatus job process
*/
this.queue.process('*', 1, async (job : Job, done) => {
  try {
    await job.log(`Start | time: ${Date.now()}`)
    await new Promise((resolve) => { setTimeout(resolve, 50) })

    for (const program of this.programs) {
      await job.log(`Program: ${program.name} start | time: ${Date.now()}`)
      ApparatusLogger.debug(`Program ${program.name} started`)
      await new Promise((resolve) => { setTimeout(resolve, 50) })

      try {
        await program.handler(job, this, program.name, program.data)
      } catch (error) {
        ApparatusLogger.error(error)
      }

      await job.log(`Program: ${program.name} stop | time: ${Date.now()}`)
      ApparatusLogger.debug(`Program ${program.name} finished`)
      if (job.returnvalue?.forceStop) break
    }

    await new Promise((resolve) => { setTimeout(resolve, 50) })
    await job.progress(100)
    await job.log(`Finished | time: ${Date.now()}`)

    if (job.data.logs === undefined) job.data.logs = []
    job.data.logs = [...job.data.logs, ...[{ [this.queue?.name ? this.queue.name : 'unknown']: await this.queue?.getJobLogs(job.id) }]]

    if (job.returnvalue?.error) { done(job.returnvalue.error); return }
    if (job.returnvalue?.skipNext) { done(); return }
    await this.sendToNextApparatus(job)
    done()
  } catch (error) {
    ApparatusLogger.error(error)
  }
})
```

Kuvio 15 Linjastolaite-sovelluksen toiminta funktio

Kun linjastolaite-sovellus on suorittanut kaikki simulointiohjelmat loppuun, aloitetaan selvittämään prosessoidun viestin seuraavaa kohdetta. Linjastolaite-sovelluksen alustuksessa asetettu liitoslista sisältää tiedot liitetyistä muista linjastolaite-sovelluksista, sekä tämän liitoksen painoarvo luvun.

Kuviossa 16 esitetään tämän listan läpikäynti logiikkaa. Ensimmäisenä tarkastetaan linjastolaite-sovelluksen laitelistasta, että se sisältää tietoja muista linjastolaite-sovelluksista, koska tuotantolinjaston viimeisillä linjastolaite-sovelluksilla kyseinen lista on tyhjä. Myös mahdolliset simulointiohjelmien lisäämät merkit otetaan huomioon tarkistuksessa. Mikäli kyseinen lista on tyhjä tai lähetä tuotevarastoon merkki on lisätty, ohjataan prosessoitu viesti varaston viestijonoon, odottamaan varastointia.

Mikäli nämä ehdot eivät täyty, aloitetaan selvittämään seuraavan linjastolaite-sovelluksen viestijonoa, mihinkä prosessoitu viesti ohjataan. Liitoslistalta löytyvän yksittäisen linjastolaite-sovelluksen liitoksen painoarvo luku vastaa lukua siitä monesko läpimennyt prosessoitu viesti ohjataan kyseisen linjastolaite-sovelluksen viestijonoon. Esimerkiksi painoarvo luvulla 2, joka toinen viesti ohjattaisiin kyseisen linjastolaite-sovelluksen viestijonoon. Vastaavasi painoarvo luvulla 1 jokainen viesti ohjattaisiin kyseisen linjastolaite-sovelluksen viestijonoon. Paino-arvo luvulle ei ole tarvetta asettaa ylärajaa, koska laskukaavassa hyödynnetään modulo laskentaa.

Modulo laskentaa hyödyntämällä lasketaan painoarvo luvun ja läpimenneiden viestien lukumäärän modulo arvo. Laskun tuloksen ollessa 0 on seuraava viestijono saatu selville. Laskukaavan toiminnan kannalta on tärkeää, että painoarvo luvut on lajiteltu laskevasti linjastolaite-sovelluksen alustuksen yhteydessä, jolloin painoarvo luvulla 1 oleva linjastolaite-sovellus tulee aina listan viimeiseksi.

```
private sendToNextApparatus = async (job: Job) : Promise<void> => {
  /**
  * Send job to next apparatus queue
  * - Calculate modulo values for all the weights. Modulo zero is the winner
  * - If next queue array is empty, then send job to the storage
  * - If job.returnvalue.sendToStorage === true, then override possible next queue and send job to the storage
  **/
  try {
    if (this.apparatusConfig === undefined) {
      throw new Error('Apparatus config is undefined')
    }

    // Check if job should be send to the storage
    if (this.apparatusConfig.next.length === 0 || job.returnvalue?.sendToStorage === true) {
      if (this.storage !== undefined) {
        await this.storage.add('product', job.data)
        ApparatusLogger.debug(`Job passed to storage: ${this.storage?.name} succesfully`)
        return
      }
      throw new Error('Storage is undefined')
    }

    // Find next queue
    const completedCount = await this.queue?.getCompletedCount()
    for (const next of this.apparatusConfig.next) {
      if (completedCount !== undefined && completedCount % next.weight === 0 && next.disabled === false) {
        const nextQueue = this.nextQueue.find(queue => queue.name === next.name)
        if (nextQueue !== undefined) {
          await nextQueue.add('product', job.data)
          ApparatusLogger.debug(`Job passed to ${next.name} succesfully`)
          return
        }
      }
    }
    throw new Error('Next apparatus not found')
  } catch (error) {
    ApparatusLogger.error(error)
  }
}
```

Kuvio 16 Prosessoidun viestin lähettäminen eteenpäin tuotantolinjastolla

Ohjainyksikkö-sovellus kerää kaikista linjastolaite-sovelluksista statistiikkaa, sekä lokia simulointiohjelmien toiminnasta. Kerätyt statistiikat ja lokit tallennetaan erilliselle InfluxDB-aikasarjatietokanta palvelimelle. Aikasarjatietokannan hyödyntäminen lokien tallennukseen helpottaa datan visualisoinnissa, koska pystytään hyödyntämään monimutkaisijakin tietokantakutsuja lokeja haettaessa. Lokeihin tallentamisen ja haku prosessien ulkoistaminen aikasarjatietokanta palvelimen hoidettavaksi keventää myös linjastolaite-sovelluksen kuormitusta. Tietokantakutsussa pystytään myös määrittämään aikaväli, jolta lokeja haetaan, sekä myös yksittäisiä laitteita voidaan määrittää haku termeiksi tietokantakutsuun.

Kuviossa 17 esitetään labelointi laitteeksi määriteltynä olevan linjastolaite-sovelluksen tuottamaa lokia. Näkymä on esitetty ohjainyksikkö-sovelluksen käyttöliittymän kautta. Lokeista selviää linjastolaite-sovelluksen yksittäisien simulointiohjelmien aloitus ja lopetus ajankohdat, sekä myös simulointiohjelmien suorituksien aikana tapahtuneet prosessoinnit digitaalisen tuotteen datalle.

Käyttäjälle tämä lokien esittäminen on tärkeä ominaisuus, koska sen avulla pystytään arvioimaan, toimiiko linjastolaite halutulla tavalla ja ovatko tuote-erässä valmistuneet tuotteen oikeanlaisia. Mahdolliset haitalliset kybervaikutukset nousevat käyttäjälle esiin myös melko nopeasti lokeja tarkastelemalla. Lokien säilyvyyttä pystytään hallinnoimaan aikasarjatietokantapalvelimelle määritetyillä asetuksilla. Esimerkiksi edellisten 6 kuukauden lokien säilyttäminen on mahdollista, koska säilytysaikaa rajoittaa vain käytettävissä oleva tallennustilan määrä palvelimella.

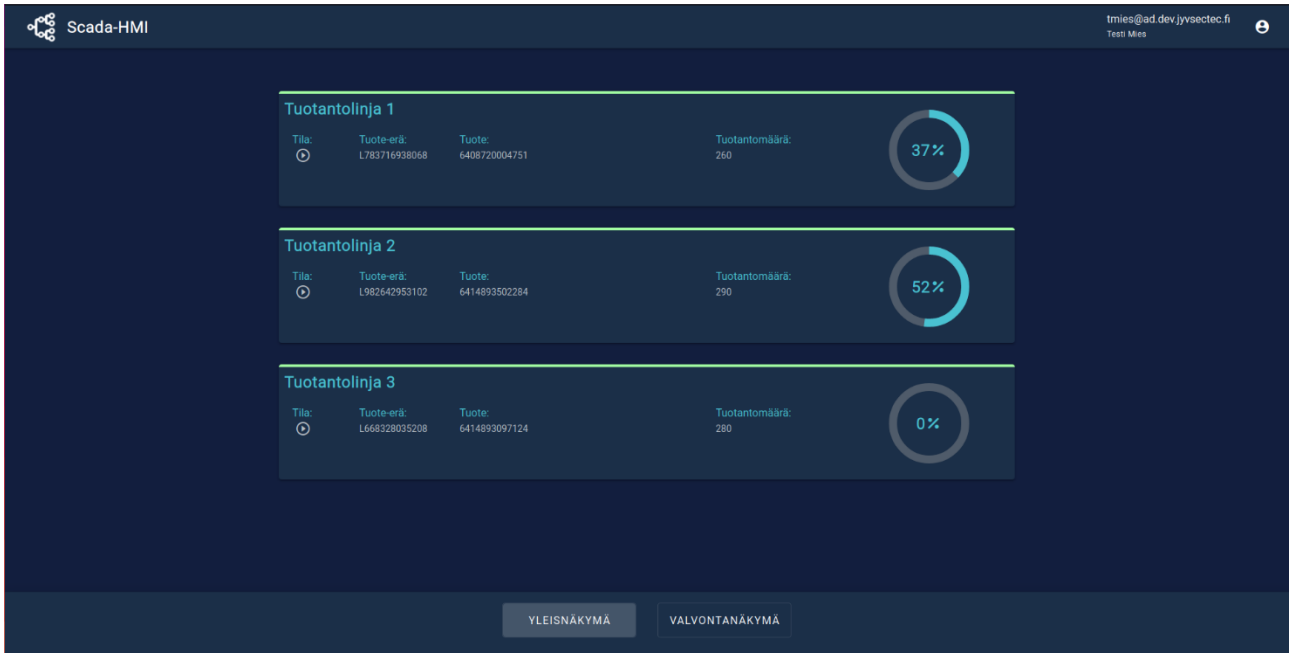
```
> Finished | time: 1682588781908
> Program: labelSequence stop | time: 1682588781857
> Program: labelSequence labeled the sequence number | sequence number: 135/270 | time: 1682588781756
> Program: labelSequence start | time: 1682588781705
> Program: labelProduct | expiration date: Mon May 22 2023 09:46:20 GMT+0000 (Coordinated Universal Time) | time: 1682588780704
> Program: labelProduct | production date: Thu Apr 27 2023 09:46:20 GMT+0000 (Coordinated Universal Time) | time: 1682588780653
> Program: labelProduct | product name: Valio raejuusto 200 g laktoositon | time: 1682588780602
> Program: labelProduct | ean: 6408430043002 | time: 1682588780551
> Program: labelProduct | lot id: L878766798556 | time: 1682588780501
> Program: labelProduct start | time: 1682588780400
> Start | time: 1682588780350
> Finished | time: 1682588780346
> Program: labelSequence stop | time: 1682588780295
> Program: labelSequence labeled the sequence number | sequence number: 134/270 | time: 1682588780194
> Program: labelSequence start | time: 1682588780143
> Program: labelProduct | expiration date: Mon May 22 2023 09:46:18 GMT+0000 (Coordinated Universal Time) | time: 1682588779141
> Program: labelProduct | production date: Thu Apr 27 2023 09:46:18 GMT+0000 (Coordinated Universal Time) | time: 1682588779091
> Program: labelProduct | product name: Valio raejuusto 200 g laktoositon | time: 1682588779040
> Program: labelProduct | ean: 6408430043002 | time: 1682588778990
> Program: labelProduct | lot id: L878766798556 | time: 1682588778939
> Program: labelProduct start | time: 1682588778838
> Start | time: 1682588778788
> Finished | time: 1682588778785
> Program: labelSequence stop | time: 1682588778733
> Program: labelSequence labeled the sequence number | sequence number: 133/270 | time: 1682588778632
> Program: labelSequence start | time: 1682588778582
> Program: labelProduct stop | time: 1682588778581
> Program: labelProduct | expiration date: Mon May 22 2023 09:46:17 GMT+0000 (Coordinated Universal Time) | time: 1682588777579
> Program: labelProduct | production date: Thu Apr 27 2023 09:46:17 GMT+0000 (Coordinated Universal Time) | time: 1682588777528
> Program: labelProduct | product name: Valio raejuusto 200 g laktoositon | time: 1682588777477
> Program: labelProduct | ean: 6408430043002 | time: 1682588777426
> Program: labelProduct | lot id: L878766798556 | time: 1682588777375
> Program: labelProduct start | time: 1682588777274
```

Kuvio 17 Lokeja labelointi linjastolaitteeksi määritetystä linjastolaite-sovelluksesta

5 Tulokset

Työn tuloksena saatiin aikaan suunnitelmien mukainen digitaalinen tehdasautomaationlinjaston prototyyppi, jonka toiminnassa hyödynnettiin viestijonotekniikan tarjoamaa asynkronista viestin välitys ominaisuutta. Prototyyppi koostuu selaimen kautta toimivasta käyttöliittymästä, palvelimella toimivasta ohjainyksikkö-sovelluksesta, sekä eri palvelimille hajautetuista linjastolaite-sovelluksista. Horisontaalisen skaalauksen avulla pystyttiin tuottamaan monia rinnakkaisia tuotantolinjastoja digitaaliselle tehtaalle. Myös tutkimuskysymykseen saatiin selkeä vastaus. Viestijonotekniikka soveltuu erinomaisesti digitaalisen tehdasautomaatiolinjaston toiminnan mallintamiseen.

Kuvioissa 18 ja 19 esitetään toteutettu käyttöliittymä. Käyttöliittymässä on täysin dynaamiset visualisoinnit digitaalisen tehtaan kaikista eri tuotantolinjastoista. Visualisoinnin avulla käyttäjällä on mahdollisuus seurata eri materiaalivirtojen liikkuvuutta tuotantolinjastoilla, sekä mahdolliset tuotannon aikaiset ongelmatilanteet pystytään havainnoimaan nopeasti visualisoinnin ansiosta.



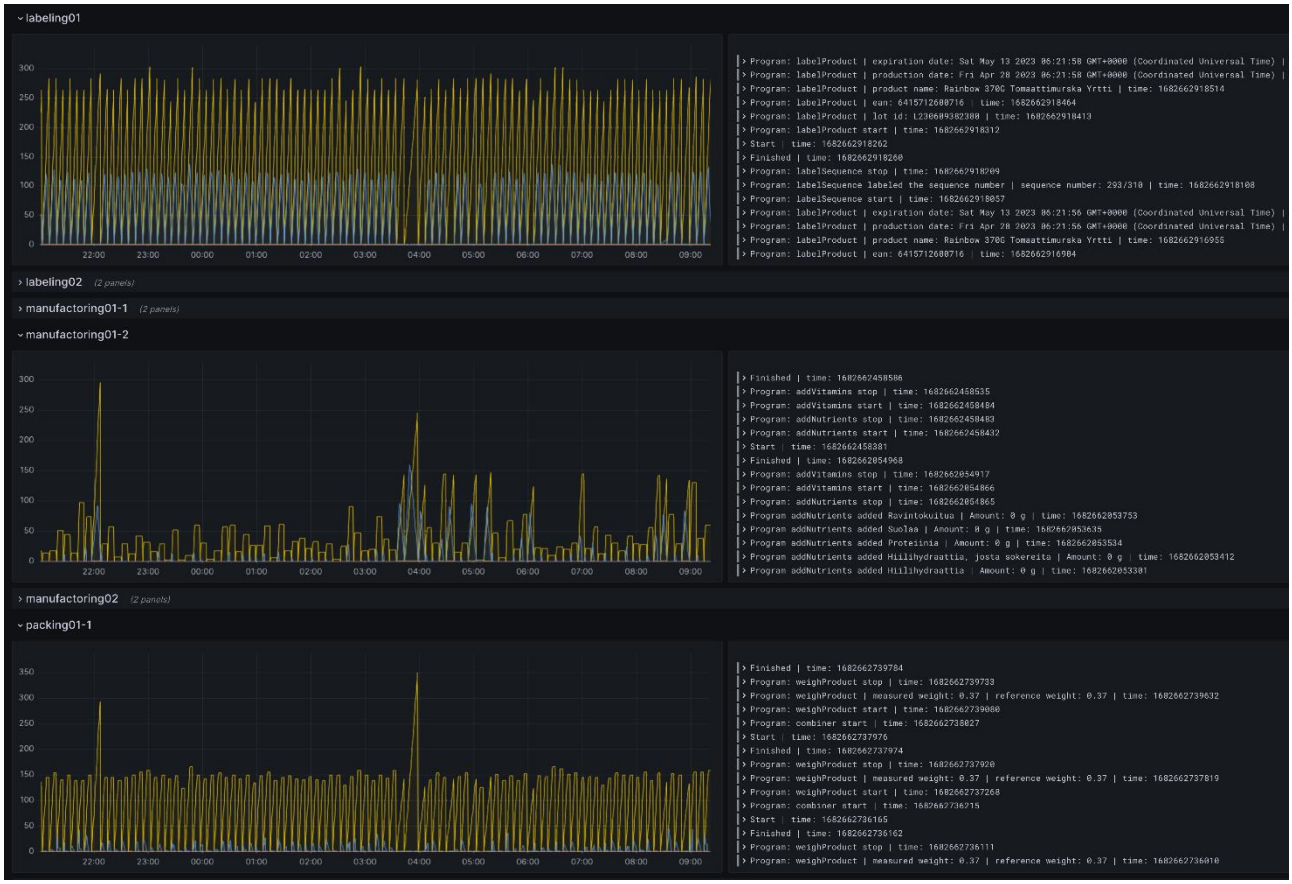
Kuvio 18 Yleisnäkymä tuotantolinjastoista käyttöliittymässä

Kuvio 19 Yksittäisen tuotantolinjaston perusnäkymä käyttöliittymässä

Ohjainyksikkö-sovelluksella pystytään keskitetysti hallinnoimaan digitaalisen tehtaan kaikkia tuotantolinjastoja. Ohjainyksikkö-sovellus kerää myös statistiikkaa linjastolaite-sovelluksien toiminnasta, jota sitten pystytään esittämään käyttäjälle erillisessä valvontanäkymässä. Ohjainyksikkö-sovellus integroitui myös osaksi digitaalisen tehtaan tuotannonohjausjärjestelmiä. Kuviossa 20 esitetään käyttöliittymässä oleva valvontanäkymä. Valvontanäkymässä käyttäjä pystyy tarvittaessa valvomaan kaikkia tuotantolinjastoja yhtä aikaa.

Kuvio 20 Tuotantolinjastojen valvontanäkymä käyttöliittymässä

Yksittäisen linjastolaite-sovelluksen toimintaan pystyttiin vaikuttamaan ohjainyksikkö-sovelluksen tekemän konfiguroinnin avulla. Konfigurointi asettaa linjastolaite-sovelluksille yksilölliset simulointiohjelmat, joita sovellus sitten suorittaa viestin saapuessa viestijonoon. Kuviossa 21 esitetään statistiikkaa, sekä lokia kolmesta eri linjastolaite-sovelluksesta, jotka on konfiguroitu tekemän eri prosesseja tuotantolinjastolla.



Kuvio 21 Kolmen eri linjastolaite-sovelluksen tuottamaa statistiikka ja lokia

Elintarvikeketjun kyberturvallisuus hankkeen pilottiharjoitus järjestettiin maaliskuussa 2023. Pilottiharjoituksessa testattiin hankkeen aikana suunniteltua ja rakennettua elintarviketuotannon kyberharjoitusympäristöä. Toteutettu prototyyppi oli osana tätä kyberharjoitusympäristöä. Pilottiharjoituksen jälkeen pystyttiin toteamaan, että kyseinen prototyyppi suoriutui hyvin tehtävästään.

6 Pohdinta

Opinnäytetyön tavoitteena oli saada toteutettua tehdasautomaatiolinjaston prototyyppi, elintarvikeketjun kyberturvallisuus hankkeen pilottiharjoitukseen. Asetettuun tutkimuskysymykseen, onko mahdollista mallintaa tehdasautomaatiolinjaston toimintaa käyttäen viestijonotekniikkaa, saatiin myös vastaus. Viestijonotekniikan käyttö soveltuu erinomaisesti tehdasautomaationlinjaston toi-

minnan mallintamiseen. Toteutetussa prototyypissä viestijonotekniikkaa hyödynnettiin muun muassa tuotantolinjastojen kuljettimien, sekä putkistojen mallintamisessa. Prototyyppi toimi hyvin koko pilottiharjoituksen ajan ja tuotti digitaalisia elintarvikkeita jalostajan varastoon ja siitä eteenpäin kauppojen käytettäväksi. Prototyypin toiminnan perusteella voidaan todeta, että opinnäytetyön tavoitteet saavutettiin. Toteutetulla prototyypillä saatiin myös tuotettua lisäarvoa elintarvikeketjun kyberturvallisuus hankkeelle, sekä sitä kautta myös pilottiharjoitukseen osallistuneille elintarviketuotannon yrityksille.

Toteutustyössä on huomioitava, että käytetty Bull.js-ohjelmakirjaston viestijonotekniikka on vain yksi monista käytettävissä olevista viestijonotekniikoista. Yleisemmin käytössä olevien MQTT- ja AMQP-protokollilla toimivien viestijonotekniikoiden käyttö olisi myös mahdollista. Molemmille protokollille löytyykin myös valmiit Node.js-kirjastot. Vaikka prototyypissä käytetty Bull.js-ohjelmakirjasto, sekä Redis-palvelin suoriutuikin tehtävästään hyvin, kannattaisi tulevaisuudessa tutkia mahdollisuutta viestijonotekniikan vaihtamiseen. Tämä siksi koska Bull.js-ohjelmakirjastoa käytettäessä voi tietyissä tilanteissa tapahtua viestien tupla prosessointeja, joista Bull.js github-sivustolla on maininta. Suositeltavaa olisi ainakin siirtyä käyttämään uudempaa Bullmq versiota Bull:ista.

Työn toteutuksen alkupuolella tehdyt suunnitelmat, sekvenssikaavioineen osoittautuivat tärkeiksi toteutuksen aikana. Niistä olikin monta kertaa apua, kun selvitettiin esimerkiksi, miten sovelluksien tulisi kommunikoida API-rajapintojen välityksellä toisilleen. Koska toteutettu ohjainyksikkösovellus liitettiin myös osaksi digitaalisen tehtaan toiminnanohjausjärjestelmiä, oli sekvenssikaavioiden suunnittelu näin laajassa ohjelmistoarkkitehtuurissa erityisen tärkeässä osassa.

Yhtenä isoimpana haasteena toteutustyössä oli saada kehitettyä linjastolaite-sovelluksesta niin monikäyttöinen, että sen avulla pystyttäisiin mallintamaan kaikki tarvittavat tuotantolinjastolaitteet. Tiettyjen simulointiohjelmien kehittämisessä oli myös aluksi vaikeuksia, mutta työn edetessä opituilla taidoilla päästiin alkuvaikeuksien ylitse ja saatiinkin kehitettyä kaikki tarvittavat simulointiohjelmat. Digitaalisen tehtaan tuotantolinjastojen ei ole pakko tuottaa pelkästään elintarvikkeita, vaan tuotettavia tuotteita rajoittaa vain simulointiohjelmat. Toteutettuun linjastolaite-sovellukseen pystytäänkin lisäämään uusia simulointiohjelmia helposti, jos sitä lähdetään jatkokehittämään.

Yhtenä jatkokehitys ideana toteutustyölle on Modbus TCP/IP-protokollan käyttö ohjainyksikkö-sovelluksen ja linjastolaite-sovelluksien välillä. Toteutetussa prototyypissä näiden sovelluksien välinen kommunikointi tapahtuu kyllä samojen tietoliikenneporttien kautta mitä Modbus TCP/IP-protokollakin käyttää, mutta itse data oli JSON-muodossa. Node.js-ympäristöön löytyy myös valmiita Modbus TCP/IP-ohjelmakirjastoja, joita pystyttäisiin tarvittaessa hyödyntämään jatkokehityksessä. Toisena kehitys ideana on toteuttaa erillinen sovellus, millä pystyttäisiin graafisen käyttöliittymän kautta suunnittelemaan tuotantolinjastoja. Tämän avulla monimutkaisienkin tuotantolinjastojen suunnittelu helpottuisi ja linjastonrakenteen näkisi suoraan käyttöliittymästä. Sovelluksen avulla pystyttäisiin myös saamaan valmis scadaConfig.json tiedosto, jota ohjainyksikkö-sovellus käyttää tuotantolinjastojen muodostamisessa. Näin myös vältyttäisiin mahdollisilta virheiltä kyseisen tiedoston tekemisessä.

Lähteet

Bull. 2021. Virallinen Bull repositorio github sivustolla. Viitattu 01.04.2023. https://github.com/OptimalBits/bull

Elintarvikeketjun kyberturvallisuus. 2021. Hankkeen kuvaus JAMK:in sivustolla. Viitattu 01.03.2023. https://www.jamk.fi/fi/tutkimus-ja-kehitys/tki-projektit/elintarvikeketjun-kyberturvallisuus

How SCADA, HMI, and PLC Work Together. 2019. Blogikirjoitus SCADA, HMI ja PLC laitteiden toiminnasta. Viitattu 20.04.2023. https://www.telstarinc.com/how-scada-hmi-and-plc-work-together

Introduction - Vue.js. N.d. Vue.js teknologian esittely sivusto. Viitattu 22.04.2023. https://vuejs.org/guide/introduction.html

Introduction to hapi.js Framework. 2020. Artikkeli Hapi-ohjelmistokehyksestä. Viitattu 21.04.2023. https://www.section.io/engineering-education/introduction-to-hapi

Introduction to message queuing. N.d. Johdatus viestijonoon. Muokattu 27.01.2023. Viitattu 01.03.2023. https://www.ibm.com/docs/en/ibm-mq/9.2?topic=overview-introduction-message-queuing

Introduction to Node.js. N.d. Node.js-teknologian esittely sivusto. Viitattu 25.04.2023. https://nodejs.dev/en/learn

Introduction to Redis. N.d. Redis-teknologian esittely sivusto. Viitattu 22.04.2023. https://redis.io/docs/about

Kyberturvallisuus vaatii jatkuvaa työtä. 2022. Artikkeli huoltovarmuuskeskus sivustolla. Viitattu 27.02.2023. https://www.huoltovarmuuskeskus.fi/a/kyberturvallisuus-vaatii-jatkuvaa-tyota

Realistic Global Cyber Environment - Overview. 2022. RGCE yleiskuvaus JYVSECTEC:in sivustolla. Viitattu 01.03.2023. https://jyvsectec.fi/cyber-range/overview

Realistic Global Cyber Environment - Technical details. 2022. RGCE tekninen kuvaus JYVSECTEC:in sivustolla. Viitattu 01.04.2023. https://jyvsectec.fi/cyber-range/technical

Salvador, L. Dai, N. & Zoltán, R. 2023. SCADA Systems: Security Concerns and Countermeasures. Viitattu 19.04.2023 https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/10044495

Tietoturvatunnustukset 2022. 2022. Vuoden tietoturvapalvelu tunnustukset. Viitattu 25.04.2023. https://tietoturva.fi/2022/10/14/tietoturvatunnustukset-2022

Toikko, T. & Rantanen, T. 2009. Tutkimuksellinen kehittämistoiminta. Tampere: Tampereen yliopistopaino oy. Viitattu 31.03.2022. https://urn.fi/URN:ISBN:978-951-44-7732-4

Toimialojen kyberkypsyyden selvitys 2022. 2023. Huoltovarmuuskeskuksen kansallinen koostera-portti. Viitattu 17.04.2022. https://www.huoltovarmuuskeskus.fi/files/29b11d0af56a115126ad490af444f1c4fd7885af/hvk-toimialojen-kyberkypsyyden-selvitys-2022.pdf

Urooj, B. Ullah, U. Shah, M. Sikandar, H & Stanikzai A. 2022. Risk Assessment of SCADA Cyber Attack Methods: A Technical Review on Securing Automated Real-time SCADA Systems. Viitattu 20.04.2023. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/9911122

What is Docker?. N.d. Docker-teknologian esittely artikkeli. Viitattu 20.04.2023. https://www.ibm.com/topics/docker

What is InfluxDB. 2023. InfluxDB-teknologian esittely sivusto. Viitattu 24.04.2023. https://www.influxdata.com/time-series-platform

What is a SCADA System. N.d. Tekninen kuvaus SCADA:n toiminnasta. Viitattu 20.04.2023. https://www.elprocus.com/scada-system-architecture-its-working

What is a SCADA System and How Does It Work?. 2022. SCADA toimintaympäristön kuvaus blogisivusto. Viitattu 01.04.2023. https://www.onlogic.com/company/io-hub/what-is-a-scada-system-and-how-does-it-work

What's a Message Queue. 2022. How It Can Simplify IT Infrastructure. 2022. Kuvaus viestijonoista. Viitattu 27.03.2023. https://www.g2.com/articles/message-queue-mq

Why You Should Use Typescript for Your Next Project. 2022. Artikkeli TypeScript:in hyödyistä. Viitattu 25.04.2023. https://www.codemotion.com/magazine/backend/why-you-should-use-typescript-for-your-next-project/s

Liitteet

Liite 1. scadaConfig.json

```
{
"apparatus" : [
    {
      "ip": "172.23.0.2",
      "name": "processing01",
      "program": [
        { "name": "startProcess" },
        { "name": "gatherMaterial" }
      ],
      "next": [
        {
          "name": "manufactoring01-1",
          "disabled": false,
          "weight": 1
        },
        {
          "name": "manufactoring01-2",
          "disabled": false,
          "weight": 2
        }
      ]
    },
    {
      "ip": "172.23.0.3",
      "name": "manufactoring01-1",
      "program": [
        { "name": "addNutrients" },
        { "name": "addVitamins" }
      ],
      "next": [
        {
          "name": "packing01",
          "disabled": false,
          "weight": 1
        }
      ]
    },
    {
      "ip": "172.23.0.4",
      "name": "manufactoring01-2",
      "program": [
        { "name": "addNutrients" },
        { "name": "addVitamins" }
      ],
      "next": [
        {
          "name": "packing01",
          "disabled": false,
          "weight": 1
        }
      ]
```

```
},
{
  "ip": "172.23.0.5",
  "name": "packing01",
  "program": [
    { "name": "combiner" },
    { "name": "weighProduct" }
  ],
  "next": [
    {
      "name": "labeling01",
      "disabled": false,
      "weight": 1
    }
  ]
},
{
  "ip": "172.23.0.6",
  "name": "labeling01",
  "program": [
    { "name": "labelProduct" },
    { "name": "labelSequence" }
  ],
  "next": [
    {
      "name": "storage01",
      "disabled": false,
      "weight": 1
    }
  ]
},
{
  "ip": "172.23.0.7",
  "name": "storage01",
  "program": [
    { "name": "qualityControl" },
    { "name": "sendToStorage" }
  ],
  "next" : []
},
{
  "ip": "172.23.0.8",
  "name": "processing02",
  "program": [
    { "name": "startProcess" },
    { "name": "gatherMaterial" }
  ],
  "next": [
    {
      "name": "manufactoring02",
      "disabled": false,
      "weight": 1
    }
  ]
},
{
  "ip": "172.23.0.9",
```

```
      "name": "manufactoring02",
      "program": [
        { "name": "addNutrients" },
        { "name": "addVitamins" }
      ],
      "next": [
        {
          "name": "packing02",
          "disabled": false,
          "weight": 1
        }
      ]
    },
    {
      "ip": "172.23.0.10",
      "name": "packing02",
      "program": [
        { "name": "combiner" },
        { "name": "qualityControl" }
      ],
      "next": [
        {
          "name": "labeling02",
          "disabled": false,
          "weight": 1
        }
      ]
    },
    {
      "ip": "172.23.0.11",
      "name": "labeling02",
      "program": [
        { "name": "labelProduct" },
        { "name": "labelSequence" }
      ],
      "next": [
        {
          "name": "storage02",
          "disabled": false,
          "weight": 1
        }
      ]
    },
    {
      "ip": "172.23.0.12",
      "name": "storage02",
      "program": [
        { "name": "qualityControl" },
        { "name": "sendToStorage" }
      ],
      "next" : []
    }
  ]
}
```