ASE-6030 Automaation reaaliaikajärjestelmät

Sellunkeittoprosessin ohjaussovellus - SEPOS

Versio 0.1

Janne Lahdentausta – H291524
Olli Laaksonen – H291628

Dokumentin tila: työversio Muokattu: 8.128.12.2024

SISÄLLYSLUETTELO

1 JOHDANTO

Tämä on sellunkeittoprosessin ohjaussovelluksen (SEPOS) suunnittelu- ja implementointiprosessia kuvaava työdokumentti. Dokumenttiin sisältyy sovelluksen vaatimusmäärittely, suunnittelu sekä toteutuskappaleet.

SEPOS on operaattorille tarkoitettu yksinkertainen sovellus, joka mahdollistaa sellunkeittoprosessin monitoroinnin sekä tärkeimpien toimintojen toteuttamisen yhdessä näkymässä. Lisäksi sovellus mahdollistaa keittoprosessin parametroinnin ja tarjoaa vikadiagnostiikkaa virhetilanteissa.

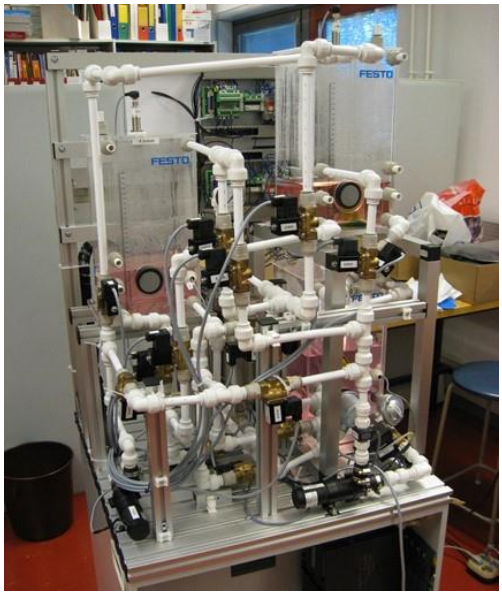
2 VAATIMUSMÄÄRITTELY

2.1 Vaatimukset

Ohjelmiston vaatimusmäärittely on dokumentti, joka sisältää kehitettävän ratkaisun vaatimusten kartoittamisen, dokumentoinnin ja analysoinnin. Sen tehtävänä on siis toisin sanoen kertoa, mitä ratkaisulla tehdään ja mitä sillä pyritään saavuttamaan. Vaatimuksia voidaan tarvittaessa luokitella: esim. toiminnallisina ja ei-toiminnallisina vaatimuksina. Käytetään tässä työssä tätä luokittelutapaa.

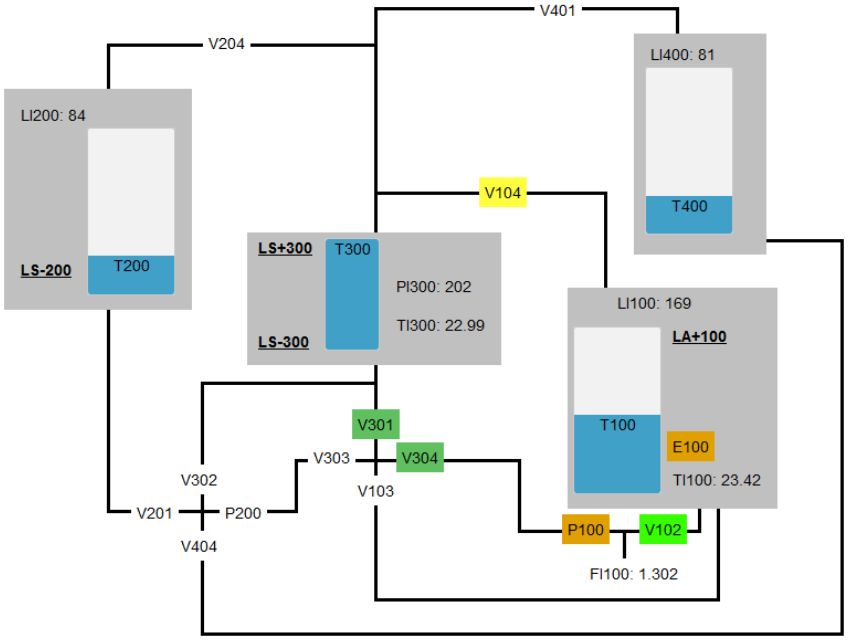
2.1.1 Sovelluksen yleiskuvaus

Sovellus ohjaa sellunkeittoprosessia, ja se hallitsee kyseistä prosessia laitteiston, kuten säiliöiden, venttiilien ja pumppujen avulla. Sovellus ja prosessilaitteisto kommunikoivat OPC UA -protokollalla, jota käytetään tiedonsiirtoon nykyaikaisissa teollisuusympäristöissä. Tampereen yliopistolla on pienoismalli sellulaitoksesta, josta on kuva alla



Sellu on paperimassaa, joka valmistetaan yleensä puuhakkeesta kemiallisella massanvalmistusmenetelmällä. Tarkemmin tarpeeksi hieno puumassa, yleensä hake, sekoitetaan sellukattilassa emäksiseen liuokseen, valkolipeään, ja keitetään korkeassa lämpötilassa. Tällöin puun sisältämää ja kuituja sitovaa polymeeriä, ligniiniä, pystytään poistamaan prosessissa tarpeeksi ja massaan syntyy selluloosakuituja.

Oikean pienoismallin sijaan, tämä työ toteutetaan virtualisoidun prosessin kanssa, joka kuvaa sellunkeittoprosessia. Simulaattorisovelluksen käyttöliittymästä on kuva alla



2.1.2 Toiminnalliset vaatimukset

Toiminnallinen vaatimus on ilmoitus siitä, kuinka järjestelmän tulee käyttäytyä. Se määrittelee, mitä järjestelmän tulee tehdä vastatakseen käyttäjän tarpeisiin tai odotuksiin. Tämän sovelluksen toiminnalliset vaatimukset on listattu alla

1. Sovellus toteutetaan annetun PFC:n (Procedure Function Chart) mukaisesti
2. Käyttäjä voi käynnistää, keskeyttää ja palauttaa prosessin alkutilaan (säiliöiden nestemäärät säilyvät). Toisinsanottuna käyttäjä voi uudelleen suorittaa tai keskeyttää ylempänä selitetyn sekvenssin
3. Käyttöliittymässä tulee näkyä sekä graafisesti että numeroina säiliöiden T100, T200, T400 pinnankorkeus ja säiliön T300 paine ja lämpötila
4. Käyttäjä voi muokata käyttöliittymässä sekvenssin parametreja: keittoaikaa, keittopainetta, keittolämpötilaa ja kyllästysaikaa
5. Edellä mainittuja parametreja ei voi muokata suorituksen aikana
6. Jos verkkoyhteys laitteistoon katkeaa, käyttäjä voi yrittää sen avaamista uudelleen

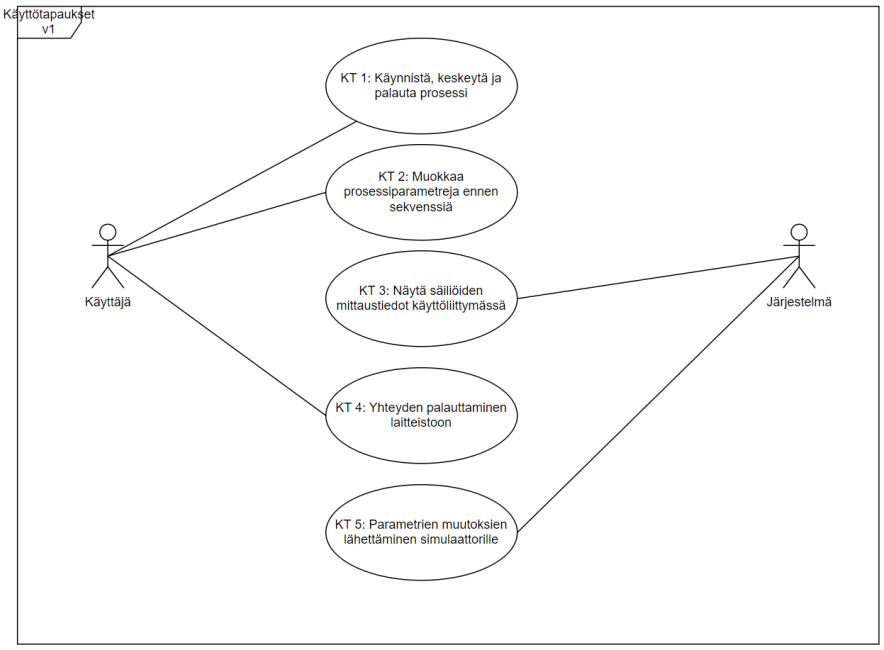
2.1.3 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset selittävät suunniteltavan järjestelmän rajoitukset ja rajoitteet. Näillä vaatimuksilla ei ole vaikutusta sovelluksen toimivuuteen. Tämän sovelluksen ei-toiminnalliset vaatimukset ovat:

1. Lämpötila saa poiketa enintään 0,3 Celsius- astetta asetusarvosta keiton aikana. (Tarkkuus- /suorituskykyvaatimus)
2. Paine saa poiketa enintään 10 hPa asetusarvosta keiton aikana. (Tarkkuus-/ suorituskykyvaatimus)
3. Sovellus ei kaadu, vaikka verkkoyhteyden kanssa olisi ongelmia. (Luotettavuusvaatimus)

2.2 Käyttötapaukset

Sovelluksen käyttötapauksien havainnollistamiseen käytetään UML-pohjaista käyttötapauskaaviota:



Kuva 1: Käyttötapauskaavio

2.2.1 KT 1: Käynnistä, keskeytä ja palauta prosessi

Suorittajat: Käyttäjä

Esiehdot: Prosessi on aloitettu ja laitteet ovat valmiustilassa.

Kuvaus: Käyttäjä voi käynnistää, keskeyttää ja palauttaa prosessin alkutilaan. Prosessin hallinta säilyttää nykyiset nestemäärät säiliöissä, joten prosessin voi uudelleen

käynnistää samasta tilasta ilman tietojen menetystä.

Poikkeukset:
(1) Prosessia ei voida käynnistää, koska laitteisto ei ole vielä valmis. Järjestelmä näyttää virheilmoituksen käyttäjälle
(2) Prosessin keskeyttäminen epäonnistuu (esimerkiksi yhteyden katkeamisen vuosi). Järjestelmä ilmoittaa, että prosessia ei voida keskeyttää turvallisesti.

Tavoite: Käyttäjä voi hallita prosessia sujuvasti käynnistämällä ja keskeyttämällä prosessin tarvittaessa.

Lopputulos: Prosessi pysäytetään tai käynnistetään uudelleen keskeytyskohdasta.

2.2.2 KT 2: Muokkaa prosessiparametreja ennen sekvenssiä

Suorittajat: Käyttäjä

Esiehdot: Prosessi ei ole käynnissä.

Kuvaus: Käyttäjä voi muokata sellunkeittoprosessin parametreja, kuten keittoaikaa, painetta, lämpötilaa ja kyllästysaikaa, ennen prosessin käynnistämistä.

Poikkeukset:
(1) Käyttäjä yrittää muokata parametreja prosessin ollessa käynnissä. Järjestelmä estää muutokset ja näyttää virheilmoituksen, että muutokset ovat mahdollisia vain ennen prosessin käynnistämistä.
(2) Käyttäjä syöttää virheellisen parametrin, (esim liian korkean lämpötilan). Järjestelmä ilmoittaa siitä ja pyytää korjausta tai asettaa järkevän arvon itse.

Tavoite: Parametrien optimointi ennen prosessin aloittamista.

Lopputulos: Muokatut parametrit tallennetaan ja otetaan käyttöön seuraavassa suorituksessa.

2.2.3 KT 3: Näytä säiliöiden mittaustiedot

Suorittajat: Käyttäjä

Esiehdot: Laitteiston ja sovelluksen välinen yhteys on aktiivinen.

Kuvaus: Käyttäjä näkee käyttöliittymässä säiliöiden pinnankorkeuden (T100, T200, T400) sekä säiliön T300 paineen ja lämpötilan graafisesti ja numerotiedoin.

Poikkeukset:
(1) Mittaustietoja ei saada laitteistosta, esim kun verkkoyhteys katkeaa. Järjestelmä näyttää käyttäjälle virheilmoituksen ja esittää viimeksi saadut mittaustiedot harmaalla.
(2) Mittauslaite on epäkunnossa. Järjestelmä ilmoittaa mittausvirheestä ja näyttää oletusarvon tai keskeyttää mittausten päivittämisen tai alaa ajas järjestelmän.

Tavoite: Reaaliaikainen mittaustietojen esittäminen käyttäjälle.

Lopputulos: Käyttäjä saa ajantasaiset tiedot säiliöiden tilasta.

2.2.4 KT 4: Yhteyden palauttaminen laitteistoon

Suorittajat: Käyttäjä

Esiehdot: Yhteys laitteistoon on katkennut.

Kuvaus: Jos sovelluksen ja laitteiston välinen yhteys katkeaa, käyttäjä voi yrittää palauttaa yhteyden.

Poikkeukset:
(1) Yhteyden palauttaminen epäonnistuu. Järjestelmä näyttää virheilmoituksen ja ehdottaa käyttäjälle toimenpiteitä. (Tarkista verkkoyhteys tai yritä myöhemmin uudelleen)
(2) Yhteys palautuu mutta tiedonsiirrossa on häiriöitä. Järjestelmä ilmoittaa tästä ja pyytää korjausta.

Tavoite: Yhteyden palauttaminen laitteiston ja sovelluksen välillä.

Lopputulos: Yhteys palautuu, tai käyttäjä saa virheilmoituksen.

2.2.5 KT 5: Lähetä prosessiparametrit laitteistolle

Suorittajat: Käyttäjä, laitteisto

Esiehdot: Parametrit on muokattu käyttöliittymän kautta ja prosessi ei ole käynnissä.

Kuvaus: Kun käyttäjä on muokannut prosessiparametreja (keittoaika, paine, lämpötila, kyllästysaika), sovellus lähettää nämä muutokset laitteistolle ennen prosessin käynnistämistä.

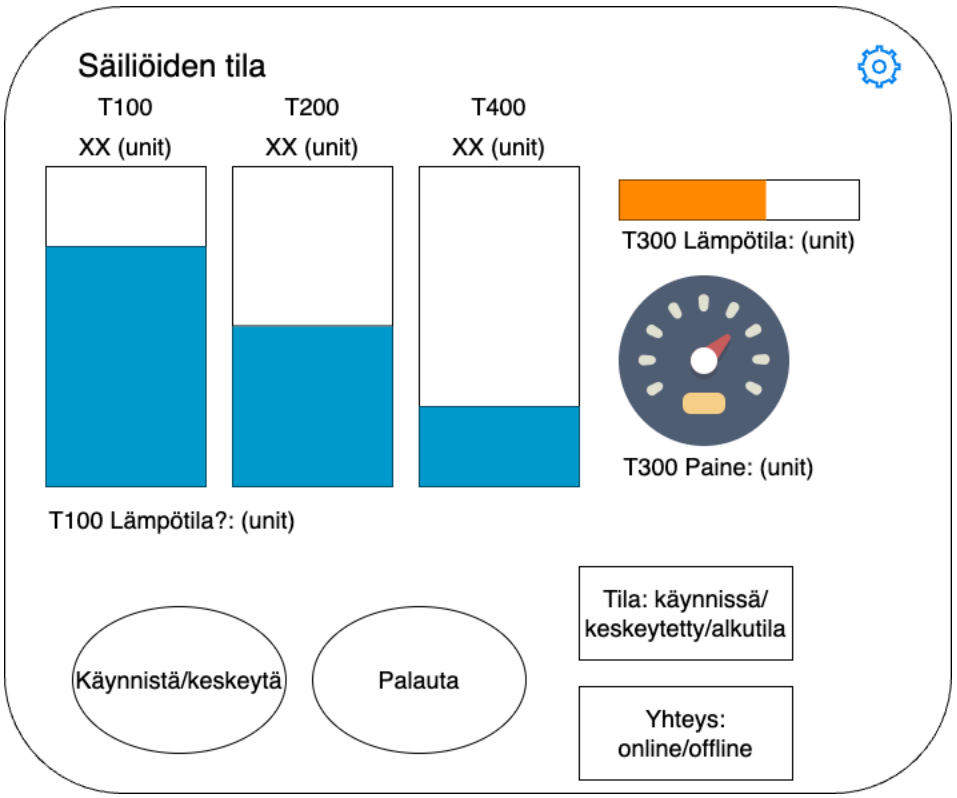
Poikkeukset:
(1) Parametrien lähettäminen epäonnistuu. Järjestelmä ilmoittaa virheestä ja yrittää lähettää parametrit uudelleen, kun yhteys palautuu
(2) Jokin parametri on virheellinen. Järjestelmä ilmoittaa virheestä ja estää virheellisten tietojen lähettämisen.

Tavoite: Varmistaa, että muokatut parametrit otetaan käyttöön ja välitetään laitteistolle oikeaan aikaan.

Lopputulos: Muokatut parametrit tallennetaan ja lähetetään laitteistolle ennen prosessin alkua.

2.3 Käyttöliittymähahmotelmat

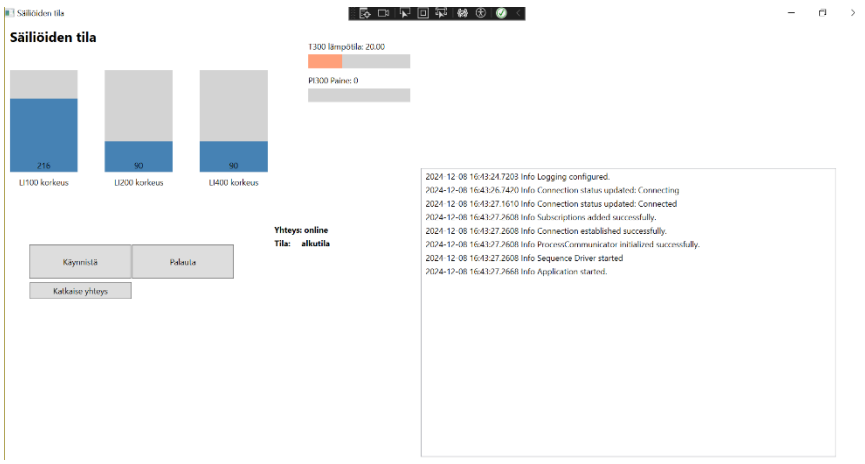
Käyttöliittymä koostuu pääkäyttöliittymästä, diagnostiikasta ja parametrien asetusikkunasta. Käyttöliittymää tukee eri virheilmoitusikkunat. Pääkäyttöliittymässä on esillä asiakasvaatimusten mukaan annetut tilatiedot säiliöiden pinnankorkeudesta sekä T300 säiliön lämpötila ja paine.



Kuva 2: Pääkäyttöliittymän hahmotelma

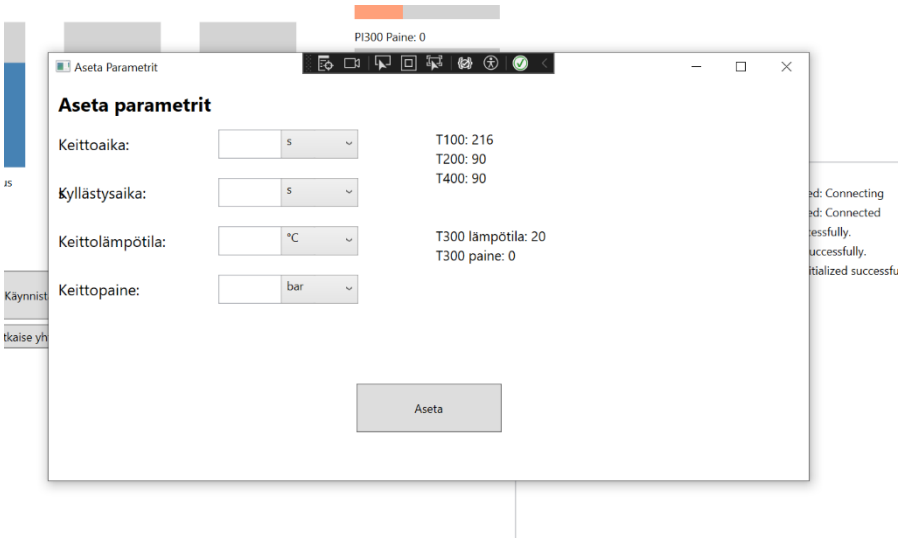
Kuvassa 2 pyöreät elementit ovat interaktiivisia, joista käyttäjä voi käynnistää, keskeyttää sekä palauttaa järjestelmän alkutilaan pinnankorkeuksia lukuun ottamatta. Pääkäyttöliittymällä on kolme eri tilaa, riippuen onko sekvenssi aloitettu vai ei ja onko järjestelmä alkutilassa.

Sekvenssin ollessa käynnissä interaktiiviset elementit antavat käyttäjälle vaihtoehdot KESKEYTÄ ja PALAUTA. Järjestelmän ollessa valmiustilassa annetut vaihtoehdot ovat KÄYNNISTÄ ja PALAUTA.



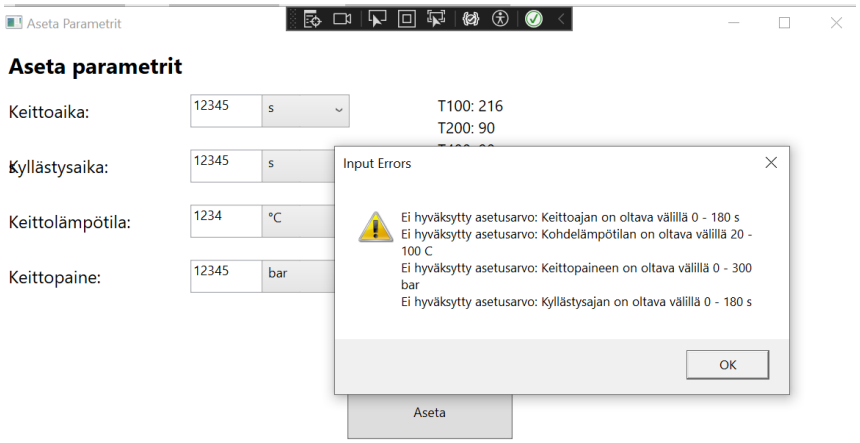
Kuva 3: Päänäkymä

Kuvassa 3 on esillä valmis päänäkymä käyttöliittymälle. Tässä on selkeästi näkyvillä myös vaatimusten mukainen diagnostiikkalaatikko, johon sovellus tulostaa tilatietoja sen eri suoritusvaiheissa ja on aina saatavilla. Lisäksi yhteyttä varten on laitettu erillinen nappula havainnollisena elementtinä. Käyttäjä saa tietoja myös virhetilanteessa, ja esimerkiksi yhteyden menettäessä käyttäjä voi painaa MUODOSTA YHTEYS –napista muodostaakseen yhteyden. Sovellus ei kaadu yhteyden menetettyään, mutta järjestelmään on saatava yhteys, jotta sekvenssejä voidaan ajaa.



Kuva 4: Parametrien asetusnäkymä

Aina käyttäjän painaessa KÄYNNISTÄ-elementtiä ja järjestelmän olevan alkutilassa, kuvan 4 mukainen parametrien asetusnäkymä tulee näkyviin. Käyttäjän tulee asettaa jokaiseen tekstikenttään hyväksyttävä arvo tietyllä alueella ja käyttäjän yrittäessä asettaa virheellisiä arvoja virheilmoitus aktivoituu ja kertoo käyttäjälle mitkä parametrit täytyy korjata kuten näkyy kuvassa 5.



Kuva 5: Esimerkki virheilmoituksesta

Jatkossa käydään vielä asiakkaan kanssa keskustelua sallittujen arvojen välistä ja tätä voidaan tarvittaessa muuttaa tarpeen mukaan.

3 SUUNNITTELU

Tässä luvussa on dokumentoituna varsinainen sovellussuunnittelu.

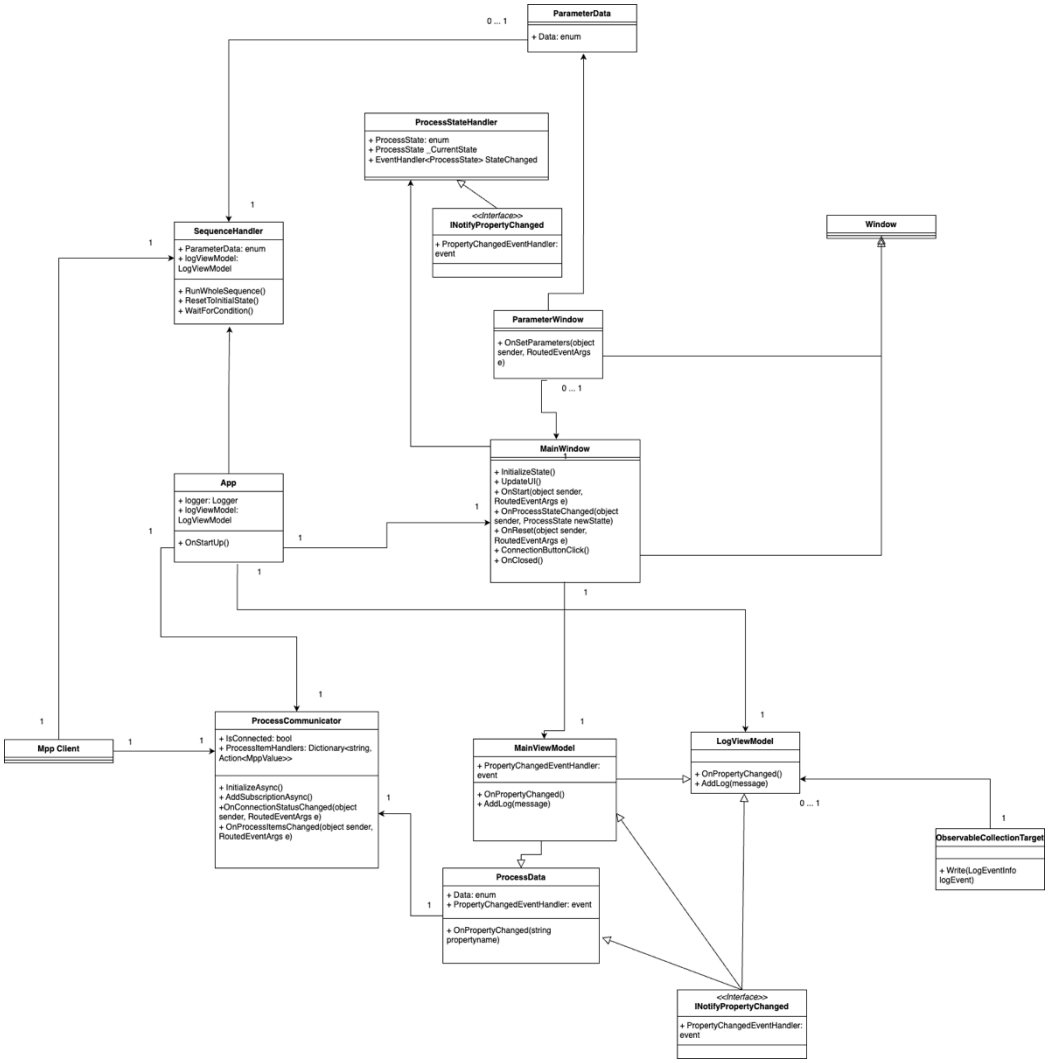
3.1 Sovellusarkkitehtuuri

Sovelluksen yksinkertaisen luonteen vuoksi ohjeellisena arkkitehtuurin mallina käytettiin ohjelmoinnista tuttua “best practices” tapaa jakaa ohjelma modulaarisiin osiin, joista jokainen osa toteuttaa aina yhden käyttötarkoituksen, ja osaohjelmat ovat kapseloituina omien rajapintojensa taakse. Toisin sanoen sovellus toteuttaa osittain SOLID suunnittelumallia, joskaan ei tiukasti käyttökohteen ollessa tässä tapauksessa spesifi ja rajattu.

3.2 Rakenne

Tässä dokumentissa ja luokkakaaviossa käydään sovelluksen toiminta läpi ylätasolla ja luokkakaaviossa etenkin erillisten datarakenteiden sisältöä voi tarkemmin tarkastella liitteenä olevasta doxygen dokumentaatiosta.

Sovelluksen sisääntulopisteenä toimii App-luokka, joka alustaa kaikki luokat, joiden täytyy olla saatavilla koko sovelluksen toiminta alueella. Nämä ovat pääikkuna (MainWindow), palvelimen kanssa kommunikoiva ProcessCommunicator, sekvenssien ajamisesta vastuussa oleva SequenceHandler sekä lokien kirjaamiseen käytetty LogViewModel, koska se on vastuussa myös App-luokan lokien kirjoittamisesta.



Kuva 62. Luokkakaavio.

Aloitetaan Appin jälkeen pääikkunan toiminnasta. Pääikkuna ylläpitää sovelluksen tilaa sekä on vastuussa sen näyttämisestä käyttäjälle. Samoin metodit eri nappeja painaessa sekä metodit tilamuutoksia varten ajetaan pääikkunassa. Pääikkuna hyödyntää ProcessStateHandler –luokkaa päivittääkseen oman tilansa ja ProcessStateHandler paljastaa sekvenssien ajajalle (SequenceHandler) tilatiedot, joilla sekvenssiajuri voi välittää prosessin tilan ProcessStatehandlerin kautta pääikkunalle.

Pääikkuna myös avaa parametrien asetusikkunan, jonka tiedot tallennetaan ParameterData –luokkaan ja joka edelleen annetaan sekvenssiajurille eteenpäin ja sekvenssi ajetaan annettujen tietojen mukaan. ParameterDatan sisältöä voi tarkastella doxygen dokumentaatiosta. Siinä näkyy annettujen parametrien tyypit ja nimet.

Seuraavana ylää mainitty SequenceHandler, eli sekvenssien ajaja, jossa on toteutettuna kaikki ajettavat sekvenssit. Sekvenssien määrän vuoksi luokkakaaviossa on esillä vain RunWholeSequence(), joka ajaa koko sellunkeittoprosesssin kaikki

sekvenssit annetun PFC:n mukaan. Doxygen dokumentaatiosta voi käydä läpi kaikki sekvenssit ja niihin liittyvät operaatiot.

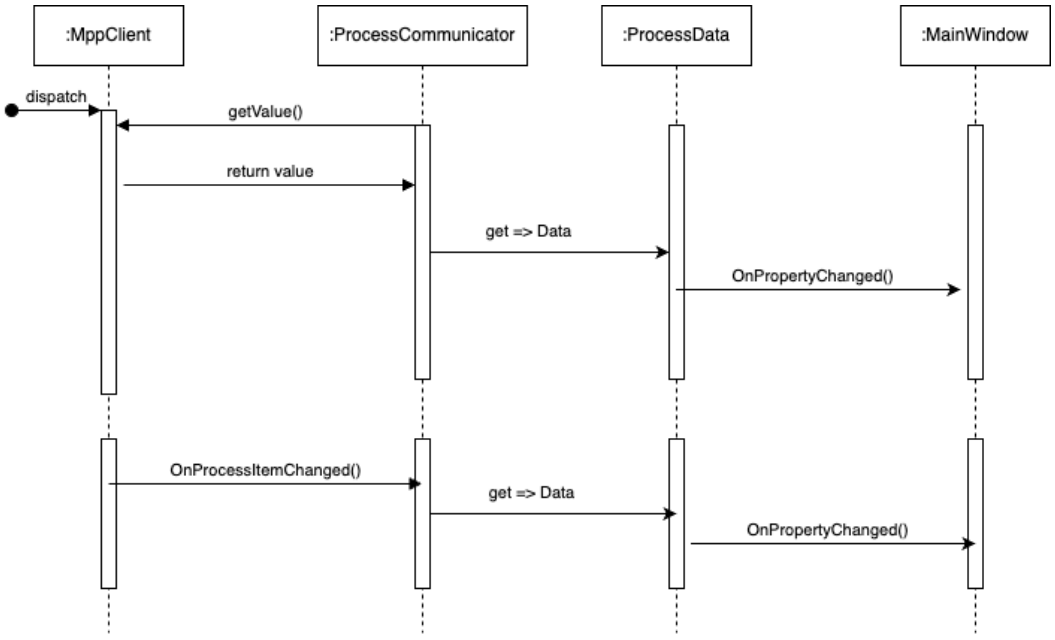
Prosessikommunikaattori (ProcessCommunicator) vastaa pääasiassa yhteyden ylläpidosta ja alustamisessa ottaa palvelimelta subscriptionit kaikkiin haluttuihin arvoihin (ventiilit, pinnankorkeudet, yms.), tekee niille handlerit ja antaa ne eteenpäin ProcessData –luokalle. ProcessDatassa on talletettuna prosessitiedot, jotka luovat tapahtumia arvojen muuttuessa. Kaikki talletetut arvot ovat eriteltynä doxygen dokumentaatiossa.

Kaikki tiedot pääikkunaan koostetaan yhteen datakontekstiin ohjelman rajoitteiden takia käyttämällä komposiittimallia MainViewModel, joka yhdistää ProcessDatan ja LogViewModelin yhdeksi malliksi. Päänäkymä ja parametrinäkymä voivat sitten sitoutua tähän yhteen malliin ja näyttää prosessidataa ja lokitietoja samanaikaisesti.

LogViewModel ja ObservableCollectionTarget ovat osana lokitietojen ylläpitoa ja hyödyntävät valmiina olevaa NLog –lokitetojen hallintaohjelmaa. ObservableCollectionTarget on siis vain osana tätä pakettia mahdollistamassa lokitetojen siistin koostamisen.

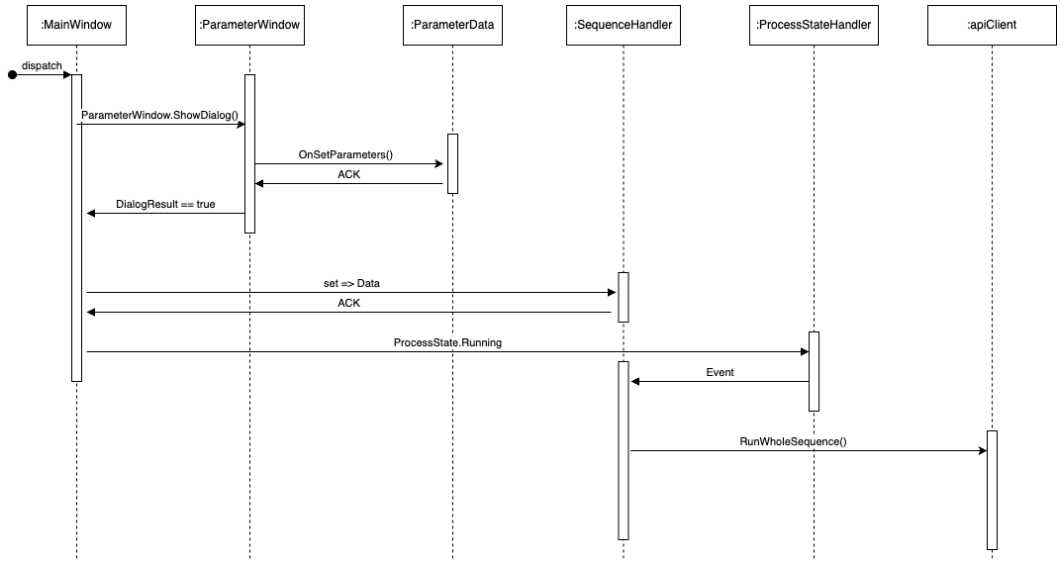
3.3 Toiminta ja tilat

Tässä luvussa käydään läpi sovelluksen perustoimintoja sekvenssikaavioiden avulla ja käydään läpi lyhyesti käyttöliittymän tilakaavio.



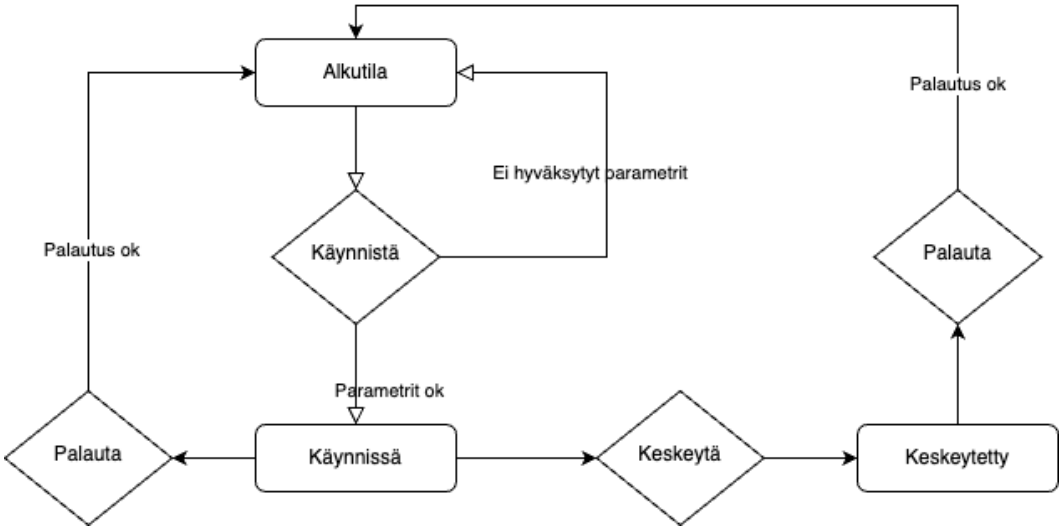
Kuva 73. Geneerisen monitorointitiedon haku

Kuvassa 7 on kuvattuna tiedonhaku aina OPC UA päätteeltä käyttöliittymään asti. Erityisesti kuvasta voi huomata luokan ProcessData roolin, joka mahdollistaa tietojen päivityksen käyttöliittymälle.



Kuva 84. Parametrien asetus

Kuvassa 8 on esitettynä parametrien asetus käyttäjän toimesta ja kuinka tieto välittyy järjestelmässä sekvenssin ajajalle (SequenceHandler), joka lopulta aloittaa keskustelun palvelimen kanssa ja alkaa ajamaan sekvenssiä.



Kuva 95. Pääkäyttöliitymän tilakaavio

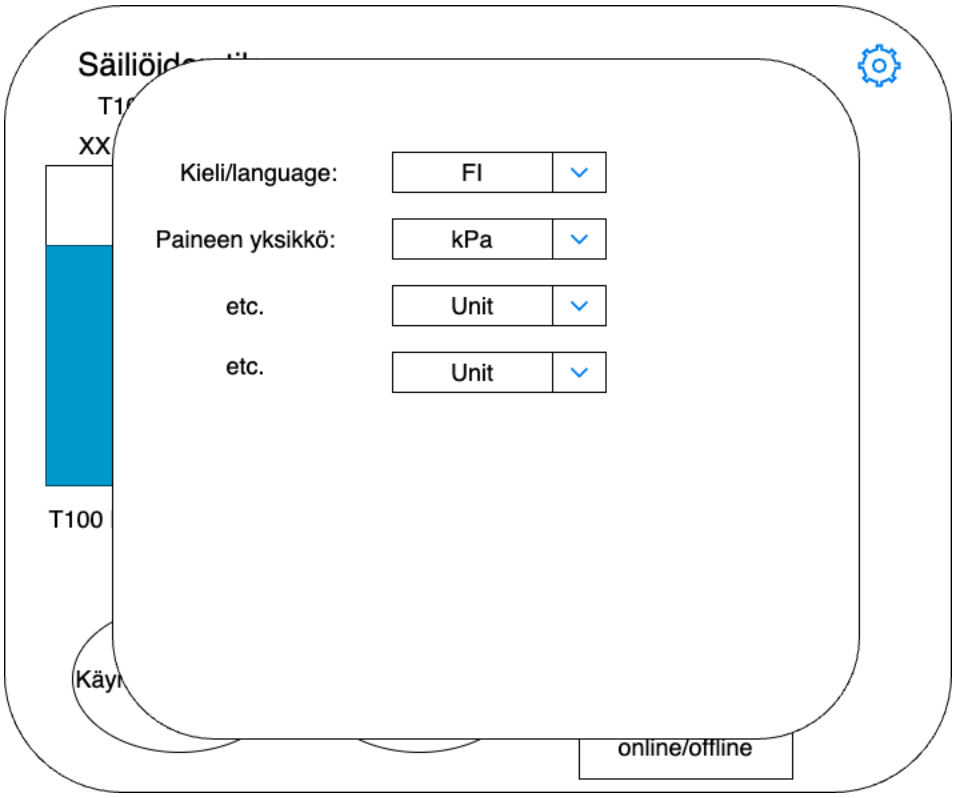
Kuvassa 9 on esitettynä pääkäyttöliittymän tilat niiden yksinkertasimmassa muidossa. Jokaisesta tilasta voidaan siirtyä erilliseen virhetilaan, jonka selvittyä prosessi täytyy palauttaa alkutilaan.

Käyttöliittymä lähtee alkutilasta ja käyttäjä voi valita prosessin aloittamisen “Käynnistä” napista. Palauta napista ei tapahdu mitään alkutilassa. Jos

parametrien asetus ei onnistu, näytetään virheilmoitus ja käyttäjä voi yrittää uudestaan. Jos ikkuna suljetaan, palataan alkutilaan.

Onnistuttaessa siirrytään tilaan käynnissä, josta käyttäjä voi valita prosessin keskeytyksen tai palauttamisen. Keskeyttäessä siirrytään tilaan keskeytetty ja palauttaessa siirrytään alkutilaan. Keskeytetty tilasta siirrytään takaisin alkutilaan painamalla palauta.

3.4 Hylätyt ratkaisuvaihtoehdot



Kuva 10. Asetusvalikko

Työssä päätettiin toistaiseksi olla tukematta kielen ja yksiköiden vaihtoa sen takia, kun sen ei koettu antavan lisäarvoa ja se saattaisi aiheuttaa turhaa hämmennystä käyttäjälle.

Käyttöliittymä tarjoaa käyttäjälle mahdollisuuden asetusvalikon kautta vaihtaa kielen sekä käytetyt mittayksiköt. Asetusvalikkoon pääsee kaikissa näkymissä oikeassa yläkulmassa olevan ratassymbolin kautta.

4 TOTEUTUS

4.1 Kehitysympäristö

Itse ohjelmointiympäristönä käytettiin Microsoft Visual Studiota (useampi versio, 2022 sekä 2024). Ohjelmointikielenä käytettiin C#:a ja kehitysympäristönä oli .NET versio 4.809037 vanhojen kirjastojen yhteensopivuuden vuoksi.

Ohjelmoitaessa prototyyppiä ohjelmointiprosessi oli iteratiivista. Sitä mukaa kun toiminnallisuutta lisättiin, koodin modulaarisuus huomioiden toiminnallisuutta jaettiin eri luokkiin mm. sillä perusteella, mikäli joitain datarakenteita täytyi laittaa muiden luokkien saataville sekä sen perusteella mikä on yksittäisen luokan tehtävä sekä oliko sen tehtävä välittää ja päivittää tietoa. Tietojen päivittämiseen ja välittämiseen oli tarjolla hyvät työkalut erilaisten EventHandlereiden muodossa.

4.2 Toteutuksen keskeiset ratkaisut

Sovelluksen keskeisenä tavoitteena oli saada ja ylläpitää yhteys palvelimelle, ottaa sieltä relevantti tieto talteen johonkin datarakenteeseen, näyttää otettu data reaaliajassa käyttöliittymässä ja välittää käskyjä palvelimelle annettujen sekvenssien muodossa.

Tällöin ainakin kommunikaattori, prosessidata ja sekvenssien ajoon liittyvä luokka täytyi olla saatavilla globaalisti sovelluksen lähdekoodissa. Koska App-luokkaa pystyi käyttämään sisääntulopisteenä, päätettiin alustaa globaalit luokat edellämainituista App-luokassa sekä pääikkuna, joka toimisi sovelluksen toiminnan solmupisteenä.

Tämä jaottelu mahdollisti myös parametri-ikkunan käyttämisen, kun kommunikaattorin tila säilyisi riippumatta missä ikkunassa käyttäjä liikkuu. Tiedon säilyttämiselle ikkunoiden yli jouduttiin myös luomaan erillinen olio parametridataa varten. Tällä tavalla vähitellen pystyttiin lisäämään toiminnallisuutta ja aina jokaisessa vaiheessa tarvittaessa jäsenneltiin koodia uuteen metodiin tai luokkaan.

Keskeisenä työkaluna käytettiin tapahtumia (events), joita voitiin nostaa aina tilan tai arvon muuttuessa. Alla kuvassa 10 on esitettynä lyhyt esimerkki.

```
/// <summary>
/// Gets or sets the level indicator value for Tank 100.
/// </summary>
/// <remarks>
/// Changes to this property trigger the <see cref="PropertyChanged"/> event.
/// </remarks>
3 references
public int LI100
{
    get => _LI100;
    set
    {
        if (_LI100 != value)
        {
            _LI100 = value;
            OnPropertyChanged(nameof(LI100));
        }
    }
}
```

Kuva 11. Esimerkki muutostapahtumasta

Tässä koodinpätkässä on kuvattuna yksittäisen arvon muuttuminen ja kuinka vain arvon muuttuessa arvon muuttumisesta asetetaan tapahtuma arvon muutosta. Kaikille arvoille on annettu oma EventHandler, joka seuraa onko kyseiseen arvoon tullut muutosta ja jos on, niin arvo haetaan ja päivitetään. Tapahtumia käytettiin niiden lähes reaaliaikaisen luonteen vuoksi ja sen takia että ainoastaan muuttunutta tietoa raportoitaisiin.

4.3 Toteutuksen onnistuminen

Sovellus toimii asiakasmäärittelyn mukaisesti, mutta sovelluksessa on edelleen kehittämisen varaa. Kaikki toiminnalliset vaatimukset saavutettiin, mutta sovelluksen kaikkia ei-toiminnallisia vaatimuksia ei aikapaineen vuoksi pystytty täysin todentamaan.

Tarkkuusvaatimukset 1. ja 2. ei pystytty vielä kunnolla testaamaan myös osittain sen takia, että sovellusta voisi myös vielä optimoida ja sen responsiivisuutta ja reaaliaikaisuutta parantaa, jotta tarkkuusvaatimuksissa voitaisiin pitäytyä paremmin.

Sen sijaan 3. vaatimuksessa onnistuttiin paremmin siinä mielessä, että ohjelma ei itse kaadu, joskin oli tilanteita, joissa prosessia ei enää pystytty palauttamaan yksin sovelluksen voimin.