SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

ÄLYKÄS TIEDONKERUUJÄR-JESTELMÄ VESITEOLLISUU-TEEN

TEKIJÄ/T: Riku Karttunen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Riku Karttunen	
Työn nimi Älykäs tiedonkeruujärjestelmä vesiteollisuuteen	
Päiväys 10.5.2019	Sivumäärä/Liitteet 38/0
Ohjaaja(t) Pasi Liimatainen, Hannu Korhonen	
Toimeksiantaja/Yhteistyökumppani(t) Preventos Informatics Oy	

Tiivistelmä

Opinnäytetyössä lähdettiin toteuttamaan pohjaa vesihuollon, kiinteistöjen ja teollisuuden vesijärjestelmien seurantaan keskittyvälle data-analytiikkajärjestelmälle alan ajankohtaisien tutkimuksien pohjalta. Tarkastelussa on myös ”Industry 4.0”-ilmiö sekä data-analytiikan merkitys nykypäivän kasvuhaluisten yrityksen kehityksessä. Työn toimeksiantajana toiminut Preventos Informatics tunnisti teollisuudessa ilmenneen tarpeen kehittää uudenlainen tiedonkeruujärjestelmä, jonka suurimpana vahvuutena olisi skaalautuvuus sekä joustavuus teollisuuden käyttökohteiden tarpeiden mukaan.

Toteutettu ratkaisu rajoittui opinnäytetyön rajauksien myötä järjestelmän rakenteen fyysisestä suunnittelusta taustajärjestelmän rajapintaan asti. Työssä tutkittiin hajautetun älyn konseptia hyvin skaalautuvan IoT-järjestelmän kontekstissa. Käytännössä ohjelmistokehitystä tehtiin sekä korkealla että matalalla tasolla ohjelmistokerroksien näkökulmasta. Korkealla tasolla suunniteltiin hajautetun järjestelmän laajempaa arkkitehtuuria, kun taas matalla tasolla keskityttiin yksittäisten piirien kanssa kommunikointiin. Työssä käytettiin tekniikoita Linux-maailman hyvin määritellyistä ja testatuista työkaluista aina Microsoftin uusimpiin kehityksiin IoT-tekniikoiden saralla.

Opinnäytetyöprosessin loppuun mennessä saatiin toteutettua demovalmis järjestelmä, jota kohtaan on osoitettu kiinnostusta teollisuuden toimijoiden puolelta. Järjestelmän kehitys on kuitenkin vasta alkuvaiheessa, joten opinnäytetyön aikana kehitetyn järjestelmän pohjalta ryhdytään tulevan kesän mittaan rakentamaan laajempaa järjestelmäkokonaisuutta.

Avainsanat
IoT, IIoT, Linux, Microsoft, Industry 4.0, Raspberry Pi, tiedonkeruujärjestelmä, teollisuus, teollisuustietokoneet, teollinen tiedonkeruu, data-analytiikka, älykkäät järjestelmät, esineiden internet

SAVONIA UNIVERSITY OF APPLIED SCIENCES THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Riku Karttunen			
Title of Thesis Smart Data Gathering System for Water Industry			
Date	10th May 2019	Pages/Appendices	38/0
Supervisor(s) Mr. Pasi Liimatainen, Senior Lecturer and Mr. Hannu Korhonen, Senior Laboratory Technician			
Client Organisation /Partners Preventos Informatics Oy			

Abstract

The goal of this thesis was to lay a foundation for a data gathering system that focuses on the aspects of industrial processes concerning water systems and water distribution networks. Emphasis was put on using the current advancements in technology and research concerning IoT and the industrial phenomenon called "Industry 4.0". The system was developed for a young start-up company called Preventos Informatics that had discovered a clear need for a scalable and flexible data gathering system that could be adjusted to the needs of clients operating in the industrial sector.

The scope of the developed system extends from the design of the physical system to the interface of the backend service dealing with the actual analysis of the gathered data. One of the main points of the thesis was to explore the concept of distributed processing of data, and the so-called smart grid, in the context of a scalable IoT system. Development touched on the higher-level design of the software architecture as well as the lower-level operations concerning communications with individual hardware chips. The technologies used spanned all the way from well-defined and well-tested Linux tools to newest advancements in IoT technologies from Microsoft.

As a result of this thesis, a demo-ready system was successfully developed, and several well-established industry operators have shown interest towards trying out the system in a demo setting. Although the specified part of the system was developed and completed, the life cycle of the whole system is still at its infancy. The development will continue during the following summer, where more features will be developed along with general testing and polishing.

Keywords
IoT, IIoT, Linux, Microsoft, Industry 4.0, Raspberry Pi, data gathering system, industry, industrial computers, industrial data gathering, data-analytics, smart systems, internet of things

ESIPUHE

Haluan kiittää Preventoksen asiantuntijoita, jotka jatkuvan tuen ja ohjauksen lisäksi auttoivat minua opinnäytetyöprosessiin liittyvien yksityiskohtien kanssa. Sain heiltä erittäin kiinnostavan opinnäytetyöaiheen sekä motivoivan työympäristön kehittää omia taitojani.

Kiitokset myös Savonian opinto-ohjaukselle, jonka sähköpostiviestien välityksellä sain tiedon tästä mahtavasta mahdollisuudesta. Lisäksi kiitokset myös opinnäytetyön ohjaajille Savonian puolelta, joiden kanssa työskentely sujui sulavasti.

Kuopiossa 10.5.2019
Riku Karttunen

TERMIT JA LYHENTEET

Iot, IIoT *((Industrial) Internet of Things)* Termi, joka kuvaa tavanomaisten laitteiden muunnosta älykkäiksi ja verkotetuiksi laitteiksi.

SaaS *(Software as a Service)* Ohjelmiston jakelumalli, jossa palvelu tarjotaan pääasiassa web-käyttöliittymien kautta tilauspohjaisesti.

SCADA, PLC, RTU *(Supervisory Control and Data Acquisition, Programmable Logic Controller, Remote Terminal unit)* Teollisuudessa käytettyjä järjestelmiä prosessien hallintaan, ylläpitoon sekä valvontaan.

RTC *(Real Time Clock)* Muusta järjestelmästä erillinen piiri, joka tarjoaa järjestelmälle mahdollisuuden seurata aikaa myös virtakatkoksien aikana.

ADC *(Analog-Digital-Converter)* Piiri, joka muuntaa analogisen signaalin digitaaliseksi.

ARM, x64 Yleisimmät mikroprosessoriarkkitehtuurit moderneissa tietokoneissa.

I2C *(Inter Integrated Circuit)* Sarjaväylä prosessorien ja mikrokontrollerien väliseen kommunikaatioon.

Kernel Ensimmäisenä laitteiston käynnistyksen yhteydessä suoritettava ohjelma, joka hallitsee kommunikaatiota komponenttien, ohjelmien ja oheislaitteiden välillä.

cron Ohjelmallisien tehtävien ajastukseen käytettävä ohjelma Linux-käyttöjärjestelmissä.

PT100 Lämpötilasensori, joka perustuu platinaelementtiin, jonka vastus määräytyy lämpötilan mukaan.

Sqlite Pienikokoinen relaatiotietokantaohjelmisto, jota käytetään tiedon varastointiin erityisesti sulautetuissa järjestelmissä.

REST *(Representational State Transfer)* Web-palvelujen luomiseen käytetty rajapinta-arkkitehtuuri.

PPP *(Point-to-Point Protocol)* Kommunikaatioprotokolla kahden verkotetun laitteen väliseen kommunikaatioon.

JSON *(JavaScript Object Notation)* Avain-arvo-pareista ja objektikokoelmista koostuva tekstiformaatti datan serialisointiin.

WM-Bus *(Wireless Meter-Bus)* Kulutusmittareiden etäluentaan kehitetty kommunikaatiostandardi.

Heartbeat Järjestelmän tilan kommunikoimiseksi käytettävä ohjelmisto tai laite, joka lähettää sykkeenomaisen signaalin tasaisin väliajoin.

SISÄLTÖ

1 JOHDANTO

Teollisuudessa ja sen prosesseissa on tapahtumassa eräs suurimmista murroksista teollisen informaatioteknologian kehityksessä. Älykkyys, analytiikka ja IoT ovat keskeisiä termejä tutkittaessa teollisuuden toimijoiden tulevaisuuden näkymiä. Kisa on käynnissä siitä, kuka pystyy valjastamaan informaation räjähdysmäisen kasvun pitkin teollisuuden prosesseja. Yrityksiltä vaaditaan kehitystä kaikilla tasoilla tiedon keräämisestä sen analyysin kautta saavutettavan kilpailuedun hyödyntämiseen asti. Tässä opinnäytetyössä lähdettiin kehittämään pohjaa tulevaisuuden data-analytiikkajärjestelmälle vesi-intensiivisen teollisuuden kontekstissa.

Asuin- ja teollisuuskiinteistöjen vesijärjestelmät ovat erinomainen käyttökohde modernille tiedonkeruujärjestelmälle niiden ollessa kriittisiä yhteiskunnan ja yrityksien toiminnan kannalta. Veden jakeluverkostoissa tapahtuvien vikatilanteiden analysoimiseksi tarvitaan hyvin suorituskykyinen sekä paikoin jopa ”älykkääksi” luokiteltava mittausjärjestelmä. Kentällä sijaitsevilla laitteistoilla on tarkoitus mitata kiinnostavia suureita kiinteistöjen vesijärjestelmistä ja prosessoida mitattu tieto analyysivalmiiksi ennen taustajärjestelmälle lähetystä.

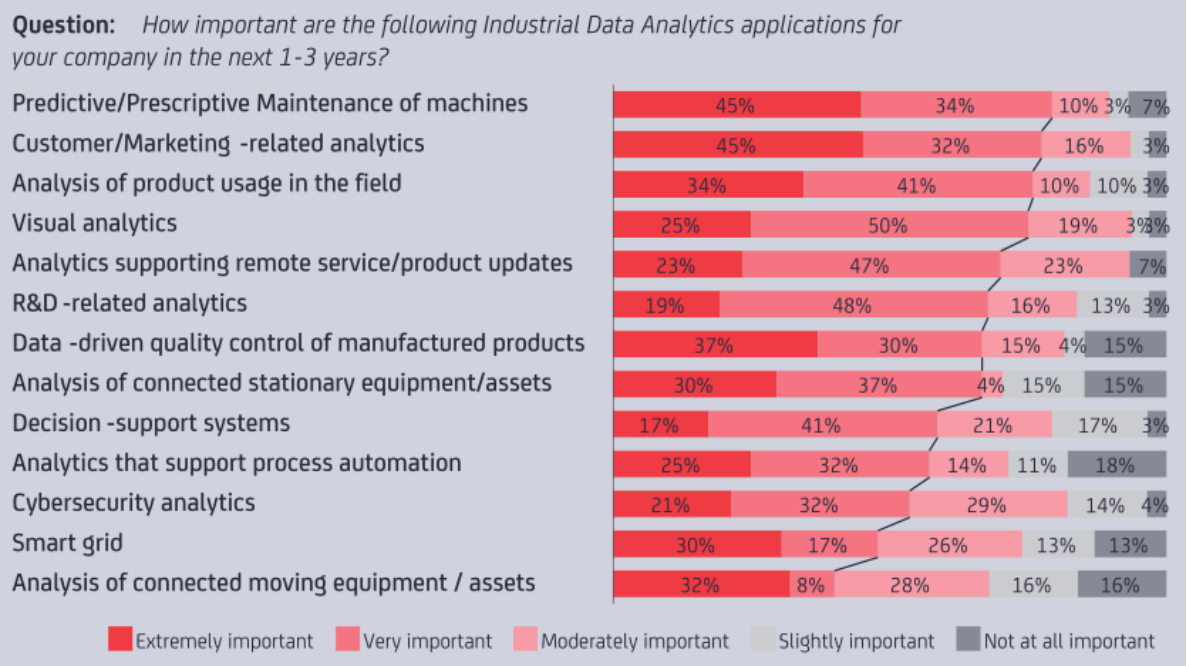
Vesihuollon ja kiinteistöjen seurantaan on olemassa jo useita erilaisia automaatiojärjestelmiä, mutta hajautetut ja ”älykkäät” järjestelmät ovat toistaiseksi vielä harvinaisuus. Tämän opinnäytetyön tavoitteena oli luoda hyvin skaalautuva järjestelmä, johon kuuluivat kentällä olevat teollisuustietokoneet, niihin kytkettävät mittalaitteet sekä taustajärjestelmä tiedon analysoimiseksi. Tavoitteena oli myös toteuttaa joustava ja monipuolinen järjestelmä, jonka luonnetta voidaan muokata kuhunkin käyttökohteeseen sopivaksi.

Työn tilaajana toimi Preventos Informatics, joka on nuori ja kasvava vesihuollon, kiinteistöjen ja teollisuuden vesijärjestelmien seurantaan keskittyvä informaatiopalveluja tarjoava yritys Kuopiosta. Toimeksiantajana Preventoksella pystyi tarjoamaan asiantuntijuutta ympäristö- ja vesitekniikan aloilta aina tietojärjestelmien ohjelmalliseen toteutukseen saakka. Opinnäytetyössä hyödynnettiin tätä voimavaraa sekä teoriapohjaisissa kysymyksissä että kokemusperäisen palautteen muodossa. Opinnäytetyön toteutuksen myötä vastataan teollisuudessa ilmenneeseen kehittyneiden tiedonkeruujärjestelmien tarpeeseen.

Opinnäytetyössä tutkittiin skaalautuvan tiedonkeruujärjestelmän arkkitehtuurin suunnittelua, toteutusta sekä kohdattuja haasteita ja valintoja. Tarkastelussa oli modernien tiedonkeruujärjestelmien rakenne sekä niiden suhde teollisuuden prosesseissa käytettäviin laitteistoihin. Esille tuotiin myös järjestelmän rakennetta matalalla tasolla ohjelmallisen toteutuksen merkeissä. Opinnäytetyön raportin lopuksi pohditaan toteutuksen merkitystä aihealueen tutkimuksen ja jatkokehityksen kannalta, sekä otetaan kantaa itse kehitysprosessiin suunnittelusta lopulliseen tuotteeseen saakka.

2 TIEDONKERUU JA ANALYTIIKKA TEOLLISUUDESSA

Teollisuuden prosesseihin kohdistuva tiedonkeruu ja siihen perustuvat data-analytiikan sovellukset tuottavat yrityksille hyötyjä niiden koko prosessiketjun laajuudelta. Kuva 1 havainnollistaa kyselyn tuloksia, jossa yrityksiltä kysyttiin tulevaisuuden näkymiä data-analytiikan hyödyntämiselle teollisuudessa. Kuva 1 voidaan päätellä, että käyttökohteet jakautuvat pääasiassa fyysisten prosessien optimointiin, erilaisten analytiikkapalveluiden tarjoamiseen sekä päätöksenteon muuttumiseen datapainotteisemmaksi.



Kuva 1 Teollisuuden data-analytiikan hyödyntäminen tulevaisuudessa (Lueth, Patsioura, Williams, & Kermani, 2016, p. 19)

Fyysisten prosessien optimointi perustuu IIoT-tyyppisten ratkaisujen tuottamaan tietoon yksittäisistä osista fyysistä prosessiketjua. Tämä mahdollistaa ennustavan huoltomenettelyn, reaaliaikaisen päätöksenteon kentällä sekä yksittäisten laitteiden optimoinnin osana suurempaa tuotantokokonaisuutta. Asiakkaalle näkyvät hyödyt esittäytyvät uusien ja räätälöityjen analytiikkapalveluiden muodossa esimerkiksi erilaisten SaaS-palveluiden kehittymisenä. Monelle futuristisimpana käyttökohteena voidaan pitää ihmisen poistumista prosessien kaikista vaiheista. Päätöksenteon perustuessa dataan, tulevaisuuden kehitys näyttää mahdollistavan niin sanotut ”älykkäät tehtaat”, joissa päätöksenteko on täysin automatisoitu. (Lueth, Patsioura, Williams, & Kermani, 2016, pp. 15-20) Tämän opinnäytetyön idea voidaan sijoittaa ”Smart grid”-lokeroon, joka esittää älykkäitä järjestelmiä erilaisten jakeluverkkojen muodossa.

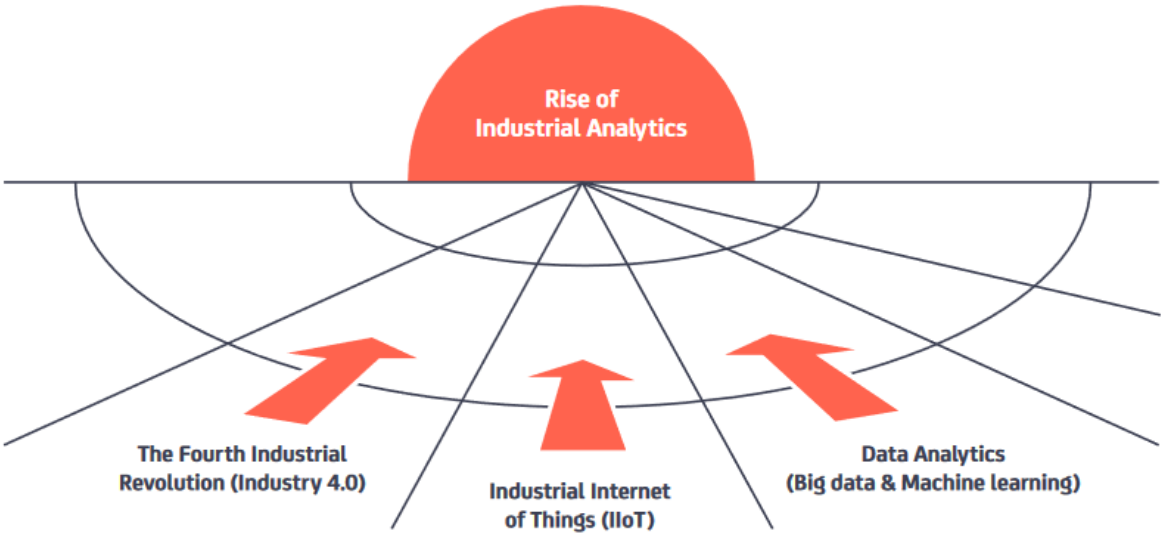
2.1 Industry 4.0

”Neljäs teollinen vallankumous” kuvaa teollisuuden ajanjaksoa, jossa hyödynnetään niin sanottuja älykkäitä järjestelmiä ja IIoT-laitteistoja data-analytiikasta saatavan kilpailuedun tuottamiseen. Edelliset ”teollisuuden vallankumoukset” käsittelivät pääasiassa tuotannon tehostumista mekaanisien ja

elektronisien laitteistojen yleistymisen myötä. (Lueth, Patsioura, Williams, & Kermani, 2016, pp. 11, 12) IIoT käsitteenä tarkoittaa datan keräämistä operationaalisista prosesseista eli teollisuuden laitteista itsestään. Matalalla tasolla kerätty data muunnetaan suurien datamäärien kautta analysoitavaksi dataksi, jolla voidaan vuorostaan tehdä prosesseista tehokkaampia. Lisääntynyt tuotto syntyykin epäsuorasti tehokkuuden parantumisen ja laiterikkojen ehkäisemisen välityksellä. (Gilchrist, 2016, p. 3)

IIoT-ratkaisujen ja niitä hyödyntävien analytiikan implementointi on toistaiseksi vielä lapsenkengissä johtuen potentiaalisten käyttäjäyrityksien epävarmuudesta uusien toimintatapojen implementoinnissa. Suurimpana kysymyksenä tuntuu olevan se, miten nykyiset järjestelmät voitaisiin sovittaa osaksi tulevaisuuden infrastruktuuria. Suurella osalla teollisuuden yrityksistä kuitenkin löytyy jo suuri osa IIoT-ratkaisujen kehittämiseen vaadittavista resursseista, joihin kuuluu mm. tehtaissa käytetyt automaatiojärjestelmät, jotka keskustelevat keskenään käyttäen samoja dataprotokollia, joihin IIoT-infrastruktuuri tyypillisesti perustuu. (Gilchrist, 2016, p. 3)

”Analytics is to data what refining is to oil: The process that turns the resource into a valuable product.” on hieno vertauskuva datan luonteesta ja siitä, miten se sen arvo nousee pinnalle prosessoinnin myötä. Dataa “rikastamalla”, joka on termi öljyn johdannaisien tuottamiselle, saadaan tiedosta paljon enemmän irti kuin päällepäin on mahdollista nähdä. Soveltamalla statistiikkaa sekä hienostuneita koneoppimisen algoritmeja, on raa’asta datasta mahdollista nostaa esille tietoa, joka ei nouse pinnalle pienempien datamäärien analyysissä. (Lueth, Patsioura, Williams, & Kermani, 2016, pp. 15, 16) On varmaa, että tämän suuren potentiaalin valjastaminen riippuu suuresti tulevaisuuden IoT-osaamisen kehittymisen nopeudesta. Monissa korkea-asteen koulutuksissa on jo aloitettu tulevaisuuden kysyntään vastaaminen.



Kuva 2 Teollisen analytiikan suurimmat vaikuttajat (Lueth, Patsioura, Williams, & Kermani, 2016, p. 11)

Kuva 2 esitetty kasvavan teollisuusanalytiikan rakenne havainnollistaa hyvin kolmea eri osatekijää kokonaisuuden rakennuspalikoina. Tämä projekti keskittyykin niistä keskimmäiseen, joskin käytännössä matalimmalla tasolla sijaitsevaan, teollisuuden esineiden internetiin (engl. Industrial Internet of Things). Internetin käyttötarkoitus on laajentunut pelkkien tietokoneiden välisestä kommunikaatiosta puhelimien ja tablettien kautta kaiken tyyppisiin ”esineisiin”. IIoT yhdistää teollisuuden prosessien ”esineet” toisiinsa ja internetiin käyttämällä kehittyvän teknologian mahdollistamia yhä pienempiä sulautettuja järjestelmiä. IIoT toimii perustuksina tulevaisuuden teollisten prosessien dataan perustuvalle analytiikalle ja päätöksenteolle. (Lueth, Patsioura, Williams, & Kermani, 2016, pp. 12, 13)

3 TIEDONKERUUJÄRJESTELMÄN RAKENNE

Automatisoidulla tiedonkeruujärjestelmällä tarkoitetaan järjestelmää, joka voidaan ohjelmoida keräämään, prosessoimaan, analysoimaan ja lähettämään haluttua dataa erilaisista mittauskohteista. Yleensä ratkaisut keskittyvät tiettyyn osa-alueeseen, mutta automaattisia tiedonkeruujärjestelmiä voidaan käyttää minkä tahansa datan keräämiseen lämpötiloista ja veden virtauksista aina videoon ja puheeseen saakka. Tässä projektissa automatisoitua tiedonkeruuta käytetään erilaisten vesijärjestelmien tilan seuraamiseen ja saadun tiedon analysointiin.

Automatisoidun ja tiheän tiedonkeruun avulla voidaan käyttää erilaisia statistiikkaan pohjautuvia metodeja, joilla voidaan muodostaa tarkkoja trendejä sekä analyysejä mittauskohteista. Käyttämällä tuoreita tietojenkäsittelyn tekniikoita kuten koneoppimista, voidaan vesijärjestelmien vikatilanteita jopa ehkäistä erilaisten mallien tuottamien tietojen perusteella. Viime vuosina on tehty suuria harppauksia korkealla lukutaajuudella operoivien mittausjärjestelmien kehityksessä. Kehittyneitä ja tehokkaita järjestelmiä voidaan hyödyntää vesijärjestelmien kontekstissa erityisesti korkean mittaustaajuuden vaativien ilmiöiden havainnointiin. (Sela;Rasekh;Shafiee;& Preis, 2018)

3.1 Hajautetut järjestelmät

Hajautettujen ja ns. ”älykkäiden” järjestelmien kehitys kasvaa kovaa vauhtia teollisuuden alalla. Tähän paradigmaan sopivia ratkaisuja on jo olemassa, mutta ne eivät ole kovin yleisiä varsinkaan vesiintensiivisessä teollisuudessa. Hajautetulla ”älyllä” tarkoitetaan tässä tilanteessa ratkaisua, jossa tiedon prosessointia on siirretty kentällä oleviin laitteistoihin. Tyypillisesti sensoreista tuleva data lähetetään jotakin kanavaa pitkin eteenpäin ja tämä tieto prosessoidaan keskitetysti toisessa järjestelmässä. Hajautetussa järjestelmässä sensoreilta mitattu tieto voidaan ensin muuntaa selväkieliseksi, esimerkiksi lämpötilaksi, jonka jälkeen tietoa voidaan prosessoida esimerkiksi suodattamalla virheelliset arvot tai keskiarvoistamalla mittaukset ennen lähetystä.

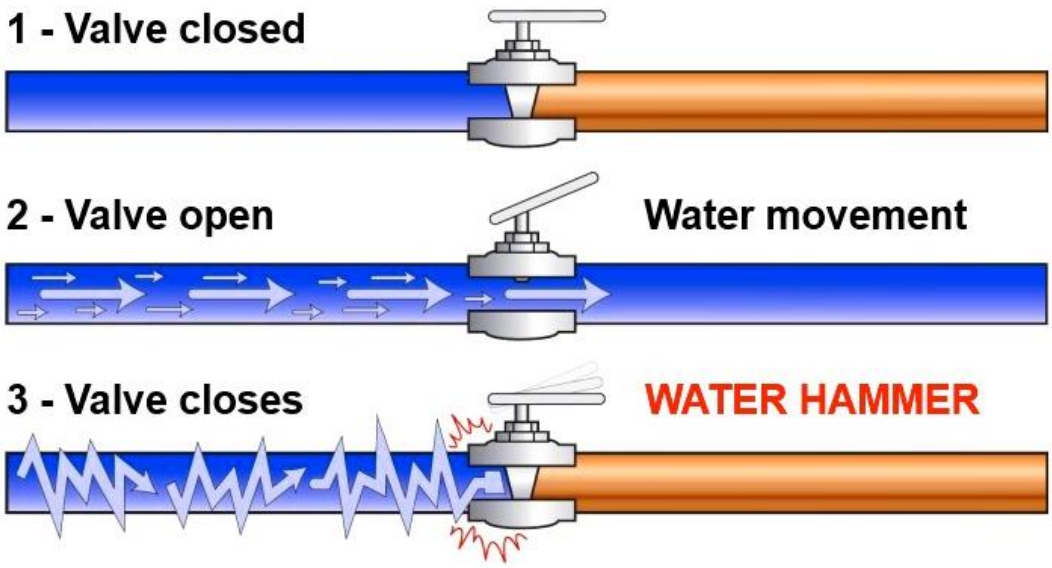
Tällaiselle IoT-maailman kasvusta syntyneelle järjestelmätyypille on ehdotettu useita hyvin kuvaavia nimityksiä. Eräs laajasti käytetty termi on ”edge computing”, joka nykypäivän pilvipalveluihin siirtymisen lomassa pyrkii siirtämään prosessointia takaisin toiseen ääripäähän. Medioissa puhutaankin siitä, miksi kaiken pilveen siirtymisen tuottaman työn jälkeen halutaan palata takaisin sinne mistä

alun pitäen haluttiin pois (Rokka, 2018). Edge computing pyrkii ratkaisemaan IoT-järjestelmien tuoman massiivisen datamäärän kasvun siirtämällä tiedon prosessointia pois pilvestä ja lähemmäksi tietoa tuottavia laitteita. Pilviratkaisut näyttävät usein vahvuutensa laskentatehon ja skaalautuvuuden merkeissä, mutta näilläkin ominaisuuksilla on rajansa, kun lähdetään puhumaan IoT-ratkaisujen ja Cisco Global Cloud Indexin ennustamasta 500 tsettatavun datamäärästä vuodelle 2019. Eräs konkreettinen esimerkki edge computing tarpeesta on nykyisten verkkoyhteyksien riittämättömyys suurien datamäärien välittämiseen. Ideana on lähettää jo jossain määrin prosessoitu ja siten arvoa kerännyt data raakadatan sijaan pilveen. (Shi, Cao, Zhang, Li, & Xu, 2016, pp. 637-639)

Hajautettujen järjestelmien etu näkyy hyvin myös järjestelmän prosessointitehon skaalautuvuudessa. Järjestelmän prosessointikapasiteettia voidaan helposti nostaa vaihtamalla kentällä olevia laitteistoja tehokkaampiin tai pilkkomalla tehtäviä useamman laitteen kesken. Tiedonkeruuoperaation kasvaessa taustajärjestelmän ei tarvitse olla tehokkaampi suuren osan prosessointia tapahtuessa jo kentällä. Toisaalta hajautettu järjestelmä on monimutkaisempi toteuttaa sekä kärsii hyvin tyypillisestä ongelmasta: mitä enemmän liikkuvia osia, sitä todennäköisemmin jokin näistä osista rikkoutuu. Lisäksi tiedon ollessa jo yksilöllisessä mitatun suureen muodossa, pitää laitteistojen ja taustajärjestelmän välinen kommunikaatioprotokolla suunnitella hyvin joustavaksi, jotta voidaan tukea suurta määrää erilaisia mittaustuloksia. (Milliman, 2015, pp. 20, 21)

3.2 Järjestelmän käyttökohde

Keräämällä tietoa asutus- ja teollisuusrakennuksien vesijärjestelmistä voidaan analysoida veden käyttöä ja mahdollisia vikatilanteita. Suureen osaan näistä vikatilanteista voidaan liittää erilaiset paineen vaihtelut jakeluverkoissa ja kiinteistöjen putkistoissa. Nämä vaihtelut aiheutuvat pääasiassa kulutuksen nopeasta vaihtelusta sekä vesijärjestelmien ongelmatilanteista, esimerkkinä erilaisten venttiilien liian nopea avaaminen tai sulkeminen. Veden virtauksen nopea pysäyttäminen aiheuttaa putkirikkoja ja vuototilanteita liikkuvan veden energian muuttuessa putkistoihin, laitteisiin ja liitoskohtiin kohdistuvaksi paineeksi. (Dudlik, Schönfeld, Schlüter, Fahlenkamp, & Prasser, 2002, p. 888)



Kuva 3 Vesijärjestelmässä tapahtunut poikkeustilanne (The Plumbing Blog, 2018)

Eräs tutkittavista ilmiöistä on putkistoissa usein tapahtuva niin kutsuttu paineisku (engl. Water hammer). Kuva 3 esittää yhtä tyypillisimmistä paineiskun aiheuttajista; liian nopea venttiilin sulkeminen. Tästä aiheutuu nopea paineisku, jonka havaitseminen on tärkeää erilaisten laiterikkojen havaitsemiseksi sekä ennakoimiseksi. Australian yliopistojen yhteistutkimuksessa (Sela;Rasekh;Shafiee;& Preis, 2018, s. 1) havaittiin, että paineiskujen nopeus tietyn tyyppisissä putkistoissa on noin 1000 m/s. Tyypillisien matalan näytteenottotaajuuden datankeräysjärjestelmien todettiin olevan riittämättömiä nopeiden paineiskujen havaitsemiseen. Opinnäytetyön toimeksiantajalla on paljon kokemusta tiedonkeruujärjestelmien toteuttamisesta erilaisiin olosuhteisiin. Aiempia järjestelmiä on yritetty soveltaa tähän käyttötarkoitukseen, mutta ilmiöiden luonteen seurauksena tarvittiin tehokkaampi korkean mittaustaajuuden järjestelmä.

Vesijärjestelmistä voidaan mitata monia muitakin suureita paineen vaihteluiden lisäksi kuten veden kulutusta, lämpötilaa ja laatua. Opinnäytetyön projekti päätettiin rajata paineiskujen havaitsemiseen ja niiden tarkkaan mallintamiseen. Järjestelmän kehittyessä voidaan alkaa implementoimaan laajempaa sekä geneerisempää järjestelmää, jotta voidaan tutkia erilaisten mittausten suhteita toisiinsa. Erityisesti vesijärjestelmissä on tyypillistä, että esimerkiksi vedenjakeluverkon käyttäjien vuorokausirytmeillä on suora yhteys erilaisten ilmiöiden tapahtumahetkeen.

3.3 Käytetyt laitteistot

Tiedonkeruujärjestelmät koostuvat yleensä jonkin tyyppisistä sensoreista, mikrofoneista tai kameroista, jotka välittävät haluttua dataa eteenpäin jonkin hyvin määritellyn protokollan mukaisesti. Näitä niin sanottuja päätelaitteita käytetään yleensä jonkin kontrollerin välityksellä, josta voidaan käyttää peitetermiä ”teollisuustietokone”. Teollisuustietokoneella ei tässä tapauksessa tarkoiteta perinteisiä automaatioon käytettyjä laitteistoja, vaan enemmänkin PC-tyyppisiä Linuxiin pohjautuvia järjestelmiä. Kontrollerin saadessa tietoa päätelaitteilta, voidaan tehdä monia erilaisia operaatioita muokkaamisesta ja tallentamisesta aina eteenpäin lähettämiseen. Täyden Linux-käyttöjärjestelmän päällä työskennellessä mahdollisuudet ovat miltei rajattomat. Modernit teollisuustietokoneet ovat hyvinkin suorituskykyisiä ja niillä voidaan suorittaa monen tyyppistä paikoin raskastakin laskentaa. Tilanteet, joissa hyödyllisen tiedon saavuttamiseksi tarvitaan paljon dataa sekä suuret määrät prosessointitehoa, ovat erinomaisia kandidaatteja teollisuustietokoneisiin pohjautuvalle tiedonkeruujärjestelmälle.

Nykypäivän teollisuuden prosessit ovat suurelta osin automatisoituja, joten niihinkin on pakko liittyä tiedon välitystä ja analyysia ainakin jollain tasolla. Opinnäytetyössä kehitettävä järjestelmä jakaakin ominaisuuksia näiden järjestelmien kanssa, keskittyen kuitenkin enemmän IoT-paradigman mukaiseen arkkitehtuuriin. SCADA-järjestelmiä käytetään pääasiassa prosessien automaatioon ja tarkkailuun teollisuudessa. Järjestelmät koostuvat mm. PLC- ja RTU-tyyppisistä laitteistoista sekä muista fyysisistä komponenteista pitkin prosessiketjua. Vaikka tämän tyyppinen järjestelmä vaikuttaakin kattavan myös opinnäytetyössä toteutettavan järjestelmän ominaisuudet, ei niitä tyypillisesti ole suunniteltu modernien data-analytiikkaratkaisujen pohjaksi. (Daneels & Salter, 1999)

Eroavaisuudet alkavat esittäytyä tutkittaessa järjestelmän soveltuvuutta hyvin erilaisten järjestelmien datan keräämiseen. Prosessiteollisuuden laitteita ei usein suunnitella toimimaan muiden valmistajien ja versioiden kanssa. Tyypillisesti valitaan jonkin tyyppinen arkkitehtuuri, eikä siitä juurikaan haarauduta. Järjestelmien niin sanottuun ”jäykkyyteen” vaikuttaa myös erilaisten lisensointikäytäntöjen yleisyys. Suurien kokoluokkien prosessijärjestelmiä on täten vaikeaa ryhtyä muokkaamaan suunnitteluvaiheessa tehtyjen valintojen jälkeen. IoT:maiseen teknologiaan perustuvat järjestelmät ovat jo määritelmänsä takia parempia tiedonkeruuratkaisuihin. Arkkitehtuurin modulaarisuutta voidaan pitää kaikkien IoT-ratkaisujen ytimenä. Yhteensopivuus on eräs tässä projektissa kehitetyn järjestelmän suurimmista vahvuuksista. Kumpikaan järjestelmä ei kuitenkaan pyri korvaamaan toista, vaan niiden tulisi toimia yhtenäisenä, joskin modulaarisena, pakettina osana teollisuuden prosesseja. (Savjani, 2018)

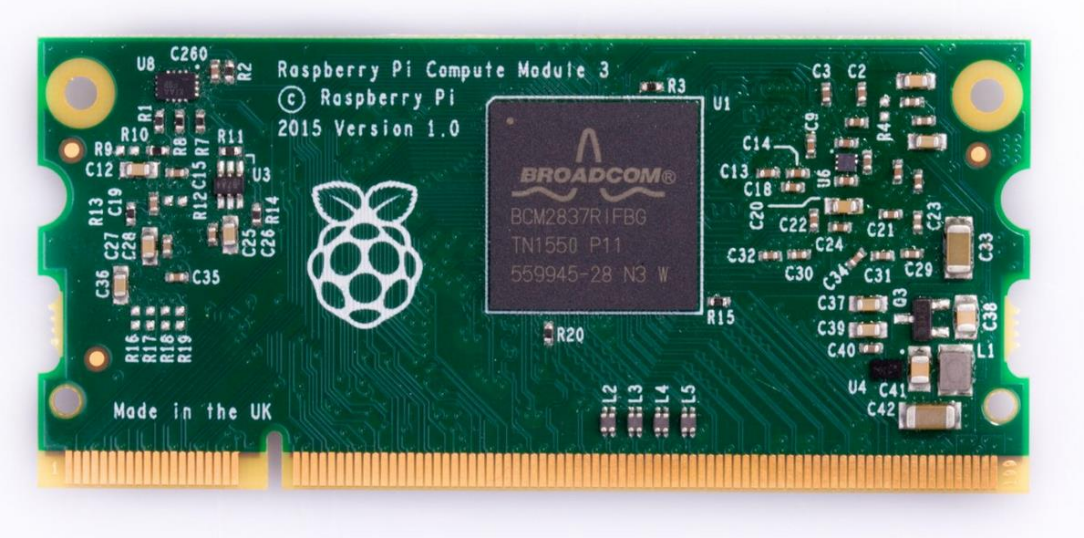
3.3.1 Teollisuustietokoneet

Teollisuuden sovelluksiin tarkoitetut tietokoneet ovat osittain hyvin erilaisia verrattuna tavallisiin tietokoneisiin, vaikka yhtäläisyyksiäkin löytyy paljon. Teollisuustietokoneista löytyy paljon samoja komponentteja kuten kuluttajamarkkinoilla olevia prosessoreita, Ethernet/USB/HDMI portteja, MMC-muistia ja monia muita laajalti käytettyjä ominaisuuksia. Eroavaisuuden alkavat näkyä, kun tutkitaan, miten järjestelmä on fyysisesti kasattu. Teollisuustietokoneilla on yleensä vaativiin olosuhteisiin suunniteltu kotelointi sekä hyvin modulaarisesti toteutettu järjestelmä. Esimerkiksi tietokoneen prosessori, keskusmuisti ja Flash-muisti on yleensä sijoitettu muusta järjestelmästä erilliselle ”tytärpiirilevylle”.

Kuva 4 Projektissa käytetty teollisuustietokone (Techbase Group Sp.z o.o.)

Mittauslaitteiksi suunnitellut teollisuustietokoneet toimivat usein akkukäyttöisesti tai ne on muuten suunniteltu viemään minimivirtaa jakeluverkosta. Jos tietoa halutaan mitata tapahtuvien poikkeustilanteiden mukaan, on usein implementoitu jonkinlainen matalan virrankäytön tila, kunnes poikkeustilanne tapahtuu. (Harrison, 2015, p. 59) Tilanteissa, joissa mittauslaitteisto on sijoitettu sähköverkon ulottumattomiin, käyttövirta täytyy hankkia hyödyntämällä saatavilla olevia resursseja kuten aurinko- tai tuulivoimaa. Nämä käyttövirran tuottomekanismit ovat ilmaisia, mutta usein myös riittämättömiä käytännön ratkaisuihin. (Suzdalenko, 2011, p. 1426)

Raspberry Pi Compute Module 3 on erityisesti teollisuuden käyttöön suunniteltu yhden piirilevyn tietokone, joka sisältää Raspberry Pi 3 -tietokoneen ominaisuudet pienemmässä mittaluokassa. Kyseinen moduuli mahdollistaa räätälöityjen piirilevyjen suunnittelun erilaisten projektien tarpeiden mukaan. Moduulin rajapintana muuhun järjestelmään toimii 200-pinninen DDR2 SODIMM -liitäntä. (Raspberry Pi Foundation) Projektissa käytetyn teollisuustietokoneen valmistaja on suunnitellut oman piirilevynsä, jossa on erilaisia projektissa hyödynnettyjä moduuleja kuten RTC, ADC, 3G-modeemi sekä useita sisään- ja ulostuloja erilaisille päätelaitteille.



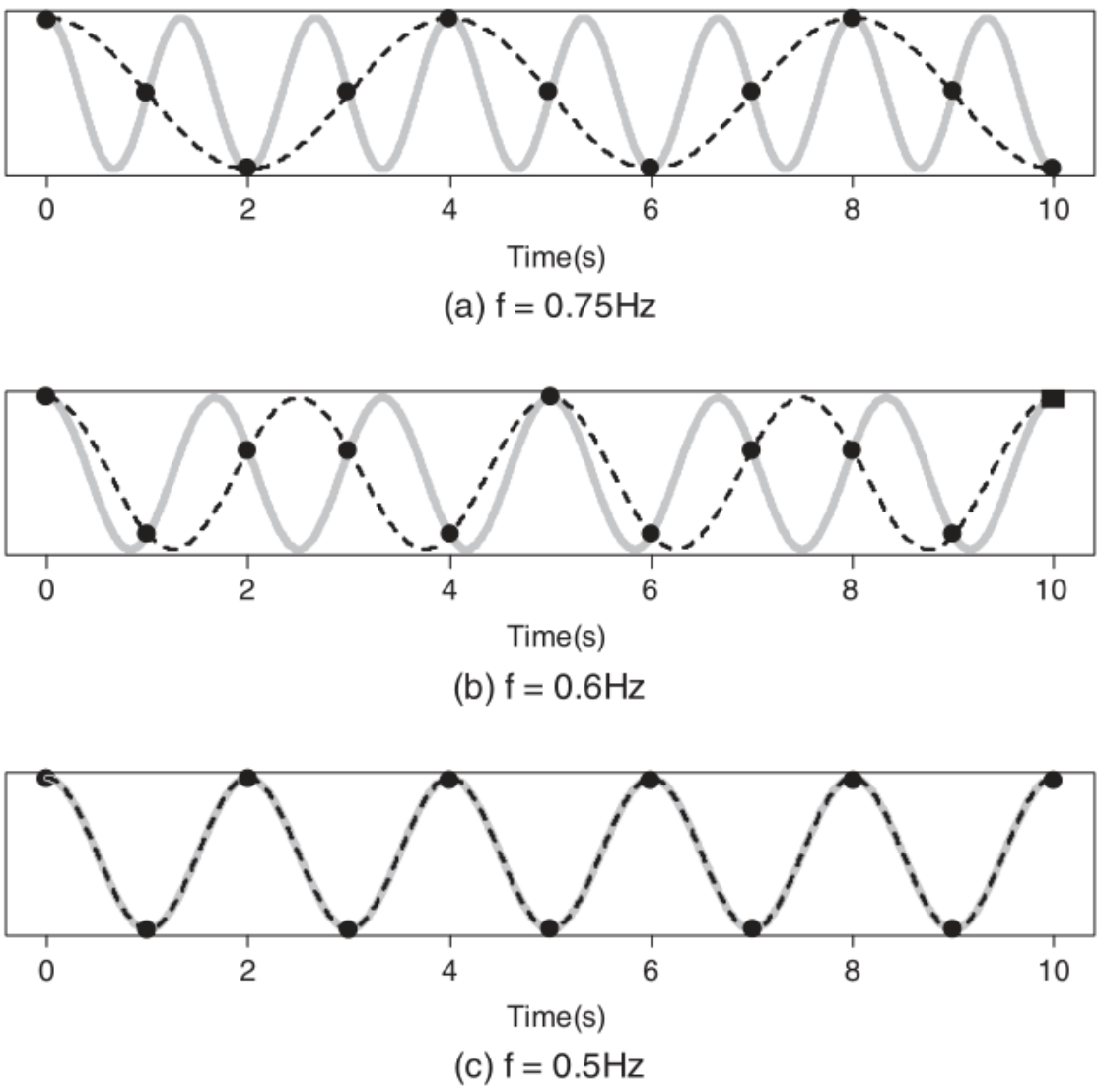
Kuva 5 Raspberry Pi Compute Module 3 (Raspberry Pi Foundation)

3.3.2 Ilmiöiden mittaaminen

Tutkittavien ilmiöiden fyysistä mittaamista voidaan pitää järjestelmien matalimpana tasona. Tyypillisiä signaalimuotoja sensoreilta luettaessa ovat mm. analogiset signaalit, pulssiluenta tai sarjaväylää pitkin välitetty informaatio. Usein datan mukana välitetään tietoa laitteesta itsestään; tämä on niin kutsuttua metatietoa. Metatietoa tarvitaan mm. kontekstin antamiseen mittaustiedolle. (Harrison, 2015, p. 61) Opinnäytetyön kehityksen aikana käytetty mittalaitteisto perustuu pääasiassa analogisiin signaaleihin, joten muiden signaalityyppien toimintaperiaatetta ei avata laajemmin. Jatkokehitysosiossa viitataan järjestelmän mahdollisiin tulevaisuuden kehityksiin laajemman sensorituen tiimoilta.

Analogisen mittaustiedon prosessointi vaatii ensin muunnoksen digitaaliseen muotoon. Analogista signaalia ei voida koskaan esittää täydellisesti digitaalisessa formaatissa johtuen siitä, että siihen vaadittaisiin ääretön määrä mittauspisteitä. Analogisien signaalien mittauksessa täytyy ottaa huomioon halutun ilmiön nopeus ja sen havaitsemiseen vaadittu näytteenottotaajuus. Jos näytteenottotaajuus on liian matala, järjestelmä ei kykene havaitsemaan ilmiöitä, jotka tapahtuvat liian nopeasti saavutettuun mittaustaajuuteen nähden. (Harrison, 2015, p. 64)

Kuva 6 havainnollistaa mittaustaajuuden merkitystä mitattavan signaalin todellisen luonteen esittämisessä. Sen lisäksi että mitattava ilmiö näyttäytyy vääristyneenä, voidaan huomata, että sitä ei välttämättä huomata lainkaan riippuen esimerkissä siniaallon vaiheesta. Vaikka kuvassa käsitelläänkin siniaallon mukaista signaalia, pätee sen havainnollistama näytteenottotaajuuden merkitys myös projektissa mitattaviin ilmiöihin korkeammalla tasolla. Esimerkkinä aiemmin mainittu vesijärjestelmän paineisku. Projektin aikana tehtyjen testien mukaan paineiskun korkeimman paineen, ja täten korkean jännitearvon, ajallinen kesto voi olla vain 50-70ms putkiston koon mukaan.



Kuva 6 Mittaustaajuuden vaikutus ilmiön havainnointiin (Harrison, 2015, p. 65)

3.3.3 Tiedon välittäminen eteenpäin

Tiedon mittauksen, muunnoksien ja prosessoinnin jälkeen data täytyy saada taustajärjestelmään data-analytiikan käsiteltäväksi. Vaihtoehtoina on datan varastointi lokaalisti myöhempää noutamista varten tai sen lähettäminen suoraan taustajärjestelmälle (Harrison, 2015, pp. 70, 71). Tiedon välittäminen eteenpäin joko muille laitteille tai taustajärjestelmälle yhdistettynä laitteessa tapahtuvaan tiedon prosessointiin tekee tästä järjestelmästä niin sanotusti ”älykkään” ja IoT-paradigmaan sopivan (Civerchia, et al., 2017, p. 6). Tieto halutaan lähettää pienellä viiveellä reaaliaikaisten monitorointisovelluksien ja/tai prosessinhallinnan mahdollistamiseksi. Vaikka järjestelmä kehitetäänkin pääasiassa data-analytiikan pohjaksi, voidaan sitä käyttää laitteistojen ja prosessien reaaliaikaiseen ja reaktiiviseen tarkkailuun.

Järjestelmän potentiaaliset käyttökohteet asettavat usein rajoja laitteiston liittämiselle verkkoon. Fyysinen yhteys käytettävään verkkoon esimerkiksi Ethernetin välityksellä on usein luotettavin tapa välittää tietoa. Tällainen verkkoyhteys on kuitenkin luksusta ja sitä käytetäänkin vain tilanteissa, joissa järjestelmän käyttökohde on myös fyysisesti kytketty verkkoon tai järjestelmältä vaaditaan maksimaalista luotettavuutta. Eräs yleisesti käytetty ratkaisu on käyttää langattomia matkapuhelinverkkoja tiedon välitykseen. Langattomien verkkojen käyttö mahdollistaa laitteen sijoittamisen lähes mihin tahansa, mutta nostaa pinnalle uusia ongelmia luotettavuuden, nopeuden sekä kuuluvuuden muodossa. (Harrison, 2015, pp. 70, 71)

3.4 Vastaavat järjestelmät

4ZeroPlatform on erään italialaisen teollisuuden analytiikkapalveluja tuottavan yrityksen järjestelmä, johon kuuluu mittauslaitteisto ja web-pohjaisia mikropalveluja datan analysointiin. TOI (Things On Internet) yrityksenä keskittyy IIoT-pohjaisten palvelujen hyödyntämiseen teollisuuden prosessien optimoinnissa. 4ZeroPlatform pyrkii tuotteistamaan teollisuuden kasvavan informaatiovirran tuottaman kilpailuedun, jota on nykyisin mahdollista saada monesta eri paikasta prosessien varrella. (TOI Srl, 2018)

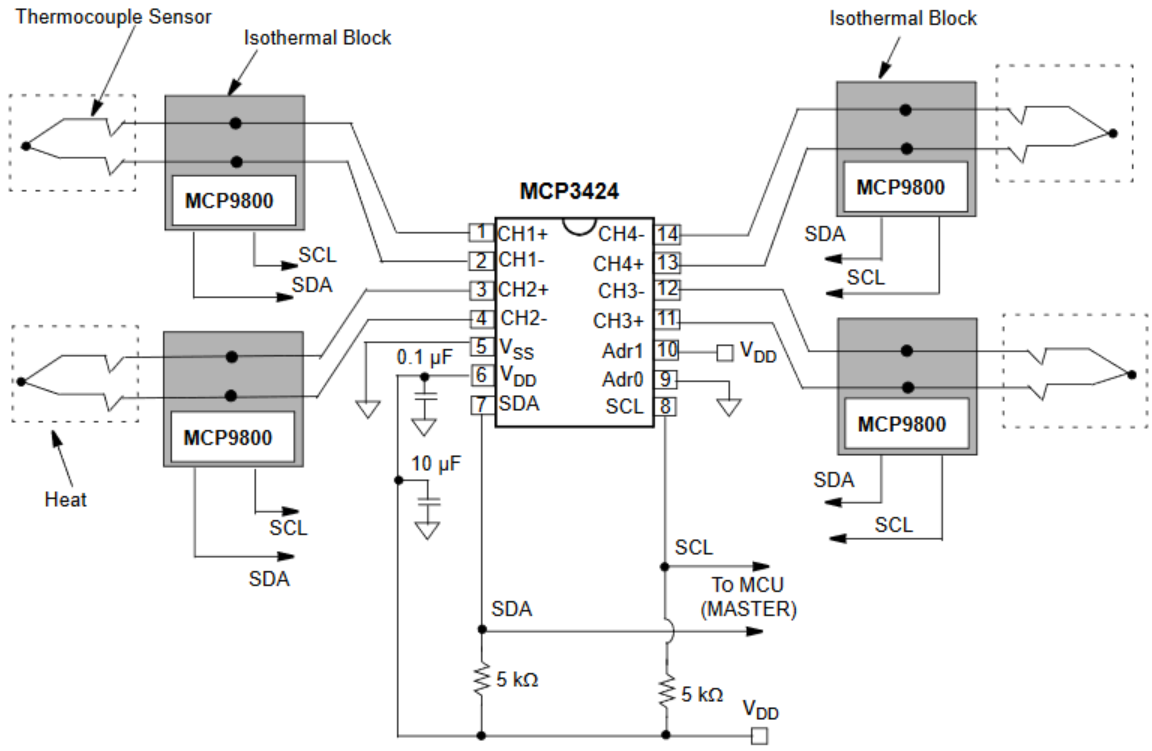
Tämän projektin idealla on hyvin paljon päällekkäisyyttä 4ZeroPlatform -ekosysteemin kanssa. Molemmissa tapauksissa pyritään hyödyntämään potentiaaliset tietolähteet moderneja tietoarkkitehtuureja hyödyntäen. Suurin ero näiden projektien välillä näkyy teollisuuden alan kohderyhmissä. TOI:n nettisivujen perusteella 4ZeroPlatformin asiakaskunta koostuu pääasiassa teollisuudesta tuotannon kontekstissa. Opinnäytetyöprojektissa kehitetyn järjestelmän kohderyhmä, vesipalveluiden ja -järjestelmien tuottajat, on siihen verraten hyvin kohdistettu ala infrastruktuurin valvontaan liittyvässä informaatioteknologiassa.

3.5 Käytetyt ohjelmistotason teknologiat

3.5.1 Käyttöjärjestelmä

Projektissa käytetyn laitteen käyttöjärjestelmänä toimii Raspbian Linux, joka on suunniteltu käytettäväksi erityisesti Raspberry Pi -pohjaisissa ARM-arkkitehtuuriin perustuvissa järjestelmissä. Raspbian on epävirallinen version Debian distribuutiosta, joka on käännetty optimoiden erityisesti ARM-prosessoreita varten. Helppokäyttöisyyden näkökulmasta Raspbian eroaa puhtaasta Debianista siten, että siihen on kerätty valmiiksi yleisimpiä elektroniikkaprojekteihin tarvittavia ohjelmistopaketteja. Linuxin kehitysmallia mukaillen Raspbian on avoimen lähteen ohjelmistoprojekti, jonka kehitys rahoitetaan lahjoituksilla. (Raspbian.org)

Sensorien lukemiseksi täytyy niihin kytkettyjen piirien kanssa keskustella käyttöjärjestelmän tarjoamien väylien kautta. Eräs näistä piireistä on Microchip MCP3424, joka on ADC-piiri sensorien jännitesignaalin muuntamiseksi digitaaliseen muotoon. MCP3424 kytkeytyy i2c-väylään, jonka välityksellä siihen voi kirjoittaa ja lukea dataa tavumuotoisena. (Microchip Technology Inc., p. 1) Kuva 7 on piirin manuaalista otettu ehdotettu kytkentä, joka on hyvin samankaltainen tämän projektin käyttökohteen kanssa. Kuvassa esitetyt lämpötilasensorit voivat olla mitä tahansa mitattavan jännitteen tuottavia sensoreita.



Kuva 7 MCP3424 -piirin ehdotettu kytkentä komponenttitasolla (Microchip Technology Inc., p. 34)

Järjestelmään kuuluvat palvelut sekä itse ohjelmisto täytyy saada käynnistymään laitteiston käynnistymisen yhteydessä. Myös ohjelmistolta tulevien viestien lokitus sekä uudelleenkäynnistäminen vikatilanteissa kuuluu käyttöjärjestelmän vastuualueisiin. Nämä tehtävät voi hoitaa systemd init-järjestelmällä, joka on osa Linux-käyttöjärjestelmää suurimmassa osassa nykypäivän jakeluja. Systemd on kokoelma järjestelmänhallintaan liittyviä työkaluja, joka käynnistetään ensimmäisenä järjestelmän käynnistyksen yhteydessä. Se hoitaa järjestelmän laajuisia ylläpitotoimia mukaan lukien prosessien hallinta sekä lokituspalvelut. (freedesktop.org, 2018)

3.5.2 Ohjelmistokehys

Suuressa osassa niin sanottuja IoT-projekteja, joissa ollaan tekemisissä yksittäisten piirien kanssa, käytetään C-ohjelmointikieltä. C on melko matalan tason kieli ja se sopiikin hyvin laitteistoläheisten projektien toteutukseen. Siitä huolimatta korkeamman tason kielillä voidaan usein saavuttaa korkeampi tuottavuus sekä turvallisuus ohjelmallisten ominaisuuksien kontekstissa. Fyysisien resurssien ollessa melko suuret (BCM2837 prosessori sekä 1GB keskusmuistia) verrattuna tyypillisiin pienen kokoluokan järjestelmiin, järjestelmän ohjelmisto pystyttiin toteuttamaan miltei millä tahansa kielellä ja/tai ohjelmistokehyksellä.

Toteutuskieleksi ja ohjelmistokehykseksi päätettiin valita C# ja .NET Core 2. Valintaan vaikutti Preventoksen asiantuntijoiden kokemus .NET -ekosysteemistä sekä .NET Coren uusimmat kehitykset IoT-projektien saralla (Microsoft, n.d.). Muita kandidaatteja järjestelmän ohjelmiston toteutuskieleksi oli mm. Java ja Python. Python on hyvin suosittu kieli harrastelijapiireissä sekä Raspberry Pi -ekosysteemiin liittyvissä projekteissa. Siihen on tehty paljon kirjastoja sekä oppaita aloitteleville rakentelijoille. Laitteiston valmistajan oma ohjelmisto oli toteutettu käyttäen Java-ohjelmointikieltä sekä siihen liittyviä kirjastoja.

.NET Core on Microsoftin kehittämä ohjelmistokehys, jonka pääpiirteisiin kuuluu avoin lähdekoodi ja tuki monelle eri arkkitehtuurille, joihin kuuluu mm. ARM ja x64. Ohjelmistokehys tukee sekä Windows- että Linux-pohjaisia käyttöjärjestelmiä. .NET Corella kehitetään pääsääntöisesti käyttäen C# -ohjelmointikieltä, vaikkakin tukea löytyy myös F#:ille ja Visual Basic:ille. .Net Core on rakennettu hyvin modulaarisesti, sillä noin 90 prosenttia sen koko lähdekoodista on alustasta riippumatonta. Loput on tietyille alustoille tehtyä yhteensopivuuskoodia. (Microsoft, 2018)

3.6 Käytettävyyden varmistus

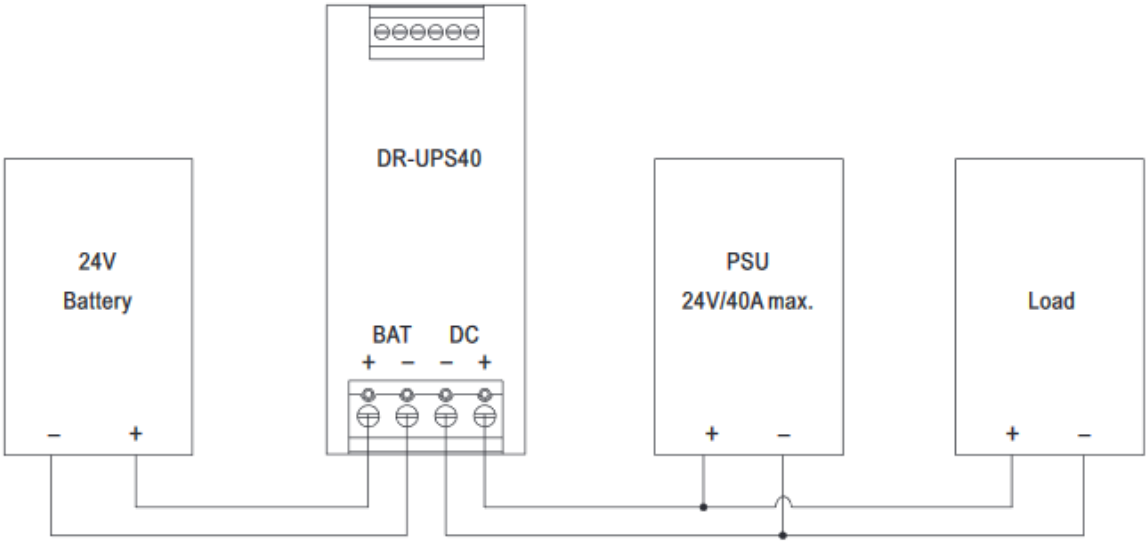
Järjestelmien käytettävyyden ja luotettavuuden varmistaminen on yksi suurimmista haasteista IoT-tyyppisissä projekteissa. Laitteistot ohjelmistoineen käydään yleensä asentamassa kohteeseen, jonka jälkeen ideaalitilanteessa ne toimivat siihen asti, kunnes niihin halutaan tehdä muutoksia. Tämä ei tietenkään toimi käytännön tilanteissa, vaan mahdolliset fyysiset sekä ohjelmalliset vikatilanteet täytyy ottaa huomioon jo järjestelmän suunnitteluvaiheessa.

Järjestelmän luotettavuudella tarkoitetaan yleensä järjestelmän kykyä toimia luotettavasti ilman virhetilanteita. Luotettavuus voi myös tarkoittaa laitteistolta saatavan informaation luotettavuutta. Käytettävyys kuvaa järjestelmän kykyä pitää halutut palvelut käyttäjän saatavilla. Molemmat näistä käsitteistä ovat hyvin vahvasti esillä IoT-projektien koostuessa suuresta määrästä verkotettuja laitteita sijoitettuna haastaviin ja virhealttiisiin ympäristöihin. (Sarkar, 2016, pp. 201, 206)

3.6.1 Fyysiset vikatilanteet

Fyysisellä tasolla järjestelmässä on useita mahdollisuuksia vikatilanteiden ilmenemiselle. Spontaaneita komponenttitason rikkoutumisia ei ole mahdollista korjata jälkikäteen asennuskohteeseen menemättä. Ainoa ehkäisevä toimenpide on varmistaa järjestelmän fyysisen ympäristön paras mahdollinen soveltuvuus suunnitellulle toiminnalle. Tähän kuuluu itse teollisuustietokoneen ja muiden järjestelmään kuuluvien laitteiden suojeleminen esimerkiksi erilaisilta nesteiltä, kaasuilta tai pölyltä tähän tarkoitukseen suunniteltujen koteloiden avulla. Laitteen fyysistä kokoonpanoa suunnitellessa täytyy ottaa huomioon ulkoisten tekijöiden lisäksi myös laitteiden käytöstä syntyvä lämpö.

Järjestelmä on fyysisten komponenttien odotetun toiminnan lisäksi riippuvainen asianmukaisen verkkovirran toimittamisesta laitteille. Verkkovirran katkos on hyvin tyypillinen vikatilanne, jos laitteen asennuskohde ei tarjoa varotoimia sen suhteen. Jos järjestelmällä mitattava ilmiö ei ole kriittinen tuntien tai jopa päivien virtakatkoksien suhteen, ei ongelmaan ole välttämätöntä keksiä ratkaisua. Jos järjestelmää käytetään esimerkiksi pitkän aikavälin keskiarvon mittaamiseen, voidaan hetkittäinen aukko mittauksissa ottaa huomioon analyysivaiheessa. Poikkeustilanteisiin perustuvassa mittaustoiminnassa on taas tärkeää, ettei järjestelmän toimintaan tule aukkoja, joissa tapahtuvia poikkeustilanteita ei havaittaisi lainkaan. Tämä ongelma voidaan ratkaista keskeytymättömän virransyötön (engl. Uninterruptible Power Supply) avulla.



Kuva 8 Esimerkkikytkentä akkuvarmennetusta järjestelmästä (MEAN WELL Enterprises Co., Ltd., 2018)

Keskeytymättömällä virransyötöllä tai virtalähteellä tarkoitetaan laitetta, joka yhdistää verkkovirran päälähteen johonkin toissijaiseen virtalähteeseen. Pääsääntöisesti käytetään verkkovirtaa, mutta virtakatkoksien aikana laitteisto vaihtaa käyttämään toissijaista virtalähdettä, joka voi olla mikä laitteisto tahansa akuista generaattoreihin. (US Patent No. US6184593B1, 1999) Keskeytymätöntä virtalähdettä käyttävä laitteisto ei välitä siitä, mistä lähteestä käyttövirta saadaan. Tässä projektissa on ideana tarjota järjestelmää keskeytymättömällä virtalähteellä tai ilman, riippuen käyttökohteesta.

3.6.2 Ohjelmistotason vikatilanteet

Ohjelmistotasolla vikatilanteita on mahdollista kohdata missä tahansa matkan varrella sensorilta mitattavasta tiedosta aina taustajärjestelmän analytiikkaan asti. Taustajärjestelmän virhetilanteita ei voida helposti korjata kentällä olevien laitteiden toimesta, joten laitteiden vastuun tiedon välityksestä voidaan sanoa loppuvan tiedon onnistuneeseen lähettämiseen. Siihen asti järjestelmän tulisi joko välttää erilaisia ohjelmallisia virhetilanteita tai sopeutua niihin minimoiden mahdollinen datan menetys.

Niin sanottu vahtikoira-ajastin (engl. watchdog timer) on usein fyysisenä komponenttina implementoitu itsenäinen osa järjestelmää, joka huolehtii järjestelmän ja/tai ohjelman uudelleenkäynnistämisestä tilanteissa, joissa jumiutumiseen johtava virhetilanne on tapahtunut. Toimintaperiaatteena on nollata tai asettaa ajastin johonkin määriteltyyn aikaan tietyn ajanjakson välein. Jos ajastin saavuttaa virhetilanteeseen oikeuttavan aikamääreen, järjestelmä tai ohjelma käynnistetään uudelleen. Jos järjestelmä sietää uudelleenkäynnistyksiä, on tämä hyvä tapa parantaa järjestelmän ”sitkeyttä” vikatilanteissa. Toinen samankaltainen tekniikka on niin sanottu laitteen syke (engl. heartbeat), joka lähettää jonkin tietopaketin laitteen tilasta taustajärjestelmälle tietyn ajan välein. Laite voi näin kertoa toimivansa määriteltyjen parametrien mukaisesti tai sykkeen puuttuessa tapahtuneesta vikatilanteesta. Toisaalta jos laitteen tarkoitus on lähettää dataa tasaisin väliajoin, datapaketit itsessään toimivat laitteen sykkeenä. (Sarkar, 2016, pp. 211, 212)

Poikkeuksellisia tilanteita tapahtuu myös järjestelmän ohjelmakoodissa, näitä kutsutaankin kuvaavasti poikkeuksiksi. Tilanteita, joissa on mahdollista, että ohjelman toiminnalle tärkeä resurssi ei käyttäydykään odotetulla tavalla, tarvitaan poikkeuksien hallintaa. C# -kielessä on tähän tarkoitukseen tehty try-catch lause. Poikkeuksien hallintaa ei tulisi kuitenkaan käyttää osana ohjelman normaalia kulkua vaan sen tulisi toimia tapana toipua mahdollisista ohjelman kaatumiseen oikeuttavista tilanteista (Hunt & Thomas, 2000, pp. 126, 127). Potentiaalisia poikkeuksia voidaan kohdata erityisesti ulkoisien resurssien kanssa kommunikoidessa, joihin tässä projektissa kuuluu mm. taustajärjestelmä sekä fyysiset tietokoneen piirit ja niihin kytketyt sensorit.

4 OHJELMISTON TOTEUTUS

Projektin alkuvaiheessa suuressa roolissa oli teollisuustietokoneen valmistajan ohjelmistojen sekä dokumentaation tutkiminen. Tavoitteena oli selvittää, mitä valmiita ohjelmistokomponentteja voitaisiin käyttää tämän projektin toteutukseen. Erittäin suureksi ongelmaksi muodostui valmistajan tekemä dokumentaatio. Dokumentaatio oli erittäin puutteellista, eikä siitä ollut hyvin määriteltyä tuoretta versiota saatavilla. Valmistajaan oltiin yhteydessä kyseisistä asioista, mutta tyhjentävää vastausta siitä, miten jo saatavilla ohjelmistoa voitaisiin käyttää hyödyksi, ei saatu. Noin kolmen viikon tutkimustyön ja testauksen tuloksena päädyttiin rakentamaan kokonaan oma ohjelmisto laitteiston fyysisien komponenttien rajapinnoista lähtien. Tämä tuotti tietenkin lisää työtä, mutta toisaalta voidaan olla varmoja, että koko prosessi mittauksesta analyysiin on täysin opinnäytetyön tilaavan yrityksen kontrollissa. Eräs mahdollisista syistä huonolle dokumentaatiolle on, että suurin osa tämän tyyppisiä laitteistoja tuottavista yrityksistä haluaa laitteen lisäksi myydä laitteen ohjelmistoon liittyviä

palveluja. Voi olla, että laitteen valmistajalla on jokin sisäinen dokumentaatio, joka määrittelee suuren osan tiedoista, joita tässä projektissa olisi tarvittu valmistajan ohjelmiston hyödyntämiseksi.

Teollisuustietokoneiden valmistajien ohjelmistojen käyttämisessä on se hyvä puoli, että kaikki matalan tason ohjelmistolliset yksityiskohdat on abstraktoitu pois, jotta käyttäjän ei tarvitsisi tietää mitään muuta kuin oman käyttökohteen vaatimukset ja haluttu tieto. Käytännössä tarvitsisi määrittää vain konfiguraatio, jossa kerrotaan mitä tietoa halutaan lukea, jonka jälkeen tieto saadaan jotakin kautta ulos. Käyttöönotto on nopeaa eikä kokemusta ohjelmallisesta toteutuksesta tarvita. Toisaalta käyttäjä ei pysty muokkaamaan tiedon lukemisen prosessia konfiguraation tarjoamien mahdollisuuksien ulkopuolella. Ainut keino saada jokin lisäominaisuus ohjelmistoon on ottaa yhteyttä laitteen valmistajaan ja neuvotella ominaisuuden implementoinnista.

Alusta asti toteutetussa ohjelmistossa on omat hyvät ja huonot puolensa. Suurin ja ilmeisin huono puoli on toteutukseen vaaditun ajan määrä. Toteutuksessa pitää rakentaa kaikki bittitasolta datan lähetykseen asti itse. Toisaalta se avaa myös mahdollisuuden ohjelmiston täydelliselle muokattavuudelle. Riippuvaisuus laitteen valmistajaan on melko mitätön, koska toteutuksesta käytetty dokumentaatio haettiin komponenttien valmistajilta kuten Microchip:iltä. Nämä valmistajat ovat suuria toimijoita piirien valmistuksessa, joten niiden dokumentaatiot on usein toteutettu todella laadukkaasti ja yksityiskohtaisesti. Jos järjestelmä on osattu toteuttaa käyttäen vain komponenttien valmistajien dokumentaatiota, voidaan esimerkiksi Raspberry Pi Compute Module 3 -pohjalle rakentaa oma piirilevy komponentteineen. Tämä on kuitenkin melko kaukainen idea, koska käytännössä tullaan käyttämään valmiiksi rakennettuja teollisuustietokoneita.

Niin sanotun tyhjän kankaan ollessa edessä toteutuksen näkökulmasta, voitiin aloittaa järjestelmän arkkitehtuurin toteutuksen tunnustelu. Eräs suurimmista ajan kuluttajista järjestelmän esivaatimuksien opiskelun ohella oli järjestelmän ytimen eli mittauksien prosessoinnin logiikan suunnittelu. Tavoitteena oli suunnitella riittävän geneerinen sekä modulaarinen ohjelmisto, jotta sitä voitaisiin käyttää mahdollisimman moneen käyttökohteeseen, sen ollessa myös helposti muokattavissa. Ideaalitilanteessa järjestelmä tukisi kaiken tyyppisiä sensoreita sekä muita oheislaitteita, joille löytyisi fyysinen liitäntä teollisuustietokoneesta. Siten ohjelmisto käyttäisi ns. kaikki saatavilla olevat resurssit. Tässä vaiheessa kuvaan astui opinnäytetyön rajauksen tulkinta. Toteutuksen edetessä päätettiin viikkopalaverien välityksellä rajata opinnäytetyö konkreettisesti analogisten sensorien lukemiseen, niiltä saadun tiedon prosessointiin ja prosessoidun tiedon lähetykseen. Jatkokehitys-osiossa käydään tarkemmin läpi ominaisuuksia, jotka jäivät rajauksen ulkopuolelle.

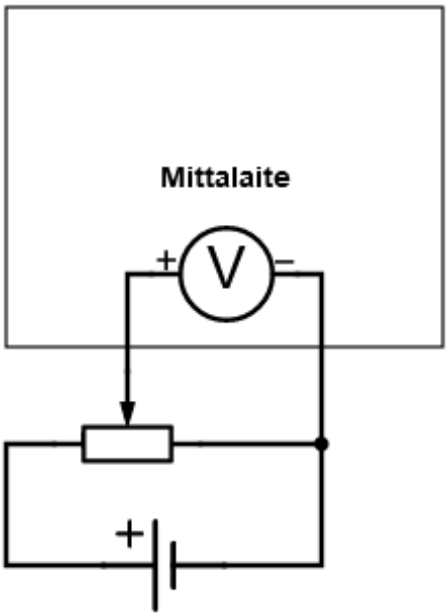
Siirryttäessä pois laitteistojen valmistajan ohjelmistoista haluttiin, että ne eivät myöskään vaikuta millään tavoin uuteen järjestelmään. Valmistajan ohjelmisto käyttää samoja laitteiston resursseja, joten konflikteja voi syntyä varsinkin komponenttitasolla kommunikoidessa. Esimerkkinä ADC-piirin lukeminen; laitteiston valmistaja oli toteuttanut kyseisen ominaisuuden käyttämällä Linuxin kernelistä löytyvää moduulia, joka kirjoitti piirin rekistereihin samaan aikaan toteutetun ohjelmiston kanssa, jonka seurauksena luettiin sensoriarvoja vääristä kohteista. Toinen valmistajan toteuttamista

ominaisuuksista oli mobiilidatayhteyden muodostus sekä siihen liittyvien ylläpitotoimien suorittaminen.

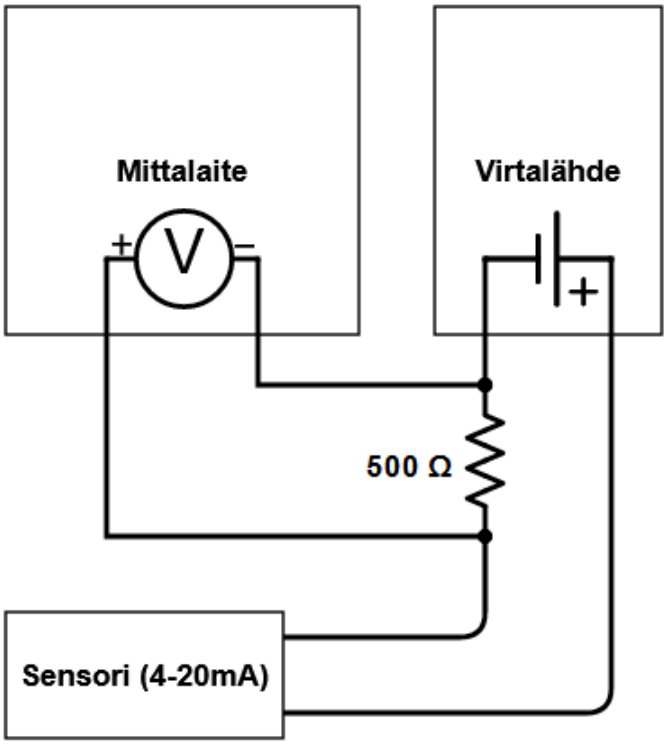
Toteutus- ja testausvaiheessa tutkittiin, voidaanko näitä valmiita ohjelmiston osia käyttää uuden tekemisen sijasta. Valitettavasti kuten aikaisemmin mainittiin, valmistajan dokumentaatio näiden ominaisuuksien suhteen oli erittäin vajavaista. Ratkaisussa päädyttiin hylkäämään valmistajan mittausohjelmiston lisäksi muutkin ”apuohjelmat”, jotka hallitsivat mm. internetyhteyden muodostamista ja yleisiä ylläpitotoimia. Käytännössä tämä tarkoitti cron:illa toteutettujen ajastettujen tehtävien puhdistusta ja tiettyjen systemd:llä ylläpidettyjen prosessien käytöstä poistamista.

4.1 Sensorien lukeminen matalalla tasolla

Mittaustiedon matalimpana tasona voidaan pitää jonkin sensorin muodostamaa jännitettä jonkin tarkkailtavan suureen muutoksen seurauksena. Jännite vaihtelee ideaalitilanteessa lineaarisesti suhteessa mitattavaan suureeseen. Johonkin vaiheeseen mittausprosessia täytyy määrittää lineaarisen muuntofunktion kerroin ja vakio. Järjestelmän kehitysvaiheessa testaukseen käytettiin yksinkertaista paristo-potentiometri kytkentää (Kuva 9), jolla pystyttiin simuloimaan jännitteen vaihtelua todellisessa mittaustilanteessa. Käytössä oli myös PT100-lämpötilasensori, jonka ulostulona oli 4-20mA virtasignaali. Käytetty teollisuustietokone ei tue virtasignaalin mittausta, joten 4-20mA signaali täytyi muuntaa 2-10V signaaliksi käyttämällä 500 Ohmin vastusta (Kuva 10).



Kuva 9 Yksinkertainen kytkentä jännitesignaalin testaamiseen



Kuva 10 Järjestelmän testaus 4-20mA sensorin avulla

4.1.1 Rajapinnat sensorien lukemiseksi

Projektin alkuvaiheessa sensoridatan lukemisen menetelmäksi ei oltu vielä keksitty yhtä ja oikeaa ratkaisua. Järjestelmän komponenteilta tulevaa tietoa voidaan lukea monella eri tavalla joko käyttäen valmiiksi kehitettyjä ratkaisuja, joissa matalan tason kommunikointi on abstraktoitu pois, eikä käyttäjän tarvitse tietää mitään käytettävästä laitteesta komponenttitasolla. Eräs tutkituista menetelmistä oli käyttää Linuxin pseudotiedostojärjestelmää nimeltä sysfs. Sysfs paljastaa rajapinnan käyttöjärjestelmän tunnistamiin laitteisiin tiedostojärjestelmän muodossa. Esimerkiksi käytetty ADC-piiri näyttäytyi hakemistona i2c-väylään kytkettyjä laitteita esittävässä hakemistossa. ADC-piirin fyysiset portit ovat yksittäisiä ”character device” -tiedostoja, joita voidaan käsitellä käyttäen normaaleja tiedostorajapinnan työkaluja. Esimerkiksi tiedostosta nimeltä ”in_voltage0_raw” voidaan lukea ensimmäiseen fyysiseen porttiin kytketyn sensorin jännite. Kirjoittamalla tiedostoon nimeltä ”in_voltage_sampling_frequency”, voidaan määrittää piirin käyttämä lukutaajuus.

Ohjelmiston ensimmäisissä versioissa ideana oli hallita piiriä käsitellen näitä tiedostoja, jolloin matalan tason logiikka pysyisi melko yksinkertaisena. Ongelmaksi nousi testauksen myötä todella hidas kirjoitus- ja lukunopeus. Ongelmaa tutkittaessa kävi ilmi, että hitaiden nopeuksien syynä oli sysfs-tiedostojärjestelmän luonne pseudotiedostojärjestelmänä. Kyseisen tiedostojärjestelmän fyysisiä komponentteja kuvaavat hakemistot ja tiedostot eivät ole todellisia tiedostoja laitteen kovalevyllä, vaan käyttöjärjestelmän luomia rajapintoja laitteisiin tiedostojen muodossa. Todellisia tiedostoja testauksen merkeissä luettaessa päästiin hyväksyttäviin nopeuksiin tämän projektin kontekstissa. Ongelman seurauksena päädyttiin tutkimaan ns. ”oikeaa” tapaa suorittaa datan lukeminen matalalla

tasolla, joka on suorittaa lukeminen i2c-väylän kautta käyttäen piirin valmistajan määrittelemää rajapintaa.

4.1.2 Sensorien lukeminen ADC-piiriltä

Sensorilta tuleva jännite mitataan teollisuustietokoneessa sijaitsevan Microchip MCP3424 ADC-piirin toimesta. Piiri tarjoaa mahdollisuuden muuttaa sen lukutaajuutta, resoluutiota sekä ennen A/D muunnosta lisättävän vahvistuksen suuruutta. Signaalia täytyy vahvistaa ennen muunnosta tilanteissa, joissa signaali on erittäin heikko. Esimerkkinä tilanne, jossa lämpötilan lukemiseen käytetään termopari -tyyppistä lämpötilasensoria. Piirin resoluutio on sidottu lukutaajuuteen, joten korkeammilla lukutaajuuksilla on käytettävissä vähemmän bittejä luetun jännitteen esittämiseen. Matalin lukutaajuus on 3,75 mittausta sekunnissa, jolloin resoluutioksi saadaan suurin arvo eli 18 bittiä. Suurin lukutaajuus on 240 mittausta sekunnissa, jolloin resoluutioksi saadaan 12 bittiä. Piiri voi mitata jännitteitä 2,048 voltista -2,048 volttiin. (Microchip Technology Inc., p. 1) Teollisuustietokoneen valmistaja on muuttanut tämän jännitealueen omilla kytkennöillään noin 10 voltista noin -10 volttiin. Valmistajan dokumentaatiossa ei ole selvitetty miten kyseinen jännitealueen muutos on tehty komponenttitasolla. Jännitteen muutos piti täten kokeilla käytännössä sensoria testaamalla.

Järjestelmän ohjelmallinen toteutus alkaa raa'an bittidatan lukemisesta ADC-piiriltä ja sen muuttamisesta asianmukaiseen esitysmuotoon. Ohjelmiston matalimman tason konfiguraatiossa on määritetty ADC-piirin osoite i2c-väylässä, siihen kytkettyjen sensorien muuntokaavat, fyysiset portit sekä mahdolliset ennen muunnosta lisättävät vahvistuskertoimet. Tiedon luku aloitetaan kirjoittamalla ADC-piirin rekistereihin konfiguraatiotavu, joka määrittelee mahdolliset asetusarvot sekä mistä portista jännitettä halutaan lukea (Kuva 11).

bit 7 **$\overline{\text{RDY}}$:** Ready Bit

This bit is the data ready flag. In read mode, this bit indicates if the output register has been updated with a latest conversion result. In One-Shot Conversion mode, writing this bit to "1" initiates a new conversion.

Reading $\overline{\text{RDY}}$ bit with the read command:

1 = Output register has not been updated.
0 = Output register has been updated with the latest conversion result.

Writing $\overline{\text{RDY}}$ bit with the write command:

Continuous Conversion mode: No effect

One-Shot Conversion mode:
1 = Initiate a new conversion.
0 = No effect.

bit 6-5 **C1-C0:** Channel Selection Bits

00 = Select Channel 1 **(Default)**
01 = Select Channel 2
10 = Select Channel 3 **(MCP3424 only, treated as "00" by the MCP3422/MCP3423)**
11 = Select Channel 4 **(MCP3424 only, treated as "01" by the MCP3422/MCP3423)**

bit 4 **$\overline{\text{O}}$/C:** Conversion Mode Bit

1 = Continuous Conversion Mode **(Default)**. The device performs data conversions continuously.
0 = One-Shot Conversion Mode. The device performs a single conversion and enters a low power standby mode until it receives another write or read command.

bit 3-2 **S1-S0:** Sample Rate Selection Bit

00 = 240 SPS (12 bits) **(Default)**
01 = 60 SPS (14 bits)
10 = 15 SPS (16 bits)
11 = 3.75 SPS (18 bits)

bit 1-0 **G1-G0:** PGA Gain Selection Bits

00 = x1 **(Default)**
01 = x2
10 = x4
11 = x8

Kuva 11 ADC-piirin konfiguraatiotavu

Konfiguraation kirjoittamisen jälkeen luetaan muunnettu jännite bitteinä sekä muunnetaan se määriteltyjen asetuksien mukaiseksi desimaaliluvuksi, joka esittää mitattavaa suuretta kuten lämpötilaa. Tässä vaiheessa lukuoperaatiota on luettu yksi fyysinen sensori. Nämä vaiheet toistetaan n-kertaa riippuen konfiguraatioon määriteltyjen sensorien määrästä. Lukuoperaatioiden nopeuden rajoituksena on piirin tarjoama lukutaajuus. Ohjelmisto on suunniteltu toimimaan siten, että yksi mittaus koostuu kaikkien määriteltyjen sensorien hetkittäisestä arvosta sekä aikaleimasta, jolloin mittaus on otettu. Joten jos piirin lukutaajuus on 240 lukua sekunnissa ja kytkettyjä sensoreita on 4 kappaletta, on ohjelmiston mittaustaajuus 60 mittausta sekunnissa. Yksittäiset sensorilukemat luetaan siis peräkkäin, joista muunnoksien jälkeen koostetaan yksi mittaus.

4.2 Tiedon älykäs esiprosessointi

Seuraavalla "tasolla" järjestelmässä on otettujen mittauksien esiprosessointi, jossa mittauksia keskiarvoistetaan sekä tarkastetaan poikkeustilaan oikeuttavien lukemien varalta. Yksittäisen laitteen tehtävänä on keskiarvoistaa "normaaleja" mittauksia sekä havaita ja tallentaa tihennettyä mittausdataa poikkeustilanteista, jotka tässä projektissa ovat esimerkiksi paineiskuja vesijohtoverkostossa, vesijärjestelmässä tai vedenjakelujärjestelmässä. Konfiguraatioon on määritelty keskiarvoistamisväli sekä sensorikohtaiset poikkeustilaan oikeuttavat arvot. Tyypillisesti datankeräysjärjestelmät eivät

juuri prosessoi tietoa mahdollisen datatyyppimuunnoksen jälkeen, vaan ne toimivat puhtaasti mittausjärjestelminä, jotka jättävät tiedon prosessoinnin taustajärjestelmälle. Tiedon prosessointia jo kentällä voidaan varauksin sanoa järjestelmän ”älykkyydeksi”.

Poikkeustilan tunnistamisen logiikan voi pääpiirteittäin jakaa seuraaviin osiin:

- poikkeustilan havainnointi
- poikkeustilaan oikeuttavan sensorilukeman tunnistus
- poikkeustilanteeseen liittyvän tiheän mittausdatan lokaali tallennus.

Tiedon myöhemmän analyysin kontekstissa kiinnostavaan dataan kuuluu myös mittaukset hieman ennen ja jälkeen tapahtuneen poikkeustilanteen. Tällaisesta niin sanotusta poikkeustilanteen historiatiedosta voi selvitä tapahtuiko jossain mitattavista suureista muutoksia, jotka aiheuttivat havaitun poikkeustilanteen. Poikkeustilanteesta eteenpäin sijoittuvasta mittausdatasta voidaan tutkia, aiheuttiko itse poikkeustilanne lisää kiinnostavia muutoksia sensorien mittaamissa arvoissa.


```
{
  "Device": {
    "AveragingInterval": "00:00:10",
    "AnomalyHistoryLength": "00:00:01",
    "AnomalyForwardLength": "00:00:01",
    "Modules": [
      {
        "Type": "MCP3424",
        "Bus": 0,
        "BusAddress": 108,
        "SampleRate": 240,
        "Inputs": [
          {
            "Id": 1,
            "Channel": 1,
            "Gain": 1,
            "AnomalyUpperLimit": 1000.0,
            "AnomalyLowerLimit": -1.0,
            "ConversionMultiplier": 1.0,
            "ConversionOffset": 0.0
          }
        ]
      }
    ]
  }
}
```


Kuva 12 Esimerkki ohjelmiston mahdollisesta konfiguraatiosta

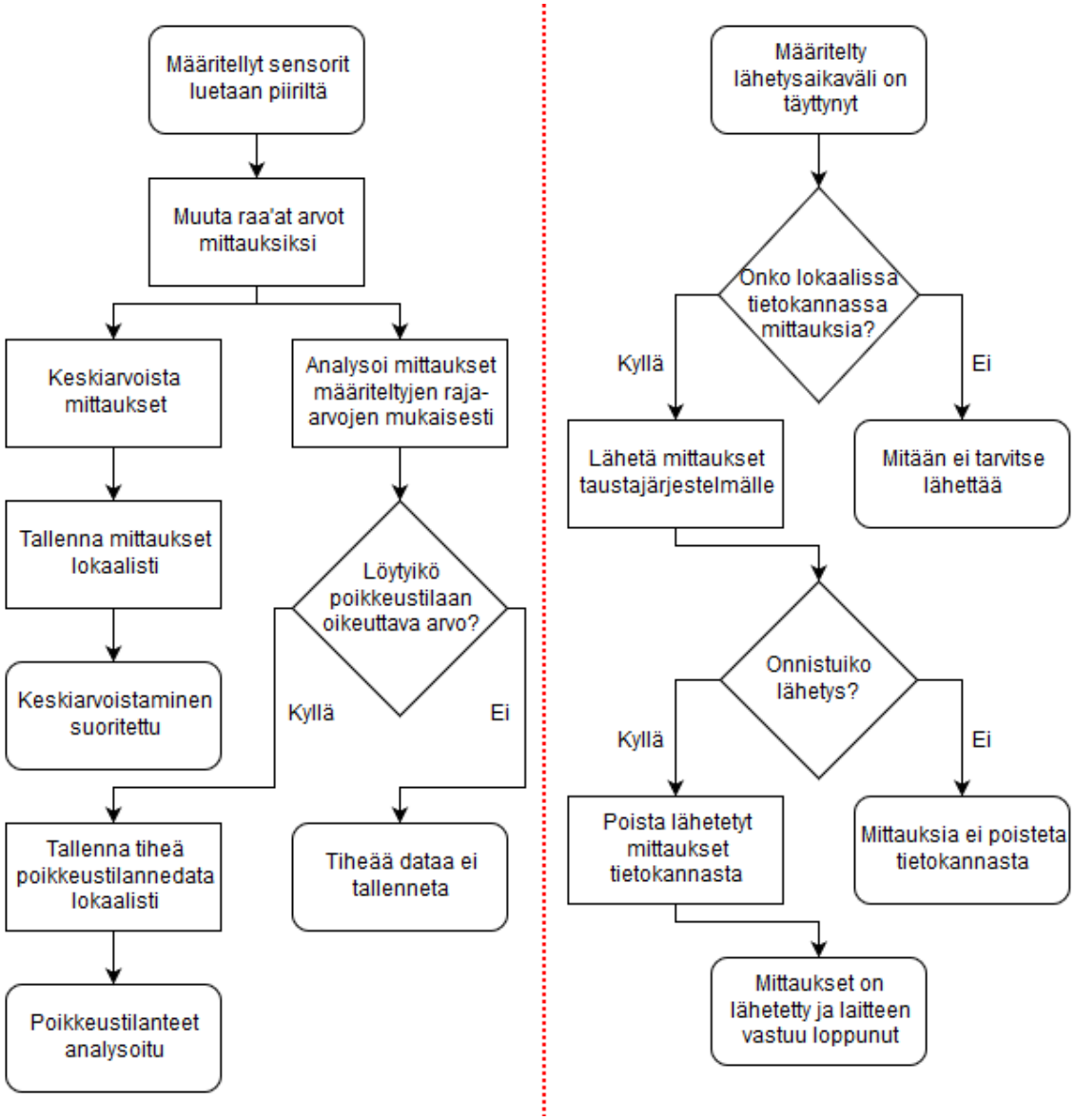
Kuva 12 havainnollistaa mahdollista, joskin yksinkertaistettua, konfiguraatiota järjestelmän ohjelmistolle. Konfiguraatiossa on määritelty edellä mainittuja parametrejä keskiarvoistukselle ja poikkeustilojen havainnoinnille. Esimerkkinä voidaan ottaa 10 minuutin ajanjakso, jossa tapahtui yksi poikkeustilanne, joka prosessoitiin kuvassa näkyvien parametrien mukaisesti. Keskiarvomittauksia otettiin yhteensä 60 kappaletta; joka kymmenes sekunti. Sen lisäksi havaittiin yksi poikkeustilanne, jonka tallennuksen laajuudeksi oli määritelty yhteensä 2 sekuntia. Tämän poikkeustilanteen, joka voisi olla esimerkiksi paineisku putkistossa, ajan otettiin 240 mittausta sekunnissa. Poikkeustilanne

on täten tallennettu 480 mittaukseen, jotka otettiin noin 4 millisekunnin välein. Kuten esimerkistä voidaan huomata; lyhyetkin poikkeustilanteet vaativat todella suuren määrän yksittäisiä mittauksia. Kuten aiemmin mainittiin; paineiskut tapahtuvat erittäin suurilla nopeuksilla, joten suuri määrä mittauksia on välttämätön ominaisuus riittävän resoluution saavuttamiseksi.

Eräs järjestelmän testausvaiheessa ilmenneistä ongelmista koski tilannetta, jossa havaittava poikkeuksellinen tilanne jatkuu pitkään. Järjestelmän ollessa suunniteltu ottamaan mittauksia erittäin suurella taajuudella poikkeustilanteissa, dataa voi kertyä massiiviset määrät lokaaliin tietokantaan. Ongelmaksi muodostuu rajoitettu tallennustila sekä suhteellisen hidas internetyhteys. Esimerkiksi tilanteissa, joissa verkko on jostain syystä alhaalla useita tunteja, dataa voi teoriassa kertyä useiden megatavujen verran. Vaikka kyseinen tilanne on melko epätodennäköinen, otettiin se huomioon lisäämällä ”älykkyyttä” poikkeustilanteiden tunnistamiseen. Poikkeustilojen kannalta kiinnostavaksi määriteltiin järjestelmää suunnitellessa vain hetki, jolloin mitattava suure ylitti määritellyn rajan. Järjestelmän konfiguraatioon voidaan määritellä arvoja, jotka estävät jatkuvien poikkeustilojen tallennuksen sekä tarjoavat hystereesin määrittelyn mahdollisuuden.

4.3 Tiedon lähetys taustajärjestelmälle

Mittausdatan prosessointivaiheen jälkeen yksittäisen laitteen tehtäväksi jää välittää analyysivalmis data eteenpäin taustajärjestelmälle. Dataa ei lähetetä suoraan prosessoinnin jälkeen, vaan sitä puskuroidaan ensin lokaalisti Sqlite-tietokantaan. Eräs järjestelmän vaatimuksista on, että lähetyksen epäonnistuessa, verkon ollessa alhaalla tai virtakatkoksen tapahtuessa kerättyä dataa ei menetetä. Ohjelmiston konfiguraatioon määritellään mittauksien lähetysväli, jolloin järjestelmä lukee tietoa, prosessoi sen, tallentaa sen tietokantaan ja yrittää lähettää sen taustajärjestelmälle määritellyn lähetysvälin mukaan.



Kuva 13 Kuvaus mittauksien luku- ja lähetysprosesseista

Kuva 13 havainnollistaa ohjelmistossa suoritettavien prosessien työn kulkua korkealta tasolta katsottuna. Molemmat näistä operaatioista sijaitsevat silmukoissa, joita ajetaan koko sovelluksen käynnissä olon ajan. Kuvasta voidaan huomata että luku- ja lähetysprosessit toimivat toisistaan itsenäisesti käyttäen rajapintana lokaalia Sqlite-tietokantaa. Käytännössä nämä prosessit on toteutettu säikeiden avulla. Mittauksia lähettävä säie tyhjentää tietokantaa sitä mukaa kun se täyttyy mittauksia lukevan säikeen tallentaessa keskiarvoja ja poikkeustilanteita. Mittauksien lähetyksen yhteydessä täytyy lähettää myös metatietoa siitä, mistä mittaukset tulevat, mitä ne sisältävät sekä mitä mittauksien sisältämät sensorilukemat tarkoittavat. Lähettämiseen liittyy myös laitteen autentikoinnin tarve, koska taustajärjestelmän rajapintana käytetään REST-tyyppistä rajapintaa. Kuva 14 esittää lähetettävää mittausta, joka koostuu järjestelmän yleisestä metatiedosta sekä yksittäisistä sensorilukemista.

```
{
    "Key": "7czYBE8M9oWF0C8QD1AE6g==",
    "Area": "Kuopio",
    "Location": "Location x",
    "Timestamp": "2019-02-28T21:06:02.4245376+01:00",
    "Data": [
      {
        "Type": "Air humidity",
        "Value": 80.464
      },
      {
        "Type": "Water pressure",
        "Value": 3.971
      },
      {
        "Type": "Air pressure",
        "Value": 0.984
      },
      {
        "Type": "Water temperature",
        "Value": 15.456
      }
    ]
}
```


Kuva 14 Esimerkki lähetettävästä mittauksesta

4.3.1 Rajallisen verkkoyhteyden tuomat haasteet

Järjestelmän internetyhteyden muodostamiseen käytetty tekniikka muodosti yhden suurimmista teknisistä haasteista projektin aikana. Järjestelmä hyödyntää PPP-protokollaa käyttääkseen edullista DNA LaiteNetti -liittymää mobiilidatan välitykseen. Liittymätyyppi on suunniteltu käytettäväksi erilaisten pienien laitteiden etäkäytön mahdollistamiseksi. Käytännössä tämä tarkoittaa erittäin pieniä datamääriä vaativien komentojen välittämistä etäkäytettäville järjestelmille. Liittymän nopeus on luokkaa 16 kilotavua sekunnissa lataukseen ja lähetykseen. Saavutettavista nopeuksista voidaan helposti päätellä, ettei liittymää ole tarkoitettu suurien datamäärien lähettämiseen tiedonkeruulaitteistolta. Järjestelmän kerätessä ja lähettäessä dataa potentiaalisesti jopa 240 Hz taajuudella, on valittu liittymä riittämätön, ellei lähetettävää dataa optimoida sekä formaatin että lähetysprotokollan tasolla.

Tämän opinnäytetyön asettamissa rajoissa päätettiin ongelmaa lähteä ratkaisemaan lähetettävän datan formaatin puolelta. Opinnäytetyössä keskityttiin kentällä olevan laitteiston ja sen ohjelmiston toteuttamiseen, eikä taustajärjestelmään ollut ideaalista tehdä muutoksia opinnäytetyön rajauksen puitteissa. Taustajärjestelmä tarjoaa http-rajapinnan mittausdatan kirjoittamiseen ja lukemiseen. Tehokkaimmaksi optimointitavaksi jäi http-pakettien sisällön optimointi. Yksittäinen datapaketti koostuu n-määrästä mittauksia, joissa on n-määrä yksittäisiä sensorilukemia, riippuen yksittäiseen laitteeseen kytketyistä sensoreista. Projektin alkuvaiheessa datapaketit muodostuivat luontaisesti Kuva 14 mukaisiksi, sisältäen suhteellisen paljon metatietoa sekä JSON-formaatin tuomia turhia merkkejä.

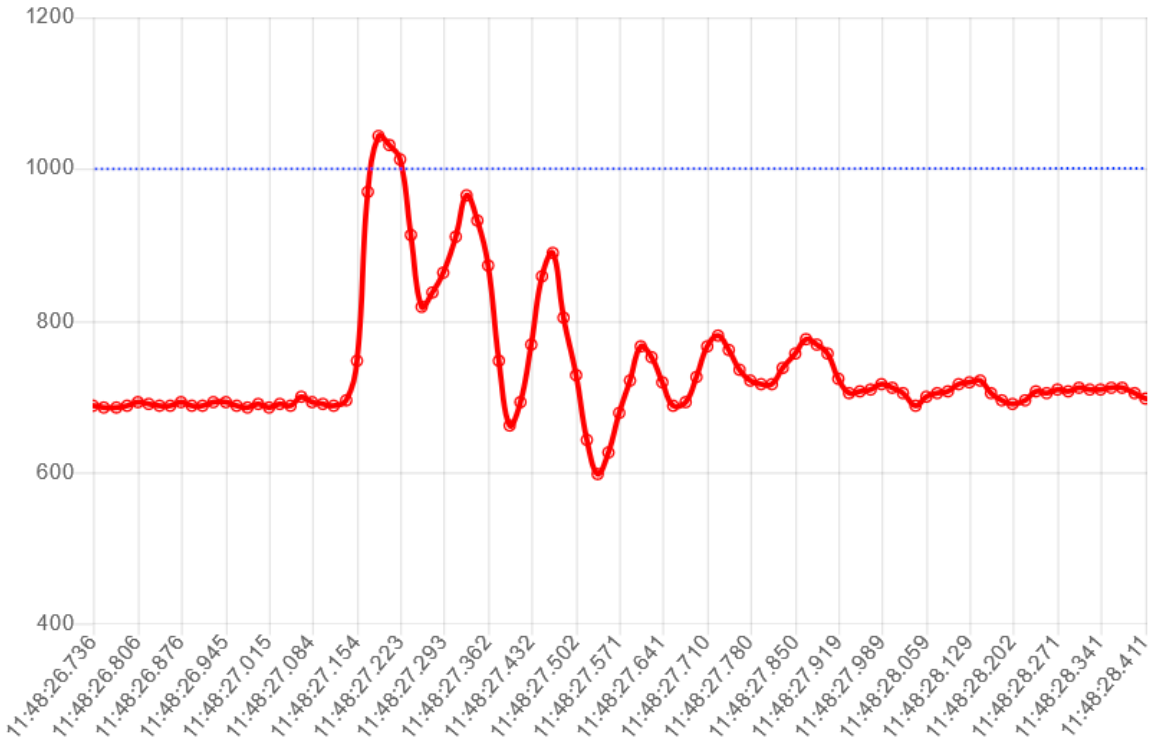
Optimointivaiheessa ryhdyttiin miettimään mikä on minimaalinen määrä lähetettävää dataa, että saadaan kommunikoitua kaikki tarpeellinen mutta ei mitään turhaa. Järjestelmältä tulleita esimerkki-datapaketteja tutkiessa huomattiin, että muuttumattomaan, ja täten turhaan dataan, kuuluu mm. jokaisessa datapaketissa lähetetty metatieto laitteesta ja siihen kytketyistä sensoreista. Ratkaisu tähän ongelmaan oli melko yksinkertainen: siirretään laitteistoon liittyvä metatieto taustajärjestelmään, jotta sitä ei tarvitse lähettää jokaisen mittauksen yhteydessä. Optimoitu datapaketti muodostui Kuva 15 mukaiseksi, josta voidaan huomata, että lähetettävään dataan kuuluu vain minimaalinen tarvittava tieto sekä itse sensorien lukemat. Kenttien arvojen formaatti on myös eräs mahdollisista optimointikohteista. Esimerkkinä aikaleima, joka voidaan tässä tapauksessa esittää Unix-formaatissa millisekunteina ajankohdasta 00.00 1.1.1970.


```
{
    "Timestamp": 1553675424617,
    "Data": [
        {
            "Value": 5.123
        },
        {
            "Value": 40.734
        },
        {
            "Value": 40.734
        },
        {
            "Value": 40.734
        }
    ]
}
```


Kuva 15 Mittauksen optimoitu lähetysformaatti

Itse lähetettävän tiedon optimoinnin lisäksi optimointia vaati myös kyseisen tiedon enkoodausformaatti. JSON on hyvä formaatti sen helppokäyttöisyyden ja luettavuuden kannalta. JSON-datasta voidaan saada irti hyödyllistä tietoa sen raakaversiossa ja jopa ilman tietoa sitä käyttävän järjestelmän luonteesta. Kyseisen formaatin hyvät puolet tuovat esille myös sen huonot puolet. Tiedon luettavuus ja helppokäyttöisyys perustuu jokaisen kentän ”avain->arvo” tyyppiseen rakenteeseen sekä selväkieliseen tekstiformaattiin.

Optimointivaiheessa päädyttiin tutkimaan binääriformaatteja, kuten käytettäväksi valittu Protobuf. Googlen kehittämä Protobuf on binääriformaatti, joka koostuu ennalta määritellyn skeeman mukaisista binääriformaatissa olevista datapaketeista. Mittaukset ovat noin 50% pienempiä Protobuf-formaattiin enkoodattuna JSON:iin verrattuna. Protobuf:in huono puoli on, että datan tulkkaamiseen vaaditaan kirjoittamiseen käytetty skeema. Ilman skeemaa tieto on vain merkityksetöntä binäärida-taa. Kuva 15 esittää lähetettävän mittauksen optimoitua formaattia ilman Protobuf-enkoodausta. Käytännössä Protobuf:ina enkoodattuna mittaukset ovat vain kenttien arvoja peräkkäin binäärimuodossa.



Kuva 16 Simuloitu poikkeustilanne

Kuva 16 esittää osaa testaustilanteessa tuotetusta poikkeustilanteesta, joka kuvaa käsiventtiiliä sulkemalla simuloitua paineiskua vesiputkessa. X-akseli kuvaa yksittäisten mittausten aikaleimoja ja y-akseli mitatun arvon suuruutta. Sininen katkoviiva esittää konfiguraatioon määriteltyä raja-arvoa, jonka perusteella poikkeustilanne havaittiin. Datan lähetyksen optimoinnin näkökulmasta voidaan huomata, että tässä poikkeustilanteessa on myös paljon tasaista dataa. Tämä tavanomainen data, jota muutenkin keskiarvoistetaan poikkeustilanteiden tunnistamisen rinnalla, on turhaa rajoitettuja resursseja kuluttavaa informaatiota. Ideaalitilanteessa tällaisen noin 100-200 mittauksesta koostuvan poikkeustilanteen jalanjälkeä voitaisiin pienentää keskiarvoistamalla myös poikkeustilanteen tasaiset arvot. Poikkeustilanteen mittauksiin kuuluisi täten tiheä data kiinnostavasta suureen vaihtelusta sekä esimerkiksi 200ms välein keskiarvoistettu data ennen ja jälkeen poikkeustilanteen. Tämä idea on yleinen tekniikka audiomaailmassa tiedostokokojen pienentämiseksi; vaativissa kohdissa audion kannalta käytetään suurempaa bitratea ja muualla matalampaa. Tekniikasta käytetään nimeä VBR eli variable bitrate.

4.4 Käytettävyyden varmistaminen

Eräs projektin IoT-luonteen ansiosta ilmenneistä haasteista on käytettävyyden varmistus. Käytettävyydellä tarkoitetaan tässä yhteydessä laitteen etäkäytön varmistaminen internetin ylitse. On todennäköistä, että järjestelmän käyttökohteet sijaitsevat hankalissa paikoissa logistiikan kannalta, joten on tärkeää varmistaa, että lähes kaikki järjestelmän ohjelmalliseen hallintaan liittyvät toimenpiteet voidaan suorittaa ssh-yhteyden välityksellä. Laite yhdistetään internetiin PPP-protokollan välityksellä, jonka luonteeseen kuuluu laitteen IP-osoitteen suuri vaihtelevuus. Jokaisella yhdistyskerralla laite saa uuden IP-osoitteen, joka täytyy kommunikoida jollain tavoin taustajärjestelmälle.

Uuden IP-osoitteen kommunikointi voidaan suorittaa monella eri tavalla, joista tässä projektissa tutkittiin seuraavia: dynaaminen DNS-palvelu, jonkin tyyppinen heartbeat-palvelu ja IP-osoitteen tallentaminen lähetetyistä datapaketeista. Valitusta metodista huolimatta, järjestelmällä täytyy olla julkinen IP-osoite, jotta siihen voidaan ottaa yhteys palveluntarjoajan verkon ulkopuolelta. Projektin internetyhteyteen käytetty liittymä on tarkoitettu mm. riistakameroiden ja muiden etähallintaa vaativien laitteiden käyttöön. Käyttökohteen luonteen takia, palveluntarjoaja tarjoaa kaksi mahdollisuutta PPP-protokollan yhteyspisteeksi, joista toinen sallii julkisen IP-osoitteen käytön. Uuden IP-osoitteen kommunikoinnin menettelyksi valittiin kunkin laitteen viimeisimmän IP-osoitteen tallentaminen taustajärjestelmään laitteen lähettämistä datapaketeista. Ratkaisu vaikutti yksinkertaisimmalta tutkituista vaihtoehdoista, eikä erillisellä ratkaisulla tuntunut olevan mitään lisäarvoa. Käytännössä jokaiselle kentällä olevalle laitteelle kuuluu taustajärjestelmään tallennettu kenttä, joka kertoo viimeisimmän IP-osoitteen kullekin laitteelle. Tämä ratkaisu kuitenkin nojaa siihen, että itse mittauksia lähettävä ohjelmisto pysyy toimintakykyisenä riippumatta siinä tapahtuneista ongelmatilanteista. Mittaus- ja verkkoyhteyssovelluksien vikasietoisuus on systemd:n varassa.

4.5 Jatkokehitys

Välittömin kehityskohde, jota on jo aloitettu opinnäytetyön rajauksen ulkopuolisena työnä, liittyy muiden sensoriprotokollien tukemiseen. Ideana on toteuttaa järjestelmä, johon pystytään liittämään mitä tahansa yleisimmistä sensoreista muokkaamatta sovellusta millään tavoin. Käyttäjän tarvitsisi vain muokata konfiguraatiota kytkettyjen laitteiden mukaan. Eräs toimeksiantajayrityksen yleisimmistä käyttökohteista järjestelmälle on erilaisten vesimittarien luenta langattomasti. Nämä vesimittarit toimivat käyttäen langatonta WM-Bus -protokollaa, joka on kehitetty erityisesti mittarien etäluentaan. Projektissa käytettyyn laitteeseen pyydettiin valmistajalta WM-Bus kommunikaation mahdollistava piiri, jolle on määritelty samankaltainen rajapinta projektissa käytetyn ADC-piirin kanssa. Erilaisten sensorien tuen implementointi vaatii kunkin protokollan määritelmään sekä sen implementoivan piirin manuaaliin tutustumista. Opinnäytetyön puitteissa toteutettu järjestelmä on kehitetty modulaariseksi juuri tämän kehitysidean seurauksena.

Vaikka taustajärjestelmän puolella tehty datan prosessointi jääkin opinnäytetyön rajauksen ulkopuolelle, on järjestelmän jatkokehityksen merkeissä tiedon analysointiin keksitty paljon mielenkiintoisia ideoita. Kuten teollisen tiedonkeruun määritelmässä mainittiin, kerätty tieto alkaa kerätä arvoa vasta analysointivaiheessa. Trendikkäät termit kuten tekoäly ja big data ovat hyvin lähellä sitä, mitä tämä projekti yrittää loppujen lopuksi tuottaa. Usein näitä termejä heitellään ikään kuin tarroina tuotteen kylkeen, jotta saadaan annettua kuva siitä miten innovatiivisia tai uraauurtavia ne ovat. Tätä projektia on kuitenkin suunniteltu ja toteutettu jo alusta asti näitä mahdollisuuksia silmällä pitäen. Järjestelmän arkkitehtuurin ollessa pääpiirteittäin toteutettu, voidaan alkaa soveltamaan näitä ideoita käytännön ratkaisuiksi.

Opinnäytetyöprosessin lähestyessä loppuaan, aloitetaan toteutetun järjestelmän laajempi testaus sekä sen tuomien mahdollisuuksien kartoittaminen tuotteistamisen kontekstissa. Työskentely jatkuu

opinnäytetyöprojektin jälkeen taustajärjestelmän kehityksen ja käytännön testien merkeissä. Järjestelmää demotaan potentiaalisille käyttäjäyrityksille samalla hankkien tietoa siitä, mihin yritykset käyttäisivät järjestelmää todellisissa tilanteissa. Järjestelmän kehitys laajenee matalan tason arkkitehtuurin toteutuksesta aina käyttäjän rajapintaan asti.

5 JOHTOPÄÄTÖKSET JA POHDINTA

Opinnäytetyön tavoitteena oli kehittää automatisoitu tiedonkeruujärjestelmä vesihuollon, kiinteistöjen ja teollisuuden vesijärjestelmien seurantaan, jolla pystytään mittaamaan erilaisia jakeluverkoissa tapahtuvia ilmiöitä. Työtä lähdettiin kehittämään aikomuksella luoda vankka pohja IIoT-paradigmaan sopivalle data-analytiikkajärjestelmälle. Opinnäytetyö saatiin toteutettua suunnitteluvaiheessa määriteltyjen vaatimuksien mukaisesti aikataulussa. Lopputuotteena valmistui demovalmis järjestelmä, joka saatiin määriteltyjen vaatimuksien ohella toteutettua lähes mihin tahansa teollisuuden käyttökohteeseen soveltuvaksi.

Kehitysprosessissa laitettiin käytäntöön mm. DAAG:in (Lueth, Patsioura, Williams, & Kermani, 2016) teettämän tutkimuksen ennustamia toimintatapoja IIoT-projektien kehityksessä. Eräs tutkimuksessa havaituista ilmiöistä liittyi yrityksien kasvavaan haluun lähteä näihin projekteihin kokeilumielessä tarkasti määritellyn mission sijaan. Opinnäytetyöprosessi näyttikin enemmän mahdollisuuksien tutkiskelulta kuin hyvin määritellyn polun seuraamiselta. Järjestelmän jatkokehityksessä aiotaan ottaa aihealueiden tuoreita tutkimuksia lähempään tarkasteluun.

Opinnäytetyössä kehitetyn järjestelmän toimivuutta on testattu onnistuneesti Savonian ympäristötekniikan tutkimuslaboratoriossa. Testauksen tuloksena saatiin konkreettista varmuutta siitä, että projektin tuotos vastaa suunniteltuja ominaisuuksia. Järjestelmää kohtaan on osoitettu kiinnostusta teollisuuden toimijoiden puolelta, ja käytännön testit on määrä aloittaa kesällä 2019. Järjestelmän kehitys jatkuu opinnäytetyöprosessin jälkeen projektin aikana kerääntyneen tietotaidon siivittämänä.

5.1 Oppiminen opinnäytetyöprosessin aikana

Opinnäytetyötä suunnitellessa ja tehdessä henkilökohtaisia haasteita löytyi varsinkin alkuvaiheessa kokemuksen ja tietämyksen puutteesta. Toteutus vaati ymmärrystä mm. projektin motiivina toimineiden teollisuuden ilmiöiden luonteesta. Toteutusta ei voitu aloittaa suoraan järjestelmään liittyvillä toimilla, vaan ensin piti tutkia vesijärjestelmien käyttäytymistä sekä niissä tapahtuvia tiedonkeruun kannalta kiinnostavia ilmiöitä. Kokemuksen puute näkyi myös melko hitaana aloituksena projektin toteutusvaiheessa. Ohjelmiston kehittäminen vaati laajaa opiskelua yksittäisten komponenttien manuaaleista aina ohjelmiston arkkitehtuuriin korkeammalla tasolla.

Projektin aihealue pysyi suurelta osin ohjelmistokehityksen piirissä, mutta ajoittain vaadittiin perustason sähkötekniikan tietämystä. Matalammalle tasolle mentäessä sähköteknisen tietämyksen vaatimukset lisääntyvät ohjelmallisen abstraktion vähentyessä. Edellinen kokemukseni ohjelmistokehityk-

sestä sijoittuu web-kehitykseen, joka on lähes toisessa ääripäässä ohjelmistojen ”tasoja” katsottaessa. Web-kehityksessä hyödynnetään lukemattomia abstraktiotasoja, jotta pystytään tuottamaan helposti käytettävä ja ominaisuusrikas ohjelmisto erittäin nopeasti. Vaikka olenkin siinä vaiheessa oppimistani, että lähes kaikki ohjelmointiin liittyvä kiinnostaa, päätin että haluan hankkia kokemusta matalamman tason ohjelmoinnista. Opinnäytetyön tekemisen myötä olen saanut ensimmäisen kosketuksen tämän tyyppiseen ohjelmistokehitykseen, enkä ole pettynyt valintaani. Matalalla tasolla työskentely on kiinnostavaa, joskin se on näyttänyt suuriakin aukkoja perusasioiden tietämyksessäni. Opinnäytetyön innoittamana olen aloittanut kertaamaan sähkötekniikan teoreettisia perustuksia järjestelmän jatkokehityksen mahdollistamiseksi.

Eräs projektin ohjelmalliseen toteutukseen kuuluneista haasteista liittyy aihealueen luonteeseen melko uutena kehitysalueena. Monet ohjelmistoalan toimijat ovat vasta aloittelemassa IoT-järjestelmien kysyntään vastaamisen. Tämän seurauksena suuri osa käytettävissä olevista ohjelmistotyökaluista on jonkinlaisessa täyttä julkaisua edeltävässä julkisessa testausvaiheessa. Tällaiset työkalut ja/tai kirjastot tukevat yleensä vain rajallista määrää arkkitehtuureja tai laitevalmistajia. Ohjelmalliset ominaisuudet on usein toteutettu vain ”proof-of-concept” -tasolla. Tavallaan ohjelmistotyökalujen rajoitettu saatavuus on myös hyödyllistä. Tilanne pakotti minut katsomaan tuttujen ja turvallisien ratkaisujen ulkopuolelle epävirallisien harrastelijaprojektien ja muuten vaikeasti saavutettavien resurssien merkeissä. Uuden kehittäminen on jännittävää, vaikkakin aina työläämpää.

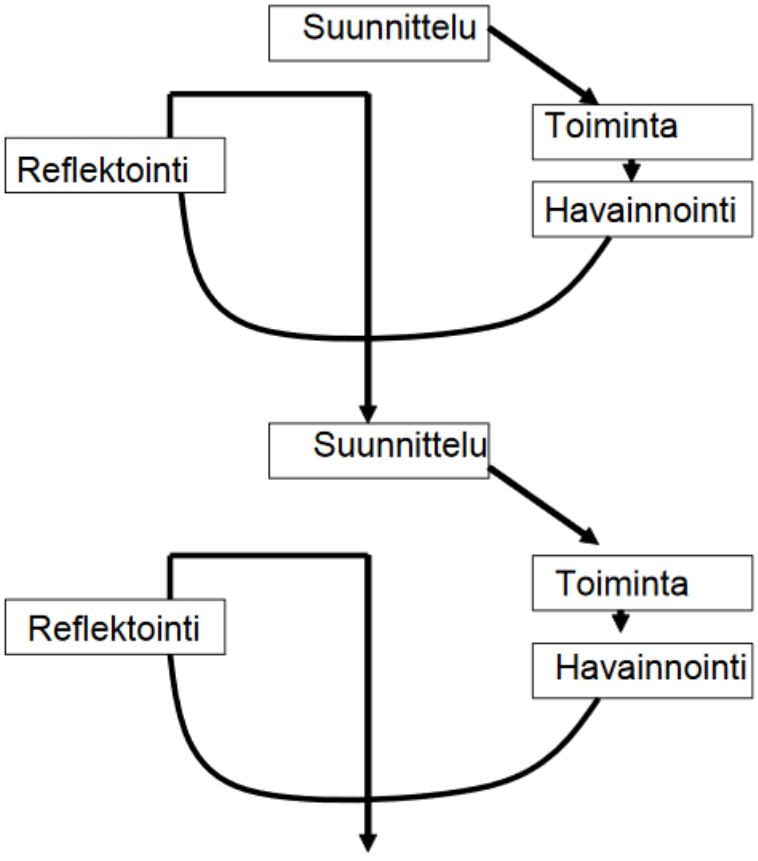
Teknisen ammattiosaamisen kasvun ohella opin paljon ohjelmistoprojektien käytännön toteutukseen vaikuttavista asioista. Mielessäni on usein naiivi uskomus siitä, että vain projektia tekevän tahon asiantuntijuus ja taito ratkaisevat miten projekti etenee. Alkuvaiheen ongelmien seurauksena ymmärsin, että tietotaidolla ei ole juuri merkitystä, jos projektin alkuvaiheessa on tehty valintoja, jotka sulkevat pois tiettyjä mahdollisuuksia. Aina ei ole mahdollista päästä mukaan kaikkiin valintoihin, jotka vaikuttavat projektin kehitysprosessiin. Eräs hyödyllisimmistä oppimistani asioista onkin joustavuus ja kompromissien tekeminen.

5.2 Opinnäytetyön kehittämisprosessi

Opinnäytetyön alkuvaiheessa tutkittiin mihin kehitystyön ”luokkaan” toteutettava projekti voisi kuulua. Työsuunnitelman tekoon liittyvässä taustatutkimuksessa selvisi, että projektin luonnetta voitaisiin kuvailla termeillä ”käytännön kehittämistyö” ja ”toiminnallinen työ”. Toiminnallisen tästä opinnäytetyöstä tekee tarkoitus kehittää konkreettinen tilaajayrityksen arvoa nostava tuote opinnäytetyöprosessin aikana (Salonen, 2013, p. 13). Toiminnallisen työn tavoitteena on käytännön ongelman ratkaisun lisäksi tuottaa uutta tietoa aiheeseen liittyen (Ojasalo, Moilanen, & Ritalahti, 2009, p. 58). Vaikka työsuunnitelmassa viitattiinkin projektiin ”tutkimuksellisena kehittämistyönä”, muodostui se opinnäytetyön edetessä muistuttamaan enemmänkin perinteistä projektityötä. Opinnäytetyössä kuitenkin lainattiin ominaisuuksia sekä tutkimuksellisesta kehittämisestä että käytännön projektityöstä.

Yksittäiset käytännön projektityöt keskittyvät pääasiassa projektin tuotoksen saavuttamiseen ilman mittavaa alan kirjallisuuteen perustuvaa pohjatyötä (Salonen, 2013, p. 11). Pääpainona on toimeksiantajan ongelman ratkaisu käytännössä. Toiminnallisesta luonteesta huolimatta projektin aikana tehtiin tutkimusta siitä, toteutetaanko jotain, mitä on ennustettu tapahtuvan projektiin liittyvien teknologioiden kehityksessä korkeammalla tasolla. ”Industry 4.0”, IIoT ja koneoppimisen sovellukset ovat eräitä paljon tutkittuja mutta vähäisesti toteutettuja alueita projektiin aihealueen kehityksessä. Eräs tutkimustyön keskeisimmistä ominaisuuksista on aihealueen tiedollisen aukon täyttämiseen pyrkiminen (Salonen, 2013, p. 10). Projektin voidaan sanoa tuottaneen uutta edellisiin tutkimuksiin perustunutta tietoa erityisesti IIoT-järjestelmien kontekstissa.

Prosessin luonnetta korkealta tasolta tarkastellessa voidaan saada hyvä kuva sen avainarvoista ja rakenteesta yleisellä tasolla. Matalammalla tasolla itse prosessi voidaan jakaa työn vaiheisiin, joista koostuu jonkin tyyppinen ideaalimalli (Salonen, 2013, pp. 15, 20). Malleja voidaan jakaa esimerkiksi niin sanottuihin lineaarisiin malleihin, joissa vaiheet seuraavat jäykästi toisiaan, tai iteratiivisiin, joissa ei ole tarkasti määriteltyjä vaiheita, vaan projektia kehitetään joustavasti ohjaten oikeaan suuntaan. Iteratiivisista kehitysmalleista käytetään myös nimitystä ”spiraalimalli” (Kuva 17).

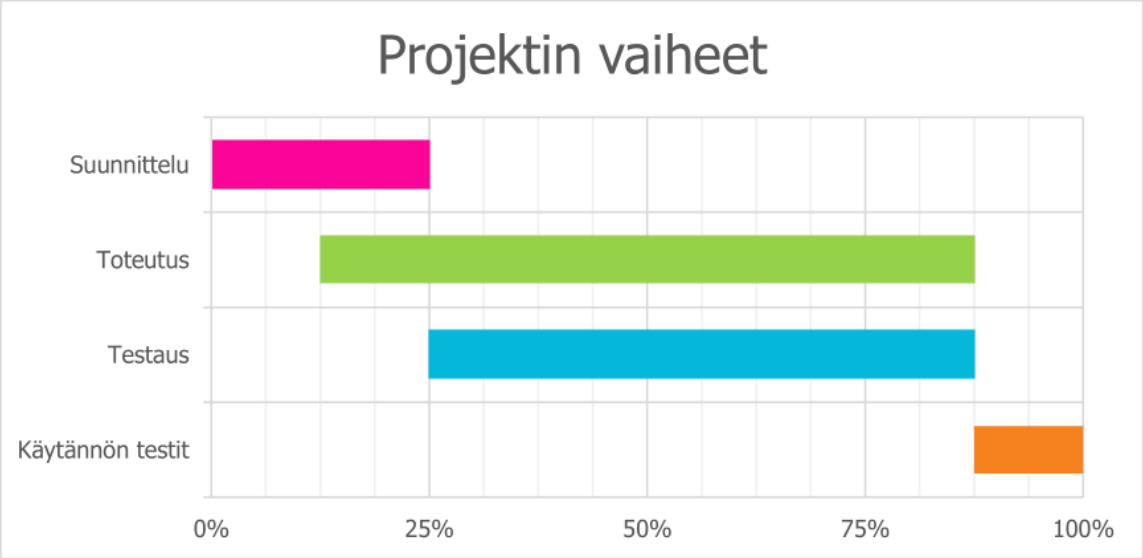


Kuva 17 Spiraalimainen kehitysmalli (Toikko & Rantanen, 2009, p. 67)

Ohjelmistokehityksen kontekstissa spiraalimainen tai iteratiivinen kehitysmalli on välttämätön vaatimuksien mukaiseen lopputulokseen pyrittäessä. Perinteisestä lineaarisesta tai niin sanotusta ”vesipu-

tousmallista" ryhdyttiin luopumaan jo 70-luvulla sen tuomien rajoituksien takia. Syynä tähän on ohjelmistoprojektien luonne erittäin dynaamisina ja vaikeasti arvioitavina prosesseina. Vaatimusmäärittelyjen muuttuminen ja niihin sopeutuminen on eräs suurimmista tekijöistä projektin onnistuneen läpiviennin kannalta. Prosessin aikana tapahtuvan oppimisen hyödyntäminen ohjaamalla projektia oikeaan suuntaan on välttämätöntä, jos kehitetään jotain, joka ratkaisee jonkin uuden ja uniikin ongelman. (Larman & Basili, 2003, pp. 47, 50)

Kuva 18 on työsuunnitelmassa havainnollistettu kuvaus projektin vaiheista. Tietämystä erilaisista kehitysmalleista ei tässä vaiheessa juuri ollut, joten tämä kuva esittää mallia, joka tuli esille luontaisesti projektia suunnitellessa. Kuvasta voidaan huomata, että malli on jonkin tyyppinen hybridi lineaari- ja spiraalimalleista. Projektille on selkeästi määritelty alku- ja loppuvaiheet, mutta itse kehitysvaihe perustuu jatkuvaan arviointiin ja testaukseen. Kuvassa esitetty "testausvaihe" tarkoittaa tehdyn työn arviointia ja testausta toimeksiantajan kanssa. Arviointia suoritettiin viikoittain, joten palautetta ja jatkokehitysehdotuksia saatiin hyvin tiheällä tahdilla. Projektin jatkuva ohjaus oikeaan suuntaan on yksi spiraalimaisien kehitysmallien suurimmista tunnusmerkeistä (Salonen, 2013, pp. 15, 16).



Kuva 18 Suunnitteluvaiheen idea projektin kulusta

Jälkikäteen mietittynä projektisuunnitelmassa kuvattu kehitysprosessi osui kohdalleen yllättävällä tarkkuudella. Alussa tehtiin pääasiassa tutkimusta ja suunnittelua, jonka seurauksena pystyttiin kartoittamaan tehtävä työ korkealta tasolta. Toteutus- ja testausvaiheen spiraalimainen rakenne toimi erittäin hyvin viikoittaisien palaverien muodossa. Vaikka viikko saattaakin tuntua erittäin lyhyeltä ajalta kerätä kasaan jotain palaverin pitämisen arvoista, pystyttiin jokaisella kerralla pohtimaan sen hetkisiä ideoita ja ongelmatilanteita ainakin jossain määrin. Pienet ohjausliikkeet koko projektin ajan kerääntyivät loppua kohden esittäytyen alkuperäistä ideaa vastaavana tuotoksena. Idea siitä, että projekti olisi toteutettu lineaarisesti vaihe vaiheelta seuraten tarkasti määriteltyä suunnitelmaa, tuntuu mahdottomalta ajatukselta verrattuna käytettyyn kehitysprosessiin.

6 LÄHDELUETTELO

Civerchia, F.;Bocchino, S.;Salvadori, C.;Rossi, E.;Maggiani, L.;& Petracca, M. (2017). Industrial Internet of Things Monitoring Solution for Advanced Predictive Mainteinance Applications. *Journal of Industrial Information Integration*, 1-21.

Daneels, A.;& Salter, W. (1999). What is SCADA? *International Conference on Accelerator and Large Experimental Physics Control Systems* (ss. 339, 343). Trieste: ICALEPCS.

Dudlik, A.;Schönfeld, S. B.;Schlüter, S.;Fahlenkamp, H.;& Prasser, H.-M. (2002). Prevention of Water Hammer and Cavitational Hammer in Pipeline Systems. *Chemical Engineering & Technology*, 888-890.

freedesktop.org. (29. 11 2018). *systemd*. Haettu 6. 4 2019 osoitteesta https://www.freedesktop.org/wiki/Software/systemd/

Gilchrist, A. (2016). *Industry 4.0: The Industrial Internet of Things.* Apress.

Harrison, R. G. (2015). Data Acquisition Systems and Initial Data Analysis. Teoksessa R. G. Harrison, *Meteorological Measurements and Instrumentation* (ss. 57-75). Hoboken: John Wiley & Sons, Ltd.

Hunt, A.;& Thomas, D. (2000). *The Pragmatic Programmer.* Addison-Wesley.

Jungreis, A. M. (1999). *US Patenttinro US6184593B1.*

Larman, C.;& Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *Computer*, ss. 47-56.

Lueth, L. K.;Patsioura, C.;Williams, Z. D.;& Kermani, Z. Z. (2016). *Industrial Analytics 2016/2017: The Current State of Data Analytics Usage in Industrial Companies.* IoT Analytics.

MEAN WELL Enterprises Co., Ltd. (12. 1 2018). *DR-UPS40.* Haettu 7. 4 2019 osoitteesta https://www.meanwell.com/Upload/PDF/DR-UPS40/DR-UPS40-SPEC.PDF

Microchip Technology Inc. (ei pvm). *MCP3422/3/4.* Haettu 6. 4 2019 osoitteesta http://ww1.microchip.com/downloads/en/devicedoc/22088b.pdf

Microsoft. (8. 1 2018). *About .NET Core*. Haettu 6. 4 2019 osoitteesta Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/core/about

Microsoft. (ei pvm). *dotnet/iot*. Haettu 6. 4 2019 osoitteesta Github: https://github.com/dotnet/iot

Milliman, P. C. (2015). Data Acquisition and Display Systems. Teoksessa M. Kutz, *Mechanical Engineers' Handbook, Volume 2: Design, Instrumentation, and Controls* (s. 1008). Hoboken: John Wiley & Sons, Inc.

Ojasalo, K.;Moilanen, T.;& Ritalahti, J. (2009). *Kehittämistyön menetelmät.* Helsinki: WSOYpro Oy.

Raspberry Pi Foundation. (ei pvm). *Compute Module 3*. Haettu 5. 4 2019 osoitteesta Raspberry Pi: https://www.raspberrypi.org/products/compute-module-3/

Raspbian.org. (ei pvm). *About Raspbian*. Haettu 5. 4 2019 osoitteesta https://www.raspbian.org/

Rokka, H. (25. 7 2018). *Edge Computing haastaa, mutta ei riko*. Noudettu osoitteesta tivi: https://www.tivi.fi/kumppaniblogit/dna/edge-computing-haastaa-mutta-ei-riko/4e845468-ed0e-37a7-8beb-ae1023cf024e

Salonen, K. (2013). *Näkökulmia tutkimukselliseen ja toiminnalliseen opinnäytetyöhön.* Turku: Turun ammattikorkeakoulu.

Sarkar, S. (2016). Internet of Things - Robustness and Reliability. Teoksessa R. Buyya;& A. Vahid, *Internet of Things - Principles and Paradigms* (ss. 201-216). Morgan Kaufmann.

Savjani, R. (7. 5 2018). *SCADA vs IoT: A comparative analysis*. Haettu 21. 4 2019 osoitteesta https://www.softwebsolutions.com/resources/scada-system-vs-iot.html

Sela, L.;Rasekh, A.;Shafiee, M. E.;& Preis, A. (2018). Characterizing pressure patterns in a water distribution network using a high-frequency monitoring system and statistical modeling.

Shi, W.;Cao, J.;Zhang, Q.;Li, Y.;& Xu, L. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal, Vol. 3, No. 5*, 637-646.

Suzdalenko, A. (2011). Guidelines for Autonomous Data Logger Design. *IEEE International Symposium on Industrial Electronics* (ss. 1426-1429). Gdansk: Institute of Electrical and Electronics Engineers.

Techbase Group Sp.z o.o. (ei pvm). *iMod X500 Industrial Computer.* Haettu 5. 4 2019 osoitteesta http://www.a2s.pl/en/imod-x500-p-7929.html

The Plumbing Blog. (2. 11 2018). *Water Hammer – How To Prevent Water Hammer*. Haettu 5. 4 2019 osoitteesta http://www.plumbingblog.co.uk/water-hammer-how-to-prevent-water-hammer/

TOI Srl. (2018). *4ZeroPlatform - TOI*. Haettu 5. 4 2019 osoitteesta http://www.thingsoninternet.it/4zeroplatform/

Toikko, T.;& Rantanen, T. (2009). *Tutkimuksellinen kehittämistoiminta.* Tampere: Tampereen yliopisto.