TAMPEREEN TEKNILLINEN YLIOPISTO

SAMI UKKONEN

HÄLYTYS- JA DIAGNOSTIIKKATIEDON KERÄÄMINEN

Diplomityö

Tarkastaja: professori Kari Systä
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 31. lokakuuta 2018

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
UKKONEN, SAMI: Hälytys- ja diagnostiikkatiedon kerääminen
Diplomityö, 45 sivua, 2 liitesivua
Marraskuussa 2018
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Kari Systä
Avainsanat: Monitorointi, Node.js, Express.js,Sqlite,PTGR,REST,GRPC,IPC

Työssä haetaan ratkaisua sopivan monitorointiohjelman etsimiseen, joka täyttää asetetut vaatimukset. Vaatimuksena on, että ohjelma pystyy kommunikoimaan REST, FTP, GRPC ja IPC kanssa. FTP kommunikoinnilla tarkoitetaan tiedonsiirtoa hakemistoon, josta kolmas osapuoli siirtää sen kohdekoneelle, missä toinen monitorointiohjelma vastaanottaa sen. Työssä tutkitaan myös oman monitorointiohjelman toteutukseen tarvittavia komponentteja, suunnittelua ja toteutusta. Sekä kuinka sellainen tehdään alustavasti ja mitä se tarvitsee, että oma toteutus voidaan tehdä. Työssä ei niinkään tutkita nopeinta monitorointiohjelmaa, vaan ohjelmaa joka soveltuu parhaiten yrityksen käyttöön. Monitorointi kustannukset eivät myöskään saa karata kovin korkeiksi.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
UKKONEN,SAMI: Collection of alarm and diagnostic information
Master of Science Thesis, 45 pages, 2 Appendix pages
November 2018
Major: Software engineering
Examiner: Professor Kari Systä
Keywords: Monitoring, Node.js, Express.js, Sqlite,PTGR,REST,GRPC,IPC

A solution is sought to find a suitable monitoring program that meets the set requirements. The requirement is that the program can communicate with REST, FTP, GRPC and IPC. FTP communication refers to a data transfer to a directory where a third party transfers it to the target machine where the other monitoring program receives it. Because the work is also studying the components, design and implementation of the implementation of your own monitoring program. Both how it is done initially and what it needs to make it happen. It is not so much the job of researching the fastest monitoring program, but the program that best suits the company, where monitoring costs do not get too high.

ALKUSANAT

Kiitokset Kari Syställe neuvoista, joita annoit opinnäytetyöhön liittyen. Kiitokset työnantajalle, joka mahdollisti opinnäytetyön tekemisen osittain työajalla ja antoi tarvittavat resurssit työn tekemiseen. Kiitokset myös perheelleni, joka antoi voimaa ja järjesti aikaa tehdä tämän lopputyön.

Versiohistoria			
Versio	**Päivämäärä**	**Tekijä**	**Muutokset**
1.0	17.3.2018	S.U.	Ensimmäinen versio.
1.1	13.7.2018	S.U.	Lisätty lähdeluettelot ja editoitu käytettäviä tekniikoita.
1.2	17.7.2018	S.U.	Tutkimus vaihtoehdoista
1.3	18.07.2018	S.U.	Alkusanat ja oman toteutuksen tekniikat
1.4	07.08.2018	S.U.	Johdantoa ja teknologia valintoja
1.5	11.08.2018	S.U.	Teknologioiden kirjoittaminen auki
1.6	08.09.2018	S.U.	Teknologioiden kirjoittaminen auki
1.7	09.09.2018	S.U.	Teknologioiden kirjoittaminen auki
1.8	10.09.2018	S.U.	Asiakasverkkojen haasteellisuus
1.9	11.09.2018	S.U.	Laitteistohaasteet
2.0	14.09.2018	S.U.	Vaatimusmäärittely kirjoittaminen
2.1	15.09.2018	S.U.	Järjestelmävaatimukset
2.2	20.09.2018	S.U.	Arkkitehtuurikuvaus
2.3	22.09.2018	S.U.	UML suunnittelukaavio, ORM ja SQL
2.4	23.09.2018	S.U.	Tapahtuma sekvenssi kaavio
2.5	25.09.2018	S.U.	Toteutus ja herra ja orja agentit
2.6	29.09.2018	S.U.	Valmiiden monitorointiohjelmistojen vertaaminen
2.7	3.10.2018	S.U.	Tietokannat ja suunnittelu
2.8	5.10.2018	S.U.	IPC -kommunikointi, tietokannat ja suunnittelu
2.9	6.10.2018	S.U.	Tietokannat ja suunnittelu
3.0	7.10.2018	S.U.	Tulokset ja niiden tarkistelu
3.1	4.11.2018	S.U.	Kolmannen kappaleen luetteloinnin purkaminen
3.2	6.11.2018	S.U.	Dokumentissa olevan tekstin korjaaminen
3.3	7.11.2018	S.U.	Dokumentissa olevan tekstin korjaaminen

3.4	8.11.2018	S.U.	Dokumentissa olevan tekstin korjaaminen
3.5	10.11.2018	S.U.	Viitteiden lisääminen ja korjaaminen
3.6	11.11.2018	S.U.	Dokumentissa olevan tekstin korjaaminen
3.7	12.11.2018	S.U.	Dokumentissa olevan tekstin korjaaminen

SISÄLLYS

KUVALUETTELO

LYHENTEET JA MERKINNÄT

PTGR	Paessler Router Traffic Grapher.
NODEJS	Palvelin puolen ajonaikainen-ympäristö.
GRPC	gRPC Remote Procedure Calls.
REST	Representational State Transfer.
FTP	File Transfer Protocol.
YARN	Pakettihallinta, nopea, luotettava ja turvallinen.
NPM	Pakettihallinta, jolla hallinnoidaan paketteja.
Sqlite	Tietokanta pieniin järjestelmiin.
Microsoft sql palvelin klusteri	Tietokanta isompiin järjestelmiin.
C#	Microsoftin kehittämä ohjelmointikieli
.NET CORE	Microsoftin kehittämä kehysjärjestelmä monialustatuella
XAMARIN	Microsoftin omistuksessa oleva mobiilikehitys järjestelmä, jonka ohjelmointi kielenä on C#.
TDD	Test-driven development: testivetoinen kehitys
Microsservice	Arkkitehtuuri, jossa monoliittinen ohjelma on hajoitettu pieniksi itsenäisiksi ohjelmiksi.
IPC	Inter-process communication
ORM	Object-relational mapping
SQL	Structured Query Language
ACP	Lapsi agentti aliohjelma
GDPR	General Data Protection Regulation
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
Agent child	Lapsi agentti
AgentMasterSlave	Agentti isäntäpalvelu
AgentSlaveService	Agentti orjapalvelu
Communication	Kommunikaatio
Monitoring	Monitorointi
Master	Isäntä
Slave	Orja
plugin	Liitännäinen

Package	Paketti
***.monitoring**	*.monitorointi
User	Käyttäjä
Longitude	Pituusaste
Latitude	Leveysaste
Watchdog timer	Vahtikoira-piiri
***.monitoring**	*.monitorointi
User	Käyttäjä
RoleHandler	Rooli käsittelijä
Connection	Yhteys
Agent	Agentti
Operator	Operaattori
Singleton	Ainokainen
Role	Rooli
RoleProtocol	Rooliprotokolla
Process	Prosessi
Transfer	Siirto
TransferStatus	Siirtotila
LevelStatus	Tasotila
Data	Tieto
Warning	Varoitus
Error	Virhe
Ok	Kunnossa
Agent	Agentti
InUseProtocol	Käytössä protokolla
AgentNodeKey	Agenttisolmuavain
Agenttiservice	Agenttipalvelu
Address	Osoite

1 JOHDANTO

Työ on rajattu niin, että kaiken ytimenä toimii oma toteutus eli taustajärjestelmä. Työhön ei kuuluu varsinaisen etujärjestelmän monitorointi, johon tuodaan taustajärjestelmästä tiedot. Tavoitteena työssä on tutkia kaupallisia monitorointiohjelmia, sekä valita oman monitoroinnin teknologia ja toteuttaa se, mikäli kaupallinen ohjelmisto ei suoraan täytä yrityksemme vaatimuksia.

Oman agentin toteutukseen olemme valinneet sellaiset teknologiat, jotka tukevat mahdollisimman hyvin monialustaista ympäristöä ja mahdollistavat sen, että voimme toimia ympäristössä, kuin ympäristössä. Tavoitteena omassa toteutuksessa on se, että tarkasteltavia elementtejä tietokoneessa pystytään laajentamaan mahdollisimman helposti niin, että on mahdollista käyttää muitakin ohjelmointikieliä, kuin Typescriptiä. Laajennuksessa käytetään sellaisia kommunikointitapoja, joita voidaan laajentaa sellaisilla komponenteilla, jotka mahdollistavat universaalisen ohjelmointikieli tuen. Useammat ohjelmointikielet antavat käyttäjälle vapauden valita, millä ohjelmointikielellä elementit tehdään ja mahdollistaa kattavan tavan tehdä elementtejä. Lisäksi ohjelmoija saa tunteen, että hän hallitsee monitorointia prosessia.

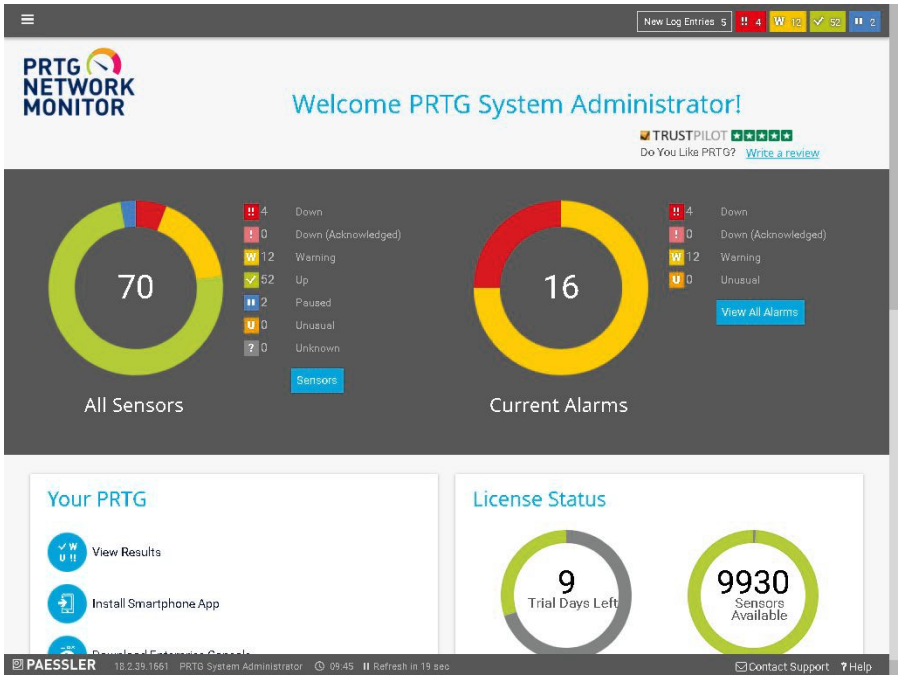
Tässä opinnäytetyössä kuvatussa järjestelmässä tietoliikenne on alhaalta ylöspäin, ei päinvastoin. Tavoitteena on mahdollisimman siisti kommunikointi. Tarkoituksena on pitää huoli siitä, että ylätason agentit eivät pääse häiritsemään alatason agentteja suoraan, vaan kaikki kommunikointi tapahtuu yhteisillä rajapinnoilla.

2 TUTKIMUS OLEMASSA OLEVISTA RATKAISUISTA

2.1 PTGR- tutkiminen

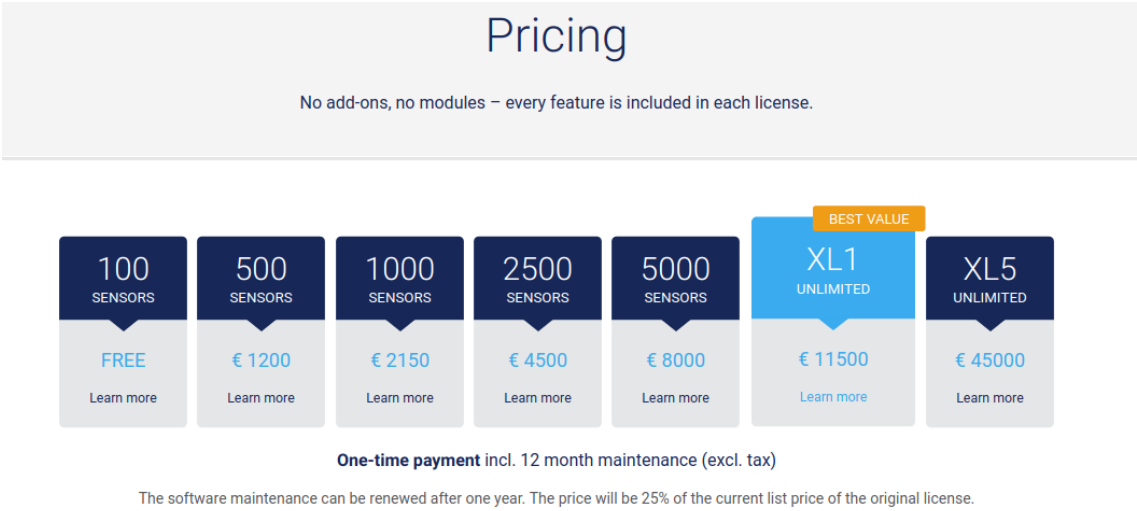
Tutkimuksiin otettiin useasta vaihtoehdosta yksi kaupallinen [3] ja yksi yritykseen räätälöity oma toteutus. Tutkimuksessa kartoitettiin kuinka kaupallinen [3] tuote soveltuisi meidän toimintaympäristöön. Toimintaympäristö on haastava, koska ympäristössä on monenlaisia tietoliikenneyhteyksiä. Tieto siirretään FTP protokollalla [22] ja REST -rajapinnan [1] kautta SAAS palvelimille, josta asiakkaalle näytetään reaaliaikaista tietoa tapahtumista. Tämän takia monitorointiohjelmalta vaaditaan, että edellä esitellyt kommunikointitavat, ovat tuettuna toimittajan monitorointiohjelmassa. Näin voimme siirtää laitteistotietoa eteenpäin ja saada asiakkaalta jo ennakkoon tietoa ongelmista.

Tutkittavana oleva kaupallinen monitorointiohjelma PTGR [3] tarjoaa kattavan liitännäisarkkitehtuurin, REST-rajapintojen toteutuksen ja erittäin hyvät lisenssivaihtoehdot.



Kuva 1. Kaupallisen monitorointiohjelmiston PTGR:n etusivu.

Lisenssin hinnoittelu (Kuva 2.) on maltillinen ja hintaan kuuluu kaikki monitorointiin liittyvät komponentit (sensorit). Muissa kuin PTGR:n monitorointivaihtoehdoissa hinnoittelu menee komponenttien mukaan. Esimerkiksi internetin monitorointi on oma komponenttinsa ja muistitilan monitorointi on oma komponenttinsa. Kun nämä kaikki komponentit lasketaan yhteen, alkaa hinta nousemaan. Asiakas joutuu kuitenkin ostamaan jokaisen tarvittavan komponentin.



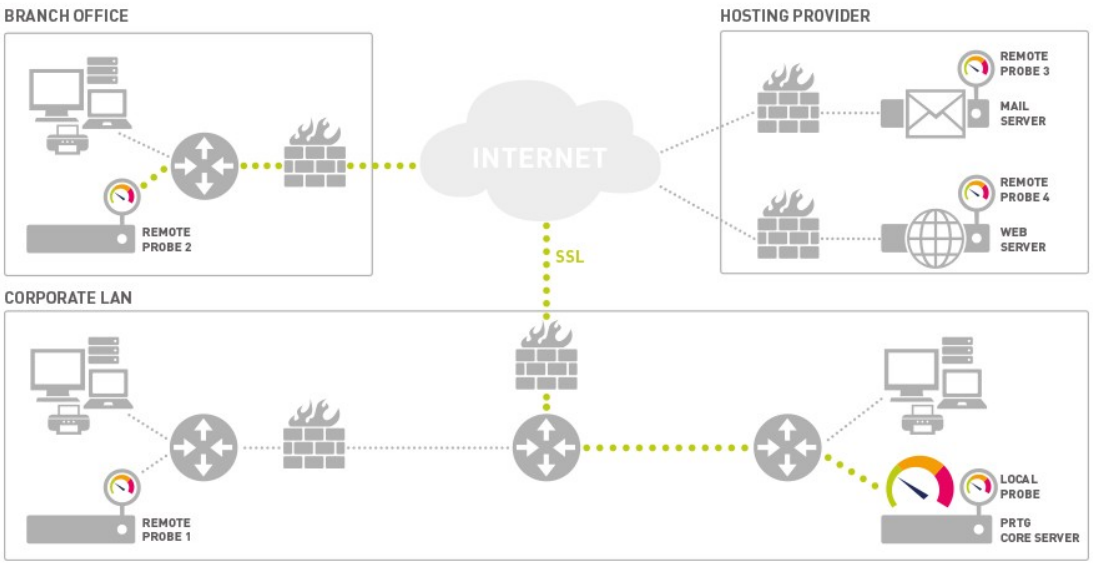
Kuva 2. PTGR- kaupallisen monitorointi ohjelmiston lisenssi hintatiedot.

PTGR:n kanssa pääsee hyvin alkuun, sillä ensimmäiset sata sensoria saa kokeiluun ilmaiseksi. Ainoa miinus puoli on, että sata sensoria on käytössä todella vähän. Sen jälkeen alkaa 14 päivän kokeiluaika, jolloin asiakkaan on ostettava sellainen lisenssi, joka kattaa sensorien lukumäärän.

Sensoreiden lukumääräksi on tavoiteltu, että niitä menisi noin 5 kappaletta yhtä tietokonetta kohden. Sensoreiden tarkoituksena on monitoroida selaimessa olevaa internet -sivua, että sivu on pysynyt paikallaan. Jos sivu on muuttunut, siitä on saatava tieto. Yksi sensori valvoo esimerkiksi prosessorin lämpötilaa tai kulutetun muistin määrää. Näistä tapahtuneista muutoksista voidaan havaita ongelmatilanteet. Viimeisenä sensorina on kiintolevyn tarkkaileminen. Kiintolevyllä on oltava riittävä määrä tilaa, jotta vältytään hidastumiselta ja muilta ongelmilta, mitä kiintolevyn liiallinen täyttäminen aiheuttaa.

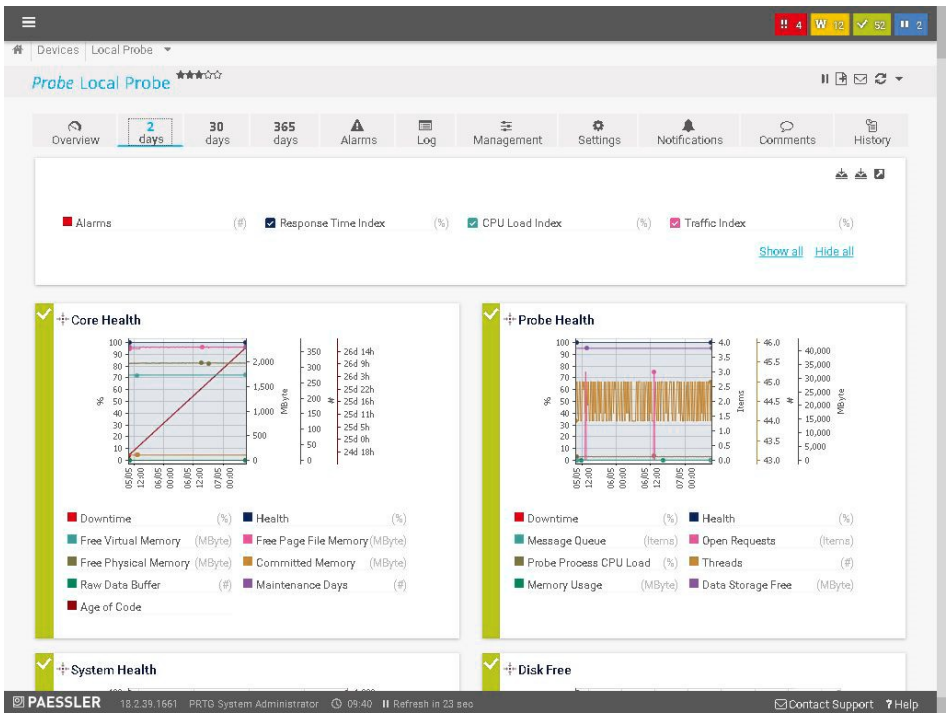
PTGR ohjelmassa on nettikäyttöliittymä, jolloin on helppo muuttaa asetuksia ja tarkkailla laitteiston tilaa. PTGR ohjelman arkkitehtuuri (Kuva 3.) on isäntä – orja tyylinen. Siinä orjat lähettävät tietoa ylöspäin isännälle. Arkkitehtuuri on suunniteltu niin, että orjat lähettävät tietoa vain yhteen suuntaan. Rekisterissä on tieto siitä, milloin tietokone on kommunikoinut isännän kanssa. Jos rekisterissä oleva tietokone ei ole hetkeen kommunikoinut isännän kanssa, nostetaan vikatila lippu pystyyn. Lipun tarkoituksena on il-

moittaa käyttäjälle mahdollisesta vikatilanteesta. Käyttäjälle ilmoitetaan joko sähköpostilla, tekstiviestillä, viestipalveluun tms.



Kuva 3. PTGR- kaupallisen monitorointiohjelmiston arkkitehtuurikuvaus.

PTGR:n isäntä internetkäyttöliittymästä voidaan tutkia tämän hetkistä, 30 päivän takaista tai jopa vuoden takaisia tilastotietoja monitoroivista orjatietokoneista. Tilastotiedoista nähdään kuinka hyvin tietokoneet ovat toimineet seurannassa (Kuvassa 4).



Kuva 4. PTGR- kaupallisen monitorointiohjelmiston kahden päivän monitorointi.

PTGR ohjelmiston haittapuoli on, että se etsii kaikki tietokoneet, jopa sellaiset joita ei olisi ajatellut edes olevan yrityksen verkossa kiinni. Tämä aiheuttaa sen, että sensorit loppuvat todella nopeasti ja PTGR menee kokeiluaika versioon. Tämän takia käyttäjä joutuu ostamaan lisenssin, joka kattaa kaikki verkon käyttämät sensorit. Ylimääräisten sensorien poistaminen on turhauttavaa ja jopa raskasta.

Tämä aiheuttaa ongelmia asiakasverkossa, jossa tietoturvan kannalta on oleellista, että mikään ohjelmisto ei ilman lupaa tutki koko verkkoa ja sen rakennetta. Tämän vuoksi ICT-osastolle koituu ylimääräistä työtä kun heidän verkon tarkistustyökalut antavat tapahtumasta heti turhia ilmoituksia, että jokin ohjelmisto tutkii verkon rakennetta.

PTGR- ohjelmisto ei tue FTP- tiedonsiirto protokollaa. Kyseisen yrityksen kanssa on ollut sähköpostikeskustelua aiheesta ja heidän mielenkiintona ei ole tehdä FTP -tiedonsiirto protokollaa tukevaa monitorointi siirtoa. Meidän ympäristössä osa asiakaslaitteistosta on vain FTP tiedonsiirto protokollan kautta kulkevaa ja saamme sitä kautta ainoastaan tiedot koneiden tilasta tai ongelmatilanteista. FTP tiedonsiirto protokolla aiheuttaa myös noin seitsemän minuutin viiveen, kun tietoa siirtyy asiakasympäristöstä meille. Näiden kahden syyn takia PTGR -ohjelmisto ei sovellu meidän tarkoitusperien mukaiseksi. Ainoa tapa saada FTP tiedonsiirto protokolla mukaan on tehdä oma räätälöity monitorointiohjelmisto.

2.2 Oman räätälöidyn toteutuksen tutkiminen

Tutkiminen aloitettiin selvittämällä, mitä teknologioita on olemassa ja missä ympäristöissä ne toimivat. Ensimmäisenä tutkittiin olisiko C# ohjelmointikielellä tehtävä toteutus soveltunut tähän, sillä ympäristöt toimivat Windows käyttöjärjestelmässä. Tämä toteutus oltaisiin voitu tehdä .NET CORE:lle [23], joka tukee useita käyttöjärjestelmäympäristöjä (Windows, MacOS ja Linux). Ongelmana oli Andoid / Ios ympäristöt joissa teknologian pitää myös toimia. Haluamme tulevaisuudessa monitoroida myös mobiilikäyttöjärjestelmiä ja saada niiden toiminnasta tietoa.

Mobiiliympäristön kehitysalustana voitaisiin käyttää C# puolella Xamarinia [24], joka on luotu mobiilialustojen kehittämiseen. Xamarin on kuitenkin maksullinen mobiilikehitysalusta ja se ei sovellu tietokoneen työpöydille käytettävien sovelluksien kehittämiseen. Tämän vuoksi toteutuksen joutuisi tekemään tietokoneen työpöydälle ja mobiiliympäristölle erikseen.

Koska kumminkin kyseessä on monialustaisen ympäristön tukeminen ei kyseeseen tule myöskään natiivinen ohjelmointiympäristö. Natiivinen ohjelmointiympäristö mahdollistaa vain yhdelle alustalle (esimerkiksi Windows käyttöjärjestelmälle) ohjelman teon. Natiivin ohjelmointiympäristön käyttäminen ei ole kannattavaa. Sen käyttöön voi liittyä useampien eri ohjelmointikielten koodipohjia ja niiden ylläpitäminen on todella haasteellista.

Tätä tukevat teknologiat toimivat tarvittavissa käyttöjärjestelmäympäristöissä ja löytyvät Nodejs:ä [25]. Tämän ympärille lähdettiin suunnittelemaan isäntä – orja pohjaista arkkitehtuuria, joka pystyy viemään FTP:n [22], REST ja GRPC:n kautta tietoa eteenpäin. Tässä tapauksessa oma monitorointiohjelma ei luo FTP:yhteyttä, vaan siirtää tiedon kansioon tietyllä tiedostopäätteellä (*.monitorointi). Kolmannen osapuolen FTP siirtoohjelma siirtää tiedoston asetetulle kohdekoneelle. Kohdekoneen agentti nappaa tiedoston vastaanottopäässä, purkaa sen ja siirtää viestin ylätasolle. Meiltä löytyy tarvittavat rakenteet jo valmiiksi ja tällätavalla päästään helpoiten kommunikoimaan asiakasympäristöissä. Ulospäin ei ole muuta mahdollisuutta, kuin FTP tiedonsiirtoprotokolla [22]. REST -rajapinta ja GRPC -protokolla ovat käytössä niissä paikoissa, joissa niitä on mahdollista käyttää. Näin mahdollistetaan nopea tiedonsiirto ylätasolle.

3 TYÖHÖN LIITTYVÄT TEKNOLOGIAT

Tässä kappaleessa esitellään kaikki ne teknologiat, joita omassa räätälöidyssä monitorointiohjelmassa tullaan käyttämään.

3.1 Pakettihallinnat

Tässä osioissa tarkastellaan pakettihallintojen vertailua ja kerrotaan kumpaan näistä varteenotettavista Yarn vai -Npm pakettihallinnasta päädyttiin.

3.1.1 Yarn vastaan Npm -pakettihallinta

Yarn on Facebookin julkaisema pakettihallintajärjestelmä Javascriptille, jonka tarkoituksena on ratkoa turvallisesti, nopeasti ja luotettavasti pakettien riippuvuuksia. Pakettien määritykset tehdään normaalisti Package.json tiedostoon, josta Yarn osaa ladata ne oikein.
Yarn voidaan asentaa kahdella eri tavalla Windows tietokoneelle. Ensimmäinen vaihtoehto on käyttää Nodejs:ä löytyvää vakio Npm pakettienhallintatyökalua. Npm pakettihallinta tulee Nodejs:n mukana. Yarn:n asentaminen Npm:n kautta tehdään Windows tietokoneen komentokehotteessa. Asennus tehdään npm install -g yarn komennolla. Komennossa kerrotaan halusta asentaa yarn käyttäen'-g' -attribuuttia. Jolloin näkyvyysalue on julkinen ja sama kuin sivun kautta ladatussa asennuspaketissa.

Toinen vaihtoehto on käydä lataamassa Yarn "https://yarnpkg.com/latest.msi" sivun kautta, mutta silloin Nodejs pitää olla asennettuna Windows tietokoneelle ennen latausta. Asennuspaketti ladataan sivun kautta ja asennetaan tietokoneelle. Asennuksen aikana valitaan, halutaanko Yarn pakettihallinta käyttää kaikissa tietokoneella olevissa poluissa. Yarn:n toimintaa voidaan rajata myös yhteen paikkaan tietokoneella. Sekä Yarn, että Npm lukevat Package.json tiedostosta asennettavien pakettien tietoja.

Tyypillisesti Package.json tiedostossa käytetään kahta pakettityyppiä. Pakettityypit ovat järjestelmästä riippuvaisia paketteja ja toinen on ohjelmointiin liittyvät pakettiriippuvuudet. Ohjelmointiin liittyviä pakettiriippuvuuksia ei viedä projektin mukana, vaan niitä käytetään projektille ohjelmointi tarkoituksiin.

3.1.2 Pakettihallintojen ongelmat ja heikkoudet

Yarn:a ja Npm:ä on molemmilla omat ongelmat ja heikkoudet. Yarn- pakettihallinnan heikkoutena on Facebook. Heillä ei ole mitään takeita siitä jatkavatko he tulevaisuudessa Yarn:n tekoa. Yarn:a on myös ongelmana, että se lähettää Facebookille asennettujen pakettien tietoja heidän palvelimille. Näin ollen he voivat tutkia minkälaisia paketteja on käytetty projekteissa.

Npm ongelmana on, että se joillakin Npm versioilla kääntyy monialustaympäristöissä ja toisella versiolla ei kännykkään. Npm pakettihallinta asentaa paketit hitaammin kuin Yarn pakettihallinta.

3.1.3 Pakettihallintojen hyvät puolet

Npm:n vahvuutena on implementointi (toteutus) Nodejs:ä ja Npm:lä on vankka käyttäjäkunta.

Yar:n etuihin kuuluu paljon monipuolisimmin ja paremmin ratkova paketti riippuvuuksien hallinta. Tällaisia riippuvuuksia tulee, kun Package.json:a määritetty Javascript paketti käyttää sisäisessä toteutuksessa useampaa Javascript pakettia.

Yarn:n ja npm:n välillä on kilpailua, joka parantaa molempien pakettihallintojen käytettävyyttä ja nopeutta.

3.1.4 Valittu pakettihallinta projektille ja tulevaisuus

Työssä päädyttiin Yarn -pakettihallintaan Npm pakettihallinnan sijaan. Jos tulevaisuudessa Yarn katoaisikin pois niin pakettihallinta voidaan vaihtaa käyttämään toista pakettihallintaa. Tietenkin pakettihallinnan vaihdon myötä voi tulla vastaan pakettiriippuvuudet, mutta nämäkin on varmasti ratkaistavissa. Pakettihallinnan vaihto tällaisessa tilanteessa on fiksua ettei projekti jää tyhjänpäälle.

3.2 Ajonaikainen-ympäristö

Työssä käytetään ajonaikaisena-ympäristönä Nodejs:ä, jonka tarkoituksena on tuottaa palvelua työn taustajärjestelmään. Sillä on keskeinen rooli agenttitoteutuksen kannalta. Kaikki projektissa käytettävät komponentit ja palvelut tarvitsevat Node.js:ä.

3.2.1 Node.js

Tietokoneelle asennettavalla Node.js:n ajonaikainen-ympäristö voidaan suorittaa palvelimella olevaa Javascripti-koodia. Node.js [25] pohjautuu Googlen Chrome V8 Javasc-

ripti-moottoriin. Projektissa käytetään Nodejs:ä, jolla on hyvä monialustatuki. Käyttöjärjestelmänä voi olla Windows, Linux, MacOs, Osx tai Android. Kyseisissä käyttöjärjestelmissä voidaan suorittaa Javascript -ohjelmointikieltä. Nodejs:ä voidaan myös kutsua tarvittaessa C++ ohjelmointikieltä. C++ ohjelmointikielen etuihin kuuluu nopeus ja sillä pääsee tarvittaessa syvempiin käyttöjärjestelmässä oleviin komponentteihin. C++:a voidaan esimerkiksi kutsua käyttöjärjestelmän resurssien tiloja. C++ ohjelmointikieltä käytettäessä on syytä tehdä rakenteet, jotka tukevat monialustaista ympäristöä.

3.3 Kehysjärjestelmä

Työssä käytetään kehysjärjestelmänä Express.js:ä. Tämän tarkoituksena on ohjata ja välittää viestejä projektin ytimessä.

3.3.1 Express.js

Pieni, nopea ja ei itsenäinen kehysjärjestelmä Nodejs:e [16]. Tällä saadaan helposti toteutettua taustalogiikka agenttijärjestelmään. Sillä voidaan hallita ja ohjata agenttien toimintaa. Hallinnassa ja ohjauksessa kirjoitetaan ja luetaan järjestelmänlokitietoja. Lokitietoja analysoimalla voidaan huomata varoitukset ja virheet, joita järjestelmä ja agentit ovat tuottaneet.

3.4 Käytettävä Javascript laajennus

Työssä käytetään Typescript laajennusta, jolla saadaan koodin rakenne mahdollisimman hierarkkiseksi ja tyypitetyksi.

3.4.1 Typescript

Typescript on Microsoftin luoma laajennusosa Javascript-kielelle [17]. Tämä tuo Javascriptistä puuttuvan tyypityksen kieleen. Tämä mahdollistaa koodissa muuttujien tyypitys tarkastuksen. Tyypityksen ansiosta ohjelmakoodissa voidaan muuttujissa käyttää omia tyypityksiä, joita on pakko käyttää ohjelmakoodissa. Tyypityksistä nähdään, mitä menee funktion attribuutteihin muuttujiksi ja mitä tyyppiä ne ovat. Myös funktion paluuarvossakin on kerrottu palautettava tyyppi. Ohjelman kääntäjä tarkastaa tyypitys-säännöt, kun ohjelmakoodi käännetään Typescript muodosta Javascriptille.

Typescript käyttää tsc-komentoa käännöksen tekemiseen. Typescriptin tsc-komentoa voisi ajatella wräpperinä. Typescript projektissa on määritetty tsconfig.json tiedosto. Tiedostossa on kerrottu esimerkiksi, mitä versiota Javascriptistä käytetään ja mihin hakemistoon Javascript käännös luodaan. Typescriptin tsc-komento tekee tarkistuksen ja

kääntää koodin Javascriptille. Koodin käännöksen aikana tarkistetaan tyypitykset, ettei niiden sääntöjä rikota. Sääntöjen tarkoituksena on estää tekstin lisäämistä muuttujaan sen ollessa kokonaislukutyyppiä. Tällöin on rikottu tyypitys sääntöä. Tyypitys säännön rikkomisesta ilmoitetaan ohjelmoijalle.

Typescript tuo myös ohjelmoijan kannalta selkeämpiä ominaisuuksia ja myös ohjelmointimallien tekeminen on helpompaa ja siistimpää. Toisen ohjelmoijan ryhtyessä tekemään projektia on hänen helpompi tutustua ja katsoa siistimpää koodia. Tällaista projektia on helpompi jatkokehittää.

Typescript ei sinänsä auta siistien ohjelmointirakenteiden tekemistä, mutta se antaa työkalut siihen. Typescriptin tuoma moduulirakenne selkeyttää ja parantaa ohjelmakoodin rakennetta.

3.5 Dokumentointi, Logitus ja testaus

Ohjelmasuunnittelussa on otettu huomioon dokumentointi, lokitus ja testaus. Työhön on tuotu ohjelmointikoodin jsdoc[8] dokumentointi työkalu. Tämän tarkoituksena on generoida ohjelmakoodin seassa olevasta kommentointiparametreista internetsivu, jossa näytetään projektissa olevat metodit, luokat yms.

Työssä käytetään lokitukseen pientä Pino[20] kirjastoa, joka generoi sattuneet virheet tai varoitukset tiedostoon. Generointi tapahtuu ohjelmakoodissa. Ohjelmoija kertoo ohjelmakoodissa mitä kohtaa lokitetaan.

Työssä käytetään Jest[6] testaus kirjastoa, jolla testataan hallintakäyttöliittymä ja taustajärjestelmä. Jokaiselle testaukselle tehdään omat tiedostot, jotka suorittavat niille määritetyt testaustehtävät.

3.5.1 Pino

Pino on erittäin pieni lokin tuottamiseen tarvittava Node.js paketti [20]. Tällä voidaan tehdä helposti JSON[41] formaatissa olevaa lokia. Lokituksen ollessa JSON formaatissa sitä on helppo esittää esimerkiksi internet-sivuilla. Tässä työssä käytettävässä Express.js kehysjärjestelmässä voidaan Pinoa käyttää lokituksien luomiseen. Esimerkiksi REST[1]-rajapintakutsuissa tai agenttien toiminnan lokittamisessa. Pino voidaan integroida Express.js kehysjärjestelmään lokittamaan Express.js:ä tapahtuvia muutoksia. Pino kirjastossa on myös prioriteettitasot, jolla käyttäjälle informoidaan vian tilasta. Pinossa on myös mahdollisuus tehdä omia muokattuja tasoja hälytyksille, jolloin se laa-

jentaa mahdollisuuksia tehdä lokituksia järjestelmästä. Pinossa voidaan informoida ongelman korjaajalle, missä tasossa järjestelmää ongelma on.

Pino oletuksena kirjoittaa rivin samaan kohtaan lokitiedostoon. Esimerkiksi työssä lokia kirjoitetaan jokaisesta tapahtumasta omalle riville. Pinoon lokitetaan kymmenen kriittisimpää ongelmatietoa joilla paikannetaan järjestelmässä olevat viat. Näin lokin koko ei kasva ja se säästää kiintolevyllä olevaa tilaa.

Pino kirjastoon löytyy monia lisäosia, joilla Pinon käytettävyyttä ja toiminnallisuutta voidaan laajentaa helposti.

3.5.2 Jest

Jest on testauskehysjärjestelmä, jonka Facebook on tehnyt React ja React-native projekteille [6]. Työssä käytetään ts-jest pakettia [5], jolla saadaan Typescriptin kautta toimimaan Jest testauskehysjärjestelmä. Raportointi on Jest:ä hyvä. Raportista nähdään hyvin työn testikattavuus ja ongelmakohdat. Ongelmakohtia työssä voi olla esimerkiksi jos yhtä funktiota on muutettu ilman testausta ja todettu sen toimivan.

Tässä projektissa käytetään TDD:ä [27]. TDD:n tarkoituksena on, että jokaiselle ohjelmoidulle komponentille on tehty testit ensin ja vasta sitten komponentin toteutus. Alussa testit tehdään aina epäonnistumaan, jotta voidaan olla varmoja testin toimivuudesta.
Kun työhön liittyvään ohjelmakoodiin tehdään muutoksia, ajetaan aina uudet testit. Nämä testit kertovat onko järjestelmä vielä kunnossa vai hajonnut. Testikattavuus pyritään pitämään alle 100 prosenttisena. Yli 99 prosentin testikattavuutta ei välttämättä saavuteta, kun ohjelmakoodi on tarpeeksi haaroittuvainen.

3.5.3 JSDoc

Työssä käytetään automaattisen dokumentoinnin luontipakettia JSDoc:a[8]. Tällä on tarkoituksena generoida dokumenttia ohjelmoitavasta koodista. Tämän tarkoituksena on helpottaa uusien ohjelmoijien perehdytystä ja vähentää manuaalisen dokumentoinnin tekemisen tarvetta. Manuaalisella dokumentoinnilla tarkoitetaan dokumentin luomista ohjelmakoodista käsin, jolloin käyttäjä joutuu tekemään itse dokumentoinnin ohjelmakoodistaan. Työssä käytettävällä JSDoc:a voidaan helposti luoda modulaarisiadokumentteja HTML[42] muodossa. HTML muodossa oleva dokumentti on helppo jakaa ohjelmistoosaajien kesken ja siitä on mukavampi katsoa ohjelmakoodissa käytettyjen komponenttien dokumentointia.

3.6 Tehtävien automatisointi ja streamaus (suoratoisto)

Työssä käytetään Gulp pakettia tehtävien automatisointiin. Tehtävien automatisoinnilla muutetaan manuaalisesti tehtävät automaattisiksi. Työssä käytettävä Merge2 streamauspaketti yhdistää useamman streamauksen yhdeksi tiedostoksi.

3.6.1 Gulp

Gulp automatisoi manuaalisesti tehtyjä tehtäviä. Gulp:a voidaan putkittaa annettuja tehtäviä [13].

Gulp paketilla voidaan esimerkiksi suorittaa useampia komentoja peräkkäin. Tällaisesta useamman komennon suorittamisesta esimerkkinä on Typescript ohjelmakoodin käännös Javascript ohjelmakoodiksi. Tästä Javascript ohjelmakoodista poimitaan JSdoc dokumentointiin tarvittavat parametrit, joista luodaan HTML dokumentti. JSdoc dokumentoinnin jälkeen käynnistetään Express.js kehysjärjestelmän palvelu. Express.js kehysjärjestelmä palvelee ulkoisessa ja sisäisessä kommunikoinnissa.

Automatisointi helpottaa manuaalisesti tehtyjen komentojen suorittamista. Automatisointi tuo selkeyttä ohjelmakoodiin. Kehittäjät näkevät yhdellä silmäyksellä suoritettavan komennon tai komentosarjat.

Gulp:n voidaan määrittää yksittäisiä tehtäviä ja näitä tehtäviä voidaan linkittää keskenään yhteen. Gulp pystyy myös tarkkailemaan esimerkiksi tiedostoja, kun niissä tapahtuu muutoksia. Tiedoston muutoksista voidaan käynnistää esimerkiksi HTML sivun renderöinti (kuvannus). Gulp:a pystytään laajentamaan lisäosilla, jotka tuovat siihen uusia ominaisuuksia. Käytännössä Gulp:n tehtävä on muuttaa manuaalisesti tehtyjä tehtäviä automaattisiksi.

3.6.2 Merge2

Merge2 yhdistää streamit yhteen tiedostoon. streamin yhdistäjää käytetään sellaisten kirjastojen kanssa, joiden metodit (toimintaketjut) palauttavat streamin. Tällä on kätevä yhdistää kaikki streamit yhteen tiedostoon. Esimerkiksi Gulp:a käyttäessä voi tällä yhdistää streamit yhteen tiedostoon. Gulp:n ja merge2:n toiminta tapa on osittain sama kuin Webpack:sa [28]. Webpack esimerkiksi niputtaa projektissa olevat kuvatiedostot, tyylitiedostot ja Javascripti tiedostot omiin tiedostolokeroihin. Esimerkiksi Webpack niputtaa Javascripti tiedostoja yhteen Javascript tiedostoon ja optimoi Javascripti tiedoston toimintaa. Tätä Webpack optimointia ei Gulp tai Merge2 yhdistelmä osaa tehdä. Jos tarvitsee tehdä vähän monimutkaisempia internet-sivuja, niin silloin kannattaa valita Gulpin tilalle Webpack.

3.7 Käyttöliittymä komponenttien vertaaminen

Työssä vertaillaan kolmen eri kehysjärjestelmän valintaa, joista yksi valitaan taustajärjestelmään. Kehysjärjestelmällä on tarkoituksena luoda taustajärjestelmä agenteille halintasivusto, jonka kautta pystytään hallinnoimaan agentteja.

3.7.1 React / Vue.js / Angular 6

Reactjs [31], Vue.js [30] ja Angular 6 [29] ovat kehysjärjestelmiä. React on Facebookin tekemä kehysjärjestelmä. React on suunniteltu komponenttipohjaiseksi. Komponenttipohjaisuudella tarkoitetaan sitä, että funktiot voidaan ajatella pieniksi komponenteiksi, joita kutsutaan HTML-tagin sisässä (liite B). React kirjastossa käytetään elementtien renderöintiin (kuvan luomiseen mallista) render() metodia. Renderöinti tarvitsee tehdä koko komponentille, kun tieto muuttuu. React:a voidaan laajentaa tarvittaessa plugineilla (lisäosilla). Angular:sa olevan kaksisuuntaisen sitomisen saa React:n tehtyä pluginilla. Kaksisuuntaisella sitomisella tarkoitetaan datamallia, joka on sidottu elementtiin. Datamallin muuttuessa se päivittää siihen sidotun elementin automaattisesti.

React:n mottoon kuuluu; ”Opiskele kerran ja käytä kaikkialla”[31]. React:n etuihin kuuluu sen selkeys ja toimivuus. React:sa on oma JSX[62] syntaksi. JSX on laajennus Javascriptille. JSX laajennusta voi verratakin pieneksi mallikieleksi, joka on täyttä Javascript voimaa.

Vue.js on vapaan lähdekoodin projekti. Vue.js kehityksen takana ei ole Facebook tai Google. Vue.js kehitystiimissä on mukana kansainvälisiä kehittäjiä. Tarkoituksena on tehdä nopea ja hyvä kehysjärjestelmä. Vue.js kehitys on helppo oppia ja sitä pääsee kehittämään heti. Hyvinä puolina tässä kehysjärjestelmässä on keveys ja sulavatoimisuus. Vue.js tukee Reactjs JSX syntaksia, mutta sitä ei ole pakko käyttää. Vue.js:ä voidaan käyttää ihan perus HTML ja CSS[43] määrityksiä käyttäen.

Angular 6 on Googlen tekemä kehysjärjestelmä. Tämä kehysjärjestelmä itsessään on jo kovin suuri ja sen mukana tulee paljon laajennoksia. Näitä laajennoksien kirjoa ei tule esimerkiksi Reacti:n tai Vue.js:n mukana. Vaan React:n ja Vues,js:n ladataan ne erikseen plugineita käyttäen. Tällaisesta ominaisuudesta on esimerkinä kaksissuuntainen sitominen, joka on valmiina Angularissa.

Angularin etuihin kuuluu kattava sisäinen laajennospaketti jota ei tarvitse ladata erikseen. Toisaalta tämä laajennoksien runsas määrä tuo raskautta ja kasvattaa projektin kokoa. Projektin ollessa pieni on Angularin käyttö liian raskasta.

Angularissa vakiona olevien laajennoksien hyvä integrointi kehysjärjestelmään vähentää pakettiriippuvuuksia. Esimerkiksi React:n ja Vue.js:n kehysjärjestelmissä tulee pakettiriippuvuuksia, koska siihen tarvitsee ladata erilliset pluginit. Saavuttaakseen samat ominaisuudet, kuin Angularissa tarvitsee plugineita ladata useampia. Plugien suuri määrä aiheuttaa myös päivitys ongelmia. Voi olla, että plugien keskinäiset riippuvuudet ovat ristiriidassa toisiinsa nähden. Tämä aiheuttaa pahimmassa tapauksessa järjestelmän hajoamisen.

3.8 Tietokannat

Työssä käytetään kahden valmistajan tietokantoja. Orja-agenttien tietokantoina käytetään Sqlite tietokantaa, joka soveltuu hyvin paikallisten orja-agenttien toimintaan. Isäntä- agentti on sijoitettu Micosoft SQL klusteri palvelimen tietokantaan. Isäntä ottaa vastaan orja-agenttien generoimaa hälytystietoa.

3.8.1 Sqlite

Sqlite[15] on itsenäisesti toimiva relaatiotietokanta, joka palvelee yhtä käyttäjää. Sqlite tietokanta on ohjelmoitu kokonaisuudessaan C-ohjelmointikielellä ja sen testikattavuus on lähelle 100 %. Sqlite- tietokantaa käytetään esimerkiksi Windows 10 käyttöjärjestelmän sisäisten tietojen ylläpidossa. Sqlite:n etuina on sen ketteryys ja avoimuus. Sqlite käyttää SQL[35]-tietokantakieltä. SQL-tietokantakieli on geneerinen (yleinen), jonka standardoitu (vakioitu) osa on tuettuna kaikilla tietokanta valmistajalla.

Sqlite tietokannan tehtävänä on kerätä tietoa orja-agenteilta. Sqlite tietokanta varastoi paikallisesti tietoa asennetuista agenteista. Tietokannassa säilytään agenttien tilatietoja ja tämän perusteella niille tehdään tarvittavia toimenpiteitä. Toimenpide voi olla se kun lapsi-agentin PID[44]:tä ei enää löydy tietokannasta ja tämä on käynnistettävä uudelleen. PID on käynnissä olevan prosessin tunniste, jolla pystytään identifioimaan (tunnistamaan) käynnissä oleva ohjelma tai prosessi.

3.8.2 Microsoft SQL palvelin klusteri

Microsoftin SQL palvelin klusteri on tietokantapalvelin hajautettuna [32]. Hajautuksella tarkoitetaan sitä että tietokantamoottori osaa vaihtaa lähdettä. Lähteen vaihto tehdään, kun ensimmäinen lähde pettää. Lähteellä tarkoitetaan palvelinta, jossa tietokannat sijaitsevat. Tässä tapauksessa tietokannat on hajautettu kahdelle palvelimelle.
Esimerkiksi kun ensimmäiselle palvelimelle tapahtuu jotain, niin silloin saadaan toinen palvelin nopeasti käynnistettyä. Toinen palvelin pitää jo valmiiksi sisällään ensimmäisen palvelimen tietokannan tiedot. Näin asiakasrajapinta ei välttämättä huomaa ongelmatilanteita.

3.9 ORM ja SQL-kyselykieli

Tässä osioissa käydään läpi sekä työssä käytettäviä laajennuksia, että SQL-kyselykielen historiaa. ORM:a[34] työssä käytetään tietokantamallien rakentamiseksi Node.js:n puolella. Näiden mallien avulla tietokantarakenteisiin pystytään kohde perustamaan tyhjästä ja malli huolehtii kaikista muista toimenpiteistä tietokantaan.

Tässä työssä SQL[35]-kyselykielellä voidaan tietokantarakenteista hakea tietoa. Sinne tallennetaan kaikki sellainen tieto, jolla pystymme hallinnoimaan kaikkien agenttien toimintaa.

3.9.1 ORM (Object-relational mapping)

ORM on tekniikka, jonka avulla voit kysellä ja muokata tietokannasta saatuja tietoja objektiivisen paradigman (ajatustavan) avulla. Objektiivisella paradigmalla tarkoitetaan, jonkin mallintamista. Esimerkki mallintamisesta on tehdä ohjelmointikielellä tyypitetty malli tietokantarakenteesta ja viedä tämä malli tietokantaan. Tyypillisesti ORM mallissa on oma API[61]. Tämän API:n kautta viedään tyypillisesti tietoa tietokantaan ja haetaan tietokannasta tarvittavaa tietoa. Voidaan ajatella, että ORM tekee virtuaalisen taulurakenteen. Virtuaaliseen taulurakenteeseen voidaan tehdä muutoksia ennen varsinaisen oikean tietokannan muutoksia. Ohjelmoijan ei tarvitse tietää SQL-kielestä mitään, sillä ORM:n API:a käyttäen se huolehtii SQL-kyselyistä. Esimerkiksi kysely ”select status from Transfer where id = 1” voidaan ORM mallia käyttäen muuttaa muotoon Transfer.findById(1).then(transfer =>{ console.log(transfer.Devicename) }). Tässä tapauksessa Transfer on ORM malli tietokantataulusta. Transfer mallista voidaan ORM:n API:a käyttäen hakea tiettyä id:tä. Tässä tapauksessa id on 1 ja se palauttaa konsoliin laitenimen.

3.9.2 SQL (Structured Query Language)

Tämä on IBM kehittämä relaatiotietokantojen kyselykieli. Kielellä voidaan hakea tietokannasta olevista tauluista tietoja. Kielellä voidaan myös päivittää tai lisätä tietoa ja lopuksi myös poistaa tietoa. SQL [35] kielestä on perusstandardi, jota kaikki tietokannan valmistajat noudattavat. Tämän standardin lisäksi on tietokannoilla omat laajennukset SQL:n päälle. Tällä tarkoitetaan sitä, että jokaisella eri tietokannalla on esimerkiksi hieman erilaiset päivämäärien käsittelyrakenteet. Mutta onneksi perusstandardissa on INSERT, UPDATE ja DELETE komennot, jotka löytyvät jokaisesta tietokannasta. Tämä helpottaa perus SQL-kyselyjen tekemistä tietokantojen välillä.

3.10 Käytettävä ORM:n Javascript -paketti

Työssä käytetään Sequelize -pakettia. Tämän lisäksi käytetään Sequelize-typescript laajennosta, jolla saadaan Typescript tuki Sequelize paketin käyttöön.

3.10.1 Sequelize

Sequelize on ORM:n[14] paketti Node.js:lle. Sequelizen tarkoituksena on automatisoida tietokannan hallintaa. Käytännössä luodaan mallit, joissa on määritetty tietokantarakenteet. Mallissa on kerrottu, mitkä taulut ovat relaatiossa (suhteessa) toistensa kanssa. Malleissa pystytään myös kertomaan, minkälaisia rajoituksia joissakin tietokanta sarakkeissa on.
Näiden mallien perusteella pystytään luomaan tietokantaan taulujen rakenteet. Kaikki määritykset ja rajoitteet mitä sequelize:lla tehdään, ei mene suoraan tietokantaan. Tietokantaan menee vain taulurakenteet ja sellaiset määritykset mitkä tietokanta voi toteuttaa. Tämä tarkoittaa, että tietokantaa ei käytetä suoraan, vaan Sequelize:n mallien kautta ohjelmakoodissa. Sequelize huolehtii, että tietojen syöttö tietokantaan menee oikein annettujen mallien kautta.

3.11 Kommunikoinnit

Työssä käytetään REST[1], FTP[22], gRPC[2] ja IPC[33] kommunikointia. Näistä REST:ä käytetään sekä sisäisessä, että ulkoisessa agentien kommunikoinnissa.

FTP tiedonsiirto protokollaa ei agentit käytä suoraan. Orja-agentti generoi tiedoston, jonka se siirtää FTP tietohakemistoon. Täältä FTP hakemistosta kolmannen osapuolen FTP siirto-ohjelma siirtää sen seuraavaan paikkaan, jossa toinen agentti (orja tai isäntä) nappaa viestin ja käsittelee sen.

gRPC kommunikointia käytetään sisäiseen ja ulkoiseen kommunikointiin. Koska gRPC on protokolla, niin on helpompi määrittää esimerkiksi palomuuriasetukset.

IPC kommunikointia käytetään vain sisäisessä kommunikoinnissa. Tällä pystytään kommunikoimaan Node.js:n sisällä oleviin agentti ohjelmiin, mutta ei eri ohjelmointikielillä tehtyjen agenttisovelluksien kanssa.

3.11.1 REST kommunikointi

REST (Representational State Transfer) [1] on arkkitehtuurimalli, joka perustuu HTTP-protokollaan. Tämä mahdollistaa tietokoneiden välisen kommunikaation verkossa. REST käyttää JSON[41] formaattia lähetyksiin palvelimen ja asiakkaan välillä. JSONin etuihin kuuluu sisällön selkeys, nopea määritteleminen ja skaalautuminen. Skaalautumi-

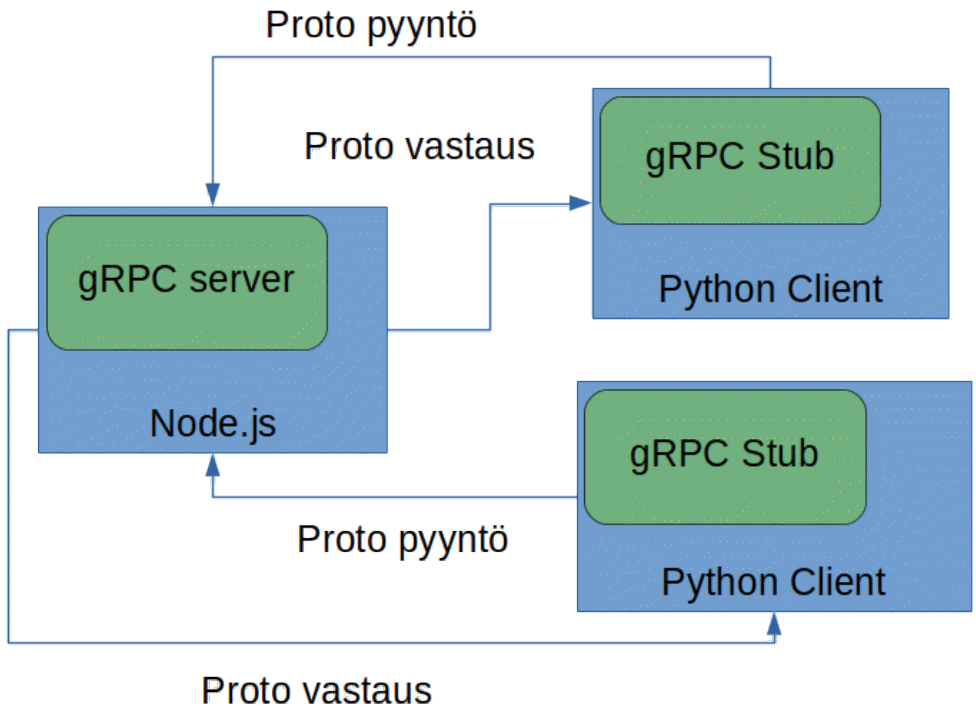
sella tarkoitetaan esimerkiksi, että jos pitää lisätä enemmän datakenttiä on niiden lisääminen nopeata ja selkeätä. Esimerkiksi SOAP:n[45] määrittelyyn menee kauemmin, koska XML rakenteissa pitää kertoa minkä tyyppistä dataa on missäkin kentässä ja tämä taas tuo joustamattomuutta. REST käyttää normaaleja HTTP metodi operaatioita, jotka ovat GET, POST, PUT ja DELETE.

3.11.2 FTP kommunikointi

FTP (File transfer protocol) [22] on TCP -protokolla kahden tietokoneen väliseen kommunikointiin. Palvelu toimii palvelimella, jossa on FTP palvelu. FTP palveluun ottaa asiakas yhteyden ja joko lataa tiedostoja tai siirtää tiedostoja palvelimelle. FTP:n historia on pitkä. FTP on kehitetty vuonna 1972. FTP:n tavoitteena oli tehdä luotettava ja tehokas tiedonsiirto mekanismi kahden tietokoneen välille. FTP voi toimia kahdessa tilassa, joko aktiivisena tai passiivisena. Aktiivinen tila hoitaa kommunikoinnin asiakkaan kanssa automaattisesti. Aktiivisessa tilassa ei odoteta, että asiakas ottaa yhteyden palvelimeen. Vaan aktiivinen tila aloittaa tiedonsiirron automaattisesti. Passiivinen tila valmistelee tiedonsiirtoa ja odottaa, että asiakas ottaa yhteyden.

3.11.3 gRPC kommunikointi

gRPC (Remote Procedure Calls) [2] on avoimen lähdekoodin kommunikointi protokolla. gRPC laajennuksen ollessa asiakasohjelmassa, voi suoraan kysyä palvelimelta olevalta ohjelmistolta metodia, jossa gPRC laajennus on mukana. Laajennuksella tarkoitetaan gRPC:n sisäänrakentamista tuetulle ohjelmakoodille. gRPC kehittäjä on Google. Googlen tavoitteena on tehdä pienen latenssin, korkeasti skaalautuvan ja hajautetun järjestelmän protokollaa laitteiden väliseen kommunikointiin. Koska gRPC on protokolla sen valvonta on helpompaa ja palomuurin gRPC protokollan lisääminen vaivatonta. Tässä protokollassa on myös otettu huomioon, että se on varma, tehokas ja tukee monia ohjelmointikieliä. Kerroksellinen muoto myös antaa mahdollisuuden lisätä laajennuksia. Laajennukset voivat olla tunnistautumiseen, kuormituksen tasapainotukseen, lokituksille ja tiedonsiirron valvontaan. gRPC:lä voi tehdä myös muitakin tarvittavia laajennuksia. gRPC on mahdollisuus käyttää monia sisältötyyppejä esimerkiksi JSON, Protobuf, Thrift, XML jne. Lisenssi on gRPC protokollassa Apache 2.0. gRPC arkkitehtuurin toimintaperiaate on kuvattu kuvassa 5. (kuvassa 5) .



Kuva 5. gRPC arkkitehtuurin toiminta periaate.

3.11.4 IPC kommunikointi

IPC[33] on rajapinta prosessien tai säikeiden väliseen informaation vaihtoon. Kommunikointi tapahtuu, joko Unix- tai Windows soketilla (verkko kommunikoinnin päätepiste). Käyttämällä Node-ipc[38] pakettia pystytään käyttämään myös, joko TCP tai UDP sokettia kommunikointiin. IPC toimii asiakas ja palvelin periaatteella. Asiakaspäästä lähetetään viesti, jonka palvelinpää ottaa vastaan ja tekee tarvittavat toimenpiteet. Tämän jälkeen palvelinpää välittää viestin takaisin asiakaspäähän. Asiakaspää lähettää palvelinpäähän viestin, että on saanut onnistuneesti viestin perille. Palvelinpään yhteyden ollessa poikki saa asiakaspää tästä tiedon.

3.12 Arkkitehtuuriratkaisu

Työssä ajatuksena arkkitehtuurista on mikropalvelin arkkitehtuuri. Tässä työssä kuitenkin toiminta periaate on monoliittinen, mutta agenttien sisäiset tarkastustoimenpiteet on mikropalvelu arkkitehtuurilla tehty. Lapsi agenttien toiminta on itsenäistä ja ne suorittavat omaa tehtäväänsä parhaiten.

Monoliittisella ohjelmalla tarkoitetaan sellaista tietokoneohjelmaa, jota suoritetaan kokonaisuudessa. Eli kun tällaiseen ohjelmaan tulee vikaa, niin koko tietokoneohjelma ajetaan alas.

Mikropalvelin arkkitehtuurissa taas ei tarvitse sammuttaa koko tietokoneohjelmaa vaan se osa siitä ohjelmasta, jota ohjelman muokkaaminen vaatii.

3.12.1 Mikropalvelin arkkitehtuuri

Tällä arkkitehtuurilla on tarkoituksena rikkoa vanhat monoliittiset ohjelmat pienimmiksi ohjelmakokonaisuuksiksi. Nämä pienet ohjelmakokonaisuudet voivat keskustella esimerkiksi REST – rajapinnan avulla toisilleen. Mikropalvelin arkkitehtuurimallissa voi olla, että jokaisella pienellä ohjelmakokonaisuudella on oma- tai yhteinen tietokanta. Näihin kahteen esiteltyyn tietokantamalliin kerätään tietoa ja haetaan tietoa.

Tällaisessa yhden tietokannan jakamisessa on se huono puoli, että jos tietokannan malliin tulee muutoksia. Tämä mallinmuutos vaikuttaa jokaiseen pieneen ohjelmaan ja voi aiheuttaa ongelmia ohjelmien toiminnassa.

Mikropalvelu arkkitehtuurilla pystytään muokkaamaan pienempiä kokonaisuuksia ilman, että häiritään kokojärjestelmän toimintaa. Myös kokonaisuuksien testaaminen on helpompaa, koska jokaista pientä ohjelmaa voidaan testata erikseen. Testaukseen voi kuulua, että palauttaako esimerkiksi nämä pienet ohjelmat oikeita arvoja.

Mikropalvelu arkkitehtuurissa voidaan lisätä pienempiä ohjelmakokonaisuuksia mukaan ja saada näin lisäominaisuuksia helposti. Tämä myös aiheuttaa ongelman. Pienempien ohjelmakokonaisuuksien lisääntyessä pitää pystyä tietämään, että mitkä pienet ohjelmat ovat päällä ja mitkä ei. Näiden pikku ohjelmien välinen kommunikointi, että mitkä pikku ohjelmat juttelevat keskenään ja mitkä ei. Lähinnä tällä tarkoitetaan esimerkiksi verkkoyhteyksien toimiminen pikku ohjelmien kommunikoinnin kanssa yms.

Tämän takia pienet ohjelmat on hajautettu esimerkiksi Dokkerilla[46] keskenään ja ne kommunikoivat REST protokollan kanssa keskenään. Pienistä ohjelmista pidetään listaa, että mitkä on hengissä ja mitkä ei. Toisaalta tällaisilla pienillä ohjelmilla on myös etunsa. Jos yksi pienistä ohjelmista kuolee, voidaan toinen samanlaisen pieni ohjelma käynnistää todella nopeasti toimintaan. Tapahtuma on yleensä huomaamaton. Huomaamattomalla tarkoitetaan, että käyttäjä ei edes huomaa vikatilannetta. Mikropalvelu arkkitehtuuria käyttää useat yritykset ja näistä voidaan mainita esimerkiksi Netflix, Amazon, Google, Microsoft jne. Tämä myös mahdollistaa rinnakkaisen ohjelmoinnin tekemisen, kun pieniä ohjelmia voi ohjelmoijat tehdä yhtä aikaa. Ohjelmoijat noudattavat pienten ohjelmien rajapintoja, jotka arkkitehtuurissa on määritetty.

4 ASIAKASYMPÄRISTÖN HAASTEET

Tässä osioissa käydään läpi asiakasverkon haasteita. Jokaisella asiakkaalla on omanlaisensa verkkoinfrastruktuuri. Osa käyttää Citrix[47] järjestelmää ja toisille tehdään VPN[48] tunneloinnilla. Ne eivät estä muuta tietoliikennettä omalta koneelta. On myös sellaisia asiakaskohtaisia Citrix[47] yritys-tunnelointeja, jotka estävät kaiken muun ulkopuolisen tietoliikenneyhteyden. Erilaiset verkkoratkaisut asettavat mukavasti haasteellisuutta. Haasteellisena voidaan pitää tiedonsiirtoa asiakasverkosta pääjärjestelmään. Asiakasverkossa on laitteet, joiden tilaa seurataan.

4.1 Asiakasverkkojen haasteet

Haasteellisena voidaan pitää asiakasverkkojen erilaisia ympäristöjä ja toimintaperiaatteita. Osassa verkoissa on VPN[48] tunnelointi asiakkaalle, osassa on Citrix[47] ja osassa on Citrix[47]:n yritys-tunnelointi työkalunsa. Citrix:n yritys-tunnelointi työkalu estää ulkopuolisen liikenteen ulospäin muilta sovelluksilta. Tällaista yhteyttä voisi kuvailla kuplaksi, josta lähtee putki asiakkaan palvelimelle ja sinä istut sen kuplan sisällä. Kuplasta ei ole muuta näköyhteyttä, kuin tunneli, jonka päässä on asiakkaan palvelin. Tämän kaltainen ympäristö tuo myös sellaisia haasteita, että kuinka saadaan tietoa liikuteltua ulospäin. Tietoa liikuteltaessa ulospäin ei saa vaarantaa asiakkaan verkkoympäristöä. Asiakas verkkoympäristön sopimuksia tietoturvasta pitää noudattaa ja heidän pelisääntöjänsä tulee kunnioittaa.

4.2 Siirto haasteet

Haasteena on, että osa tiedonsiirrosta toimii FTP[22]:n tietoliikenneprotokollan yli. Tarkkailuohjelman pitää osata siirtää agenttien tuottamat tiedot FTP[22] hakemistoon, josta kolmannen osapuolen ohjelmisto noukkii ne ja siirtää ne toiselle tietokoneelle. Tällä toisella tietokoneella pitää olla agenttiohjelma, joka osaa poimia tiedot ja tallentaa ne tietokantaan.

FTP siirroissa on myös sellaisia tapauksia, joissa FTP tiedonsiirto pitää tehdä kolmannen koneen kautta. Tästä kolmannesta koneesta on VPN[48]-tunnelointi meidän verkkoomme, josta tiedon siirto tapahtuu muulla tekniikalla kuin FTP:llä.

Lähinnä FTP:n haasteena on, että jos siirrot toimivat hitaasti voi viiveet olla yli 5 minuutin luokkaa. FTP:n hidas tiedonsiirto asettaa haasteen kuinka tulkitaan, onko tiedonsiirto virheessä vai kunnossa.

Joissakin tapauksissa kommunikointiin voimme käyttää REST rajapintaa tai gRPC protokollaa. Näiden kahden kommunikoinnin kautta voimme siirtää tarkkailuohjelmasta toiseen informaatiota ja sieltä siirtää informaatiota itse isäntä-agenttikoneeseen. Näiden kommunikointi tapojen esteenä on, että saammeko tarpeeksi oikeuksia esimerkiksi palomuurin asetuksien vaihtoon ja saammeko tämän kaltaisen tietoliikenteen toimimaan orja-agenttien ja isäntä-agenttikoneen välillä.

4.3 Laitteistohaasteet

Laitteistoympäristö on kattava ja haasteellinen. Laitteistoympäristöissä on tietokoneita, tulostimia yms. Osa asiakastietokoneista on meidän hallinnoimia. Meidän asiakasverkoissa on myös joitakin yrityksellemme kuuluvia palvelimia. On myös sellaisia tietokonelaitteita, joita hallinnoi asiakkaan ICT-tuki. Asiakkaan ICT-tuen tehtävinä on hoitaa päivitykset ja uudelleen käynnistykset. Tällaisissa moniosaisessa ympäristöissä voi käydä ihan mitä vain. Esimerkiksi tulee päivityksiä palvelimiin tai tietokoneisiin. Nämä päivitykset sitten aiheuttavat eräänlaisen ketjureaktion, joka lopettaa meidän tekemän ohjelmiston toiminnon. Päivityksien tuomat ongelmatilanteet tulevat yleensä yllätyksenä.

Tietokonelaitteisto ikääntyy ja tätä kautta tulee erilaisia ongelmatilanteita. Ongelmatilanteet voivat olla rautavikoja tai ohjelmistot vanhenee käsiin. Tällainen rautavika voi olla esimerkiksi kiintolevyn hajoaminen. Tietokoneverkoissa voi tulla viaksi muutostyöt asiakkaan hallinnoimaan verkkoon ja verkkomuutoksista ei ole ilmoitettu meidän asiantuntijoille. Tällaiset yllättävät verkkomuutokset tuovat ongelmia tiedonsiirtoon ja aiheuttavat ylimääräistä työtä. Tietokoneverkoissa oleva kytkin voi hajota, joka aiheuttaa verkon katkeamisen.

Asiakkaan verkoissa voi olla rakennettu tietoliikenneverkot langattomaksi. Tällaiset langattomat verkot voivat tuoda yllättäviä ongelmia. Esimerkiksi langattomien verkkojen antennien suuntaukset voivat olla väärin suunnattu. Tällainen väärä suuntaus tulee silloin, kun antennien pitäisi olla suoraan suunnattu. Toinen mahdollinen ongelmatilanne tulee, kun tiedonsiirto kahden koneen välillä pitäisi olla suurta ja aktiivista. Suuren tiedonsiirron tarkoituksena on, että verkkolaitteet ei tietokoneissa sammu.

Tiedonsiirrossa asiakkailla käytetään matkapuhelinverkkoja[53]. Matkapuhelinverkon yhteyksien haasteina on laitokset, jotka on rakennettu syvälle metsään. Tyypillisesti täl-

laisiin metsä paikkoihin ei matkapuhelinoperaattorit ole tehneet verkkomastoja. Matkapuhelinmastojen löytyessä voi olla, että mastojen kuuluvuus ei kata koko asiakkaan aluetta. Tällainen katvealue laitoksen alueelle tuo haasteita, koska tiedonsiirto toimii vain silloin kun verkkoon kytketty laite tulee katvealueelta pois. Tällainen vaatii hyvän tiedonsynkronoinnin, että siirrettävät tiedot ovat aina oikein.

5 SUUNNITTELUPROSESSI

Tässä osiossa kuvaillaan, mitä suunnitteluprosessissa on tehty. Osiossa kuvataan tulevan toteutuksen arkkitehtuuri.

5.1 Vaatimusmäärittely

Tässä luvussa käsitellään työhön liittyvät vaatimukset. Työssä esitellään arkkitehtuuritasolla mitä kokonaisuudessaan ohjelman pitäisi tehdä. Ohjelmassa toiminnallisena vaatimuksena on, että sen pitää lähettää kohde koneelta tilatietoa. Tilatieto kertoo, onko kohdekoneessa ongelmia.

Kommunikoinnin toiminnallisena määrityksenä on, että tuetut protokollat on FTP, REST ja gRPC. Näiden protokollien pitää tukea tiedon lähetystä ja vastaanottoa.

Aliohjelmien toiminnallisena määrityksenä on, että aliohjelmat voidaan tehdä vapaasti halutulla ohjelmointikielellä. Kuitenkin eri ohjelmointikielellä tehtyjen aliohjelmien pitää toteuttaa määritellyt rajapinnat.

Käyttöliittymän toiminnallisessa määrittelyssä on huomioitu, kuinka aliohjelmia voidaan hallinnoida. Hallinnointiin kuuluu aliohjelmien poisto, päälle laitto, pois päältä laitto ja uusien aliohjelmia lisääminen järjestelmään.

Tietokantojen toiminnallisessa määrittelyssä on otettu huomioon, että tietokanta luodaan aina ORM:n kautta. ORM:a käyttäen saadaan uusi agentti asiakkaan ympäristöön nopeasti toimintaan. Orja-agentti pystyy tämän jälkeen lähettämään ylemmälle tasolle tietokoneen tilatietoja. Tilatiedot määräytyvät aliohjelmille määritettyjen tietokone seurantojen perusteella.

Vahtikoira-piirin (Watchdog timer)[49] toiminnallisessa määrittelyssä pitää luoda vahtikoira-piiri, joka valvoo aliohjelmia ja käynnistää niitä tarvittaessa uudelleen.

Aliohjelmien toiminnallisen määrittelyn tarkoituksena on kuvata, mitä aliohjelmia kohdekoneelta valvotaan. Valvonnan piirissä pitää olla kiintolevy, muistit, internet-sivun URL[50] osoite, verkkoliikenne, ajurit ja niin edespäin. Kyseiset valvonnat on kuvattu tässä dokumentissa.

Ei-toiminnallisessa-määrittelyssä otetaan huomioon, että käyttöliittymän taustajärjestelmä toteuttaa sellaisen standardin, jossa otetaan huomioon punavihersokeus. Tällaista käyttöliittymä suunnittelua toteutetaan automaatiojärjestelmissä, joissa käytetään harmaa sävyjen toteutusmallia.

Kokonaisjärjestelmän reunaehto on, että se tukee Windows, Linux, MacOS, iOS ja Android käyttöjärjestelmiä. Näiden käyttöjärjestelmien rajoitukset on otettava huomioon aliohjelmissa. Aliohjelmat voivat käyttää sellaisia komponentteja jotka toimivat vain kyseisissä käyttöjärjestelmissä. Reunaehtona on myös, että agentteja valvoo vahtikoira-piiri. Vahtikoiran havaitessa agentin kaatumisen tai jumittumisen on sen tehtävänä käynnistää aliohjelma huomaamattomasti uudelleen. Tarkkailukoira lähettää tiedot kaatumisesta ylemmälle tasolle.

5.2 Laitteistovaatimukset

Tässä osioissa kuvaillaan tarvittavat laitteistovaatimukset. Laitteistovaatimuksella tarkoitetaan, että minkälaisilla laitteilla agentit ja ohjelmistot toimivat.

5.2.1 Tietokoneiden verkko

Koneista pitää löytyä verkkokortti CAT 5 ja internet-yhteys. Verkon vasteajat riippuvat siitä millä yhteydellä ollaan kiinni. Yhteydet voivat luoda esimerkiksi Ethernet[51], valokuitu[52] tai matkapuhelinverkon kautta. Kommunikointi tavan ollessa REST tai gRPC on yhteyden vasteaika oltava alle 1 minuutin luokkaa. Kommunikointitavan ollessa FTP ja kolmannen osapuolen siirto-ohjelma käytetty FTP:n vasteaika luokkaa alle 7 minuuttia. Vasteajat ovat FTP siirroissa suuret, koska paketit menevät monen palvelimen kautta.

Laitteistovaatimuksissa on, että käytetään verkkokortin valmistajan ajureita.

5.2.2 Matkapuhelimen verkko

Matkapuhelimessa olisi hyvä olla internet-yhteys ja yhteys voi olla 3G tai parempi. Yhteydestä riippumatta vasteaika voi olla alle 2 minuuttia tai nopeampi.

5.2.3 Työasemat

Suosittelemme, että koneet ovat standardimallia. Standardimallilla tarkoitetaan, että jokaisessa koneessa on samat komponentit. Tämä tuo varmuuden, että jokaisessa koneessa

toimii samat sovellukset. Tämä poistaa epävarmuuden siitä, että ohjelmat ei toimisikaan oikein.

Vähimmäisvaatimukset työasemille

- Suoritin 1000 Mhz tai parempi
- Keskusmuistia 2 GB

Enimmäisvaatimus keskusmuistista nousee, kun koneelle asennetaan SQLite tietokannan lisäksi agentti- ja muitakin sovelluksia. Esimerkki muista sovelluksista on MSSQL tietokanta, joka vaatii keskusmuistia toimiakseen sulavasti.

- Vapaata levytilaa yli 15 GB

Vapaata levytilaa suositellaan yli 15 GB. Työasemalle asennetaan muitakin ohjelmistoja, kuin agentti. Esimerkiksi MSSQL -tietokanta voi olla työasemalle asennettu ja työasemalla on lokituksia, jotka tallennetaan levylle. Nämä vievät luonnollisesti levytilaa.

- Näyttö ja näytönohjain

Työasemassa pitää olla kosketusnäyttö. Työasemaa käytetään sellaisessa ympäristössä, jossa kosketusnäytön kautta operoidaan tietokonetta. Resoluutio koneessa pitää olla vähintään 1024*768 pisteen tarkkuudella tai saa olla myös parempi.

- Suojausluokka

Tietokoneiden suojausluokka on IP64. Tietokoneet on pääasiallisesti sijoitettu ulkokäyttöön ja koneissa oleva kosketusnäytön suojakalvo pitää olla suoralta auringonpaisteelta UV[54] -suojattu.

5.2.4 Palvelimet

Palvelimet voivat olla fyysisiä tai virtuaalisia. Niissä pitää olla mahdollisuus lisätä suorittimia ja keskusmuistia tarvittaessa.

- Suorittimen teho 1 GHz tai parempi.
- Vapaata keskusmuistia 4 GB tai enemmän.

Keskusmuistin määrä suositellaan olevan suurempi, kuin 4 GB tai voi olla suurempikin. Palvelimella tyypillisesti pyörii MSSQL tietokanta ja muita sovelluksia, jolloin muistinmäärä on suotavaa olla suurempi, kuin 4 GB.

- Vapaata levytilaa vähintään 150 GB.

Levytilan pitää olla alustettu Windows käytössä NTFS[55] -tyyppiseksi, Linux käytössä EXT4[56] ja Apple käytössä APFS[57] (Apple File System).

- Palvelimessa näyttö ei ole pakollinen, mutta näytönohjain on.

5.2.5 Mobiililaitteet

Mobiililaitteiden vaatimuksena on, että laitteet ei ole kolmea vuotta vanhempia. Laitteiden ollessa uudempia saadaan varmuus yhteensopivuudesta agentti ohjelmiston kanssa.

- Suorittimen teho 1 Ghz tai parempi.

- Vapaata keskusmuistia 2 GB tai enemmän.
- Vapaata levytilaa 4 GB tai enemmän.

5.3 Ohjelmistovaatimukset

Tässä osiossa käydään läpi tietokoneiden ohjelmistovaatimukset ja mitä niistä pitää löytyä, jotta järjestelmä toimii oikein.

5.3.1 Yleiset vaatimukset työasemaohjelmistolta järjestelmässä

- Käyttöjärjestelmä Windows 7, Windows 10, Linux tai MacOS.
- Selaimen versio Firefox 37.1 tai Chrome.
- Tietokantana Sqlite ja MSSQL 2014. Molemmat pitää olla tarvittaessa asennettuna.
- Nodejs, Yarn ja Typescript pitää olla asennettua koneelle, jotta Packagen.json määritellyt paketit voidaan ajaa kohde tietokoneelle.

Huom! Rajoituksena on, että omat vanhat laiteajurit toimivat Windows ympäristössä. Tulevaisuudessa omat laiteajurit tulevat toimimaan Linux:a tai MacOS:ä. Tällä hetkellä Linux ja MacOS ei ole tuettuna.

5.3.2 Yleiset vaatimukset palvelin järjestelmässä

- Käyttöjärjestelminä voi olla Windows Server 2012, Linux tai MacOS.
- Käyttöjärjestelmä versio Windows serveristä voi olla uudempi, kuin 2012 versio.
- Selaimen versio Firefox 37.1 tai Chrome.
- Tietokantana Sqlite ja MSSQL 2014. Molemmat pitää olla tarvittaessa asennettuna.
- Nodejs, Yarn ja Typescript pitää olla asennettua koneelle, jotta packagen.json määritellyt paketit voidaan ajaa kohde koneelle.

Huom! Rajoituksena on, että omat vanhat laiteajurit toimivat Windows ympäristössä. Tulevaisuudessa omat laiteajurit tulevat toimimaan Linux tai MacOS käyttöjärjestelmissä. Tällä hetkellä Linux ja MacOS ei ole tuettuna. Seuraavana rajoituksena pitää myös sekin huomioida, että kaikki koneet ei ole meidän palvelimia. Palvelimet ovat asiakkaan palvelimia ja näihin pitää pyytää lupa ohjelmistoasennuksille.

5.3.3 Yleiset vaatimukset tietokantapalvelimelta

- Käyttöjärjestelmä Windows palvelin 2012 tai uudempi.
- Microsoft SQL klusteri 2014 tietokanta asennettuna.

Huom! Tietokanta palvelimeen tarvitaan yhteyksien avaukset, jotta ORM:n kautta luodut tietokanta muutokset onnistuvat oikein.

5.3.4 Yleisimmät vaatimukset mobiililaitteistolle

- Matkapuhelimen käyttöjärjestelmänä voi olla Applen iOS tai Googlen Android.
- Riippuen puhelimen käyttöjärjestelmästä voi selain olla Chrome tai Safari.
- Nodejs, Yarn ja Typescript pitää olla asennettua mobiililaitteelle, jotta Packagen.json määritellyt paketit voidaan ajaa kohde mobiililaitteelle.

Huom! Mobiililaitteen Node.js, Yarn ja Typescript asennuksissa voi tulla ongelmia, jos edellä mainittuja paketteja ei voida suoraan asentaa. Ongelmat pitää selvittää ja ratkaista ne. Ratkaisu pitää olla käyttäjän kannalta helppo, jotta ohjelmiston käyttöönotto olisi helppo.

5.4 Arkkitehtuurikuvaus

Agenttijärjestelmän peruskuvassa (Kuvassa 6.) on esitelty, miten agenttijärjestelmän taustajärjestelmä toimii. Työssä tehdään vain taustajärjestelmä osuus. Työstä jää pois käyttöliittymän monitorointi ja käyttäjähallinta, jotka eivät kuulu tämän työn piiriin. Järjestelmän isäntätaso toimii Microsoftin SQL Server klusterin päällä. Isäntä koneella toimii agentti-isäntäpalvelu, joka poimii alatasolta tulleita hälytyksiä ja vie ne Microsoft SQL palvelimen klusteri tietokantaan. Orja-agenttien kommunikointi tapahtuu FTP, gRPC ja REST:ä. Tässä tapauksessa REST kommunikoi JSON:n- avoimen standardin tiedostomuotoa käyttäen.

5.4.1 FTP tietoliikenneyhteydet asiakasverkoissa

FTP tietoliikenneyhteydet on asetettu asiakaspalvelimelta yrityksen palvelimille. Tällainen FTP tietoliikenneyhteys voi kiertää useammankin palvelimen kautta, ennen kuin saapuu yrityksen palvelimelle. Kolmannen osapuolen siirto-ohjelma siirtää tiedostot FTP:n kautta. Tiedostot nimetään *.monitorointi, jolla tunnistetaan ne monitoriohjelman lähettämiksi. Tämä auttaa tiedostojen käsittelyssä, kun monitoriohjelma tietää mikä tiedosto kuuluu sille. *.monitorointi tiedoston sisällä on JSON -dokumentti. JSON dokumentissa kerrotaan, tarvittavat tiedot kohdekoneiden järjestelmän tilasta.

5.4.2 Agentti-orjapalveluohjelma

Yhteyksiä pitää yllä agentti-orjapalveluohjelma, joka kerää tiedot alatason lapsi-agentti aliohjelmille. Agentti-orjapalvelu voi myös odottaa alemman tason koneilta tiedon tuloa. Tämä tiedon odottaminen tarkoittaa, että se osaa odottaa hitailta FTP yhteyksiltä viestin tuloa ilman virhetilaa. Agentti-orjapalvelu osaa myös tutkia meneekö siirtotiedostojen vastaanottoon liikaa aikaa, jolloin se osaa informoida ylätasolle verkko-ongel-

mista. Tästä voidaan päätellä, minkä asiakkaan ongelma on akuutti verkko-ongelmien suhteen.

5.4.3 ORM ja tietokanta

Sqlite tietokantaan generoidaan ORM:n kuvauksen avulla tietokanta taulut. ORM:sa on määritetty rajapinta parametrit. Rajapinta parametreilla tarkoitetaan sellaisia parametreja, joilla estetään ohjelmoijaa lisäämästä tietokantatauluihin vääriä tietoja. Väärät tiedot voivat rikkoa taululiitoksia tai ne rikkovat tietokantatauluun tehtäviä sääntöjä muutoksia, joilla yritetään pitää tiedot eheänä tietokannassa. Näin voidaan luottaa, että tietokantatauluissa on vain eheää tietoa.

5.4.4 Lapsi-agentti ja tietokanta

Sql:lite tietokannassa on määritetty alustavat lapsi-agentit, jotka alkavat heti keräämään tietoa tietokoneen järjestelmästä. Ylläpitäjä voi hallita internet-käyttöliittymän kautta näitä lapsi-agentteja. Agenttien toimintaa voidaan myös seurata internet-käyttöliittymän avulla. Lapsi-agentti aliohjelma voi seurata myös tietokoneen tulostinta. Tulostimelta saadaan tietoa, onko tulostin verkossa ja onko siinä jokin vikatila päällä. Lapsi agentti aliohjelma voi tutkia esimerkiksi kiintolevy:n tilaa, muistin käyttöä ja niin edespäin.

5.4.5 Lapsi-agentti, sisäinen kommunikaatio ja ohjelmointikielituki

Lapsi-agentti aliohjelma käyttää sisäiseen kommunikointiin gRPC, REST:ä ja IPC:ä [33]. Sisäinen kommunikointi on agentti-orjapalvelun ja lapsi-agentti aliohjelman välillä käytävää. Ohjelmointikielituki lapsi-agenttien aliohjelmien tekemiseen ei ole rajoitettu Javascriptiin. Lapsi-agentti aliohjelmien tekoon voit käyttää muitakin ohjelmointikieliä, kunhan se toteuttaa agentti-orjapalvelun päässä olevan rajapinnan. Näin järjestelmää pystytään laajentamaan muillakin ohjelmointikielillä, kuin yhdellä ainoalla. Tarvittaessa toisella ohjelmainkielellä päästään syvemmin järjestelmän resursseihin kiinni. Esimerkiksi Windows järjestelmissä on C#:pi ohjelmointikieli, jolla päästään syvemmin kiinni sen järjestelmän resursseihin kiinni. Myös Windows:ta löytyvällä Powershell:lä[59] pystyy luomaan lapsi-agentti aliohjelmia, joilla tutkitaan järjestelmää syvemmin.

5.4.6 Lapsi-agentti ja OS X

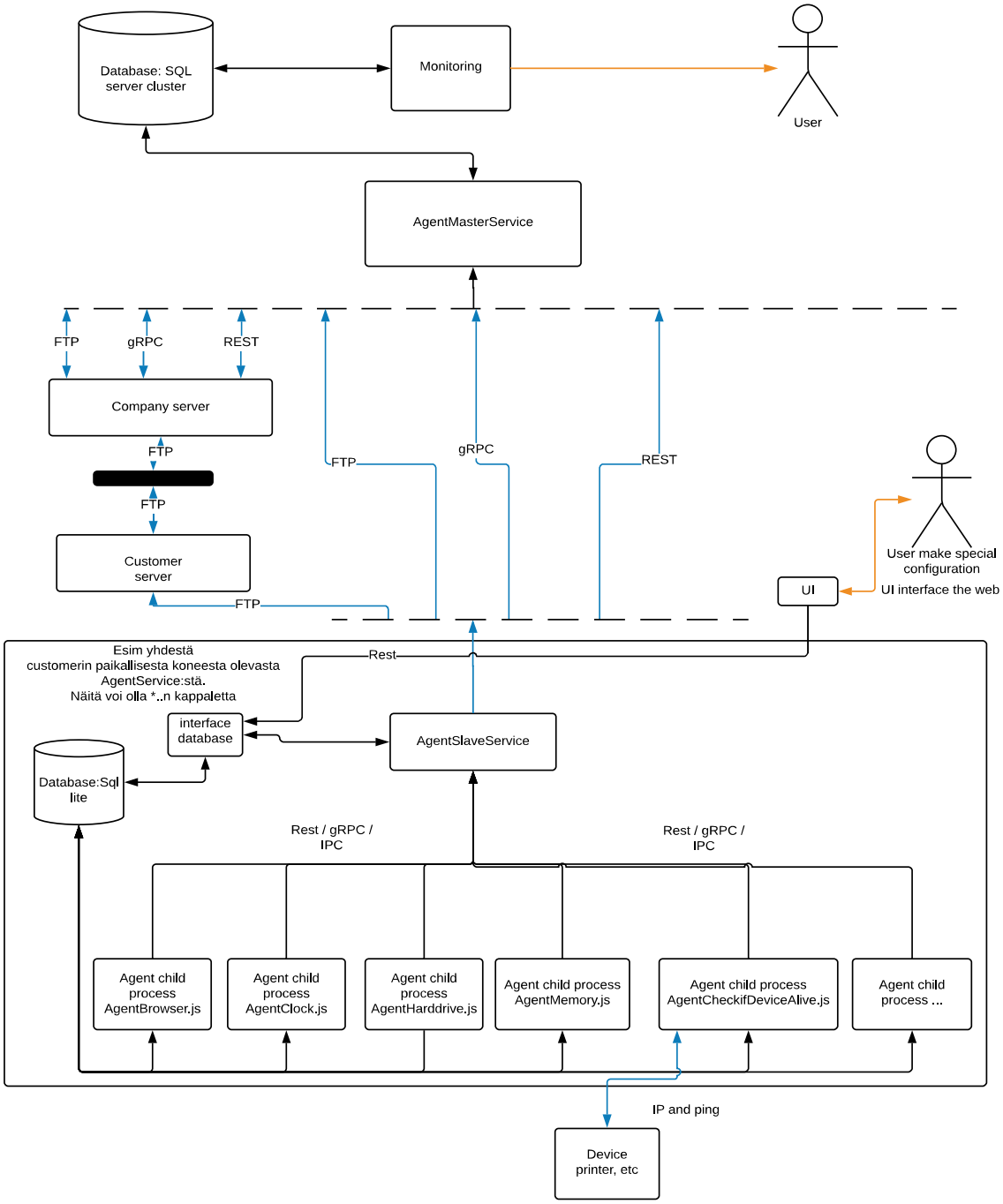
OS X järjestelmissä voidaan käyttää Swift-ohjelmointikieltä, kun halutaan syvemmin päästä resursseihin OS X järjestelmässä kiinni. Toinen ohjelmointikieli on Object-C, joilla OS X järjestelmässä päästään syvemmin resursseihin kiinni. Linux ja OS X koneissa toimii Bash[58]. Bash:la voidaan tehdä myös aliohjelmia, jotka hakevat haluttua tietoa järjestelmästä ja lähettävät niitä agentti-orjapalvelulle.

5.4.7 Lapsi-agentti ja Linux ympäristö

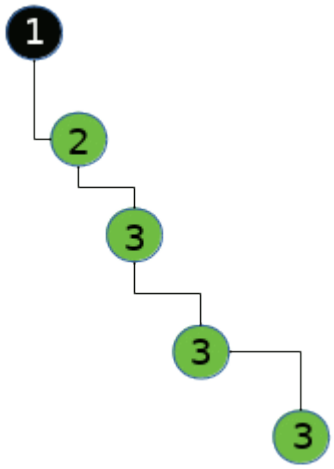
Linux ympäristössä pääsee monellakin ohjelmointikielellä tähän järjestelmään kiinni. Lapsi agenttien aliohjelmien teko on siitä kiinni, onko ohjelmointikieli tukea Linux ympäristöön. Linux käyttöjärjestelmälle on vuosien saatossa tullut paljonkin tukea ohjelmistokehitykseen ja siihen löytyy hyvät rajapintadokumentaatiot. Rajapintadokumentaatiossa kerrotaan, kuinka Linux ympäristössä pääsee laiterajapintaan kiinni ja pystytään kysymään esimerkiksi kiintolevyn täyttöastetta.

5.4.8 Agentti-orjapalvelu ja sen toiminta

Agentti-orjapalvelu pyörii jokaisessa alatason orjakoneessa. Jos asiakkaalla on vaikka kolmella tasolla tietokoneita, niin näistä tehdään puumainen kuva rakenne (Kuvassa 7.).



Kuva 6. Arkkitehtuurikuvaus on toiminta kuvaus agenttijärjestelmässä.



Kuva 7. Isäntä ja orja agenttien rakenne on puumainen.

5.5 UML suunnittelukaavio

Työnsuunnittelussa on huomioitu komponenttiajattelumalli. Jokaisella komponentilla on oma tarkoituksensa ja ne toteuttavat sille asetetut tehtävät. Työssä käytettävät komponentit ovat roolikäsittelijä, agentti, kommunikaatio ja yhteys. Työssä erillisiä kokonaisuuksia on käyttöliittymä ja internetsivu. Käyttöliittymä kommunikoi REST:n välityksellä ja tällä voidaan muokata agenttiorjapalvelussa käytettäviä lapsi-agentti aliohjelmia. Komponenttien jaottelussa on huomioitu etteivät ne tee päällekkäisiä tehtäviä. UML - suunnittelukaavio on (Kuvassa 8.) esitettynä.

5.5.1 Roolikäsittelijäkomponentti

Tämä roolikomponentti hallitsee lapsi-agentti aliohjelmia ja tämän sisällä on myös oma vahtikoira-piirin aliohjelma. Vahtikoira-piirin tarkkailu varmistaa lapsi-agenttien aliohjelmien toiminnassa olemisen. Vahtikoira-piiri osaa myös käynnistää lapsi-agentti aliohjelman uudelleen, jos aliohjelma on kaatunut tai sammunut. Tällainen mahdollisuus on, jos lapsi-agentti aliohjelma on kaatunut tuntemattomasta syystä. Operaattori on ainokainen[60] luokka, joka osaa käynnistää tai sammuttaa lapsi-agentti aliohjelmia, jotka näkyvät sille rooleina.

Ainokainen[60] luokalla tarkoitetaan sellaista luokkaa, joka tekee itsestään esiintymän. Esiintymiä ei voi tehdä useampia ainokaisesta luokasta. Joten ainokaisia luokkia voi olla vain yksi kerrallaan toiminnassa.

5.5.2 Agenttikomponentti

Agenttikomponentin tehtävä on ottaa vastaan lapsi-agentti aliohjelman tilatietoja. Agenttikomponentille on kerrottu millä protokollalla järjestelmä kommunikoi ulos. Tämä kertoo sen kommunikaatiokomponentille, jotka toteuttavat asetetun kommunikointi tavan isäntä- tai orjatason agentille. Agenttikomponentin tehtävä on lisätä tietokantaan rivejä ja näitä rivejä syntyy, kun lapsi-agentti aliohjelma lähettää tilatietoja agentti komponentille. Agentti tukee kolmea sisäistä kommunikointi tapaa ja ne ovat REST, IPC ja gRPC.

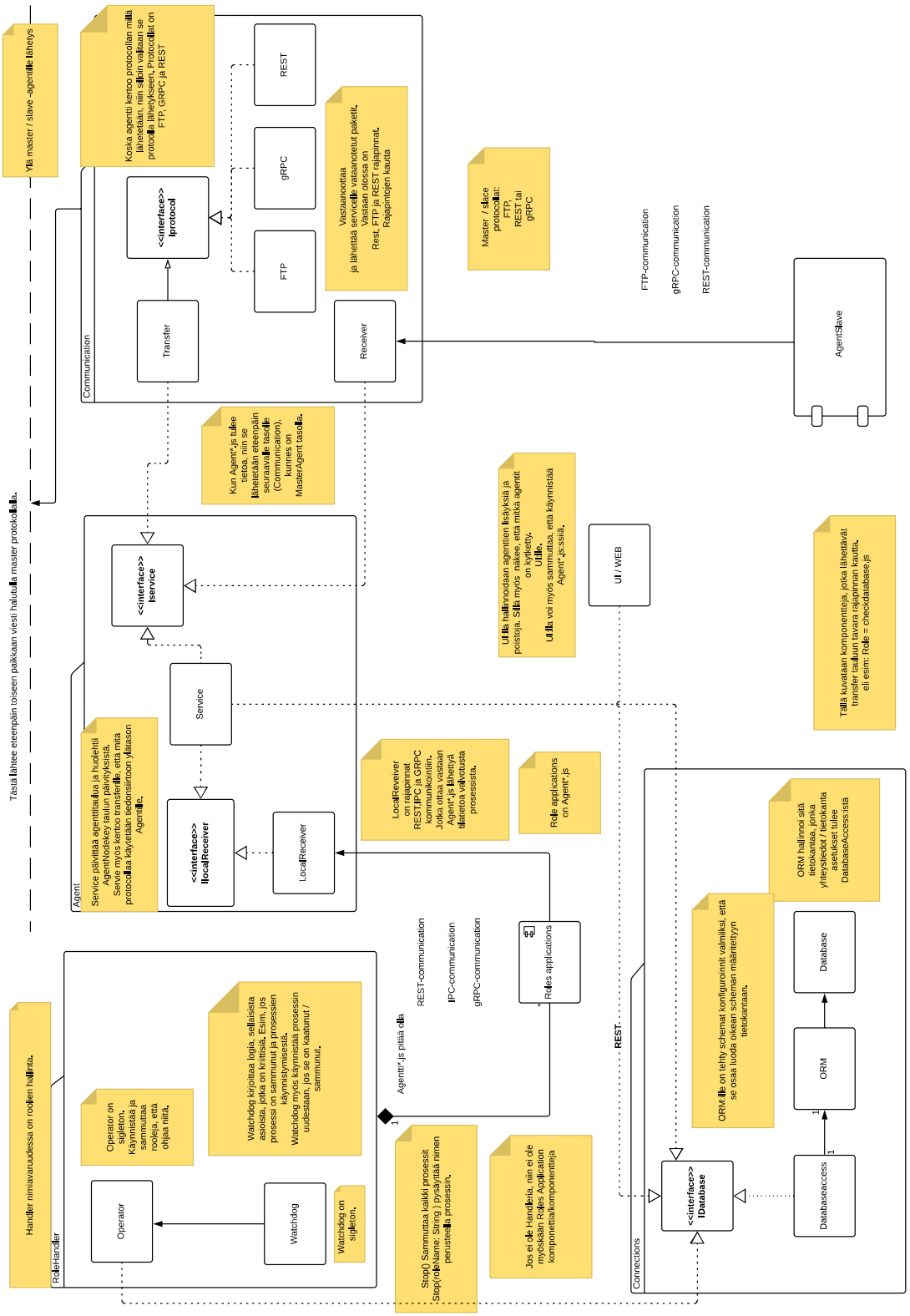
5.5.3 Kommunikaatiokomponentti

Kommunikaatiokomponentin vastuulla on tiedon lähettäminen orja- tai isäntätason agentille. Tämä toteuttaa kolmea ulospäin suuntautuvaa kommunikointi tapaa, jotka ovat FTP, gRPC ja REST. Jos kommunikointi tapana on FTP, niin tämä ei varsinaisesti käytä FTP protokollaan tiedon siirtämiseen. Vaan luo tiedoston nimeltä *.monitorointi, joka siirretään FTP hakemistoon. Täältä FTP hakemistosta kolmannen osapuolen FTP siirto-ohjelma siirtää sen orja- tai isäntätason agentille. Vastaanottopäässä on kuuntelija. Kuuntelija tarkkailee, että onko *.monitorointi päätteistä tiedostoa tullut vastaanottavan FTP hakemistoon. Kun FTP hakemistoon on tullut *.monitorointi tiedosto, niin silloin se välittää tiedoston sisällön agenttikomponentille. Agenttikomponentti lisää tilatiedon tietokantaan ja rakentaa puumaista hierarkiaa. Tietokannan tauluja ovat agentti ja agenttisolmuavain. Edellä esiteltyjä tauluja käytetään puumaisen hierarkian tekemiseen tietokantaan. Näin pystytään paremmin selvittämään, mistä paikasta tilatietojen viestit ovat tulleet. FTP -siirroissa on myös suuremmat viiveet, kuin muissa kommunikointi siirto tavoissa.

Muissa kommunikointi tavoissa (paitsi FTP) on suoremmat kommunikointiyhteydet. Niillä voidaan lähettää tilatietoja suoraan vastaanottopäässä olevalle orja- tai isäntä-agentille.

5.5.4 Yhteyskomponentti

Yhteyskomponentin vastuulla on huolehtia tietokannasta. Yhteyskomponentti huolehtii tietokantaan viedystä, päivitetystä ja poistetusta tiedosta. Tämä komponentti käyttää ORM:n[34] mallintamista tietokanta rakenteiden tekemiseen. ORM:n mallirakenteet huolehtivat tietokannan tiedon eheydestä ja ORM:n API:a käytetään tietokannan tiedon hallintaan.



Kuva 8. UML suunnittelukaavio.

5.6 Tapahtuma sekvenssikaaviot (Muutamasta tapauksesta)

Tässä osiossa on esitelty kolme tapahtuma sekvenssikaaviota järjestelmästä. Valinnat on tehty sen mukaan, kuinka tärkeitä ne järjestelmän kannalta ovat. Komponentit ovat agentti-orjapalvelu ja agentti-isäntäpalvelu. ACP:llä tarkoitetaan lapsi-agentti aliohjelmaa.

5.6.1 ACP kommunikointi

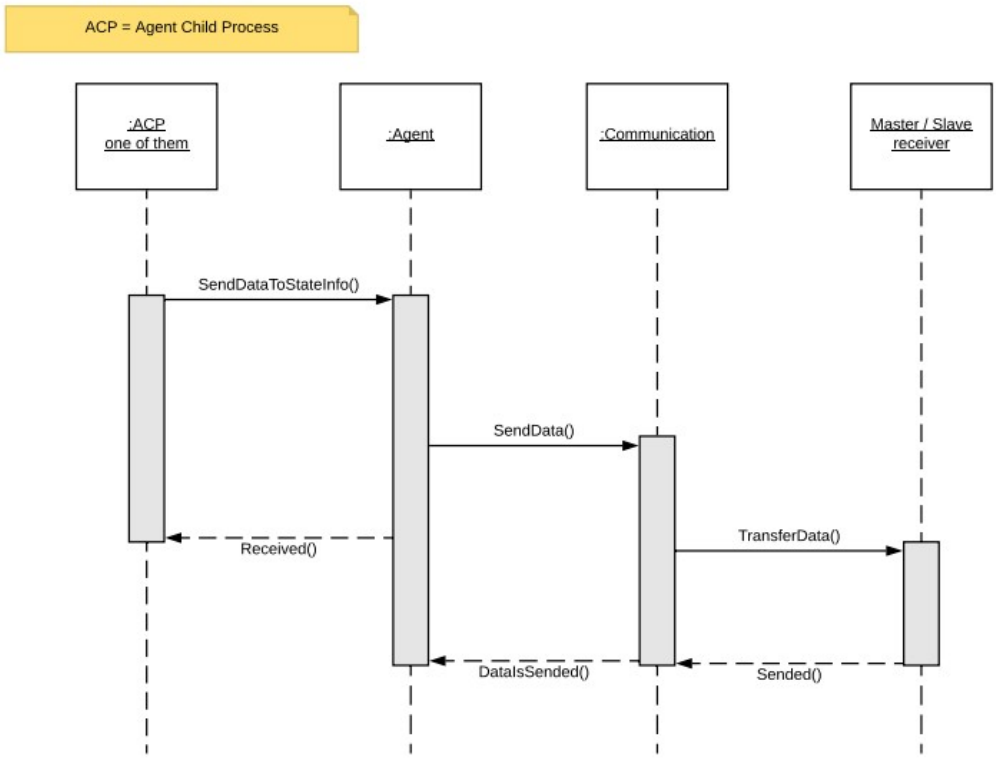
Tapahtumasekvenssikaaviossa (Kuvassa 9.) on esitelty ACP kommunikointia. Tässä tapauksessa ACP lähettää agentti komponentille tilatiedot, joka välittää sen kommunikaatiokomponentille. Kommunikaatiokomponentti lähettää tiedot, joko orja- tai isäntä-agenttitasolle. Agenttikomponentti määrittää sen, että millä kommunikointitavalla kommunikaatiokomponentti välittää tiedot näille kahdelle esitellylle tasolle.

5.6.2 ACP ja agenttikommunikointi taso

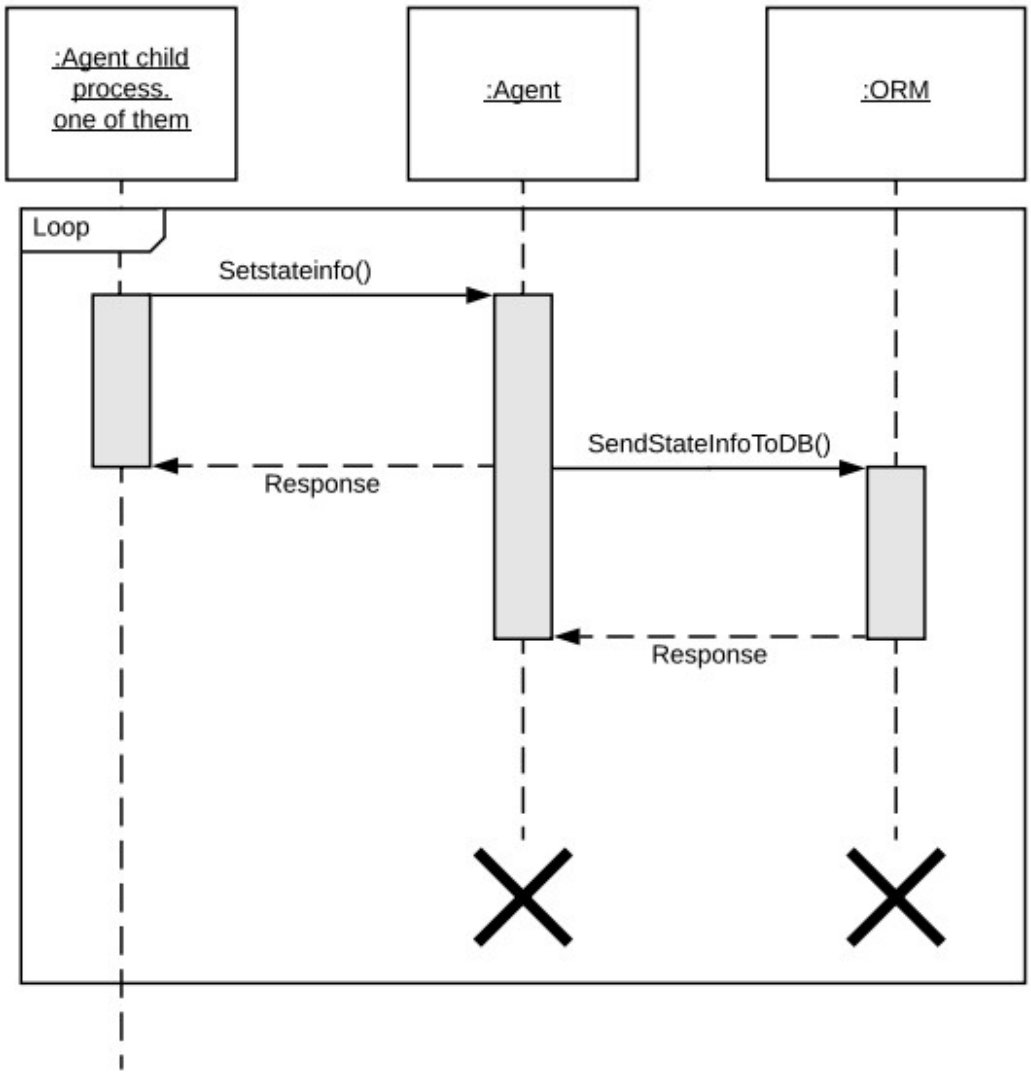
Tapahtumasekvenssikaaviossa (Kuvassa 10.) on esitelty, kuinka ACP kommunikoi agenttikomponentin kanssa. Agenttikomponentti välittää ORM:a käyttäen tietokantaan tiedot. Kuvassa ei ole näytetty tietokantaa, koska ORM:i on virtuaalinen tietokanta. ORM:n vastuulla on välittää tiedot ja pitää huolta määritetyistä rajoituksista.

5.6.3 Käyttäjä asettaa ACP:n toimintaan

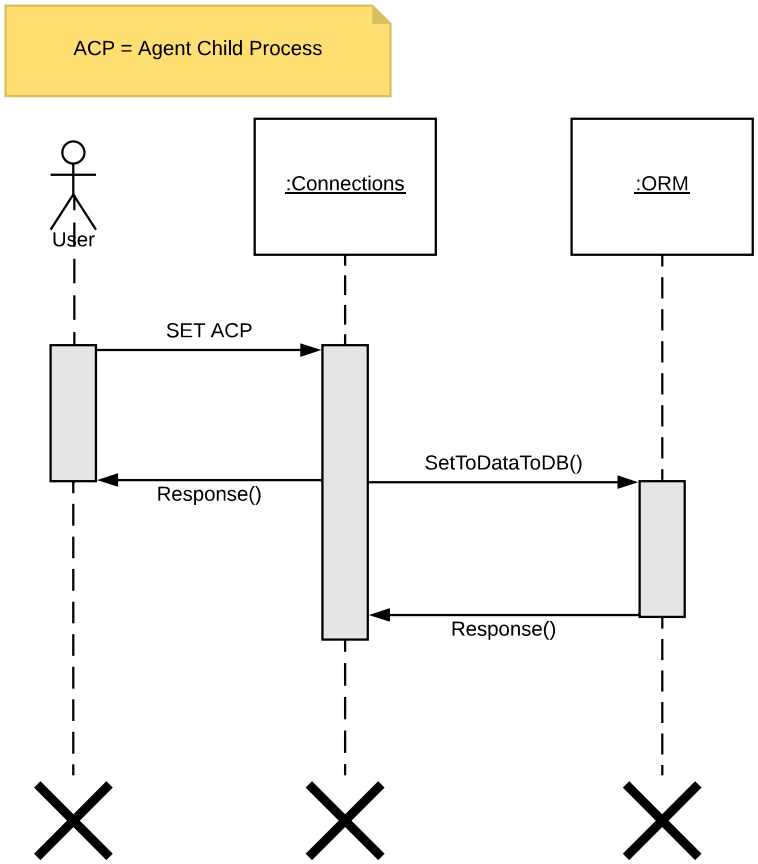
Tapahtumasekvenssikaaviossa (Kuvassa 11.) on kuvattu, kuinka käyttäjä asettaa ACP:n käyttöön järjestelmään. Yhteyskomponentti asettaa halutun ACP:n järjestelmään ja tekee sen ORM:n kautta. Käyttäjä voi lisätä järjestelmään ihan millä hetkellä hyvänsä uuden ACP:n, joka toteuttaa sille toteutetun tehtävän ja rajapinnan agentille.



Kuva 9. ACP kommunikointi agentille ja siitä toisille tasoille.



Kuva 10. ACP tilatiedon tallentaminen agentin kautta tietokantaan.



Kuva 11. Käyttäjän ja ACP hallinnoiminen.

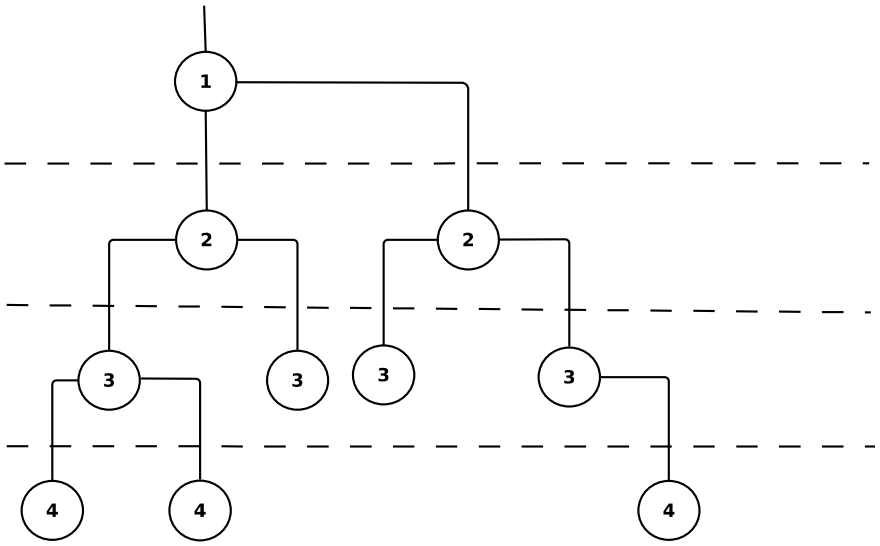
6 TOTEUTUS

Tässä osiossa käydään läpi, miten lähdimme ratkaisemaan monitorointi ongelmaamme. Tarkoituksena on pystyä monitoroimaan maailmalla olevia laitteistoja. Laitteistot voivat olla palvelimia, tietokoneita tai mobiililaitteita. Valmiiden monitorointiohjelmistojen vertailussa isoimpaan rooliin pääsi PRTG[3] ohjelmisto. PRTG ohjelmisto nousi vertailussa kustannuksiltaan ylivoimaisesti ykköseksi. Kuitenkin PRTG:n puutteellisen tietoliikenne kommunikoinnin takia, emme voineet valita sitä käyttöömme. Tämä pakotti miettimään oman toteutuksen tekemistä ja arkkitehtuurin suunnittelua. Tavoitteena oli tehdä oma monitorointiohjelmisto. Monitorointiohjelmisto koostuu isäntä- ja orja agentti palveluista. Isäntä- ja orjapalvelun sisäinen toteutus koostuu lapsi-agentti (APC) aliohjelmista, joiden ohjelmointi on helppoa ja vaivatonta. Oman monitorointiohjelman hallinta pitää olla helppoa ja ohjelman käyttöönoton pitää olla vaivatonta.

6.1 Isäntä- ja orja agenttipalvelut

Agenttien tasot ovat näytetty kuvassa (Kuvassa 12.). Lyhyesti kuvailtuna isäntä-agenttipalvelutaso alkaa ensimmäiseltä tasolta. Isäntä-agenttipalveluja voi olla järjestelmässä vain yksi kerrallaan. Isäntä-agenttipalvelu ottaa vastaan orja-agenttipalvelulta tilatietoja. Nämä tilatiedot tallennetaan MSSQL klusteri palvelimen tietokantaan, jossa isäntä-agenttipalvelu toimii. Orja-agenttipalvelun tasot alkavat toiselta tasolta. Orja-agenttipalvelun tehtävä on kerätä tilatietoja ja siirtää sitä isäntä-agenttipalvelutasolle. Tasolla kolme on orja- agenttipalveluja, jotka siirtävät tilatietoja toisen tason orja-agenttipalvelulle. Taso neljä on oheislaitetaso ja sillä tasolla ei tule pyörimään orja-agenttipalveluja.

Orja-agenttipalvelun kolmas taso tarjoaa rajapinnan neljännelle tasolle, josta saamme tilatietoa oheislaitteilta ja ne lähettävät orja-agenttipalvelulle tilatietoa. Orja-agenttipalvelu lähettää laitteistolta saadut tilatiedot isäntä-agenttipalvelulle.



Kuva 12. Isäntä-agenttipalvelu on ensimmäisen tason solmu ja loput solmut ovat orja -agenttipalveluiden.

6.2 Valmiiden monitorointiohjelmistojen vertaaminen

Otimme käsittelyyn viisi monitorointiohjelmistoa. Teimme monitorointiohjelmistojen vertailut sen mukaan, miten ne kykenisivät palvelemanaan meidän tarkoituksiamme parhaiten. Monitorointiohjelmien pitää pystyä muokkautumaan eri tilanteiden mukaan. Monitorointi vertailussa huomioimme käyttöjärjestelmätukea, hintaa, monitorointi ominaisuuksia, kuinka ne asennetaan tietokoneille ja verkkotukea. Tutkimme myös monitorointiohjelmien kommunikointia, miten ne kommunikoivat ulospäin.

6.2.1 Paessler (PTGR) [3]

Käyttöjärjestelmätuki: Monialustatuki
Hinta: 500 sensoria 1200€ vuodessa, 1000 sensoria 2150€ ja XL1 rajoittamaton sensorimäärä 12,900€ vuodessa.
Monitorointi ominaisuudet:

- Verkkosivun tarkkaileminen.
- Muistin käytön seuranta.
- Kiintolevyn tarkkaileminen käyttöasteen mukaan.
- verkkoliikenteen seuranta.
- Tietokantojen tarkkaileminen. Esimerkiksi tietokannan tarkkaileminen taulukohtaisesti.
- Ohjelmien tarkkaileminen. Esimerkiksi palvelinohjelmien tarkkaileminen.

Verkkotuki: REST.
Huomio: Sensorilla tarkoitetaan yhtä monitorointi ominaisuutta.

6.2.2 Nagios XI[36]

Käyttöjärjestelmätuki: Monialustatuki
Hinta: Riippuu lisenssi tyypistä. Heillä on kolmea lisenssi tyyppiä: standardi lisenssi, yrityslisenssi ja ilmainen lisenssi. Standardi lisenssi maksaa 1731 euroa ja yrityslisenssi 3034 euroa ja ilmainen antaa seitsemän solmua ilmaiseksi.
Monitorointi ominaisuudet:

- Verkkosivun tarkkaileminen.
- Muistin käytön seuranta.
- Kiintolevyn tarkkaileminen käyttöasteen mukaan.
- verkkoliikenteen seuranta.
- Tietokantojen tarkkaileminen. Esimerkiksi tietokannan tarkkaileminen taulukohtaisesti.
- Ohjelmien tarkkaileminen. Esimerkiksi palvelinohjelmien tarkkaileminen.

Verkkotuki: REST ja SNMP
Huomio: Yksi solmu tarkoittaa yhtä konetta tai verkkokytkintä, joka on Nagios XI:hin kiinni. Nagios XI:ssä SNMP protokolla on verkkokytkimien tilatietoihin mahdollistava protokolla ja siinä on kaksi agentti tyyppiä. Agentti tyyppejä on passiivinen ja aktiivinen. Agentit valvovat tietokoneita, eli passiivinen agentti lähettää vain Nagios XI palvelimelle tilatietoa ja aktiivinen agentti, jossa Nagios XI palvelin lähettää agentille pyynnön. Tämä agentti vastaa pyyntöön ja kertoo tietokoneen tilasta. Agentti myös lähettää tilatiedon tietokoneesta Nagios XI palvelimelle.

6.2.3 SolarWind[37]

Käyttöjärjestelmätuki: Monialustatuki
Hinta: Moduulipohjainen ja heidän hinnoittelu alkaa 2168 eurosta. Jos tarvitsee verkkoliikenteen monitoroinnin, sovelluksien monitoroinnin ja tietokanta monitoroinnin. Tällaisen moduulikombinaation hinta nousee lähemmäksi 6500 euroa.
Monitorointi ominaisuudet:

- Verkkosivun tarkkaileminen.
- Muistin käytön seuranta.
- Kiintolevyn tarkkaileminen käyttöasteen mukaan.
- verkkoliikenteen seuranta.
- Tietokantojen tarkkaileminen. Esimerkiksi tietokannan tarkkaileminen taulukohtaisesti.
- Ohjelmien tarkkaileminen. Esimerkiksi palvelinohjelmien tarkkaileminen.

Verkkotuki: REST
Huomio: Moduulipohjainen, jolloin pitää tietää tarkalleen mitä halutaan.

6.2.4 Loppu yhteenveto

Kaikissa tarkastelussa olevissa monitorointiohjelmistoissa on REST kommunikointi ulospäin. Mutta kaupallisissa monitorointiohjelmistoissa ei ole sellaista hälytyksien generointia tiedostoksi ja sen siirtoa FTP hakemistoon. Tästä syystä on oman monitorointiohjelman tekeminen järkevämpää ja se saadaan räätälöityä juuri sellaiseksi kun halutaan. Omaa monitorointiohjelmaa pystytään laajentamaan mahdollisuuksien mukaan ja hallinta on kokonaisuudessa yrityksellämme.

6.3 Tietokannat ja suunnittelu

Tietokantoina käytämme Sqlite ja Microsoft SQL palvelin klusteria. Sqlite tietokantaa käytämme orja-agentipalveluissa ja Microsoftin SQL palvelin klusteria käytämme isäntä- agentipalveluissa.

6.3.1 Sqlite[15] tietokanta

Sqlite tietokantaan tuotetaan tietorakenteet ORM:a[34] hyödyntäen. ORM:i pitää huolen kaikista ehdoista ja säännöistä. Näitä sääntöjä käyttäen tietokanta pysyy eheänä, kun sinne tietoja lisätään. ORM:n[34] rakenteiden kautta tietokantaan viedään tietoa, ei kuitenkaan ikinä suoraan tietokantaan. ORM:i[34] auttaa myös siinäkin, kuinka nopeasti voidaan luoda uusi agenttiympäristö uuteen kohteeseen.

Sqlite[15] tietokantaa käytetään työssä orja-agenttipalveluiden ja orja-agenttipalveluiden tuottaman tiedon varastoimiseen. Tietokantarakenne on suunniteltu puumaisesti. Puumaisella rakenteella tiedämme aina, millä solmulla on äiti ja lapsi suhde. Jokaisella solmulla on tieto, mistä koordinaateista se on peräisin. Solmu tietää myös, miltä roolinimeltä viesti on peräisin. Tätä rakennetyyppiä voidaan miettiä ketjuttamiseksi. Kun siirrymme agentti-isäntäpalvelu tasolle, voidaan tätä ketjumaista rakennetta hyödyntäen liikkua agentti-orjapalvelu solmun kautta alimmalle agentti-orjapalvelu tasolle. Näiden tasojen välillä voidaan tutkia, millä tasolla ongelmat ovat tai eivät ole. Tietokantarakenteet on suunniteltaessa pidetty hyvin yksinkertaisina. Tietokantarakenteissa on pyritty siihen, että sen toiminta on yksinkertaista ja tehokasta.

6.3.2 Microsoft SQL palvelin klusteri[32]

Tämän tietokannan tehtävä on kerätä kaikkien agentti-orjapalveluiden tuottama tieto yhteen paikkaan ja tämän avulla analysoidaan tietoja. Nämä tiedot tullaan näyttämään käyttöliittymässä. Tämä työ ei kuitenkaan keskity käyttöliittymään, vaan taustajärjestelmään. Taustajärjestelmän tarkoituksena on tuottaa tietoa. Suunnittelussa on mietitty passiivista lähettämistä, joka vain ottaa vastaan tietoa. Passiivinen lähettäminen ei lähetä

mitään takaisin päin alemmalle orja-agenttipalvelu tasolle. Kuitenkin pidetään kirjaa kommunikoinnista, mistä tietoa on tullut ja kuinka kauan se on kestänyt. Jos tiedon tuleminen on kestänyt liian kauan, annetaan siitä varoitustieto. Tämä varoitustieto indikoi mahdollisesta verkko-ongelmasta, järjestelmän jollain asiakasympäristö tasolla.

6.3.3 Suunnittelu

Tietokannan suunnittelu aloitettiin kartoittamalla meidän tarpeet. Suunnittelussa käytimme iteraatiomallia. Tässä mallissa mietittiin, minkälaisia tietoja meidän tarvitsee tallentaa tietokantaan ja mitä tietoja tarvitsee ylläpitää. Suunnittelua tehdessä on tietokannan rakenteet jaettu selkeästi kolmeen osaan. Ensimmäisessä osassa on roolit, toisessa osassa on lähettäminen ja kolmannessa osassa on agentit. Suunnittelun (Kuvassa 13.) kuvasta nähdään, miten taulurakenteet on jaettu.

6.3.4 Rooli osa (Roolitaulu, Rooliprotokollataulu ja Prosessitaulu)

Roolitaulussa on tiedot, mitä lapsi-agentteja ladataan alussa. Roolitaulussa on myös kerrottu, millä kommunikoinnilla lapsi-agentti keskustelee agentti-orjapalvelun tai agenttiisäntäpalvelun kanssa. Lapsi-agentin käytössä on IPC,REST ja GRPC kommunikoinnit. Yhdessä lapsi-agentissa voi olla siis vain yksi kommunikointi tapa. Roolitauluun tallennetaan tieto, missä PID:sä aliohjelma tunniste pyörii. Tämän tunnisteen avulla pystytään jäljittämään aliohjelma. Aliohjelman PID tunnisteen kautta on helpompaa käynnistää uudelleen aliohjelma tai lopettaa aliohjelma. Roolilla voi olla useampia samanlaisia lapsi-agentteja käytössä, mutta niille jokaiselle pitää olla oma PID tunniste. Tämän PID tunnisteen avulla tiedetään, mikä aliohjelma on milläkin lapsi-agentilla. Roolitaulussa on sarakekenttä, johon voidaan laittaa ulkopuolisen laitteen ip-osoite. Tämän ip-osoitteen avulla lapsi-agentti aliohjelma voi sitten kysellä laitteen tilaa. Laite voi olla vaikka tulostin.

6.3.5 Lähettämisenosa (Siirtotaulu, Siirtotilataulu, Tasotilataulu ja Tietotaulu)

Lähettämisenosa tarkoituksena on huolehtia, miten lähetettävät tiedot ovat lähetetty. Siirtotaulussa on tieto laitteennimestä, roolinimestä, tilatieto, siirtoavain, pituusaste, leveysaste, vanhemman siirtoavain ja tiedonkeräilijä. Roolinimi on lapsi-agentin nimi ja tilatieto tietää onko lähetys onnistunut.

Tilatieto päivitetään, kun siirto on tapahtunut. Tämä sen takia, koska järjestelmä toimii passiivisen tiedon lähettäjänä.

Siirtoavain on tieto lähetettävän orja- tai isäntäagentin lähetysavaimesta. Orja- tai isäntäagentti osaa myös kertoa siirrolle, millä kommunikointitavalla lähetys tapahtuu eteenpäin.

Pituusaste ja leveysaste kertoo, mistä koordinaateista lähetykset ovat tapahtuneet, jolloin päästää kiinni maantieteelliseen sijaintiin. Tämä tieto edesauttaa selvitystä, mistäpäin tiedot ovat tulleet agentti-orjapalvelulle.

Vanhemman siirtoavain kertoo, mikä avain on ylimmän isäntä- tai orja agenttipalvelutason siirtoavain puun solmussa. Näin pystytään paremmin selvittämään ketjua, mistä puusta kyseinen tieto on tullut.

Tasotilatieto kertoo lähetettävän agentin statuksen. Statukset ovat kunnossa, varoitus tai virhe. Tasotila kertoo, onko lapsi-agentti aliohjelman tutkittava kohde kunnossa. Esimerkiksi kiintolevyn tarkistaminen voidaan hoitaa niin, että täyttöasteen ollessa alle 85 prosenttia on status kunnossa. Kiintolevyn täyttöasteen ollessa yli 85 prosenttia ja alle 90 % on statuksena varoitus. Kiintolevyn täyttöasteen ollessa yli 90 %, niin silloin lähetetään virhetieto. Näiden tasotilatietojen perusteella aletaan tekemään korjauksia kohdekoneelle, jotta järjestelmässä ei tule odottamattomia virhetiloja.

Tiedonkeräilijän tietokanta sarakkeen tarkoituksena on viedä enemmän tietoa yleisesti järjestelmästä. Tiedonkeräilijän tietokanta saraketta tarvitaan erikoistietojen vientiin orja- tai isäntäagenttipalvelulle.

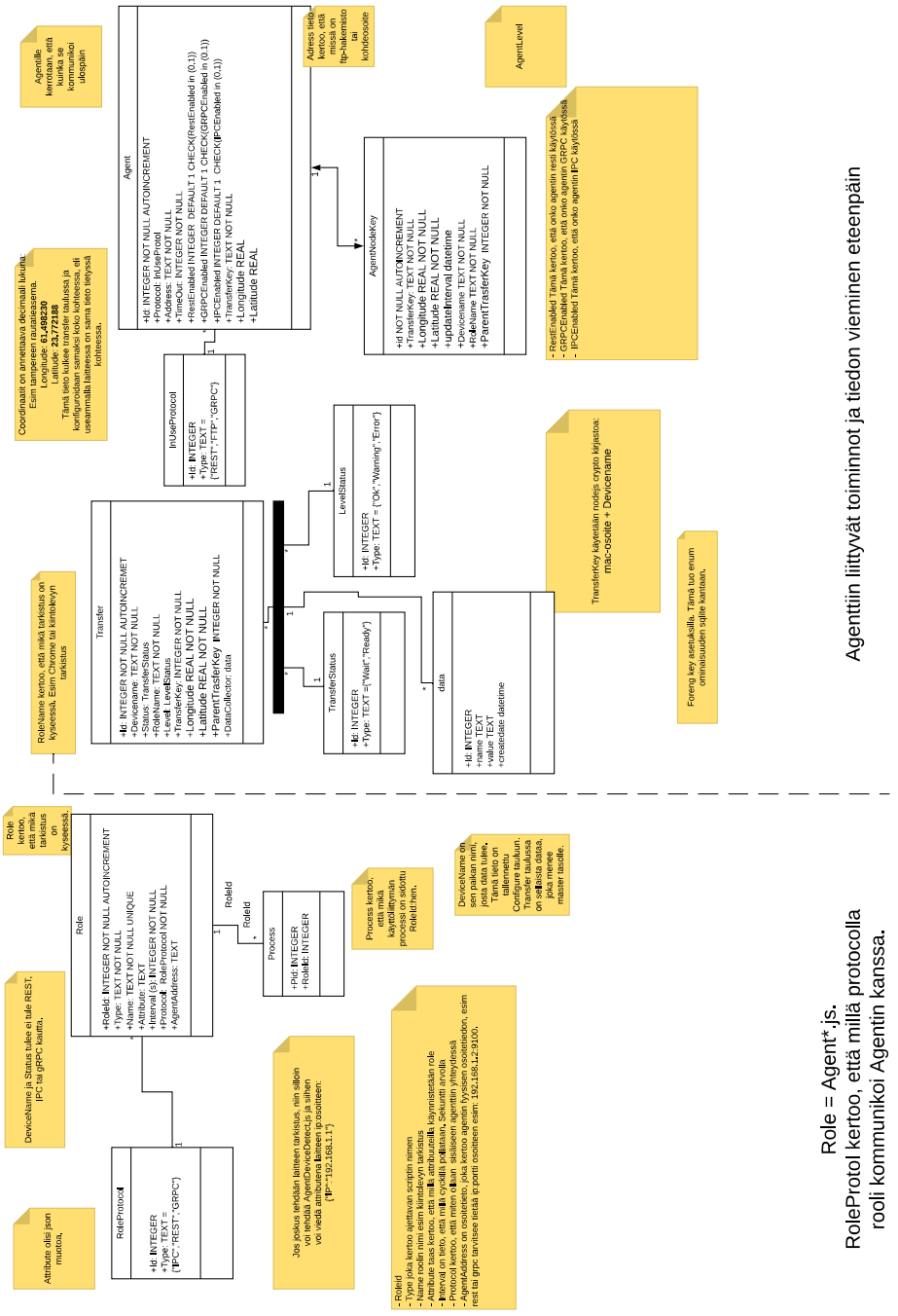
6.3.6 Agentti (Agenttitaulu, käytössä protokollataulu ja agenttisolmuavaintaulu)

Agentille on kerrottu, miten se kommunikoi ulkopuolelle muille agentti-orjapalveluille tai agentti-isäntäpalveluille. Agentille voidaan kertoa myös, mitä sisäisiä kommunikointi tapoja on käytössä. Tarvittaessa voidaan estää joidenkin kommunikointitapojen käyttö sisäisessä kommunikoinnissa.

Agenttitaulussa on osoitetieto, jota käytetään FTP kommunikoinnissa. Osoitetietoon laitetaan tieto, mihin hakemistoon agentti tulee tekemään siirrettävän tiedoston. Tiedoston rajapintakuvauksen mukainen tieto on tiedoston sisällä (Kuvassa 14). Tiedoston pääte on *.monitorointi. Tällä kerrotaan toisen pään agentille, että kyseessä on agentilta tuleva viesti.

Siirtoavain muodostetaan koneen MAC[63]-osoitteesta ja laitenimestä, jotta se olisi uniikki ja tunnistettavuus säilyisi. Agentilla on myös pituusaste ja leveysaste tieto, missä koordinaateissa laite on.

Agenttitaulun alitauluna on agenttisolmuavaintaulu, jossa on vanhemman siirtoavaimen tieto. Tällä tiedetään, mikä agentti on agenttisolmuavaimesta vanhempi solmu. Näin muodostetaan puumainen rakenne ja niin sanotusti ketjutetaan. Tällä rakenteella pystytään menemään äiti-solmusta uusimpaan lapsi-solmuun. Agenttisolmuavaintaulusta löytyy myös koordinaattitiedot, laitenimi, roolinimi ja siirtoavain. Siirtoavain on uniikki tieto agenttisolmuavain taulussa.



Kuva 13. Tietokannan suunnittelu on dokumentointi UML-kaaviolla.

```
[{
"AgentSlaveServiceJSON": {
"Devicename": "{Kertoo, että mistä paikasta tiedot ovat tulleet}",
"RoleName": "{Kertoo, että onko kyseessä agentHardrive tarkistus yms.}",
"Level": "{'Ok','Warning','Error'} tasot, kertoo, että kuinka kriittinen on tilanne. Ok on normaalitilanne,
Warning on varoitus ja Error on virhetilanne.",
"Transferkey": "Generoitu" ,
“Longitude”:"{Pituusaste, tämän voi hakea google mapsista}"
“Latitude:"{leveusaste, Tämän voi hakea google mapsista}"}",
“ParentTrasferKey”:" Haetaan parentin trasnferkey",
“Data”: {array}
}]
```

Kuva 14. Siirtorakenteen kuvaaminen rajapintasuunnittelusta.

Ilman hyvää (Kuvassa 14.) rajapintasuunnittelua ei pystytä toteuttamaan sellaista järjestelmää, joka tukee monia ohjelmointikielijä ja yhteneväisiä kommunikointitapoja. Tätä suunnitelmaa voidaan muuttaa tietenkin niin, että kaikki tasot monitoroinnin alla tukevat tätä uutta rajapintakuvausta. Rajapintasuunnittelussa on pyritty siihen, että mahdollisimman vähän tietoa kulkisi jokaisen agentti-orjapalvelun kautta ylimmälle agentti-isäntäpalvelun järjestelmätasolle. Tällä pyritään minimoimaan tietomassat, jotta ylin agentti-isäntäpalvelu taso ei kuormittuisi liikaa.

7 TULOKSET JA NIIDEN TARKISTELU

Monitorointisovelluksia oli useampiakin tarkastelussa, mutta yksi niistä nousi parhaiten edukseen. Lähimpään tarkasteluun otettiin PTGR[3] monitorointiohjelmisto. Tässä monitorointi sovelluksessa on alustavasti parhain keino rakentaa erilaisia hälytys seurantoja. PTGR ei ole myöskään riippuvainen tietystä ohjelmointikielestä, koska se tukee useita Scripti ohjelmointikieliä. PTGR:ä ei puhuta agenteista vaan sensoreista.

Kaupallisten ohjelmistojen rinnalla aloimme pohtimaan, jos tekisimme oman toteutuksen monitorointiohjelmistosta joka omaisi mahdollisimman laajan tuen. Omalla monitoroinnilla voisimme monitoroida juuri ne koneet mitä haluamme. Oma monitorointiohjelma tulisi tukemaan juuri niitä kommunikointi tapoja, joita verkkoinfrastruktuurimme yleisesti tukee. Oman kehittämisessä tulee aina väkisinkin vastaan kustannukset ja riskit. Riskeinä voisi olla kokonaissuunnittelun onnistuminen ja toteutuksen onnistunut tekeminen.

7.1.1 PTGR tositoimissa

Asensimme testikoneelle yhden PTGR[3] ilmaisversion, jossa on 100 sensoria käytössä. Kyseisellä testiasennuksella, voidaan monitoroida seurattavaa konetta. Testin tarkoituksena oli vain selvittää, mitenkä sovellus käyttäytyy verkkoympäristössä. PRTG testin tuloksena oli, että ohjelmisto heti käynnistyttyään tarkasti automaattisesti koko verkkoinfrastruktuurin. PTGR ohjelmisto löysi sellaisia koneita verkosta, joista ei ainakaan ole mitään kerrottu asianomaisille. Samalla se käytti kaikki sensorit, joten siitä pitäisi heti ostaa maksullinen versio, kun kokeiluversio päättyy. Koska testin tarkoituksena oli selvittää, mitenkä se soveltuu asiakkaalle asennettavaksi. Testin tuloksena oli punaisen lipun nostaminen pystyyn. Kun PTGR:n asentaa asiakkaan koneelle, niin asiakkaan ICT henkilöiden niskakarvat nousevat pystyyn. Tämä johtuu siitä syystä, että asiakkaan verkossa oleva verkkoliikenne lisääntyy PTGR:n skannauksen aikana. Tämä aiheuttaa ei toivottuja hälytyksiä ICT-henkilöille. Hälytykset tulevat koska ohjelma skannaa sellaisia tietokoneita, jotka ovat verkossa piilossa. Tällainen verkontutkiminen aiheuttaa epäilyksen vihamielisestä liikennöinnistä.

PTGR ohjelmistosta käyttää REST kommunikointitapaa tiedon siirtämiseen. Pelkkä REST kommunikointi ei sovellu yrityksemme käyttöön, koska osa meidän verkkoliikenteestä menee perinteisen FTP -tietoliikenneprotokollan kommunikoinnin kautta ulko-

maailmaan. PTGR on vahvistanut tämän meille, että he eivät tue FTP -tietoliikenneprotokollaa. PTGR ei osaa tehdä tiedostoa FTP hakemistoon, josta kolmannen osapuolen FTP -ohjelmisto hakisi tiedoston ja siirtäisi sen kohde koneelle. Tästä PTGR hakisi tiedon ja ilmoittaisi, mikä on ollut koneen tila. Tällaiseen rakenteeseen ei PTGR pysty.

PTGR on varmasti hyvä sellaisessa ympäristössä, jossa on käytössä REST kommunikointi. Ja monitoroiva ympäristö toimii omassa konesalissa, josta tiedetään kaikkien laitteiden kuuluvan yrityksen omistukseen ja hallinnointiin. Tämä ei aiheuta ICT:le harmaita hiuksia, kun on rajattu oikeaoppisesti PTGR:n ohjelmiston toimintaympäristö.

7.1.2 Monitorointi (Oma toteutus)

Oman toteutuksen lähtökohtana oli, että sitä pystytään laajentamaan helposti ja tästä syystä järjestelmää voidaan laajentaa lapsi-agentti ohjelmien kautta. Oma monitorointiohjelma tukee olemessa olevaa rajapintakuvausta (Kuvassa 14). Lapsi-agentti aliohjelmat voidaan tehdä ihan millä tahansa ohjelmointikielellä, joka tukee REST kommunikointia tai gRPC:tä. Jos toteutus on, tehty Node.js:llä voidaan sisäisessä kommunikaatiossa käyttää IPC-soketti yhteyttä. Monitorointiohjelmiston ohjelmointikieleksi valittiin Typescript, joka on Javascript laajennus. Hyvinä puolina voidaan pitää, että Typescript tekee Javacsript tiedoston. Javascript tiedosto luodaan siinä vaiheessa, kun ohjelma käännetään tsc-komennolla. Yhtenä tarpeena on monitoroinnin laajentaminen matkapuhelinpuolelle, jolla valvotaan yhteyksien toimintaa ja ohjelmistoversioita. Matkapuhelinpuolen valvonnassa on tietenkin huomioitava GDPR[39], koska laitteen tietojen valvonnassa ei saa missään nimessä lähettää henkilötietoja tai mitään sellaista, jolla henkilö voidaan jäljittää.

Suunnittelu on nyt sellaisessa tilassa, että sen toteuttaminen on voitu aloittaa. Toteutuksen tekemisessä on haasteensa, koska Node.js:sä olevat paketit päivittyvät aika tiuhaan ja ovat toisistaan riippuvaisia. Tämä aiheuttaa sen, jos yksikin riippuvuus suhde menee hajalle tilaan. Silloin projektin kehittäminen ei ole käytännössä mahdollista. Tämän takia käytämme Node.js:än pakettihallinnassa ja ’^’ väkästä versionumeron edessä. Väkänen lukitsee pakettiversion. Tätä pakettiversiota käytetään, vain tässä projektissa ja sitä ei saa ilman lupaa päivittää uudempaan versioon. Tietenkin tässäkin on haasteensa, koska jos projekti asennetaan nettiyhteyden yli pakettihallinnan kautta. Silloin voi olla, että kyseinen versio on jo poistettu ja tämä aiheuttaa omia ongelmia projektissa. Tämän takia onkin suositeltavaa viedä kohdekoneelle kaikki asennetut paketit ja näin varmistaa, että ohjelmisto toimii oikein. Tämä oikein toiminta varmistetaan, kun ohjelma käynnistetään kohdekoneella. Toinen tapa estää pakettien ristiriidat on tehdä asennuspaketti. Esimerkiksi Windows ympäristöön ”.exe” tai Linux ympäristöön ”.sh”. Asennuspaketista on tietenkin sekin etu, että se pysyy vakiona kaikkineen pakettiriippuvuuksineen.

8 YHTEENVETO

Työ on vasta aluillaan ja sitä tehdään niin hyvin, kuin mahdollista. Tämän perusteella, esimerkiksi kun suunnittelee tällaista järjestelmää, on hyvä syy tehdä kuvia ja miettiä toimintaa. Kuvat auttavat miettimään kommunikointia ja luokka hierarkiaa. Kuvista on helpompi löytää suunnittelumalleja. Suunnittelumallien löytäminen auttaa ohjelmistokehittäjiä. Ohjelmansuunnittelussa on käytetty UML[40]-mallinnusta sekä tietokantasuunnittelussa että ohjelmansuunnittelussa.

Tarkoituksella ohjelmansuunnittelu on tehty todella abstraktiotasolla, jolloin ohjelmoijalla on suuremmat mahdollisuudet tehdä suunnitelman mukainen toteutus. Hänellä on myös mahdollisuus vaikuttaa sisäisten funktioiden, attribuuttien ja luokkien nimeämiseen. Korkeammalta tasolta katsottaessa huomataan, että kaikki työssä käytettävät moduulit hoitavat omaa tehtäväänsä.

Tietokantasuunnitelma on toisinpäin verrattuna ohjelmansuunnittelun. Tietokannassa on annettu valmiiksi jo kaikki taulunimet ja siihen liittyvät sarakenimet. Tämä sen takia, koska monitorointiohjelmaan on tehty ORM[34]:la tietokantamallit ja tämä generoi automaattisesti kaikki taulurakenteet tietokantaan. Ohjelmoija käyttää ORM[34]:n kautta tietokantaa ja ei näin tee itse suoraan tietokantaan mitään muutoksia. Jos menee tekemään muutoksia ja joku ajaa migraation (uudemman muutoksen) tietokantaan, niin nämä ulkopuoliset muutokset tulevat katoamaan.

Työn tekeminen oli mielekästä. Sain itse tutkia ja valita työhön sopivimman pakettihallintajärjestelmän ja samalla myös pääsin tutustumaan uusiin teknologioihin. Näiden uusien teknologioiden avulla sain omaa osaamistani ylöspäin. Varsinkin kun ohjelmointia tehdessä sain suoraan kirjoitettua dokumenttia, joka käännetään suoraan internet-sivuksi. Tämä auttaa muita ohjelmistokehittäjiä, kun lukevat dokumenttia. Dokumentista löytää nopeammin käytetyt luokat, funktiot ja tietorakenteet. Työssä on mukana myös automaattitestaus, jolla pystytään tekemään TDD[27]:ä. TDD:lä ja kirjoittamaan testit ensin ja sitten luomaan sen testin ympärille ohjelmakoodin. Ohjelmakoodi voidaan testata testin avulla. Normaalisti tällainen testin tekeminen aloitetaan siitä, että testi epäonnistuu ensimmäisellä kerralla. Tämän jälkeen aletaan rakentamaan ohjelmakoodia ja samalla voidaan tehdä testiinkin tarvittavat muutokset. Testikoodin suorittaminen antaa joko virheen tai onnistumisen. Tästä on se hyöty, jos joku menee muuttamaan koodia ja ajaa testit. Testeistä hän huomaa heti tapahtuneen virheen, joka on aiheutunut siihen ohjelma-

koodilohkoon. Tähän ohjelmakoodilohkoon hän on tehnyt muutoksen, josta virhe on syntynyt. Ohjelmakoodilohkossa tämä tarkoittaa tässä tapauksessa funktion eri paluu arvon palautusta ja tämä on rikkonut kyseisen ohjelmakoodilohkon. Tilanteesta selvitään, kun kyseinen virheellinen paluuarvo on korjattu.

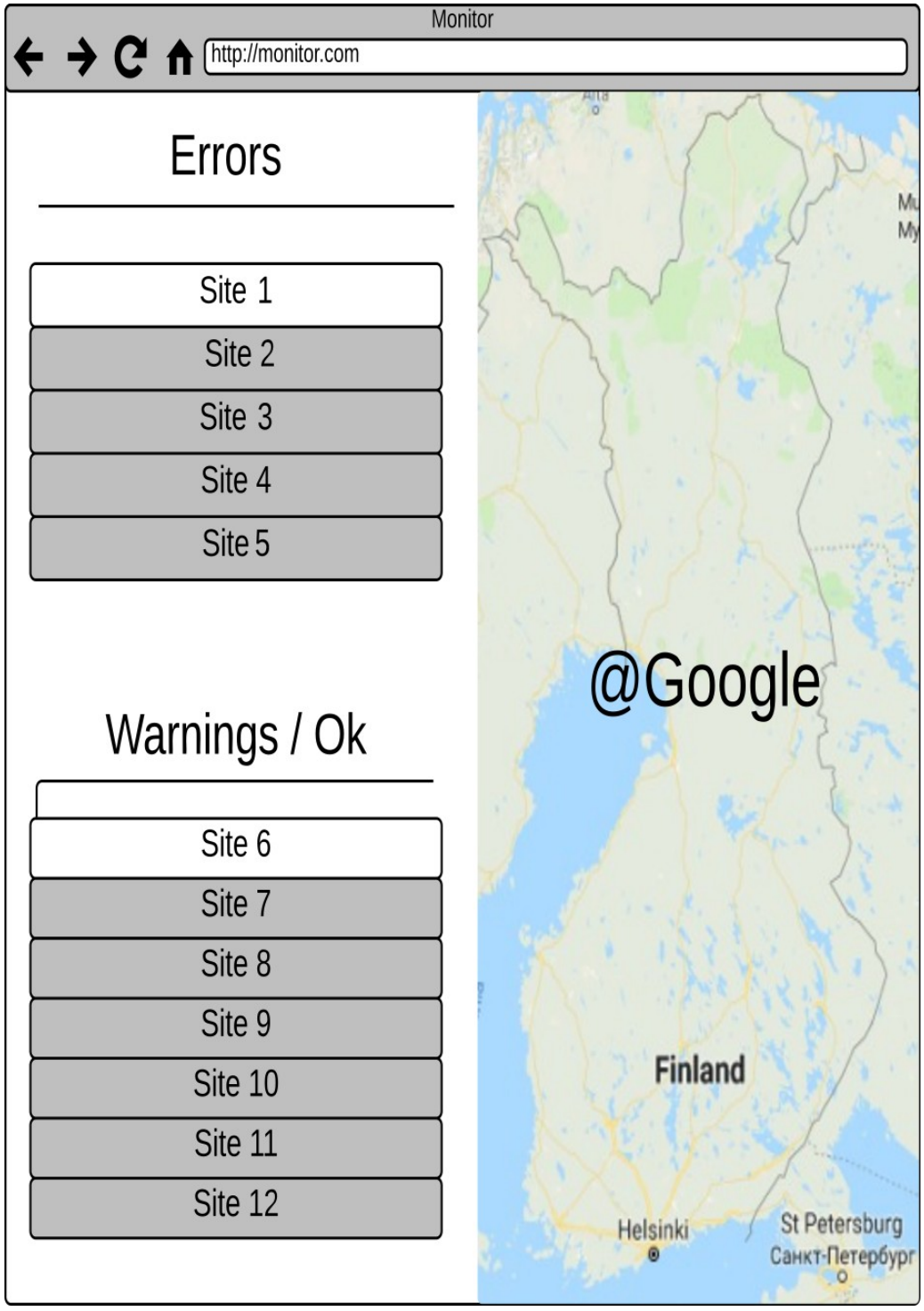
Työässä käytettävä Typescript, joka on laajennus Javascriptin päälle tuo ohjelmointiin vakautta. Vakautusta tulee, koska on staattisesti tyypitetty kieli. Tämän avulla pystytään luomaan parempaa ohjelmakoodia ja joitakin Javascriptissä olevia monimutkaisia rakenteita voidaan helpottaa käyttämällä Typescript laajennusta ohjelmoinnissa. Tietyllä tavalla Typescript tuottaa myös haastavuutta, koska kaikkiin paketteihin ei ole tehty ”@types/paketinnimi” rajapintaa. Rajapinnan avulla pystytään käyttämään alkuperäistä Javascript pakettia. Typescript tuo myös moduulit, jolla ohjelmakoodia pystytään paremmin hajottamaan omiin kokonaisuuksiin . Tämä tuo ohjelmakoodiin selkeät ohjelmarakenteeseen.

Kun työhön kuuluva taustajärjestelmä on saatu valmiiksi ohjelmoitua ja siihen muutamat lapsi-agentti ratkaisut. Lapsi-agentti tarkkailee tietokoneen resursseja ja palauttaa ylätasolle tarvittavat rajapintakuvauksen antamat tiedot. Näiden tietojen perusteella voidaan lopputulosta pitää hyvänä. Monitorointiohjelmassa on sekin hyvä, kun tarvitaan monenlaisia tarkkailuja. Voidaan tarkkailuja helposti luoda ja generoida. Tässä suhteessa monitorointiohjelma on pitkäikäinen ja tulevaisuudessa voi se olla yksi tuote muiden yritystuotteiden joukossa.

Ennen kuin monitorointiohjelmistoa päästetään asiakasympäristöihin. Pitää se testata omassa testiympäristössä, joka vastaa tämän dokumentin laitteistovaatimuksia.

Seuraava askel on työssä tehdä etujärjestelmä. Etujärjestelmä ei kuulu tämän työn piiriin vaan se on enemmänkin tulevaisuuden suunnitelmia. Etujärjestelmällä pitäisi saada kaikki irti monitorointiohjelmasta. Koska järjestelmässä tiedetään koordinaatit. Näiden koordinaattitietojen perusteella pystytään karttaan tuomaan tiedot, missäpäin maailmaa kyseinen ongelma on. Ongelma pystytään paremmin indikoimaan etujärjestelmässä. Etujärjestelmästä on piirretty pieni esivedos (Kuvassa 15.), jossa annetaan suuntaa käyttöliittymän suunnittelusta. Käyttöliittymässä voisi olla suomenkartta, josta nähdään kunnossa, varoitus ja virhe -pisteet. Se kertoo heti, että millä asiakkaalla on virhepäällä. Käyttöliittymässä näytetään aina virhetilat(Varoitus) ensin ja niitä näytetään kokoajan kunnes tilanne normalisoituu. Varoitus ja kunnossa -tilat näytetään samassa. Varoitus tilat kuitenkin näytetään niin, että varoitukset näytetään aikajärjestyksessä ensin uusin ja tämän jälkeen vanhimmat. Kunnossa tilat taas menee ensin uusin ja sitten vanhin perään. Tämä sen takia, että pystytään heti reagoimaan ja tarkistamaan ongelma tilanteet. Esimerkiksi miksi varoitustila on päällä. Tarkistetaan ensin varoitus tila, saadaanko se

ensin pois päältä. Tämä sen takia, että ehditään ensin korjaaman ongelma. Ennen kuin ongelma menee virhetilaan. Tällainen virhetila kertoo tulipalosta järjestelmässä ja se pitää mahdollisimman nopeasti sammuttaa.



Kuva 15. Kuvassa oleva käyttöliittymä on etujärjestelmän ulkoasun toteutus visio.

Tulevaisuudessa monitorointiohjelmaan voidaan rakentaa tekoälyjärjestelmä. Tekoäly pystyy itsenäisesti päättelemään, onko tila tulipalo vaiko ihan normaali tila. Tällaiseen tekoälyratkaisuun päästään, kun normaali monitorointiohjelma on kerännyt tarvittavan tietomassan tietokantajärjestelmään. Tämän tietomassan avulla pystytään opettamaan tekoälyä havaitsemaan normaalitilan ja virhetilan väliltä.

LÄHTEET

[1] REST:in, toiminta periaatteen kuvaus. web page. Available (accessed 13.7.2018): https://www.codecademy.com/articles/what-is-rest.

[2] GRPC, Googlen tiedonsiirto protokolla. web page. Available (accessed 13.07.2018): https://grpc.io/.

[3] PRTG, Kaupallinen tietokoneiden valvonta järjestelmä. web page. Available (accessed 11.07.2018): https://www.paessler.com/.

[4] @types/jest, wrapperi jestille, joka validoi jestin, että se toimii typescriptillä. web page. Available (accessed 11.07.2018): https://yarnpkg.com/en/package/@types/jest.

[5] ts-jest, Jest paketti laajennus typescriptille. web page. Available (accessed 11.7.2018): https://yarnpkg.com/en/package/ts-jest.

[6] Jest, Testauspaketti ohjelmistolle. web page. Available (accessed 11.07.2018): https://jestjs.io/.

[7] Gulp-jsdoc3, Gulp integrointi jsdoc:ille, että saadaan automaattisesti dokumentointi generoitua, kun sellainen halutaan. web page. Available (accessed 11.07.2018): https://yarnpkg.com/en/package/gulp-jsdoc3#readme.

[8] JSDoc, Dokumentin generointi työkalu. Tietyllä kommentointi syntaksilla tämä toteuttaa automaattisesti dokumennoin. web page. Available (accessed 11.07.2018): https://yarnpkg.com/en/package/jsdoc.

[9] Sequelize-cli-typescript, Sequelize komentokehote työkalu typescriptille. web page. Available (accessed 11.07.2018): https://yarnpkg.com/en/package/sequelize-cli-typescript.

[10] Sequelize-typescript, Tämä paketti laajentaa sequelizen toimimaan typescriptillä. web page. Available (accessed 11.07.2018): https://www.npmjs.com/package/sequelize-typescript.

[11] Merge2, Yhdistä useita virtoja yhdeksi virraksi järjestyksessä tai rinnakkain. web page. Available (accessed 11.07.2018): https://www.npmjs.com/package/merge2.

[12] Gulp-typescript, Tuo typescriptille automatisoinnin, kun käytetään gulppia. web page. Available (accessed 11.07.2018): https://yarnpkg.com/en/package/gulp-typescript.

[13] Gulp, Automatisoi ja tehostaa työnkulkua. web page. Available (accessed 11.07.2018): https://gulpjs.com/.

[14] Sequelize, Paketti, joka tuo orm:in ominaisuudet. web page. Available (accessed 11.07.2018): http://docs.sequelizejs.com/.

[15] Sqlite3, Paketti auttaa ottamaan yhteyden sqlite tietokantaan. web page. Available (accessed 11.7.2018): https://github.com/mapbox/node-sqlite3.

[16] Express, Minimalistinen web-kehys node.js:lle. web page. Available (accessed 11.07.2018): https://expressjs.com/.

[17] Typescript, Javascriptin päälle rakennettu laajennus, joka tuo tyypityksen Javascript-kielelle. web page. Available (accessed 11.07.2018): https://www.typescriptlang.org/.

[18] Yarn package manager, Nopea, luotettava ja turvallinen pakettihallinta. web page. Available (accessed 11.7.2018): https://yarnpkg.com/lang/en/.

[19] NPM package manager, NPM pakettihallinta, jolla hallinnoidaan paketteja. web page. Available (accessed 11.7.2018): https://www.npmjs.com/.

[20] PINO, Node.js logittaja, joka tekee json logia. web page. Available (accessed 13.07.2018): https://github.com/pinojs/pino.

[21] GRPC, Pikaohje, kuinka voi aloittaa protokollan käyttämisen. web page. Available (accessed 13.07.2018): .

[22] FTP-protokolla, kerrottuna auki, web page. Available (accessed 17.07.2018): https://fi.wikipedia.org/wiki/FTP.

[23] .NET CORE, Ohjelmointi ympäristö. https://www.microsoft.com/net/learn/get-started/linux/rhel.

[24] Xamarin, ohjelmointi ympäristö. web page. Available (accessed 18.07.2018): https://visualstudio.microsoft.com/xamarin/.

[25] Nodejs, on Javascript runtime, joka perustuu googlen V8 moottoriin. web page. Available (accessed 18.07.2018): https://nodejs.org/en/.

[26] R.C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition, 1st ed. Prentice Hall, 2017, .

[27] TDD, Test-driven development: testivetoinen kehitys, web page. Available (accessed 11.8.2018): http://agiledata.org/essays/tdd.html.

[28] Webpack, Niputa tiedostot tyypit omiin tiedostoihin. web page. Available (accessed 08.09.2018): https://webpack.js.org.

[29] Angular, Laaja web kehysjärjestelmä https://angular.io.

[30] Vue.js, Pieni web kehitysjärjestelmä, web page. Available (accessed 08.09.2018): https://vuejs.org.

[31] Reactjs, Melkein kehysjärjestelmä. web page. Available (accessed 08.09.2018): https://reactjs.org/.

[32] Microsoft SQL server cluster, web page. Available (accessed 08.09.2018): https://www.microsoft.com/fi-fi/sql-server/sql-server-2017.

[33] IPC, Inter-process communication, web page. Available (accessed 20.09.2018): https://fi.wikipedia.org/wiki/Prosessien_v%C3%A4linen_kommunikaatio.

[34] ORM, Object-relational mapping, web page. Available (accessed 22.09.2018): https://en.wikipedia.org/wiki/Object-relational_mapping.

[35] SQL, Structured Query Language, web page. Available (accessed 22.09.2018): https://fi.wikipedia.org/wiki/SQL.

[36] Nagios, Monitorointiohjelmisto, web page. Available (accessed 29.09.2018): https://www.nagios.org/.

[37] SolarWind, Monitorointi ohjelmisto, web page. Available (accessed 29.09.2018):https://www.solarwinds.com/

[38] node-ipc, ipc kommunikointi nodejs:llä, web page. Available (accessed 5.10.2018): http://riaevangelist.github.io/node-ipc/.

[39] GDPR, Yleinen tietosuoja-asetus web page. Available (accessed 7.10.2018): https://en.wikipedia.org/wiki/General_Data_Protection_Regulation.

[40] UML, Unified Modeling Language, web page. Available (accessed 7.10.2018): https://fi.wikipedia.org/wiki/UML-mallinnus.

[41] JSON (JavaScript Object Notation), tiedosto muoto. web page. Available (accessed 6.11.2018): https://www.json.org/.

[42] HTML,Hypertext Markup Language, web page. Available (accessed 6.11.2018): https://fi.wikipedia.org/wiki/HTML.

[43] CSS, Cascading Style Sheets, web page. Available (accessed 8.11.2018): https://fi.wikipedia.org/wiki/Cascading_Style_Sheets.

[44] PID,Process identifier, web page. Available (accessed 9.11.2018): https://en.wikipedia.org/wiki/Process_identifier.

[45] SOAP (Simple Object Access Protocol), web page. Available (accessed 9.11.2018): https://fi.wikipedia.org/wiki/SOAP.

[46] Docket, Container application, web page. Available (accessed 10.11.2018): https://www.docker.com/.

[47] Citrix Virtual Apps and Desktops, web page. Available (accessed 10.11.2018): https://www.citrix.fi/products/citrix-virtual-apps-and-desktops/.

[48] VPN, Virtual Private Network, web page. Available (accessed 10.11.2018): https://fi.wikipedia.org/wiki/VPN.

[49] Watchdog timer, Vahtii tietokoneen toimintoja. web page. Available (accessed 11.11.2018): https://en.wikipedia.org/wiki/Watchdog_timer.

[50] URL, Uniform Resource Identifier, web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/URI.

[51] Ethernet, Lähiverkkoratkaisu. web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Ethernet.

[52] Valokuitu. Tietoliikenneyhteys, web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Valokuitu.

[53] Matkapuhelinverkko. web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Matkapuhelinverkko.

[54] UV, Ultraviolettisäteily. web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Ultraviolettis%C3%A4teily.

[55] NTFS, New Technology File System, web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/NTFS.

[56] EXT4, fourth extended filesystem. web page. Available (accessed 11.11.2018): https://www.linux.fi/wiki/Ext4.

[57] APFS, Apple File System. web page. Available (accessed 11.11.2018): https://en.wikipedia.org/wiki/Apple_File_System.

[58] Bash, Bourne again shell, web page. Available (accessed 11.10.2018): https://fi.wikipedia.org/wiki/Bash.

[59] Windows PowerShell, Windowssin komentokehote, web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Windows_PowerShell.

[60] Singleton, Ainokainen, web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Singleton.

[61] API,Application programming interface, web page. Available (accessed 11.11.2018): https://fi.wikipedia.org/wiki/Ohjelmointirajapinta.

[62] JSX, Pieni mallikieli, web page. Available (accessed 15.11.2018): https://reactjs.org/docs/introducing-jsx.html.

[63] MAC-osoite, Media Access Control, web page. Available (accessed 16.11.2018): https://fi.wikipedia.org/wiki/MAC-osoite.

LIITE A: TYÖSSÄ KÄYTETTÄVÄT PAKETIT

Asennetut paketit:	Versionumero:	Lisenssi:
"@types/express":	"^4.16.0"	MIT
"@types/mocha":	"^5.2.1"	MIT
"@types/node":	"^10.11.1"	MIT
"@types/sqlite3":	"^3.1.3"	MIT
"@types/sequelize":	"^4.27.26"	MIT
"express":	"^4.16.3"	MIT
"sqlite3":	"^4.0.2"	BSD-3-Clause
"typescript":	"^3.1.1"	Apache License 2.0
"sequelize":	"^4.39.00"	MIT
"gulp"	"^3.9.1"	MIT
"gulp-typescript"	"^5.0.0-alpha.3"	MIT
"@types/merge2"	"^1.1.4"	MIT
"merge2"	"^1.2.2"	MIT
"sequelize-typescript"	"^0.6.6"	MIT
"sequelize-cli-typescript"	"^3.2.0-c"	MIT
"jsdoc"	"^3.5.5"	Apache-2.0
"gulp-jsdoc3"	"^2.0.0"	Apache-2.0
"jest"	"^23.6.0"	MIT
"ts-jest"	"^23.10.3"	MIT
"@types/jest"	"^23.3.2"	MIT
"pino"	"^5.6.4"	MIT

"@types/pino"	"^5.6.0"	MIT
"node-ipc"	"^9.1.1"	DBAD
"@types/node-ipc"	"^9.1.1"	MIT
"grpc"	"^1.15.1"	Apache-2.0

LIITE B: REACT KOMPONENTTI AJATTELUMALLI

```
function Tervetuloa(props) {
    return <h1>Terve, {props.name}</h1>;
}

function Applikaatio() {
  return (
    <div>
      <Tervetuloa name="Agentti" />
    </div>
  );
}

ReactDOM.render(<Applikaatio />, document.getElementById('root'));
```