



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MAURI MUSTONEN
SÄHKÖASEMAN ÄLYKKÄÄN ELEKTRONIIKKALAITTEIDEN
VIESTIEN TILAUS JA PROSESSOINTI
Diplomityö

Tarkastaja: Prof. Kari Systä

Jätetty tarkastettavaksi 17. touko-
kuuta 2018

TIIVISTELMÄ

MAURI MUSTONEN: sähköaseman älykkään elektroniikkalaitteiden viestien tilaus ja prosessointi

Tampereen teknillinen yliopisto

Diplomityö, 47 sivua, 3 liitesivua

Toukokuu 2018

Tietotekniikan koulutusohjelma

Pääaine: Ohjelmistotuotanto

Tarkastaja: Prof. Kari Systä

Avainsanat: IEC 61850, MMS, AMQP

Tiivistelmä on suppea, 1 sivun mittainen itsenäinen esitys työstä: mikä oli ongelma, mitä tehtiin ja mitä saatiin tulokseksi. Kuvia, kaavioita ja taulukoita ei käytetä tiivistelmässä.

Laita työn pääkielellä kirjoitettu tiivistelmä ensin ja käännös sen jälkeen. Suomenkielisellemme kandidaatintyölle pitää olla myös englanninkielinen nimi arkistointia varten.

ABSTRACT

MAURI MUSTONEN: Substation's intelligent electronic devices messages subscription and processing

Tampere University of Technology

Master of Science thesis, 47 pages, 3 Appendix pages

May 2018

Master's Degree Programme in Information Technology

Major: Software Engineering

Examiner: Prof. Kari Systä

Keywords: IEC 61850, MMS, AMQP

The abstract is a self-contained, concise description of the thesis: what was the problem, what was done, what was the result. Do not include charts or tables in the abstract.

ALKUSANAT

Mistä tämän diplomityönaiheen sain ja kiittää eri ihmisiä ketä työssä oli sidoshenkilöinä.

Tampereella, 19.4.2018

Mauri Mustonen

SISÄLLYSLUETTELO

1.	JOHDANTO	1
2.	TEORIA.....	3
2.1	IEC 61850 -standardi yhteiseen kommunikointiin	3
2.1.1	Standardin eri osat ja niiden merkitykset.....	4
2.1.2	Abstraktimallin käsitteet ja kuinka se tapahtuu	4
2.1.3	Loogisen noodin luokkien ja attribuuttien rakentuminen	8
2.1.4	Attribuuttien viittaus hierarkiassa.....	10
2.1.5	Attribuuttien funktionaalinen rajoite ja niistä muodostetut datajoukot	12
2.1.6	Viestien tilaus ja tilauksen konfigurointi	14
2.1.7	Raportointi-luokan määrittäminen ja toiminta.....	16
2.1.8	Viestin rakenne ja kuinka sen sisältö muodostuu	18
2.1.9	Abstraktimallin sovitukset MMS-protokollaan	21
2.2	Advanced Message Queuing Protocol (AMQP).....	22
2.2.1	Advanced Message Queuing -malli ja sen osat	23
2.2.2	Vaihde (exchange) ja reititysavain (routing-key).....	24
2.2.3	Suoravaihde (direct exchange).....	25
2.2.4	Hajautusvaihde (fanout exchange).....	25
2.2.5	Aihepiirivaihde (topic exchange).....	26
2.2.6	Otsikkovaihde (headers exchange)	27
2.2.7	Jonon määrittäminen ja viestien kuitaaminen.....	28
3.	PROJEKTIN LÄHTÖKOHDAT.....	29
3.1	Demoversio ja sen toiminta.....	29
3.2	Ongelmakohtat ja analysointi.....	31
4.	SUUNNITTELU	35
4.1	Kokonaiskuva.....	35
4.2	Järjestelmän hajautus ja arkkitehtuuri.....	36
4.3	Suorituskyky ja kielen valinta.....	37
4.4	Prosessoidun viestin muoto ja rakenne	38
5.	TOTEUTUS	40
5.1	Ohjelmiston toteutuksen valinta.....	40
5.2	Kielen valinta	40
5.3	RabbitMQ	40
5.4	Käytettävät kirjastot	40
5.4.1	libiec61850	40
5.4.2	rabbitmq-c.....	41
5.4.3	JSON-formatointi	41
5.5	Jatkokehitys.....	41
6.	ARVIOINTI	42

7. TULOKSET	43
8. YHTEENVETO	44
LÄHTEET	45
LIITE A: VIESTISTÄ PROSESSOITU JSON-RAKENNE	48

KUVALUETTELO

Kuva 1.	<i>IEC 61850 -standardin osat ja niiden väliset relaatiot.</i>	5
Kuva 2.	<i>Sähköaseman fyysisten laitteiden abstrahointi IEC 61850 -standardin käsitteillä (pohjautuu kuvaan [7, s. 17]).</i>	7
Kuva 3.	<i>Standardin käsitteiden hierarkkinen rakenne ja niiden nimeämisen esimerkki</i>	8
Kuva 4.	<i>IEC 61850 -standardin määrittämä viitteen rakenne.</i>	11
Kuva 5.	<i>Puskuroitu viestien tilausprosessi tilaajan ja IED-laitteen välillä.</i>	15
Kuva 6.	<i>Standardin määrittämä lähetetyn viestin rakenne (pohjautuu kuvaan [8, s. 104]).</i>	19
Kuva 7.	<i>BRCB-instanssi tarkkailee sille määritettyä datajoukkoa ja generoi viestin tapahtuman liipaistessa.</i>	20
Kuva 8.	<i>Toteutetun ohjelmiston osuus ja rooli käytettävässä kokonaisuudessa tietoliikenteen kannalta.</i>	22
Kuva 9.	<i>AMQ-mallin osat ja viestin kulku niiden läpi julkaisijalta tilaajalle (pohjautuu kuvaan [2, s. 11]).</i>	23
Kuva 10.	<i>Suoravaihde (engl. direct exchange), reitittää suoraan sidoksen reititysavaimeen mukaan (pohjautuu kuvaan [24]).</i>	25
Kuva 11.	<i>Hajautusvaihde (engl. fanout exchange), reitittää kaikkiin siihen sidottuihin jonoihin riippumatta reititysavaimesta (pohjautuu kuvaan [1]).</i>	26
Kuva 12.	<i>Aihepiirivaihde (engl. topic exchange), reitittää kaikkiin siihen sidottuihin jonoihin, joiden reitityskaava sopii viestin reititysavaimeen (pohjautuu kuvaan [25]).</i>	27
Kuva 13.	<i>Rubylla toteutetun demoversion arkkitehtuuri ja tiedonsiirto.</i>	29
Kuva 14.	<i>libIEC61850-kirjaston kerrosarkkitehtuurin komponentit, vihreällä Ruby toteutukseen lisätyt osat (pohjautuu kuvaan [15]).</i>	30
Kuva 15.	<i>Sekvenssikaavio kaikkien RCB-instanssien tilaukseen ja niiden viestien tallentamiseen yhdeltä IED-laitteelta Ruby-ohjelmalla.</i>	32
Kuva 16.	<i>Ruby-tulkin globaalin lukituksen toiminta, joka vuorottaa ajossa olevia säikeitä.</i>	34
Kuva 17.	<i>Suunnitellun järjestelmän toiminta ja viestin kulkeminen ja muoto eri osapuolten välillä.</i>	35

TAULUKKOLUETTELO

Taulukko 1.	<i>IEC 61850 -standardin pääkohtien ja niiden alakohtien dokumentit. ...</i>	5
Taulukko 2.	<i>IEC 61850 -standardin katkaisijaluokan XCBB -määrittäminen.</i>	9
Taulukko 3.	<i>IEC 61850 -standardin DPC-luokan määrittäminen.</i>	10
Taulukko 4.	<i>Osa IEC 61850 -standardin määrittämisestä funktionaalista rajoitteista (FC).</i>	12
Taulukko 5.	<i>Viitteen nimeäminen lyhenteellä funktionaalisen rajoituksen kanssa.</i>	14
Taulukko 6.	<i>BRCB-luokan määritetyt attribuutit ja niiden selitteet.</i>	17
Taulukko 7.	<i>RCB-luokan OptFlds-attribuutin arvot ja niiden selitteet.</i>	18

LYHENTEET JA MERKINNÄT

Kun työ on valmis. Lisää tähän kaikki lyhenteet aakkosjärjestyksessä.

ACSI	engl. <i>Abstract Communication Service Interface</i> , IEC 61850 -standardin käyttämä lyhenne kuvaamaan palveluiden abstraktimalleja
AMQP	engl. <i>Advanced Message Queuing Protocol</i>
FFI	engl. <i>Foreign Function Interface</i> , mekanismi, jolla ajettava ohjelma voi kutsua toisella kielellä implementoitua funktiota
GIL	engl. <i>Global Interpreter Lock</i> , tulkattavassa kielissä oleva globaali lukitus, joka rajoittaa yhden säikeen suoritukseen kerrallaan
HAL	engl. <i>Hardware Abstraction Layer</i> , laitteistoabstraktiotaso abstraktoimaan laitteen toiminnallisuus lähdekoodista
IED	engl. <i>Intelligent Electronic Device</i> , sähköaseman älykäs elektroninen laite, joka tarjoaa toimintoja monitorointiin ja kontrollointiin
MMS	engl. <i>Manufacturing Message Specification</i>
RCB	engl. <i>Report Control Block</i> , raporttien konfigurointiin ja tilaukseen tarkoitettu lohko asiakasohjelmalle
XML	engl. <i>Extensible Markup Language</i> , laajennettava merkintäkieli, joka on ihmis- ja koneluettava

1. JOHDANTO

Tämä diplomityö oli tehty Alsus Oy:lle, joka oli työn tekohetkellä tekijän työpaikka vuonna 2018. Tekijä valitsi työn aiheen mielenkiinnon ja ajankohdan sopiivuden takia. Työ liittyi sopivasti ajanhetkellä sen hetkisiin työtehtäviin.

Sähköverkko koostuu tuotantolaitoksista, sähkölinjoista ja sähköasemista. Sähköasemien tehtävä verkossa on toteuttaa erilaisia toiminnallisuuksia, kuten muuntaja, jakaminen ja verkon toiminnan tarkkailu. Lisäksi nykypäivänä asemien toiminnallisuutta voidaan seurata ja ohjata etänä. Sähköaseman yksi tärkeä tehtävä on suojata ja tarkkailla verkon toimivuutta ja vikatilanteessa esimerkiksi katkaista linjasta virrat pois. Tällainen vikatilanne voisi olla kaapelin poikkimeno ja virta pääsisi tätä kautta maihin.

Sähköasemien funktionaalisuutta ja ohjausta nykypäivänä toteuttaa niin sanottu älykäs elektroniikkalaitte (engl. Intelligent Electronic Device, lyhennetään IED). IED-laitte voidaan kytkeä ja konfiguroida toteuttamaan monta aseman eri funktionaalisuutta ja ne on myös kytketty aseman verkkoon. IED:t voivat kommunikoida verkon yli aseman muun logiikan ja muiden IED-laitteiden kanssa, ja näin toteuttaa aseman toiminnallisuutta. Nykypäivänä verkon nopeus ja mahdollistaa reaaliaikaisen kommunikoinnin asemilla sen eri laitteiden välillä. IED-laitteet voivat myös kommunikoida aseman verkosta ulospäin, esimerkiksi keskitettyyn ohjauskeskukseen. Yksi IED-laitte voidaan esimerkiksi konfiguroida hoitamaan sähkölinjan kytkimenä oloa, joka myös tarkkailee linjan toimintaa mitaamalla konfiguroituja arvoja, kuten jännitettä ja virtaa. Vikatilanteen sattuessa IED katkaisee linjan virrasta enemmän vahingon välttämiseksi. Linjan korjauksen jälkeen virta kytketään takaisin päälle.

Monen eri toimijan toimiessa laitteita tuottavalla allalla ja sähköasema suuren elektronisen laitteiden määrän takia. On määritetty maailmanlaajuinen standardi nimeltä IEC 61850, jonka tarkoitus on määrittää yhteinen kommunikointiprotokolla aseman kaikkien eri laitteiden ja valmistajien välille. Standardi määrittää eri valmistajien IED-laitteille samat yhteiset kommunikointiprotokollat, joita noudattamalla eri valmistajien laitteet sopivat yhteen.

Standardissa on määritetty säännöt, millä IED-laitteen ulkopuolinen ohjelma voi tilata viestejä verkon yli IED-laitteelta. Tilatut viestit voivat esimerkiksi sisältää mitattuja kolmivaihe jännitteitä tai muuta haluttua tietoa IED-laitteesta ja sen tilasta. Tässä työssä keskityttiin tämän asiakasohjelman suunnitteluun ja toteutukseen. Asiakasohjelman tarkoitus oli tilata viestit, prosessoida ne ja julkaista eteenpäin jonopalvelimelle muille ohjelmille saataviksi. Koska ohjelman toteutukseen tärkeäksi osaksi liittyy IEC 61850, ja käytetyn

jonopalvelimen standardit. Käsitellään nämä osiot työn teoriaosuudessa ensin ennen suunnittelua ja toteutusta.

Tämän työn tekijä oli jo ennen tämän työn aloitusta Alsus Oy:ssä toteuttanut yksinkertaiseen demoversion kyseisestä ohjelmasta. Toteutus oli puutteellinen ja siinä oli toimintahäiriöitä, mutta kuitenkin todisti eri osien toimivuuden mahdollisuuden ja opetti tekijää asian suhteen. Tässä työssä tätä demoversiota käytettiin pohjana kokonaan uuden toteutuksen suunnittelulle. Työssä analysoidaan sen toimintahäiriöitä ja mitkä ne aiheutti. Näitä tietoja käytettiin pohjana uuden toteutuksen liittyvien päätöksien tekemiseen.

Tämän työn tutkimustyön osuus on miettiä ja tutkia uuden toteutuksen arkkitehtuuria ja toteutusta. Tarkoitus on täyttää kaikki uudelle toteutukselle asetetut tarpeet ja estää demoversion liittyvät toimintahäiriöt. Työlle asetettiin tutkimuskysymyksiä, joita peilataan työn lopussa saavutettuihin tuloksiin ja pohditaan kuinka hyvin niihin päästiin. Työlle asetettiin seuraavat tutkimuskysymykset:

- Mitkä ohjelmiston arkkitehtuurin suunnittelumallit (engl. design patterns) olisivat sopivia tämän kaltaisen ongelman ratkaisemiseen? Mitä niistä pitäisi käyttää ja mitä ei?
- Kuinka järjestelmä hajautetaan niin että tiedon siirto eri osapuolten välillä on mahdollista ja joustavaa (push vs pull, message queue jne.)?
- Mitkä olivat syyt demoversion toimintahäiriöihin ja kuinka nämä estetään uudessa toteutuksessa?
- Järjestelmän hajautuksessa, mikä olisi sopiva tiedon jakamisen muoto eri osapuolten välillä?

2. TEORIA

Tässä osiossa lukijaa perehdytetään työn kannalta tärkeään teoriaan. Teoriaosuuden kokonaan lukemalla lukija ymmärtää, mitä IEC 61850 -standardi tarkoittaa sähköasemien kannalta ja mihin sitä käytetään. Lisäksi kuinka standardi määrittää viestien tilauksen mekanismit ulkopuoliselle ohjelmalle ja mitä siihen liittyy. Standardi on todella laaja ja tässä osuudessa siitä käsitellään vain tämän työn kannalta oleellinen asia. Tässä työssä toteutettu ohjelmisto julkaisi prosessoidut viestit eteenpäin jonopalvelimelle, mistä muut ohjelmat pystyivät tilaamaan viestejä. Käytetyn jonopalvelin toteutus pohjautui AMQP-standardiin (engl. Advanced Message Queuing Protocol). Teorian viimeisessä osassa perehdytään AMQP-standardiin ja kuinka jonopalvelin sen pohjalta toimii.

2.1 IEC 61850 -standardi yhteiseen kommunikointiin

Sähköasemilla nykypäivänä käytössä olevilla älykkäillä elektronisilla laitteilla (engl. Intelligent Electronic Device, lyhennetään IED) toteutetaan aseman toiminnallisuuden funktioita. Aseman toiminnallisuuteen liittyy sen kontrollointi ja suojaus. Aseman komponenttien suojauksen lisäksi, siihen kuuluu myös asemalta lähtevät sähkölinjat. Hyvä esimerkki sähköaseman suojauksesta on korkeajännitelinjan katkaisija, joka katkaisee virran linjasta vikatilanteissa, kuten linjan poikkimeno kaatuneen puun tai pylvään takia. Fyysisistä katkaisijaa ohjaa aseman automatiikka, joka toteutetaan IED-laitteilla. Eli IED-laite voi olla kytketty fyysisesti ohjattavaan laitteeseen [7, s. 63–64]. Koko sähköaseman toiminnallisuus koostuu monesta eri funktiosta, jotka on jaettu monelle IED-laitteelle. Jotta systeemi pystyy toimimaan, täytyy IED-laitteiden kommunikoida keskenään ja vaihtaa informaatiota toistensa kanssa. IED-laitteiden täytyy myös kommunikoida asemalta ulospäin erilliselle ohjausasemalle monitorointia ja etäohjausta varten [3, s. 1]. On selvää, että monimutkaisen systeemin ja monen valmistajien kesken tarvitaan yhteiset säännöt kommunikointia varten.

Maailmanlaajuisesti määritetty IEC 61850 -standardi määrittää sähköaseman sisäisen kommunikoinnin säännöt IED-laitteiden välillä. Standardi määrittää myös säännöt asemalta lähtevään liikenteeseen, kuten toiselle sähköasemalle ja ohjausasemalle [7, s. 10]. Ilman yhteistä standardia, jokainen valmistaja olisi vapaa toteuttamaan omat säännöt ja protokollat kommunikointiin. Seurauksena olisi, että laitteet eivät olisi keskenään yhteensopivia eri valmistajien kesken. Standardin tarkoitus on poistaa tämä ja määrittää yhteiset pelisäännöt kommunikoinnin toteuttamiseen [13, s. 1].

Todella tärkeä ja iso osa standardia on sähköaseman systeemin funktioiden abstrahointi mallien kautta. Standardi määrittää tarkasti kuinka abstraktit mallit määritellään aseman

oikeista laiteista ja niiden ominaisuuksista. Tarkoituksena on tehdä mallit tekniikasta ja toteutuksesta riippumattomaksi. Tämän jälkeen määritellään säännöt kuina abstrahoidut mallit mallinnetaan erikseen toimivaksi jollekin tekniikalle. Abstrahoituja malleja käytetään myös määrittämään sähköaseman IED-laitteiden ja aseman muiden osien konfigurointi. Tekniikasta riippumattomien mallien ansiosta standardi on pohjana tulevaisuuden laajennoksille ja tekniikoille. Uusien tekniikoiden ilmaantuessa, voidaan standardiin lisätä osa, joka mallintaa abstraktimallit kyseiselle tekniikalle [3, s. 2]. Tässä työssä standardin malleja ja palveluita käytettiin MMS-protokollamallinnuksen avulla (engl. Manufacturing Message Specification). MMS-protokolla on maailmanlaajuinen ISO 9506 -standardi, joka on määritetty toimivaksi TCP/IP:n pinon päällä [18]. Eli jokainen verkkoon kytketty IED-laitte tarvitsee IP-osoitteen kommunikointiin.

2.1.1 Standardin eri osat ja niiden merkitykset

IEC 61850 -standardi on todella laaja kokonaisuus. Tämän takia se on pilkottu erillisiin dokumentteihin, joista jokainen käsittelee omaa asiaansa. Historian saatossa standardiin on lisätty uusia dokumentteja laajentamaan standardia [12, 19] [5, s. 13]. Tämän työn kirjoitushetkellä standardiin kuului lisäksi paljon muitakin dokumentteja, esimerkiksi uusiin mallinnuksiin muille tekniikoille ja vesivoimalaitoksien mallintamiseen liittyviä dokumentteja. Laajuudesta huolimatta standardin voi esittää 10:llä eri pääkohdalla ja näiden alakohdilla. Taulukossa 1 on esitetty standardin pääkohdan dokumentit ja niiden alkupe- räiset englanninkieliset otsikot [17, s. 2] [12]. Kuvassa 1 on esitetty kaikki standardin eri osat ja niiden väliset relaatiot toisiinsa [7, s. 14] [5, s. 22]. Kuvaan on merkitty yhteinäisellä viivalla ne osat, jotka ovat tämän työn kannalta tärkeitä. Ja katkoviivalla ne, jotka eivät ole. Kuvassa käytetään standardin osien englanninkielisiä otsikoita.

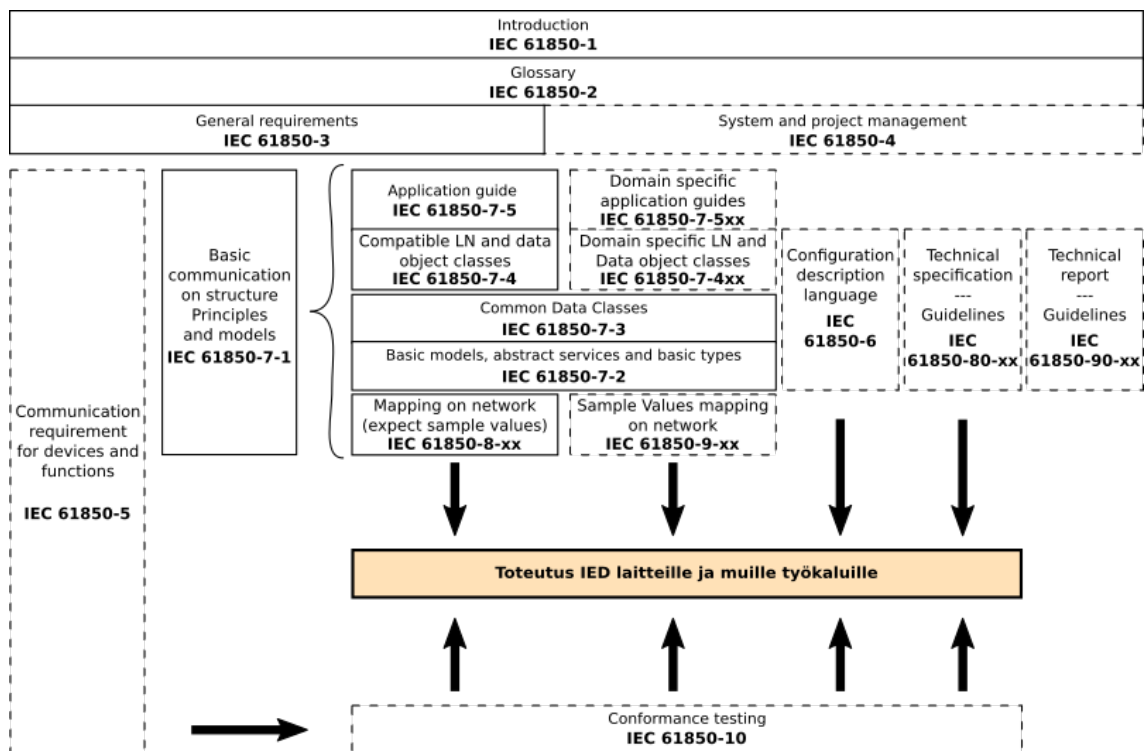
Standardin ensimmäiset osat 1–5 kattavat yleistä kuvaa standardista ja sen vaatimuksista. Osiossa 6 käsitellään IED-laitteiden konfigurointiin käytetty XML (engl. Extensible Markup Language) -pohjainen kieli [6, s. 7–8]. Tämä osuus ei ole tämän työn kannalta tärkeä ja sitä ei sen tarkemmin käsitellä. Osat 7-1–7-4 käsittelevät standardin abstraktia mallia, niiden palveluita ja kuinka se rakentuu. Abstrahoidut palvelut ja mallit standardissa lyhennetään ACSI (engl. Abstract Communication Service Interface), ja samaa lyhennettä käytetään tässä työssä [7, s. 72]. Osissa 8–9 ja niiden alakohdissa käsitellään abstraktimallien mallintamista erillisille protokollille, jolloin malleista tulee kyseisestä tekniikasta riippuvaisia. Tässä työssä käytettiin osaa 8-1, joka mallintaa abstrahit mallit MMS-protokollalle. Osa 10 käsittelee testausmenetelmiä, joilla voidaan varmistaa standardin määritysten noudattaminen. Tämä osuus ei myöskään ole tämän työn kannalta tärkeä, ja sitä ei teoriassa sen tarkemmin käsitellä. [7, s. 15]

2.1.2 Abstraktimallin käsitteet ja kuinka se tapahtuu

IEC 61850 -standardin lähtökohtana on pilkkoa koko sähköaseman toiminnallisuuden funktiot pieniksi yksilöiksi. Pilkotut yksilöt abstrahoidaan ja pidetään sopivan kokoisi-

Taulukko 1. IEC 61850 -standardin pääkohtien ja niiden alakohtien dokumentit.

Osa	Otsikko englanniksi
1	Introduction and overview
2	Glossary
3	General requirements
4	System and project management
5	Communication requirements for functions and device models
6	Configuration description language for communication in power utility automation systems related to IEDs
7-1	Basic communication structure - Principles and models
7-2	Basic information and communication structure - Abstract communication service interface (ACSI)
7-3	Basic communication structure - Common data classes
7-4	Basic communication structure - Compatible logical node classes and data object classes
8-1	Specific communication service mapping (SCSM) - Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3
9-2	Specific communication service mapping (SCSM) - Sampled values over ISO/IEC 8802-3
9-3	Precision time protocol profile for power utility automation
10	Conformance testing



Kuva 1. IEC 61850 -standardin osat ja niiden väliset relaatiot.

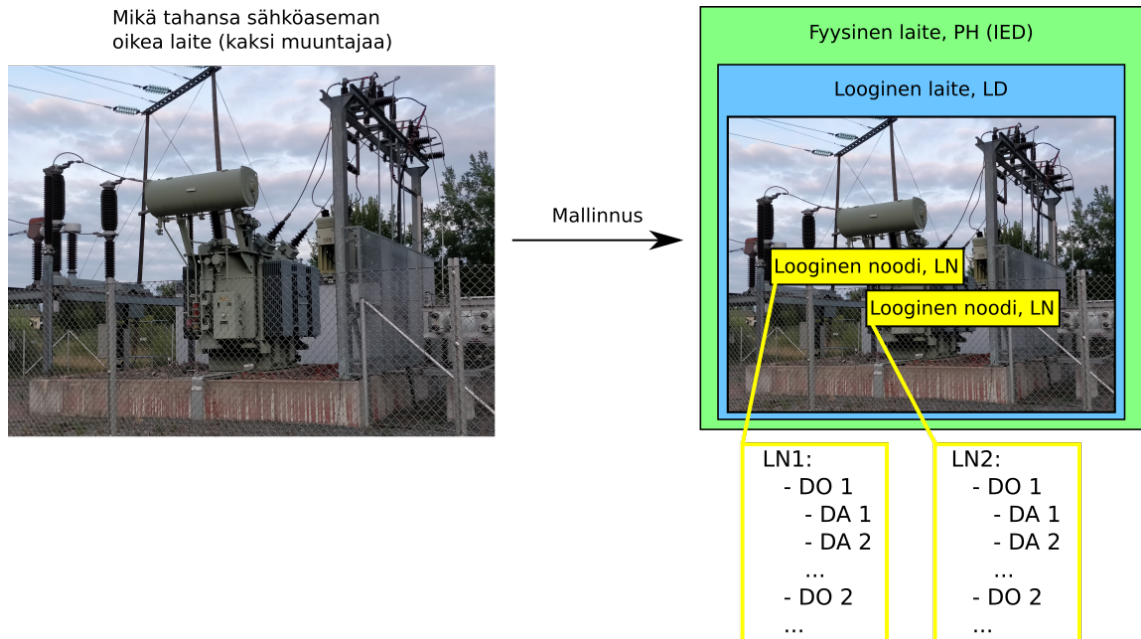
na, jotta ne voidaan konfiguroida esitettäväksi erillisellä IED-laiteella. Yksi aseman funktio voidaan hajauttaa monelle eri IED-laitteelle. Esimerkiksi linjan suojaukseen liittyvät komponentit, katkaisija (engl. circuit braker) ja ylivirtasuojia (engl. overcurrent protec-

tion). Toimiakseen yhdessä, laitteiden täytyy vaihtaa informaatiota keskenään verkon yli [7, s. 31]. Standardi määrittää seuraavat käsitteet sähköaseman funktioiden mallintamiseen:

- fyysinen laite (engl. physical device, lyhennetään PD),
- looginen laite (engl. logical device, lyhennetään LD),
- looginen noodi (engl. logical node, lyhennetään LN),
- dataobjekti (engl. data object, lyhennetään DO),
- data-attribuutti (engl. data attribute, lyhennetään DA).

Käsitteet muodostavat mallista hierarkisen puurakenteen ja ne on listattu hierarkisessa järjestyksessä. Puun juurena on fyysinen laite, sen alla voi olla yksi tai useampi looginen laite, loogisen laitteen alla yksi tai useampi looginen noodi jne. Käsitteillä standardissa virtualisoidaan aseman funktiot, esimerkiksi suojaus. Kuvassa 2 on esitetty, kuinka sähköaseman fyysiset laitteet voidaan mallintaa standardin määrittämällä käsitteillä. Samaa periaatetta käytetään kaikille aseman laitteille. Kuvassa ensin uloimpana on fyysinen laite, joka ohjaa laitteita ja tarkkailee niiden toimintaa. Tämä laite voi olla IED-laite, joka on myös samalla kytketty aseman verkkoon ja sillä on IP-osoite. Yksi IED-laite voi olla samaan aikaan kytkettynä aseman moneen muuhun oikeaan laitteeseen. Tämän jälkeen mallinnetaan aseman joukko laitteita loogiseksi laitteeksi. Tällainen voi esimerkiksi olla tietyn jännitetasen komponentit, kuten katkaisijat, muuntajat jne. (engl. bay). Kuvassa kaksi muuntajaa on mallinnettu yhdeksi loogiseksi laitteeksi, koska ne kuuluvat samaan jännitetasoon. Looginen laite koostuu loogisista noodeista se mallintaa jotakin aseman ohjattavaa yksittäistä laitetta. Kuvassa kaksi muuntajaa mallinnetaan loogisiksi noodeiksi. Jotta oikeaa fyysistä muuntajaa voidaan kuvata mallilla. Täytyy siitä pystyä esittämään mitattavia tai kuvaavia arvoja, esimerkiksi mitatut jännitteen arvot. Näihin tarkoituksiin käytetään käsitteitä dataobjekti ja data-attribuutti. Looginen noodi koostuu dataobjekteista ja dataobjekti koostuu data-attribuuteista. Data-attribuutti esittää yhtä mitattavaa tai kuvaavaa arvoa laitteesta, esimerkiksi sen hetkinen jännite tai laitteen tila. Dataobjekti on tapa koostaa yhteen kuuluvat data-attribuutit saman käsitteen alle, esimerkiksi mittaukseen tai ohjaukseen liittyvät data-attribuutit. [4, s. 2] [5, s. 24]

IEC 61850 -standardin käsitteiden avulla sähköaseman laitteet ja funktiot voidaan esittää malleilla. Malleja voidaan käyttää IED-laitteiden konfiguroinnin määrittämiseen ja tietona, jotka voidaan siirtää verkon yli laitteelta toiselle. Jotta käsitteitä voidaan käyttää konfigurointiin ja kommunikointiin standardi määrittää lisää tarkuutta käsitteisiin ja kuinka niitä käytetään. MMS-protokollan kanssa fyysinen laite yksilöidään IP-osoiteella ja tätä käsitettä ei käytetä kommunikointiin tai konfigurointiin. Fyysisen laitteen käsite on olemassa standardissa, jotta se voidaan pitää abstraktina toteutettavasta tekniikasta. Looginen laite yksilöidään nimellä, joka on yksilöllinen IED-laitteessa. Standardi ei ota kantaa loogisen laitteen nimeämiseen. Looginen noodi yksilöidään IED-laitteella myös nimellä. Looginen noodi esitetään IED-laitteella jonkin standardissa määritettyjen luokan

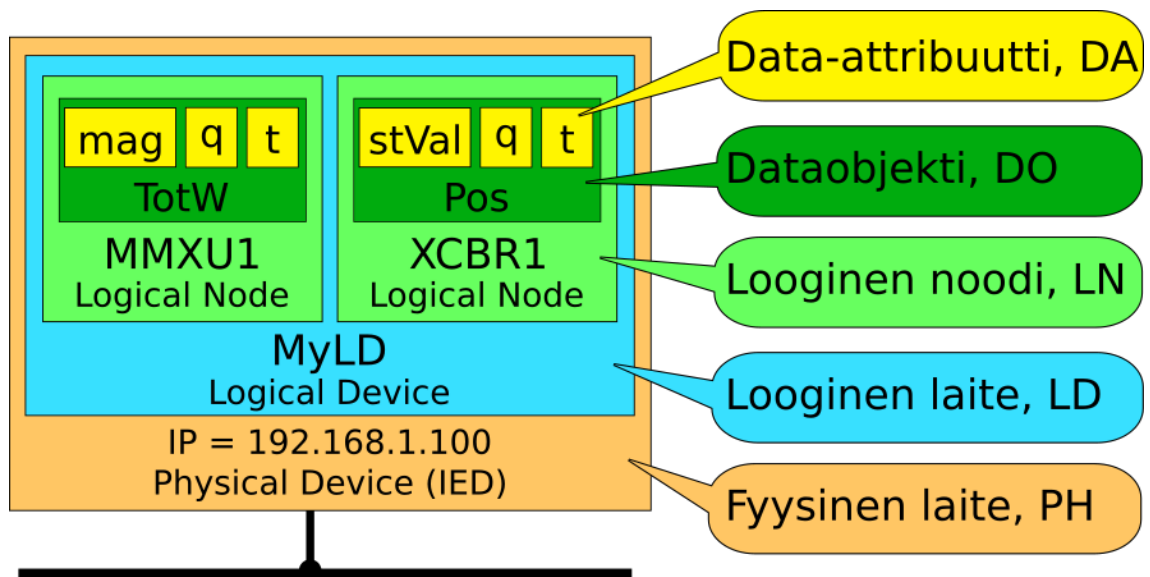


Kuva 2. Sähköaseman fyysisten laitteiden abstrahointi IEC 61850 -standardin käsitteillä (pohjautuu kuvaan [7, s. 17]).

instanssina. Standardin osassa 7-4 määritetään iso lista valmiita luokkia käytettäväksi eri laitteen esittämiseen. Aikaisemmassa esimerkissä katkaija on määritetty luokkana tyyppiltään XCBR (engl. circuit braker) [10, s. 105–106]. Sähköaseman insinööri, joka konfiguroi IED-laitteen, määrittää konfiguraatiodostossa, että kytketty katkaisija esitetään XCBR-luokan instanssina ja nimeää sen standardin ohjeiden mukaan. Näin IED-laite tietää mitä laitetta se esittää ja ohjaa. IED-laitteessa kaikki eri luokkien instanssit yksilöidään nimillä ja niitä käytetään kun olioon viitataan esimerkiksi palvelukutsulla tai konfiguraatiolla. Looginen noodi koostui dataobjekteista. Standardissa dataobjektit on myös määritetty luokkina, joista tehdään instansseja. Erona on, että loogisen noodin luokkan tyyppi määrittää mitä dataobjektin luokkia insansioidaan, ja millä nimellä ne esitetään loogisen noodin instanssissa. Standari määrittää dataobjektien luokkien tyypit standardissa osassa 7-3. Dataobjekti koostuu data-attribuuteista. Kuten loogisen noodin luokan tyyppi, dataobjektin luokka määrittää käytettävät data-attribuutit ja niiden nimet. Tällä kertaa data-attribuutti ei ole välttämättä suoraan ole luokka. Data-attribuutit voivat olla primitiivisiä tyyppisiä, kuten integer ja float. Tai ne voivat olla ns. rakennettuja data-attribuutteja (enlg. constructed attribute classes), jotka pitävät sisällään tarkempia data-attribuutteja. Hyvä esimerkki on data attribuutti nimeltään q, jonka tyyppi on Quality. Standardin mukaan tällä tyyppillä on vielä aliattribuutteina mm. validity, detailQual jne [9, s. 11]. Standardi ei rajoita että dataobjektin alla pitää aina data-attribuutteja. Joissakin tapauksissa dataobjektin alla on toinen dataobjekti ja tämän alla vasta data-attribuutit. Kaikki luokkien tyyppisiin määritetyt kentät ja niiden nimet voi löytää standrdista. Kuinka yllä esitettyjen luokkien rakenne hierarkisesti rakentuu, käydään vielä tarkemmin läpi seuraavassa kappaleessa esimerkin kanssa. [5, 7, 8, 9]

2.1.3 Loogisen noodin luokkien ja attribuuttien rakentuminen

IEC 61850 -standardissa kaikki luokat määritellään taulukoilla, joissa on standardoitu kentän nimi, tyyppi, selitys ja onko kenttä optionaalinen. Tässä teoriaosuudessa mennään syvemmälle luokkien määrittämiseen. Ja esitetään esimerkkinä kuinka standardin pohjalta instansioitu looginen noodi ja sen alitason dataobjektit ja data-attribuutit rakentuvat. Esimerkissä käytetään kuvan 3 rakennetta. Nimet ja luokkien instanssit konfiguroidaan IED-laitteelle XML-pohjaisella konfiguraatiotiedostolla. Tämä määrittellään standardin osassa 6. Kuvassa 3 fyysinen laite on IED-laite ja siihen verkossa viitataan IP-osoitteella 192.192.1.100. IED-laitteelle on konfiguroitu looginen laite nimeltä MyLD. Eri loogiset laitteet IED-laitteella yksilöi vai sen nimi. Loogisella laitteella on kaksi instanssia loogisen noodin luokista nimillä MMXU1 ja XCBR1. MMXU1 instanssi on tyyppiä MMXU (engl. measurement) [10, s. 57–58] ja XCBR1 on tyyppiä XCBR (engl. circuit breaker). Kyseessä on siis vastaavasti mittaukseen liittyvä laite ja aikaisemmin mainittu linjan katkaisija. XCBR1 loogisella noodilla on dataobjekti nimeltään Pos (engl. position), joka on tyyppiä DPC (engl. controllable double point). Ja MMXU1 nimeltään TotW (engl. total active power), joka on tyyppiä MV (engl. measured value). Loogisilla noodeilla on määritetty enemmänkin dataobjekteja eri nimillä, mutta kuvassa 3 on esitetty vai yhdet yksinkertaisuuden takia. Pos dataobjektilla on data-attribuutit nimeltään stVal, q ja t. Ja TotW dataobjektilla on data-attribuutit mag, q ja t. Esimerkin data-attribuutti q on tyyppiä Quality, jolla on alidata-attribuutteja ja attribuutti StVal on tyyppiä boolean.



Kuva 3. Standardin käsitteiden hierarkinen rakenne ja niiden nimeämisen esimerkki

Standardissa osassa 7-4 on lista kaikista sen määrittämistä loogisen noodin luokista eri tarkoituksiin. Taulukossa 2 on esitetty XCBR-luokan määrittäminen. Taulukosta voi nähdä luokan instanssille määritetyt kenttien nimet ja viimeinen sarake M/O/C, kertoo onko kenttä pakollinen (Mandatory, M), optionaalinen (Optional, O), vai konditionaalinen (Conditional, C) [10, s. 106]. Taulukosta voi nähdä kuvan 3 esimerkin XCBR1-instanssin data-objektin

nimeltä Pos ja sen tyypin DPC. Standardissa dataobjektien luokkia kutsutaan yleisiksi luokiksi (engl. Common Data Class, lyhennetään CDC). Näin sen takia, koska samaa dataobjektin luokkaa voidaan käyttää monessa eri loogisen noodin luokassa. Standardin dataobjektin luokat on tarkoitettu kerätä yhteen samaan asiaan liittyvät data-attribuutit. CDC-luokkien määrittymiset löytyvät standardin osasta 7-3 [5, s. 26]. Joillakin CDC-luokkien attribuutteina voi olla vielä muita CDC-luokkia. Tällöin standardissa puhutaan yleisistä aliluokista (engl. sub data object). Esimerkkinä tästä on CDC-luokka WYE, jolla on attribuuttina phsA niminen kenttä, joka on tyyppiä CMV. CMV on CDC-luokka, jolla on taas omat data attribuutitnsa. [8, s. 51,61] [9, s. 36]

Taulukko 2. IEC 61850 -standardin katkaisijaluokan XCBR -määrittys.

Data objektin nimi	Englanniksi	CDC-luokka	M/O/C
Selitys			
EEName	External equipment name plate	DPL	O
Tila informaatio			
EEHealt	External equipment health	ENS	O
LocKey	Local or remote key	SPS	O
Loc	Local control behaviour	SPS	M
OpCnt	Operation counter	INS	M
CBOpCap	Circuit breaker operating capability	ENS	O
POWCap	Point on wave switching capability	ENS	O
MaxOpCap	Circuit breaker operating capability	INS	O
Dsc	Discrepancy	SPS	O
Mitatut arvot			
SumSwARs	Sum of switched amperes, resettable	BRC	O
Kontrollit			
LocSta	Switching authority at station level	SPC	O
Pos	Switch position	DPC	M
BlkOpn	Block opening	SPC	M
BlkCls	Block closing	SPC	M
ChaMotEna	Charger motor enabled	SPC	O
Asetukset			
CBTmms	Closing time of braker	ING	O

Taulukossa 3 on esitetty XCBR-luokan Pos-attribuutin, DPC-luokan määrittys [9, s. 44]. Taulukosta voi nähdä kuvan 3 esimerkissä esitetyt data-attribuutit stVal, q ja t ja niiden tyypit. Attribuuttien tyyppejä on paljon enemmänkin ja lukija voi tarvittaessa tarkistaa kaikki tyypit standardista. Tällä periaatteella standardi rakentaa kaikki muutkin luokat hierarkisesti ja sen avulla voidaan selvittää mitä dataobjekteja looginen noodi sisältää, mitä data-attribuutteja mikäkin data objekti sisältää. Taulukossa 3 on myös määritetty data-attribuuttien funktionaaliset rajoitteet (engl. Functional Constraint, lyhennetään FC), sekä mahdolliset liipaiseimet (engl. trigger options, lyhennetään TrgOp). Nämä kaksi asiaa käsitellään teoriassa tuonnempana.

Kaikkien yllämainittujen luokkien kenttien määrittysten lisäksi standardi määrittää palveluita jokaiselle luokkatyypille erikseen. Määritetyt palvelut ovat abstrakteja ja ne malli-

Taulukko 3. IEC 61850 -standardin DPC-luokan määrittäminen.

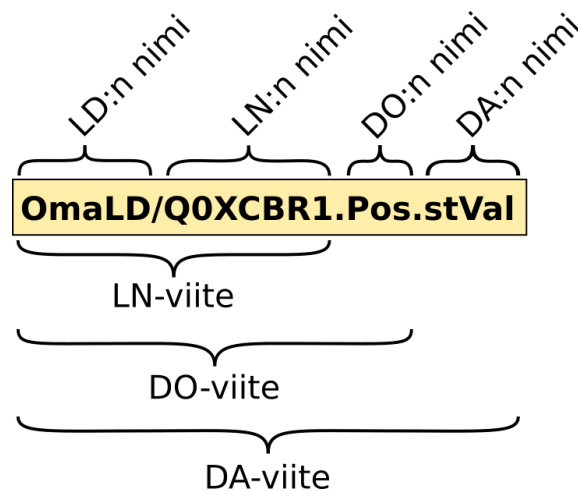
Data attribuutin nimi	Tyyppi	FC	Liipaisin (TrgOp)
Tila ja ohjaus			
origin	Originator	ST	
ctlNum	INT8U	ST	
stVal	CODEC ENUM	ST	dchg
q	Quality	ST	qchg
t	TimeStamp	ST	
stSeld	BOOLEAN	ST	dchg
opRcvd	BOOLEAN	OR	dchg
opOk	BOOLEAN	OR	dchg
tOpOk	TimeStamp	OR	
Vaihtoehtoinen ja estäminen			
subEna	BOOLEAN	SV	
subVal	CODED ENUM	SV	
subQ	Quality	SV	
subID	VISIBLE STRING64	SV	
blkEna	BOOLEAN	BL	
Asetukset, selitys ja laajennos			
pulseConfig	PulseConfig	CF	dchg
ctlModel	CtlModels	CF	dchg
sboTimeOut	INT32U	CF	dchg
sboClass	SboClasses	CF	dchg
operTimeout	INT32U	CF	dchg
d	VISIBLE STRING255	DC	
dU	UNICODE STRING255	DC	
cdcNs	VISIBLE STRING255	EX	
cdcName	VISIBLE STRING255	EX	
dataNs	VISIBLE STRING255	EX	

netaan tekniikalle erillisellä standardin osalla. Palveluita voi ajatella esimerkiksi suoritettavina funktioina. Esimerkkinä palveluista kaikille dataobjekteille on mm. GetDataValues, joka palauttaa kaikki dataobjektin attribuuttien arvot. SetDataValues kirjoittaa annetut data-attribuuttien arvot. Ja GetDataDirectory palauttaa kaikki data-attribuuttien viitteet kyseisessä dataobjektissa. Näitä ja muita abstrahoituja malleja viitataan standardissa lyhentellä ACSI (engl. abstract communication service interface) [8, s. 15,45–46] [7, s. 26].

2.1.4 Attribuuttien viittaus hierarkiassa

IEC 61850 -standardi määrittää erilaisia palvelukutsuja eri luokkatyypeille. Jotta kutsuja voitaisiin tehdä verkon yli IED-laitteelle ja sen arvoja lukea ja asettaa hierarkiassa. Pitää tiettyyn data-attribuuttiin tai data-objektiin voida viitata yksilöivästi. Siksi standardissa on määritetty viittausformaatti, jota käytetään kun IED-laitteelle kutsuja tehdään. Kutsussa olevan viitteen perusteella IED-laite tietää, mihin instanssiin kutsu kohdistuu ja pystyy toimimaan sen mukaan. Tärkeää on myös mainita, että määritettyjä kutsuja lukemiseen ja asettamiseen voidaan käyttää useaan data-attribuuttiin yhtä aikaa. Kutsuja ei ole rajoitettu

käsittelmään yhtä data-attribuuttia kerrallaan. Viitten lisäksi aikaisemmin mainittu funktionaalinen rajoite, kertoo mihin data-attribuutteihin kutsu kohdistuu. Tämä tullaan käsittelemään tarkemmin kappaleessa 2.1.5. Kuvassa 4 on esitetty kuinka standardi määrittää viitteen muodostumisen loogisesta laitteesta data attribuuttiin asti. Viite alkaa loogisen laitteen nimestä ja ei sisällä fyysistä laitetta. Tähän on syynä se, että fyysisellä laitteella ei ole nimeä ja sillä on yksilöivä IP-osoite MMS-protokollan tapauksessa. Fyysinen laite on standardissa abstraktio laitteesta, kuten IED:stä. [7, s. 93].



Kuva 4. IEC 61850 -standardin määrittämä viitteen rakenne.

Viite muodostuu suoraan laitteessa olevien luokkien instanssien nimien ja hierarkian mukaan. Loogisen laitteen (LD) ja loogisen noodin (LN) erottimena käytetään kauttaviivaa, ja muiden osien erottimena käytetään pistettä. Loogisella laitteella on aseman insinöörin määrittämä oma nimi, mutta kuitenkin alle 65 merkkiä. Muuten loogisen laitteen nimeen standardi ei puutu. Loogisen noodin instanssin nimi koostuu alku-, keski- ja loppuosasta. Alkuosan käyttäjä voi itse päättää, kuvassa 4 Q0. Voi sisältää numeroita ja kirjaimia, mutta täytyy alkaa kirjaimella. Keskiosan täytyy olla loogisen luokan nimi, josta instanssi on tehty. Tässä tapauksessa jo aikaisemmin mainittu katkaisijan luokka, XCBR. Tämä osuus on aina 4 kirjainta pitkä ja on aina isoilla kirjaimilla. Loppuosan instanssin numeerinen arvo, joka ei sisällä kirjaimia. Loppuosan käyttäjä voi itse päättää, jonka ei tarvitse välttämättä olla juokseva numero. Alku- ja loppuosan yhteenlaskettu merkkien pituus täytyy olla alle 13 merkkiä, eli koko loogisen noodin nimen pituus voi olla maksimissaan 17 merkkiä. Data objektien (DO) ja attribuuttien (DA) niminä käytetään standardin määrittämiä nimiä, jotka määritetään niitä vastaavissa luokissa osissa 7-3 ja 7-4 (katso taulukot 2 ja 3). Riippuen viittauksesta, näistä muodostuu loogisen noodin viite, dataobjektin viite ja data attribuutin viite. Jos data-objektin alla on toinen dataobjekti, jonka alla on vasta itse data-attribuutit. Viittausta vain jatketaan instanssien nimiä liittämällä toisiinsa pisteellä aina data-attribuuttiin asti. Samoin toimitaan kun data-attribuutti on tyypiltään rakennettu tyyppi, kuten Quality, jolla on alidata-attribuutteja. [8, s. 181–182] [7, s. 93–95]

Standardissa määritetään kaksi näkyvyysaluetta (engl. scope) viittaukselle, jotka ovat palvelin- ja looginen laite -näkyvyysalueet. Palvelin tässä yhteydessä tarkoittaa verkkoon

kytkettyä laitetta, eli IED-laitetta. Palvelinnäkyvyysalueelle viitataan ottamalla viittauksesta pois loogisen laitteen nimi. Eli kuvassa 4 viittaus tulisi muotoon /Q0XCBR1.Pos.stVal. Edellemainittua viittausta käytetään silloin, kun loogisen noodin instanssi sijaitsee loogisen laitteen ulkopuolella, mutta kuitenkin palvelimella. Looginen laite -näkyvyysalueessa viittaus sisältää loogisen laitteen nimen ennen kauttaviivaa, toisin kuin palvelin-näkyvyysalueessa. Esimerkiksi kuvassa 4 oleva viittaus OmaLD/Q0XCBR1.Pos.stVal. Loogisen laitteen -näkyvyysaluetta käytetään silloin kun loogisen noodin instanssi sijaitsee loogisen laitteen sisällä sen hierarkiassa. Tässä työssä jatkossa käytetään pelkästään loogisen laitteen -näkyvyysaluetta. [8, s. 183]

Standardi määrittää maksimipituuksia viittauksille. Seuraavaksi kerrotut pituusmäärytykset ovat voimassa kummallekin edelle mainitulle näkyvyysalueen viittaukselle. Ennen kauttaviivaa saa olla maksimissaan 64 merkkiä. Tämän jälkeen kauttaviiva, josta seuraa uudelleen maksimissaan 64 merkkiä. Eli koko viittauksen maksimipituus saa olla enintään 129 merkkiä, kauttaviiva mukaan lukien. [8, s. 24,183]

2.1.5 Attribuuttien funktionaalinen rajoite ja niistä muodostetut datajoukot

Standardin CDC-luokat, määrittävät käytettävät data-attribuutit (katso taulukko 3). Nämä luokat määrittävät myös jokaiselle data-attribuutille aikaisemmin mainitun funktionaalisen rajoitteen (engl. functional constraint, lyhennetään FC). Funktionaalinen rajoite kuvaa attribuutin käyttötarkoitusta ja sitä mitä palveluita attribuuttiin voidaan käyttää. Esimerkiksi kaikki attribuutit, jotka liittyvät laitteen tilaan (engl. status), niillä on funktionaalinen rajoite ST (standardissa engl. status information). Standardi määrittää paljon erilaisia funktionaalisia rajoitteita, jotka ovat kaikki kahden ison kirjaimen yhdistelmiä. Taulukossa 4 on esitetty joitain tärkeimpiä funktionaalisia rajoitteita. Funktionaalinen rajoite määrittää myös, onko attribuutti kirjoitettava tai luettava [8, s. 54].

Taulukko 4. Osa IEC 61850 -standardin määrittämistä funktionaalisista rajoitteista (FC).

Lyhenne	Selite	Luettava	Kirjoitettava
ST	Laitteen tilatieto (status)	Kyllä	Ei
MX	Mittaustieto (measurands)	Kyllä	Ei
CF	Laitteen asetusarvo (configuration)	Kyllä	Kyllä
DC	Selitystieto (description)	Kyllä	Kyllä

Funktionaalista rajoitetta käytetään IED-laitteelle tehtävässä kutsussa viitteen kanssa suodattamaan mitä data-attribuutteja tehty kutsu koskee. Funktionaalinen rajoite on pakollinen tieto kutsuissa, jotka lukevat tai kirjoittavat arvoja. Seuraavaksi esitetään esimerkki kuinka yhdellä kutsulla viitataan moneen data-attribuuttiin. Esimerkkinä otetaan kuvassa 4 olevasta viitteestä osa, joka viittaa data-objektiin. Eli OmaLD/Q0XCBR1.Pos, jolloin viite on DO-viite. Kutsun vaikutusalue on aina hierarkiassa alaspäin. Eli nyt viit-

teellä viitataan Pos-dataobjektin kaikkiin alla oleviin data-attribuutteihin. Katso taulukko 3, jossa on esitetty kaikki Pos-dataobjektin alla olevat data-attribuutit, johon nyt viitataan. Huomiona, jos viittauksen alla olisi alidata-objekteja, niidenkin data-attribuutit kuuluvat viittauksen piiriin. Viittauksen vaikutuksen voi siis ajatella jatkuvan viittauskohdasta alaspäin rekursiivisesti kaikkiin ali-instansseihin. Funktionaalista rajoitetta käytetään suodattamaan kaikista viitatuista data-attribuuteista ne, jotka halutaan kirjoittaa tai lukea. Esimerkkinä jos kutsuun viitteellä `OmaLD/Q0XCBR1.Pos` lisättäisiin funktionaalinen rajoite `ST`. Rajoitettaisiin kutsu koskemaan Pos-dataobjektin alidata-attribuuteista vain niitä attribuutteja, joilla on funktionaalinen rajoite `ST`. Eli taulukon 3 mukaan attribuutit olisivat `origin`, `ctlNum`, `stVal`, `q`, `t` ja `stSeld`. Muut data-attribuutit suodatetaan pois kutsun vaikutuksesta. Sama suodatus tapahtuu rekursiivisesti hierarkiassa alaspäin kaikille alidata-attribuuteille. Esimerkissä olevat arvot voisi vain lukea, ei kirjoittaa. Tämä sen takia koska taulukon 4 mukaan funktionaalinen rajoite `ST` sallii vain lukemisen. IEC 61850 -standardissa määritetään funktionaalinen rajoite `XX`, joka on sama kuin mikä tahansa muu funktionaalinen rajoite. Kuitenkin standardin osassa 8-1 joka tekee mallinnuksen MMS-protokollalle, tämä ei ole tuettu toiminnallisuus. Eli toisin sanoen, jos MMS-protokollan kanssa halutaan lukea kaikki yhden data-objektin data-attribuutit. Joudutaan tekemään kutsu jokaista data-objektin funktionaalista rajoitetta kohti.

Viittauksen ja funktionaalisen rajoitteen avulla siis suodatetaan rekursiivisesti hierarkiassa alaspäin olevia data-attribuutteja. IEC 61850 -standardissa on määritelty nimitykset käytettäväksi kun jotakin viittausta suodatetaan funktionaalisella rajoitteella. Nämä ovat `FCD` (engl. functional constrained data) ja `FCDA` (engl. functional constrained data attribute). Nämä nimitykset ovat standardissa vain käsite, joka ei mallinnu mitenkään tekniikalla. Taulukossa 5 on esitetty viittauksia eri tyyppisiin instansseihin funktionaalisella rajoitteella. Taulukosta selviää viitattu instanssi dataobjekti (`DO`) tai data-attribuutti (`DA`), instanssin tyyppi ja käytetty nimitys viittaukselle `FCD` tai `FCDA`. `FCD` nimitys on silloin kun vain hierarkian ensimmäistä dataobjekti rajoitetaan funktionaalisesti. `FCDA` nimitys on käytössä kaikille muille viittauksille hierarkiassa alaspäin, joita rajoitetaan funktionaalisesti. Huomaa taulukossa 5 viittaus `OmaLD/MMXU1.PhV.phsA`, joka viittaa `PhV` dataobjektin alidataobjektiin. Tämä on `FCDA`-viittaus, vaikka kyseessä onkin dataobjekti. Ainoa ero `FCD:n` ja `FCDA:n` nimitysten välillä on vain se, että `FCD`-viittaus on aina vain hierarkian ensimmäiseen dataobjektiin ja `FCDA`-viittaus siitä eteenpäin hierarkiassa. Riippumatta mitä tyyppisiä viitattut instanssit hierarkiassa alaspäin ovat. [8, s. 55] [11, s. 63]

Funktionaalista rajoitetta käytetään viitteen kanssa suodattamaan viitatuista kohdasta alaspäin kaikki data-attribuutit. Tätä toiminnallisuutta käytetään hyväksi, kun tehdään kirjoittavia tai lukevia kutsuja ja rajoitetaan kutsulla vaikutettavia data-attribuutteja. Tätä samaa mekanismia käytetään hyväksi kun IED-laitteeseen määritellään datajoukkoja. IEC 61850 -standardissa datajoukko koostuu joukosta IED-laitteessa olemassa olevista data-attribuuteista. Datajoukko on tapa koostaa yhteen kiinnostavat data-attribuutit IED-laitteelta. Datajoukko nimetään ja sijoitetaan IED-laitteen hierarkiaan. Näin siihen

Taulukko 5. Viitteen nimeäminen lyhenteellä funktionaalisen rajoitteen kanssa.

FC	Viite	Instanssi	Tyyppi	Nimitys
ST	OmaLD/XCBR1.Pos	DO	DPC	FCD
ST	OmaLD/XCBR1.Pos.t	DA	TimeStamp	FCDA
ST	OmaLD/XCBR1.Pos.ctlNum	DA	INT8U	FCDA
MX	OmaLD/MMXU1.PhV	DO	WYE	FCD
MX	OmaLD/MMXU1.PhV.phsA	DO	CMV	FCDA
MX	OmaLD/MMXU1.PhV.phsA.t	DA	TimeStamp	FCDA

voidaan viitata kutsuilla kuten mihin tahansa muuhun hierarkian instanssiin. Datajoukot IED-laitteelle rakennetaan käyttämällä FCD ja FCDA viitteitä. Datajoukko koostuu siis joukosta FCD- ja FCDA -viitteitä. Jokaisella viitteellä on jokin funktionaalinen rajoite, joka suodattaa viitteen alla olevat attribuutit ja sisällyttää ne kyseiseen datajoukkoon. Esimerkkinä datajoukon rakentamisesta taulukon 5 viittet. Näistä viitteistä voitaisiin rakentaa oikea standardin mukainen datajoukko, nimetä se nimellä Testi1, ja lisätä IED-laitteen hierarkiaan kohtaan OmaLD/LLN0.Testi1. Nyt datajoukkoon voisi viitata ja vaikka lukea kaikki sen arvot yhdellä kertaa. Jotta datajoukko saadaan näin tehtyä, tieto tästä pitäisi lisätä IED-laitteen asetustiedostoon. Datajoukkoja IED-laitteessa käytetään muodostamaan joukkoja tärkeistä data attribuuteista, joita voidaan esimerkiksi lukea ja kirjoittaa yhdellä kutsulla. Datajoukkoja käytetään myös tilattavien viestien sisältönä. Viestejä voi standardin mukaan tilata vain datajoukoista olevista data-attribuuteista. [8, s. 61–68]

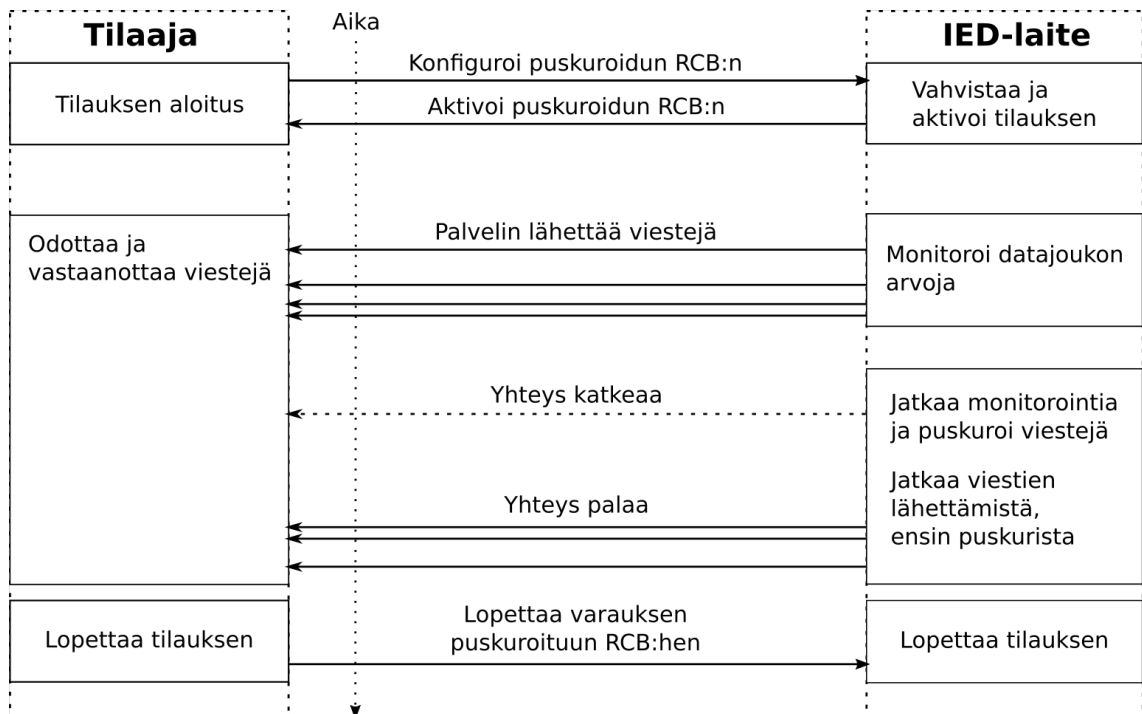
2.1.6 Viestien tilaus ja tilauksen konfigurointi

IEC 61850 -standardi määrittää, kuinka IED-laitteen ulkopuolinen ohjelma voi tilata kiinnostavien data-attribuuttien arvoja verkon yli. Viesti voidaan esimerkiksi lähettää tilaajalle, kun mitatun jännitteen arvo muuttuu. Kyseessä on tilaaja-julkaisija arkkitehtuurimalli, jossa ulkopuolinen ohjelma on tilaaja ja IED-laitte on julkaisija. Standardi määrittää, että viestejä voidaan tilata vain datajoukoissa viitatuilla data-attribuuteilta. Milloin viestin lähetys tilaajalle tapahtuu, riippuu siitä kuinka tilaaja liipaisimet asettaa tilauksen yhteydessä. Standardissa määritellään käytettäväksi erilaisia liipaisimia joilla tilaaja voi muokata millä muutoksella viesti pitäisi lähettää. Standardissa on myös määritetty mekanismit, jolla tilaaja voi pyytää kaikki arvot kerralla tai tilata jaksottaisia viestejä tietyn aikavälein.

Standardissa määritetään luokka, jonka tehtävä on hoitaa tilausta ja sen asetuksia. Tässä kappaleessa käydään läpi luokan yleistä toiminnallisuutta, kappaleessa 2.1.7 käsitellään luokan attribuutteja ja toimintaa syvällisemmin. Niinkuin muutkin luokat standardissa, tästä tehdään instanssi, sille annetaan yksilöivä nimi ja se lisätään IED-laitteen hierarkiaan. Nämä määritellään IED-laitteen asetustiedostossa, kuten kaikki muutkin instanssit. Yksilöivän nimen avulla tilaaja voi viitata kutsulla instanssiin, muuttaa luokan asetuksia ja aloittaa tilauksen. Nämä luokat standardissa ovat puskuroitu viestintäluokka (engl. Buffered Report Control Block, lyhennetään BRCB) ja ei puskuroitu luokka (engl. Unbuffered Report Control Block, lyhennetään URCB). Tekstissä kumpaakin luokkaan viitatessa

käytetään lyhennettä RCB. Ainoa ero luokkien toiminnan välillä on, että BRCB puskuroid viestejä jonkin aikaa yhteyden katkettua. Yhteyden palautuessa, se lähettää puskuroidut viestit järjestyksessä asiakkaalle. BRCB takaa viestien järjestyksen ja saatavuuden. URCB lähettää viestejä asiakkaalle ilman puskurointia ja yhteyden katketessa, viestit menetetään. Standardissa määritetään, että yksi RCB-instanssi voi palvella vain yhtä tilaajaa kerrallaan. Eli IED-laitteeseen täytyy määrittää instansseja sen tilaajien määrän mukaan.

Kuvassa 5 on esitetty tilaajan ja IED-laitteen välinen viestien tilauksen prosessi. Kuvassa ensin asiakas tilaa puskuroidun BRCB-instanssin. Ensimmäisessä kutsussa tilaaja kirjoittaa RCB-luokan arvot, kuten käytettävät liipaisimet jne. Kutsussa tilaajan on merkittävä RCB-instanssin varauksi, jotta tilaus käynnistyy. IED-laite aloittaa viestien julkaisun tilaajalle määritettyjen ehtojen mukaan. Jos tilaaja ja IED-laitteen välinen yhteys katkeaa, BRCB-instanssi puskuroid viestejä johonkin järkevään rajaan asti. Kun yhteys tilaajan palaa, IED lähettää viestit järjestyksessä tilaajalle alkaen ensin puskurista. Tilaaja voi lopettaa tilauksen ja instanssin varauksen merkitsemällä sen taas vapaaksi.



Kuva 5. Puskuroidu viestien tilausprosessi tilaajan ja IED-laitteen välillä.

Standardissa määritetään, että viestejä voidaan tilata vain datajoukoista. IED-laitteen asetustiedostossa täytyy myös määrittää mitä datajoukkoa RCB-instanssi käyttää. Tämän jälkeen instanssi tarkkailee datajoukon attribuuttien muutoksia ja lähettää viestin, jos tilaajan asettama liipaisin täsmää. Koska yksi RCB voi palvella vai yhtä tilaajaa kerrallaan, täytyy samaan datajoukkoon viitata monella eri RCB-instanssilla. Näin monta eri tulevat tilaajaa saamaan viestin samasta tapahtumasta.

Standardissa on määritetty seuraavat liipaisimet data-attribuuteille, joita RCB tarkkailee ja reagoi niihin:

- datan muutos (engl. data change, standardissa lyhenne dchg),
- laadun muutos (engl. quality change, standardissa lyhenne qchg), ja
- datan päivitys (engl. data update, standardissa lyhenne dupd).

Jokaiselle data-attribuutille määritellään erikseen mitä liipaisimia se tukee. Nämä määritellään standardin luokkien määrittelyissä. Esimerkkinä aikaisemmin mainittu DPC-luokan määrittely taulukossa 3, jossa TrgOp-sarake kertoo attribuutin liipaisimen. Data muutos ja päivitys liipaisimen ero on, että datan päivitys liipaisee tapahtuman, vaikka attribuutin uusi arvo olisi sama. Datan muutos ei liipaise tapahtumaa, jos uusi arvo on sama kuin edellinen arvo. Laadun muutos liipaisin tarkoittaa, että data attribuuttiin liitetty laatuarvo muuttui. Laatuarvo kertoo tilaajalle ja arvojen lukijalle, voiko attribuutien arvoihin luottaa. Laatuarvo on tyyppiä Quality ja tästä voi tarvittaessa lukea enemmän standardista. [7, s. 90]

2.1.7 Raportointi-luokan määrittely ja toiminta

BRCB-luokalla on erilaisia attribuutteja, joita tilaaja voi kirjoittaa ja lukea ennen tilauksen aloittamista. BRCB ja URCB -luokat eivät eroa paljon attribuuteilla toisistaan, joten tässä kappaleessa keskitytään vain BRCB-luokan toimintaan. Tarkka määrittely luokkien eroista löytyy standardin osasta 7-2. Taulukossa 6 on esitetty standardin määrittämän BRCB-luokan attribuutit, attribuutin nimi englanniksi ja sen selite. Taulukossa ei ole esitetty attribuuttien tyyppejä, koska ne voi lukija tarvittaessa tarkemmin lukea standardin omasta määrittelyksestä. Lisäksi tässä kappaleessa käydään läpi luokan attribuuttien toiminta pääpiirteittäin ja loput tiedot lukija voi tarkistaa standardista. [8, s. 93–118].

Tilaaja voi vapaasti RCB-instanssin arvoja kirjoittaa ja lukea ennen tilauksen aloittamista monella peräkkäisellä kutsulla. Tärkein attribuutti luokassa on RptEna, joka on boolean tyyppiä. Kun attribuutti kirjoitetaan arvoon tosi, aloittaa instanssi tilauksen ja varaa sen tilaajalle. Tilauksen olleassa päällä, tilaaja voi edelleen lukea ja kirjoittaa sen arvoja, mutta rajoitetusti. Joidenkin arvojen kirjoitus pitää tapahtua ennen tilausta tai samassa kutsussa kun RtpEna laitetaan arvoon tosi. Tilaaja lopettaa tilauksen, jos yhteys on poikki tarpeeksi kauan tai RptEna kirjoitetaan arvoon epätosi.

RCB-luokan TrgOps-attribuutti on binääritietue, jossa yksittäinen bitti ilmaisee mikä liipaisin aiheuttaa viestin lähettämisen. Tällä attribuutilla tilaaja voi päättää mitä liipaisimia haluaa käyttää. TrgOps sisältää seuraavat liipaisimet:

- datan muutos (engl. data change, standardissa lyhenne dchg),
- laadun muutos (engl. quality change, standardissa lyhenne qchg), ja
- datan päivitys (engl. data update, standardissa lyhenne dupd),
- yleinen kysely (enlg. general-interrogation, standardissa lyhenne GI), ja
- jatkuva viestintä väliajoin (engl. intergrity).

Taulukko 6. BRCB-luokan määritetyt attribuutit ja niiden selitteet.

Attribuutti	Englanniksi	Selite
BRCBName	BRCB name	Objektin nimi
BRCBRef	BRCB reference	Objektin viite
RptID	Report identifier	RCB-instanssin yksilöivä id lähetettyihin viesteihin, asiakas voi asettaa
RptEna	Report enable	Varaa RCB:n ja aloittaa viestien lähetyksen
DatSet	Data set reference	Tarkailtavan datajoukon viite
ConfRev	Configuration revision	Juokseva konfiguraation numerointi, muutos kasvattaa numerointia
OptFlds	Optional fields	Mitä optionaalisia kenttiä viestiin lisätään
BufTm	Buffer time	Puskurointi-aika, ennen viestin lähetystä. Tänä aikana tapahtuvat liipaisut yhdistetään samaan viestiin
SqNum	Sequence number	Juokseva lähetetyn viestin numerointi
TrgOps	Trigger options	Millä liipaisimilla viesti lähetetään
IntgPd	Integrity period	Periodisen viestien väli millisekunteina, arvolla 0 ei käytössä
GI	General-interrogation	Käynnistää yleiskyselyn, joka sisältää kaikki datajoukon attribuutit seuraavaan viestiin
PurgeBuf	Purge buffer	Puhdistaa lähettämättömät viestit puskurista
EntryID	Entry identifier	Puskurissa olevan viimeisimmän viestin id. Arvo 0 tarkoittaa tyhjää puskuria
TimeOfEntry	Time of entry	Puskurissa olevan viimeisimmän viestin aikaleima
ResvTms	Reservation time	Instanssin varausaika sekunteina kun yhteys katkeaa, arvo -1 tarkoittaa konfiguraation aikaista varausta ja 0 että ei varausta
Owner	Owner	Yksilöi varaavan asiakkaan, yleensä IP-osoite tai IED-laitteen nimi. Arvo 0 että RCB on vapaa tai ei omistajaa

Kolme ensimmäistä liipaisinta dchg, qchg ja dupd ovat aikaisemmin kappaleessa 2.1.6 määritettyjen data attribuuttien liipaisimia. Asiakas voi tilata viestejä esimerkiksi vain datan muutoksista ja ei muista. RCB-luokka määrittää data attribuuttien liipaisimien lisäksi vielä kaksi liipaisinta lisää, yleinen kysely ja jatkuva viestintä väliajoin. Yleinen kysely on viesti, johon RCB sisällyttää kaikki datajoukon attribuutit. Ja jonka asiakas voi liipaista asettamalla luokan attribuutin GI arvoksi tosi ja TrgOps attribuutissa liipaisin on päällä. Tällöin RCB käynnistää viestin generoinnin ja lähettää sen asiakkaalle. Jos liipaisin ei ole päällä TrgOps attribuutissa, ja GI arvoksi asetetaan tosi. RCB ei generoi viestiä. Viestin lähetyksen jälkeen RCB itse asettaa GI:n arvoksi epätosi. Jatkuva viestintä liipaisin on jatkuvaa viestin lähettämistä tilaajalle väliajoin, johon sisältyy kaikki datajoukon attribuutit, kuten yleisessä kyselyssä. Toiminnon saa päälle kun asiakas asettaa RCB-luokassa attribuutit IntgPd arvoksi muu kuin 0, ja TrgOps-attribuutin arvossa kyseinen liipaisin on päällä. Attribuutti IntgPd kertoo minkä väliajoin viesti generoidaan ja lähetetään asiakkaalle. Jos IntgPd arvo on muu kuin 0 ja TrgOps attribuutissa liipaisin ei ole päällä, ei viestiä generoida ja lähetetä asiakkaalle väliajoin.

RCB-luokan attribuuttin OptFlds avulla asiakas voi asettaa mitä vaihtoehtoisia kenttiä viestiin sisällytetään. Attribuutin OptFlds on binääritietue, niin kuin ja TrgOps, ja taulukossa 7 on esitetty sen asetettavat arvot [8, s. 98]. Taulukon yksittäinen kenttä vastaa OptFlds arvon yhtä bittiä. Missä järjestyksessä bitit ovat, määräytyy standardin tekniikalle mallinnuksesta, kuten MMS-protokollalle. Taulukon arvoilla tilaaja voi määrittää mitä lisätietoa viestiin sisällytetään. Esimerkiksi asettamalla reason-for-inclusion bitin päälle, liitetään viestin arvon yhteyteen miksi tämä arvo viestiin sisällytettiin. Viestin rakennetta ja kuinka OptFlds-attribuutin arvoilla sen sisältöön voi vaikuttaa käydään läpi tarkemmin kappaleessa 2.1.8.

Taulukko 7. RCB-luokan OptFlds-attribuutin arvot ja niiden selitteet.

Arvo	Selite
sequence-number	Jos tosi, sisällytä RCB-luokan attribuutti SqNum viestiin
report-time-stamp	Jos tosi, sisällytä RCB-luokan attribuutti TimeOfEntry viestiin
reason-for-inclusion	Jos tosi, sisällytä syy miksi arvo(t) sisällytettiin viestiin
data-set-name	Jos tosi, sisällytä RCB-luokan attribuutti DataSet viestiin
data-reference	Jos tosi, sisällytä datajoukon liipaisseen kohdan rakentamiseen käytetty FCD- tai FCDA-viite viestiin
buffer-overflow	Jos tosi, sisällytä viestiin tieto onko puskuri vuotanut yli kentällä BufOvfl (engl. buffer overflow)
entryID	Jos tosi, sisällytä RCB-luokan attribuutti EntryID viestiin
conf-revision	Jos tosi, sisällytä RCB-luokan attribuutti ConfRev viestiin

Lähetetyt viestit voivat sisältää vaihtelevan määrän sisällytettyjä arvoja. RCB-instanssi mittaa aikaa ensimmäisestä liipaisusta sen attribuutin BufTm verran ja tämän ajan jälkeen pakkaa kaikki liipaisseet attribuutit samaan viestiin. Tilaaja voi muuttaa arvoa jos haluaa käyttää pitempää tai lyhyempää puskurointi-aikaa.

2.1.8 Viestin rakenne ja kuinka sen sisältö muodostuu

IED:n lähettämä viesti on rakenteeltaan hiukan monimutkainen ja lisäksi siihen vaikuttaa RCB-instanssin OptFlds-attribuutin asetetut bitit (taulukko 7). Tässä kappaleessa käsitellään viestin mallia, joka on tekniikasta riippumaton. Minkälainen viestin rakenne on MMS-protokollan tasolla, siitä ei tarvitse välittää. Toteutetussa ohjelmassa käytettiin kirjastoa, joka hoitaa matalan tason asiat ja tarjoaa helppokäyttöisen rajapinnan viestin sisältöön. Kuitenkin viestin rakenteesta täytyy ymmärtää kuinka vaihtoehtoiset kentät siihen vaikuttavat ja kuinka attribuuttien arvot viestiin sisällytetään. Kuvassa 6 on esitetty standardin määrittämän viestin rakenne ja mitä kenttiä OptFlds-attribuutti kontrolloi. Viestin rakenteen voisi ajatella koostuvan kahdesta osasta. Ensin viestissä on yleinen tieto ja viimeisenä taulukko datajoukon alkioista 1–n:ään, jotka liipaisivat viestin lähetyksen.

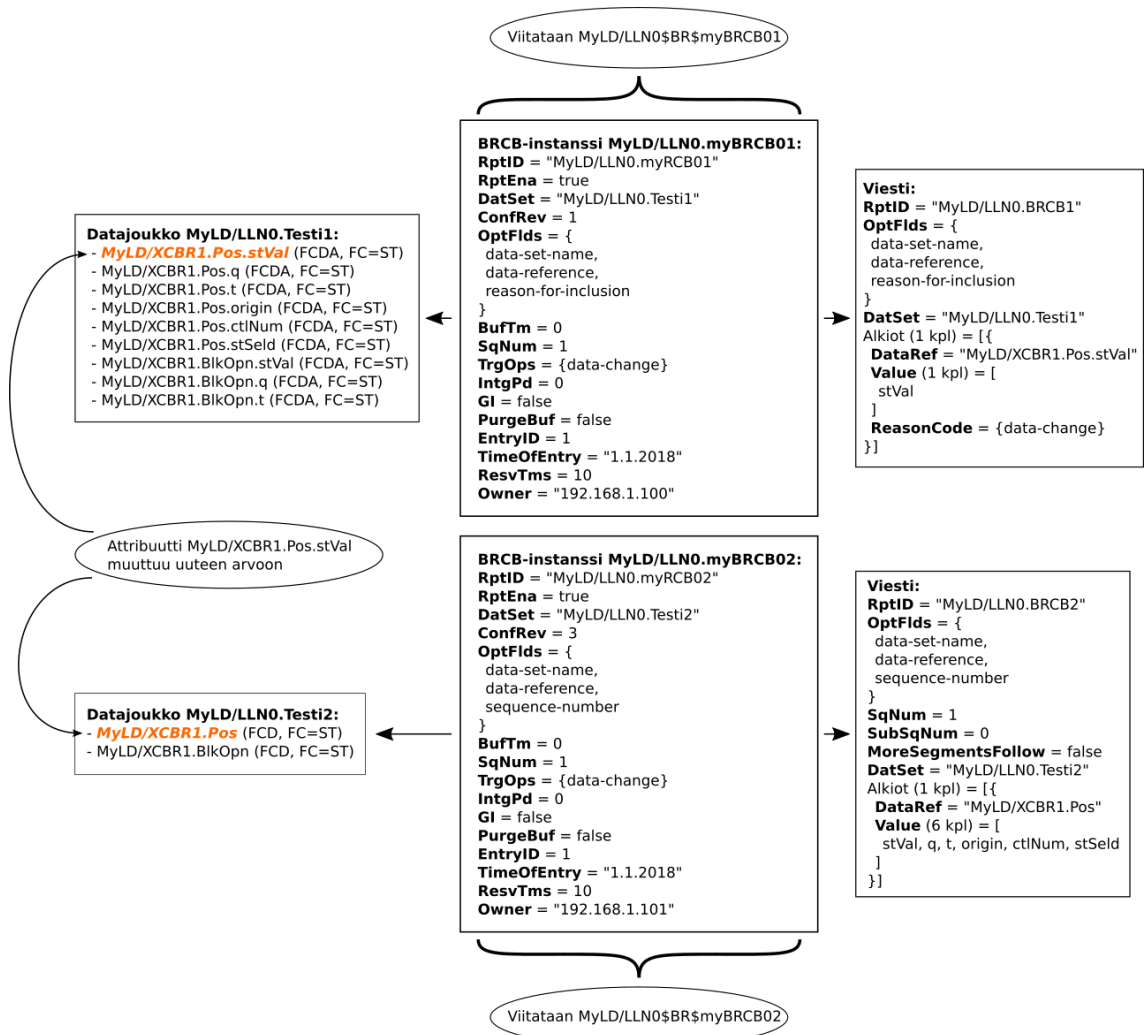
Kuvassa 7 on esitetty yleinen kuva kahden viestin lähetyksestä liipaisun tapahtuessa. Kuvassa keskellä on kaksi BRCB-instanssia myBRCB01 ja myBRCB02, jotka tarkkailevat datajoukkoja Testi1 ja Testi2 vastaavasti. Kummatkin instanssit lähettävät viestin, jotka ovat kuvassa oikealle. BRCB-instansseista voi nähdä, mitä niille asetetut attribuuttien

Viestin rakenteellinen sisältö		
Parametrin nimi	Englanniksi	Selitys
RptID	Report identifier	RCB-instanssin yksilöivä id.
OptFlds	Optional fields	Mitä optionaalisia kenttiä viestiin on sisällytetty
Jos sequence-number = tosi		
SqNum	Sequence number	Juokseva lähetetyn viestin numerointi
SubSqNum	Sub sequence number	Pilkotun viestin juokseva alinumerointi
MoreSegmentsFollow	More segments follow	Tosi jos samalla juoksevalla päänumerolla saapuu vielä lisää viestejä
Jos data-set-name = tosi		
DatSet	Data set	Tarkailtavan datajoukon viite
Jos buffer-overflow = tosi		
BufOvfl	Buffer overflow	Jos arvo on tosi, on viestien puskurit vuotaneet yli
Jos conf-revision = tosi		
ConfRev	Configure revision	Juokseva konfiguraation numerointi
Viestin data		
Jos report-time-stamp = tosi		
TimeOfEntry	Time of entry	Aikaleima milloin viesti generoitiin
Jos entryID = tosi		
EntryID	Entry id	Viestin yksilöivä numero
Liipaissut datajoukon alkio [1..n]		
Jos data-reference = tosi		
DataRef	Data reference	Liipaisseen datajoukon alkion FCD- tai FCDA -viite
Value	Value	Sisältää arvon tai arvot liipaisseesta datajoukon alkoista.
Jos reason-for-inclusion = tosi		
ReasonCode	Reason code	Syykoodi miksi tämä datajoukon kohta on sisällytetty viestiin

Kuva 6. Standardin määrittämä lähetetyn viestin rakenne (pohjautuu kuvaan [8, s. 104]).

arvot ovat ja datajoukoista näkee mistä FCD- ja FCDA -viitteistä ne koostuvat. Kuvassa attribuutin MyLD/XCBR1.Pos.stVal arvo muuttuu ja tämä liipaisee viestin lähetyksen kummassakin BRCB-instanssissa. Viesteistä voi nähdä sen sisällön ja myös miten BRCB-instanssien OptFlds-attribuutin arvot vaikuttavat sen sisältöön. Lähetettyjen viestin rakennetta ja sisältöä voi verrata kuvassa 6 määritetyn viestin rakenteeseen. Kuvassa on esitetty myös kuinka BRCB-instansseihin viitataan MMS-protokollan tapauksessa.

Tätä käsitellään tarkemmin kappaleessa 2.1.9.



Kuva 7. BRCB-instanssi tarkkailee sille määritettyä datajoukkoa ja generoi viestin tapahtuman liipaistessa.

Viestissä kenttä RptID sisältää viitteen RCB-instanssiin, mistä viestin on peräisin. OptFlds sisältää binääritietueen viestin vaihtoehtoisista kentistä. Tämä kenttä on suoraan verrattavissa viestin kenttiä SqNum, SubSqNum ja MoreSegmentsFollow käytetään kertomaan asiakkaalle, jos päätason viesti on liian pitkä ja se on pilkottu alaosiin. Kenttä SqNum on RCB-instanssin samanniminen kenttä ja on juokseva numerointi päätason viesteille. Kenttä SubSqNum on juokseva numerointi alkaen 0, jos päätason viesti, eli saman SqNum arvon sisältävä viesti on pilkottu osiin. Kentän MoreSegmentsFollow ollessa tosi asiakas tietää että päätason viesti on pilkottu osiin ja seuraava osa on odotettavissa palvelimelta. Kun viestin kaikki osat on lähetetty, palvelin asettaa viimeisessä viestissä kentän MoreSegmentsFollow arvoksi epätosi ja seuraavassa päätason viestissä SubSqNum kentän arvoksi 0. Kenttä DataSet sisältää viitteen datajoukkoon mistä viestin on peräisin. Puskuroidussa BRCB-instanssissa kenttä BufOvlf kertoo onko viestipuskuri vuotanut yli. ConfRev kertoo juoksevan konfiguraation numeron, tämä tulee suoraan RCB-instanssin samannimisestä attribuutista. TimeOfEntry kertoo milloin viesti generoitiin IED-laitteen päässä.

EntryID on viestin yksilöivä numerointi. Tämä kenttä tulee suoraan RCB-instanssin samannimisestä kentästä. Tämän jälkeen viestissä tulee taulukko, joka sisältää liipaisseet datajoukon alkioit. Jokainen taulukon alkio sisältää Value-kentän ja vaihtoehtoiset DataRef ja ReasonCode kentät. DataRef sisältää datajoukon FCD- tai FCDA-viitteen, joka liipaisi tapahtuman. ReasonCode kenttä kertoo mikä RCB-instanssin TrgOps-attribuutilla asetetuista liipaisimista liipaisi tapahtuman ja aiheutti alkion sisällytyksen viestiin. Kenttän mahdolliset arvot ovat samat kuin RCB-instanssin TrgOps-attribuutin arvot.

Tärkeä tieto Value-kentästä on ymmärtää, että se voi sisältää yhden tai monta data-attribuutin arvoa. Tämä riippuu viittaako datajoukon liippaissut alkion FCD- vai FCDA-viitteellä kuinka moneen data-attribuuttiin. Viittauksen ollessa FCDA-viite, joka viittaa vain yhteen data-attribuuttiin, sisältää Value-kenttä vain kyseisen data attribuutin arvon. Jos viittaus on FCD- tai FCDA-viite, joka viittaa moneen attribuuttiin hierarkiassa alaspäin. Sisältää Value-kenttä kaikki nämä viitattut arvot, vaikka niistä olisi liippaissut vain yksi attribuutti. FCD- ja FCDA-viittauksen toimintaa ja mitä attribuutteja se viittaa hierarkiassa alaspäin, käydään läpi kappaleessa 2.1.5. Esimerkki tästä on kuvassa 7, jossa liipaisu yhdes- sä attribuutissa aiheuttaa eri määrän arvoja kumpaankin viestiin. Tähän vaikuttaa kuinka liipaisevaan attribuuttiin on viitattu datajoukossa. Kuvassa datajoukossa Testi1 attribuuttiin MyLD/XCBR1.Pos.stVal on viitattu FCDA-viitteellä, jossa funktionaalinen rajoite on ST. Eli FCDA-viite viittaa vain stVal attribuuttiin, ei muihin. Tämän takia myBRCB01-instanssilta tuleva viestin Value-kenttä sisältää vain stVal-attribuutin arvon. Kun taas datajoukossa Testi2 attribuuttiin MyLD/XCBR1.Pos.stVal sisältyy datajoukon ensimmäiseen FCD-viitteeseen funktionaalisella rajoitteella ST. Koska FCD-viite viittaa kaikkiin Pos-instanssin alla oleviin attribuutteihin, joilla funktionaalinen rajoite on ST. Lisätään kaikki nämä attribuutit viestiin, joka lähetetään tilaajalle. BRCB-instansilta myBRCB02 tuleva viestin Value-kenttä sisältää kaikki viitattut attribuutit ja viestin DatRef-kenttä sisältää datajoukossa käytetyn viitteen. Dataobjektin Pos kaikki attribuutit voi tarkistaa taulukosta 3. [7, s. 40–44] [8, s. 108]

2.1.9 Abstraktimallin sovitus MMS-protokollaan

Tähän asti käsitelty IEC 61850 -standardin mallit ja palvelut ovat olleet abstrahoituja ja tekniikasta riippumattomia. Tässä työssä käytettiin IEC 61850 -standardin MMS-protokollan mallinusta (engl. Manufacturing Message Specification). Tästä mallinnuksesta on tarkemmin määritetty IEC 61850 -standardin osassa 8-1. MMS-protokolla on maailmanlaajuinen ISO 9506 -standardi viestintään, joka on määritetty toimivaksi TCP/IP:n pinnon päällä [18]. Tämän työn kannalta lukijan ei ole tarvitse ymmärtää MMS-protokollaa ja sen toimintaa. Suunnittelussa ohjelmistossa käytettiin apuna kirjastoa, joka hoitaa matlan tason kommunikoinnin IED-laitteen kanssa. Tässä osiossa käsitellään työn kannalta tärkeitä tietoja, mitä mallinnuksesta MMS-protokollalle kuitenkin tarvitsee tietää. [26]

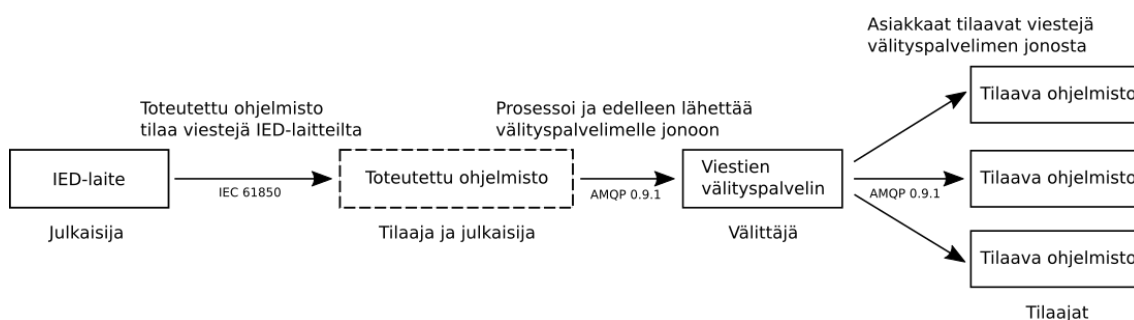
IEC 61850 -standardin mallinnuksessa aikaisemmin esitetty instanssien viittaus hierarkiassa muuttuu ja nyt viittaus sisältää myös funktionaalisen rajoitteen. Esimerkkinä ku-

vassa 4 oleva viite "OmaLD/Q0XCBR1.Pos.stVal"funktionaalisella rajoitteella ST, muuttuu muotoon "OmaLD/Q0XCBR1\$ST\$Pos\$stVal". Tässä viittauksessa pisteet (.) korvataan dollari-merkillä (\$). Ja kaksikirjaiminen funktionaalinen rajoite sijoitetaan loogisen noodin ja ensimmäisen data objektin nimien väliin. Muuten viittaus säilyy identtisenä alkuperäiseen ja samat rajoitteet ja nimeämiskäytännöt ovat voimassa edelleen. [11, s. 34–35, 111]

Tämän uuden viittauksen takia jokaiselle viitattavalle kohteelle täytyy olla funktionaalinen rajoite. Niinpä esimerkiksi RCB-luokkien instansseille täytyy olla myös funktionaalinen rajoite. Puskuroitua RCB-instanssia viitataan funktionaalisella rajoitteella BR. Ja puskuroimatonta funktionaalisella rajoitteella RP. Esimerkin tästä viittauksesta voi nähdä aikaisemmin mainitusta kuvasta 7. [11, s. 32–34, 75]

2.2 Advanced Message Queuing Protocol (AMQP)

Työssä toteutetussa ohjelmistossa IED-laitteelta verkon yli tilatut viestit ohjelma prosessoi ja lähetti viestin eteenpäin välittäjälle (engl. message broker) jonoon. Välittäjä on verkossa oleva erillinen palvelin, mistä muut ohjelmat pystyivät tilaamaan viestejä tarpeidensa mukaan. Kuvassa 8 on esitetty lopullisen toteutuksen tietoliikenne eri osapuolten välillä. Tässä työssä toteutettu ohjelmisto on merkitty kuvaan katkoviivalla. Toteutuksessa oli kyse julkaisu ja tilaus -arkkitehtuurimallista (engl. publish-subscribe pattern), jossa työn toteutettu ohjelmisto oli tilaaja yhdeltä IED-laitteelta ja julkaisija välityspalvelimelle. Ja välityspalvelimen toisessa päässä olevat ohjelmistot olivat tilaajia. Tässä teoriaosuudessa perehdytään viestien välittäjän teoriaan, ja mitä siitä täytyy tietää ohjelmistokehityksen kannalta.



Kuva 8. Toteutetun ohjelmiston osuus ja rooli käytettävässä kokonaisuudessa tietoliikenteen kannalta.

Työssä välittäjänä käytettiin RabbitMQ-ohjelmistoa¹, joka on avoimen lähdekoodin välittäjäpalvelin ja perustuu avoimeen AMQP-standardiin² (engl. Advanced Message Queuing Protocol). AMQP määrittää yhteisen protokollan viestintään eri ohjelmistojen välillä verkon yli välityspalvelimen avulla. Verkon ansiosta välityspalvelin voi sijaita eri koneella

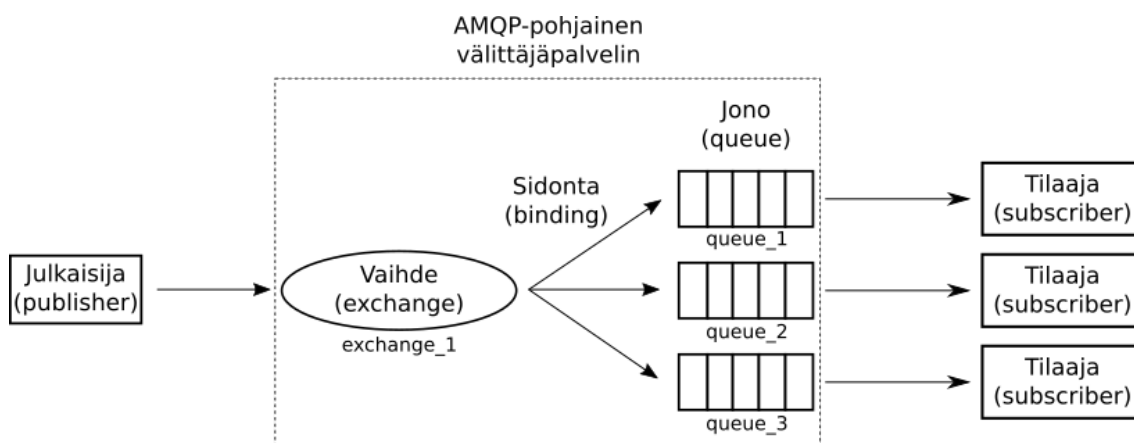
¹<https://www.rabbitmq.com/>

²<https://www.amqp.org/>

kuin sitä käyttävät ohjelmistot. Ajan saatossa standarista on julkaistu monta eri versiota, ja työn tehokkella viimeisin versio oli 1.0. Kuitenkin RabbitMQ-ohjelmisto oli suunniteltu käytettäväksi suoraan standardin version 0.9.1 kanssa, ilman asennettuja lisäosia. Versioiden välinen ero oli suuri ja siirto suoraan uuteen ei olisi mahdollista, koska standardin versiot eivät olleet keskenään yhteensopivat. RabbitMQ tuki versiota 0.9.1 ja sen kehittäjät mieltävät standardin version 1.0 kokonaan eri protokollaksi [23]. Kuvassa 8 on tietoliikenteen kohtiin merkitty mikä standardi vaikuttaa minkäkin osapuolen kommunikointiin. Tässä työssä välityspalvelin ja siihen yhteydessä olevat ohjelmistot käyttävät AMQP-standardista versiota 0.9.1.

2.2.1 Advanced Message Queuing -malli ja sen osat

AMQP-standardi määrittää komponentteja, joiden läpi viestin täytyy kulkea julkaisijalta tilaajalle. Standardissa nämä komponentit määrittää AMQ-malli (engl. AMQ-model). Kuvassa 9 on esitetty viestin kulku julkaisijalta tilaajalle mallin eri komponenttien läpi. Mallin komponentit ovat *vaihde* (engl. *exchange*), *jono* (engl. *queue*) ja näiden välinen *sidonta* (engl. *binding*). Välityspalvelimen tehtävän voi tiivistää niin, että se ottaa vastaan viestejä julkaisijoilta vaihteeseen. Vaihde reittittää viestejä tilaajille jonoihin jonon ja vaihteen välisten sidosten mukaan. Jos tilaaja ei kerkeä prosessoida viestejä tarpeeksi nopeasti, palvelin pitää viestit jonossa tilaajalle. Vaihde voi välittää viestin moneen eri jonoon ja yhtä jonoa voi tilata monta eri asiakasta.



Kuva 9. AMQ-mallin osat ja viestin kulku niiden läpi julkaisijalta tilaajalle (pohjautuu kuvaan [2, s. 11]).

AMQP on ohjelmoitava protokolla siinä mielessä, että julkaisija ja tilaaja voivat määrittää komponentteja ja reitityksiä palvelimelle verkon yli ajon aikana tarpeidensa mukaan. Välittäjäpalvelin ei määritä kuin oletus vaihteet valmiiksi käytettäväksi. Eli julkaisuja voi luoda vaihteita ja tilaaja voi luoda jonoja ja sidoksia vaihteiden ja jonojen välille. Voidaan sanoa että julkaisija ja tilaaja tekevät uusia instansseja AMQ-mallin komponenteista palvelimelle. Vaihteiden ja jonojen instansseilla täytyy olla välityspalvelimella yksilölliset nimet, jokainen nimi asetetaan instanssin luonnin yhteydessä. Esimerkkinä kuvassa

9 on AMQ-mallin komponenttien alla niille määritetyt nimet. Vaihteella on esimerkiksi nimi `exchange_1` ja ylimmällä jonolla `queue_1`. Tällä ohjelmoitavalla ominaisuudella välityspalvelin voidaan konfiguroida toteuttamaan erilaisia skenaarioita vapaasti ja se antaa kehittäjille vapautta toteutukseen.

2.2.2 Vaihde (exchange) ja reititysavain (routing-key)

Jotta viesti voidaan välittäjäpalvelimen läpi kuljettaa, täytyy julkaisijan aloittaa määrittämällä sen käyttämä vaihde (engl. exchange) ja sen tyyppi, tai käyttää palvelimen oletusvaihdetta. Vaihde on komponentti, joka ottaa vastaan viestejä ja reitittää niitä jonoihin vaihdetyypin (engl. exchange type) ja sidosten mukaan. Vaihteet eivät ikinä tallenna viestejä. Vaihde voi tiputtaa viestin, jos se ei täsmää minkään määritetyn reitityksen kanssa. AMQ-malli määrittää seuraavat käytettävät vaihdetyypit:

- suoravaihde (engl. direct exchange),
- hajautusvaihde (engl. fanout exchange),
- aihepiirivaihde (engl. topic exchange) ja
- otsikkovaihde (engl. header exchange).

Näitä tyyppejä ja kuinka ne toimivat, käydään tarkemmin läpi tulevissa kappaleissa. Tyyppin lisäksi vaihteella on myös attribuutteina nimi (engl. name), kestävyys (engl. durability), automaattinen poisto (engl. auto-delete). Nimi yksilöi vaihteen palvelimella ja tilaaja käyttää tätä nimeä sidoksen tekemiseen jonon ja vaihteen välille. AMPQ-standardissa oletetaan, että nimi on jo tiedossa etukäteen julkaisijalla ja tilaajalla. AMPQ ei tarjoa toiminnallisuutta instanssien nimien noutamiseen. Kestävyys parametrilla julkaisija voi kertoa palvelimelle, että välittäjä säilyttää vaihteen uudelleenkäynnistysten jälkeen. Jos ei, julkaisijan täytyy määrittää vaihde uudelleen käynnistytyn jälkeen. Automaattinen poisto kertoo poistaako välittäjä vaihteen automaattisesti, kun viimeinen siihen sidottu jono on poistettu ja julkaisija ei ole enää yhteydessä.

Kaikki julkaisijan ja tilaajan kutsut välittäjäpalvelimelle, jotka tekevät uuden instanssin komponentista, ovat esitteleviä (engl. declare). Tarkoittaa että palvelin tekee tarvittaessa uuden instanssin komponentista, jos sitä ei ole jo olemassa, ja vastaa samalla tavoin onnistuneesti molemmissa tapauksissa. Tilanne tulee esimerkiksi silloin kun kaksi julkaisijaa käyttävät samaa vaihdetta keskenään. Toinen ei tiedä onko toinen jo määrittänyt instanssin vaihteesta palvelimelle, esimerkiksi silloin kun ohjelmat käynnistyvät eri aikaan. Jos kummatkin julkaisijat eksplisiittisesti määrittävät saman käytettävän vaihteen. Palvelin vastaa kummallekin onnistuneesti ja tuloksena palvelimella on vain yksi instanssi halutusta vaihteesta. Sama toiminta pätee kaikkiin välittäjäpalvelimen kutsuihin, jotka tekevät uusia instansseja komponenteista.

Vaihde reitittää viestejä jonoihin sen sidosten ja tyyppin mukaan. Kuitenkin reititykseen liittyy yksi tärkeä asia kuin reititysavain (engl. routing-key). Reititysavain on kuin virtuaalinen osoite viestissä, jonka julkaisija liittää viestiin julkaisun yhteydessä. Tilaja käyttää

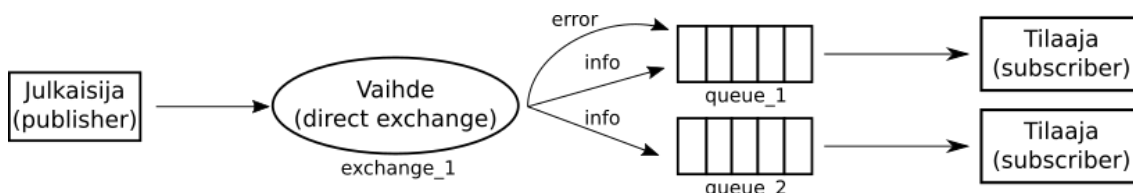
myös reititysavainta jonon määrittelyn yhteydessä. Vaihte, tyypistä riippuen, voi käyttää tätä avainta reititykseen eri jonoihin. Viestin reititysavainta voi hyvin verrata lähetettävän sähköpostin saaja-kenttään. Saaja kertoo vastaanottajan sähköpostiosoitteen, johon viesti on tarkoitus lähettää. Reititysavain toimii juurikin näin suorassa viestin lähetyksessä, mutta eroaa muissa.

2.2.3 Suoravaihte (direct exchange)

Julkaisija voi määrittää vaihteen instanssin tyypiksi suoravaihteen (engl. direct exchange). Suoravaihte reitittää viestin jonoihin suoraan vastaavan reititysavaimen perusteella. Suoravaihte reitittää seuraavasti:

- tilaaja määrittää sidoksen reititysavaimella K,
- julkaisija julkaisee viestin reititysavaimella R,
- vaihte välittää viestin jonoon jos $K = R$,
- muuten vaihte tiputtaa tai palauttaa viestin lähettäjälle.

Kuvassa 10 on esitetty suoravaihteen toiminta. Vaihteeseen on tehty sidoksia reititysavaimilla *error* ja *info*. Yksi tilaaja voi luoda sidoksia samaan vaihteeseen monella eri reititysavaimella. Näin tilaaja voi tilata viestejä mistä on kiinnostunut. Kuvassa 10 julkaisija julkaisee viestin reititysavaimella *info*. Viesti päättyy molempiin queue_1 ja queue_2 jonoon. Reititysavaimella *error*, viestit päättyvät vain jonoon queue_1. Välittäjäpalvelin tarjoaa suoravaihteesta oleutusvaihteen nimeltä *amq.direct*. [2, s. 27]



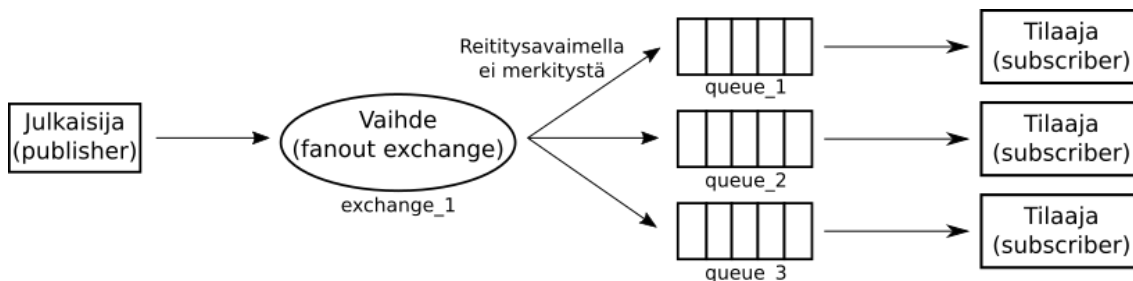
Kuva 10. Suoravaihte (engl. direct exchange), reitittää suoraan sidoksen reititysavaimen mukaan (pohjautuu kuvaan [24]).

2.2.4 Hajautusvaihte (fanout exchange)

Julkaisija voi määrittää vaihteen instanssiksi hajautusvaihteen (engl. fanout exchange). Hajautusvaihte reitittää viestit kaikkiin sen jonoihin reititysavaimesta välittämättä. Hajautusvaihte toimii seuraavasti:

- tilaaja määrittää sidoksen vaihteeseen reititysavaimella K,
- julkaisija julkaisee viestin reititysavaimella R,
- vaihte välittää viestin kaikkiin siihen sidottuihin jonoihin, reititysavaimesta riippumatta.

Kuvassa 11 on esitetty hajautusvaihteen toiminta. Vaihteeseen exchange_1 on tehty kolme eri sidosta jonoihin queue_1, queue_2 ja queue_3. Julkaisijan lähettämä viesti lähetetään kaikkiin kolmeen sidottuun jonoon, viestin ja jonojen reititysavaimista riippumatta. Välittäjäpalvelin tarjoaa hajautusvaihteesta oletusvaihteen nimeltä amq.fanout. [2, s. 27]



Kuva 11. Hajautusvaihde (engl. fanout exchange), reitittää kaikkiin siihen sidottuihin jonoihin riippumatta reititysavaimesta (pohjautuu kuvaan [1]).

2.2.5 Aihepiirivaihde (topic exchange)

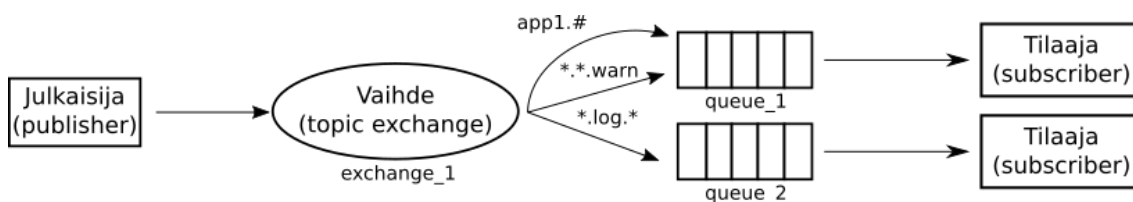
Aihepiiri vaihdetyyppi (engl. topic exchange) reitittää viestejä sidottuihin jonoihin reititysavaimen mukaan, kuten suoravaihde, mutta tarjoaa lisäksi sääntöjä monen avaimen samanaikaiseen yhteensopivuuteen. Sidoksen reititysavaimen sijaan voidaan puhua reitityskaavasta (engl. routing pattern). Aihepiiri vaihde toimii seuraavasti:

- tilaaja määrittää sidoksen vaihteeseen reitityskaavalla P,
- julkaisija julkaisee viestin reititysavaimella R,
- vaihde välittää viestin jonoon, jos sen reitityskaava P sopii reititysavaimeen R.

Aihepiirivaihteen yhteydessä AMQP-standardi määrittää että viestin reititysavain täytyy olla lista sanoja, jotka ovat erotettu pisteillä ja maksimissaan 255 merkkiä pitkä [2, s. 35]. Sanat saavat sisältää kirjaimia A-Z ja a-z, ja numeroita 0-9. Yleensä avaimeen sijoitetaan sanoja mitkä liittyvät viestin sisältöön. Tilaajan määrittämä sidoksen reitityskaava voi olla samaa muotoa kuin reititysavain, mutta sanojen tilalla voidaan käyttää seuraavia erikoismerkkejä:

- * (tähti), voi vastata mitä tahansa yhtä sanaa,
- # (risuaita), voi vastata nolla tai monta sanaa. [2, s. 27]

Kuvassa 12 on esitetty aihepiirivaihteen toiminta. Vaihteeseen exchange_1 on sidottu jono queue_1 reitityskaavoilla **app1.#** ja ***.*.warn**. Ja jono queue_2 reitityskaavalla ***.log.***. Oletetaan että julkaisija lähettää viestejä avaimella muodossa *ohjelma.kanava.taso*, jossa sana ohjelma kuvaa julkaisijan nimeä. Kanava, kuvaa lokitusväylää ja taso kuvaa viestin tasoa (warning, error, info jne.). Voisi sanoa että queue_1 on kiinnostunut kaikista ohjelmalta app1 tulevista viesteistä ja myös kaikista varoitustason (warning) viesteistä kaikilta ohjelmilta. Jono queue_2 on taas kiinnostunut kaikista log-väylän viesteistä.



Kuva 12. Aihepiirivaihde (engl. topic exchange), reitittää kaikkiin siihen sidottuihin jonoihin, joiden reitityskaava sopii viestin reititysavaimeen (pohjautuu kuvaan [25]).

Nyt jos julkaisija lähettää viestin avaimella **app1.debug.warn**. Vaihte välittää viestin jonoon queue_1, mutta ei jonoon queue_2. Avaimella **app2.log.info** viesti välitetään vain jonoon queue_2. Avaimella **app1.log.warn** viesti lähetetään molempiin jonoihin. Kun taas avaimella **app2.debug.info** viestiä ei lähetetä yhteenkään jonoon.

Aihepiirivaihde on vaihdetyypeistä monimutkaisin, mutta kattaa ison määrän erilaisia käyttötapauksia. Vaihteen avulla tilaajat voivat tilata viestejä, joista ovat esimerkiksi kiinnostuneita. Aihepiirivaihdetta voi käyttää kuin aikaisempia vaihdetyyppejä. Jos jono sidotaan reitityskaavalla #, se vastaanottaa kaikki viestit kyseiseltä vaihteelta ja käyttäytyy kuin hajautusvaihte. Jos jono sidotaan ilman merkkejä * ja #, niin se käyttäytyy samalla tavalla kuin suoravaihte. [25]

2.2.6 Otsikkovaihde (headers exchange)

Otsikkovaihde (engl. headers exchange) on vaihdetyyppi joka ei käytä reititysavainta ol- lenkaan reititykseen, vaan reititys perustuu viestin ja sidoksen otsikkotietoihin. Otsikko- tiedot koostuvat avain–arvo-pareista. Otsikkovaihde toimii seuraavasti:

- tilaaja määrittää sidoksen vaihteeseen otsikkotiedoilla H,
- julkaisija julkaisee viestin otsikkotiedoilla O,
- vaihte välittää viestin jonoon jos otsikkotiedot O vastaavat otsikkotietoja H, riippuen sidoksen otsikkotiedoissa olevasta **x-match** kentän arvosta.

Jonon sidoksen määrittämisen yhteydessä tilaaja voi asettaa kentän **x-match** otsikkotietoi- hin ja sille arvon kahdesta eri mahdollisuudesta **all** tai **any**. Arvot toimivat seuraavasti:

- **all** kertoo vaihteelle, että jokainen viestin otsikkotieto täytyy vastata sidoksen otsik- kotietoja (boolean algebrassa AND-operaatio), jotta viestin lähetetään jonoon,
- **any** kertoo vaihteelle, että mikä vain viestin otsikkotiedoista löytyy sidoksen otsik- kotiedoista (boolean algebrassa OR-operaatio), lähetetään viesti jonoon. [2, s. 28]

Otsikkotiedoissa arvot ovat vaihtoehtoisia asettaa. Jos kentän arvoa ei ole asetettu, vas- taavuus on kun kentän nimet ovat samat. Jos kentän arvo on asetettu, vastaavuus on jos molemmat nimi ja arvo vastaavat toisiaan. [2, s. 28]

2.2.7 Jonon määrittäminen ja viestien kuittaaminen

AMQ-mallissa jono (engl. queue) on vaihteen ja tilaajan välissä oleva puskuri (kuva 9), joka tallentaa tilaajalle tulevia viestejä. Jono pitää viestejä jonossa tilaajalle, kunnes tämä kerkiää prosessoida ne. Yksi jono voi puskuroida viestejä monelle eri tilaajalle. Tilaaja sitoo (engl. binding) jonon nimellä johonkin palvelimelle jo olevaan vaihteeseen mistä viestejä haluaa. Tilaajan täytyy tietää vaihteen nimi jo etukäteen. Jonolla tilaaja voi määrittää attribuutteja. Jotkin attribuutit ovat samoja kuin vaihteella. Tilaaja voi määrittää jonolle nimen (engl. name), kestävä (engl. durable), eksklusiivinen (engl. exclusive) ja automaattinen poisto (auto-delete). Nimi yksilöi jonon palvelimella. Tilaaja voi halutessaan pyytää palvelinta generoimaan yksilöivän nimen jonolle automaattisesti. Kestävyys säilyttää jonon palvelimella uudelleenkäynnistyksen jälkeen. Eksklusiivinen rajoittaa jonon vain yhdelle tilaajalle, ja palvelin poistaa jonon kun yhteys tilaajaan katkeaa. Automaattinen poisto poistaa jonon palvelimelta automaattisesti, kunnes yhteys viimeiseen tilaajaan on katkennut. [1]

Jono lähettää viestin vain yhdelle jonossa olevalle tilaajalle. Sama viesti lähetetään ainoastaan toiselle tilaajalle, jos se edelleenlähetetään virheen tai peruutuksen seurauksena. Jos samassa jonossa on monta eri tilaajaa, jono lähettää viestejä monelle tilaajalle kierto- vuorottelun (engl. round-robin) periaatteen mukaan. [2, s. 11–12]

Tilaajan täytyy määrittää jonolle sen käyttämä viestin kuittaamisen (engl. acknowledge) malli, ennen kuin jono poistaa viestin puskurista. Malleja on kaksi:

- automaattinen, jolloin palvelin poistaa viestin jonosta heti kun se on lähetetty tilaajalle,
- eksplisiittinen, jolloin palvelin poistaa viestin vasta kun tilaaja on lähettänyt kuittauksen palvelimelle.

Tilaaja voi lähettää viestistä kuittauksen milloin vain prosessoinnin aikana. Heti kun viesti on vastaanotettu, tai silloin kun viesti on prosessoitu. [2, s. 29]

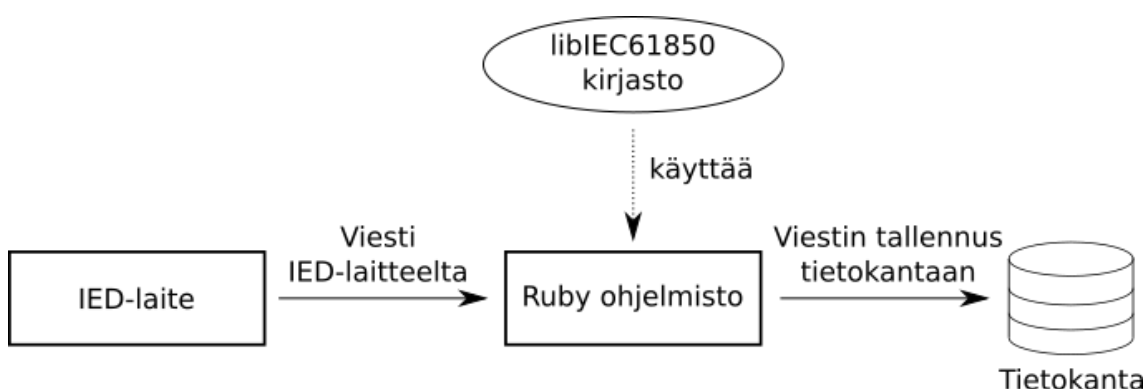
3. PROJEKTIN LÄHTÖKOHDAT

Ennen tämän työn aloittamista yrityksessä oli jo kehitetty ensimmäinen versio ohjelmasta, joka kykeni tilaamaan viestejä IED-laitteelta. Prosessoimaan viestit ja tallentamaan ne relaatiotietokantaan myöhempää käyttöä varten. Tässä ohjelmistossa oli havaittuja ongelmia ja se ei myöskään tukenut kaikkia IEC 61850 -standardin viesteihin liittyviä ominaisuuksia. Tämän ohjelmiston toimintaperiaate ja siinä olleet ongelmat toimivat pohjana uuden version suunnittelulle ja toteutukselle. Tarkoituksena oli poistaa havaitut ongelmatkohdat ja miettiä olisiko jokin muu arkkitehtuuri parempi kyseiseen toteutukseen. Ensimmäistä toteutusta ohjelmasta voisi nimittää ensimmäiseksi protoversioksi tai demovaiheeksi (engl. proof of concept), jonka pohjalta tultiin tekemään toimiva lopullinen versio. Tekstissä eteenpäin sanalla demoversio viitataan tähän ohjelmistoon.

Tässä osiossa pohjustetaan työn alkua lukijalle ja mistä lähdettiin liikkeelle. Mitä ongelmia demovaiheen toteutuksessa oli ja niiden analyysi. Demovaiheen ohjelmasta käsitellään sen arkkitehtuuria, mitkä olivat sen komponentit ja niiden toiminnallisuus. Tässä käsitellyt ongelmat toimivat pohjana uuden version suunnittelulle ja auttavat tekemään siihen liittyviä ratkaisuja.

3.1 Demoversio ja sen toiminta

Demoversio oli ohjelmoitu Ruby-ohjelmointikielellä. Ohjelman arkkitehtuuri oli todella yksinkertainen. Kuvassa 13 on esitetty demoversion arkkitehtuuri korkealla tasolla.

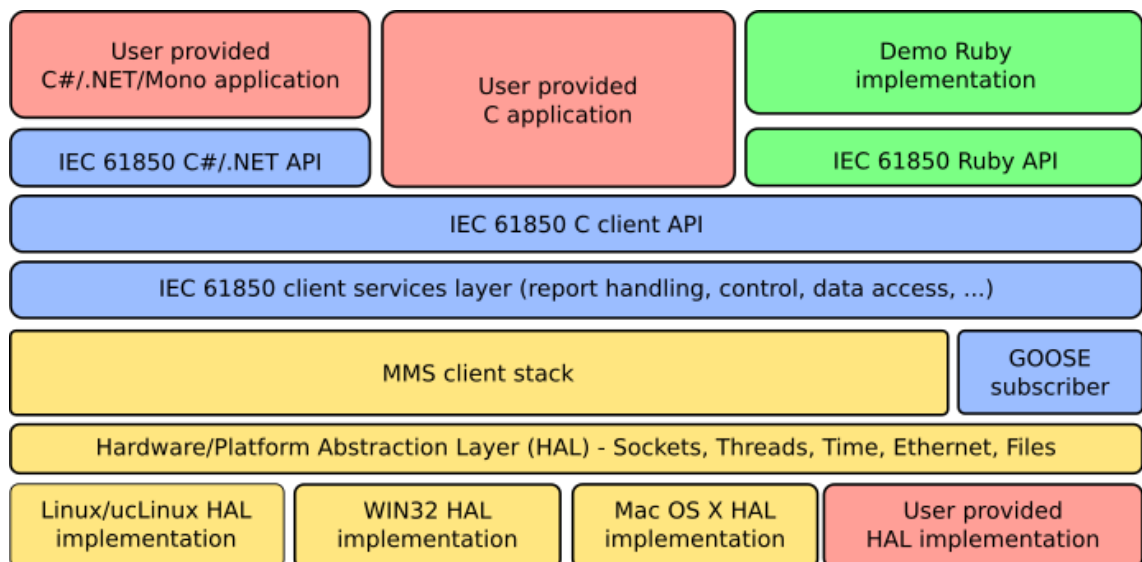


Kuva 13. Rubylla toteutetun demoversion arkkitehtuuri ja tiedonsiirto.

Yksi ajettu demoversion prosessi pystyi tilaamaan yhden IED-laitteen kaikki RCB-luokkien instanssit. Tiedon instanssien olemassaolosta ohjelma pystyi lukemaan relaatiotietokantasta. Prosessoimaan viestit ja tallentamaan ne relaatiotietokantaan myöhempää käyt-

töä varten. Ruby-ohjelmistossa tärkeässä osassa oli libIEC61850-kirjasto¹. libIEC61850-kirjasto on avoimen lähdekoodin C-kielellä toteutettu kirjasto, joka abstrahoi IEC 61850 -standardin matalan tason määrittämiä palvelukutsuja ja datarakenteita helpokäyttöiseksi rajapinnaksi. Kirjasto tarjosi toiminnallisuuden IED-laitteella olevan serveriohjelmiston, sekä IED-laitetta käyttävän asiakaohjelmiston toteuttamiseen. IED-laitteen serverille kirjasto tarjosi funktioita ja rakenteita IEC 61850 määrittämien luokkien ja hierarkian rakentamiseen ja käsittelyyn. IED-laitteen asiakasohjelmalle kirjasto tarjosi funktioita ja rakenteita standardin määrittämiin palveluihin, kuten arvojen lukuun ja asettamiseen, datajoukkojen käyttöön ja viestien tilaamiseen. Tätä samaa kirjastoa käytettiin myös tämän työn toteutetussa ohjelmistossa. Koska demoversiossa ja tämän työn toteutuksessa keskitytään vain asiakasohjelmiston tekemiseen, käytetään kirjastosta vain sen asiakasohjelman toteutuksen ominaisuuksia.

Kirjasto oli rakennettu käyttämään MMS-protokollaa tiedonsiirrossa IED-laitteen ja sen asiakasohjelman välillä, kuten IEC 61850 -standardin osassa 8-1 määritetään. Kuvassa 14 on esitetty kirjaston kerrosarkkitehtuuri asiakasohjelmalle. Kirjastoon oli toteutettu laiteabstraktiokerros (engl. hardware abstraction layer, lyhennetään HAL). HAL:in avulla kirjasto voi toimia monella eri laitealustalla, ja käyttäjä voi tarvittaessa lisätä oman HAL-implementaation. Demoversiota ajettiin Linux-käyttöjärjestelmällä, joten kirjastosta käytettiin olemassa olevaa Linux HAL toteutusta. Kuvassa 14 on punaisella merkitty laitkot, jotka kirjaston käyttäjä voi tarjota, keltaisella kirjaston uudelleenkäytettävät MMS-protokollan osuudet ja sinisellä IEC 61850 -standardin toteuttavat osuudet. Kuvaan on merkitty vihreällä demoversioon toteutetut osuudet, eli Ruby-kielelle liitos C-kieleen ja tämän päälle Rubylla ohjelmoitu demo.



Kuva 14. libIEC61850-kirjaston kerrosarkkitehtuurin komponentit, vihreällä Ruby toteutukseen lisätyt osat (pohjautuu kuvaan [15]).

¹<http://libiec61850.com>

Ruby-koodista C-kielen funktioiden kutsuminen ei ole suoraan mahdollista, vaan kielten väliin täytyy toteuttaa liitos. Demoversiossa liitos oli tehty käyttäen Rubyille saatavaa `ruby-ffi`-kirjastoa² (engl. Foreign Function Interface, lyhennetään FFI). Liitoksen avulla Ruby voi kutsua C-kielen funktioita ja käyttää sen struktuureita ja muuttujia. Demossa kirjasto hoiti matalan tason IEC 61850 asiat, ja Ruby-koodi keskittyi liitoksen avulla korkean tason viestin parsintaan ja tallennukseen tietokantaan.

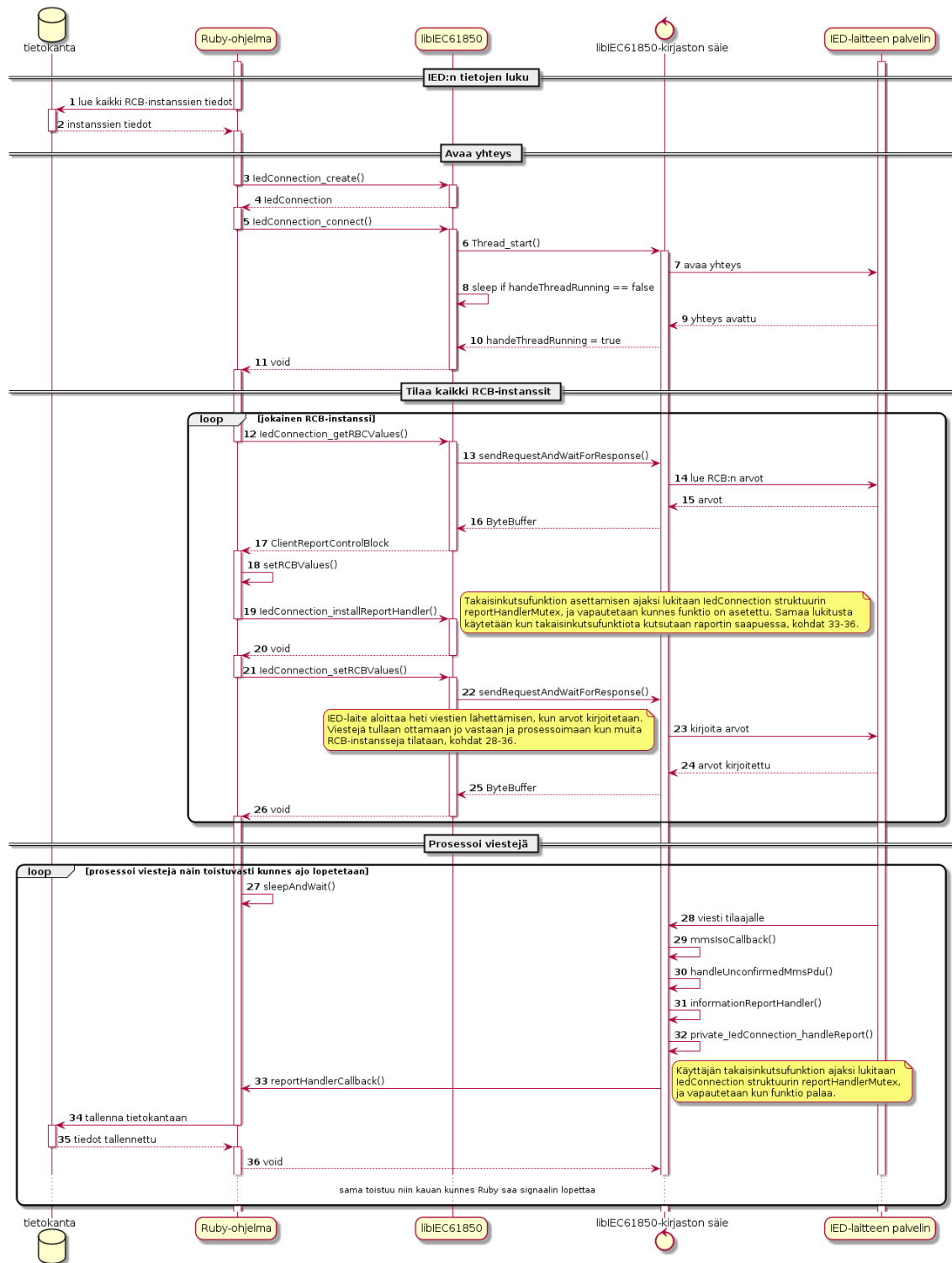
3.2 Ongelmakohdat ja analysointi

Demoversiossa ohjelma oli toteutettu Ruby on Rails kehiksen päällä ajettavaksi. Ruby on Rails kehys on tarkoitettu web-sovellusten toteuttamiseen Ruby kielellä. Se tarjoaa Active Record nimisen ORM-kerroksen (engl. Object-relational Mapping) tietokannan käsittelyn helpottamiseen. ORM-kerros abstrahoi relaatiotietokannan käyttämisen oliopohjaiseksi ja kyselyitä tietokantaan voi suoraan tehdä Ruby-kielellä. Demoversio käytti Railsin Active Record ORM-kerrosta tietokannan käyttämiseen. Eli ennen ohjelman ajamista ohjelmaan täytyi ladata Railsin ajoympäristö muistiin, joka aiheutti sen että yksinkertaisen ohjelman täytyi varata iso määrä muistia ennen suoritusta. Linuxin `htop`-ohjelmalla katsottuna, prosessi varasi noin 150 Mt muistia ajoa varten.

Ohjelma luki tietokannasta IED-laitteen, sekä sen kaikki RCB-instanssien tiedot. Tietojen avulla ohjelma tiesi mikä IED-laitteen IP-osoite on ja mitkä olivat RCB-instanssien referenssit. Ohjelmaan pystyi syöttämään eri tietoja ainoastaan tietokannan kautta ennen ajoa. Tämän jälkeen ohjelman toiminta, jokaisen RCB-instanssin viestien tilaukseen ja prosessointiin on esitetty sekvenssikaaviossa kuvassa 15. Kuvassa ohjelman kaksi eri silmukkaa on esitetty kahdella eri loop-laatikolla. Sekvenssikaaviossa osallisena ovat tietokanta, Ruby-ohjelma, `libIEC61850`-kirjasto, `libIEC61850`-kirjaston natiivisäie ja IED-laitteen palvelinohjelma. Rubyn ja `libIEC61850`-kirjaston liitos oli tehty `ruby-ffi`-kirjastolla ja kirjaston natiivisäie on vastuussa yhteyden ylläpidosta ja datan siirtämisestä. Sekvenssikaavioon on merkitty paksulla suorituksessa olevat palkit minäkin ajan hetkenä, esimerkiksi IED-laitteen palvelinohjelmisto on koko ajan ajossa.

Tietokannasta luettujen tietojen jälkeen ohjelma muodostaa yhteyden IED-laitteelle, ensin tekemällä instanssin `IedConnection` struktuurista funktiolla `IedConnection_create()`. Tämän jälkeen struktuuri annetaan `IedConnection_connect()` funktiolle, joka avaa yhteyden IED-laitteelle ja palaa vasta kun vastaus saapuu. Tässä vaiheessa `libIEC61850`-kirjasto käynnistää erillisen natiivisäieen yhteyden viestien vastaanottoon. Tämä tapahtuu kirjaston lähdekoodissa `src/mms/iso_client/iso_client_connection.c` funktiossa `IsoClientConn` riveillä 429–434 [21]. Tätä säiettä kirjasto käyttää tulevien viestien vastaanottoon ja lähettämiseen. Yhteyden avauksen jälkeen jokainen RCB-instanssi tilataan lukemalla ensin sen arvot IED-laitteelta funktiolla `IedConnection_getRCBValues()`. Funktiokutsu nukkuu ja palaa vasta kunnes erillinen säie ilmoittaa että vastaus on saapunut, tai yhtey-

²<https://github.com/ffi/ffi>



Kuva 15. Sekvenssikaavio kaikkien RCB-instanssien tilaukseen ja niiden viestien tallentamiseen yhdeltä IED-laitteelta Ruby-ohjelmalla.

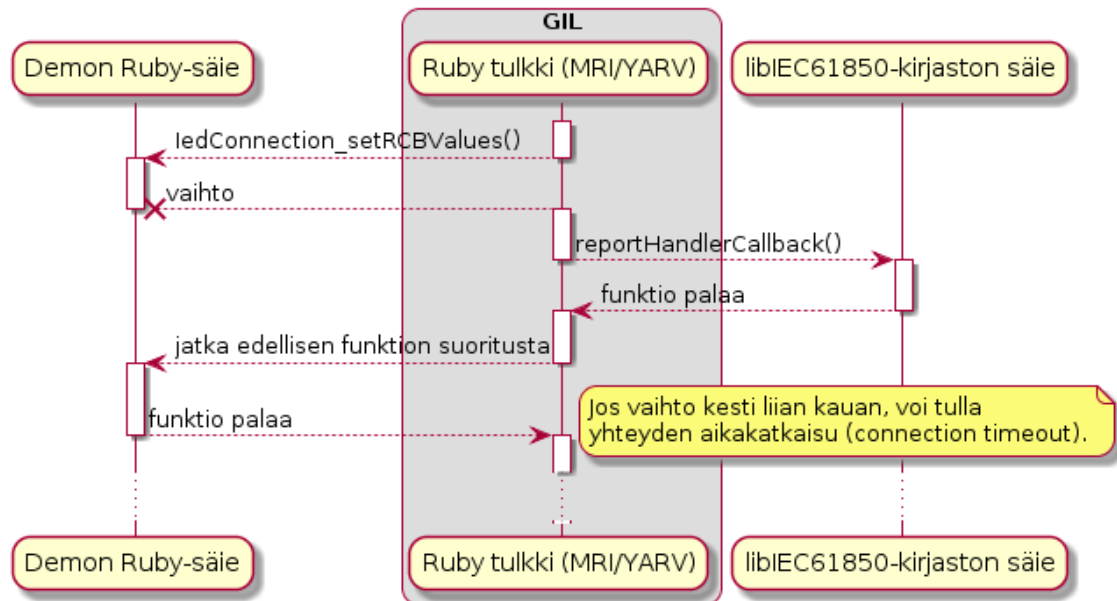
den aika ylittyy. Kirjaston funktio, joka tämän hoitaa on `sendRequestAndWaitForResponse()` ja on määritetty `src/mms/iso_mms/client/mms_client_connection.c` riveillä 345–418 [21]. RCB-arvot luettuaan, kirjasto palauttaa struktuuriin `ClientReportControlBlock`, joka sisältää luetut tiedot RCB-instanssista. Samaa struktuuria käytetään arvojen muutta-

miseen ja niiden takaisin kirjoittamiseen IED-laitteelle. Ennen muunneltujen RCB-arvojen takaisin kirjoittamista ja viestien tilaamista, täytyy kirjastolle asettaa takaisinkutsufunktio, jota kirjastoo kutsuu aina kun tilattu viesti saapuu IED-laitteelta. Takaisinkutsufunktio asetetaan `IedConnection_installReportHandler()`, joka ottaa parametrikseen funktiopointterin ja vaihtoehtoisen parametripointterin. Asetuksen ajaksi kirjasto lukitsee `reportHandlerMutex`. Jos lukituksen aikana saapuu viesti, joutuu erillinen säie nukkumaan ja odottamaan lukituksen vapautusta, kohdat 33–36. Tämän jälkeen arvot kirjoitetaan takaisin IED-laitteelle funktiolla `IedConnection_setRCBValues()`. Tämä funktio myös palaa vasta kunnes IED vastaa, tai yhteyden aika ylittyy, kuten aikaisemmin. Heti arvojen kirjoitusten jälkeen IED aloittaa lähettämään viestejä tilaajalle. Eli samalla kun muita RCB-instansseja tilataan, jo tilatut RCB-instanssit lähettävät jo viestejä ja aiheuttavat takaisinkutsufunktion suorittamisen. Kun kaikki RCB-instanssit on tilattu, ohjelma jää viimeiseen silmukkaan odottamaan ja prosessoimaan viestejä. Kun viesti saapuu, säie kutsuu ensin sisäisesti `mmsIsoCallback()` funktiota, joka kutsuu muita kirjaston sisäisiä funktioita ja lopuksi asetettua takaisinkutsufunktiota. Takaisinkutsufunktio on liitetty Ruby-funktioon ja funktio tallentaa raportin tiedot tietokantaan. Ruby-funktion suorituksen ajaksi kirjasto lukitsee `reportHandlerMutex`, ja vapautetaan kunnes Ruby-funktion suoritus palaa. Tätä jatkuu niin kauan kunnes ohjelmalle lähetetään jokin signaali, joka lopettaa sen suorituksen. [14, 27]

Demossa isoimpana ongelmana oli sen huono suorituskky ja toiminnan epävarmuus RCB-instanssien määrän ollessa enemmän kuin muutama. RCB-instanssien määrän ollessa liian suuri, ohjelma saattoi epäonnistui joidenkin tilaamisessa, koska yhteys aikakatkaisi arvojen kirjoituksessa tai luvussa. Lisäksi, jotta kaikki RCB-instanssit saatiin edes tilattua, saattoi ohjelmalta kestää siinä noin puoli minuuttia esimerkiksi tilata 13 RCB-instanssia.

Huonoon suorituskkyyn oli syynä muutama asiaa. Yksi niistä oli Ruby-kielen huonompi suorituskky verrattuna natiivisti käännettyyn C-kieleen. Ruby on tulkattava kieli kuten esimerkiksi Python, joka tulkataan rivi kerrallaan ja suoritetaan. Lähdekoodia ei käännetä kokonaan ensin konekäskyiksi erillisellä kääntäjällä, kuten C-kielessä. Valmiiksi käännetty lähdekoodi tarvitsee vain ajaa, kun taas tulkattavassa kielessä rivi täytyy ensin tulkata ja sitten ajaa. Rubyssa käytettiin sen oletustulkkia MRI/YARV (engl. Matz's Ruby Interpreter, lyhennetään MRI tai Yet another Ruby VM, lyhennetään YARV). Ruby versiosta 1.9 eteenpäin käyttää YARV tulkkia. Toinen syy oli Ruby-kielen oletustulkissa oleva globaali tulkkilukitus (engl. global interpreter lock, lyhennetään GIL, tai global virtual machine lock, lyhennetään GVL). GIL pakottaa Ruby-ohjelman ajoon vain yhdellä CPU:lla ja vain yksi säie vuorossa kerrallaan ja on riippumaton käyttöjärjestelmän kernelin vuorottajasta [20, s. 131–133]. Kuvassa 16 on esitetty kuinka Ruby-tulkki vuorottaa kahta ajossa olevaa säiettä. Kuvassa Demon Ruby koodi kutsuu `IedConnection_setRCBValues()` funktiota, ajo jää kesken ja tapahtuu vaihto, koska viesti saapui. Takaisinkutsufunktio suoritetaan ja suoritus palaa takaisin aikaisempaan funktion suoritukseen. Tässä vaiheessa jos vaihto on huonolla hetkellä vaihto kesti liian kauan, tulee yhteyden aikakatkaisu ja

RCB-instanssi jää tilaamatta. Huonoon suorituskykyyn mahdollisesti vaikutti myös lukitus `reportHandlerMutex`, jota kirjastossa käytetään kun takaisinkutsufunktio asetetaan ja takaisinkutsufunktio suoritetaan. Lukitus aiheuttaa säikeen nukkumisen niin kauan kunnes lukitus vapautuu. Tässä tapauksessa, jos viestin prosessointi kestää kauan (kuvas-
sa 15 kohdat 33–36). Ja vielä muita RCB-instansseja tilataan silmukassa (kohdat 12–26). Joutuu säie odottamaan lukituksen vapautusta kun takaisinkutsufunktioita asetetaan (kohdat 19–20). Ratkaisuna tähän olisi pitää takaisinkutsufunktio mahdollisimman lyhyenä suoritusajan suhteen.



Kuva 16. Ruby-tulkin globaalin lukituksen toiminta, joka vuorottaa ajossa olevia säikeitä.

Tämän lisäksi demototeutuksessa oli muistivuoto, joka söi muistia ja sitä ei ikinä vapautettu. Muistivuoto johtui todennäköisesti jostakin ohjelmointivirheestä ruby-ffi -kirjaston liitoksen kanssa. Kun liitos Rubystä tehdään C-kieleen, täytyy ohjelmoidan miettiä roskien keruuta tarkasti. Tätä ei normaalisti tarvitse miettiä ollenkaan Rubyssä, koska tulkki implementoi automaattisen roskien keruun. Muistivuoto havaittiin kun ohjelman jättää ajoon pitemmäksi aikaa, on ohjelma varannut melkein kaiken käyttöjärjestelmän muistista itselleen. Lisäksi jos ohjelmaa ajaa ja tarkkailee Linuxin `htop`-ohjelmalla, voi `MEM%`-sarakkeesta huomata prosentuaalisen osuuden kasvavan koko käyttöjärjestelmän muistista. Tulevaisuutta ajatellen lopullinen tiedon tallennuspaikka ei ole muiden tietoa tarvitsevien ohjelmien kannalta järkevä. Näiden ohjelmien pitäisi koko ajan olla kyselemässä uusinta tietoa tietokannasta erikseen. Tämä kuormittaisi turhaan tietokantaa ja varsinkin jos tietoa tarvitsevia ohjelmia on useita.

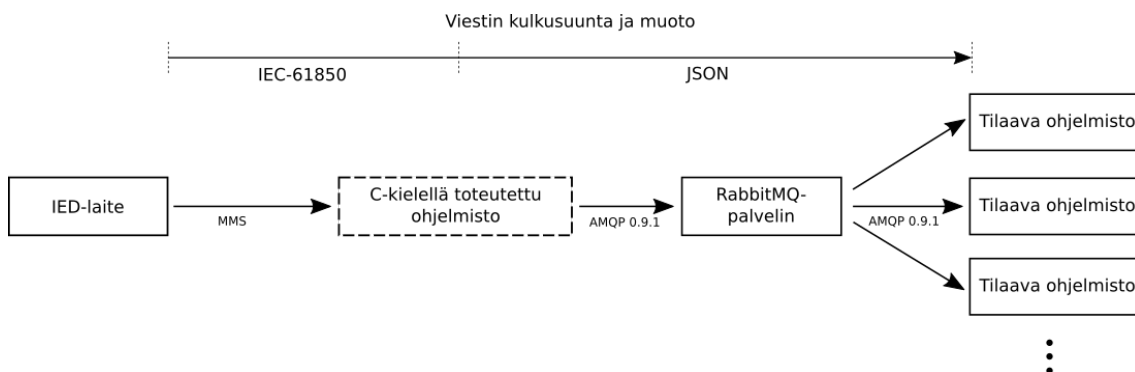
4. SUUNNITTELU

Pitäisikö tähän kirjoittaa ohjelman ajosta ja siihen liittää sekvenssikaavio perustoiminnasta? Kirjoita jos tuntuu että tarvetta.

Tässä osuudessa käydään toteutetun ohjelman suunnittelu läpi ja kerrotaan miten ja miksi ratkaisuihin päädyttiin. Kappaleissa vertaillaan eri vaihtoehtoja ja peilataan demoversion ongelmia ja niiden perusteella yritetään löytää toimiva ratkaisu ongelmaan. Ensinnäkin suunnittelusta ohjelmasta annetaan kattava kokonaiskuva lukijalle ja tämän jälkeen tulevissa kappaleissa mennään jokaisen kohdan yksityiskohtiin tarkemmin.

4.1 Kokonaiskuva

Aikaisemmin kappaleessa 3.1 kuvassa 13 esiteltiin demoversion arkkitehtuuri ja sen toiminta. Kuinka viestit IED-laitteelta kulkee ohjelman läpi ja tallennetaan tietokantaan. Tietokannasta muut ohjelmat lukevat tietoa kyselemällä sitä erikseen. Suunnittelun jälkeen demoversion järjestelmästä päädyttiin kuvassa 17 olevaan järjestelmän arkkitehtuuriin. Kuvassa katkoviivalla on merkitty tässä kappaleessa suunniteltu ohjelmisto. Ja kuvan yläreunassa oleva viiva kuvaa viestin kulkua järjestelmän eri osapuolten läpi ja missä muodossa viesti on missäkin kohtaa.



Kuva 17. Suunnittelun järjestelmän toiminta ja viestin kulkeminen ja muoto eri osapuolten välillä.

Suunnittelussa arkkitehtuurissa C-kielellä toteutettu ohjelma on komentorivipohjainen ja ei käyttänyt tietokantaa. Kaikki ohjelman ajoon annettavat parametrit annetaan komentoriviparametreille ennen ohjelman käynnistämistä, verrattuna demoversion toteutukseen, joka luki tiedot tietokannasta. C-ohjelma voi tilata yhdellä IED-laitteella olevia RCB-instansseja. Tilattuaan RCB-instanssit, ohjelma odottaa viestejä IED-laitteelta IEC 61850-standardin määrittämässä muodossa. Kun viesti saapuu, ohjelma prosessoi sen ja julkaisee AMPQ-standardin pohjaiselle jonopalvelimelle JSON-muodossa (engl. JavaScript Ob-

ject Notation). Lopullisessa toteutuksessa jonopalvelimena käytettiin RabbitMQ-nimistä ohjelmistoa, joka pohjautuu AMPQ-standardin versioon 0.9.1. Jonopalvelimelta muut tilaavat ohjelmat voivat tilata viestejä, ja viestin saapuessa palvelin ilmoittaa siitä asiakkaalle. Toteutettu C-ohjelmisto käytti edelleen demoversiosta tuttua libiec61850-kirjastoa hoitamaan matalan tason IEC 61850 -standardin määrittämän funktionaalisuuden.

4.2 Järjestelmän hajautus ja arkkitehtuuri

Järjestelmän hajauttaminen oli vaatimus uudelle arkkitehtuurille, joka täytyisi ottaa huomioon. Hajautuksella tarkoitetaan että viesteistä kiinnostuneet ohjelmat, pystyisivät niitä tilaamaan ja ottamaan vastaan helposti. Ongelmia ei saisi tulla jos asiakasohjelmia olisi tulevaisuudessa enemmänkin. Demossa erilliset ohjelmat joutuivat lukemaan viestejä jatkuvasti tietokannasta, ilman tietoa siitä milloin uusi viesti olisi saapunut. Tällainen ratkaisu ei tulisi toimimaan pitemmän päälle ja tilanne olisi pahentunut jos tietoa tarvitsevia ohjelmia olisi enemmänkin tulevaisuudessa. Lisäksi tässä toteutuksessa tietokanta on jatkuvan turhan lukemisen ja kuormituksen kohteena. Tilanteeseen tarvittaisiin ratkaisu, jossa tilaava ohjelma voisi tilata viestin ja saada ilmoituksen kun tieto on saatavilla, tilaaja-julkaisija -arkkitehtuuri.

Ratkaisuna olisi voinut ajatella että tietoa tarvitsevat ohjelmat, olisi voineet suoraan tilata viestit IED-laitteelta. Näin kaikki ohjelmat saisivat saman viestin. Kuitenkin tässä esteenä on, että IEC 61850 -standardin määrittäksen mukaan yksi RCB-instanssi voi olla vain tilattuna yhdellä asiakkaalla kerrallaan, niinkuin teorian kappaleessa 2.1.6 käsiteltiin. Ja IED-laitteiden RCB-instanssit ovat rajalliset ja päätetty laitteen konfiguroinnin yhteydessä. Lisäksi IED-laitteet pystyvät rajoittamaan päällä olevien yhteyksien määrää johonkin lukuun. Tavoitteena siis olisi minimoida avoimet yhteydet IED-laitteelle, ja samalla tarjota sama viesti mahdollisimman monelle siitä kiinnostuneelle ohjelmalle. Näistä vaatimuksista päästään ratkaisuun, missä yksi ohjelma tilaa kaikki halutut RCB-instanssit yhdeltä IED-laitteelta. Odottaa viestejä ja lähettää ne edelleen muille niitä tarvitseville ohjelmille. Viestejä tarvitsevien ohjelmien määrä voi vaihdella tarpeen mukaan. Tästä päästään vaatimukseen, että IED-laitteelta viestejä tilaavan ohjelmiston ei tarvitsisi tietää muista tilaavista ohjelmista mitään. Ohjelman pitäisi pystyä julkaisemaan viestit eteenpäin, välittämättä siitä kuka viestejä vastaanottaa.

Ratkaisuna yllä mainittuihin vaatimuksiin oli sijoittaa IED-laitteen ja muiden tilaavien ohjelmien väliin väliohjelmisto, kuten kuvassa 17 on C-ohjelma sijoitettu. Näin pystyttiin minimoimaan yhteyksien määrä IED-laitteelle yhteen. Lisäksi sijoittamalla C-ohjelman ja muiden tilaavien ohjelmien väliin jonopalvelin, saadaan aikaan joustavuus mitä haluttiin. C-ohjelman ei tarvitse välittää siitä kuka viestejä vastaanottaa ja jonopalvelimen avulla yhden julkaisijan voi tilata monta erillistä tilaaja. Jonopalvelimen avulla jokainen tilaaja saa saman alkuperäisen viestin, mutta kopiona. Koska standardi ei määrittänyt muita viestien tilaamisen mahdollisuuksia, tämä suunnitelma arkkitehtuurista täytti kaikki sille asetetut vaatimukset.

Demoversiossa ohjelma luki IED-laitteen tiedot kuten IP-osoitteen ja RCB-instanssien referenssit tietokannasta ja tallensi saapuneet viestit tietokantaan. Nyt kun viestit julkaistiin erilliselle jonopalvelimelle, niin tietokantaa ei siihen enää tarvinnut. C-ohjelman tarkoitus oli vain olla väliohjelma viestien välittämiseen eteenpäin, joten siihen ei tarvittu käyttöliittymääkään. Ohjelmasta päätettiin tehdä komentorivipohjainen toteutus, jolle kaikki tiedot voitaisiin syöttää komentorivillä parametreilla käynnistyksen yhteydessä. Tällä suunnitelmalla toteutus ei tarvitsisi tietokantaa ollenkaan, joten se voitiin tiputtaa pois suunnitelmasta.

4.3 Suorituskyky ja kielen valinta

Demoversio oli ohjelmoitu Ruby-kielellä ja siinä oli paikoin suoritukseen liittyviä ongelmia ja epävarmuutta, etenkin viestien ja RCB-instanssien määrän olessa suurempi. Syitä ja ongelmia käytiin läpi kappaleessa 3.2. Oli selvää että ohjelman suorituskykyä täytyi saada parannettua ja siinä olevat ongelmat korjattua esimerkiksi muistivuoto. Ennen koko ohjelman uudelleenkirjoitusta, Ruby-ohjelmaa kokeiltiin saada toimimaan JRuby¹ nimisellä Ruby-tulkilla. Tavoitteena saada demoversion toteutus toimimaan ilman GIL:iä ja säikeet suoritukseen rinnakkain. JRuby on Ruby-koodin tulkki, joka suorittaa Ruby-lähekoodia Java virtuaalikoneen (engl. Java Virtual Machine, lyhennetään JVM) päällä. JRuby mahdollistaa säikeiden suorituksen rinnakkain JVM:n omilla säikeillä ja näin ollen suorituksen pitäisi olla nopeampaa [28]. Jos tämä lähtökohta olisi toiminut, olisi edelleen järjestelmän arkkitehtuuria pitänyt muuttaa samaan suuntaan, kuin kappaleessa 4.2 kuvattiin. Tämän lisäksi demossa oleva muistivuoto olisi pitänyt korjata. JRuby ei kuitenkaan toiminut ja nopean yrityksen jälkeen päätettiin vain palata suunnitelmaan kirjoittaa koko ohjelma uudestaan. Syynä tähän oli että demoversio oltiin tehty osaksi isompaa Rails projektia, joka toimi Rubyn oletustulkin päällä. Ja JRuby ei tukenut kaikkia projektin kirjastoja mitä se käytti. Rubyssä kirjastoja kutsutaan jalokiviksi (engl. gem). Seurauksena olisi ollut saman projektin ylläpitäminen kahdelle eri tulkille tai asennettavien pakettien erottaminen. Kuitenkaan yrittämisen jälkeen tätäkään ei saatu toimimaan loppupelissä. Kysymyksenä tämän aikana tuli ajan käyttö ja fakta että demosta olisi pitänyt korjata ja paikata monta asiaa. Päätettiin toteuttamaan koko ohjelmisto uudestaan erillisellä kielellä jossa ei olisi suorituskykyongelmia. Samalla uudessa toteutuksessa ohjelman pystyi alusta asti tekemään asetetut tavoitteet mielessä ja demoversion ongelmia ei tarvitsisi korjata.

Uuden toteutuksen kieleksi valittiin C-kieli. Isona syynä kielen valintaan oli tekijän iso mieltymys matalan tason ohjelmointiin ja C-kieleen. Lisäksi C-kieli käännetään alustalle suoraan konekäskyiksi, joiden suoritus on nopeampaa kuin tulkattavan kielen, kuten Ruby ja Python. Kielen valinnan yhteydessä kuitenkin oli hyvä varmistaa kaikkien suunniteltujen liitosten mahdollisuus. C-kielelle löytyi kirjastoja RabbitMQ-jonopalvelimen käyttämiseen ja lisäksi JSON rakenteen muodostamiseen. Hyötynä vielä C-kielen valinnasta

¹<http://jruby.org/>

oli, että demossa käytettyä libIEC61840 kirjastoa pystyi käyttämään suoraan ilman erillistä liitosta, koska kirjasto oli myös tehty C-kielellä. Tarkemmin käytettyihin kirjastoihin ja toteutukseen mennään kappaleessa 5.

4.4 Prosessoidun viestin muoto ja rakenne

Saapuva viesti esitettiin libIEC61850-kirjastossa ClientReport struktuurin instanssina. Stuktuuri sisältää viestin datan ja sen voi lukea käyttämällä kirjaston tarjoamia funktioita [16]. Saapunut viesti haluttiin jakaa jonopalvelimen läpi muille osapuolille, joten viestin täytyi olla helposti luettavassa muodossa muille ohjelmille. Viesti päädyttiin muuttamaan helposti ymmärrettäväksi JSON-rakenteeksi. JSON-rakenteen voi helposti ihminen lukea ja se on nykypäivänä paljon käytetty tiedonsiirtomuoto erilaisissa web-palveluissa ja rajapinnoissa. Myöskin JSON-rakenteiden lukemiseen on monelle eri kielellä olemassa valmiita kirjastoja sen monikäyttöisyyden takia [22].

Liitteessä A on esitetty prosessoidun JSON-rakenteen muoto johon tässä työssä päädyttiin. Ja minkä tässä työssä toteutettu C-ohjelma lopulta julkaisi RabbitMQ-jonopalvelimelle. Standardin määrittämää viestin rakennetta ja sisältöä käytiin läpi kappaleessa 2.1.8. JSONin rakenne pääasissa noudattaa standardin määrittämää viestin rakennetta, mutta joitakin asioita on tehty toisin. Lisäksi C-ohjelma myös lisäsi viestiin lisää tietoa attribuuteista selkeyden takia kuten viitteen, tyypin ja koon. Kuinka tämä toteutettiin käsitellään tarkemmin kappaleessa 5.

Standardin viestin kenttien määrää pystyi säätämään RCB-instanssin OptFlds-attribuutilla. JSONiin kuitenkin haluttiin lisätä kaikki mahdolliset kentät selkeyden vuoksi. Joten jos kenttä viestistä puuttui, asetettiin sen arvoksi JSONissa null. Esimerkiksi liitteessä A kentän confRevision arvo on null. Eli tällöin RCB-instanssissa OptFlds-attribuutin confRevision on ollut epätosia. Sama käytäntö toistettiin kaikille muillekin vaihtoehtoisille kentille. Tällä periaatteella viestin OptFlds-kenttä voitiin jättää pois JSONista. JSONiin päädyttiin lisäämään FCD- ja FCDA-viitteiden alla viitattut oikeat attribuutien viitteet, tyypit ja koot arvojen lisäksi. Tämä toteutettiin selkeyden takia, mitkä arvot oikeasti kuuluvat viestiin ja mitkä ovat niiden viitteet. Standardissa viesti sisälsi vain datajoukon FCD- tai FCDA-viitteen ja taulukon arvoja mitä sen alla viitattiin. Liitteessä A oleva JSONin rakenteessa ensimmäinen values-attribuutti on siis lista datajoukon FCD- tai FCDA-viitteitä ja siihen liittyvät kentät mitä viestin rakenteessa oli (kuva 6). Eli viestin Reason Code on laitettu reasonForInclusion attribuuttiin. Viestin DataRef-kenttä on pilkottu kolmeen eri kenttään mmsReference, reference ja functionalConstraint. Viestien viitteet tulevat MMS-protokollamäärittäytksen muodossa, eli pisteet (.) on korvattu dollarilla (\$) ja viite sisältää funktionaalisen rajoitteen. Nyt mmsReference sisältää viestin alkuperäisen MMS-viitteen, reference sisältää standardin abstraktin viitteen ja functionalConstraint sisältää funktionaalisen rajoitteen. Nämä on erotettu selkeyden takia, koska todennöisesti jotkin asiakasohjelmat tarvitsivat standardin käyttämää abstraktia viitettä. Tällä asiakasohjelma välttää teksimuunnokset. JSONin sisempi values-attribuutti sisältää taulukon itse viestin

arvoista, mutta C-ohjelma lisäsi niihin niiden oikeat viitteet, tyyppin ja koon. Poikkeuksena boolean ja utc-time tyyppit, jolla ei ole kokoa ollenkaan. Koko kertoo monellako bitillä kyseinen attribuutti esitetään ja se voi vaihdella saman tyyppin välillä (esimerkiksi bit-string). Myöskin bit-string tyyppille päädyttiin lisäämään kaksi eri arvoa valueLittleEndian ja valueBigEndian. Tämä sen takia, koska tavujärjestys ei ole välttämättä tiedossa missä järjestyksessä bitit muuttujassa ovat. Päätettiin tarjota kummatkin vaihtoehdot asiakkaalle. Ajat päätettiin antaa suoraan siinä formaatissa ja tyyppinä mitä ne tulevat viestistä. Eli viestin päätason aikaleima on millisekunteja UNIX-ajanlaskun alusta 1. tammikuuta 1970 klo 00:00:00 UTC tähän hetkeen. Attribuuteissa tyyppiltään utc-time, luku on sekunteja samasta UNIX-ajanlaskusta tähän hetkeen [8, s. 26–27].

5. TOTEUTUS

Kirjoita tähän osioon siitä kuinka suunniteltu arkkitehtuuri toteutettiin ja millä tekniikoilla. Tämä osio käyttää lyhyitä koodiesimerkkejä hyväkseen selittämään lukijalle kuinka toteutus tehtiin, jotta lukija voisi itse toteuttaa samanlaisen ohjelmiston.

5.1 Ohjelmiston toteutuksen valinta

Kirjoita tähän miksi päädyttiin tietynlaiseen ohjelmiston toteuttamiseen. Työssä on mietitty komentorivipohjaista toteutusta. Lisäksi mille alustalle ohjelmisto suunnitellaan Windows vai Linux.

5.2 Kielen valinta

Kirjoita tähän mikä kieli valittiin toteutuksen tekemiseen ja miksi tämä. Alustava suunnitelma on toteuttaa C-kielillä.

5.3 RabbitMQ

Kirjoita tähän RabbitMQ toteutuksesta. Kirjasto toteuttaa AMQP-standardin määrittämiä eri viestintämalleja. Kerro kuinka sitä hyödynnetään tässä työssä ja vähän sen että mitä vaatii.

5.4 Käytettävät kirjastot

Kirjoita tähän erilaisista kirjastoista mitä toteutukseen valittiin ja miksi. Alaotsikoita voi lisätä jos toteutukseen tarvitaan muita kirjastoja.

5.4.1 libiec61850

IEC 61850 -standardin toteuttava C-kirjasto joka tekee raskaan työn standardin määrittämien palveluiden toteuttamiseen ja muodostamiseen. Kirjasto tarjoaa rajapinnat serveri- ja asiakasohjelmiston toteuttamiseen, mutta vain asiakasohjelmiston rajapintoja käytetään. Kirjasto tarjoaa myös rajapinnat haluttujen raporttien tilaamista varten. Kirjaston nettisivu täältä: <http://libiec61850.com/libiec61850/>.

5.4.2 rabbitmq-c

RabbitMQ:n rajapinnan toteuttava kirjasto C-kielen ohjelmille. Kirjastolla voidaan toteuttaa julkaisevia ja tilaavia ohjelmistoja. Kirjastosta käytetään julkaisevan puolen toteutusta. Kirjasto löytyy täältä: <https://github.com/alanxz/rabbitmq-c>.

5.4.3 JSON-formatointi

Joku kirjasto JSON formatointiin C-kielelle. Näkyy olevan parikin vaihtoehtoa. Perustele tähän valinta ja miksi.

5.5 Jatkokehitys

Kirjoita tähän ideoita mitä jää jatkokehitykseen ja mitä ohjelmistossa on puutteita tai mitä jäi tekemättä.

6. ARVIOINTI

Kirjoitta tähän arviota työn tuloksista.

7. TULOKSET

Kirjoita tähän lopputuloksen analysoinnista ja peilaa saatuja tuloksia työlle alussa asetettuihin kysymyksiin. Mitä jäi saavuttamatta, mitä saavutettiin ja miten hyvin? Mitä olisi voinut parantaa? Voi jakaa aliotsikoihin jos tarvetta.

8. YHTEENVETO

Kirjoita tähän ensin arviointi ja yhteenveto työstä ja sen lopputuloksista. Mitä hyötyjä työnantaja työstä saa ja jatkokehitysideoita. Mitä työssä meni hyvin ja mitä olisi voinut tehdä toisin?

LÄHTEET

- [1] AMQP 0-9-1 Model Explained, RabbitMQ verkkosivu. Saatavissa (viitattu 31.7.2018): <https://www.rabbitmq.com/tutorials/amqp-concepts.html>
- [2] AMQP Advanced Message Queuing Protocol v0-9-1, Protocol Specification, mar. 2008, 39 s. Saatavissa (viitattu 10.7.2018): <http://www.amqp.org/specification/0-9-1/amqp-org-download>
- [3] C. Brunner, IEC 61850 for power system communication, teoksessa: 2008 IEEE/-PES Transmission and Distribution Conference and Exposition, April, 2008, s. 1–6.
- [4] B. E. M. Camachi, O. Chenaru, L. Ichim, D. Popescu, A practical approach to IEC 61850 standard for automation, protection and control of substations, teoksessa: 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), June, 2017, s. 1–6.
- [5] IEC 61850-1 Communication networks and systems for power utility automation – Part 1: Introduction and overview, International Electrotechnical Commission, International Standard, maa. 2013, 73 s. Saatavissa (viitattu 15.6.2018): <https://webstore.iec.ch/publication/6007>
- [6] IEC 61850-6 Communication networks and systems for power utility automation – Part 6: Configuration description language for communication in electrical substations related to IEDs, International Electrotechnical Commission, International Standard, jou. 2009, 215 s. Saatavissa: <https://webstore.iec.ch/publication/6013>
- [7] IEC 61850-7-1 Communication networks and systems in substations - Part 7-1: Basic communication structure for substation and feeder equipment - Principles and models, International Electrotechnical Commission, International Standard, hei. 2003, 110 s. Saatavissa (viitattu 16.5.2018): <https://webstore.iec.ch/publication/20077>
- [8] IEC 61850-7-2 Communication networks and systems for power utility automation - Part 7-2: Basic information and communication structure - Abstract communication service interface (ACSI), International Electrotechnical Commission, International Standard, elo. 2010, 213 s. Saatavissa (viitattu 16.5.2018): <https://webstore.iec.ch/publication/6015>
- [9] IEC 61850-7-3 Communication networks and systems for power utility automation - Part 7-3: Basic communication structure - Common data classes, In-

- ternational Standard, jou. 2010, 182 s. Saatavissa (viitattu 16.5.2018): <https://webstore.iec.ch/publication/6016>
- [10] IEC 61850-7-4 Communication networks and systems for power utility automation - Part 7-4: Basic communication structure - Compatible logical node classes and data object classes, International Standard, maa. 2010, 179 s. Saatavissa (viitattu 16.5.2018): <https://webstore.iec.ch/publication/6017>
 - [11] IEC 61850-8-1 Communication networks and systems for power utility automation - Part 8-1: Specific communication service mapping (SCSM) - Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3, International Standard, kes. 2011, 386 s. Saatavissa (viitattu 16.5.2018): <https://webstore.iec.ch/publication/6021>
 - [12] IEC 61850:2018 SER Series, International Electrotechnical Commission, verkkosivu. Saatavissa (viitattu 9.6.2018): <https://webstore.iec.ch/publication/6028>
 - [13] K. Kaneda, S. Tamura, N. Fujiyama, Y. Arata, H. Ito, IEC61850 based Substation Automation System, teoksessa: 2008 Joint International Conference on Power System Technology and IEEE Power India Conference, Oct, 2008, s. 1–8.
 - [14] S. Kozlovski, Ruby's GIL in a nutshell, syysk. 2017. Saatavissa (viitattu 13.8.2018): <https://dev.to/enether/rubys-gil-in-a-nutshell>
 - [15] libiec61850 API overview, libiec61850 verkkosivu. Saatavissa (viitattu 3.8.2018): <http://libiec61850.com/libiec61850/documentation/>
 - [16] libIEC61850 documentation, libiec61850 verkkosivu. Saatavissa (viitattu 18.8.2018): <https://support.mz-automation.de/doc/libiec61850/c/latest/index.html>
 - [17] R. E. Mackiewicz, Overview of IEC 61850 and Benefits, teoksessa: 2006 IEEE PES Power Systems Conference and Exposition, Oct, 2006, s. 623–630.
 - [18] MMS Protocol Stack and API, Xelas Energy verkkosivu. Saatavissa (viitattu 9.7.2018): http://www.xelasenergy.com/products/en_mms.php
 - [19] New documents by IEC TC 57. Saatavissa (viitattu 9.6.2018): <http://digitalsubstation.com/en/2016/12/24/new-documents-by-iec-tc-57/>
 - [20] R. Odaira, J. G. Castanos, H. Tomari, Eliminating Global Interpreter Locks in Ruby Through Hardware Transactional Memory, SIGPLAN Not., vsk. 49, nro 8, hel. 2014, s. 131–142. Saatavissa (viitattu 16.5.2018): <http://doi.acm.org/10.1145/2692916.2555247>

- [21] Official repository for libIEC61850, the open-source library for the IEC 61850 protocols <http://libiec61850.com/libiec61850>, GitHub verkkosivu. Saatavissa (viitattu 17.5.2018): <https://github.com/mz-automation/libiec61850>
- [22] A. Patrizio, XML is toast, long live JSON, kes. 2016. Saatavissa (viitattu 18.8.2018): <https://www.cio.com/article/3082084/web-development/xml-is-toast-long-live-json.html>
- [23] RabbitMQ Compatibility and Conformance, RabbitMQ verkkosivu. Saatavissa (viitattu 11.7.2018): <https://www.rabbitmq.com/specification.html>
- [24] RabbitMQ Tutorial Routing, RabbitMQ verkkosivu. Saatavissa (viitattu 31.7.2018): <https://www.rabbitmq.com/tutorials/tutorial-four-python.html>
- [25] RabbitMQ Tutorial Topics, RabbitMQ verkkosivu. Saatavissa (viitattu 31.7.2018): <https://www.rabbitmq.com/tutorials/tutorial-five-python.html>
- [26] K. Schwarz, Introduction to the Manufacturing Message Specification (MMS, ISO/IEC 9506), NettedAutomation verkkosivu, 2000. Saatavissa (viitattu 9.7.2018): https://www.nettedautomation.com/standardization/ISO/TC184/SC5/WG2/mms_intro/index.html
- [27] J. Storimer, Nobody understands the GIL, kes. 2013. Saatavissa (viitattu 16.5.2018): <https://www.jstorimer.com/blogs/workingwithcode/8085491-nobody-understands-the-gil>
- [28] P. Youssef, Multi-threading in JRuby, hel. 2013. Saatavissa (viitattu 18.8.2018): <http://www.restlessprogrammer.com/2013/02/multi-threading-in-jruby.html>

LIITE A: VIESTISTÄ PROSESSOITU JSON-RAKENNE

```
{
  "dataSetName": "LD0_CTRL/LLN0$StatUrg",
  "sequenceNumber": 0,
  "confRevision": null,
  "timestamp": 1534993167923,
  "bufferOverflow": false,
  "values": [
    {
      "reasonForInclusion": "GI",
      "mmsReference": "LD0_CTRL/CBCILO1$ST$EnaCls",
      "reference": "LD0_CTRL/CBCILO1.EnaCls",
      "functionalConstraint": "ST",
      "values": [
        {
          "reference": "LD0_CTRL/CBCILO1.EnaCls.stVal",
          "type": "boolean",
          "value": false
        },
        {
          "reference": "LD0_CTRL/CBCILO1.EnaCls.q",
          "type": "bit-string",
          "size": 13,
          "valueLittleEndian": 0,
          "valueBigEndian": 0
        },
        {
          "reference": "LD0_CTRL/CBCILO1.EnaCls.t",
          "type": "utc-time",
          "value": 1534845456
        }
      ]
    },
    {
      "reasonForInclusion": "GI",
      "mmsReference": "LD0_CTRL/CBCSWI1$ST$Loc",
      "reference": "LD0_CTRL/CBCSWI1.Loc",
      "functionalConstraint": "ST",

```

```

"values": [
  {
    "reference": "LD0_CTRL/CBCSWI1.Loc.stVal",
    "type": "boolean",
    "value": true
  },
  {
    "reference": "LD0_CTRL/CBCSWI1.Loc.q",
    "type": "bit-string",
    "size": 13,
    "valueLittleEndian": 0,
    "valueBigEndian": 0
  },
  {
    "reference": "LD0_CTRL/CBCSWI1.Loc.t",
    "type": "utc-time",
    "value": 1534845456
  }
]
},
{
  "reasonForInclusion": "GI",
  "mmsReference": "LD0_CTRL/CBCSWI1$ST$Pos",
  "reference": "LD0_CTRL/CBCSWI1.Pos",
  "functionalConstraint": "ST",
  "values": [
    {
      "reference": "LD0_CTRL/CBCSWI1.Pos.stVal",
      "type": "bit-string",
      "size": 2,
      "valueLittleEndian": 0,
      "valueBigEndian": 0
    },
    {
      "reference": "LD0_CTRL/CBCSWI1.Pos.q",
      "type": "bit-string",
      "size": 13,
      "valueLittleEndian": 2,
      "valueBigEndian": 2048
    },
    {
      "reference": "LD0_CTRL/CBCSWI1.Pos.t",

```

```

        "type": "utc-time",
        "value": 1534845480
    },
    {
        "reference": "LD0_CTRL/CBCSW1.Pos.stSeld",
        "type": "boolean",
        "value": false
    }
]
}

```

Ohjelma 1. Viestin prosessoitu JSON-rakenne.