TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MAURI MUSTONEN

SÄHKÖASEMAN ÄLYKKÄÄN ELEKTRONIIKKALAITTEIDEN VIESTIEN TILAUS JA PROSESSOINTI

Diplomityö

Tarkastaja: Prof. Kari Systä

Jätetty tarkastettavaksi 17. toukokuuta 2018

TIIVISTELMÄ

MAURI MUSTONEN: sähköaseman älykkään elektroniikkalaitteiden viestien tilaus ja prosessointi
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua
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. Suomenkieliselle 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, 45 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 Abstraktimalli ja sen osat 5
2.1.3 Loogisen noodin luokkien ja attribuuttien rakentuminen 7
2.1.4 Attribuuttien viittaus hierarkiassa 10
2.1.5 Attribuuttien funktionaalinen rajoite ja niistä muodostetut datajoukot 11
2.1.6 Viestien tilaus ja tilauksen konfigurointi 12
2.1.7 Raportointi-luokan määritys ja toiminta 13
2.1.8 Viestin rakenne ja kuinka sen sisältö muodostuu 16
2.1.9 Abstraktimallin sovitus MMS-protokollaan 20
2.2 Advanced Message Queuing Protocol (AMQP) 20
2.2.1 Advanced Message Queuing -malli ja sen osat 21
2.2.2 Vaihde (exchange) ja reititysavain (routing-key) 22
2.2.3 Suoravaihde (direct exchange) 23
2.2.4 Hajautusvaihde (fanout exchange) 24
2.2.5 Aihepiirivaihde (topic exchange) 24
2.2.6 Otsikkovaihde (headers exchange) 25
2.2.7 Jonon määritys ja viestien kuittaaminen 26
3. PROJEKTIN LÄHTÖKOHDAT 28
3.1 Demoversio ja sen toiminta 28
3.2 Ongelmakohdat ja analysointi 29
4. SUUNNITTELU 34
4.1 Kokonaiskuva 34
4.2 Järjestelmän hajautus ja arkkitehtuuri 35
4.3 Suorituskyky ja kielen valinta 36
4.4 Prosessoidun viestin muoto ja rakenne 37
5. TOTEUTUS 38
5.1 Ohjelmiston toteutuksen valinta 38
5.2 Kielen valinta 38
5.3 RabbitMQ 38
5.4 Käytettävät kirjastot 38
5.4.1 libiec61850 38
5.4.2 rabbitmq-c 39
5.4.3 JSON-formatointi 39
5.5 Jatkokehitys 39
6. ARVIOINTI 40

7. TULOKSET 41
8. YHTEENVETO 42
LÄHTEET 43

KUVALUETTELO

Kuva 1. *IEC 61850 -standardin osat ja niiden väliset relaatiot.* 6
Kuva 2. *IEC 61850 -standardin abstraktimallin osat ja niiden hierarkia.* 6
Kuva 3. *IEC 61850 -standardin määrittämä viitteen rakenne.* 10
Kuva 4. *Puskuroitu viestien tilausprosessi asiakkaan ja palvelimen välillä.* 14
Kuva 5. *Standardin määrittämä lähetetyn viestin rakenne.* 17
Kuva 6. *BRCB-instanssi tarkkailee sille määritettyä datajoukoa ja generoi viestin tapahtuman liipaistessa.* 19
Kuva 7. *Toteutetun ohjelmiston osuus ja rooli käytettävässä kokonaisuudessa tietoliikenteen kannalta.* 21
Kuva 8. *AMQ-mallin osat ja viestin kulku niiden läpi julkaisijalta tilaajalle (pohjautuu kuvaan [2, s. 11]).* 22
Kuva 9. *Suoravaihde (engl. direct exchange), reitittää suoraan sidoksen reititysavaimen mukaan (pohjautuu kuvaan [24]).* 24
Kuva 10. *Hajautusvaihde (engl. fanout exchange), reitittää kaikkiin siihen sidottuihin jonoihin riippumatta reititysavaimesta (pohjautuu kuvaan [1]).* 24
Kuva 11. *Aihepiirivaihde (engl. topic exchange), reitittää kaikkiin siihen sidottuihin jonoihin, joiden reitityskaava sopii viestin reititysavaimeen (pohjautuu kuvaan [25]).* 25
Kuva 12. *Rubylla toteutetun demoversion arkkitehtuuri ja tiedonsiirto.* 28
Kuva 13. *libIEC61850-kirjaston kerrosarkkitehruurin komponentit, vihreällä Ruby toteutukseen lisätyt osat (pohjautuu kuvaan [15]).* 30
Kuva 14. *Sekvenssikaavio kaikkien RCB-instanssien tilaukseen ja niiden viestien tallentamiseen yhdeltä IED-laitteelta Ruby-ohjelmalla.* 31
Kuva 15. *Ruby-tulkin globaalin lukituksen toiminta, joka vuorottaa ajossa olevia säikeitä.* 33
Kuva 16. *Suunnitellun järjestelmän toiminta ja viestin kulkeminen ja muoto eri osapuolten välillä.* 34

TAULUKKOLUETTELO

Taulukko 1. *IEC 61850 -standardin pääkohtien ja niiden alakohtien dokumentit. ...* 5
Taulukko 2. *IEC 61850 -standardin katkaisijaluokan XCBR -määritys. .....* 8
Taulukko 3. *IEC 61850 -standardin DPC-luokan määritys. .....* 9
Taulukko 4. *Osa IEC 61850 -standardin määrittämistä funktionaalisista rajoitteitteista (FC). .....* 11
Taulukko 5. *BRCB-luokan määritetyt attribuutit ja niiden selitteet. .....* 15
Taulukko 6. *RCB-luokan OptFlds attribuutin arvot ja niiden selitteet, ...* 16

LYHENTEET JA MERKINNÄT

Kun työ on valmis. Lisää tähän kaikki lyhenteet aakkosjärjestyksessä.

ACSI	engl. *Abstract Communication Service Interface*, IEC 61850 -standardin käyttämä lyhenne kuvaamaan palveluiden abstraktimalleja
AMQP	engl. *Advanced Message Queuing Protocol*
FFI	engl. *Foreign Function Interface*, mekanismi, jolla ajettava ohjelma voi kutsua toisella kielellä implementoitua funktiota
GIL	engl. *Global Interpreter Lock*, tulkattavassa kielissä oleva globaali lukitus, joka rajoittaa yhden säikeen suoritukseen kerrallaan
HAL	engl. *Hardware Abstraction Layer*, laitteistoabstraktiotaso abstraktoimaan laitteen toiminnalisuus lähdekoodista
IED	engl. *Intelligent Electronic Device*, sähköaseman älykäs elektroninen laite, joka tarjoaa toimintoja monitorointiin ja kontrollointiin
MMS	engl. *Manufacturing Message Specification*
RCB	engl. *Report Control Block*, raporttien konfigurointiin ja tilaukseen tarkoitettu lohko asiakasohjelmalle
XML	engl. *Extensible Markup Language*, 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 elektroniikkalaite (engl. Intelligent Electronic Device, lyhennetään IED). IED-laite 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-laite voidaan esimerkiksi konfiguroida hoitamaan sähkölinjan kytkimenä oloa, joka myös tarkkailee linjan toimintaa mittaamalla 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 demoversioon 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 mitä kaikkea se määrittää. 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. Suunniteltu 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 toiminnalisuuden 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. Fyysistä 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 välillä. Standardin on tarkoitus 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 abstrahoidut mallit mallinnetaan erikseen jollekin tekniikalle, joka sen mahdollistaa. Abtrahoituja malleja käytetään myös määrittämään sähköaseman IED-laitteiden ja aseman muiden osien konfigurointi. Abstrahoitujen mallien ansiosta standardi on pohjana tulevaisuuden laajennoksille ja tekniikoille. Uusien tekniikoiden ilmaantuessa, voidaan standardin 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 kytkety 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 lisäki 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 alkuperäiset 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. Abstrakteja malleja ja niiden mallintamista tekniikalle käsitellään teoriassa erikseen. 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 Abstraktimalli ja sen osat

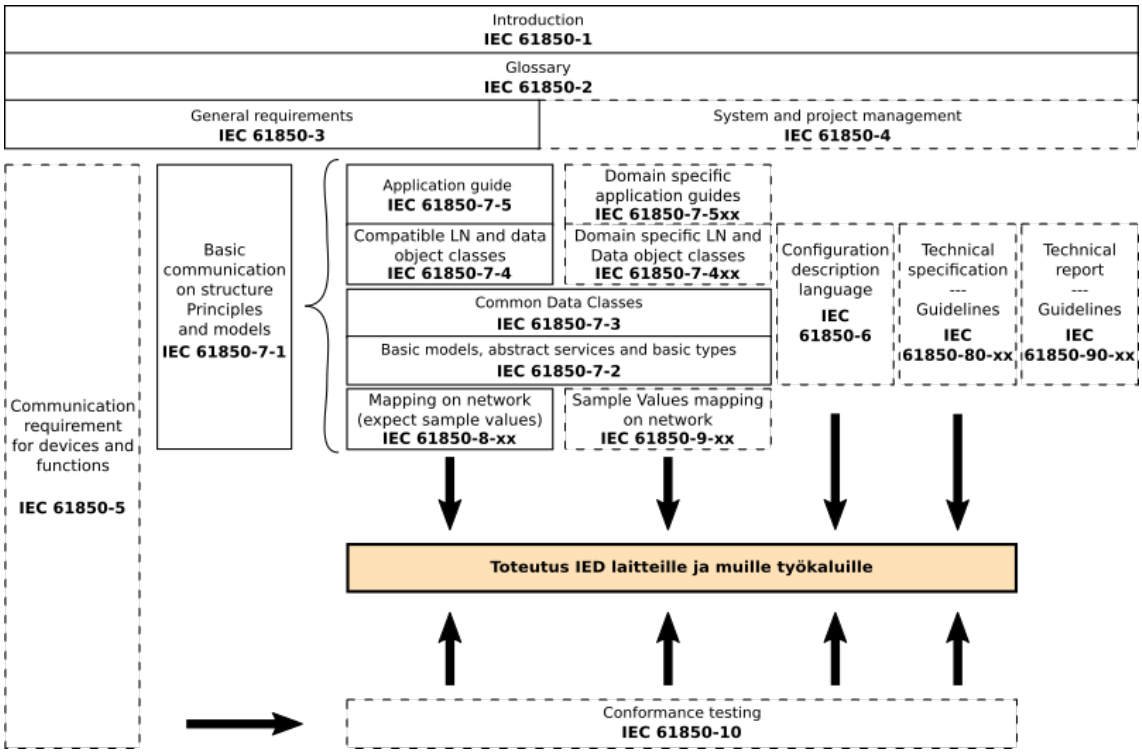
IEC 61850 -standardin lähtökohtana on pilkkoa koko sähköaseman toiminnalisuuden funktiot pieniksi yksilöiksi. Pilkotut yksilöt abstrahoidaan ja pidetään sopivan kokoisina, jotta

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

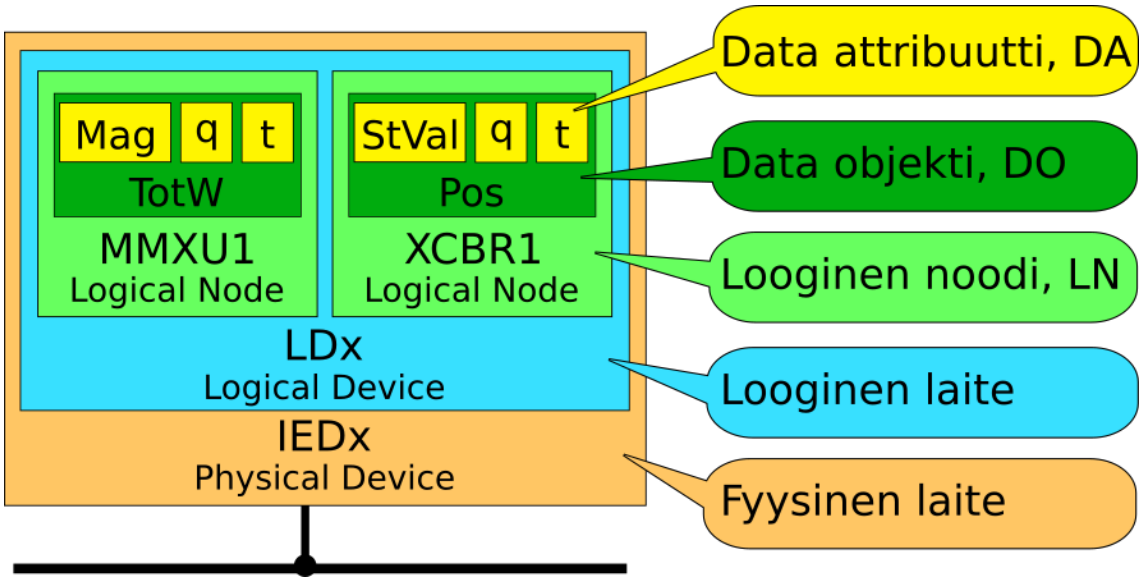
Osa	Otsikko englanniksi
1	Introduction and overview
2	Glossary
3	General requirements
4	System and project management
5	Communication requirements for functions and device models
6	Configuration description language for communication in power utility automation systems related to IEDs
7-1	Basic communication 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

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 ylivirtasuoja (engl. overcurrent protection) omilla IED-laitteillaan. Toimiakseen, laitteiden täytyy vaihtaa informaatiota keskenään [7, s. 31]. Näitä pilkottuja yksilöitä kutsutaan standardissa nimellä looginen noodi (engl. logical node ja lyhennetään LN). Loogiset noodit siis mallinetaan jostakin systeemin käsiteellisestä osasta. Loogisia noodeja käytetään rakentamaan looginen laite (engl. logic device, lyhennetään LD). Looginen laite on aseman ohjausyksikkö ja jokin fyysisen laitteen osa, joka toteuttaa loogisten noodien ohjauksen yhtäaikaa. Ylläolevasta esimerkistä looginen laite olisi aseman linjan suojaukseen liittyvät osat sisältä laite. Looginen laite siis vastaa aseman fyysistä laitetta, joka on kytketty aseman verkkoon ja sillä on IP-osoite. Yksi aseman fyysinen laite voi hoitaa monen loogisen laitteen funktionaalisuuden. Kuvassa 2 on esitetty standardin mallin eri osien hierarkia ja kuinka ne rakentuvat [4, s. 2] [5, s. 24].

Standardin määrittämissä osien hierarkiassa looginen laite on ylin yksilö, joka sisältää loogisia noodeja. Kuvassa 2 IEDx ja LDx vastaavasti. Looginen noodi sisältää data objekteja (engl. data object, lyhennetään DO). Kuvassa 2 loogiset noodit XCBR1 ja MMXU1. Ja data objektit Pos ja TotW. Data objekti sisältää data attribuutteja (engl. data attribute, lyhennetään DA). Kuvassa 2 Mag, stVal, q ja t. Data objekti on tapa koostaa yhteen samaan asiaan liittyvät data attribuutit. Data attribuutit ovat laitteen konfiguroitavia ja luettavia datapisteitä. Data attribuutit kuvaavat esimerkiksi fyysisen laitteen tilaa ja mittausarvoja. Esimerkiksi mitattua jännitettä tai katkaisimen tilaa (kiinni tai auki). Standardin määrittämät data- objektit ja attribuutit voidaan lajitella 5 eri ryhmään:



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



Kuva 2. *IEC 61850 -standardin abstraktimallin osat ja niiden hierarkia.*

- yleinen loogisen noodin informaatio,
- tila informaatio,
- asetukset,
- mitatut arvot ja
- ohjaus [5, s. 25].

Tässä vaiheessa on hyvä mainita, että standardi pyrkii esittämään aseman funktioiden toiminnallisuutta hierarkialla ja oliopohjaisesti. Oliopohjaisesti siten, että standardi määrittää valmiita luokkia erilaisille loogisille noodeille. Esimerkiksi katkaisijalle on määritelty luokka nimeltä XCBR (circuit braker) [10, s. 105–106]. Ja mittaukselle MMXU (measurement) [10, s. 57–58]. Kun aseman toiminnallisuutta esitetään konfiguraatiossa ja IED-laitteella, näitä luokia instanssioidaan tarpeen mukaan, jotta haluttu funktionaalisuus voidaan esittää. Esimerkiksi kuvassa 2 aikaisemmin mainitu XCBR-luokka on instanssioitu nimellä XCBR1 olioksi, ja MMXU-luokka nimellä MMXU1.

Standardi määrittää todella paljon erilaisia valmiita luokkia erilaisille loogisille noodeille valmiina käytettäväksi. Standardi myös määrittää laajennoksien mahdollisuudet luokkia käyttäen. Kaikki määritetyt luokat loogisille noodeille voi löytää standardin osasta 7-4.

2.1.3 Loogisen noodin luokkien ja attribuuttien rakentuminen

IEC 61850 -standardissa määritettyjen luokkien rakennetta on lähestytty oliopohjaisesti. Kaikki luokat määritellään standardissa taulukoilla, joissa on standardoitu kentän nimi, tyyppi, selitys ja onko kenttä optionaalinen. Tässä teoriaosuudessa esitetään esimerkkinä kuinka standardin pohjalta instansioitu looginen noodi ja sen alitason data objektit ja data attribuutit rakentuvat. Esimerkkinä käytetään aikaisemmin esitettyä kuvan 2 mallia katkaisijan instanssia XCBR1. Seuraavassa esimerkissä hierarkiassa mennään ylhäältä alaspäin.

Kuvassa 2 oleville fyysiselle- ja loogiselle laitteelle ei ole olemassa luokkia. Ainoastaan loogiselle laitteelle määritellään yksilöivä nimi, jotta sen alle määritettyjä olioita voidaan viitata. Fyysisen laitteen, eli IED:n yksilöi sille määritetty IP-osoite. Määrityksiä ja mitä luokkia instansioidaan IED-laitteeseen, määritellään sille annetussa XML-pohjaisessa konfiguraatiotiedostossa. Tämän standardi määrittää aikaisemmin mainittussa standardin osassa 6.

Standardissa osassa 7-4 on lista kaikista sen määrittämistä loogisen noodin luokista eri tarkoituksiin. Sähköaseman suunnitteleva insinööri voi ottaa näistä mitä tahansa luokkia ja instansioida niitä IED-laitteeseen, jotta saadaan aikaiseksi haluttu toiminnallisuus. Taulukossa 2 on esitetty XCBR-luokan määritys. 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].

Taulukko 2. *IEC 61850 -standardin katkaisijaluokan XCBR -määritys.*

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

Loogisen noodin luokan kentät, kuten Pos, rakentuvat standardin yleisistä luokista (engl. Common Data Class, lyhennetään CDC). CDC-luokat ovat luokkia jotka sisältävät data attribuutteja, ja ovat yhteisiä monelle eri loogisen noodin luokalle. CDC-luokkien määritykset löytyy standardin osasta 7-3 [5, s. 26]. Taulukossa 3 on esitetty XCBR-luokan Pos-attribuutin, DPC-luokan määritys (engl. controllable double point) [9, s. 44].

Taulukossa 3 voi nähdä data attribuutit stVal, q ja t. Jotka voi myös nähdä kuvassa 2 XCBR-luokan instanssin XCBR1 alla. Tällä periaatteella standardi rakentaa kaikki muutkin mallit hierarkisesti toisiinsa ja sen avulla voidaan selvittää mitä data objekteja looginen noodi sisältää, mitä data attribuutteja mikäkin data objekti sisältää. Taulukossa 3 on esitetty myös attribuuttien tyypit. Jotkin näistä ovat hyvin kuvaavia kuten INT8U tarkoittaa 8-bitin pituista etumerkitöntä kokonaislukua (engl. unsigned interger). Standardi määrittää myös erilaisia rakennettuja attribuuttiluokkia (enlg. constructed attribute classes), joille on määritetty vielä ali data attribuutteja. Hyvä esimerkki taulukosta 3 on data attribuutti q, jonka tyyppi on Quality. Standardin mukaan tällä tyypillä on vielä aliattribuutteina mm. validity, detailQual jne [9, s. 11]. Tyyppejä on paljon enemmänkin ja niitä ei käsitellä sen enempää. Lukija voi tarvittaessa tarkistaa eri tyypit helposti standardista.

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

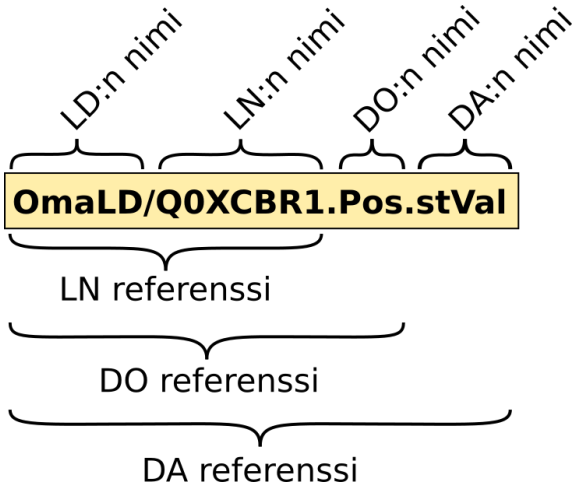
Taulukko 3. *IEC 61850 -standardin DPC-luokan määritys.*

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	SboClassses	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	

CDC-luokka, jolla on taas omat data attribuuttinsa. [8, s. 51,61] [9, s. 36]

Taulukossa 3 on myös määritetty funktionaaliset rajoitteet (engl. Functional constraint, lyhennetään FC), sekä mahdolliset liipaiseimet (engl. trigger options, lyhennetään TrgOp)jokaiselle data attribuutille. Nämä kaksi asiaa käsitellään teoriassa tuonnempana.

Kaikkien yllämainittujen luokkien kenttien määritysten lisäksi standardi määrittää palveluita jokaiselle luokkatyypille erikseen. Määritetyt palvelut ovat abstrakteja ja ne mallinetaan tekniikalle erillisellä standardin osalla. Esimerkkinä palveluista kaikille data objekteille on mm. GetDataValues, joka palauttaa kaikki data objektin attribuuttien arvot. SetDataValues kirjoittaa annetut data attribuuttien arvot. Ja GetDataDirectory palauttaa kaikki data attribuuttien viitteet kyseisessä data objekstissa. Näitä ja muita abstrahoituja malleja viitataan standardissa lyhentellää ACSI (engl. abstract communication service interface) [8, s. 15,45–46] [7, s. 26].



Kuva 3. *IEC 61850 -standardin määrittämä viitteen rakenne.*

2.1.4 Attribuuttien viittaus hierarkiassa

IEC 61850 -standardi määrittää tarkasti kuinka hierarkian eri kohtia viitataan IED-laitteessa. Viitteitä käytetään kun IED-laitteelle tehdään standardin osan 7-2 ACSI-määritysten mukaisia palvelukutsuja. Esimerkiksi jonkin attribuutin arvon asettaminen (SetDataValues) tai lukeminen (GetDataValues). Viitteen avulla IED-laite tietää, mihin loogisen noodin instanssiin ja sen data attribuuttiin palvelypyyntö kohdennetaan. Kuvassa 3 on esitetty kuinka standardi määrittää viitteen muodostumisen loogisesta laitteesta data attribuuttiin asti [7, s. 93].

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 voi olla käyttäjän oma määrittämä nimi, mutta kuitenkin alle 65 merkkiä. Loogisen laitteen nimeen standardi ei puutu. Loogisen noodin instanssin nimi koostuu alku-, keski- ja loppuosasta. Alkuosan käyttäjä voi itse päättää, kuvassa 3 "*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. Loppuosa on instanssin numeerinen arvo, joka ei sisällä kirjaimia. Loppuosan käyttäjä voi itse päättää, jonka ei tarvitse välttämättä olla juokseva numero. Alku- ja loppuosaan muotoon standardi ei anna määrityksiä. Alku- ja loppuosan yhteenlaskettu merkkien pituus täytyy olla alle 13 merkkiä. Data objektien (DO) ja attribuuttien (DA) niminä käytetään standardin määrittämiä nimiä, jotka määritetään niitä vastaavissa luokissa osissa 7-3 ja 7-4 (katso taulukkot 2 ja 3). Riippuen viittauksesta, näistä muodostuu loogisen noodin referenssi, data objektin referenssi ja data attribuutin referenssi. [8, s. 181–182] [7, s. 93–95]

Standardissa määritetään kaksi näkyvyysaluetta (engl. scope) viittaukselle, jotka ovat palvelin- ja looginen laite -näkyvyysalueet. Serverinäkyvyysalueelle viitataan ottamalla viittauksesta pois loogisen laitteen nimi. Eli kuvassa 3 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 3 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 määrittä maksimipituuksia viittauksille. Seuraavaksi kerrotut pituusmääritykset 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 yleiset luokat (CDC) määrittävät käytettävät data attribuutit. Luokat määrittää myös jokaiselle data attribuutille aikaisemmin taulukossa 3 mainitun funktionaalisen rajoitteen (engl. *functional constraint*, lyhennetään FC). Funktionaalinen rajoite kuvaa attribuutin käyttötarkoitusta ja sitä mitä palveluita attribuuttiin voidaan käyttää. Funktionaalinen rajoite voidaan ymmärtää niin sanottuna suodattimena data objektin data attribuuteille. 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 myös määrittä onko attribuutti kirjoitettava tai luettava [8, s. 54].

Taulukko 4. *Osa IEC 61850 -standardin määrittämistä funktionaalisista rajoitteitteista (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

Funktionaalisia rajoitteita käytetään, kun IED-laitteeseen tehdään ACSI-palveluiden mukaisia kutsuja. Esimerkiksi jos halutaan lukea kuvassa 3 OmaLD/Q0XCBR1.Pos polussa olevan data objektin kaikki tilan arvot yhdellä kutsulla, käytettäisiin funktionaalista rajoitetta ST taulukosta 4. Kutsu jättää lukematta kaikki muut data objektin attribuutit, paitsi ne joilla funktionaalinen rajoite on ST. Esimerkkinä taulukon 3 mukaan tulisi vain kentät origin, ctlNum, stVal, q, t, ja stSeld, ja näiden kenttien mahdolliset ala attribuutit.

Funktionaalista rajoitetta voidaan käyttää suodattamaan data attribuutteja data objektista ja niiden ali data objekteista. Toisin sanoen hierarkiassa referenssipisteestä alaspäin oleviin kentiin. Standardi määrittä lyhtenteen FCD (engl. functional constrained data), jota käytetään silloin kun hierarkian ensimmäistä data objektia suodatetaan funktionaalisella rajoitteella, ei ali data objekteja. Aikaisemmin mainittu OmaLD/Q0XCBR1.Pos funktionaalisella rajoitteella ST, on FCD-suodatus. Tässä Pos on ensimmäinen data objekti Q0XCBR1 loogisen noodin jälkeen. Tämän lisäksi määritetään lyhenne FCDA (engl. functional constrained data attribute), jota käytetään silloin kun viitteessa funktionaalisella rajoitteella suodatetaan ensimmäistä data objektia alempia kohtia. Alempia kohtia voi olla mm. ali data objektit ja rakennetut attribuutit, joilla on vielä omia data attribuutteja. FCDA-viite on myös silloin kun viitataan vain yhteen data attribuuttiin sen funktionaalisella rajoitteella. Ainoa ero FCD ja FCDA -viitteiden luokittelun välillä on että FCD kohdistuu ensinmäiseen data objektiin hierarkiassa ja FCDA sitä alempiin kohtiin. [8, s. 55] [11, s. 63]

FCD- ja FCDA -viitteitä käytetään rakentamaan datajoukkoja IED-laitteelle. 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. Datajoukkoja IED-laitteessa käytetään muodostamaan joukkoja tärkeistä data attribuuteista, joita voidaan esimerkiksi lukea ja kirjoittaa yhdellä kutsulla, muutoksia tilata viesteinä ja lokittaa myöhempää käyttöä varten. Näitä määrityksiä käytetään myöhemmin viestien liipaisemiseen ja mitä kenttiä viesteihin sisältyy. Standardi määrittää että datajoukot nimetään ja sijoitetaan jonkin loogisen noodin alle, jotta siihen voidaan viitata. Esimerkkinä viittaus "MyLD/LLN0.Testi1", joka viittaa datajoukkoon nimeltä Testi1, ja joka sijaitsee loogisessa noodissa LLN0. [8, s. 61–68]

2.1.6 Viestien tilaus ja tilauksen konfigurointi

IEC 61850 -standardi määrittää erilaisia liipaisimia data attribuuteille, joita voidaan käyttää liipaisemaan jokin tapahtuma IED-laitteessa. Esimerkiksi DPC-luokan määritys taulukossa 3. Standardi määrittää seuraavia liipaisimia data attribuuteille:

- 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*).

Edellä mainituissa ero datan muutoksen ja päivityksen välillä on se, 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 tarkoittaa, että data attribuuttiin liitetty laatuarvo muuttuu. Laatuarvo kertoo, voiko attribuutien arvoihin luottaa. [7, s. 90]

Standardi määrittää kaksi mahdollisesti liipaistavaa tapahtumaa IED-laitteessa, jotka ovat raportointi ja lokitus. Lokitus on IED-laiteessa tapahtuvien tapahtumien lokitusta myö-

hempää käyttöä ja tarkastelua varten. Esimerkiksi attribuutin arvon muutos. Raportointi on tapahtuma, jossa generoidaan viesti tapahtuman liipaisseista attribuuteista. Tämä viesti lähetetään niitä tilaaville asiakkaille. Jos tilaavaa asiakasta ei ole, viestiä ei generoida. Standardi käyttää sanaa "raportti"(engl. report) näiden viestien kuvaamiseen. Kuitenkin tässä työssä on käytetty sanaa "viesti"tästä eteenpäin "raportin"sijaan. Tämä sen takia, koska suomenkielessä raportti-sana voi tarkoittaa lukijalle muuta merkitystä, kuin verkon yli asiakkaan ja palvelimen välistä viestiä. Tässä työssä keskitytään edelle mainittuihin viesteihin, ei lokitukseen. Ja lopullinen ohjelmisto nimenomaan käsitteli näitä viestejä.

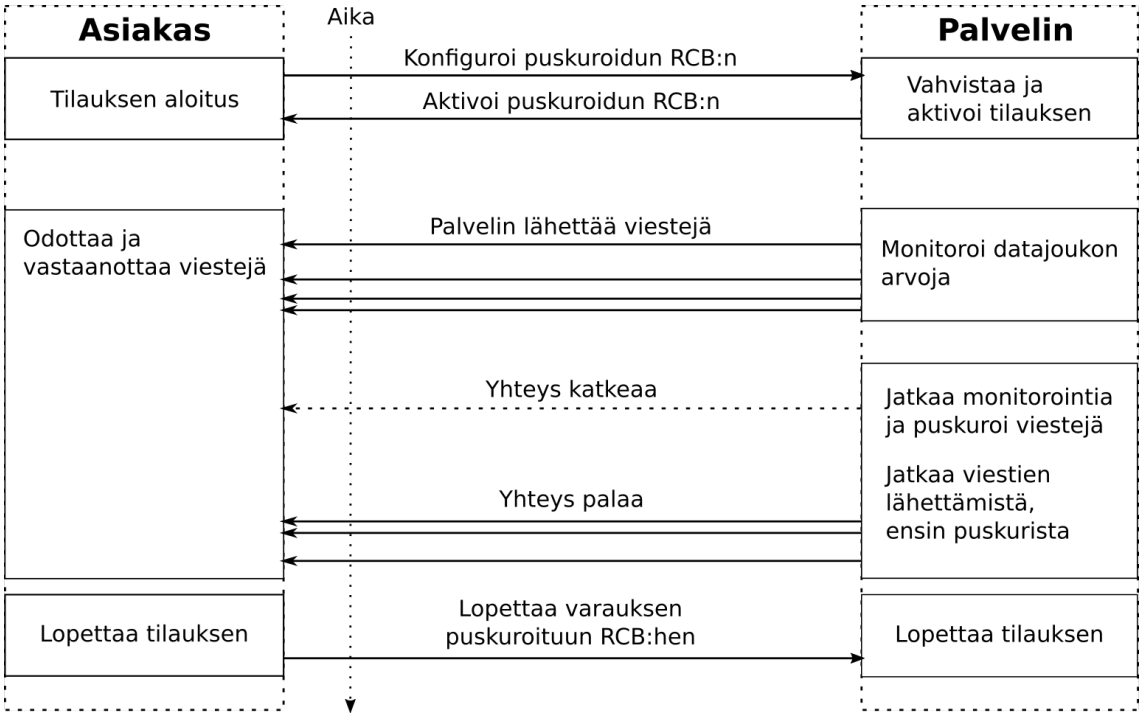
Standardi määrittelee kaksi luokkaa viestien tilaamisen ja konfigurointiin. Luokat ovat puskuroitu viestintälohko (engl. *Buffered Report Control Block*, lyhennetään **BRCB**) ja ei puskuroitu lohko (engl. *Unbuffered Report Control Block*, lyhennetään **URCB**). Tekstissä kumpaakin luokkaan viitatessa käytetään lyhennettä **RCB**. Ainoa ero luokkien välillä on, että BRCB puskuroi 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. Yhteyden katketessa, viestit menetetään. IED-laitetta konfiguroitaessa, luokista tehdään instansseja asiakkaiden tarpeen mukaan. Standardi määrittää, että tilaavan asiakkaan on varattava yksi RCB-instanssi itselleen ja tänä aikana muut asiakkaat eivät voi kyseistä RCB:tä käyttää. Niinpä IED-laitteelle on määritettävä RCB-instansseja sen käyttötarkoitusten mukaan.

Jokainen RCB-instanssi kytketään johonkin muodostettuun datajoukkoon, jota se tarkkailee ja josta viestit generoidaan. Yhteen datajoukkoon voi olla kytkettynä monta RCB-instanssia. Jolloin yhden data attribuutin liipaistessa, jokainen siihen kytketty RCB generoi viestin asiakkaalle.

RCB-luokat sisältävät attribuutteja, joita asiakas konfiguroi ennen tilausta omien tarpeidensa mukaan. Tämän jälkeen asiakas varaa RCB:n kirjoittamalla konfiguroidut arvot ja asettamalla kentän RptEna arvoksi tosi (katso taulukko 5). Tämän jälkeen RCB on varattu kyseiselle asiakkaalle ja IED-laite aloittaa datajoukon attribuuttien tarkkailun. Asiakas jää odottamaan viestien tuloa palvelimelta ilman erillistä kyselyä. Jos konfiguroitu liipaisin liipaisee tapahtuman, RCB lähettää viestin asiakkaalle. Kuvassa 4 on esitetty yllämainittu prosessi asiakkaan ja palvelimen välillä käyttäen puskuroitua BRCB-luokkaa. Kuvassa yhteyden katketessa, palvelin puskuroi viestejä. Yhteyden palautuessa samalta asiakkaalta, palvelin lähettää viestit oikeassa järjestyksessä asiakkaalle. Tilaus lopetetaan asiakkaan pyynnöstä tai yhteyden ollessa poikki tarpeeksi kauan.

2.1.7 Raportointi-luokan määritys ja toiminta

BRCB-luokalla on erilaisia attribuutteja, joita asiakas voi kirjoittaa ja lukea ennen tilauksen aloittamista. Taulukossa 5 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 määrityksestä.



Kuva 4. *Puskuroitu viestien tilausprosessi asiakkaan ja palvelimen välillä.*

Ja standardissa muutenkin kuvataan luokan eri attribuuttien toiminta paljon perusteellisemmin. Tässä työssä riittää että lukija ymmärtää luokan päätoiminnan hyvin. URCB-luokka on melkein samanalainen kuin taulukossa 5 määritetty BRCB-luokka [8, s. 94–103]. Tarkka määritys ja BRCB ja URCB luokkien erot löytyvät standardin osasta 7-2 [8, s. 93–118].

RCB-luokan TrgOps-attribuutti on binääritietue, jossa yksittäinen bitti ilmaisee mikä liipaisin voi aiheuttaa viestin lähettämisen. Asiakas 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*).

Kolme ensimmäistä dchg, qchg ja dupd ovat aikaisemmin määrittettyjen data attribuuttien liipaisimia. Asiakas voi tilata viestejä esimerkiksi vain data 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

Taulukko 5. *BRCB-luokan määritetyt attribuutit ja niiden selitteet.*

Attribuutti	Englanniksi	Selite
BRCBName	BRCB name	Objektin nimi
BRCBRef	BRCB reference	Objektin referenssi
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 referenssi
ConfRev	Configuration revision	Juokseva konfiguraation numerointi, muutos kasvattaa numerointia
OptFlds	Optional fields	Mitä optionaalisia kenttiä viestiin lisätään
BufTm	Buffer time	Puskurointiaika, 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

itse asettaa GI:n arvoksi epätosi. Jatkuva viestintä on viestin lähettäminen asiakkaalle tietyn 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 atribuutissa liipaisin ei ole päällä, ei viestiä generoida ja lähetetä asiakkaalle väliajoin.

Viestien tilaus aloitetaan kun asiakas kirjoittaa RptEna-attribuutin arvoksi tosi. Tilauksen aikana kirjoitus joihinkin RCB-instanssin attribuutteihin muuttuu, verrattuna ennen tilausta. Esimerkiksi yleisen kyselyn tekeminen on mahdollista tilauksen aikana kirjoittamalla GI:n arvoksi tosi. Tilauksen aikana kirjoittamalla TrgOps-attribuutin aiheuttaa puskurin tyhjentämisen. Ja Attribuutin OptFlds kirjoitus aiheuttaa epäonnistuneen vastauksen palvelimelta.

RCB-luokan attribuuttin OptFlds avulla asiakas voi asettaa mitä vaihtoehtoisia kenttiä

viestiin sisällytetään. Attribuutin OptFlds on binääritietue, niin kuin ja TrgOps ja taulukossa 6 on esitetty sen asetettavat arvot [8, s. 98].

***Taulukko 6**. 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 DatSet viestiin
data-reference	Jos tosi, sisällytä datajoukon liipaisseen kohdan rakentamiseen käytetty FCD- tai FCDA-referenssi 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

Kuinka attribuutit vaikuttavat viestin rakenteeseen ja mitä syitä arvon tai arvojen sisältymiseen viestissä voi olla, käsitellään seuraavassa kohdassa.

2.1.8 Viestin rakenne ja kuinka sen sisältö muodostuu

Kuvassa 5 on esitetty standardin määrittämän viestin rakenne ja kuinka optionaaliset kentät vaikuttavat viestin sisältöön [8, s. 104]. Kuvasta voi helposti nähdä mitä kohtia optionaaliset kentät viestiin lisäävät.

Viestin kenttiä SqNun, 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. [8, s. 105–106]

Puskuroidun BRCB-instanssin puskurin täyttyessä viesteistä, esimerkiksi laiterajoitteesta johtuen. Asettaa RCB-instanssi seuraavaan viestiin kentän BufOvlf arvoksi tosi. Tästä kentästä asiakas voi päätellä onko tapahtunut tiedon menetystä. Kenttä sisällytetään viestiin vain jos RCB-instanssin OptFlds-attribuutissa on buffer-overflow bitti asetettu arvoon tosi. [8, s. 106–107]

Tärkein rakenne viestistä on ymmärtää kuinka liipaissut datajoukon alkio viestiin on lisätty. Yksi viesti voi sisältää 1:stä n:ään kappaletta alkioita. Tämä arvo riippuu onko RCB-instanssilla käytössä puskurointiaika BufTm. Tämän ajan sisällä liipaiseet datajoukon alkiot sisällytetään samaan viestiin. Jokainen sisällytetty alkio voi sisältää kentät DataRef

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 referenssi
Jos buffer-overflow = tosi		
BufOvfl	Buffer overflow	Jos arvo on tosi, on bufferoidut viestit 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 referenssi
Value	Value	Sisältää arvon tai arvot liipaisseesta datajoukon alkiosta
Jos reason-for-inclusion = tosi		
ReasonCode	Reason code	Syykoodi miksi tämä datajoukon kohta on sisällytetty viestiin

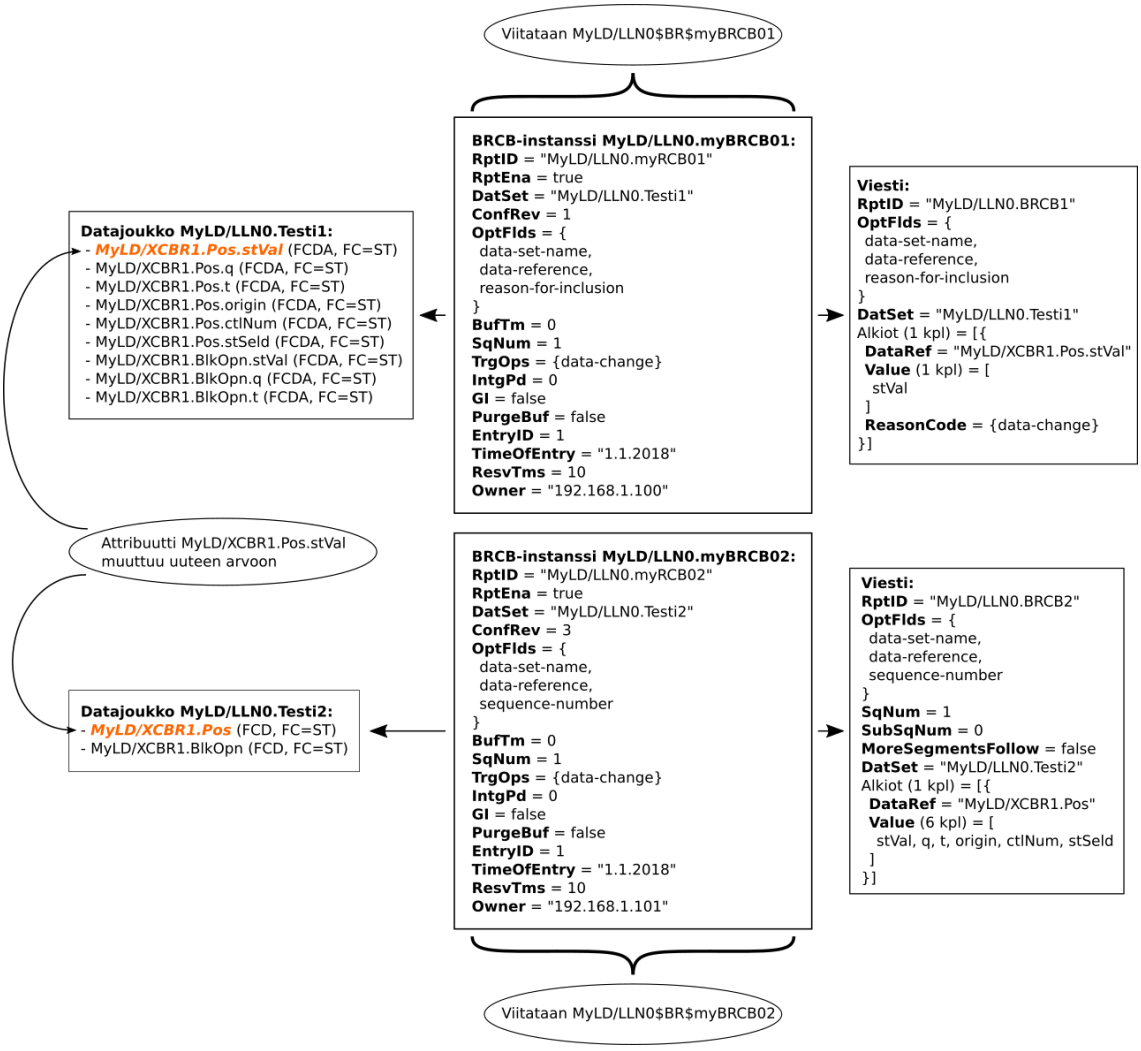
***Kuva 5**. Standardin määrittämä lähetetyn viestin rakenne.*

tai ReasonCode. Jokaiselle alkiolle pakollinen tieto on Value. 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-referenssi kuinka moneen data attribuuttiin. Viittauksen ollessa FCDA-referenssi, joka sisältää vain yhden data-attribuutin. Sisältää Value-kenttä vain kyseisen data attribuutin arvon. Jos viittaus on FCD tai FCDA-referenssi, joka sisältää monta data attribuuttia. Sisältää Value-kenttä kaikki kyseiset arvot referenssin alapuolelta hierarkiassa, vaikka niistä olisi liipaissut vain yksi attribuutti. Tätä tapahtumaa on selitetty tarkemmin alla ja kuvassa 6 on esitetty malli kuinka yksi attribuutin liipaisu aiheuttaa eri viestin generoinnin kahdelta eri RCB-instanssilta. Viesteissä Value-kenttä sisältää eri arvot samasta tapahtumasta. [7, s. 107–108]

Kuvassa 6 on määritetty kaksi datajoukkoa MyLD/LLN0.Testi1 ja MyLD/LLN0.Testi2. Joista Testi1 sisältää 9 eri FCDA-referenssiä funktionaalisella rajoitteella ST. Ja Testi2 sisältää kaksi FCD-referenssiä funktionaalisella rajoitteella ST. Nyt IED-laitteessa attribuutissa MyLD/XCBR1.Pos.stVa tapahtuu arvon muutos uuteen. Tämä liipaisee tapahtuman IED-laitteessa ja se huomaa että attribuutti on viitattuna kahdessa eri datajoukossa. Näitä kahta datajoukkoa viittaa kaksi BRCB-instanssia MyLD/LLN0.myBRCB01 ja MyLD/LLN0.myBRCB02. Joista kummatkin on tilattu (RptEna on tosi) ja TrgOps kenttä sisältää data-change liipaiseminen. Eli tapahtuma liipaisee viestin generoinnin ja lähettämisen asiakkaalle. Kuvassa on esitetty RCB-instanssien sen hetkiset asetetut arvot. RCB-instanssit generoivat tapahtumasta kaksi viestiä, joiden arvot voi myös nähdä kuvassa. Tärkeänä aikaisemmin mainittu viestin Value-kenttä, joka voi sisältää useamman kuin yhden arvon. Instanssilta myBRCB01 tuleva viesti sisältää stVal attribuutin arvon, koska vain se liipaisi kyseisessä datajoukossa tapahtuman ja sillä ei ole muita aliattribuutteja. Instanssilta myBRCB02 tuleva viesti sisältää kaikki Pos-instanssin (Common Data Class -luokka) alla olevat funktionaalisella rajoitteella ST olevat attribuutit, vaikka muutos tapahtui vain stVal attribuuttiin. Eli Value-kenttä sisältää arvot attribuuteille stVal, q, t, origin, ctlNum, stSeld. [8, s. 108] [7, s. 40–44]

Jokaisessa viestin datajoukon alkiossa oleva vaihtoehtoinen kenttä ReasonCode kertoo, miksi alkio on sisällytetty viestiin. Kentä kertoo mikä RCB-instanssin TrgOps-attribuutilla asetetuista liipaisimista liipaisi tapahtuman ja aiheutti alkion sisällytyksen viestiin. Kentän arvot ovat suoraan verrattavissa TrgOps-attribuutin arvoihin. Esimerkin tästä voi nähdä kuvassa 6 myBRCB01-instansilta tulevalta viestiltä, joka sisältää ReasonCode-kentän ja sillä on arvo data-change. Toinen viesti kuvassa ei sisällä ReasonCode-kenttää, koska sitä ei ole laitettu päälle RCB-instanssin OptFlds-attribuutista.[8, s. 28–29]

Kuvassa 6 on esitetty erilainen viittaus RCB-luokkan instansseihin, kuin tähän asti on esitetty. Syynä on mallinnus erilliselle protokollalle, jota käsitellään seuraavassa osiossa tarkemmin.



***Kuva 6**. BRCB-instanssi tarkkailee sille määritettyä datajoukoa ja generoi viestin tapahtuman liipaistessa.*

2.1.9 Abstraktimallin sovitus MMS-protokollaan

Tähän asti käsitellyt IEC 61850 -standardin mallit ja palvelut ovat olleet abstrahoituja. Nyt abstrahoidut mallit ja palvelut voidaan mallintaa tekniikalle erillisillä standardin osalla. Tämän työn kannalta käytetty mallinnus ja tekniikka oli MMS-protokolla (engl. Manufacturing Message Specification). Tästä mallinuksesta on tarkemmin määritetty IEC 61850 -standardin osassa 8-1 [11]. MMS-protokolla on maailmanlaajuinen ISO 9506 -standardi. MMS on viestintä standardi, joka on määritetty toimivaksi TCP/IP:n pinon päällä [18]. Tämän työn kannalta lukijan ei ole tarvitse ymmärtää MMS-protokollaa. Ja ei ihan tarkasti kuinka eri mallit ja palvelut siihen mallinnetaan. Tässä teoriaosuudessa käsitellään työn kannalta tärkeitä tietoja, mitä mallinnuksesta MMS-protokollalle tarvitsee tietää. Kaikkein tärkeintä kuitenkin työn ymmärtämisen kannalta on ymmärtää kaikki edelle esitetty teoria abstrahoiduista malleista ja palveluista, ja niiden toiminnasta. [26]

IEC 61850 -standardin ACSI mallinnuksessa aikaisemmin esitetty instanssien viittaus hierarkiassa muuttuu ja nyt viittaus sisältää myös funktionaalisen rajoitteen. Kuvassa 3 oleva viite "OmaLD/Q0XCBR1.Pos.stVal"funktionaalisella rajoitteella ST, muuttuu muotoo "OmaLD/Q0XCBR1STPos$stVal". Tässä viittauksessa pisteet (.) korvataan dollarimerkillä ($). 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]

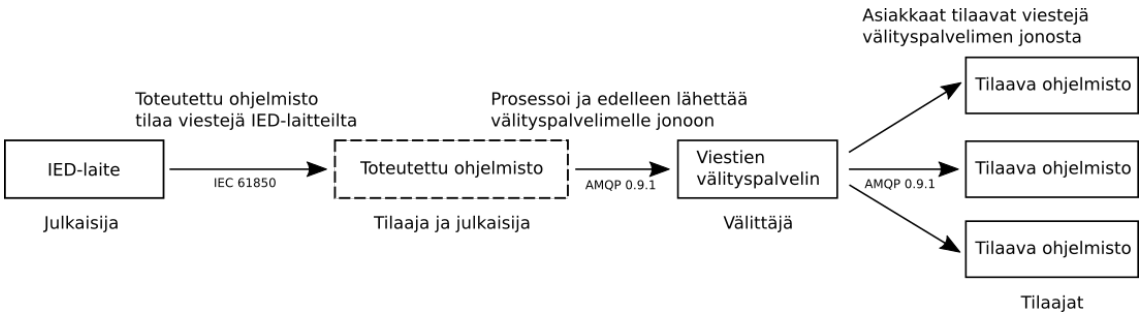
Tämän uuden viittauksen takia jokaiselle viitattavalle kohteelle täytyy olla funktionaalinen rajoite. Niinpä esimerkiksi RCB-luokkien instansseille täytyy olla myös funktionaalinen rajoite. Puskuroitua RCB-instanssia viitataan funktionaalisella rajoitteella BR. Ja puskuroimatonta funktionaalisella rajoitteella RP. Esimerkin tästä viittauksesta voi nähdä aikaisemmin mainitusta kuvasta 6. [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 7 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.

Työssä välittäjänä käytettiin RabbitMQ-ohjelmistoa[1], joka on avoimen lähdekoodin välit-

[1] https://www.rabbitmq.com/



***Kuva 7**. Toteutetun ohjelmiston osuus ja rooli käytettävässä kokonaisuudessa tietoliikenteen kannalta.*

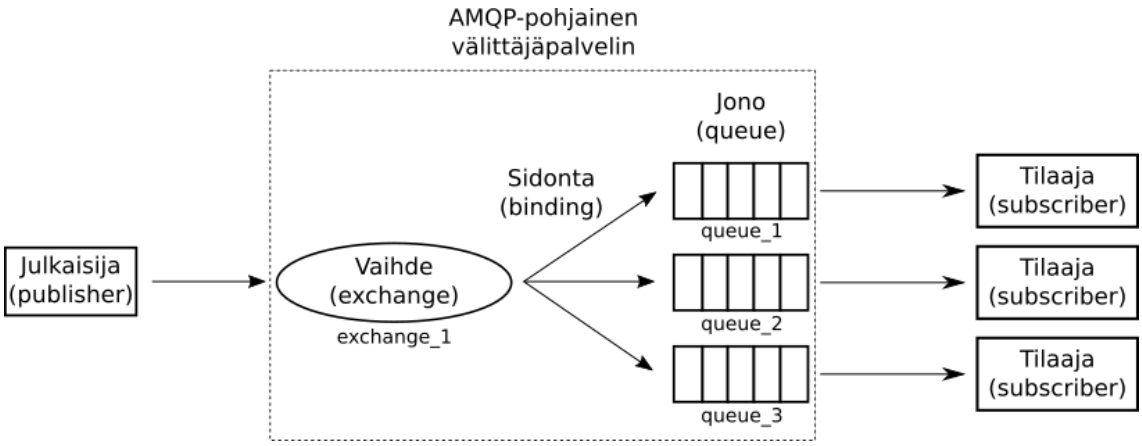
täjäpalvelin ja perustuu avoimeen AMQP-standardiin[2] (engl. Advanced Message Queuing Protocol). AMQP määrittää yhteisen protokollan viestintään eri ohjelmistojen välillä verkon yli välityspalvelimen avulla. Verkon ansiosta välityspalvelin voi sijaita eri koneella kuin sitä käyttävät ohjelmistot. Ajan saatossa standarista on julkaistu monta eri versiota, ja työn tekohetkellä 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 7 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 määrittä komponentteja, joiden läpi viestin täytyy kulkea julkaisijalta tilaajalle. Standardissa nämä komponentit määrittää AMQ-malli (engl. AMQ-model). Kuvassa 8 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 reittitää viestejä tilaajille jonoihin jonon ja vaihteen välisten sidosten mukaan. Jos tilaaja ei kerkeä prosessoida viestejä tarpeeksi nopeasti, palvelin pitää viestit jonossa tilaajelle. Vaihde voi välittää viestin moneen eri jonoon ja yhtä jonoa voi tilata monta eri asiakasta.

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 vaihdeiden ja jonojen välille. Voidaan sanoa että julkaisija ja tilaaja tekevät uusia instansseja AMQ-mallin komponenteista

[2] https://www.amqp.org/



***Kuva 8**. AMQ-mallin osat ja viestin kulku niiden läpi julkaisijalta tilaajalle (pohjautuu kuvaan [2, s. 11]).*

palvelimelle. Vaihteiden ja jonojen instansseilla täyttyy olla välityspalvelimella yksilöivät nimet, jokainen nimi asetetaan instanssin luonnin yhteydessä. Esimerkkinä kuvassa 8 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. Tyypin 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 välitäjä säilyttää vaihteen uudelleenkäynnistysten jälkeen. Jos ei, julkaisijan täytyy määrittää vaihde uudelleen käynnistyksen 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.

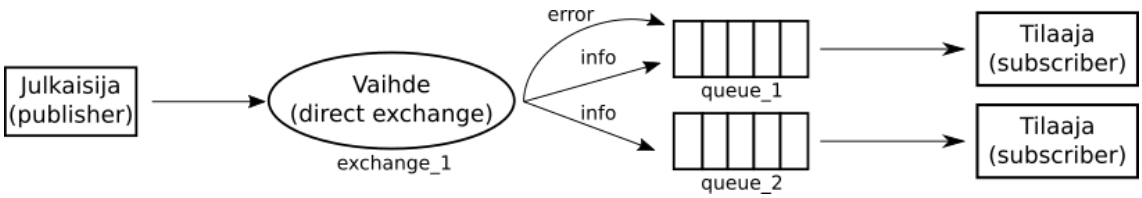
Vaihde reitittää viestejä jonoihin sen sidosten ja tyypin mukaan. Kuitenkin reititykseen liittyy yksi tärkeä asia kuin reititysavain (engl. routing-key). Reititysavain on kuin virtuaalinen osoite viestissä, jonka julkaisija liittää viestiin julkaisun yhteydessä. Tilaaja käyttää myös reititysavainta jonon määrityksen yhteydessä. Vaihde, 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 Suoravaihde (direct exchange)

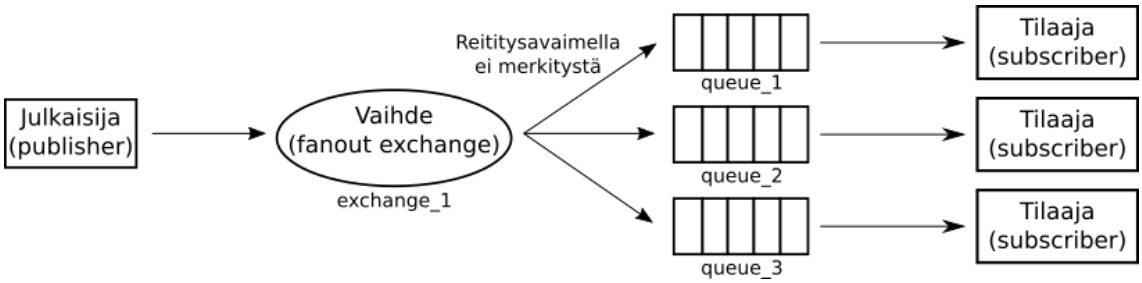
Julkaisija voi määrittää vaihteen instanssin tyypiksi suoravaihteen (engl. direct exchange). Suoravaihde reitittää viestin jonoihoin suoraan vastaavan reititysavaimen perusteella. Suoravaihde reitittää seuraavasti:

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

Kuvassa 9 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 9 julkaisija julkaisee viestin reititysavaimella info. Viesti päätyy molempiin queue_1 ja queue_2 jonoon. Reititysavaimella error, viestit päätyvät vain jonoon queue_1. Välittäjäpalvelin tarjoaa suoravaihteesta oleutusvaihteen nimeltä amq.direct. [2, s. 27]



Kuva 9. *Suoravaihde (engl. direct exchange), reitittää suoraan sidoksen reititysavaimen mukaan (pohjautuu kuvaan [24]).*



Kuva 10. *Hajautusvaihde (engl. fanout exchange), reitittää kaikkiin siihen sidottuihin jonoihin riippumatta reititysavaimesta (pohjautuu kuvaan [1]).*

2.2.4 Hajautusvaihde (fanout exchange)

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

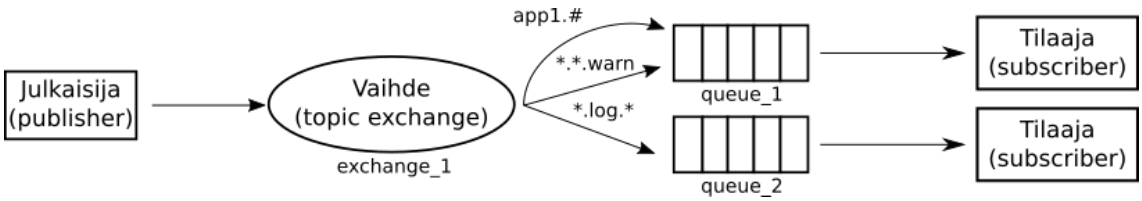
- tilaaja määrittää sidoksen vaihteeseen reititysavaimella K,
- julkaisija julkaisee viestin reititysavaimella R,
- vaihde välittää viestin kaikkiin siihen sidottuihin jonoihin, reititysavaimesta riippumatta.

Kuvassa 10 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]

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,



***Kuva 11**. Aihepiirivaihde (engl. topic exchange), reitittää kaikkiin siihen sidottuihin jonoihin, joiden reitityskaava sopii viestin reititysavaimeen (pohjautuu kuvaan [25]).*

- 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 11 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.

Nyt jos julkaisija lähettää viestin avaimella **app1.debug.warn**. Vaihde 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 lähetään 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 hajautusvaihde. Jos jono sidotaan ilman merkkejä ***** ja **#**, niin se käyttäytyy samalla tavalla kuin suoravaihde. [25]

2.2.6 Otsikkovaihde (headers exchange)

Otsikkovaihde (engl. headers exchange) on vaihdetyyppi joka ei käytä reititysavainta ollenkaan 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,
- vaihe välittää viestin jonoon jos otsikkotiedot O vastaavat otsikkotietoja H, riippuen sidoksen otsikkotiedoissa olevasta **x-match** kentän arvosta.

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

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

Otsikkotiedoissa arvot ovat vaihtoehtoisia asettaa. Jos kentän arvoa ei ole asetettu, vastaavuus 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 määritys ja viestien kuittaaminen

AMQ-mallissa jono (engl. queue) on vaihteen ja tilaajan välissä oleva puskuri (kuva 8), joka tallentaa tilaajalle tulevia viestejä. Jono pitää viestejä jonossa tilaajelle, 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 tilaajan 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 tilaajaja, jono lähettää viestejä monelle tilaajalle kiertovuorottelun (engl. round-robin) periaatteen mukaan. [2, s. 11–12]

Tilaajan täytyy määrittä 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

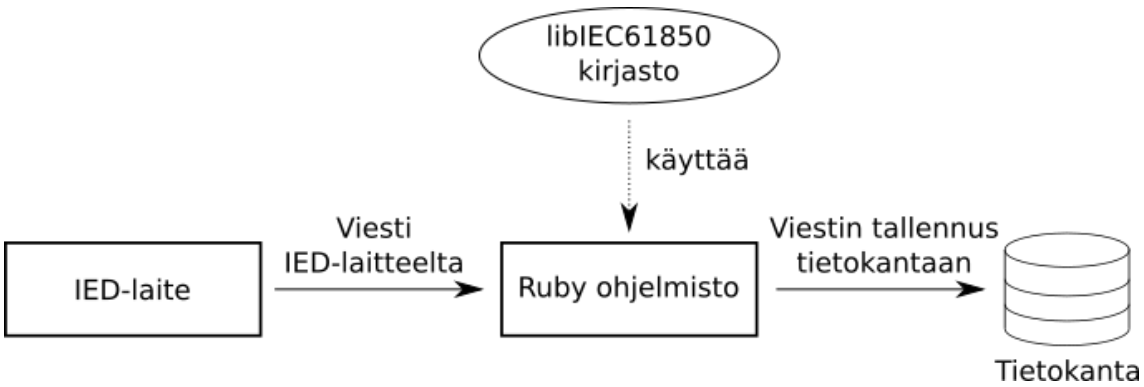
Ennen tämän työn aloittamista yrityksessä oli jo kehitetty ensimmäinen versio ohjelmasta, joka kykeni tilaamaan viestejä IED-laitteelta. Prosessoimaan viestit ja tallentamaan ne relaatiotietokantaan myöhempää käyttöä varten. Tässä ohjelmistossa oli havaittuja ongelmia ja se ei myöskään tukenut kaikkia IEC 61850 -standardin viesteihin liittyviä ominaisuuksia. Tämän ohjelmiston toimintaperiaate ja siinä olleet ongelmat toimivat pohjana uuden version suunnittelulle ja toteutukselle. Tarkoituksena oli poistaa havaitut ongelmakohdat ja miettiä olisiko jokin muu arkkitehtuuri parempi kyseiseen toteutukseen. Ensimmäistä toteutusta ohjelmasta voisi nimittää ensimmäiseksi protoversioksi tai demovaiheeksi (engl. proof of consept), 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 12 on esitetty demoversion arkkitehtuuri korkealla tasolla.

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



Kuva 12. *Rubylla toteutetun demoversion arkkitehtuuri ja tiedonsiirto.*

töä varten. Ruby-ohjelmistossa tärkeässä osassa oli libIEC61850-kirjasto[1]. 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-laittetta 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.

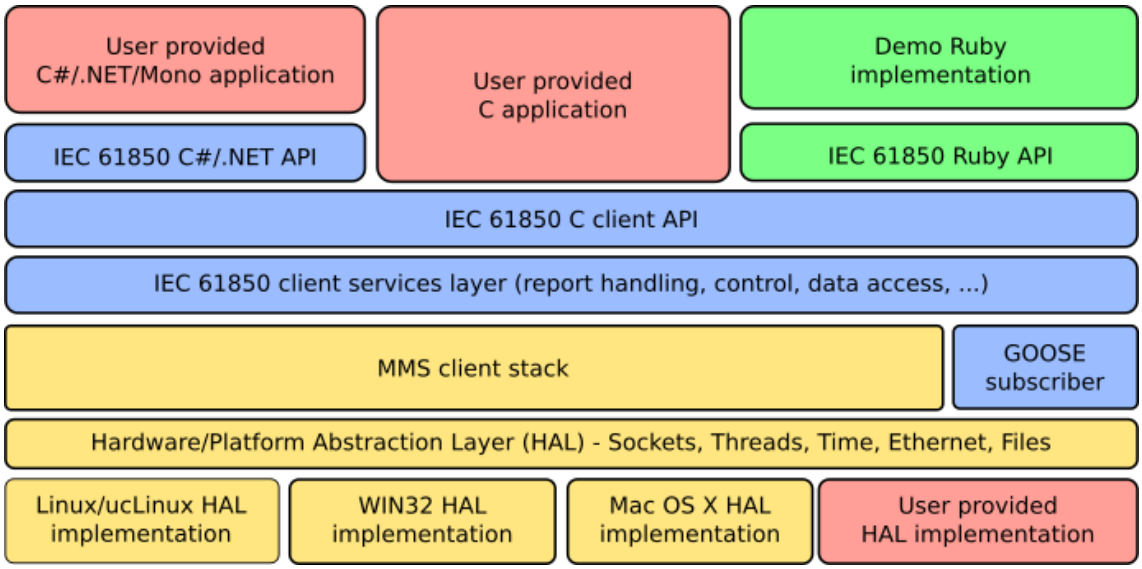
Kirjasto oli rakennettu käyttämään MMS-protokollaa tiedonsiirrossa IED-laitteen ja sen asiakasohjelman välillä, kuten IEC 61850 -standardin osassa 8-1 määritetään. Kuvassa 13 on esitetty kirjaston kerrosarkkitehtuuri asiakasohjelmalle. Kirjastoon oli toteutettu laiteabstraktiokerros (engl. hardware abstraction 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 13 on punaisella merkitty laatikot, 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.

Ruby-koodista C-kielen funktioiden kutsuminen ei ole suoraan mahdollista, vaan kielten väliin täytyy toteuttaa liitos. Demoversiossa liitos oli tehty käyttäen Rubylle saatavaa ruby-ffi -kirjastoa[2] (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 kehyksen 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

[1] http://libiec61850.com
[2] https://github.com/ffi/ffi

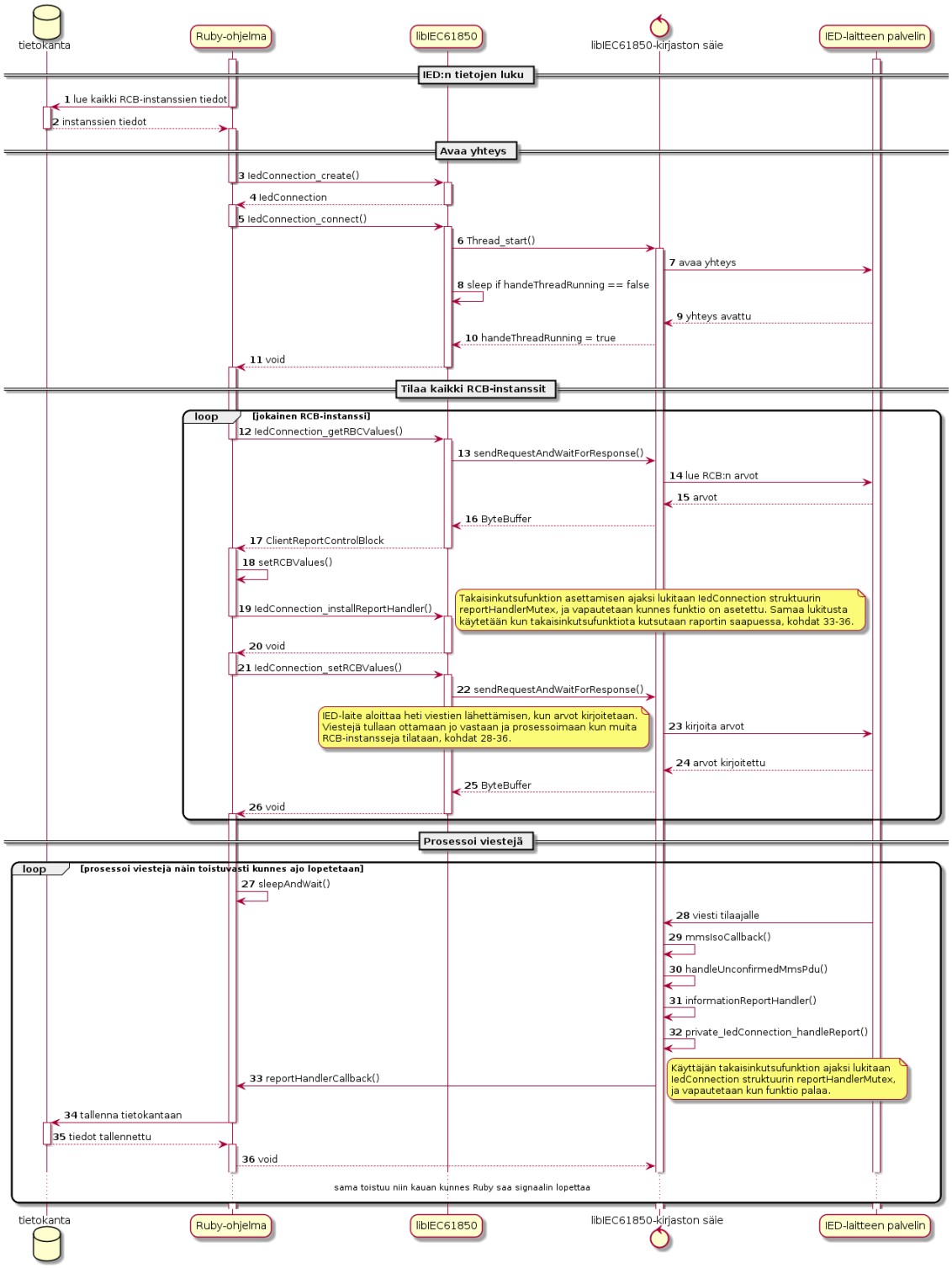


Kuva 13. *libIEC61850-kirjaston kerrosarkkitehruurin komponentit, vihreällä Ruby toteutukseen lisätyt osat (pohjautuu kuvaan [15]).*

täytyi ladata Railsin ajoympäristö muistiin, joka aiheutti sen että yksinkertaisen ohjelman täyti 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 14. Kuvassa ohjelman kaksi eri silmukkaa on esitetty kahdella eri loop-laatikolla. Sekvenssikaaviossa osallisena ovat tietokanta, Ruby-ohjelma, libIEC61850-kirjasto, libIEC61850-kirjaston natiivisäie ja IED-laiteen 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 yhteyten 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 natiivisäikeen 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 yhteyden aika ylittyy. Kirjaston funktio, joka tämän hoitaa on `sendRequestAndWaitForResponse()`

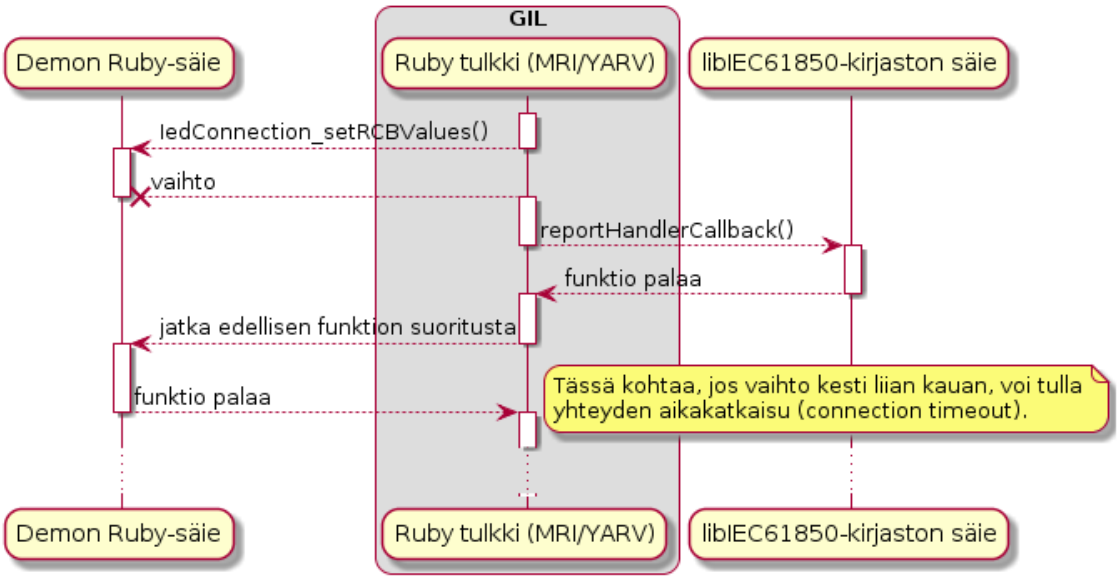


Kuva 14. *Sekvenssikaavio kaikkien RCB-instanssien tilaukseen ja niiden viestien tallentamiseen yhdeltä IED-laitteelta Ruby-ohjelmalla.*

ja on määritetty src/mms/iso_mms/client/mms_client_connection.c riveillä 345–418 [21]. RCB-arvot luettuaan, kirjasto palauttaa struktuurin `ClientReportControlBlock`, joka sisältää luetut tiedot RCB-instanssista. Samaa struktuuria käytetään arvojen muuttamiseen 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 suorituskyky 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 suorituskykyyn oli syynä muutama asiaa. Yksi niistä oli Ruby-kielen huonompi suorituskyky 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 15 on esitetty kuinka Ruby-tulkki vuorottaa kahta ajossa olevaa säiettä. Kuvassa Demon Ruby koodi kutsuu `IedConnection_setRCBValues()`



Kuva 15. *Ruby-tulkin globaalin lukituksen toiminta, joka vuorottaa ajossa olevia säikeitä.*

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 (kuvassa 14 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.

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 Rubysta 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. Lisäki 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

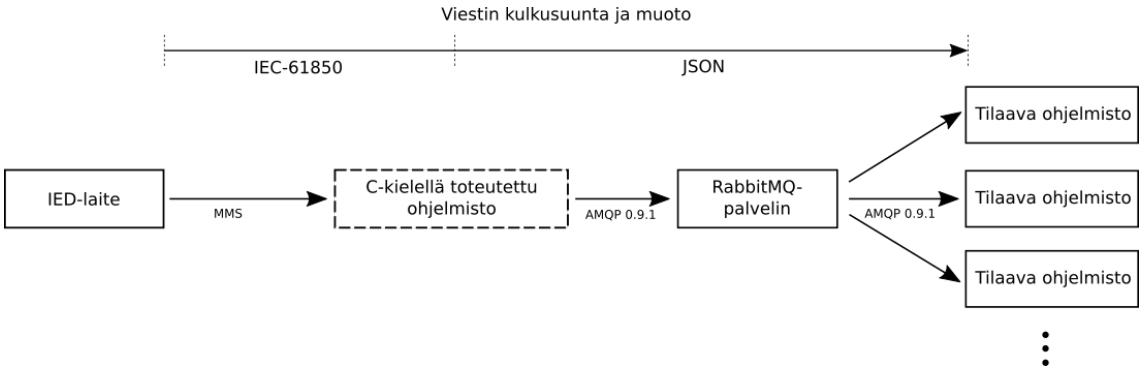
Pitäisikö tähän kirjoittaa ohjelman ajosta ja siihen liittää sekvenssikaavio perustoimminnasta? Kirjoita jos tuntuu että tarvetta.

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. Ensin suunnitellusta ohjelmasta annetaan kattava kokonaiskuva lukijalle ja tämän jälkeen tulevissa kappaleissa mennään jokaisen kohdan yksityiskohtiin tarkemmin.

4.1 Kokonaiskuva

Aikaisemmin kappaleessa 3.1 kuvassa 12 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 päätyttiin kuvassa 16 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.

Suunnitellussa 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 julkai-



***Kuva 16**. Suunnitellun järjestelmän toiminta ja viestin kulkeminen ja muoto eri osapuolten välillä.*

see AMPQ-standardin pohjaiselle jonopalvelimelle JSON-muodossa (engl. JavaScrip Object Notation). Lopullisessa toteutuksessa jonopalvelimena käytettiin RabbitMQ-nimistä ohjelmistoa, joka pohjautuu AMPQ-standrdin 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 määrityksen mukaan yksi RCB-instanssi voi olla vain tilattuna yhdellä asiakkaalle 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 pystyisi julkaisemaan viestit eteenpäin, välittämättä siitä kuka viestejä vastaanottaa.

Ratkaisuna yllä mainittuihin vaatimuksiin oli sijoittaa IED-laitteen ja muiden tilaavien ohjelmien väliin väliohjelmisto, kuten kuvassa 16 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 käyttöliittmääkään. Ohjelmasta päätettiin tehdä komentorivipohjainen toteutus, jolle kaikki tiedot voitaisiin syöttää komentorivillä parametereillä 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-instassien 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 JRuby[1] nimisellä Ruby-tulkilla. Tavoitteena saada demoversion toteutus toimimaan ilman GIL:iä ja säikeet suoritukseen rinnakkain. JRuby on Ruby-koodin tulkki, joka suorittaa Ruby-lähdekoodia 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. Kysymksenä tämän aikana tuli ajan käyttö ja fakta että demosta olisi pitänyt korjata ja paikata monta asiaa. Päätyttiin 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 käyt-

[1] http://jruby.org/

tämiseen ja lisäksi JSON rakenteen muodostamiseen. Hyötynä vielä C-kielen valinnasta 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

Kirjoita tähän mihin muotoon viestit lopussa tallennetaan esim. JSON. Miksi tähän valintaan päädyttiin. Kerro myös kuinka raportin alkuperäistä rakennetta muokattiin uuteen muotoon sopivaksi. Lisäksi jos liitteisiin liittää kopion lopullisesta JSON-rakenteesta miltä se näyttää. Käsittele tässä kaikkia viestin kenttiä mitä mukana tulee, samalla peilaa RCB-instanssia mistä ne tulee ja miten ne uudelleen järjestettiin lopulliseen JSON-rakenteeseen.

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 ihminen helposti lukea ja nykypäivänä on paljon käytetty tiedonsiirto muoto erilaisissa web-palveluissa ja rajapinnoissa. Myöskin JSON-rakenteiden lukemiseen on monelle eri kielellä olemassa valmiita kirjastoja sen monikäyttöisyyden takia [22].

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-kielellä.

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, syy. 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