TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MAURI MUSTONEN

SÄHKÖASEMAN ÄLYKKÄÄN ELEKTRONIIKKALAITTEEN VIESTIEN TILAUS JA PROSESSOINTI

Diplomityö

Tarkastaja: Professori Kari Systä

Jätetty tarkastettavaksi 17. toukokuuta 2018

TIIVISTELMÄ

MAURI MUSTONEN: sähköaseman älykkään elektroniikkalaitteen viestien tilaus ja prosessointi
Tampereen teknillinen yliopisto
Diplomityö, 14 sivua
Toukokuu 2018
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Professori Kari Systä

Avainsanat: Ohjelmistotuotanto, IEC 61850, MMS, AMQP

Kirjoita yleiskataus tiivistelmä työstä tähän. Kerro lyhyesti mitä työssä tullaan tekemään.

ALKUSANAT

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

Tampereella, 19.4.2018

Mauri Mustonen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

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*, konfiguroitava sähköaseman älykäs elektroninen laite, joka toteuttaa aseman toiminnallisuutta
MMS	engl. *Manufacturing Message Specification*
RCB	engl. *Report Control Block*, raporttien konfigurointiin ja tilaukseen tarkoitettu lohko asiakasohjelmalle

1. JOHDANTO

Kirjoita tähän johdantoa työstä ja aiheesta. Kuinka työ valittiin ja miksi tekijä valitsi tämän työn. Kirjoita myös mitä tehtiin. Kokonaiskuva työstä pitäisi saada johdannosta.

2. TEORIA

Tähän kohtaan kirjoittaa teoriaa mitä tarvitaan työn toteutuksen ymmärtämisen kannalta. Kaikki työssä tarvittava teoria kuvataan tämän otsikon alla.

2.1 MMS-protokolla

Selitä lyhyesti mikä on MMS-protokolla ja vähän sen tietotyypeistä. Tämän tarkoitus on pohjustaa tulevaa IEC 61850 abstraktien olioiden (ACSI) sovitusta tämän protokollan päälle.

2.2 IEC 61850 -standardi

Kirjoitta yleisesti mikä on IEC 61850 -standardi ja mitä varten se on olemassa. Kerro myös kuinka standardi on pilkottu pienempiin dokumentteihin ja mitä kukin käsittelee.

2.2.1 Standardin määrittämä abstraktimalli

Kirjoita tähän mitä standardin IEC 61850-7-2 osuudessa määritellään abstraktoimalla fyysisiä laitteita ja palveluita rajapinnoiksi ja olioiksi. Käsittelee standardin Abstract communication service interface (ACSI).

2.2.2 Viestiblokin konfigurointi ja tilaus

Kirjoita tähän IEC 61850 -standardin määrittästä abstraktista raportointimallista. Tätä raportointi mekanismia tullaan käyttämään raporttien tilauksessa ja sen konfigurointi täytyy ymmärtää toteuttettavan ohjelmiston kannalta.

2.2.3 Viestin rakenne

Kirjoita tähän standardin määritämästä viestin rakenteesta ja mitä tietoa se sisältää. Kerro myös sen vaihtoehtoisista kentistä.

2.2.4 Abstraktimallin sovitus MMS-protokollaan

Kirjoita kuinka ylempi ACSI sovitetaan MMS-protokollan palveluiksi ja tietotyypeiksi standardin IEC 61850-8-1 osuuden mukaan. Tähän myös miten raportointi toimii MMS-protokollan päällä.

2.3 Advanced Message Queuing Protocol

Kirjoita tähän AMQP määrittävästä standardista, mikä sen tarkoitus on ja mihin sitä voidaan käyttää.

2.3.1 Viestien välitysmekanismit

Mitä mekanismeja AMQP tarjoaa viestien välittämiseen osapuolille. Näitä on jono, reititys suoraan osapuolien välillä ja viestin julkaisu ja tilaaminen.

2.3.2 Tilaus ja julkaisu -mallin osat

Kirjoita tähän AMQP tarjoamista viestien julkaisu ja tilaus -mallin osista osapuolten kesken. Kerro mitä eri osat tekevät ja mikä niiden tehtävä viestien välittämisessä on. Englanniksi osia ovat esim. exchange, queue, publisher ja consumer.

3. ALKUTILANNE

Pohjusta miksi suunniteltava ohjelmisto tarvitaan toteuttaa yritykseen johon työn teen. Alustava suunniteltu ohjelmiston toteutus olisi tilata IEC 61850 -standardin määrittämiä raportteja ja muokata ne uuteen muotoon ja julkaista ne eteenpäin tilaavalle ohjelmalle käyttäen AMQP-standardin määrittämää viestintää. Jonon tilaava asiakasohjelmisto voi olla mikä tahansa muu ohjelmisto. Viestien lopullinen muoto voisi olla JSON.

Nykyisin sähköasemilla älykkäillä elektronisilla laitteilla (engl. *Intelligent Electronic Devices*, **IED**) asemilla voidaan toteuttaa tuhansia eri datapisteitä, jotka kuvaavat aseman toiminnallisuutta ja konfiguroitavuutta. Tämän konfiguroitavuuden ansiosta IED:tä voidaan asemalla käyttää erilaisina sähkölaitteina, kuten sulakkeina tai muuntajina. IEC 61850 -standardin abstraktit datamallit määrittävät IED:n datapisteiden rakenteet, muodot ja tyypit. Standardin mukaan erillisistä datapisteistä voidaan muodostaa haluttuja datajoukkoja (engl. *data set*). Datajoukkot ovat helppo tapa kuvata halutut tai tärkeät datapisteet yhdeksi yhteinäiseksi joukoksi. [2].

Asiakas-palvelin-arkkitehtuurissa asiakkaan on mahdollista tilata datajoukkojen sen hetkisiä arvoja IEC 61850 -standardin määrittäminä raportteina konfiguroitavilla parametreilla, jotka konfiguroidaan ennen tilausta. Arkkitehtuurissa asiakas tekee tilauksen palvelimelle (tässä tapauksessa IED), jonka jälkeen palvelin lähettää raportteja asiakkaalle automaattisesti, jonkin asiakkaan konfiguroiman ehdon täyttyessä. Standardi määrittää kuinka raporteja voidaan esim. välittää TCP/IP-protokollan avulla. Yksi raporti sisältää mm. tietojoukon sen hetkisiä arvoja ja syyn raportin lähetykseen (esim. arvon muuttuminen). Palvelin ylläpitää tilausta kunnes asiakas lopettaa tilauksen tai yhteys osapuolten välillä katkeaa. Asiakas tilaa raportit konfiguroimalla palvelimella olevan erillisen raportointilohkon (engl. *Report Control Block*, **RCB**). Lohkolla voi konfiguroida mm. raporttien sisältämiä vaihtoehtoisia kenttiä ja erilaisia liipaisimia raporttien generointiin. Standardi määrittää että yhdellä RCB:llä voi olla vai yksi tietojoukko ja yksi tietojoukko voi olla viitattuna monessa eri RCB:ssä. Yhdessä IED:ssä voi olla määrittetynä monta RCB:tä. Yhtä tilaavaa asiakasta kohden on yksi RCB instanssi. [1, s. 91–130].

Tulevissa kappaleissa pohjustetaan työn alussa olemassa olevan ohjelmiston arkkitehtuuria, mitkä olivat sen komponentit ja niiden toiminnallisuus. Tämän jälkeen pohditaan toteutuksen ongelmia, ja mitä työssä pyritään ratkaisemaan uudella toteutuksella. Asetettujen tutkimuskysymysten ja ongelmien kautta pyritään löytämään uudelle ohjelmiston arkkitehtuurille pohjaa ja ratkaisua siihen liittyviin päätöksiin.

3.1 Kokonaiskuva

Kirjoita tähän osioon kokonaiskuva alkutilanteesta missä oltiin ennen työn aloittamista. Selvennä kuvilla alkutilanteen arkkitehtuuria.

Työn aloitusvaiheessa oli jo toteutettuna ohjelmisto raporttien tilaukseen ja käsittelyyn. Tämä toteutus oli puutteellinen, ei helposti skaalautuva, ja huono suorituskyvyltään. Alkuperäinen ohjelmisto oli lähellä enemmän ensimmäistä prototyyppiä ennen todellista toteutusta. Työn tarkoituksena oli suunnitella ja toteuttaa uusi toteutus, joka ratkaisisi entisen ongelmakohdat.

Alkuperäisessä toteutuksessa asiakasohjelmisto oli toteutettu Ruby-ohjelmointikielellä. IEC 61850 -standardin määrittämien palveluiden ja tietorakenteiden toteutukseen käytettiin avoimen lähdekoodin libIEC61850-kirjastoa[1]. Kirjasto on ohjelmoitu C-kielellä ja sen avulla voidaan toteuttaa kumpikin palvelin- ja asiakasohjelmisto. Tässä toteutuksessa tarvittiin vain asiakasohjelman osuutta Ruby-osuuden toteutukseen. Kirjasto abstraktoi standardin määrittämiä palveluita ja tietorakenteita ohjelmoijalle helpoiksi funktioiksi ja C-kielen rakenteiksi. Normaalisti C-koodin funktioiden kutsuminen Rubysta suoraan ei ole mahdollista ilman erillistä liitosta. Seurauksena Rubyyn oli tehty laajennos libIEC61850-kirjastoon käyttäen Rubylle saatavaa ruby-ffi -kirjastoa[2] (engl. *Foreign Function Interface*, **FFI**). Liitoksen avulla libIEC61850-kirjasto voi hoitaa standardin vaatiman matalan tason toiminnan ja Ruby-ohjelmisto voi keskittyä vaadittuun toiminnallisuuteen.

Kirjasto toteuttaa raporttien vastaanoton palvelimelta erillisellä säikeellä. Säie käynnistetään kun asiakasohjelma asettaa funktion takaisinkutsuntaa varten raportin saapuessa ja aloittaa tilauksen. Asetettua funktiota kutsutaan asynkronisesti erillisestä säikeestä raportin saapuessa asiakkaalle. Takaisinkutsun suorituksen jälkeen, suoritus palaa takaisin säikeeseen.

3.2 Ratkaistavat ongelmat

Kirjoita tähän mitä ongelmia edellisen toteutuksen kanssa on ja mitä yritään ratkoa. Mainitse suorituskykyongelmista Rubylla ja libiec61850-kirjastoa käyttäen.

Työn alussa olevan ohjelmiston ongelmia oli mm. ei helposti skaalautuvuus, huono suorityskyky raporttien määrän ollessa suuri, eikä ohjelmisto tukenut kaikkia standardin määrittämiä toiminnallisuuksia. Ohjelmistoa voisi enemmän pitää ensimmäisen toteuksen prototyyppinä. Ohjelman suoritusalustana käytettiin Linuxia.

Ohjelmisto pystyi tilaamaan ja vastaanottamaan raportteja yhdeltä IED:ltä ja siinä monelta määritellyltä RCB:ltä. Ohjelmisto prosessoi ja tallensi raportteja tietokantaan muuta

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

käyttöä varten. Tilanteessa, jossa raportteja tilaavassa järjestelmässä on monta osaa, jotka kaikki tarvitsevat raporttien tietoja reaaliajassa. Joutuvat eri osat tässä tilanteessa kyselemään tietoja tietokannasta, ilman erillistä tietoa niiden saapumisesta. Tämä aiheuttaa turhaa kuormaa tietokannalle ja tietojen saaminen reaaliajassa ei ole mahdollista. Myöskin jos komponentti tarvitsee tietyn tyypin raportteja, ei kaikkea tietoa, ongelma on sama.

Ohjelmiston suorituskyky paikoin raporttien määrän ollessa suuri aiheutti ongelmia. Syynä Rubyn toteutuksessa oli oletustulkissa (*CRuby*) oleva globaali lukitus (engl. *Global Interpreter Lock*, **GIL**). Vaikka Rubyn säie on oma käyttöjärjestelmän tarjoama säie, GIL estää säikeiden yhtäaikaisen suorituksen ja vain yksi säie on suorituksessa kerrallaan [3, s. 131–133]. Linux-pohjaisella käyttöjärjestelmällä libIEC61850-kirjaston laitteistoabstraktiokerros (engl. *Hardware Abstraction Layer*, **HAL**) käyttää POSIX-säikeitä [4]. Linux-käyttöjärjestelmän säikeet ovat suorituksessa yhtä aikaa ja moniytimisellä prosessoreilla asioita tapahtuu samalla ajan hetkellä. Nyt raportin saapuessa, C-prosessin säikeen suoritus kutsuu takaisinkutsuntaan asetettua funktiota, joka on implementoitu Rubyn puolella. On funktion suoritus GILin alaista suoritusta. Ruby-prosessin myös suorittaessa muuta toimintaa takaisinkutsujen välissä, on Rubyn suorituskyky ohjelmiston pullonkaulana raporttien määrän ollessa tiheää.

Kirjoita tähän vielä ongelmasta kun tilataan monta RCB:tä. Raporttien tullessa Rubyn puolelle, ei Rubyn muu koodi saa tilattua loppuja RCB:tä kirjaston lukitusten takia. Ja yhteys aikakatkeaa tämän takia. Selitä lukituksista tarkemmin ja myös liitä pätkiä libIEC61850-kirjaston koodista. Syyn selityksen voi siirtää muualle. Kirjoittaa vain että on ongelma, ja selvitys miksi, muualla.

3.3 Tutkimuskysymykset

Esitä tässä työlle asetettuja tutkimuskysymyksiä. Näitä voisi olla esim. seuraavat:

- *Mikä on syynä huonoon suorituskykyyn alkutilanteen toteutuksella?*
- *Kuinka suorituskyky paremmaksi verrattuna nykyiseen toteutukseen?*
- *Mitkä ohjelmiston arkkitehtuurin suunnittelumallit (design patterns) olisivat sopivia tämän kaltaisen ongelman ratkaisemiseen? Mitä niistä pitäisi käyttää ja mitä ei?*
- *Mikä olisi sopiva lopullisen prosessoidun tiedon muoto?*
- *Kuinka järjestelmä hajautetaan niin että tiedon siirto eri osapuolten välillä on mahdollista ja joustavaa (push vs pull, message queue jne.)?*

4. SUUNNITTELU

Kirjoittaa tähän kuinka toteutettava arkkitehtuuri suunniteltiin ja kuinka päätöksiin päädytiin. Kirjoitusta myös miten tilattuja raportteja käsitellään ja kuinka niitä julkaistaan eteenpäin. Tarkoituksena olisi saada raportit nykyaikaiseen JSON muotoon.

4.1 Suorituskyvyn parantaminen

Miksi entisen toteutuksen suorituskyky on huono ja mitä voitaisiin tehdä sen parantamiseksi. Kirjoita vaikutuksista tähän ja mihin tuloksiin päädyttiin.

4.2 Järjestelmän hajautus

Lähde erilaisista hajautukista (pull vs pull, message queue) ja päätä mikä sopii tähän toteutukseen parhaiten ja miksi.

4.3 Ohjelmiston parametrisointi

Kirjoita mitä asiakasohjelman pitää tehdä jotta raportit saadaan tilattua ja mitä parametrejä ohjelmisto tarvitsee toimiakseen. Kuinka käyttäjä kontrolloi ohjelman eri asetuksia?

4.4 Arkkitehtuurin suunnittelu

Määritä ohjelman tarkempaa arkkitehtuuria mitä voidaan käyttää asetettujen ja yllämainittujen asioiden saavuttamiseen ja tarkentamiseen. Mitä jos käyttäjä tilaa monta viestiblokkia, niin missä järjestykssä asiat tehdään jne.

4.5 Prosessoidun viestin muoto

Kirjoita tähän mihin muotoon viestit lopussa tallennetaan esim. JSON. Miksi tähän valintaan päädyttiin. Kerro myös kuinka raportin alkuperäistä rakennetta muokattiin uuteen muotoon sopivaksi.

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?

Kommentteja työtä aloittaessa:

- *Olisiko hyvä, että lähdet työssäsi erilaisista hajautus paradigmoista (push vs pull; message queue), perustelet valintasi ja sitten menet suunnitteluun ja toteutukseen?*
- *Ja olisi hyvä, että työ perustelee miksi tuota MQ arkkitehtuuria yleensä (ja rabbitMQ:ta) käytetään.*

Things to do now:

- *Laittaa aihe hyväksyntään.*
- *Lähde kirjoittamaan teoriaa ja ennen sitä yleistä tasoa missä ollaan. Yleinen korkea taso sen takia, että lukija ymmärtää mistä edes on kyse. Pidä koko ajan kirjoittaessa mielessä top-down lähestymistapa! Erittäin tärkeä!!!*
- *Loppu otsikoida niin että ensin on tulokset, niiden arviointi ja yhteenveto mainitussa järjestyksessä.*
- *Kirjoittaessa miettiä asioita mistä kirjoitetaan ja pitää kontekstista kiinni.*
- *Pidä lauseet simppelineinä ja helppolukuisina! Älä turhaan vaikeuta hommaa lukijalle ja se ei tuo työhön yhtään mitään lisäarvoa! Todella tärkeä asia ajatella! Jos lause käsittää monta asiaa, pilko se pienempiin erillisiin lauseisiin.*
- *Muihinkin lähteisiin voi viitata kuin tieteellisiin. Toki yritä löytää tieteellisiä julkaisuja mahdollisuuksien mukaan. Osoittaa että olet perehtynyt asiaan paremmin.*
- *Kun kirjoitat asiaa esim. että entisessä ohjelmassa oli ongelma että ei skaalaudu helposti tai on huono suorityskyky. Kerro mistä johtopäätös tulee. Tämä ei ole lukijalle selvää tietoa.*
- *Teorien ja yleisen osuuden kirjoittamisen jälkeen, sovi palaveri Karin kanssa.*
- *Työn otsikko on hyvä, ei tarvitse olla erikseen "ohjelmallisesti"sanaa.*
- *Työn päätason otsikoita laittaa enemmän kuvaavimmiksi kuin "Alkutilanne"ja "Teoria".*
- *Käytä työssä viesti sanaa raportin sijaan. Tuo lukijalle esille että se tarkoittaa standardin mukaisia raportteja.*

LÄHTEET

[1] 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, Aug. 2010, 213 p. Saatavissa (viitattu 16.5.2018): https://webstore.iec.ch/publication/6015

[2] R.E. Mackiewicz, Overview of IEC 61850 and Benefits, 2006. Saatavissa (viitattu 14.5.2018): https://ieeexplore.ieee.org/document/4075831

[3] R. Odaira, J.G. Castanos, H. Tomari, Eliminating Global Interpreter Locks in Ruby Through Hardware Transactional Memory, SIGPLAN Not., Vol. 49, Iss. 8, Feb. 2014, pp. 131–142. Saatavissa (viitattu 16.5.2018): http://doi.acm.org/10.1145/2692916.2555247

[4] 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