TAMPEREEN TEKNILLINEN YLIOPISTO

PAAVO PITKÄNEN

ALUSTA BIOSIGNAALIEN REAALIAIKAISEEN KÄSITTELYYN MONITOROINNIN YHTEYDESSÄ

Diplomityö

Tarkastajat: professori Tarmo Lipping
sair.fyys. Jukka Kinnunen
Tarkastajat ja aihe hyväksytty tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 7. syyskuuta 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
PITKÄNEN, PAAVO: Alusta biosignaalien reaaliaikaiseen käsittelyyn monitoroinnin yhteydessä
Diplomityö, 96 sivua, 17 liitesivua
Toukokuu 2012
Pääaine: Ohjelmistotekniikka
Tarkastajat: professori Tarmo Lipping, sairaalafyysikko Jukka Kinnunen
Avainsanat: biosignaalit, aivosähkökäyrä, EEG, reaaliaikaisuus, reaaliaikainen käsittely, suodatus, MATLAB, puskurointi, monisäikeisyys, TCP/IP

Lääketieteellisen tutkimuksen yhteydessä tehtävissä aivosähkökäyrän (EEG) rekisteröinneissä voi ilmetä tilanne, jossa käytössä oleva mittausjärjestelmä ei ominaisuuksiltaan kykene vastaamaan rekisteröinnin vaatimuksiin (esim. mittaustiedon välitys lähiverkkoon tai tiettyjen trendien laskenta). Tässä työssä toteutetaan sovellus, joka kykenee aivosähkökäyrän reaaliaikaiseen vastaanottoon, käsittelyyn ja lähetykseen. Mittaustietoa vastaanotetaan Mega Elektroniikka Oy:n toteuttamalta NeurOne-järjestelmältä ja lähetetään palvelimelle, jonka toteutuksesta vastaa Girf Oü. Tietoa voidaan lisäksi käsitellä .NET- tai MATLAB-tekniikoihin pohjautuvilla algoritmeilla.

Työ jakaantuu kolmeen osaan: aluksi tutustutaan aivosähkökäyrään ja sen käyttökohteisiin teho-osastolla sekä luodaan katsaus saatavilla oleviin mittausratkaisuihin. Toisessa osassa kootaan sovellukselle asetetut vaatimukset ja käsitellään työssä toteutettavan sovelluksen haasteita reaaliaikaisuuden sekä tiedon käsittelyn ja lähetyksen kannalta. Lopuksi kuvataan sovelluksen oleellisimmat komponentit ja näiden toiminta sekä sovelluksen testauksesta saadut tulokset.

Työn tuloksena syntynyt sovellus toimii EEG-mittauslaitteiston rinnalla ja kykenee sekä käsittelemään että toimittamaan mitattuja signaaleja lähiverkon yli reaaliajassa. Se on modulaarisen rakenteensa vuoksi helposti kolmannen osapuolen laajennettavissa. Sovellus ei ole sidoksissa pelkästään NeurOne-järjestelmään ja EEG-signaaleihin, vaan voidaan liittää helposti myös muihin mittausjärjestelmiin. Lisäksi se kykenee käsittelemään myös muita yksiulotteisia biosignaaleja (mm. EKG, EMG) tai esimerkiksi audiota. Toteutettua sovellusta ei ole lääketieteellisesti varmennettu eikä se täten sovellu tilanteisiin, joissa täytyy tehdä sen tuloksiin pohjautuvia hoitoratkaisuja.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
PITKÄNEN, PAAVO: A platform for realtime processing of biosignals for monitoring purposes
Master of Science Thesis, 96 pages, 17 Appendix pages
May 2012
Major: Software Engineering
Examiners: professor Tarmo Lipping, hospital physicist Jukka Kinnunen
Keywords: biosignals, electroencephalogram, EEG, realtime, realtime processing, filtering, MATLAB, buffering, multithreading, TCP/IP

When EEG recordings are carried out during medical research, a situation may arise in which the utilized equipment doesn't fulfill the requirements of the recording scenario (e.g. relaying the measurement data to local area network or calculating certain trends). This thesis is about the implementation of an application that is capable of receiving, processing and transmitting EEG measurement data in real time. The data is received from NeurOne measurement system developed by Mega Electronics Ltd and sent to a server provided by Girf Oü. In addition the signals may undergo processing with .NET or MATLAB based algorithms at the measurement site.

The thesis comprises three sections: first EEG and its use in the intensive care unit (ICU) are introduced. This section also covers a brief review of EEG measurement systems available from different manufacturers. The second section focuses on gathering application requirements and investigating the challenges in real time data processing and transfer. The final section covers the actual implementation describing the primary software components of the application and evaluation via approval and performance testing.

The developed application functions in parallel with EEG measurement system and is capable of both real time processing and transfer of measured data. It's modular architecture makes it easily extendable by third parties. Use of the application is not limited to the NeurOne system or EEG signals, for it can be interfaced to measurement solutions provided by other manufacturers. It is also capable of processing other one-dimensional data such as other biosignals (e.g. ECG or EMG) as well as audio. The application hasn't been medically certified, thus data handled by it should not be used in decision making regarding patient treatment.

ALKUSANAT

Tullessani vuonna 2008 Poriin opiskelemaan valitsin pääaineekseni ohjelmistotekniikan ja sivuaineekseni signaalinkäsittelyn. En tuolloin osannut aavistaakaan kuinka mielenkiintoista molemmat aineet kattavaa diplomityötä pääsen tekemään. Työ ulottui edellä mainittujen lisäksi myös lääketieteen ja tietoliikenteen alueille, joten se oli aihealueiltaan todella laaja. Näistä lääketieteellisen osuuden kirjoittaminen oli haastavinta, sillä en käytännössä ennestään omannut tämän alueen tietämystä.

Projektin parissa ahertaminen on ollut mielekästä, ja on vihdoin tullut aika saattaa työ päätökseen. Sen aikana on kerääntynyt myös kosolti lisää ideoita, jotka on ollut pakko jättää jatkokehitykseen; intoa riittäisi, aikaa ei. Tästä huolimatta on ollut ilo seurata aikaansaannoksensa kehittyvän vastaamaan asetettuihin vaatimuksiin. Tyytyväisyyttä on entisestään lisännyt tieto siitä, että Mega Elektroniikka Oy aikoo integroida työn tuloksia omiin tuotteisiinsa.

Työskentely oli pääsääntöisesti itsenäistä; joitakin käyttöliittymän linjauksia tehtiin työtä ohjanneen professori Lippinging kanssa. Lisäksi mittaustiedon lähetykseen käytetyn protokollan ensimmäisen version määritti palvelintoteutuksesta vastannut Girf Oü. Kyseistä protokollaa parannettiin tästä vikasietoisammaksi ehdotusteni pohjalta.

Epäilyksettä suurimmat kiitokset kuuluvat työtä ohjanneelle professori Tarmo Lippingille; kanssanne työskentely on ollut etuoikeus. Olette olleet suuri innoittaja kiinnostuksessani signaalinkäsittelyn lääketieteellisiin sovelluksiin.

Iso kiitos myös Mega Elektroniikka Oy:lle tutkimuksen osittaisesta rahoituksesta, Jukka Kinnuselle työn tarkastamisesta, Jari Happoselle NeurOne-ohjelmistotuesta ja Olli Heikkiselle työssä toteutetun sovelluksen testaamisesta. Toivon, että saatte työn tuloksista maksimaalisen hyödyn.

Yhteistyöstä haluan osoittaa kiitokset myös Andres Anierille (Girf Oü), toivottaen samalla palvelinratkaisullenne menestystä. Erityiskiitos työn lääketieteellisen osuuden (luvut 2 - 3) tarkastamisesta myös dosentti Ville Jäntille, sekä kieliopillisesta avusta tyttöystävälleni/diplomityöleskelle Lillille.

Porissa 10. toukokuuta 2012

_____

Paavo Pitkänen

SISÄLLYS

TERMIT JA MÄÄRITELMÄT

API	Application Programming Interface, sovellusrajapinta. Määritetty säännöstö, jota noudattaen ohjelmistokomponentit voivat kommunikoida keskenään.
Attribuutti (olio-ohjelmointi)	Luokkaan määritetty tietokenttä (jäsenmuuttuja), johon säilötään luokasta tehdyn olion ilmentymään liittyvää tietoa.
CAO	Katso Client Activated Object
Client Activated Object	Remoting-kommunikaation tyyppi, missä palvelimella sijaitsevan objektin elinikä määräytyy asiakkaan mukaan.
EEG	Electroencephalogram, aivosähkökäyrä. Aivokuorelta mitattuja ajan funktioina muuttuvia sähkökentän potentiaalieroja. Lyhenteellä voidaan tarkoittaa myös elektroenkefalografiaa, joka tarkoittaa aivosähkökäyriä tieteellisenä tutkimusalueena.
EDF	European Data Format. Laajalti käytössä oleva biosignaalien vaihtoon eri järjestelmien välillä suunniteltu standardi.
EDF+	EDF-standardin uudempi, paranneltu versio.
Ilmentymä (olio-ohjelmointi)	Rakenne- ja toimintomäärityksestä (luokka) koostettu olio, jolla on oma tila.
IP	Internet protocol, internetin tietoliikenneprotokolla, jota käytetään reitityksessä.
IPC	Interprocess communication, käyttöjärjestelmän prosessien välinen kommunikaatio.
Käsittelyistunto	Käsittelysovelluksessa ilmenevä mittaustietoa käsittelevä prosessi, joka noudattaa käsittelyprotokollaa.
Käsittelyprotokolla	Toimintaohjeet, joiden mukaan käsittelysovellus käsittelee mittaustietoa.
Käsittelysovellus	Tässä työssä toteutettu modulaarinen ohjelmisto, joka vastaanottaa, käsittelee ja lähettää mittaustietoa.
Liitännäinen	Ohjelmistomoduuli, joka liitetään isäntäsovellukseen ja laajentaa ko. sovelluksen toiminnallisuutta.
MATLAB	MATrix LABoratory. Mathworksin valmistama, laajalti käytetty laskentaohjelmisto.
MATLAB Compiler Runtime, MCR	Ympäristö, jossa voidaan suorittaa MATLAB-sovellusohjelmistolla tuotettuja algoritmeja. Ei sisällä käyttöliittymää, kuten MATLAB.

Metodi (olio-ohjelmointi)	Katso operaatio.
Microsoft .NET	Microsoftin tarjoama, windows-sovelluskehitykseen tarkoitettu ohjelmistoalusta.
Mittausprotokolla	Katso Protokolla (NeurOne).
Mittaustieto	Kaikki mitattavasta kohteesta saatu mittaukseen liittyvä tieto, kuten mitatut signaalit ja tapahtumat.
Mutex, Muteksi	Lyhenne sanoista mutual exclusion. Moniajokäyttöjärjestelmien tarjoama synkronointiprimitiivi.
.NET Remoting	Microsoftin kehittämä hajautettuun prosessienväliseen kommunokaatioon suunnattu rajapinta.
Ohjelmistorajapinta	Katso API.
Open Systems Interconnection	Malli, jossa tietojärjestelmien välinen kommunikaatio jaetaan toisistaan abstraktoituihin kerroksiin, jotka ovat erikoistuneita tiettyihin tehtäviin.
Operaatio (olio-ohjelmointi)	Olion tarjoama palvelu jota jokin ulkoinen taho tai olio itse voi pyytää. Operaatiosta käytetään myös termiä jäsenfunktio.
OSI	Katso Open Systems Interconnection.
PDA	Personal Digital Assistant. Mobiili, henkilökohtainen päätelaite. Esimerkiksi taulutietokone tai älypuhelin.
Plugin	Katso liitännäinen.
Prosessi (käyttöjärjestelmä)	Tietokoneohjelman ilmentymä, joka nykyaikaisissa käyttöjärjestelmissä eristää ohjelman suorituksen ja datan muista ohjelmista. Prosessissa on aina vähintään yksi säie.
Prosessointilohko	Käsittelysovelluksessa toimiva algoritmi, joka käsittelee sille syötettyä mittaustietoa.
Protokolla (tietoliikenne)	Sovittu käytäntö (viestirakenne ja säännöstö) jonka mukaan järjestelmät tai ohjelmat kommunikoivat.
Protokolla (NeurOne)	NeurOne-ohjelmistolla tehtävän mittauksen asetukset (mm. mitä kanavia mitataan ja mitä mittaustiedolle tehdään).
Prototyyppi (olio-ohjelmointi)	Luokkaan tehty operaatio- tai kenttämäärite, jonka mukaisen kentän tai operaation luokan perillisten on toteutettava.
RTP	Real-time Transport Protocol. Protokollakehys joka määrittää standardin videon ja audion siirtoon IP-verkoissa.
SCTP	Stream Control Transmission Protocol. Viestiorientoitunut kulje-

	tuskerroksen tietoliikenneprotokolla, joka tarjoaa luotettavan viestien perille toimituksen.
Signaali (mittaus)	Lukujono, jonka arvot kuvaavat mitatun analogisen suureen arvoja säännöllisin väliajoin.
Sivutustiedosto	Virtuaalimuistin osa, joka sijaitsee tietokoneen massamuistissa.
Säie (käyttöjärjestelmä)	Pienin skeduloitava ohjelman osa, jota suoritetaan prosessoriytimessä.
Tapahtuma (mittaus)	Epäsäännöllisesti ilmenevä joko yhteen ajanhetkeen tai ajanjaksoon sidottu mittaukseen liittyvä tieto (esim. “potilas avasi silmänsä” tai “ärsyke annettu”).
Tarkistussumma	Annetusta tietolohkosta laskettu lyhyt, kiinteän mittainen tietoalkio, jonka avulla tietolohkon eheys voidaan myöhemmin varmentaa.
TCP/IP	Transmission Control Protocol / Internet Protocol. TCP- ja IP-protokollista koostuva protokollapino, joka tarjoaa luotettavan tiedonsiirron.
UDP	User Datagram Protocol. Viestiorientoitunut minimalistinen kuljetuskerroksen tietoliikenneprotokolla.
Virtuaalimuisti	Muistinhallintatekniikka, joka mahdollistaa keskusmuistia laajemman osoiteavaruuden.

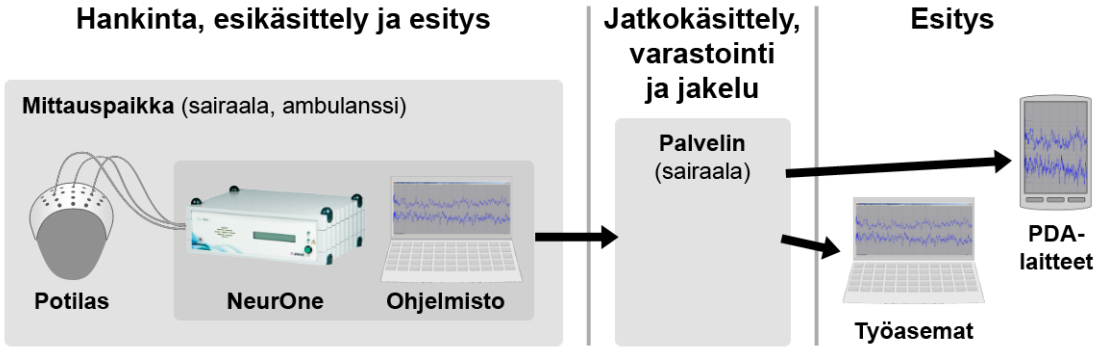
1 JOHDANTO

Moni meistä lienee kuullut sanonnan “Oculus animi index” tai tuttavallisemmin “silmät ovat sielun peili”. Tässä lienee jotain perää, sillä kasvoja ja silmiä tarkkailemalla voi usein tehdä subjektiivisia johtopäätöksiä henkilön sisäisestä tilasta. Neurotieteiden myötä “sieluun” on avautunut ikkunoita, joiden kautta henkilön sisäistä tilaa on mahdollista tarkastella kvantitatiivisesti.

Aivosähkökäyrä (Elektroenkefalogrammi, EEG) on eräs neurotieteen aloilla käytetyistä työkaluista. Sillä on yli satavuotinen historia, ja se on vakiinnuttanut asemansa apuvälineenä niin tutkimuksessa, diagnosoinnissa kuin seurannassakin. EEG-mittauslaitteisto voikin löytyä sairaalasta kliinisen neurofysiologian (KNF) osaston lisäksi myös teho-osastolta tai leikkaussalista. KNF-osastolla EEG-mittauksia käytetään tyypillisesti epilepsian sekä erinäisten tajuttomuus- ja tulehdustilojen diagnosoinnissa. Leikkaussalissa neurokirurgi voi vaativan leikkauksen aikana (intraoperatiivinen monitorointi, IOM) esimerkiksi valvoa potilaan hermoratojen toimintaa niin kutsutuilla *herätevasteilla,* joista on kerrottu tarkemmin luvussa 2.2. Jos leikkauksen aiheuttama hermovaurio havaitaan ajoissa itse operaation aikana, sen korjaamismahdollisuudet paranevat huomattavasti. Teho-osastolla EEG toimii potilaan tilan tarkkailun apuna; esimerkiksi koomapotilailta voidaan säännöllisesti monitoroida aivotoimintaa, etsien mahdollisia virkoamisen merkkejä. [1, luvut 6 ja 7]

EEG:n mittaamiseen on tarjolla laaja kirjo ratkaisuja, joihin luodaan katsaus luvussa 3. Kuopiolainen Mega Elektroniikka Oy toi hiljattain markkinoille NeurOne-nimisen mittalaitteiston, jolla voidaan monipuolisesti rekisteröidä fysiologisia signaaleja. Laitteisto soveltuu erityisesti paikallaan tapahtuvaan monikanavaisen EEG:n pitkäaikaiseen mittaamiseen esimerkiksi teho-osastolla, minkä ansiosta se on hankittu TTY:n Porin yksikölle tutkimusaineiston keräämistä varten. Sen mukana saadun ohjelmiston toiminnallisuutta halutaan kuitenkin täydentää ominaisuudella, joka mahdollistaa kerätyn mittaustiedon siirtämisen verkon yli palvelimelle. Tavoitteena on, että yön aikana tehdyistä rekisteröinneistä voidaan laskea valmiiksi erinäisiä trendejä, jotka ovat aamulla tarkasteltavissa esimerkiksi KNF-työaseman kautta (kuva 1.1). Signaaleja halutaan myös käsitellä ja esittää eri tavoin jo mittauspaikalla. Tähän liittyy olennaisesti käsittelyalgoritmien käyttöönoton helppous. Diplomityön suunnitteluprosessin aikana on ideoitu myös ambulanssissa tapahtuvia, 3G-dataliittymän yli siirrettäviä EEG-mittauksia. Tämä on kuitenkin rajattu työn ulkopuolelle, sillä kyseisen ominaisuuden suunnittelusta, toteutuksesta ja testauksesta saa laajuuden puolesta oman diplomityönsä. Työn päätavoitteena on tuottaa NeurOne-sovellusohjelman rinnalla toimiva ohjelmistokomponentti, joka kykenee käsittelemään mitattuja signaaleja ja lähettämään niitä IP-tietoverkon kautta palvelimelle. Palvelinpuolen toteutuksesta vastaa vuonna 2000 perustettu Tallinnassa toimiva

ohjelmistoalan yritys nimeltä Girf Oü. Toissijaisena tavoitteena on tuottaa Mega Elektroniikka Oy:lle ohjelmistomateriaalia (esim. ohjelman toimintamalleja, uudelleenkäytettäviä ohjelmistokomponentteja ja lähdekoodia), jota voidaan käyttää tulevissa NeurOne-ohjelmiston versioissa.



Kuva 1.1. *Tiedon kulku mittauspaikalta palvelimelle ja tilaajille.*

Aluksi ohjelmistokomponentin loppukäyttäjinä tulee toimimaan algoritminkehittäjiä, jotka haluavat testata algoritmeja oikeassa mittaustilanteessa, sekä sairaalan teho-osastoja tai yksiköitä, joilla on kiinnostusta pilotoida uusia monitorointimenetelmiä. Ohjelmiston saattamista lääketieteellisten varmennusprosessien läpi ei ole sisällytetty työn aihepiiriin, koska kyseinen prosessi voi kestää hyvinkin pitkään. Tämän johdosta työn tuloksena syntyvää ohjelmistokomponenttia ei voi käyttää sellaisissa signaalinkäsittelyketjuissa, joiden tuloksia käytetään potilaan hoitoon vaikuttavassa päätöksenteossa. Tuloksena toteutettava komponentti on siis suunnattu vain tutkimuskäyttöön, joskin se voidaan myöhemmin kaupallistaa ja sertifioida.

Työssä käsitellään aluksi EEG:n sovellusalueita yleisistä käyttökohteista teho-osastoon sekä esitellään muutamia saatavilla olevia mittausratkaisuja (luvut 2 ja 3). Luvussa 3 kuvataan myös lyhyesti muutama jo olemassa oleva EEG:n reaaliaikaiseen käsittelyyn soveltuva analyysialusta (mm. FieldTrip [2] sekä OpenEEG Project [3]).

Työhön liittyvään NeurOne-järjestelmään tutustutaan luvussa 4. Luvussa 5 edetään konstruktiivisen tutkimuksen toimintamalleja käyttäen rakentamaan ongelma-alueen käsitteistö sekä malli, keräämään sovellukselle asetetut vaatimukset sekä pohtimaan sovellusalueen yleistä problematiikkaa. Toteutusta käsitellään luvussa 6 ja sen evaluointi esitetään luvussa 7. Toteutuksessa käytetään ongelmanreduktion heuristiikkaa jakaen ratkaistavana oleva ongelma selkeisiin ongelma-alueisiin ja näihin sopiviin toiminnallisuuksiin, joiden määrittely ja toteutus etenevät rinnakkain. Suunnittelussa ja toteutuksessa pyritään huomiomaan myös tuotettavan sovelluksen laajempi hyödynnettävyys. [4, luku 5]

Teknisessä toteutuksessa kohdattavien ongelmien ratkaisuvaihtoehtoja pyritään kartoittamaan kirjallisuudesta, olemassa olevista vastaavan tiedonsiirtoprofiilin ohjelmistotuotteista sekä online-materiaalista. Vaihtoehtojen arviointi on kolmitasoinen ja etenee reduktiivisesti: ensimmäisenä eliminoidaan vaihtoehdot, joita ei voida toteuttaa kustannussyistä (raha, aika). Jäljelle jääneitä vaihtoehtoja tarkastellaan teorian näkökulmasta

käyttäen apuna teknistä kirjallisuutta ja muiden mahdollisesti tekemiä tutkimuksia karsien pois ne, jotka vaikuttavat huonosti soveltuvilta. Jäljelle jääneitä vaihtoehtoja verrataan tekemällä jokaisesta vaihtoehdosta prototyyppi ja valitsemalla näistä parhaiten soveltuva (laajennettavuus, ylläpidettävyys, suorituskyky). Ajallisten resurssien vuoksi pyritään siihen, että tarve rakentaa prototyyppejä on minimaalinen.

Työn alussa oleva termistö on pyritty pitämään mahdollisimman koheesiivisena sisällyttäen siihen vain työssä käytettyjen tietoteknisten termien selitykset. Muiden alojen termejä (esim. lääketieteelliset termit luvussa 2) on avattu tarpeen mukaan alaviitteissä.

2 EEG-MONITOROINTI

Viimeisten sadan vuoden aikana ihmisen keskushermoston tutkimuksen työkaluiksi on kehitetty useita eri menetelmiä. Kun tutkimus kohdistuu aivoihin, puhutaan aivokuvantamisesta, joka puolestaan voidaan jakaa joko rakenteen (anatomia) tai toiminnan kuvantamiseen. EEG kuuluu näistä jälkimmäiseen.

Anatomisen kuvantamisen menetelmiä ovat mm. röntgensäteisiin pohjautuvat ventrikulografia, angiografia ja CAT-kuvaus *(Computer Aided Tomography)* sekä magneettiresonanssikuvaus (MRI). Toimintaa voidaan kuvantaa esimerkiksi toiminnallisella magneettiresonanssikuvauksella (fMRI), positroniemissiotomografialla (PET), magnetoenkefalografialla (MEG) sekä elektroenkefalografialla (EEG).

Rakennetta tutkiville menetelmille on ominaista hyvä spatiaalinen erotuskyky, kun taas toiminnallisuutta tutkittaessa tarvitaan hyvä ajallinen resoluutio. Joissain tapauksissa voidaan yhdistää eri menetelmiä (esim. fMRI ja EEG), jolloin saadaan EEG:n ajallisesti tarkka kuva aivojen sähköisestä aktiviteetista sekä fMRI:n tuottama spatiaalisesti tarkka kuva aivojen verenkierrosta. Käytettävissä on siis useita eri menetelmiä, joita voidaan rajallisesti yhdistää. Tässä työssä keskitytään näistä vain elektroenkefalografiaan.

Ymmärtääksemme ympäristön jossa työssä tuotettavaa ohjelmistoa tullaan käyttämään, on aluksi tarpeen tutustua itse mittausmenetelmään – elektroenkefalografiaan (luku 2.1) – ja sen käyttökohteisiin. Toteutettavan ohjelmiston kannalta olennaisin käyttötilanne on monitorointi teho-osastolla, joten luvussa 2.2 käsitelläänkin pääasiallisesti juuri siihen liittyviä EEG:n käyttökohteita. Huomattakoon, että käsitellyt käyttökohteet eivät välttämättä ole yksinomaan teho-osastoon kuuluvia; esimerkiksi epileptistä toimintaa tutkitaan enimmäkseen teho-osaston ulkopuolella. Lisäksi EEG:lla on myös muita käyttökohteita (esim. aivo–tietokone-liitynnät, kognitiivisen neurotieteen tutkimukset ja ADHD-diagnosointi [5]).

2.1 EEG:n synty ja mittaus

Ensimmäiset EEG-mittaukset teki englantilainen Richard Caton vuonna 1875 tutkiessaan apinoiden ja kanien aivoja sekä ihmisen päänahkaa galvanometrillä. EEG-signaalien löytäjänä voidaan kuitenkin pitää alfarytmin[1] vuonna 1929 havainnutta saksalaista Hans Bergeriä. Ensimmäisissä mittauksissaan hän käytti Lippmannin elektrometriä, mutta siirtyi parempien tuloksien toivossa herkempiin galvanometreihin. Mittauslaitteistot ovat sittemmin suurimmaksi osaksi digitalisoituneet. [1, s.74; 6; 7, s. 2]

1. Takaraivolta mitatun EEG:n spektrissä 8–12 Hz taajuusalueella esiintyvä värähtely, mikä on voimakkaimmillaan henkilön silmien ollessa suljettuna.

Tärkeimpänä etuna muihin toiminnallisen kuvantamisen menetelmiin EEG:ssa on erittäin hyvä ajallinen erotuskyky. Lisäksi se on mittausmenetelmänä hyvin skaalautuva; nykyään on saatavilla laaja kirjo eri käyttötarkoituksiin ja budjettiluokkiin sopivia mittauslaitteistoja muutaman elektrodin taskukokoisista tallentimista yli 500 elektrodin mittausjärjestelmiin. EEG-mittauslaitteistot ovat PET- ja MRI-laitteistoihin verrattuna huomattavasti pienempiä, vaativat paljon vähemmän tehoa eivätkä rajoita potilaan liikkuvuutta yhtä paljon. EEG-mittausta voidaan käyttää myös leikkaussalissa sekä kurarisaation[2] että sedaation[3] yhteydessä. Sitä käytetään varsinkin epilepsian diagnoosin vahvistamisessa ja hoidon seurannassa sekä aivokuoleman todentamisessa. Lyhytaikaisen monitoroinnin avulla voidaan ajoissa havaita akuutteja aivotoimintaa uhkaavia muutoksia, ja pidempiaikaisista monitoroinneista saadaan prognoosin[4] muodostamista tukevaa tietoa. Lisäksi sen avulla voidaan paikantaa aivoista fyysisesti vaurioituneita alueita (kasvain, trauma tai halvaus) sekä testata afferenttien[5] hermoratojen toimintaa (herätepotentiaalit, luku 2.3). Leikkaussalissa sitä käytetään mm. anestesian syvyyden arviointiin. [7, s. 9; 8, s. 75; 9; 10, s. 221]

EEG:ssa mitataan pääasiassa aivokuorella sijaitsevien hermosolujen eli neuronien (harmaa aine) hermoimpulsseja – tai tarkemmin ilmaistuna neuroniryhmien aktiopotentiaalien keskiarvoja [10, s. 217]. Jotta ymmärtäisimme tämän hieman paremmin, tarkastellaan aluksi hermosolujen toimintaa.

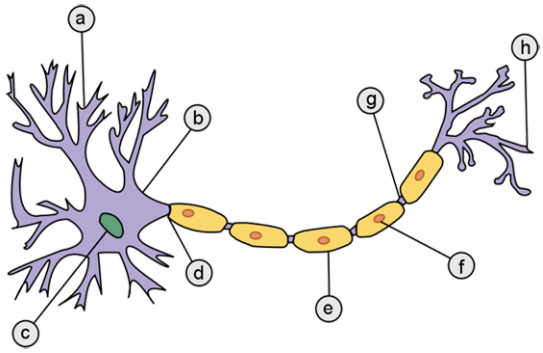
Aivokuori (korteksi) on isoaivojen uloin osa (kuva 2.1). Se on 2–5 mm paksu poimuttunut kerros, jonka pinta-ala on noin 2000 cm^2 ja sen on arvioitu sisältävän kymmeniä miljardeja neuroneja [11, s. 4]. Neuronin rakenne on esitetty kuvassa 2.2; jokaisella neuronilla on yleensä useita tuojahaarakkeita (dendriitti) sekä yksi viejähaarake (aksoni). Neuroni vastaanottaa muiden neuronien lähettämiä hermoimpulsseja dendriittien välityksellä. Kynnysarvon ylittyessä se lähettää hermoimpulssin pitkin aksonia, joka on kytköksissä useisiin muiden hermosolujen dendriitteihin; aksonin ja dendriitin välistä yhteenliittymää kutsutaan synapsiksi. Korteksin neuronit ovat erittäin vahvasti kytköksissä toisiinsa – yhteen aksoniin voi olla liittyneenä tuhannesta sataan tuhanteen synapsia. Yksilön vanhetessa tämä hermoverkko menettää neuroneja, mutta vastaavasti niiden välisten synapsien määrä kasvaa. Aikuisilla onkin noin $5 \cdot 10^{14}$ (viisisataa biljoonaa) synapsia. [7, s. 8]. Vertailun vuoksi mainittakoon, että kokonaisuudessaan ihmisen aivoissa on arvioitu olevan satoja miljardeja neuroneja. Kytköksiä on siis huomattavasti enemmän kuin neuroneja.

EEG-mittaukset voidaan jakaa invasiivisiin ja ei-invasiivisiin; invasiivisissa menetelmissä nimensä mukaisesti “tunkeudutaan” kehon sisälle. EEG:n tapauksessa tämä tarkoittaa joko ihonalaisia neulaelektrodeja tai elektrodien viemistä suoraan aivokuorelle, joka puolestaan edellyttää kraniotomiaa[6]. Invasiiviset mittaukset ovatkin huomattavasti

2. Lihasten osittainen lamaannuttaminen antamalla potilaalle curare-myrkyn kaltaisia yhdisteitä, jotka estävät hermoimpulssin välittymisen lihaksen ja hermon välisessä liitoksessa.
3. Potilaan rauhoittaminen antamalla keskushermostoon vaikuttavaa ainetta.
4. Ennuste potilaan tilan kehityksestä.
5. Aistimista keskushermostoon vieviä.
6. Kirurginen toimenpide, jossa pääkallo avataan.

harvinaisempia valmistelun vaatiman ajan sekä teknisten ja eettisten rajoitteiden vuoksi. Viemällä elektrodit aivokudoksen sisälle voidaan tutkia hyvinkin paikallisia ilmiöitä päästen jopa yksittäisten neuronien tasolle (syväelektrodimittaus). Suoraan aivokuoren pinnalta mitattuun EEG:hen voidaan viitata myös lyhenteellä ECoG (electrocorticogram). Invasiivisesti mitattujen signaalien ominaisuudet (taajuussisältö ja vaihekäyttäytyminen) ovat hyvin riippuvaisia mittauselektrodien koosta ja sijainnista. Eräs invasiivisuudesta saatu hyöty on, että mitattaviin signaaleihin tulee vähemmän häiriöitä. [11, s. 4]

Kuva 2.1. *Aivojen poikkileikkaus edestäpäin, kuvassa aivokuori on tummennettu uloin osa.*



Kuva 2.2. *Neuronin rakenne: a) dendriitti, b) solukeskus (sooma), c) tuma, d) aksonikeko, e) Schwannin solun myeliinituppi, f) Schwannin solun tuma, g) Ranvierin kurouma, h) aksonin pääte. [12]*

Ei-invasiivisissa mittauksissa elektrodit sijoitetaan päänahalle. Sähkönjohtavuuden parantamiseksi päänahka–elektrodi-rajapinnassa käytetään lisäksi erinäisiä geelejä tai suolaliuoksia. Näiden mittausten spatiaalinen tarkkuus ei ole kovin hyvä, sillä aivokuoren ja elektrodien välissä olevat pehmytkudos- ja luukerrokset levittävät signaalit laajalle alueelle. Yksittäinen ei-invasiivinen elektrodi mittaa arviolta noin 100 miljoonan...miljardin synkronisti toimivan neuronin postsynaptisten potentiaalien keskiarvoa. Silti näistä mittauksista saatu aivotoiminnan kokonaiskuva voi olla erittäin hyödyllinen. Käytännössä kallon sisältä ja päänahalta mitatut signaalit sisältävät luonteiltaan erilaista tietoa; toisin sanoen ei voida väittää että toinen menetelmä olisi yleisesti parempi kuin toinen. [11, s. 4]

Mittaustapahtuma on hyvin herkkä sähkömagneettisille häiriöille. Esimerkiksi mittaelektrodien liikkeistä johtuvat impedanssivaihtelut tai lähellä olevien lihasten sähkömagneettinen toiminta (EMG[7]) aiheuttavat mitattuihin signaaleihin artefakteja, jotka huomattavasti aivokuorelta mitattuja signaaleja voimakkaampina hautaavat nämä alleen. Ei-invasiivisesti mitattu EEG on amplitudiltaan kymmenien mikrovolttien luokkaa[8], kun taas esimerkiksi silmän liikkeistä aiheutuvat signaalit (EOG, *Electro-Oculogram*) ovat satojen mikrovolttien suuruisia. Lisäksi mittausta voi häiritä niin kutsuttu ballistokardiografinen artefakta (BCGa). Tämä artefakta syntyy sydämen sykkeen tahdissa heilah-

7. EMG on lyhenne termistä electromyograph, joka tarkoittaa ihmisen tietoisesti hallitsemista lihaksista mittattua sähköistä aktiviteettia, eli lihassähkökäyrää.
8. Epileptiset piikit voivat olla voimakkuudeltaan jopa millivoltteja.

televien elektrodien impedanssivaihteluista (sydämen toimintakierron systolisen[9] vaiheen aiheuttama verenpaineaalto voi esimerkiksi nytkäyttää päätä tai päänahan verisuoniin tullessaan liikuttaa päänahkaa). [13, s. 55]

Myös kaikki mittausympäristön sähkömagneettiset lähteet voivat aiheuttaa puutteellisen maadoituksen yhteydessä ongelmia; esimerkiksi sähköverkosta voi mittaukseen indusoitua hyvinkin voimakas 50 hertsin (euroopassa käytetty sähköverkon perustaajuus) taajuinen häiriösignaali. Tätä voidaan jälkikäteen kompensoida esimerkiksi notch-suodatuksella.

Jotta eri potilaista tehtyjä mittauksia voitaisiin vertailla, on tärkeää varmistaa, että elektrodit sijaitsevat samoissa paikoissa suhteessa aivokuoreen. Ei-invasiivisissa mittauksissa – ilman tietoa kallonsisäisestä topografiasta – täytyy luottaa elektrodien sijoitteluun päänahalla. Tätä varten suunniteltu vuonna 1958 julkaistu 10-20-järjestelmä on nykyään laajalti käytössä. Vaikka järjestelmästä on sittemmin havaittu lukuisia puutteita, sitä ei ole paranneltu. Eräs epäkohta on se, että järjestelmä edellyttää aivojen ja kallon symmetrisyyttä. Valtaosalla ihmisistä tämä symmetria ei kuitenkaan toteudu. [1, s. 24–29]

Mittaavat elektrodit voidaan kytkeä vahvistimeen unipolaarisesti tai bipolaarisesti. Unipolaarisessa kytkennässä käytetään yhteistä referenssiä, joka voi olla myös keskiarvo usealta elektrodilta mitatuista potentiaaleista. Keskiarvo voidaan laskea esimerkiksi kaikista elektrodeista tai kunkin elektrodin lähellä olevien muiden elektrodien yli. Bipolaarisen kytkennän tapauksessa jokainen kanava määräytyy kahdelta elektrodilta mitattujen potentiaalien erotuksena. Elektrodien kytkennällä on vaikutusta mitatun tiedon tulkintaan ja vertailtavuuteen potilaiden välillä – siksi onkin tärkeää, että myös tämä tieto säilyy mitatun datan yhteydessä. [1, s. 31]

2.2 EEG teho-osastolla

Teho-osastolla tarvitaan yleensä sekä lyhyen että pitkän aikavälin seurantaa. Muutokset aivotoiminnassa ovat nähtävissä EEG:ssa 1–10 sekunnin viiveellä, joten lyhyen aikavälin monitorointi auttaa puuttumaan ajoissa uhkaaviin aivotoiminnan muutoksiin. Tämän osalta Yli-Hankala ilmaisee asian yksiselitteisesti:

> “Neurofysiologisen monitoroinnin päätavoite on auttaa ehkäisemään keskushermoston vahingoittumista.” [1, s. vii]

Pitkäaikainen monitorointi puolestaan tuottaa tietoa prognoosin tekemisen tueksi. EEG-laitteiden kehitys paperille tallentavista digitaalisiksi on mahdollistanut useiden tuntien, jopa päivien, kestoiset rekisteröinnit. Digitaalista tallennetta voidaan myöhemmin kopioida, prosessoida ja analysoida sekä lisäksi siitä voidaan katsoa useiden tuntien aikana tapahtuneet olennaiset tapahtumat yhdellä silmäyksellä (*Cerebral Function Monitor* (CFM), ja amplitudi-integroitu EEG (aEEG)). Aikaa tiivistävät rekisteröinnin esitykset ovat erityisen hyödyllisiä seurattaessa potilaan aivojen tilaa, joka muuttuu hyvin

9. Systolisessa vaiheessa sydämen kammiot supistuvat ja veri virtaa kammioista valtimoihin.

hitaasti tuntien ja vuorokausien mittaan. Tällaisia hitaita muutoksia voivat aiheuttaa esimerkiksi puutteellisesta verenkierrosta (iskemia[10]) aiheutuneet vauriot, päähän kohdistuneet fyysiset vauriot (traumat) tai kemialliset häiriöt. On syytä painottaa, että EEG:a tulkittaessa on aina huomioitava potilaan kokonaisvaltainen tila, sillä esimerkiksi vireystila, annetut lääkkeet tai ennestään diagnosoimattomat patologiat voivat vaikuttaa keskushermostoon. [8; 9]

EEG-monitoroinilla on useita konkreettisia käyttökohteita. Käsitellään seuraavaksi näistä yleisimpiä teho-osastolla.

Päähän kohdistunut fyysinen trauma voi aiheuttaa kudosrakenteiden vaurioitumista sekä eriasteista iskemiaa ja hypoksiaa[11]. Näiden johdosta kallonsisäinen paine (*Intracranial Pressure,* ICP) voi nousta, jolloin verenvirtaus aivoissa vähenee edelleen. Tämä puolestaan voi johtaa aivoverenvuotoon tai aivoödeemaan[12], joka edelleen vähentää veren virtausta johtaen mahdollisesti iskemiaan. Tämä, sekundaarinen hypoksis-iskeeminen vaurio, voi johtaa aivovaurion pahenemiseen jopa vuorokausien kuluessa. Kallonsisäistä painetta pyritään vähentämään anesteettisilla aineilla, ja EEG:aa voidaan käyttää apuna anestesian hallinnassa. Teho-osastolla EEG auttaa kliinisesti huomaamattomien iskeemisten muutosten ja purkaustoiminnan havaitsemisessa. Tutkimuksissa on saatu myös viitteitä siitä, että pitkäaikaisella monitoroinnilla kyetään havaitsemaan aivokuoren iskemiaa varhaisessa vaiheessa, jolloin pysyvät vauriot ovat vielä ehkäistävissä. [1, s. 221–225; 14]

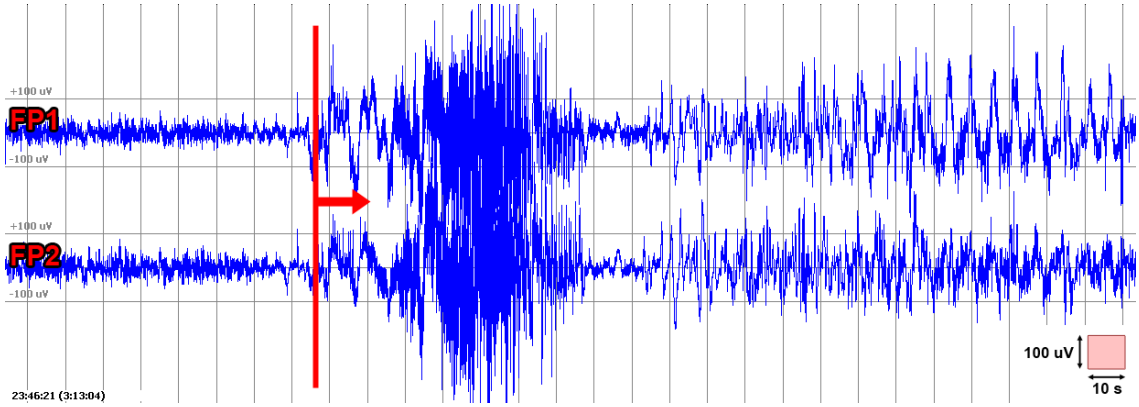
Sydänpysähdys johtaa verenkierron lakkaamiseen ja pitkittyneenä koko kehon laajuiseen (globaaliin) iskemiaan. Aivojen osien herkkyys verenkierron puutteeseen vaihtelee; esimerkiksi aivorunko kestää sitä paremmin kuin aivokuori. Sydänpysähdyksen aikana EEG indikoi selvästi aivoverenkierron pysähtymistä. Välittömän elvytyksen palautettua riittävän aivoverenkierron palaa myös EEG pian normaaliksi ja potilas toipuu ilman merkkejä vauriosta. Jos elvytyksen aloitus kuitenkin viivästyy tai elvytyksessä ei saada palautettua aivoverenkiertoa, voi jälkiseuraamuksia ilmentyä. Välittömässä hoidossa tärkeintä onkin ehkäistä aivoille mahdollisesti syntyvät vauriot. Tämän osalta on tutkittu useita eri aineita (mm. barbituraatteja, lidoflasiinia ja nimodipiiniä) sekä lievää hypotermiaa. Näistä vain hypotermia on osoittautunut selkeästi hyödylliseksi – sen tarkkaa vaikutusmekanismia ei tosin vielä tunneta. Kuten fyysisen traumankin tapauksessa, voi myös sydänpysähdyksen aiheuttaman iskemian johdosta esiintyä purkaustoimintaa joka voidaan havaita EEG-mittauksissa. Jos potilas on sedaatiossa, ei kohtauksen kliinisiä oireita, kuten nykimistä, ehkä ilmene. EEG:lla nämä voidaan sen sijaan havaita. Lisäksi EEG:aa voidaan käyttää prognoosin muodostamiseen yhdessä somatosensoristen herätepotentiaalien (*Somatosensory Evoked Potential*, SEP) kanssa. [1, s. 216; 15]

Aivokuume (enkefaliitti) aiheuttaa tyypillisesti kognitiivisia häiriöitä (ajattelun vaikeutuminen, puhehäiriöt, muistihäiriöt, sekavuus). Se voi myös ilmetä kouristuskohtauksena ja aiheuttaa tajuttomuutta tai johtaa kuolemaan. Parantumisennuste vaihtelee

10. Verenkierron estyminen kudoksessa, jolloin solujen hapen ja ravinnonsaanti estyy.
11. Hapenpuute kudoksissa.
12. Nesteen kerääntyminen aivokudoksiin niin, että aivot alkavat turvota.

täydellisestä toipumisesta yli vuoden kestäviin kognition jälkioireisiin, taudinaiheuttajasta riippuen. Diagnoosi on vaikeaa ja perustuu aina muiden syiden poissulkemiseen. Enkefaliitti voi näkyä muutoksina EEG:ssa; yleisin näistä on normaalien taustataajuuksien hidastuminen. Lisäksi saattaa esiintyä fokaaleja[13] ajallisesti lyhyitä suuriamplitudisia piikkejä, ja reagointi visuaalisiin herätevasteisiin (*Visual Evoked Potential*, VEP) saattaa muuttua. EEG:sta on apua enkefaliitin hoidon seurannassa. [16; 17; 18; 19]

Epileptinen toiminta voidaan nähdä EEG:ssa jo kohtausten välisellä puolen tunnin rekisteröinnillä. Potilaalla voi olla myös jatkuva epileptinen purkaus (non-konvulsiivinen status epilepticus), jolloin kohtauksesta ei välttämättä ole havaittavissa ulkoisia merkkejä, vaikka purkaustoimintaa tapahtuu aivoissa. Tällöin jatkuvan purkauksen tunnistaminen potilasta pelkästään kliinisesti arvioimalla on mahdotonta ja epilepsiadiagnoosin vahvistamiseen tarvitaan EEG:aa [8; 9]. Lisäksi EEG on tällä hetkellä ainoa keino seurata kuinka epilepsian hoito toimii [8, s. 75]. Esimerkki epilepsiakohtauksen ilmenemisestä EEG:ssä on nähtävillä kuvassa 2.3.



Kuva 2.3. *Konvulsiivisen, toonis-kloonisen epileptisen kohtauksen alkaminen [20; 21] (punainen pystyviiva ja nuoli). Kuvassa kaksi signaalia otsalohkosta (FP1, FP2).*

Teho-osastolla potilailla voi esiintyä useita **tiloja, jolloin tietoisuuden taso heikentyy** (esim. luonnollinen uni, anestesia ja sedaatio, vegetatiivinen tila ja kooma). Ihmisen tietoisuudelle ei vielä ole täysin tarkkaa yleisesti hyväksyttyä määritelmää. Kliinisissä tarkoituksissa tietoisuus voidaan jakaa kahteen elementtiin: tajunta *(awareness)* ja vireys *(arousal)*. Tajunnalla tarkoitetaan tietoisuuden sisältöä (yksilön ajatuksia ja tunteita) ja vireydellä puolestaan yksilön hereilläoloa sekä kykyä reagoida ärsykkeisiin. Eri tajunnan tiloja (kuva 2.4) voi olla vaikea erottaa toisistaan puhtaasti kliinisin menetelmin – EEG on hyödyllinen näiden tilojen objektiivisessa arvioinnissa. [22; 23]

Kooma on vuodesta 1941 saakka määritelty seuraavasti:

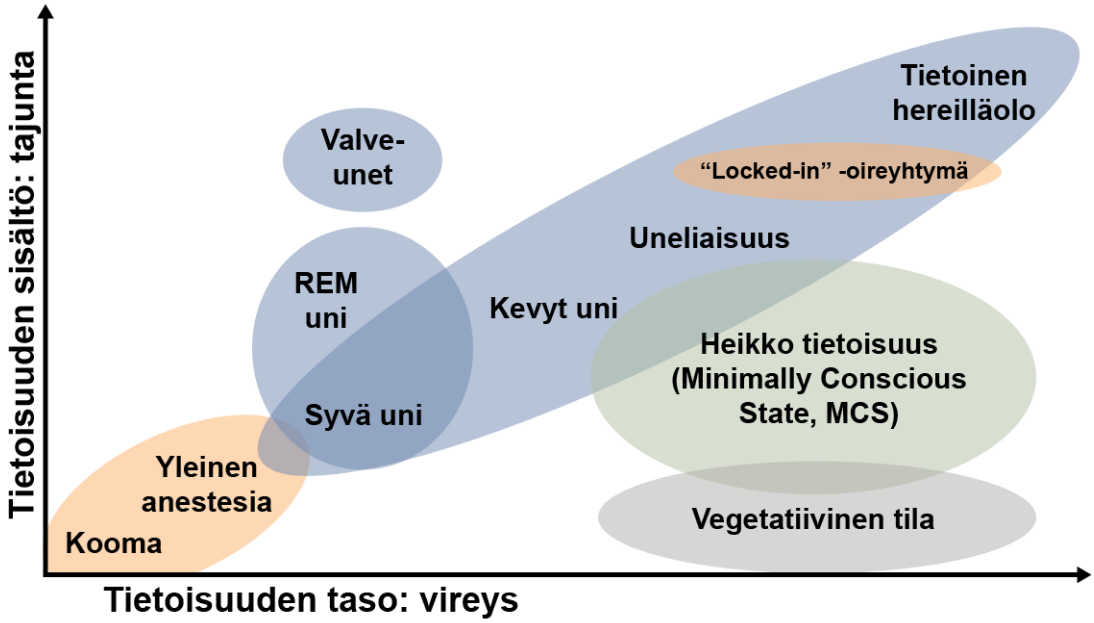
> "Täydellinen tiedottomuuden tila, joka ilmenee kyvyttömyytenä tuottaa minkäänlaista psykologisesti ymmärrettävää vastetta ulkoisiin ärsykkeisiin tai sisäisiin tarpeisiin." [1, s. 190]

EEG:n avulla saadaan tietoa potilaan keskushermoston vallitsevasta tilasta sekä nähdään, tapahtuuko siinä mitään kehitystä (esim. tekemällä herätepotentiaalikokeita pitkäl-

13. Paikallisia, pesäkkeitä.

lä aikavälillä). Hoidon jatkamisen kannattavuuden lisäksi prognoosilla voi olla myös merkittävä psykologinen vaikutus potilaan omaisille. [1, s. 189]

Kooman arviointiin on kehitetty mm. Glasgown kooma-asteikko (*Glascow Coma Scale*, GCS), jonka avulla voidaan arvioida kooman syvyyttä erilaisten fyysisten vasteiden avulla (silmien avaaminen, pupillin reagointi, äänen tuottaminen sekä liikekyky) kliinisessä[14] tutkimuksessa. Tämäntyyppiset mittaukset ovat helppoja toteuttaa, mutta alttiina lääkärin subjektiiviselle tulkinnalle sekä muihin potilaan tilaan vaikuttaville tekijöille (patologiset syyt, hermo- ja kudosvauriot, lääkkeiden vaikutus). Neurofysiologisen monitoroinnin avulla potilaan tilasta saadaan lisätietoa, joka yhdessä kliinisten luokittelujärjestelmien kanssa toimii prognoosin tekemisen ja päätöksenteon aputyökaluna. Kooman arviontiin EEG:llä voidaan käyttää esimerkiksi Synekin laatimaa anoksisen[15] ja traumaattisen[16] kooman arviointiin sopivaa yhteenvetoa [24], joka pohjautuu Scollo-Lavizzarin [25] ja Hockadayn [26] tutkimuksiin. [1, s. 3; 8; 27]



Kuva 2.4. *Tietoisuus Laureyn mukaan [23].*

Potilas voi olla myös ulkoisesti koomalle vaikuttavassa tilassa, mutta silti täysin tajuissaan. Tällaista tilaa kutsutaan nimellä **"Locked-in" -oireyhtymä** (*Locked In Syndrome*, LIS). Söderholm et al kuvaavat tämän artikkelinsa alussa seuraavasti:

> "'Locked-in'-oireyhtymä on laajan aivorunkovaurion aiheuttama tila, jossa potilaan raajat, vartalon lihakset, kasvot sekä puhumiseen ja nielemiseen tarvittava lihaksisto ovat halvaantuneet. Silmien liikkeet ovat useimmiten säilyneet. Ajattelu, muisti, järkeily, kielellinen kyky ja tunteet ovat entisellään, ainoastaan keinot itsensä ilmaisemiseen puuttuvat." [28]

14. Lääketieteellistä (lääkkeiden tai hoitomuotojen vaikutusten) tai terveydenhuollon (potilaan diagnosointi) tutkimusta.
15. Happivajauksesta aiheutuneen.
16. Fyysisestä vahingosta johtuvan.

EEG voi auttaa tunnistamaan potilaat, jotka voisivat muutoin saada koomadiagnoosin. Tätä voidaan yrittää esimerkiksi pitkäaikaisilla EEG-monitoroinneilla, joissa etsitään uni-valverytmiä tai lyhyemmillä mittauksilla etsien valvetilan aktiivisempaa aivotoimintaa. [1, s. 196–198]

Kliinisen määritelmän [22] mukaan **vegetatiivisessa tilassa** oleva potilas on hereillä, muttei tietoinen itsestään tai ympäristöstään. Yleensä hengitys- ja refleksitoiminta on ennallaan sekä potilas saattaa toimia spontaanisti, kuten avata silmänsä tai liikuttaa raajojaan ei-tarkoitushakuisesti. Lisäksi potilas saattaa osoittaa spontaanisti tai ei-verbaalien ärsykkeiden johdosta tunnereaktioita kuten raivoa, itkua, kiljumista tai hymyilyä. [22]

Äskettäin tehty tutkimus on antanut viitteitä siitä, että moni vegetatiiviseksi diagnosoitu potilas voi olla ainakin osittain tietoinen. Tutkimuksessa seurattiin 16 vegetatiivisen diagnoosin saaneen potilaan EEG:ja ja huomattiin, että kolmen potilaan EEG reagoi yksinkertaisiin komentoihin. Potilaat, jotka eivät osoittaneet behavioristisia vasteita ärsykkeisiin, pystyivät vastaamaan kyllä/ei kysymyksiin aktivoimalla motorisen aivokuorensa eri alueita, kuvitellen joko käsien tai varpaidensa liikuttamista. Näillä tuloksilla voi olla suuria vaikutuksia vegetatiivisen tilan diagnosoinnin kannalta. [29]

Aivokuolemalla tarkoitetaan peruuttamatonta toimintojen lakkaamista isoaivoissa ja/tai aivorungossa. EEG voi myös tukea aivokuoleman diagnoosia [8], joskin käytännössä neurofysiologisia testejä voidaan käyttää ainoastaan varmentamaan diagnoosi, ei itse diagnoosin pohjana [8, s. 75].

Hypnootit lääkkeet kuten propofoli ja barbituraatit aiheuttavat EEG-muutoksia, joiden avulla on yritetty arvioida anestesian syvyyttä (spektraalientropia ja bispektraali-indeksi (BIS)). Tällä hetkellä ei kuitenkaan ole olemassa täysin luotettavaa EEG tai EP-pohjaista anestesian syvyyden mittausjärjestelmää. Jotkut yritykset ovat kehittäneet monitorointiratkaisuja, jotka kertovat anestesian syvyyden yksinkertaisella indeksillä (esim. luvulla väliltä 0–100). Nämä mittarit ovat kuitenkin äärimmäisiä yksinkertaistuksia keskushermoston toiminnasta ja voivat muuttua epävakaiksi mittausartefaktien tai potilaan saamien kohtausten vuoksi. Myös fysiologiset muutokset (esim. arousal-reaktiot) voivat aiheuttaa harjaanjohtavia indeksiarvoja. Ongelmana on, että anestesian syvyys ei ole yksiselitteinen käsite, vaan esimerkiksi aivojen toinen puolisko voidaan nukuttaa syvään anestesiaan, jolloin potilas on yhä hereillä. [1, s. 155–157; 8, s. 75; 9; 30; 31].

2.3 Heräte- ja tapahtumapotentiaalit sekä niiden käyttökohteet

Heräte- ja tapahtumapotentiaalit ovat aivoissa tai aivorungossa ilmeneviä sähköisiä vasteita. Ne ovat seurausta jostain aivojen ulkoisesta tai sisäisestä tapahtumasta, joka voi liittyä matalan tason fyysisten stimulusten käsittelyyn (esim. näkö, kuulo, tunto) tai korkeamman tason kognitiivisiin prosesseihin (esim. huomion kohdentuminen tai muisti). Korkeamman tason vasteita nimitetään tapahtumapotentiaaleiksi (*Event Related Potential*, ERP) ja niitä käytetään lähinnä kognitiivisen neurotieteen tutkimuksissa. Aisteihin

liittyviä, eksogeenisistä herätteistä seuraavia vasteita kutsutaan herätepotentiaaleiksi (*Evoked Potential*, EP). Näistä yleisimpiä tutkimuksissa käytettyjä ovat näköaistin stimuloinnista seuraava VEP *(Visual EP)*, kuuloaistiin liittyvät AEP ja BAEP *(Auditory EP* ja *Brainstem Auditory EP)* sekä tuntoaistin SEP (*Somatosensory EP*). [32, luku 4; 33, luku 1]

Herätepotentiaalien avulla voidaan objektiivisesti testata hermoston toimintaa; vasteen viivästyminen, heikko amplitudi tai puuttuminen voivat viitata hermostollisiin ongelmiin. Visuaalisia herätevasteita käytetään pääasiassa silmän ja näköhermon toimintahäiriöiden tutkimuksissa. BAEP-vasteita käytetään mm. intraoperatiivisessa monitoroinnissa (IOM) neurologisten vaurioiden ehkäisemiseksi, ja MLAEP-vasteissa[17] ilmenevillä viiveillä voidaan puolestaan valvoa anestesian syvyyttä leikkauksen aikana. Somatosensoristen vasteiden avulla voidaan tunnistaa afferenteissa hermoradoissa ilmeneviä ongelmia, kuten tukoksia tai johtokyvyn heikkenemistä, jotka ovat tyypillisiä esimerkiksi MS-taudille (multippeliskleroosi). Niitä käytetään myös selkärankaan kohdistuvissa leikkausoperaatioissa monitoroinnin työkaluna sekä selkäydinvaurioiden diagnosoinnissa ja hoidon seurannassa. [32, luku 4]

Heräte- sekä tapahtumapotentiaalin muoto ja voimakkuus riippuvat herätteen tyypistä ja voimakkuudesta, mittauselektrodien sijainnista sekä koehenkilön kognitiivisesta tilasta (huomiokyky, hereilläolo, odotukset ynnä muut tekijät). Yksittäiset mitatut potentiaalit ovat voimakkuudeltaan hyvin heikkoja, noin 0,1–10 μV:n suuruisia, ja eivät ole silmämääräisesti erotettavissa muusta signaalissa olevasta EEG-aktiviteetista. Ne voidaan kuitenkin saada esille erinäisiä kohinansuodatusmenetelmiä käyttäen; näistä yksinkertaisimmassa keskiarvoistetaan useita peräkkäisiä herätevasteita. Tämä on mahdollista, koska vaste ilmenee yleensä tietyn ajan kuluttua stimuluksesta. Se on siis ajallisesti lukittu suhteessa stimulukseen, kun taasen muu sen peittävä EEG-aktiviteetti on satunnaisempaa. Pelkkä keskiarvoistus ei kuitenkaan toimi hyvin, jos vasteiden muoto muuttuu ajan myötä. [11, s. 24; 32, luku 4]

17. Middle Latency AEP, n. 10–50 ms ajan kuuloherätteen jälkeen kuuloaivokuorella ilmenevä vaste.

3 EEG:N MITTAUS- JA ANALYYSIRATKAISUJA

Elektroenkefalografian käyttö tutkimuksessa ja kliinisissä tarkoituksissa on kasvanut viimeisten kolmenkymmenen vuoden aikana huomattavasti. Markkinoilla on saatavilla laaja valikoima eri valmistajien EEG-monitorointiin soveltuvia tuotteita. Monet valmistajat ovat eriyttäneet laitteet ja ohjelmistot toisistaan, jolloin useita saman valmistajan laitteita voidaan käyttää yhdellä ohjelmistolla. Sama pätee myös toisin päin – yksi laite toimii useiden ohjelmistojen kanssa. Jotkin valmistajat tarjoavat myös ohjelmistojensa kylkiäisenä kehityspaketin (*Software Development Kit*, SDK), jonka avulla kolmannet osapuolet voivat laajentaa kyseisten ohjelmistojen toiminnallisuutta. Tämän lisäksi saatavilla on useita avoimia ja suljettuja ohjelmistoja kerätyn tiedon analysointiin.

Luvussa 3.1 käydään läpi sekä tutkimus- että kliiniseen käyttöön suunnattuja mittausratkaisuja. Vaikka mittausratkaisu ei olisikaan saanut sertifiointia kliiniseen käyttöön, voidaan sitä silti käyttää tutkimuskäyttöön teho-osastolla – olettaen että laitteisto on sertifioitu potilasturvalliseksi. Mittausratkaisujen jälkeen tarkastellaan joitakin EEG:n analyysiratkaisuja, jotka ovat tämän DI-työn sovellusalueen kanssa osittain päällekkäisiä, esimerkiksi reaaliaikaisen käsittelyn osalta (luku 3.2).

3.1 Kaupalliset monitorointiratkaisut

Kliiniseen käyttöön soveltuvia mittausratkaisuja on saatavilla huomattavasti vähemmän kuin tutkimuskäyttöön, mikä on ymmärrettävää, sillä lääketieteellisen hyväksynnän saamiseksi laitteiden ja ohjelmistojen on käytävä läpi pitkä ja kallis sertifiointiprosessi. Sairaalassa käytettävän mittausjärjestelmän täytyy olla lääketieteellisesti hyväksytty. Esimerkiksi laitteistolla täytyy olla EMC-hyväksyntä[18]; se ei saa häiritä muita sairaalan laitteistoja – potilaan terveyden vaarantamisesta puhumattakaan.

Kaikkia tarjolla olevia ratkaisuja ei ole lueteltu ja käyty läpi, vaan esiteltäväksi on pyritty valitsemaan eri tarkoituksiin soveltuvia mittausjärjestelmiä. Tämä osio pohjautuu pääasiassa internetistä valmistajien kotisivuilta saatuun esittely- ja markkinointimateriaaliin. Työhön liittyvä Mega Elektroniikka Oy:n mittausratkaisu (NeurOne) käsitellään erikseen luvussa 4.

Tässä luvussa käsitellään muun muassa laitteiden teknisiä ominaisuuksia. Jotta samojen termien toistamiselta vältyttäisiin, ilmaistaan laitteistoihin liittyvät kanavien määrä, näytteenottotaajuus ja bittitarkkuus muodossa *kanavat / taajuus / bittiresoluutio.* Esimerkiksi ilmaisulla “20 ch / 1024 Hz / 16 bit” tarkoitetaan, että laite tukee maksimis-

18. *Electromagnetic compatibility*, sähkömagneettinen yhteensopivuus. Ominaisuus, joka takaa ettei sähkömagneettinen toimija aiheuta ympäristössään ei-haluttuja vaikutuksia (emissio) ja ettei ympäristö vaikuta häiritsevästi toimijaan (immuniteetti).

saan 20 kanavaa 1024 Hz näytteenottotaajuudella ja yksittäinen näyte ilmaistaan 16 bitin tarkkuudella. Kaikki edellä kuvatut termit ilmaisevat tarkkuutta; kanavien määrä kertoo mittauksen spatiaalisesta tarkkuudesta, näytteenottotaajuus temporaalisesta ja bittiresoluutio amplitudillisesta.

GE Healthcare / CARESCAPE-monitorit

GE Healthcarelta on saatavilla useita potilasmonitorointiratkaisuja. Yksittäinen, potilaan vierelle sijoitettava CARESCAPE-laite voidaan konfiguroida lisäämällä siihen tarvittavat fyysiset mittausmoduulit kuten ECG, spirometri tai EEG. Sitä voidaan käyttää yksinään tai se voidaan yhdistää toisiin laitteisiin ja sairaalan lähiverkkoon, josta laitteistoa pystyy myös ohjaamaan. EEG-moduulilla voidaan mitata 2 tai 4 kanavaa 100 Hz näytteenottotaajuudella. Bittitarkuutta ei ole mainittu moduulin esitteessä, mutta herkkyydeksi ilmoitetaan 60 nV ja toiminta-alueeksi ± 400 μV. Tämän perusteella A/D-muunnoksen tarkkuuden voidaan arvioida olevan 14 bittiä. [34]

Brain Products GmbH

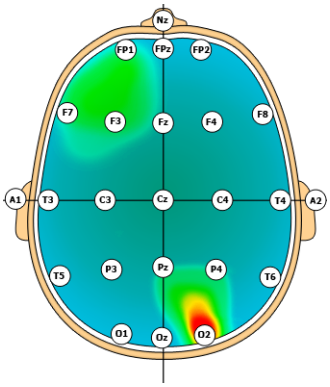
Brain Products GmbH tarjoaa laajan kirjon ratkaisuja perusmittauksista fMRI- ja MEG-kuvantamisen yhteydessä suoritettavaan EEG-taltiointiin. Lisäksi tarjolla on myös kevyitä sekä langattomia laitteistoja nauhoitustilanteisiin, joissa potilaan liikkuvuutta ei saa rajoittaa.

Brain Productsin ratkaisuissa ohjelmisto on jaettu kahden käyttötarkoituksen – tiedon tallennuksen ja analysoinnin – mukaan. Tallennus tapahtuu käyttäen BrainVision Recorder-sovellusta, jolla on myös mahdollista tarkastella mitattuja herätepotentiaaleja. Tallennettujen signaalien analyysi ja käsittely onnistuu mittauksen päätyttyä BrainVision Analyzer 2 -sovelluksessa, jonka toiminnallisuutta voidaan laajentaa .NET pohjaisilla liitännäisillä ja joka integroituu MATLAB-laskentaohjelmistoon. Tämä ratkaisu siis mahdollistaa mitattujen signaalien ei-reaaliaikaisen analysoinnin MATLAB-funktioilla. [35]

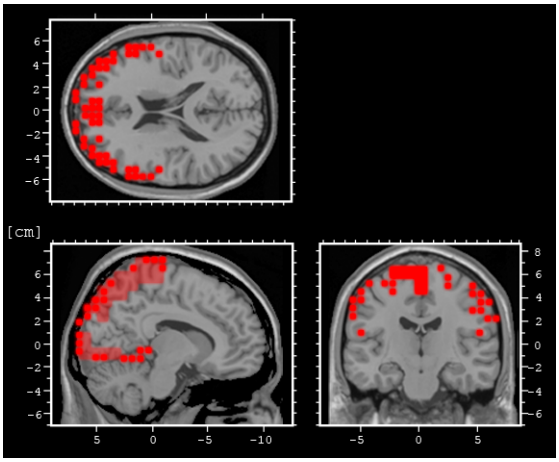
Recorder-sovelluksesta mittaustieto voidaan lähettää yhteen tai useampaan BrainVision RecView-sovellukseen, joka voi sijaita samalla tietokoneella kuin Recorder tai vaihtoehtoisesti jossain lähiverkossa. RecView mahdollistaa mittaustiedon reaaliaikaisen käsittelyn; siinä voidaan rakentaa suodinhierarkia, jonka läpi mitatut signaalit ajetaan. Ohjelman sisäänrakennettuja suotimia ei kuitenkaan ole suunnattu varsinaiseen analysointiin, vaan pikemminkin valvontatyökaluiksi mittauksen laadunvarmistukseen. Map-suodin visualisoi käyttäjälle mitattujen signaalien jännite- tai taajuusjakauman päänahalla, interpoloiden mittaelektrodien välit (havainnollistettu kuvassa 3.1). LORETA-suodinta (*Low Resolution Electromagnetic Tomography* [36]) puolestaan voidaan käyttää tuottamaan kolmiulotteinen visuaalinen approksimaatio EEG-signaalien jännitelähteistä ihmisen pään sisällä (kuva 3.2). Lisäksi tarjolla on näitä tukevia suotimia kuten perussuodatus (low-, high- ja bandpass, sekä notch), FFT-muunnos, signaalien aikaikkunointi, taajuus- ja aikaikkunoiden keskiarvoistus sekä MRI-mittauksen aiheuttamien artefaktien suodatus. RecView-sovellus ei sisällä toiminnallisuutta käsitellyn tiedon edel-

leenlähettämiseksi tai tallentamiseksi. Tarvittaessa myös RecView-sovellusta voidaan laajentaa .NET pohjaisilla liitännäisillä. [37; 38]

Käyttöohjeiden mukaan sekä Recorder että RecView -ohjelmat on suunnattu vain tutkimuskäyttöön, eikä niitä tule käyttää potilaan hoidossa, diagnosoinnissa tai monitoroinnissa. Koska Recorder on ainoa ohjelma, jota Brain Products GmbH tarjoaa mittauslaitteistojensa tueksi, ei mittauslaitteistoja oletettavasti voida käyttää kliinisiin tarkoituksiin. [38; 39]



Kuva 3.1. *Havainnollistus Map-suotimen tuottamasta visualisaatiosta.*



Kuva 3.2. *Havainnollistus LORETA-suotimen tuottamasta visualisaatiosta. Anatomiset kuvat ovat peräisin Wikimedia Commons-projektista [40].*

Compumedics Ltd

Toinen suuri EEG-mittausratkaisujen tuottaja on Compumedics Ltd. Kyseinen yritys koostuu useasta jaostosta. Näistä Compumedics Neuroscan tarjoaa mittalaitteita ja -ohjelmistoja tutkimuskäyttöön ja Compumedics Sleep unidiagnostiikkaratkaisuja kliinisiin tarkoituksiin. Neuroscanilta on saatavissa kaksi laitetta: SynAmp ja NuAmp. Yksittäisellä SynAmp-laitteella päästään tarkkuuteen 70 ch / 20 kHz / 24 bit ja NuAmp-laitteella vastaavasti 40 ch / 1 kHz / 22 bit. SynAmpista on saatavilla myös langattomia, itsenäisesti toimivia versioita, jotka soveltuvat mobiileihin mittauksiin. Kumpaakin laitetta voidaan käyttää yhdessä SCAN-ohjelmiston tai itse laitteen ohjauskerroksen (ACCESS SDK) päälle kehitetyn omatekoisen ohjelmiston kanssa. SCAN-ohjelmisto soveltuu sekä jatkuvan EEG:n että herätepotentiaalien mittaukseen ja kykenee myös mittaustiedon reaaliaikaiseen käsittelyyn. Reaaliaikaisena toiminnallisuutena SCAN tarjoaa mm. herätepotentiaalien keskiarvoistuksen, kanavien lineaarisen yhdistelyn, artefakta-ilmaisun, signaalien topografisen esityksen päänahalla (esimerkki kuvassa 3.1), pääkomponenttien- *(Principal Component Analysis, PCA)* ja toisistaan riippumattomien komponenttien *(Independent Component Analysis, ICA)* laskennan sekä tiedon lähetyksen IP-verkkoon. Internetissä on myös viitteitä siitä, että SCAN-ohjelmistoa olisi mahdollista laajentaa omilla liitännäisillä (plugin), mutta tästä ei ole mainintaa tuote-esitteissä. [41; 42; 43; 44]

Lisäksi Neuroscanilta on saatavissa analyysiohjelma CURRY, joka on tarkoitettu kallonsisäisten jännitelähteiden visualisointiin. CURRY-ohjelmasta voidaan viedä mittaustietoa MATLAB:iin, mutta mittauksen aikainen signaalien käsittely ei ole mahdollista. [45]

Compumedics Sleep -jaosto tarjoaa useita eri laitteistoja ja näiden käyttöön kahta eri ohjelmistoa: ProFusion PSG3[19] ja ProFusion EEG4. ProFusion PSG3 on unidiagnostiikan käyttöön suunnattu unen syvyyden arviointityökalu *(sleep scoring)*. Se kykenee automaattiseen univaiheiden luokitteluun (hypnogrammi), jonka tuloksia KNF-henkilöstö voi käyttää pohjana manuaalisessa luokituksessa. Saman mittauksen voi luokitella useita henkilöitä ja näiden tuloksia voidaan verrata keskenään konsensuksen muodostamiseksi. Lisäksi ohjelma tarjoaa käyttäjälle mittauksesta ja sen tuloksista tiivistetyn esityksen ja kykenee käsittelemään tietoa (esim. univaiheen luokitteluun) myös mittauksen aikana. ProFusion EEG soveltuu valmistajan mukaan EEG:n ja sen liitännäissignaalien (esim. video) keräämiseen, katseluun ja arkistointiin. Ohjelmistosta löytyy myös toiminto kvantitatiiviseen EEG-analyysiin (*Quantitative EEG*, qEEG) sekä toiminto, joka ilmoittaa purkaustoiminnasta reaaliajassa. Kummankin ProFusion-ohjelmiston toiminnallisuutta on mahdollista laajentaa käyttäen ProFusion SDK -kehityspakettia. [46; 47]

Laitteisto on jaettavissa kannettaviin ja työpöytämalleihin. P-Series, Siesta ja Safiro soveltuvat ambulatorisiin[20] mittauksiin, joten ne kykenevät toimimaan myös itsenäisesti. Siesta on mahdollista liittää langattomaan lähiverkkoon ja monitoroida mittausta reaaliajassa, Safiro puolestaan on suunniteltu mahdollisimman pieneksi ja kevyeksi. Näille kannettaville malleille on ominaista rajallinen näytteenottotaajuus ja bittiresoluutio (0,5–1 kHz / 16 bittiä). Työpöytämalleja edustavat E-Series, Scan LT ja Grael -laitteet. E-series on IP-kytkentäinen mittausjärjestelmä EEG-monitorointiin, unitutkimuksiin ja pitkiin epilepsiamonitorointeihin. Scan LT on kevyt USB-liitäntäinen ja helposti liikuteltava. Grael on näistä uusin ja valmistajansa mukaan ensimmäinen high-definition mittalaite tarjoten 48 kanavaa 24 bitin resoluutiolla. Se kytketään tietokoneeseen Ethernet-yhteydellä kuten E-Series. [48; 49; 50; 51; 52; 53]

CareFusion (VIASYS Healthcare, Nervus)

CareFusion tarjoaa laajan valikoiman terveydenhoitoalan ratkaisuja leikkausinstrumenteista hallinnon järjestelmiin. EEG:aan liittyvät ratkaisut jakautuvat kahteen brandiin: SomnoStar ja Nicolet. Näistä ensimmäinen koostuu puhtaasti PSG-sovellusalueen järjestelmistä z4 ja NOX-T3. Nicolet-brandin sovellusalue on laajempi, kattaen mm. diagnostiikka- ja monitorointiratkaisut teho-osastolla, EP- ja EMG-kokeet, unitutkimukset, pitkäaikaismonitoroinnin ja intraoperatiivisen monitoroinnin.

SomnoStar-järjestelmien mukana tuleva ohjelmisto sisältää mm. toiminnallisuuden unen syvyyden luokitteluun ja HRV-analyysiin *(Heartrate Variability)*. Etäkäyttö on esitteen mukaan mahdollista Citrix- tai Terminal Server -yhteyden kautta. Todennäköisesti esitteessä tarkoitetaan pikemminkin Terminal Services (etätyöpöytä) -yhteyttä, joka

19. PSG on lyhenne termistä polysomnography, ja sillä viitataan unen aikana mitattuihin signaaleihin, kuten EEG (aivosähkökäyrä), ECG (sydänkäyrä), EMG (lihassähkökäyrä) ja EOG (silmän liikkeet).
20. Osaston ulkopuolella tapahtuviin.

on Microsoft Windows -käyttöjärjestelmän mukana toimitettava etäkäyttöratkaisu. [54]

Esitteissä ei yksiselitteisesti mainita, sisältääkö Nicolet-brandi yhden vai useampia laitteistoja ja ohjelmistoja. Ainakin käytettävät vahvistimet *(headbox)* ovat osittain valittavissa, ja järjestelmä on saatavissa ambulatorisena, kiinteästi asennettuna sekä kärryasennuksena. Ohjelmiston mainitaan kykenevän epilepsiaan liittyvän purkaustoiminnan ja kohtauksien havaitsemiseen, erinäisten trendien laskentaan sekä unisyvyyden luokitteluun. Lisäksi mittaustiedon reaaliaikainen tarkkailu ja mittauksen jälkeinen tarkastelu on mahdollista tehdä etänä verkon yli. [54; 55]

Selkeästi erillisenä, Nicolet-brandin alaisena ratkaisuna mainittakoon lisäksi IOM:iin tarkoitettu Nicolet Endeavor CR. Sen tarkoituksena on auttaa ehkäisemään neurologisten ja fysiologisten vaurioiden syntymistä selkäydin-, kallo-, verisuoni- ja ortopedisten[21] leikkausten aikana. Sen ohjelmisto sisältää mm. reaaliaikaisten trendien laskentatoimintoja sekä mahdollisuuden katsella mittaustietoa etänä verkon yli. [55]

Micromed S.p.A.

Micromed on italialainen valmistaja, joka tarjoaa tuotteita PSG-, EP-, EMG-, epilepsia- sekä yleisiin EEG-tutkimuksiin. Heidän LTM[22]-lippulaivansa on BrainQuick SD LTM Express -järjestelmä, joka koostuu 1–4 ketjutettavasta mittayksiköstä. Yhdellä yksiköllä päästään tarkuuteen 64 ch / 2048 Hz / 22 bit. Neljällä yksiköllä kanavien määrä nousee vastaavasti 256:een. BrainQuick SD LTM:ää voidaan käyttää yhdessä tietokoneen kanssa, mutta se kykenee myös mittaamaan itsenäisesti 18 tuntiin saakka. [56]

EP- ja EMG-mittauksiin suunnattu MYOQUICK Matrix Line on mittaus- / stimulointiasema, joka soveltuu myös käytettäväksi IOM:ssa. Siinä ei ole niin monta mittauskanavaa kuin BrainQuick SD LTM -laitteessa, mutta näytteenottotaajuus on huomattavasti suurempi: 17 ch / 65536 Hz / 16 bit. [57]

Micromedin tarjoamissa mittausratkaisuissa käytetään SystemPLUS EVOLUTION -ohjelmistoa, joka mahdollistaa mittauksen reaaliaikaisen seurannan (mittausdata, video) verkon välityksellä. Lisäksi verkon kautta on mahdollista ohjata mittauspaikalla sijaitsevaa videokameraa. SystemPLUS EVOLUTION -ohjelmiston tavoitteena on mahdollistaa datan hankinta, tarkastelu, analyysi, raportointi ja arkistointi tehtäväksi samalla koneella tai hajautettuna verkon yli. Verkkosivuilla tai tuote-esitteissä ei ilmoiteta, vaihtelevatko ohjelmiston ominaisuudet riippuen siitä, minkä laitteiston mukana se toimitetaan. Tuote-esitteiden perusteella ohjelmistossa on toiminnallisuutta ainakin EP- ja ERP-kokeisiin sekä PSG- ja epilepsiatutkimuksiin. [56; 57; 58]

Micromed tarjoaa myös intraoperatiiviseen monitorointiin soveltuvaa ratkaisua AXON Eclipseä. Kyseinen järjestelmä on yhdysvaltalaisen AXON Systems:in toteuttama eikä liity Micromedin ratkaisuihin.

Cadwell Laboratories Inc.

Cadwell jakaa EEG-tuotteensa pitkäaikaismonitorointiin ja rutiini-EEG:aan (Easy III

21. Luihin tai tukielimiin kohdistuvien.
22. Long Term Monitoring, ajallisesti pitkään jatkuva monitorointi, voi kestää jopa päiviä.

EEG ja Easy III Ambulatory EEG), unitutkimuksiin (Easy III PSG ja Easy III Ambulatory PSG), teho-osastolla tapahtuvaan monitorointiin (Envisio-järjestelmä) sekä intraoperatiiviseen monitorointiin (Cascade-järjestelmä) soveltuviksi. Näistä kolme ensimmäistä pohjautuvat samaan laitteistoon (Easy-vahvistin), josta on saatavilla 40–56-kanavaisia (pöytämallit) sekä 23- ja 32-kanavaisia (ambulatoriset) versioita. Leikkaussalikäyttöön tarkoitettua Cascadea tarjotaan 16- tai 32-kanavaisena. Easy III -vahvistimet liitetään mittaustietokoneeseen lähiverkon kautta, jolloin tietokoneen ei tarvitse sijaita mittauspaikan välittömässä läheisyydessä. [59; 60; 61; 62]

Easy EEG/PSG -ohjelmistot mahdollistavat mittauksen seurannan verkon välityksellä. Potilas- ja mittaustietoja sekä videota voi seurata mittauksen aikana tai tarkastella jälkikäteen toimistosta tai jopa kotoa VPN-yhteyden välityksellä. Lisäksi kyseiset ohjelmistot kykenevät reaaliaikaiseen trendilaskentaan (mm. EEG-epäsymmetria, *relative alpha variability* sekä purskevaimentumaan pohjautuvat trendit) ja neuroverkkoihin pohjautuvaan adaptiiviseen häiriönpoistoon; jos ohjelmisto päättelee kanavassa olevan jatkuvia häiriöitä, se voi ajon aikana poistaa kanavan trendilaskennasta parantaakseen trendin paikkansapitävyyttä. Pitkäaikaismonitorointien avuksi ohjelmistossa on neuroverkkoihin pohjautuva potilaskohtaisesti oppiva kohtauksenhavaitsemistoiminto *(seizure detection)*. [60; 61]

Cascade käyttää omaa ohjelmistoratkaisua, joka on optimoitu multimodaalisen[23] tiedon käsittelyyn. Myös sillä tapahtuvia mittauksia on mahdollista seurata etänä reaaliajassa. [62]

BIOPAC Systems Inc.

Sekä tutkimus- että opiskelukäyttöön soveltuvia järjestelmiä tarjoavalta BIOPAC Systemsiltä löytyy tarjonnastaan mm. tutkimuskäyttöön suunnatut USB-liitäntäinen MP36 (4 chan / 100 kHz / 24 bit) ja lähiverkkoliitäntäinen MP150 (16 chan / 200 kHz / 16 bit). Opetuskäyttöön on saatavilla esimerkiksi yksinkertainen Biopac Science Lab -järjestelmä, joka sisältää tietokoneeseen äänikortin kautta kytkettävän MP40-mittalaitteen lisäksi opiskelijaa mittauksissa opastavan ohjelmiston. [63; 64; 65]

Tutkimuskäyttöön suunnattu AcqKnowledge-ohjelmisto sisältää myös reaaliaikaisia algoritmeja; näitä ovat mm. IIR- ja adaptiivinen FIR-suodatus sekä keskiarvoon tai mediaaniin pohjautuva pehmennys. Kyseisen ohjelmiston toiminnallisuutta voi laajentaa erillisellä analyysimoduulilla, joka tarjoaa toimintoja mm. univaiheiden luokitteluun, keskimääräisen entropian laskentaan, HRV-analyysiin sekä EOG- ja MRI-artefaktojen poistoon. Analyysimoduuli ei ole käytettävissä mittauksen aikana. [66; 67]

Lisäksi BIOPAC myy lisenssipohjaisia kehityspaketteja mittausratkaisunsa eri osiin; esimerkiksi BIOPAC Hardware API:n (BHAPI) avulla kuka tahansa voi kehittää heidän laitteistoaan käyttävän ohjelman. AcqKnowledge software API:a (ACKAPI) käyttäen kehittäjät voivat lukea ja kirjoittaa AcqKnowledge-ohjelmiston tiedostoformaattia.

23. Useaa eri mediatyyppiä (esim. mitatut signaalit, audio ja video) sisältävän.

3.2 EEG:n analyysialustoja

On myös olemassa tämän työn sovellusalueeseen liittyviä aiempia ohjelmistoratkaisuja, jotka kykenevät biosignaalien reaaliaikaiseen suodatukseen; esimerkkeinä mainittakoon FieldTrip-ohjelmisto [2; 68] ja OpenEEG Project [3].

FieldTrip on laajalti käytetty [69] avoimen lähdekoodin MATLAB-laajennus (toolbox), joka tarjoaa EEG:n ja MEG:n käsittelyn työkaluja kuten ERP-, spektraali-, lähde- ja lähteiden yhteenliittyvyysanalyysit sekä toiminnallisuutta luokitteluun ja tilastolliseen analyysiin. Käytännössä FieldTrip on lajitelma korkean tason funktioita, joita voidaan kutsua MATLAB-skripteistä. Käyttäjä siis kirjoittaa oman analyysiskriptin, joka kutsuu FieldTrip:n tarjoamia toimintoja toteuttamaan halutut operaatiot. Lisäksi FieldTrip mahdollistaa tiedon jakamisen verkon yli sekä verkossa hajautetun laskennan (FieldTrip Buffer). Tässä mittausohjelmistolta saatu mittaustieto lähetetään palvelimella sijaitsevaan puskuriin, josta muut FieldTrip-asiakasohjelmat voivat lukea tietoa ja tallentaa tuloksia. Tätä työtä kirjoitettaessa tuki näyttää olevan saatavilla useiden eri valmistajien mittausratkaisuille – mm. tässä työssä aiemmin käsitellyille BrainVisionin ja Micromedin alustoille (luku 3.1) [2; 68; 70; 71].

FieldTrip on ladattavissa ilmaiseksi, ja sen käyttöön tarvitaan MATLAB-laskentaohjelmisto. Se ei sellaisenaan pysty vastaanottamaan tietoa tässä työssä käytetyltä NeurOne-sovellukselta, mutta tarjoaa mielenkiintoisia jatkokehitysnäkymiä. Tässä työssä kehitetty sovellus voitaneen esimerkiksi yhdistää FieldTrip:iin mittaussignaalien tietolähteeksi ja täten hajauttaa mittaussignaaleille tehty laskenta usealle tietokoneelle.

OpenEEG Project tarjoaa avoimia laitteisto- ja ohjelmistoratkaisuja, kohdistuen lähinnä EEG:n käyttöön aivo-tietokoneliitynnöissä (*Brain Computer Interfaces*, BCI). Tarjotuissa ohjelmistoissa pääpainona on visualisointiin ja audioon pohjautuva biopalaute. Lisäksi mukaan mahtuu erään tarjotun mittalaitteen kanssa yhteensopiva ohjelmisto (NeuroServer) sekä aivorytmeillä ohjattava peli, Brainathlon. [3]

Kuriositeettinä voidaan mainita myös Pythonilla toteutettu hajautetun rinnakkaisprosessoinnin ratkaisu PaPy [72]. Kyseinen sovellus on tarkoitettu – muttei suinkaan rajattu – lähinnä bioinformatiikan hajautettuun offline-laskentaan, joka koskee geneettistä ja kemiallista dataa (kodonit, aminohapot, jne.). PaPy on tarkoitettu tilanteisiin, joissa laskennan kustannus on merkittävästi tiedonsiirron kustannusta suurempi. Kaikki tiedon käsittelylogiikka kirjoitetaan python-ohjelmana, eikä PaPy integroidu suoraan muihin järjestelmiin (esim. NeurOne tai MATLAB). [73]

Tämän diplomityön tavoitteen kannalta olemassa olevien ratkaisujen suurin puute on, etteivät ne sellaisenaan tue NeurOne-mittausratkaisua. Mahdollista valmisratkaisua voisi laajentaa tukemaan NeurOne-sovellusta, mutta tulevaisuuden kannalta tällaisen ohjelmiston lisensointi voi aiheuttaa ongelmia. Lisäksi tutkimuksen tavoitteena on tuottaa Mega Elektroniikka Oy:lle jatkossa hyödynnettävää lähdekoodia.

4 NEURONE-JÄRJESTELMÄ

Tässä luvussa esitellään työssä käytetty Mega Elektroniikka Oy:n valmistama NeurOne-mittausratkaisu. Esittely alkaa tutustumalla sen arkkitehtuuriin, josta edetään laitteistoon (luku 4.1) ja siitä edelleen ohjelmistoon (luku 4.2). Lopuksi tarkastellaan rajapintaa, jonka kautta mittausratkaisun ohjelmistototeutusta on mahdollista laajentaa kolmannen osapuolen toiminnallisuudella (luku 4.3).

Mittausratkaisuna NeurOne on suunnattu neurotieteelliseen tutkimuskäyttöön. Se soveltuu pelkkien EEG-mittausten lisäksi myös EP-, ERP- ja EMG-kokeisiin sekä käytettäväksi transkraniaalisen magneettistimulaation (TMS)[24] ja toiminnallisen magneettikuvauksen (fMRI) yhteydessä. Järjestelmä koostuu mittauslaitteistosta, siihen ethernet-verkkoliitännällä yhdistettävästä PC-tietokoneesta ja kyseisellä tietokoneella toimivasta sovellusohjelmasta. Mega Elektroniikka Oy on määrittänyt PC-tietokoneen minimivaatimukset. Tällä pyritään takaamaan, ettei se muodosta mittaukselle pullonkaulaa. Järjestelmän arkkitehtuurikaavio on esitetty kuvassa 4.1.

Kuva 4.1. *NeurOne-mittauslaitteiston arkkitehtuuri.*

4.1 Laitteisto

Laitteisto koostuu yhdestä neljään esivahvistimesta *(headbox)*, pääyksiköstä *(main unit)* sekä pääyksikölle virtaa syöttävästä medical-luokan 12 V verkkoadapterista. Mitatun tiedon siirto esivahvistimista pääyksikköön tapahtuu optisia kaapeleita pitkin. Esivahvistimelle voidaan syöttää virtaa joko pääyksiköstä tai sen omasta akusta, jolloin esivahvistin saadaan galvaanisesti erotettua pääyksiköstä vähentäen tuntuvasti verkkovirrasta aiheutuvaa häiriötä. Lisäksi laitteiston tietokone ja näyttö kytketään suojaerotusmuuntajan kautta sähköverkkoon ja verkkoisolaattorin kautta tietoverkkoon. [74]

Esivahvistimen tehtävä on suorittaa mitatuille signaaleille A/D-muunnos, alasnäytteistää ne haluttuun näytteenottotaajuuteen ja toimittaa ne pääyksikölle. Se sisältää 24-bittisen A/D-muuntimen, joka toimii 80 kHz taajuudella. Sisäänrakennettu DSP (Digital

24. *Transcranial Magnetic Stimulation* on ei-invasiivinen menetelmä, jossa aivokuoren toimintaa stimuloidaan aiheuttamalla sille pieniä sähkövirtauksia käyttäen lyhyitä, voimakkaita magneettipulsseja.

Signal Processor) tekee alasnäytteistyksen haluttuun näytteenottotaajuuteen, jolla tieto siirretään pääyksikköön optista kaapelia pitkin. Yksi esivahvistin sisältää 40 mittauskanavaa, joista 32 on unipolaarisia ja 8 bipolaarisia. [74]

Pääyksikkö toimii järjestelmän ohjauskeskuksena, joka yhdistää esivahvistimilta saamansa mittaussignaalit mahdollisiin liipaisutietoihin ja lähettää nämä ethernet-verkon välityksellä tietokoneella toimivalle ohjelmistolle. Laitteisto käsittelee näytteitä 32-bittisinä etumerkillisinä kokonaislukuina, joissa yksi bitti vastaa yhtä nanovolttia. Laitteiston lähettämän mittaustiedon vaatima kaistanleveys voidaan laskea kaavalla

$$B = N_C \cdot R \cdot F_S \tag{4.1}$$

jossa N_C on mitattavien kanavien määrä, R yksittäisen näytteen bittiresoluutio ja F_S näytteenottotaajuus. Mitattavien kanavien maksimimäärä saadaan kertomalla esivahvistimen sisältämien kanavien määrä pääyksikön tukemien esivahvistimien määrällä:

$$N_{CMAX} = 40 \cdot 4 = 160 \tag{4.2}$$

Sijoittamalla kaavaan 4.1 laitteiston maksimit saadaan suurin mittaustiedon vaatima kaistanleveys[25] [26]:

$$\begin{aligned} B_{MAX} &= 160 \cdot 32 \frac{b}{näyte} \cdot 80\,000 \frac{näytettä}{s} \\ B_{MAX} &\approx 391 \frac{Mb}{s} = 49 \frac{MB}{s} \end{aligned} \tag{4.3}$$

Lisäksi tiedonsiirrossa käytetyt verkkoprotokollat tarvitsevat jonkin verran kaistaa signalointiin. Tämä asettaakin vaatimuksen pääyksikön ja tietokoneen väliselle ethernet-liitännälle; näin paljon kaistaa vaativissa mittauksissa tarvitaan käytännössä Gigabit Ethernet -tyyppinen liitäntä. Lisäksi tällaisen tietomäärän reaaliaikainen prosessoiminen asettaa rajoitteita sen käsittelyalgoritmien kompleksisuudelle.

Pääyksikkö on suunniteltu kytkettäväksi suoraan tietokoneeseen, mutta periaatteessa niiden välissä voi olla myös lähiverkko. Tässä tosin on huonoja puolia; ensinnäkin käytettävyys huononee, sillä laitteistoa ohjataan tietokoneella toimivasta sovellusohjelmasta. Toiseksi mittaustiedon siirrossa voi aiheutua ongelmia, jos lähiverkon laitteet eivät kykene välittämään mittaustietoa riittävän nopeasti. Lisäksi lähiverkkoon kytkentää rajoittaa laitteen kiinteä IP-osoite. Mega Elektroniikka Oy ei suosittele tai tue laitteen kytkemistä lähiverkkoon.

Kokoonpanoon voi myös kuulua erinäisiä – joko kolmannen osapuolen tai Mega Elektroniikka Oy:n valmistamia – lisälaitteita. Mega Elektroniikka Oy:ltä on saatavilla mm. liipaisuyksikkö *(trigger module)* ja videon synkronointimoduuli *(video trigger)*

25. Huomattakoon, että lasketussa kaistanleveydessä on otettu huomioon vain mitatut EEG-signaalit. Lisäksi mittauksessa voidaan taltioida myös videota tai audiota, jotka vaativat lisää kaistenleveyttä. NeurOne-järjestelmällä voidaan tallentaa audiota aivan kuten EEG-signaalejakin, mutta video ei liity millään tavalla järjestelmän läpi kulkevaan mittaustietoon.
26. Laitteiston nykyversiossa on 100 Mb/s ethernet-piiri. Tästä syystä laitteisto tulee 160 kanavan mittaamista maksimissaan 10 kHz näyttöönottotaajuudella. Tämä rajoite voi kuitenkin tulevaisuudessa muuttua.

sekä lukuisia mittausantureita (esim. kallistuskulma- ja kiihtyvyysantureita). Liipaisuyksiköllä voidaan tuottaa tallenteisiin aikakoodattuja merkintöjä tietokoneelta saatujen herätteiden mukaisesti. Se lähettää liipaisutapahtumia pääyksikölle, joka yhdistää ne mittaustietoon, sekä jakaa liipaisutiedon muihin siihen liitettyihin laitteisiin (esim. synkronointimoduulille). Synkronointimoduuli tuottaa mittauksen yhteydessä mahdollisesti käytettävälle videokameralle ääniherätteen. Tämän herätteen perusteella tallennettu video voidaan synkronoida tallennetussa mittaustiedossa olevaan aikakoodattuun merkintään.

4.2 Ohjelmisto

NeurOnen ohjelmisto on rakennettu modulaariseksi, ja kaikki mittauksen aikana tiedon käsittelyyn liittyvät komponentit toimivat pääohjelman liitännäisinä *(plugin)*. Tämä tarjoaa joustavan tavan sovellusohjelman toimintojen lisäämiseen. Myös useat sovelluksen perustoiminnot pohjautuvat tähän arkkitehtuuriin: esimerkiksi signaalin esitys ruudulla, herätepotentiaalien laskenta sekä mittaustiedon tallennus tapahtuvat ohjelman mukana toimitettavilla liitännäisillä. Mega Elektroniikka Oy on määrittänyt liitännäisille ohjelmistorajapinnan (*Application Programming Interface*, API), jonka pohjalta myös kolmas osapuoli voi toteuttaa NeurOne-liitännäisiä. Kyseinen rajapinta ei kuitenkaan toistaiseksi ole julkinen. Tulevaisuudessa se tullaan todennäköisesti paljastamaan ainakin NeurOne-järjestelmän ostajille.

NeurOne tallentaa mitatut signaalit ja niihin liittyvät tapahtumat tiedostoihin, joiden formaatti on Mega Elektroniikka Oy:n määrittelemä [74]. Kyseinen formaatti on julkinen, joten kuka tahansa voi käyttää sitä. Näin on esimerkiksi mahdollista laajentaa kolmannen osapuolen tekemää analyysiohjelmaa sellaiseksi, että se kykenee lukemaan NeurOnen mittaustiedostoja. Sovellusohjelmalla voidaan myös myöhemmin tarkastella mitattuja tuloksia sekä tallentaa tulokset mm. EDF-tiedostoformaattiin *(European Data Format)*. Tässä konversiossa esitystarkkuus tosin kärsii, sillä Mega Elektroniikka Oy:n formaatti on 32-bittinen, kun EDF:ssa näytteen arvo esitetään 16 bitillä [75; 76].

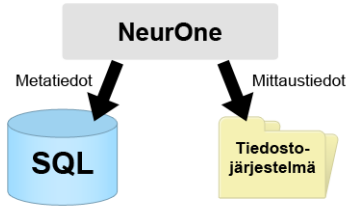
Mittaukseen liittyy tallennettujen signaalien lisäksi myös erinäisiä metatietoja, kuten potilaskohtaiset tiedot (nimi, osoite, ikä, paino jne.), sekä tietoja mittaavasta ogranisaatiosta, kuten organisaation ja teknikon nimi. Nämä tiedot sovellusohjelma säilöö omaan MS-SQL tietokantaansa (kuva 4.2).

Mittauksella *(measurement)* tarkoitetaan sovellusohjelmassa ajanjaksoa, jolta signaaleja tallennetaan. Mittaus kuuluu aina mittausistuntoon, jossa järjestelmä on aktiivisena ja valmiina aloittamaan signaalien tallennuksen. Mittausistuntoon voi kuulua yksi tai useampia mittauksia. Tämä vastaa sitä, että potilas on pysynyt paikallaan ja kiinni laitteistossa, mutta välillä signaalien tallennus on pysäytetty esimerkiksi testiasetelman toimenpiteiden (elektrodien korjaus impedanssin parantamiseksi, stimuluslaitteiden testaus, jne.) vuoksi (kuva 4.3).

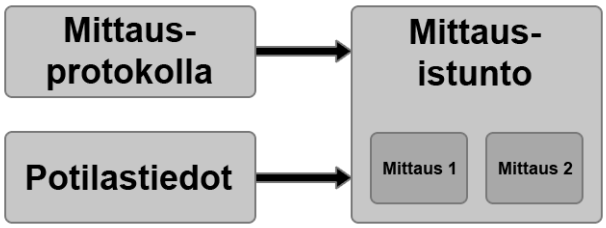
Jokainen mittausistunto noudattaa *mittausprotokollaa*, jossa on määritetty mm. mitattavat kanavat ja mitatun tiedon käsittelyyn käytettävät liitännäiset. Esimerkkinä mit-

tausprotokollassa voi olla määritetty mitattavaksi 1 kHz näytteenottotaajuudella kaksi kanavaa esivahvistimen syötteistä 1 ja 2, ja että kanava 1 esitetään näytöllä (Raw monitor -liitännäinen) sekä signaalit tallennetaan kiintolevylle (Datawriter-liitännäinen).

Liitännäisten asetusten määrittäminen tapahtuu mittausprotokollan määrityksen yhteydessä olevan taulukkomuotoisen käyttöliittymän kautta (liite 1), eikä niillä täten ole omaa graafista käyttöliittymää asetusten syöttämistä varten[27]. Käytännössä liitännäisillä voi olla oma käyttöliittymä vain mittausistunnon aikana.



Kuva 4.2. *NeurOnen tietovarastot.*



Kuva 4.3. *Mittausprotokollan, potilastietojen ja mittausten suhde.*

Liitännäisen elinkaari on sidottu mittausistuntoon; kun käyttäjä aloittaa mittausistunnon, käynnistää sovellusohjelma siihen konfiguroidut liitännäiset. Kun käyttäjä päättää mittausistunnon, odottaa sovellusohjelma kaikkien liitännäisten sammuvan ja palauttaa lopuksi kontrollin käyttäjälle. Liitännäinen ei siis voi toimia mittausistunnon ulkopuolella.

Nämä kaksi edellä esitettyä seikkaa – sidonnaisuus käyttöliittymään sekä rajoitettu elinkaari – asettavat rajoituksia liitännäisen käytölle ja toteutukselle. Esimerkiksi tilanteessa, jossa mittaus on saatettu päätökseen, mutta liitännäinen ei ole ehtinyt prosessoida tietoa reaaliajassa, joutuu käyttäjä odottamaan prosessoinnin valmistumista ja tätä seuraavaa liitännäisen sammumista. Ohjelman käyttöliittymä ei tällä välin vastaa minkäänlaiseen syötteeseen.

NeurOne-sovellusohjelman mukana tulee neljä liitännäistä; ‘Data Writer’ tallentaa mitatut signaalit ja tapahtumat tiedostoihin, ‘Impedance test’ toimii apuna mittausta edeltävässä elektrodien potilaaseen kiinnittämisessä ja ‘Raw’-liitännäinen esittää mitatut signaalit tietokoneen näytöllä. ‘Response monitor’ -liitännäinen on suunnattu herätepotentiaalimittauksiin; se poimii mitatuista signaaleista ikkunoita liipaisutapahtumien yhteydessä ja esittää niistä keskiarvoistettuja signaaleja. Sovellusohjelmassa ei sellaisenaan ole toiminnallisuutta, jolla voitaisiin esimerkiksi tuottaa mitatuista signaaleista reaaliaikainen taajuusesitys tai lähettää niitä verkon kautta palvelimelle.

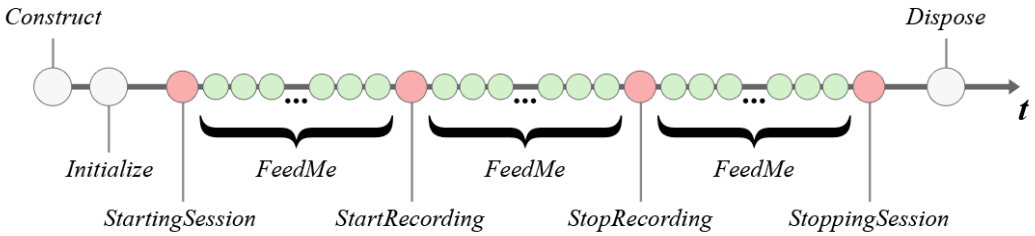
4.3 Liitännäisten rajapinta (API)

Kuten edellä (luku 4.2) esitettiin, kaikki mittaustiedon laitteelta vastaanottamista seuraava käsittely tapahtuu liitännäisissä, joita pääohjelma käyttää niille sovitun yhteisen so-

27. Mega Elektroniikka Oy ilmoitti syksyllä 2011 NeurOne-sovellusohjelmaan kohdistuvista muutoksista, joiden jälkeen liitännäisten konfiguraatio ei ole enää sidottu sovellusohjelman taulukkomuotoiseen käyttöliittymään.

vellusrajapinnan kautta. Kyseinen rajapinta on kokoelma .NET-pohjaisia ohjelmistorajapintoja ja luokkia, joihin liitännäiset pohjautuvat. Tässä luvussa tutustutaan näistä liitännäisten kehityksen kannalta keskeisimpiin. Käsiteltävän julkistamattoman materiaalin esitykseen on saatu Mega Elektroniikka Oy:n lupa.

Rajapinnan keskeisimmät osat on esitetty liitteessä 2. Kaikista tärkein osa API:a on IDataListener-rajapinta. Se määrittää millaiset olio-ohjelmointiparadigman mukaiset attribuutit *(field, property)* ja operaatiot *(method)* liitännäisen täytyy tarjota sitä kutsuvalle sovellukselle. Jokainen NeurOne-liitännäinen toteuttaa tämän rajapinnan, ja se on yksi kriteereistä, jolla pääohjelma tunnistaa käytettävissä olevat liitännäiset käynnistyessään. Rajapinnan kautta pääohjelma saa mm. tiedon, onko liitännäisellä mittauksenaikaista käyttöliittymää (HasUI-attribuutti) sekä montako kappaletta kyseisiä liitännäisiä voi olla yhtä aikaa käynnissä yhden mittauksen aikana (MaximumCount-attribuutti). Lisäksi rajapinta sisältää prototyypit liitännäisen alustus- ja prosessointikutsuille (Initialize- ja FeedMe -operaatiot) sekä operaation SendMessage, jonka kautta liitännäinen vastaanottaa pääohjelmalta ja muilta liitännäisiltä saapuvia viestejä kuten ilmoitukset mittausistunnon tai nauhoituksen alkamisesta. Näiden operaatioiden ajoitus mittausistunnon aikana on esitetty kuvassa 4.4.



Kuva 4.4. *IDataListener-operaatioiden ajoitus mittauksen aikana.*

Aluksi NeurOne luo liitännäisen ilmentymän ja kutsuu tämän jälkeen sen Initialize-operaatiota, välittäen parametrinä mm. näytteenottotaajuuden ja mittauskanavien ominaisuudet. Seuraavaksi liitännäinen saa NeurOnelta SendMessage-operaation kautta viestin StartingSession, joka ilmaisee mittausistunnon alkamista. Tämän jälkeen mittaus käynnistyy myös NeurOne-laitteistossa ja liitännäinen alkaa saada mittaustietoa FeedMe-operaatiokutsujen kautta. Kun käyttäjä käynnistää tallennuksen, saa liitännäinen viestin StartRecording ja vastaavasti tallennuksen päättyessä viestin StopRecording. Käyttäjän lopettaessa mittausistunnon välittyy liitännäiselle viesti StoppingSession, jonka saatuaan liitännäisen tulee lopettaa prosessointi. Lopuksi NeurOne vapauttaa liitännäisen ja sen varaamat resurssit kutsumalla .NET-ohjelmointimallin mukaisesti Dispose-operaatiota.

Yksittäinen FeedMe-operaatiokutsu saa tyypittömänä parametrinään joko ikkunallisen näytteitä tai yksittäisen tapahtuman. Tämä parametri muunnetaan oikeaan tietotyyppiin toisen, FeedType-parametrin perusteella (ohjelma 4.1). Näyteikkuna välitetään operaatiokutsun yhteydessä kaksiulotteisena “taulukoiden taulukkona” *(jagged array)* ja ta-

pahtumat abstraktin Event-luokan perillisinä. Kutsu palauttaa FeedMeReturnValue-tyyppisen arvon, joka ilmaisee sovellusohjelmalle onnistuiko käsittely.

```
public FeedMeReturnValue FeedMe(FeedType feedType, Object
dataAsObject ...)
{
  switch (feedType)
  {
    // EVENT: clone event into the internal buffer
    case FeedType.Event:
      Event ev = (Event)dataAsObject;
      break;

    case FeedType.Samples:
      // sampData1[0]: samples for first input
      // sampData1[0][0]: first sample of first input
      Int32[][] sampData1 = (Int32[][])dataAsObject;
      break;
  }

  return FeedMeReturnValue.AllOk;
}
```

Ohjelma 4.1. *FeedMe-operaatio.*

Esitetyn rajapinnan avulla liitännäiset kykenevät vastaanottamaan näytteitä mitatuista signaaleista sekä mittauksen aikana mahdollisesti esiintyneitä tapahtumia. Kuten kuvassa 4.4 on esitetty, alkaa liitännäinen saada mittaustietoa heti mittausistunnon alkaessa. Täten liitännäisen toiminta ei ole sidottu siihen, tallennetaanko NeurOnella signaaleja vai ei.

5 VAATIMUKSET JA RAJOITTEET

Tässä luvussa tarkastellaan aluksi, mitä konstruktiolla halutaan saavuttaa (tavoitetila); samassa yhteydessä muodostuu myös konstruktiota koskeva käsitteistö ja malli (luku 5.1). Tavoitetilan määrittelystä siirrytään tarkastelemaan reaaliaikaisuuden problematiikkaa tämän työn osalta (luku 5.2) sekä rajoitteita algoritmien välisissä kytkennöissä (luku 5.3) ja mittaustiedon verkon yli välittämisessä (luku 5.4). Luvussa muodostetaan useita premissejä, jotka toimivat sovelluksen toteutuksen perustana luvussa 6. Nämä premissit on merkitty tekstiin tunnisteella *AP.*

5.1 Tavoitetila, käsitteistö ja malli

Tarkastellaan ensin toivottua tavoitetilaa; NeurOne-järjestelmän toiminnallisuutta halutaan laajentaa *lähettämään mittaustietoa* tietoverkon kautta *palvelimelle*. Lisäksi mittaustietoa täytyy pystyä *käsittelemään* erinäisin *algoritmein* jo ennen sen lähettämistä.

Mittaustieto koostuu mitatuista *signaaleista* ja *tapahtumista*. Signaalit ovat reaaliarvoista koostuvia lukujonoja, jotka kuvaavat mitattujen analogisten signaalien arvoja säännöllisin väliajoin (näytteenottoväli). Jokainen signaali välitetään omassa *kanavassaan*. Tapahtumat ovat epäsäännöllisesti ilmeneviä, joko yhteen ajanhetkeen tai ajanjaksoon liittyviä tietorakenteita. Tapahtumia ovat mm. mittaavan hoitajan tekemät annotaatiot[28] sekä merkinnät, joilla synkronoidaan mittauslaitteiston ulkoisia signaaleja (esim. videonauhoite) mitattuihin signaaleihin.

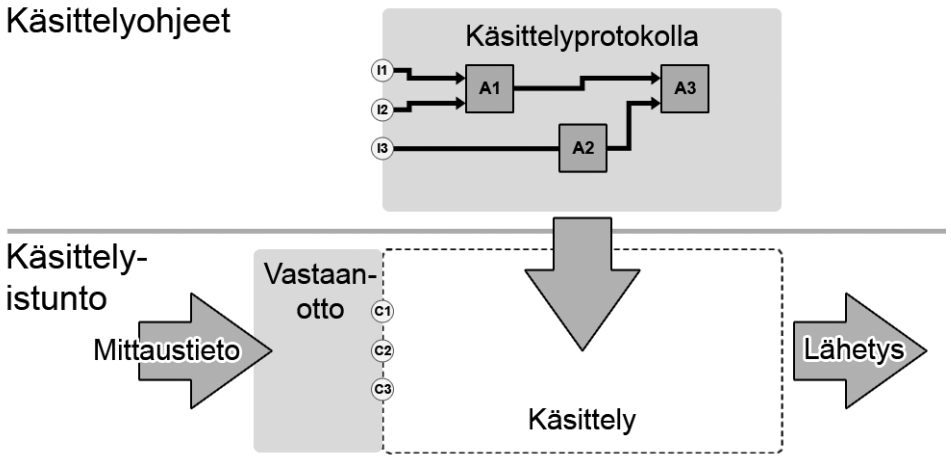
Algoritmi on selkeästi määritetty ja ohjelmistosta helposti irrotettava osa, joka tekee jotain mittaustiedon perusteella. Se voi esimerkiksi muokata mittaustietoa tai muodostaa täysin uusia signaaleja mittaustiedosta tehdyn laskennan perusteella.

Käsittely ei saa olla sovellukseen kiinteästi rakennettua, vaan käsittelyn toiminnan tulee olla käyttäjän muokattavissa. Käsittelyyn voi liittyä yksi tai useampia algoritmeja, ja yksi algoritmi voi saada syötettä usealta muulta algoritmilta. Lisäksi käsittely voi haarautua useaksi rinnakkaiseksi käsittelyketjuksi. Tässä työssä tällaisia käyttäjän muokattavissa olevia käsittelyohjeita kutsutaan *käsittelyprotokollaksi*. Käsittely perustuu aina käsittelyprotokollaan – toisin sanoen noudattaa aina ennalta määrättyjä ohjeita.

Mittauksen alkaessa käynnistyy siihen liittyvä *käsittelyistunto*, joka koostuu mittaustiedon vastaanottamisesta mittauslaitteelta, mittaustiedon käsittelystä käsittelyprotokollan perusteella ja tiedon lähettämisestä eteenpäin (kuva 5.1). Käsittelyistunto on osa laajempaa, johdannossa kuvattua signaalinkäsittelyketjua (potilas–mittalaite–mittausohjelmisto–lähetys–palvelin–toimitus–katselin) ja realisoituu mittauspaikalla toimivassa mit-

28. Tekstimuotoiset kommentit, jotka liittyvät johonkin tapahtumaan mittauksessa, kuten “potilas avasi silmänsä”, “valot sammutettu” tai “ärsyke annettu”.

tausohjelmistossa (kuva 1.1).



Kuva 5.1. *Tavoitetilan malli; käsittelyprotokollan liittyminen käsittelyistuntoon.*

Sovelluksen käyttötilanne voi edetä esimerkiksi näin:

> Lääkäri päättää potilaan tilasta riippuen, millä tavoin potilasta monitoroidaan; hänellä on tieto käytettävissä olevista analyysialgoritmeistä sekä suosituksista, millaisia trendejä signaaleista kannattaa seurata tietyn diagnoosin yhteydessä ja mitä ko. trendit ilmaisevat. Yhdessä hoitajan ja sairaalafyysikon kanssa he määrittävät käytettävän mittaus- ja käsittelyprotokollan. Tämän tiedon perustella sairaalafyysikko valmistelee mittausjärjestelmän sekä siirtyy konfiguroimaan tässä työssä toteutettua toiminnallisuutta. Sairaalafyysikko luo uuden käsittelyprotokollan (tai muokkaa olemassa olevaa), valmistaen sen vastaanottamaan mitatut kanavat ja kytkemällä nämä kanavat algoritmeihin. Aluksi hän syöttää mitatut kanavat algoritmille, joka toteuttaa verkkohäiriötä poistavan notch-suotimen ja tämän tuottaman suodatetun signaalin edelleen algoritmille, joka leikkaa signaalista halutun kokoisia ikkunoita tiettyjen tapahtumien yhteydessä. Lopuksi hän määrittää alipäästösuodatetun signaalin tallennettavaksi paikalliseen tiedostoon ja ikkunoidun signaalin lähetettäväksi tietoverkon yli palvelimelle. Hoitaja valmistelee potilaan mittausta varten ja käynnistää mittauksen. Samalla käynnistyy myös sairaalafyysikon edellä määrittämään käsittelyprotokollaan pohjautuva käsittelyistunto.

Tietoverkolla tarkoitetaan tämän työn yhteydessä pakettikytkentäistä IP-pohjaista *(Internet Protocol)* tiedonsiirtoverkkoa ja palvelimella tässä verkossa sijaitsevaa toimijaa, joka kykenee vastaanottamaan lähetettyä mittaustietoa.

Tavoitetilan toteutumisella on vaikutuksia usealle alueelle. Potilasmonitoroinnissa otetaan askel kohti etätyöskentelyä; yhdistettynä työn tulokset Girf Oü:n toteuttamaan palvelinratkaisuun, voidaan mitatut ja käsitellyt signaalit jakaa verkkoon reaaliajassa. Lääkäri voi tarkastella signaaleja ja/tai niistä laskettuja trendejä työasemalta tai PDA-laitteelta *(Personal Data Assistant)* sairaalassa – tai ehkä jopa kotoaan, edellyttäen että

käytetyn verkkoyhteyden tietoturva on kunnossa.

Algoritminkehittäjien on mahdollista tuottaa ja kaupallistaa käsittelyalgoritmeja laitteistosta riippumattomana. Algoritmien toteutus ja testaaminen eivät vaadi edes perinteisten ohjelmointikielien osaamista, vaan kehitys voidaan tehdä suoraan MATLAB-skripteillä, jotka ovat osa algoritminkehittäjien luontaista toimintaympäristöä. Näin kehitettyjen algoritmien tuloksista saadaan uutta tutkimustietoa. Algoritmille voi saada sertifikaatin kun se on riittävästi testattu, ja sertifioituja algoritmeja voidaan käyttää myös potilaan diagnosoinnissa sekä hoidon suunnittelussa. On kuitenkin syytä huomioida, ettei tässä työssä toteutettava sovellus ole sertifioitu. Täten myös sen yhteydessä käytettyjen algoritmien tuloksia ei tule käyttää potilaaseen vaikuttavassa päätöksenteossa.

Lisäksi Mega Elektroniikka Oy voi tarjota työssä toteutettua ohjelmistokomponenttia NeurOne-sovelluksen kylkiäisenä, laajentaen sen käytettävyyttä tutkimuksissa. Tämä voi olla ratkaiseva tekijä, jos NeurOne-sovellus ei sellaisenaan täytä potentiaalisen ostajan mittausratkaisulle asettamia vaatimuksia (esim. mittaustietojen reaaliaikainen lähetys palvelimelle). NeurOne-mittausratkaisun käyttö tutkimuksissa puolestaan antaa positiivista PR-arvoa Mega Elektroniikka Oy:lle, lisäten mittausratkaisun kysyntää.

5.2 Reaaliaikaisuus

Ennen kuin ryhdytään käsittelemään reaaliaikaisuuden asettamia vaatimuksia, täytynee määrittää mitä reaaliaikaisuudella tämän työn yhteydessä tarkoitetaan. Yleisessä merkityksessä järjestelmää voidaan pitää reaaliaikaisena, jos se reagoi syötteeseen rajoitetun ajan sisällä. Signaalinkäsittelyn yhteydessä tätä voidaan vielä tarkentaa: kyseessä on laskentajärjestelmä, jossa laskennalliset tavoitteet pyritään saavuttamaan määritetyssä aikakehyksessä. [77, luku 1]

Buttazzo et al antavat tästä kirjassaan havainnollisen esimerkin:

> “Jos robotti etenee tiettyä nopeutta ja havaitsee radallaan esteen, täytyy sen kyetä jarruttamaan tai vaihtamaan suuntaansa (robotin nopeudesta ja esteen etäisyydestä riippuvan) maksimiviiveen sisällä. Muutoin robotti ei kykene väistämään estettä, aiheuttaen kolarin”. [77, s. 2]

Reaaliaikaisuus voidaan vaatimuksista riippuen jakaa kovaan ja pehmeään reaaliaikaisuuteen. Kovan reaaliaikaisuuden järjestelmissä laskennalliset tavoitteet on saavutettava aina annetun aikakehyksen sisällä; yksittäinenkin myöhästyminen voi johtaa katastrofaalisiin seurauksiin. Esimerkiksi prosessoinnin myöhästyminen sydämentahdistimessa voi aiheuttaa vakavia henkilövahinkoja. Monesti prosessointiviiveestä halutaan vakio, jotta järjestelmän ajallinen käyttäytyminen on determinististä. [77, luku 1]

Reaaliaikaisiin, näyte näytteeltä prosessoiviin digitaalisuotimiin liittyen Kuo, Lee ja Tian summaavat ongelman hyvin:

> “Reaaliaikaisen järjestelmän yhden näytteen käsittelyssä ei saa kulua enemmän aikaa, kuin näytteenottoväli on.” [78, s. 16]

Siinä missä kova reaaliaikainen järjestelmä joko toimii tai epäonnistuu, tulkitaan pehmeiden reaaliaikaisten järjestelmien toimintaa eri valossa; pehmeissä järjestelmissä ei ole suoraa epäonnistumista, vaan tuloksen laatu heikkenee sulavasti vasteajan kasvaessa. Prosessointiin kulutetun ajan (käsittelyaika) täytyy olla keskimäärin sallituissa rajoissa. Jos tämä ehto ei toteudu, tietoa joko häviää tai kasautuu järjestelmässä oleviin puskureihin. Ajan myötä järjestelmän ulostulo jää yhä enemmän jälkeen syötteestä, ja kasautuneen tiedon määrä kasvaa lineaarisesti. Järjestelmän tulosten viivästyessä huononee myös niiden laatu. [77, luku 1]

Toisaalta järjestelmä voi ehtiä käsittelemään tietoa, mutta signaalit viivästyvät käsittelyn aikana kiinteän ajanjakson verran. Algoritmi voi esimerkiksi tarvita yhden ulostulokanavan näytteen laskemiseen sata näytettä sisääntulosta. Koska järjestelmä ei voi nähdä tulevaisuuteen (kausaalisuus), on sen odotettava saavansa loput 99 näytettä. Tästä aiheutuu 99 näytteen mittainen viive, johon voidaan vaikuttaa ainoastaan algoritmin suunnittelussa – sitä ei voida kompensoida muualla sovelluksessa. Tämä on esimerkki puhtaasti algoritmin asettamasta reaaliaikaisuusrajoitteesta, josta käytetään tässä työssä termiä *algoritmiviive*.

Tämän työn kannalta reaaliaikaisuusvaatimukset kumpuavat koko signaalinkäsittelyketjusta tiedon mittauksesta sen esitykseen (kuva 1.1). Kerätyt signaalit on määrä käsitellä ja esittää reaaliajassa sekä taltioida myöhempää käsittelyä tai tarkastelua varten. Muita reaaliaikaisuusvaatimuksia ei käytännössä ole. Työn tapauksessa reaaliaikaisuudella on siis väliä vain ihmisten kannalta; riittää, että mittaustieto ja siitä johdetut tulokset ovat läpäisseet järjestelmän *riittävän nopeasti* – toisin sanoen niiden täytyy olla tarkasteltavissa ajallisesti suhteellisen lähellä mittaushetkeä. Vaadittu suhteellinen läheisyys puolestaan riippuu mittaus- ja tarkastelupaikkojen välisestä etäisyydestä ja mittausjärjestelyn käyttötarkoituksesta. Esimerkiksi EP-tutkimuksissa herätepotentiaalin olisi hyvä päivittyä mittauspaikalla näytölle korkeintaan muutaman sekunnin viiveellä. Jos mittaustietoa katsotaan etänä (esim. jossain muualla sairaalassa olevalta KNF-työasemalta) voi viive olla pidempikin.

Signaalien kulku sovelluksen läpi voidaan jakaa kolmeen ongelma-alueeseen: vastaanotto mittauslaitteistolta, käsittely algoritmeilla ja lähetys (kuva 5.2). Tiedon siirto osa-alueiden välillä on yksisuuntaista. Signaali ei esimerkiksi voi päätyä sovelluksen sisällä vastaanottovaiheeseen sieltä lähdettyään. Yllä kuvattu jako helpottaa ongelmien hahmotusta sekä yksinkertaistaa sovelluksen suunnittelua ja toimintaa.

Kuva 5.2. *Ongelma-alueiden jako.*

Erottelun myötä hahmottuu myös uusia ongelmia; kuinka tulisi toimia tapauksessa, jossa käsittelyaika kasvaa niin paljon, ettei vastaanotettua mittaustietoa ehditä käsittelemään reaaliajassa? Vastaava tilanne voi ilmetä myös lähetyksessä, jos käytössä olevan

verkkoyhteyden kapasiteetti ei riitä. Jokainen näistä alueista sisältää reaaliaikaisuuden kannalta omat rajaehtonsa ja yhteen ketjutettuna ne muodostavat rajoitteet sille, kuinka reaaliaikaiseen signaalien esitykseen järjestelmällä päästään.

Työn kannalta oleellisimpia reaaliaikaisuuteen vaikuttavia tekijöitä ovat käsittelyn yhteydessä tapahtuva ikkunointi (luku 5.2.1) sekä algoritmien prosessointivaatimukset ja mahdollisesti edellyttämä puskurointi (luku 5.2.2). Lisäksi mittaustiedon siirrossa mittauspaikalta palvelimelle ja edelleen palvelimelta tilaajille voi ilmetä tietoverkosta johtuvaa viivettä (luku 5.2.3). Huomattakoon myös, että tässä työssä toteutettu käsittelyohjelmisto toimii Microsoft Windows -käyttöjärjestelmän päällä. Koska Windows ei ole reaaliaikakäyttöjärjestelmä, ei ole taattu, että mikään sovelluksen säie saa riittävästi prosessoriaikaa [79, s.103]. Tämä ongelma voi ilmetä suorituskyvyn äärirajoilla, jolloin tietokoneen laitteistoresursseista (suoritinaika, keskusmuisti tai muistinsiirtoväylän leveys) alkaa olla pula.

5.2.1 Näytteiden käsittelyn rajoitteet

Tiedonkäsittelyjärjestelmät voidaan kategorisoida *offline-* ja *online-järjestelmiin*. Offline-järjestelmä saa kerralla syötteenään kaiken tarvitsemansa tiedon. Esimerkiksi MATLAB-algoritmi, joka käyttää syötteenään tiedostoa – johon ei enää kirjoiteta tietoa ja siten se sisältää algoritmin syötteen kokonaisuudessaan – on offline-järjestelmä. Online-järjestelmille syötetään tietoa ikkunallinen kerrallaan, eikä järjestelmälle tulevaa tietoa välttämättä tiedetä ennalta. Järjestelmän reaaliaikaisuus riippuu välillisesti siitä kuinka suuri ikkuna tietoa kerätään ennen kuin se syötetään algoritmille.

Oletetaan, että halutaan käsitellä sekvenssi, jonka pituus on N näytettä. Valitaan algoritmille syötettävän ikkunan kooksi W näytettä. Jos $W = N$, vastaa tilanne edellä kuvattua offline-prosessointia, jossa algoritmi saa kaiken käsiteltävän tiedon kerralla. Toinen ääripää, $W = 1$, vastaa online-prosessoinnin reaaliaikaisinta muotoa, missä algoritmille syötetään yksi näyte kerrallaan.

Tarkastellaan ohjelmistolla toteutettujen algoritmien toimintaa. Seuraavassa tarkastelussa termillä *kustannus* tarkoitetaan käytettyä aikaresurssia. Yksittäisen näyteikkunan käsittelyn kustannus C_W koostuu algoritmin *kutsumiskustannuksesta* $C_C(W)$ ja algoritmista riippuvasta näytteiden *laskentakustannuksesta* $C_S(W)$:

$$C_W = C_C(W) + C_S(W) \tag{5.1}$$

Kutsumiskustannus $C_C(W)$ aiheutuu tiedon siirtämisestä algoritmin käsittelyä varten sekä kaikesta ohjelman suorituksesta ennen ja jälkeen varsinaista algoritmin tietoa käsittelevää ohjelmakoodia. Esimerkiksi kutsuttaessa natiivikoodia *(unmanaged code)* hallitusta *(managed)* .NET-koodista, täytyy .NET-ympäristön tietyissä tapauksissa kopioida kutsussa välitettävää tietoa ulos .NET-ympäristön muistista sellaiselle muistialueelle, jota natiivikoodin voi antaa käsitellä *(interop marshaling)*. Käsittelyn jälkeen tulokset on kopioitava takaisin hallitun tilan muistiin. Lisäksi algoritmin ja sitä kutsuvan ohjelman välillä voi olla useita eristyskerroksia, joiden läpi kulkeminen kasvattaa kus-

tannusta. [80]

Jos laskentakustannus $C_S(W)$ on positiivisesti sidonnainen ikkunan kokoon W, nousee kutsumiskustannus dominoivaksi tekijäksi ikkunakoon pienentyessä:

$$W \to 1 \;\Rightarrow\; C_S(W) \ll C_C(W) \tag{5.2}$$

Toisin sanoen suurin osa ajasta menee algoritmin kutsumiseen, ei tiedon käsittelyyn itse algoritmissä. Reaaliaikaisuuden ja käsittelykustannuksen optimaalinen ikkunakoko löytyy jostain edellä mainittujen ääripäiden välistä:

$$W \in \mathbb{N}^+ \wedge W \in (1,\, N) \tag{5.3}$$

Käytännössä ikkunakoon ylärajan asettaa se, kuinka reaaliaikaista käsittelystä halutaan. Alarajaan vaikuttaa edellä esitetty kutsumiskustannuksen ja laskentakustannuksen suhde. Lisäksi on huomioitava, että ikkunassa täytyy olla kokonaisluvullinen määrä näytteitä (lauseke 5.3). Jos ikkunoita muodostetaan tasaisin aikavälein T_W kanavista, joiden näytteenottotaajuudet ovat $F_0 \ldots F_n$, täytyy aikaväli valita siten, että ikkuna sisältää kokonaisluvullisen määrän näytteitä jokaisesta kanavasta:

$$T_W \cdot F_i \in \mathbb{N}^+ \wedge i \in [0,n] \tag{5.4}$$

Yleisesti signaalinkäsittelyssä näytteenottotaajuudet esitetään hertseinä (Hz), tarkoittaen yhden sekunnin aikana otettujen näytteiden määrää. Kokonaisluvulliset näytteenottotaajuudet ovat signaalinkäsittelyssä hyvin yleisiä. Myös tässä työssä käytetyssä NeurOne-ympäristössä kanavien näytteenottotaajuudet ovat aina kokonaislukuja. Näille järjestelmille pienin taattu lausekkeessa 5.4 esitetyn ehdon täyttävä ikkunakoko on yksi sekunti. Monissa tilanteissa näytteenottotaajuuksista voi löytyä yhteisiä tekijöitä, jolloin ikkunakokoa voidaan tästä vielä pienentää. Tästä saadaan ensimmäinen sovellusta koskeva vaatimus (premissi):

(AP1) Näytteenottotaajuuksien täytyy olla vakioita ja kokonaislukuja.

Ikkunoinnin myötä saattaa ilmentyä tilanteita, joissa algoritmi tarvitsee laskennassa näytteitä edellisistä ikkunoista. Käytännössä tämän ratkaisemiseen on kaksi keinoa: joko algoritmien on itse säilöttävä saamansa ikkunat myöhempää käyttöä varten tai algoritmeille on tarjottava keino kysyä aiempia ikkunoita ympäröivältä sovellukselta. Kummassakin lähestysmistavassa on hyvät ja huonot puolensa; jos jokainen algoritmi säilöö tarvitsemaansa tietoa, on sama tieto mahdollisesti tallennettuna useaan paikkaan, lisäten virtuaalimuistille aiheutuvaa kuormaa. Jos ympäröivä sovellus velvoitetaan varastoimaan mennyttä tietoa, joudutaan vastaamaan kysymyksiin kuten *kuinka pitkän ajan tietoa säilötään* sekä *kuinka algoritmit pyytävät tietoa sovellukselta*. Säilöntäajan pituus riippuu algoritmeistä, joten algoritmien ulkoista muistia varten täytyy mahdollisesti toteuttaa mekanismi, jolla algoritmi voi ilmoittaa tämän sovellukselle sekä pyytää säilöttyjä ikkunoita. Kyseisen mekanismin toteuttaminen monimutkaistaa algoritmien ja niitä kutsuvan sovelluksen rajapintaa, vaikeuttaen algoritmien kehitystä. Koska suhteessa

harva algoritmi tarvitsee näytteitä pitkältä aikaväliltä, voidaan jättää näytteiden varastointi algoritmien vastuulle. Algoritmin tekijä voi täten itse päättää mitä varastoi ja kuinka pitkäksi aikaa. Algoritmeille asetetuiksi premisseiksi saadaan:

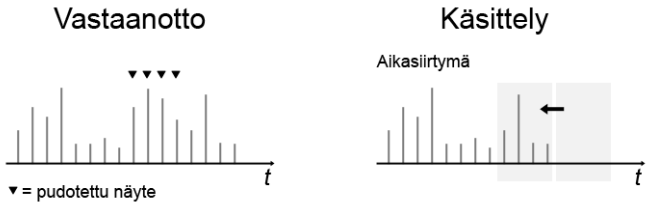
(AP2) Algoritmeille syötetään tietoa ikkuna kerrallaan (online-järjestelmä).
(AP3) Ikkuna sisältää kokonaisluvullisen määrän näytteitä.
(AP4) Algoritmi on kausaali, toisin sanoen se ei tiedä tulevia ikkunoita.
(AP5) Jos algoritmin täytyy viitata tietoon edellisissä ikkunoissa, on algoritmin itsensä huolehdittava ko. tiedon säilömisestä.

Tarvittaessa edellä esitetty premissi (AP1) voidaan rikkoa, jolloin voidaan käsitellä myös ei-kokonaisluvullisia näytteenottotaajuuksia. Jos tämän yhteydessä käytetään sekunnin pituisia ikkunoita, tulee ikkunan ajallisesta pituudesta T_W vaihteleva, minkä johdosta ikkunoissa olevien kanavien keskinäinen synkronointi ja ikkunoiden varastointi hankaloituvat. Toinen vaihtoehto on kasvattaa ikkunan ajallista pituutta siten, että lauseke 5.4 tulee tyydytetyksi myös vakiolla ikkunanpituudella.

5.2.2 Algoritmien asettamat rajoitteet

Käsittelyvaiheessa reaaliaikaisuutta rajaa käytettyjen algoritmien vaatima prosessointiteho suhteessa laitteistoalustaan. Vaadittu prosessointiteho puolestaan on riippuvainen mitattujen signaalien kaistanleveydestä (kaava 4.1). Lähetysvaiheessa rajaavia tekijöitä ovat lähetettävien signaalien kaistanleveys sekä mahdolliset signaalien pakkaukseen liittyvät prosessointivaatimukset suhteessa laitteistoalustaan.

Pyrittäessä reaaliaikaisuuteen, on eräs ratkaisu yksinkertaisesti jättää käsittelemättä se, mitä ei ehditä käsitellä, toisin sanoen pudottaa näytteitä (kuva 5.3). Käsittelemättä jättäminen on esityksen kannalta ongelmallista; kuinka usein ja kuinka suuria aikasegmenttejä signaaleista voidaan hylätä, kunnes perille toimitettu tieto ei enää hyödytä? Mittausta etänä katsellessa voi olla hyväksyttävää, että esimerkiksi sekunnin pituinen segmentti jää uupumaan signaalista korkeintaan joka kymmenes sekunti. Toisaalta juuri pudotettu segmentti voi sisältää kriittistä tietoa. Hyväksyttävät rajat riippuvat pitkälti mittauksen tarkoituksesta.



Kuva 5.3. *Näytteiden hylkääminen.*

Myös algoritmien kannalta näytteiden pudottaminen on ongelmallista; algoritmien täytyy olla varautuneita kyseiseen tapahtumaan, mikä nostaa niiden monimutkaisuutta vaikeuttaen kehitystä. Jos kanavista pudotetaan ajallisesti eri määrä tietoa, ne ajautuvat

epäsynkroniaan. Toisaalta näytteitä voidaan hylätä koko ikkunallinen synkroniaa rikkomatta, sillä premissien (AP1), (AP2) ja (AP3) johdosta jokaisen kanavan ensimmäinen näyte on ikkunassa samalta ajanhetkeltä. Näytteiden pudottaminen voi aiheuttaa signaaleihin epäjatkuvuuksia, joista voi aiheutua (käytetyistä algoritmeistä riippuen) artefakteja jatkokäsittelyn tuloksiin. Lisäksi tiedon siitä, että välistä on pudotettu näytteitä, täytyy kulkea aina mittaustiedon mukana, jotta myöhemmin vältettäisiin tiedon virheellinen tulkinta – myös sitä esitettäessä.

Näytteiden pois pudottamisen voidaan tulkita myös vastaavan sitä, että signaalien näytteenottotaajuus muuttuu hetkellisesti hyvin pieneksi. Esimerkiksi lineaaristen aikariippumattomien (LTI) suotimien toiminta edellyttää, että signaalia on näytteistetty tasaisin väliajoin [81, s. 38]. LTI-suotimet ovat ehkä yleisin suodatintyyppi signaalinkäsittelyssä [1, s. 319]. EEG:n yhteydessä niitä käytetään yleensä esimerkiksi sähköverkon aiheuttaman 50/60 Hz häiriön poistamiseksi signaalista tai vähentämään EMG-artefaktojen osuutta alipäästösuodatuksella. [32, luku 3.2.3]

Näytteiden käsittelemättä jättäminen siis vaikuttaa myös niitä seuraavien näytteiden käsittelyyn ainakin LTI-järjestelmissä. Jos näytteitä jätetään käsittelemättä, eivät LTI-tyyppiset käsittelyalgoritmit toimi oikein, mikä puolestaan monimutkaistaa mittaustiedon tulkintaa ja myöhempää käsittelyä. Lisäksi tämän salliminen vaikeuttaa algoritmien kehitystä. Tästä saadaan ensimmäinen ehto signaalien käsittelylle sovelluksessa:

(AP6) Näytteiden käsittelyn täytyy tapahtua ajallisesti jatkuvassa järjestyksessä siten, että ikkunaa W_n ennen käsitellään ikkunat W_k, $k < n$, eikä välistä saa jättää pois näytteitä.

5.2.3 Tiedonsiirron rajoitteet

Edellä johdettujen premissien mukaan tietoa käsitellään ikkunoittain (AP2), jotka seuraavat toisiaan ajallisesti jatkuvassa järjestyksessä (AP6). Ajallisen jatkuvuuden vaatimus ei ole niin ehdoton siirrettäessä tietoa palvelimelle. Jos tietoverkon kapasiteetti ei riitä tiedon reaaliaikaiseen välitykseen, voidaan välitettävää tietoa priorisoida esimerkiksi antamalla uusimmalle tiedolle lähetysprioriteetti vanhemman yli. Näin tehtäessä joutuvat palvelimella toimivat algoritmit odottamaan, että yllä kuvattu ajallinen jatkuvuusehto (AP6) toteutuu ennen kuin ne voivat prosessoida signaaleja. Tässäkin lähestymistavassa on ongelmansa. Tarkastellaan esimerkiksi seuraavaa tilannetta:

> Mittauspaikka tuottaa välitettävää tietoa tasaista tahtia, u MB/s. Verkon kapasiteetti on riittämätön, esimerkiksi muun liikenteen aiheuttaman kuormituksen takia, ja efektiivinen maksimisiirtonopeus on tämän sovelluksen kannalta puolet vaaditusta: $0{,}5 \cdot u$ MB/s. Jos tiedon lähetyksessä uusimmalla tiedolla on korkeampi prioriteetti ja tieto kuljetetaan lähetysikkunoissa[29] (jotka eivät välttämättä vastaa edellä mainittuun käsittelyyn liittyviä ikkunoita), päätyy optimitapaukses-

29. Lähetysikkunalla tarkoitetaan tässä OSI-kerrosmallin sovellustasolle sijoittuvaa ikkunaa, ei esimerkiksi TCP/IP-kerroksen (tasot 3...5) tai Ethernet (taso 2) ikkunointia.

sa palvelimelle reaaliaikaisesti vain joka toinen ikkuna. Lähettämättömät ikkunat kerääntyvät mittauspaikalla sijaitsevaan puskuriin odottamaan ettei lähetettävänä ole uudempaa tietoa. Jos palvelimella toimii algoritmeja, jotka tarvitsevat prosessoinnissaan esimerkiksi kahden peräkkäisen ikkunallisen verran tietoa, on palvelimen prosessointi käytännössä pysähtynyt.

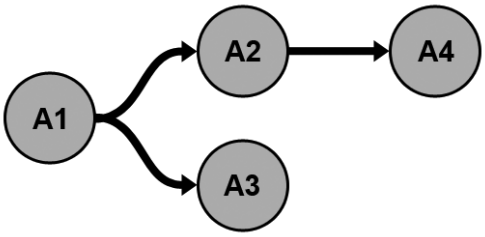
Ongelmaan on useita ratkaisuja; verkko voidaan konfiguroida priorisoimaan tähän sovellukseen liittyvät paketit esimerkiksi QoS-provisioinnin *(Quality Of Service)* [82, luku 13.3] avulla, jolloin verkkoa mahdollisesti ruuhkauttava muu liikenne ei ole ongelma. Toinen vaihtoehto on tehdä lähetettävistä ikkunoista niin suuria, että palvelimella toimivat algoritmit pystyvät prosessoimaan vastaanottamaansa tietoa. Tämä kuitenkin aiheuttaa edelleen ongelmia lähetysikkunoiden reunoilla, eli niissä kohdissa, joissa lähetettävä signaali katkeaa ajallisesti. Lisäksi tiedonvälityksen viive nousee lähetysikkunoiden pituuden kasvaessa. Kolmas vaihtoehto on, että sovellusta käyttävä organisaatio takaa verkkokapasiteetin, ja on sen kesken loppuessa valmis kärsimään reaaliaikaisuusvaatimuksen loukkauksen (esim. biopalautteeseen liittyvän tutkimusasetelman epäonnistumiseen).

5.3 Prosessointiverkon luonne

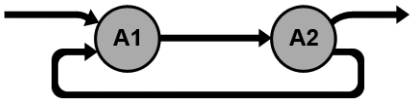
Edellä (luku 5.1) määritettiin, että käsittelyistuntoon saatu mittaustieto syötetään käyttäjän käsittelyprotokollaan määrittämille algoritmeille. Algoritmit voivat olla kytkettynä toisiinsa esimerkiksi siten, että algoritmin A1 tulokset syötetään algoritmeille A2 ja A3, sekä algoritmin A2 tulokset edelleen algoritmille A4. Yhdessä nämä algoritmit muodostavat *prosessointiverkon*, jossa jokaista algoritmia voidaan kuvata solmulla *(vertex)* ja kahden algoritmin välistä kytkentää (yksi tai useampia kanavia) kaarella *(edge)* (kuva 5.4). Verkko on suunnattu, eli kaikkia siinä olevia kaaria voi kulkea vain yhteen suuntaan. Tämä kuvaa hyvin algoritmien riippuvuussuhteita; kaari A1:sta A2:een kertoo, että A2 on riippuvainen A1:n tuloksista. Täten A1:n on prosessoitava tieto ennen A2:a. A1:n tulokset puolestaan riippuvat prosessiin syötettävästä mittaustiedosta.

5.3.1 Algoritmien väliset takaisinkytkennät

Ennen kuin kyseisen verkon luonnetta määritetään tarkemmin, tarkastellaan algoritmien välisiä takaisinkytkentämahdollisuuksia. Otetaan esimerkiksi tilanne, jossa käsittelyprotokollaan on konfiguroitu algoritmit A1 ja A2 (kuva 5.5). Vastaanotettu mittaustieto syötetään algoritmille A1, jonka ulostulo ohjataan algoritmille A2. Lisäksi piirissä on takaisinkytkentä algoritmin A2 ulostulosta algoritmin A1 sisääntuloon. Algoritmi A1 on riippuvainen algoritmin A2 tuloksista, jotka puolestaan riippuvat algoritmin A1 tuloksista.



Kuva 5.4. *Esimerkki prosessointiverkosta.*



Kuva 5.5. *Algoritmien välinen takaisinkytkentä.*

Takaisinkytkennät ovat tietyissä rajoin mahdollisia; niiden kautta voidaan syöttää algoritmille edellisten prosessoitujen ikkunoiden tuloksia. Luvussa 5.2.1 määritetyn premissin (AP2) myötä takaisinkytkennän minimiviiveeksi muodostuu käsittelyssä käytettävä ikkunan pituus. Algoritmin A1 on ensin käsiteltävä saamansa ikkuna, jonka jälkeen tulokset voidaan ohjata algoritmille A2. Algoritmi A1 voi saada A2:n tulokset käyttöönsä vasta prosessoidessaan seuraavaa ikkunaa. Tämä jättää silti auki kysymyksen, mitä algoritmin A1 pitäisi saada syötteenään prosessoidessaan ensimmäistä ikkunaa, jolloin algoritmin A2 edellisen prosessointikierroksen aikana tuottamia tuloksia ei ole vielä saatavissa. Lisäksi on syytä huomioida, että pienin taattu käsittelyikkunan pituus on yksi sekunti (luku 5.2.1). Monilla algoritmeilla tämä viive voi olla liian suuri. Ikkunaa pienentämällä takaisinkytkennän viivettä voitaisiin laskea, mutta tällöin kutsumiskustannus kasvaa suhteessa laskentakustannukseen (kaava 5.2). Voidaan kysyä, kuinka moni algoritmi tällaisesta takaisinkytkennällisyydestä hyötyisi.

Useat seikat tukevat algoritmien välisten takaisinkytkentöjen poissulkemista; ensinnäkin edellä esitetty ikkunointiin liittyvä rajoitus pätee vain algoritmien väliseen tiedonsiirtoon. Algoritmin kehittäjä voi rakentaa haluamansa takaisinkytkennät toteuttamansa algoritmin sisälle. Toiseksi ei ole viitteitä siitä, että takaisinkytkennät olisivat sovellusalueen kannalta erityisen tarpeellisia – on pikemminkin ominaista, että signaalit kulkevat putkimaisissa työnkuluissa. Tästä saadaan prosessointiverkon luonnetta rajoittava ehto:

(AP7) Algoritmien väliset takaisinkytkennät eivät ole sallittuja.

5.3.2 Verkon jakaminen käsittelyvaiheisiin

Jos prosessointiverkossa ei sallita takaisinkytkentöjä se muuttuu luonteeltaan asykliseksi. Tämä tarkoittaa käytännössä sitä, että jos verkkoa lähdetään kulkemaan solmusta *V*, ei voida päätyä takaisin samaan solmuun *V*. Tämän luvun alussa esitettiin, että verkko on suunnattu. Näiden ominaisuuksien ansiosta prosessointiverkko kuuluu erääseen hyvin tutkittuun verkkojen luokkaan: asyklisiin suunnattuihin verkkoihin (*Acyclic Directed Graph*, ADG). ADG-verkoilla on tämän sovelluksen kannalta eräitä hyödyllisiä ominaisuuksia: ensinnäkin verkossa on aina yksi tai useampia alkusolmuja sekä yksi tai useampia päätesolmuja. Alkusolmut ovat sellaisia solmuja, joihin ei saavu yhtäkään kaarta ja päätesolmut sellaisia, joista ei lähde yhtäkään kaarta. Nämä edustavat proses-

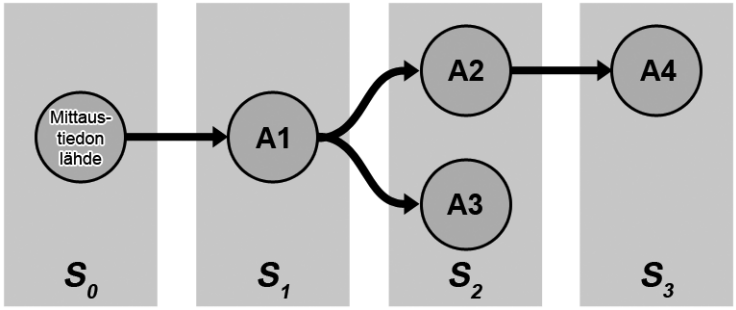
sointiverkon ensimmäisiä ja viimeisiä prosessointialgoritmeja; ensimmäiset algoritmit (A1 kuvassa 5.4) eivät riipu minkään toisen algoritmin tuloksista, ja mikään algoritmi ei ole riippuvainen viimeisten algoritmien (A3 ja A4 kuvassa 5.4) tuloksista. Lisäksi verkon solmut voidaan asettaa järjestykseen (topologinen järjestys), missä kuhunkin solmuun tulee kaaria vain sitä edeltäneistä solmuista. Toisin sanoen algoritmit, joita verkon solmut kuvaavat, voidaan järjestää niiden keskinäisen riippuvuuden mukaan. [83, s. 13–15; 84, s. 6]

Suunnatun verkon *D* voidaan matemaattisesti esittää koostuvan solmuista $V(D)$ sekä niitä yhdistävistä suunnatuista kaarista

$$A(D) = \{(x, y) : x \in V(D) \wedge y \in V(D) - \{x\}\}, \tag{5.5}$$

jossa kaaren suunta on lähtösolmusta *x* kohdesolmuun *y.* Kaarien määritelmä sallii myös useita samansuuntaisia kaaria solmusta *x* solmuun *y,* muttei kaarta, jonka lähtösolmu olisi samalla sen kohdesolmu. Prosessointiverkossa kaaret vastaavat algoritmien välillä kytkettyjä kanavia.

Topologisessa järjestyksessä verkon *D* solmut $V(D) = \{v_1, v_2, \ldots, v_n\}$ on järjestetty siten, että kaikki kaaret toteuttavat ehdon $\forall(v_i, v_j) \in A(D) \Rightarrow i < j$. Toisin sanoen solmut on mahdollista asettaa järjestykseen, jossa solmusta lähtee kaaria vain sitä seuraaviin solmuihin. Esimerkiksi kuvan 5.4 verkolla on kolme mahdollista topologista järjestystä: $(A1, A2, A3, A4)$, $(A1, A2, A4, A3)$, sekä $(A1, A3, A2, A4)$. Tämän järjestyksen pohjalta verkko voidaan jakaa peräkkäisiin vaiheisiin $S(D) = (S_1, S_2, \ldots, S_n)$, joista jokainen vaihe S_k riippuu vain sitä edellisten vaiheiden $S_0 \ldots S_{k-1}$ tuloksista (kuva 5.6). Kuvaan on merkitty myös ensimmäinen vaihe S_0, mikä ei sisällä algoritmeja, vaan toimii käsiteltävän tiedon lähteenä. [83, s. 13–15]



Kuva 5.6. *Prosessointiverkon vaihejako.*

Vaihejako määrää, missä järjestyksessä prosessointiverkon algoritmeja on kutsuttava. Esimerkiksi vaiheen S_2 algoritmit voivat käsitellä tietoa vasta kun vaiheen S_1 algoritmit ovat kyseisen tiedon niille tuottaneet. Kuvan 5.4 verkosta voidaan muodostaa kaksi vaihejärjestystä: $(\{A1\}, \{A2, A3\}, \{A4\})$ ja $(\{A1\}, \{A2\}, \{A3, A4\})$. Algoritmin A3 kannalta ei siis ole väliä suoritetaanko se vaiheessa S_2 vai S_3, sillä mikään algoritmi ei riipu sen tuloksista. Määrätään tämän vuoksi lisäehto: algoritmi kuuluu vaiheeseen k jos ja vain jos sillä on riippuvuuksia vaiheeseen $k-1$. Kun tämä ehto otetaan huo-

mioon, voidaan kuvan 5.4 verkosta muodostaa vain vaihejärjestys $\left(\{A1\}, \{A2, A3\}, \{A4\}\right)$ (kuva 5.6). Vaihejaon mahdollisuus on premissi sovelluksen toteutukselle:

(AP8) Prosessointiverkko voidaan jakaa peräkkäisiin vaiheisiin $\left(S_1, S_2, \ldots, S_n\right)$, joista jokainen vaihe S_k riippuu vain sitä edellisten vaiheiden $S_0 \ldots S_{k-1}$ tuloksista.

5.3.3 Rinnakkainen käsittely

Fyysisistä rajoitteista johtuen suorituskykyä ei voida kasvattaa loputtomasti yhtä prosessoriydintä tai prosessoria parannellen. Tehokkaampaa on käyttää useita yksinkertaisia ytimiä rinnakkain. Käytännössä kaikissa moderneissa tietokonealustoissa on useampi kuin yksi prosessoriydin, joiden utilisointi on mahdollista hyödyntämällä käyttöjärjestelmän tai ohjelmistoalustan tarjoamia säikeitä *(thread)*. [84, s. 1]

Ongelmaksi muodostuu, kuinka jokin laskennallinen tehtävä tulisi pilkkoa, jotta rinnakkaislaskennasta olisi mahdollisimman suuri hyöty. Tähän vaikuttavat suurimmaksi osaksi tehtävien laskennalliset riippuvuudet, jotka käyvät ilmi myös edellisessä luvussa muodostetun prosessointiverkon vaihejaosta.

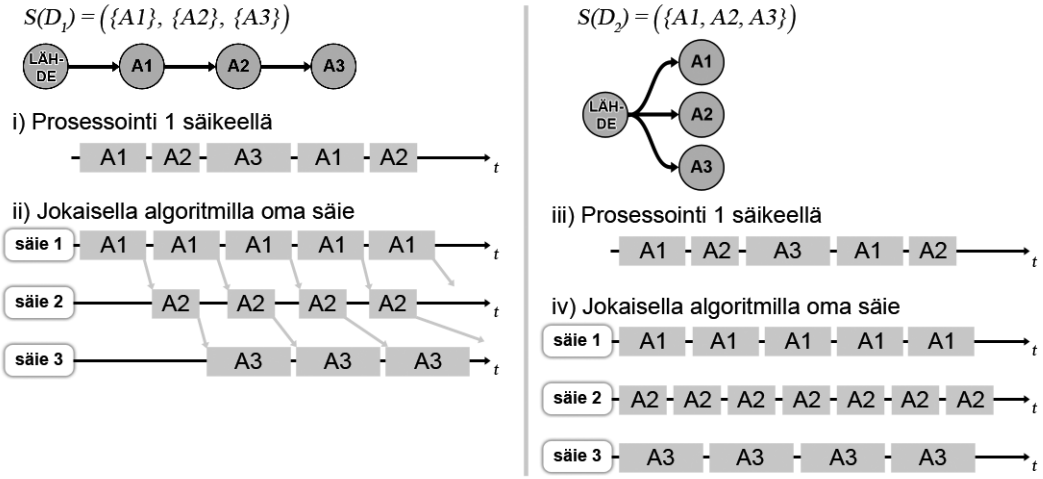
Tarkastellaan aluksi täysin putkimaisen algoritmiverkon vaihejärjestystä $S\left(D_1\right) = \left(\{A1\}, \{A2\}, \{A3\}\right)$ (kuva 5.7, vasen laita). Tässä algoritmi A3 riippuu algoritmista A2, joka riippuu edelleen algoritmista A1. Algoritmien suoritusta on kuvattu vaakatason aika-akselille *t* sijoitetuilla suorakulmioilla, joiden leveys ilmaisee ajallista käsittelykustannusta C_W. Tässä tekstissä tullaan viittaamaan yksittäisten algoritmien käsittelykustannukseen kyseisen algoritmin nimellä; esimerkiksi termillä *T(A1)* ilmaistaan algoritmin A1 ajallista käsittelykustannusta ja termillä *T(A1) + T(A2)* vastaavasti algoritmien A1 ja A2 yhteenlaskettua kustannusta.

Algoritmien suoritukseen useilla säikeillä vaikuttaa algoritmien luonne (sisäinen säikeistys, muistiosoitusten määrä, laskuoperaatioiden laatu jne.), prosessointiverkon rakenne[30], sekä säikeistykseen vaikuttavia ympäristötekijöitä kuten käyttöjärjestelmän tapa jakaa suoritinaikaa ja käytettävissä olevien prosessoriytimien määrä. Seuraavaksi tarkasteltavissa esimerkeissä on oletettu, että algoritmien käsittelykustannus C_W on joka kutsumiskerralla sama[31] ja että sovelluksen säikeet saavat keskenään saman vakiomäärän prosessointiaikaa. Lisäksi algoritmien välisen tiedonsiirron kustannukset on jätetty huomiotta. Kuvan vasemmassa laidassa on verrattu keskenään kahta eri säikeistysmallia (i) ja (ii). Ensimmäisessä (i) kaikki algoritmit suoritetaan samassa säikeessä, jolloin niiden suoritus on peräkkäistä. Yksittäisen näyteikkunan käsittely kestää $T(A1) + T(A2) + T(A3)$.

Verrattaessa tätä tapaukseen (ii), jossa kukin algoritmi suoritetaan omassa säikees-

30. Algoritmien keskinäiset riippuvuussuhteet ja suhteellinen laskentakuorma toisiinsa verrattuna.
31. Tämä pätee hyvin digitaalisille LTI-suotimille, muttei esimerkiksi sellaisille algoritmeille jotka käsittelevät tietoa vain satunnaisesti ilmenevien tapahtumien yhteydessä.

sään, huomataan jälkimmäisen olevan ajallisesti tehokkaampi – sillä olettamuksella, että laitteisto kykenee suorittamaan algoritmeja aidosti rinnakkain. Ensimmäiset tulokset algoritmilta A3 saadaan tapauksissa (i) ja (ii) samalla ajanhetkellä, mutta tapauksessa (ii) jokaisen algoritmin seuraava prosessointikierros alkaa aiemmin. Algoritmin A2 laskentakierrosten välissä sen sijaan on hukka-aikaa; tämä johtuu siitä, että A1:n laskenta kestää A2:ta pidempään, jolloin A2 *nääntyy* – se ei saa tietoa niin nopeasti kuin ehtisi laskea. Huomattakoon myös, että tapauksessa (ii) A3:n tulokset jäävät jälkeen A1:n ja A2:n tuloksista. Kysessä on nääntymisongelman vastakohta; tietoa tulee A3:lle nopeammin kuin se ehtii sitä käsitellä. Voidaan sanoa, että A3 *ruuhkautuu.* Tapauksessa (ii) käsittelynopeuden määrää algoritmi, jonka käsittelyaika on suurin, eli A3.



Kuva 5.7. *Rinnakkaisprosessointi kahdessa esimerkkitapauksessa.*

Kuvan 5.7 oikeassa laidassa on esitetty putkimaisen verkon vastakohta, leveä verkko $S(D_2) = (\{A1, A2, A3\})$, jossa algoritmien välisiä riippuvuuksia ei ole. Tällä verkolla yhden säikeen prosessointi (iii) toimii samoin kuin tapauksessa (i). Rinnakkaisprosessoinnista tapauksessa (iv) on sen sijaan huomattavia suorituskyvyllisiä hyötyjä. Koska algoritmien välisiä riippuvuuksia ei ole, mikään haara ei näänny tai ruuhkaudu. Jokainen algoritmi toimii omaa vauhtiaan, jonka johdosta niiden ulostulot eivät ole keskenään synkronissa. Esimerkiksi A2 ehtii käsitellä 6 ikkunallista näytteitä siinä ajassa, missä A3 ehtii vain 4. Tästä ei muodostu ongelmaa, ellei kyseisten algoritmien tuloksia tarvitse yhdistää reaaliaikaisesti jossain muualla toimitusketjussa (esimerkiksi lähetysvaiheessa tai palvelimella).

Lisäksi on olemassa monisäikeinen ratkaisuvaihtoehto, jossa tietoa käsitellään lineaarisesti kuten kuvan 5.7 vaihtoehdoissa (i) ja (iii), mutta siten, että jokainen prosessointikierros suoritetaan omassa säikeessään. Täten esimerkiksi mittaustietoikkunoita 1 ja 2 käsitellään rinnakkain. Ongelmaksi tässä kuitenkin muodostuu se, että algoritmien sisäinen tila voi muuttua käsittelyn seurauksena. Jos ikkuna 2 tulee käsitellyksi ennen ikkunaa 1, on mahdollista että algoritmi tuottaa virheellisiä tuloksia. Tämä on mahdollista huomioida algoritmeja kehitettäessä, joskin kehitys vaikeutuu ja ohjelmointivirheiden riski kasvaa.

Edellä esitetyt viisi esimerkkiä antavat jonkinlaisen kuvan rinnakkaiskäsittelyn hyödyistä ja ongelmista. Näiden ilmeneminen riippuu olennaisesti verkon rakenteesta ja algoritmien luonteesta, jotka kummatkin ovat premissien (AP1) ... (AP8) rajoissa täysin käyttäjän päätettävissä. Tarkasteltujen esimerkkien perusteella monisäikeiset ratkaisut vaikuttaisivat tehokkaammilta. Tapauksessa (ii) havaittu ruuhkautuminen muodostaa kuitenkin ongelman; A3:n edustalle voi kerääntyä pitkällä aikavälillä suuria määriä tietoa (EEG-rekisteröinnit voivat kestää kymmeniä tunteja).

Pohditaan tätä vielä toisen esimerkin (kuva 5.8) avulla. Kuvan verkossa algoritmi A3 muodostaa pullonkaulan; suorituksen aikakaaviosta nähdään, ettei A3 ehdi prosessoida tietoa yhtä nopeasti kuin A2. Tämän johdosta tietoa kerääntyy A3:n sisääntuloihin odottamaan käsittelyä. Vaikutus ei rajoitu A3:een, vaan ilmenee myös A4:n edustalla, jonne kerääntyy tuloksia A2:lta. A4 ei voi näitä käsitellä, koska se ei ole saanut niihin liittyvää rinnakkaistietoa A3:lta. Tietoa kerääntyy siis sekä A3:n että A4:n edustalle.

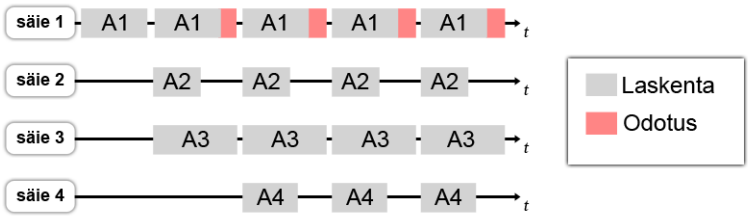
Kuva 5.8. *Esimerkki ruuhkautumisesta.*

Jos tiedon kerääntyminen algoritmien sisääntuloihin sallitaan, täytyy sisääntuloissa olla puskurit. Tämä puolestaan nostaa sovelluksen muistivaatimuksia, joita voidaan lieventää varastoimalla odottavaa tietoa levylle. Tällöin ongelmaksi muodostuu levylle kirjoittamisen ja siltä lukemisen hitaus eikä suorituskyky parane A3:n maksimista. Lisäksi vikatilanteista toipuminen vaikeutuu; algoritmien välisissä puskureissa voi olla varastoituna suuria määriä tietoa, jonka kokoaminen vikatilanteissa on hankalaa. Siksi voikin olla kannattavaa estää yksittäisiä algoritmeja toimimasta liian nopeasti. Toimintamalli voisi olla kuvan 5.8 esimerkissä seuraava (tämän ajoituskaavio kuvassa 5.9):

> A1 saa ikkunallisen tietoa ja käsittelee sen. Se alkaa syöttämään tuloksia A2:n ja A3:n sisääntuloihin, mutta havaitsee, ettei A3 ole ehtinyt käsitellä edellisiä A1:n tuloksia. A1 odottaa, että A3 saa käsiteltyä nämä tulokset, ja tallentaa tuloksensa A2:lle ja A3:lle vasta sen jälkeen. Odottaessaan A1 ei käsittele uusia ikkunoita. Täten myös A2 odottaa lisää käsiteltävää tietoa, eivätkä A3 ja A4 ruuhkaudu.

Tätä mekanismia voidaan verrata tulipalon sammutuksessa ihmisten muodostamaan linjastoon, jossa vesiämpäri annetaan aina seuraavalle jonossa. Yksittäinen ihminen ei voi vastaanottaa uutta vesiämpäriä, ennen kuin on antanut edellisen ämpärin seuraavalle. Se siis rytmittää algoritmiverkon toimintaa hitaimman algoritmin mukaiseksi. Kuvan 5.9 ajoituskaaviota tarkastellessa huomataan, että toiminta rytmittyy ajallisesti synkronoituihin suorituskierroksiin, joissa jokainen algoritmi käsittelee yhden ikkunallisen tietoa. Sivuvaikutuksena nopeimmat algoritmit tosin pääsevät nääntymään. Hyvänä puole-

na verkko on ajoitusten osalta adaptiivinen, sopeutuen algoritmien mahdollisesti muuttuviin käsittelyaikoihin $T(A)$. Lisäksi toiminta kuvassa 5.6 esitetyn kaltaisilla, alkuvaiheessa haarautuvilla verkoilla, ei ole optimaalista; jos toinen käsittelyhaaroista (A2 → A4 tai A3) ruuhkautuu, jää A1 odottamaan tätä haaraa, näännyttäen toisen haaran.



Kuva 5.9. *Säikeiden synkronointi.*

Tilanne ei välttämättä ole näin paha; Windows-käyttöjärjestelmä jakaa säikeille suoritusaikaa kvanteissa, joiden pituus voi muuttua ajon aikana (luvussa 5.2 mainittiin, ettei Windows takaa säikeille jaettua minimisuoritinaikaa). Lisäksi säikeen suoritus keskeytyy, jos sen prioriteettia suuremman prioriteetin omaava säie voidaan ottaa suoritukseen[32] [79, luku 5.7]. Samalla Windows-alustalla on toiminnassa muitakin ohjelmia, jotka tarvitsevat suoritinaikaa; esimerkiksi NeurOnen sovellusohjelma voi vaatia paljonkin resursseja suurikaistaisissa mittauksissa. Jos prosessointiverkkoa ajetaan laitealustalla, jossa on vähemmän prosessoreja kuin verkossa algoritmeja, voivat odottavat algoritmit luovuttaa prosessoriaikaa niille algoritmeille joilla on vielä tietoa käsiteltävänä. Esimerkin (kuva 5.9) mukaisessa verkossa A1:n, A2:n ja A4:n luovuttama suoritinaika voi yhden prosessorin järjestelmässä nopeuttaa algoritmin A3 toimintaa.

On siis löydetty jo kaksi eri käsittelyvaihtoehtoa; (a) käsittely yhdessä säikeessä sekä (b) kaikkien algoritmien rinnakkaistaminen ja synkronointi. Myös näiden välimaastosta löytyy ratkaisumalleja; esimerkiksi käsittelyvaiheen sisällä olevat algoritmit voidaan suorittaa rinnakkain, mutta itse käsittelyvaiheet peräkkäin yhdessä säikeessä. Tämä on loogista, sillä kaikki seuraavan käsittelyvaiheen S_{k+1} algoritmit riippuvat vaiheen S_k tuloksista. Vaihtoehto (b) on kuitenkin mahdollisesti tätä optimaalisempi, sillä siinä käsittely porrastuu ja rinnakkaistuu myös eri vaiheiden välillä; vaihe S_1 voi prosessoida uutta ikkunaa sillä välin kun vaihe S_2 käsittelee vielä edellistä [85, s. 96–106]. Lisäksi algoritmin tekijä voi jakaa itse algoritmin rinnakkaisiin säikeisiin – tai mahdollisuuksista riippuen suunnitella algoritmin siten, etteivät sen kanavat ole keskenään riippuvaisia. Hyvä esimerkki tällaisesta algoritmista on FIR-suodin:

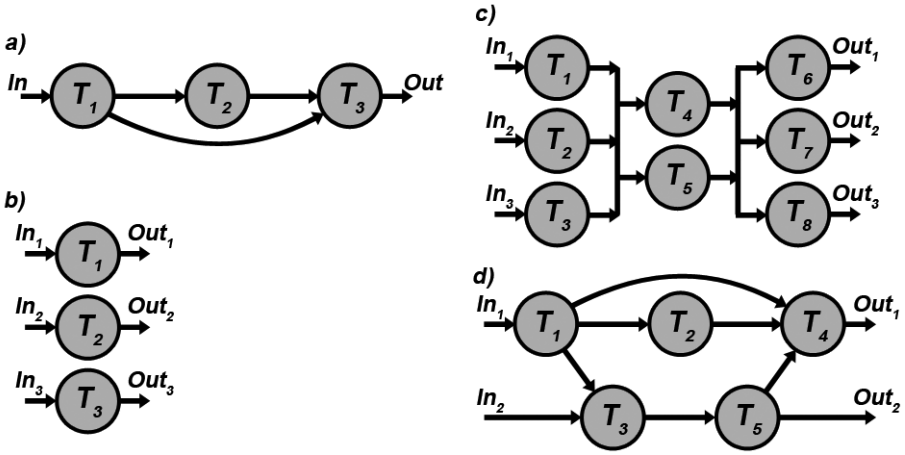
> Oletetaan, että käsittelyprotokollan suunnittelija haluaa suodattaa tällaisella FIR-suotimella neljä kanavaa; hän lisää käsittelyprotokollaan kaksi FIR-suotimen toteuttavaa algoritmia. Ensimmäiseen hän liittää kanavat 1 ja 2 sekä toiseen kanavat 3 ja 4. Jos algoritmit suoritetaan säikeissä rinnakkain, rinnakkaistuu myös kyseisten kanavanippujen laskenta.

Rinnakkaisia algoritmeja käsittelevässä teoksessaan [84] Gebali esittää, että algorit-

32. Tätä kutsutaan *keskeyttäväksi vuorontamiseksi (pre-emptive scheduling)*.

mi voidaan jakaa tehtäviksi (task), joista muodostuu algoritmin oma, sisäinen prosessointiverkko. Algoritmi voidaan luokitella kyseisen verkon rakenteen perusteella peräkkäiseksi *(Serial Algorithm)*, rinnakkaiseksi *(Parallel Algorithm)*, peräkkäis-rinnakkaiseksi (*Serial-Parallel Algorithm*, SPA), ei-peräkkäis-rinnakkaiseksi (*Nonserial-Parallel Algorithm*, NSPA) tai iteratiiviseksi (*Regular Iterative Algorithm*, RIA). Näistä kaikki paitsi RIA on esitetty kuvassa 5.10. SPA (c) ja NSPA (d) poikkeavat toisistaan siten, ettei SPA:ssa voi ilmetä kuvan 5.10 kohdassa (d) esitettyä $T_1 \rightarrow T_4$ kaltaista riippuvuutta.

Gebalin periaatteita voidaan soveltaa myös tässä työssä kuvattuun algoritmeistä muodostuvaan verkkoon, ajatellen sitä yksittäisenä Gebalin algoritmina, missä algoritmit ovat Gebalin tehtäviä. Tässä työssä kuvattu prosessointiverkko kuuluu NSPA-algoritmien luokkaan (kuva 5.10, d). Myös Gebali päätyy jakamaan NSPA-algoritmit käsittelyvaiheisiin, muttei ota kantaa näiden rinnakkaistamiseen. [84, luku 8]



Kuva 5.10. *Gebalin algoritmijako: a) peräkkäinen, b) rinnakkainen, c) peräkkäis-rinnakkainen ja d) ei-peräkkäis-rinnakkainen*

Suorituskyvyn tarve riippuu paljolti käsiteltävien signaalien kaistanleveydestä ja algoritmien laskennallisesta monimutkaisuudesta suhteessa alustan tarjoamiin laskenta- ja muistiresursseihin. Usein monitoroidaan pientä määrää kanavia suhteellisen matalilla näytteenottotaajuuksilla, esimerkiksi kymmenen kanavaa 500 Hz näytteenottotaajudella tuottaa mittaustietoa vain

$$B = 10 \cdot 32 \frac{b}{näyte} \cdot 500 \frac{näytettä}{s} \approx 20 \frac{kB}{s}. \qquad (5.6)$$

Käsittelyn rinnakkaistamisesta ei välttämättä ole hyötyä pienillä kaistanleveyksillä ja yksinkertaisilla käsittelyprotokollilla. Rinnakkaisuuden toteutus tämän työn yhteydessä on alttiina tietyille rajoituksille (luku 6.5.2) ja sen kannattavuuden arviointi vaatii konkreettisia tutkimuksia. Tämän vuoksi aiheeseen ei paneuduta tässä työssä tarkemmin, vaan se jätetään jatkokehityksen osa-alueeksi.

5.4 Mittaustiedon välitys verkon yli

Johdannossa esitettiin vaatimus mittaustiedon välityksestä tietoverkkoon. Luvussa 5.1 tarkennettiin kyseessä olevan pakettikytkentäisen, IP-protokollaan pohjautuvan tietoverkon. Tämä vaatimus kumpuaa yksinkertaisesti siitä, että kyseinen verkkotyyppi on niin yleinen – käytännössä kaikissa sovelluksen mahdollisissa käyttöympäristöissä tiedonsiirto tapahtuu IP-verkon kautta. IP mahdollistaa myös verkkokapasiteettia säästävän tiedonvälityksen usealle palvelimelle yhtä aikaa *(multicast)*. Tähän ominaisuuteen ei kuitenkaan puututa työssä tarkemmin. Käytössä oletetaan olevan IPv4-protokollan – sen uudempaan versioon IPv6:een ei oteta kantaa.

Mittaustiedon välitykseen on useita lähestymistapoja; sovellus voi huolehtia tiedonvälityksestä täysin itsenäisesti tai vaihtoehtoisesti luottaa ulkoisiin komponentteihin. Esimerkkinä jälkimmäisestä mainittakoon Windows-käyttöjärjestelmän *Background Intelligent Transfer Service* -palvelu (BITS) [86]. Tässä luvussa tarkastellaan tiedonvälitystä ensimmäisen lähestymistavan mukaan, sillä täten varmistetaan yhteensopivuus kolmansien osapuolien palvelinratkaisujen kanssa.

Pakettikytkentäisissä verkoissa lähetettävä tieto pilkotaan paketeiksi, jotka lähetetään verkkoon. Verkko ja sen toimilaitteet huolehtivat pakettien toimituksesta kohteeseen. Paketit voivat kulkea kohteeseen eri reittejä ja saapua perille epäjärjestyksessä. Osa lähetetyistä paketeista voi hävitä matkalla; esimerkiksi verkon ruuhkautuessa jokin toimilaite saattaa jättää paketin käsittelemättä *(packet loss)*. Lisäksi paketin sisältämä tieto voi muuttua matkalla esimerkiksi ohjelmistovirheen tai viallisen laitteiston seurauksena. IP ei ota kantaa näihin ongelmiin vaan ne on ratkaistava ylemmän tason protokollalla (esim. *Transmission Control Protocol*, TCP), jolla on huomattava vaikutus tiedonvälityksen luotettavuuteen sekä sovelluksen kykyyn mukautua tietoverkon muuttuviin olosuhteisiin. [82, luku 2.2]

Lisäksi on syytä pohtia tiedonsiirrossa käytettävää formaattia, joka määräytyy luvussa 5.4.1 esitetyn sovelluskerroksen protokollan perusteella. Formaatti määrää, kuinka tarkasti tieto välitetään ja täten sillä on huomattavia vaikutuksia tiedonsiirron kaistavaatimuksiin.

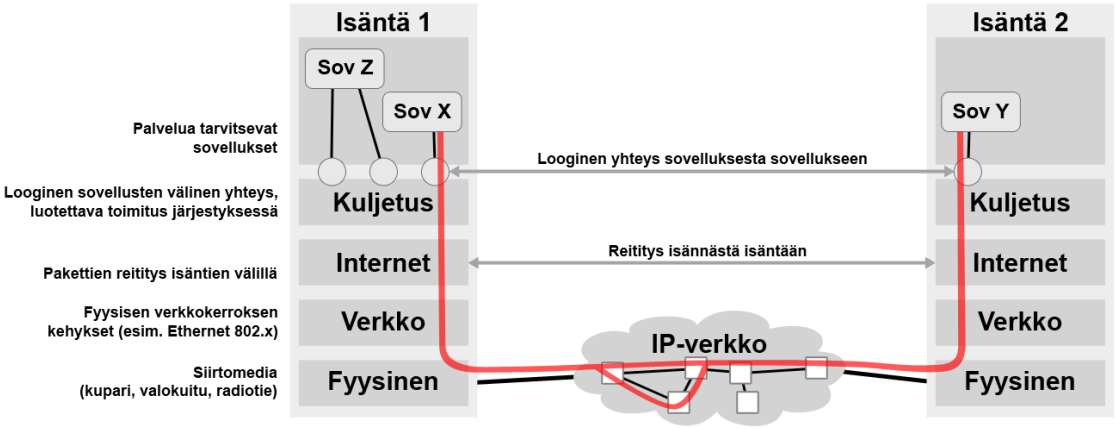
Viimeisimpänä – muttei vähäisimpänä – ongelmana on tietoturva. Mitatut ja käsitellyt signaalit ovat potilaaseen liittyvää henkilökohtaista lääketieteellistä tietoa. Ulkopuoliset tahot eivät saisi päästä tätä muokkaamaan tai lukemaan.

5.4.1 Käytettävä kuljetusprotokolla

Tiedon välitys kahden isännän välillä tietoverkossa vaatii aina jonkin sovitun yhteyskäytännön (protokollan), jota kumpikin isäntä ja näiden välillä olevat verkkolaitteet ymmärtävät. Oletettavasti keskenään tietoa vaihtavilla isännillä toimii jokin sovellus, jossa konkreettinen tiedon välityksen tarve ilmenee. Tällöin myös sovellusten täytyy kommunikoida yhteisesti sovittujen käytäntöjen mukaisesti. Nämä kommunikaation eri tasot (verkkolaite-verkkolaite, isäntä-isäntä sekä sovellus-sovellus) voidaan ryhmitellä kerroksiin. Tätä mallia voidaan yleisesti kuvata esimerkiksi OSI-kerrosmallilla *(Open Sys-*

tems Interconnection) tai hieman yksinkertaisemmalla TCP/IP-kerrosmallilla (kuva 5.11). Molemmassa mallissa on sama idea; jokainen kerros tarjoaa yläpuolellaan olevalle kerrokselle palveluja ja vastaavasti nojaa alapuolellaan olevan kerroksen palveluihin. Ylimpänä on *sovelluskerros*, jossa itse kommunikaatiotarve syntyy ja alimpana *fyysinen kerros*, missä tieto välittyy esimerkiksi valopulsseina tai jännitetason vaihteluina. Kerrosten välisissä palvelupyynnöissä kuljetettava tieto välitetään käytännössä lohkoissa (*Protocol Data Unit*, PDU). [82, luvut 2.2–2.3]

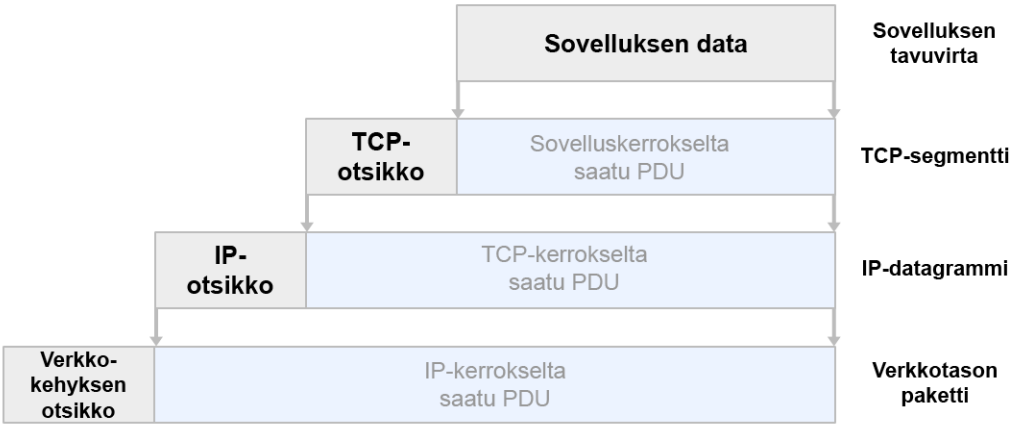
Tässä luvussa tarkastellaan tiedonsiirtoa TCP/IP-kerrosmallin pohjalta, kuitenkaan TCP-protokollaan rajoittumatta. TCP/IP protokolla-arkkitehtuuri on jaettu viiteen kerrokseen (kuva 5.11); fyysinen kerros kattaa tiedon siirron verkkolaitteen (esim. tietokone) ja fyysisen siirtomedian (esim. kuparikaapelointi) välillä. *Verkkokerros* määrittää tiedon siirron verkkolaitteen (esim. tietokone) ja verkon välillä. Tämän tason ohjelmisto riippuu verkosta; esimerkiksi lähiverkoissa käytetään yleisesti IEEE 802.3 tai 802.11 -protokollaa. Tiedon reititys IP-verkosta toiseen tapahtuu *internet-kerroksella* (IP-protokolla). Tiedon luotettavasti järjestyksessä perille toimituksesta vastaa *kuljetuskerros*, jolla TCP-protokolla toimii. Ylimpänä arkkitehtuurissa on sovelluskerros, jossa tietoa lähettävien ja vastaanottavien sovellusten logiikka sijaitsee. Tällä tasolla määritetään mm. välitettävien viestien sisältö (teksti, kuva, ääni). Käytännön toteutumaa tästä arkkitehtuurista kutsutaan *protokollapinoksi*. [82, luku 2.2]



Kuva 5.11. *TCP/IP -kerrosmalli [82, s. 36].*

Käydään läpi yksinkertaistettu esimerkki tiedon välittymisestä eri kerroksien läpi. Isännällä 1 oleva sovellus X haluaa lähettää tietoa isännällä 2 toimivalle sovellukselle Y (kuva 5.11). Oletetaan, että käytössä on TCP/IP-protokollapino ja että yhteys on jo muodostettu. Sovellus X pyytää kuljetuskerroksen (tässä esimerkissä TCP) palvelua lähettämään antamansa datapaketin (PDU) (kuva 5.12). TCP-kerros pilkkoo välitettävän datan tarvittaessa pienempiin paketteihin (TCP-segmentteihin) ja lisää näihin omat otsikkotietonsa, jotka identifioivat mm. pakettien järjestyksen ja kohteena olevalla isäntäkoneella toimivan sovelluksen (portti). Tämän jälkeen TCP-kerros antaa paketit edelleen IP-kerroksen välitettäväksi, joka lisää niihin omat otsikkotietonsa (mm. kohdeisännän IP-osoitteen) ja voi pilkkoa paketteja edelleen. Vastaava jatkuu, kunnes lähetettävä tieto on

kulkenut koko protokollapinon läpi siirtomedialle (kuva 5.11). Matkallaan tietoverkon läpi paketti saattaa kulkea useiden siirtomedioiden (kuparijohto, valokuitu) ja TCP/IP protokollapinojen (reitittimet) läpi. Peräkkäiset paketit saattavat kulkea verkon läpi eri reittejä pitkin ja päätyä kohdeisännälle eri järjestyksessä kuin ne lähetettiin. Kohdeisännän vastaanotettua tiedon fyysiseltä siirtomedialta kulkee tieto protokollapinon läpi kuljetustasolle saakka, missä TCP järjestää vastaanottamansa segmentit oikeaan järjestykseen ja tarjoaa nämä tietoa vastaanottavalle sovellukselle. [82, luku 2.2]



Kuva 5.12. *PDU-paketointi TCP/IP kerrosmallissa [82, s. 37].*

Optimaalinen kuljetusprotokolla riippuu tilanteesta; toimitettaessa tietoa vain katselua varten halutaan säilyttää reaaliaikaisuus ja tiedon häviäminen on ehkä sallittavissa. Toisaalta jos tietoa käsitellään myös palvelimella, on tärkeää että kaikki tieto välittyy luotettavasti ja järjestyksessä.

Saatavilla on useita kuljetusprotokollia, yleisimpinä TCP *(Transmission Control Protocol)* ja UDP *(User Datagram Protocol)*. TCP syntyi 1970-luvulla ja UDP vuonna 1980. Kumpaankin on toki tehty jälkeenpäin laajennuksia ja parannuksia sekä molemmat ovat erittäin laajalti käytössä ja hyvin tuettuja esimerkiksi verkkolaitteiden osalta. Lisäksi saatavilla on uudempia protokollia, kuten RTP *(Real-time Transport Protocol)* ja SCTP *(Stream Control Transmission Protocol)*.

Mittaustiedon välitykseen tarvitaan lisäksi sovellustason protokolla: tietoa vastaanottavan sovelluksen täytyy ymmärtää vastaanottamansa viesti. Osa tästä liittyy mittaustiedon formaattiin (esim. ovatko näytteet 16 vai 32 bitin tarkkuudella, luku 5.4.2), mutta sovellus voi myös haluta kuljettaa jokaisen mittaustietoikkunan yhteydessä metatietoa, kuten ikkunan alkuaika. Lisäksi voi olla haluttavaa toteuttaa erillinen sovellustason kuittausmekanismi vastaanotetulle tiedolle, jotta voidaan toipua mahdollisista tiedonsiirto- ja yhteysvirheistä.

Seuraavissa kappaleissa käsitellään lyhyesti kutakin protokollavaihtoehtoa. Näiden ominaisuuksista tehty yhteenveto on nähtävillä taulukossa 5.1 (sivu 52). Protokollien ominaisuuksista on käyty läpi vain sovellusalueen kannalta välttämättömimmät ja niiden toiminta esitetään yksinkertaistettuna.

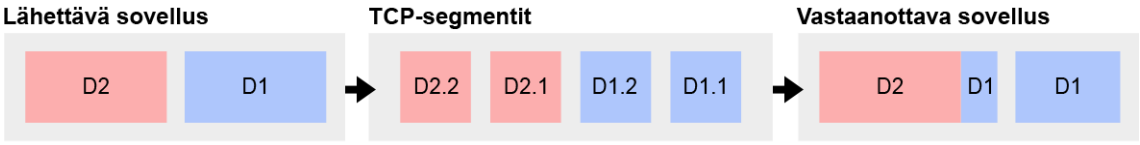
Transmission Control Protocol (TCP)

TCP on hyvin laajalti käytetty yhteydellinen, tietovirtaorientoitunut protokolla, joka tarjoaa tiedon luotettavan järjestyksessä toimituksen. Yhteydellisyys tarkoittaa että kommunikoidakseen sovellusten täytyy ensin muodostaa looginen TCP-yhteys. Tämä muodostaa TCP:aa käyttävän sovelluksen näkökulmasta jatkuvasti auki olevan kanavan, vaikka todellisuudessa tiedonsiirto tapahtuukin IP-verkkokerroksen paketeissa. Yhteyden muodostettuaan sovellukset voivat lähettää toisilleen tietoa, sekä lopuksi sulkea yhteyden. Yhteys muodostetaan aina kahden päätepisteen (esim. sovellus) välillä; jos halutaan lähettää tietoa usealle päätepisteelle, täytyy näistä jokaiseen muodostaa oma TCP-yhteys.

Tietovirtaorientoituneisuudella puolestaan tarkoitetaan, että TCP tulkitsee kaiken yhteydelle annetun lähetettävän tiedon virraksi peräkkäisiä tavuja. TCP tuottaa näistä segmenttejä, joiden välittämiseen se käyttää IP-kerroksen tarjoamia palveluja. Järjestyksessä toimitus puolestaan tarkoittaa sitä, että siirrettyjen tavujen järjestys säilyy. Luotettavuudella tarkoitetaan, että toimitettavaksi annettu tieto saapuu perille muuttumattomana – tai yhteys asetetaan virhetilaan jos toimitus ei onnistu (esim. yhteyden katketessa). Käytännössä tämä varmistetaan otsikkotiedoista ja kuljetettavasta datasta laskettavilla tarkistussummilla sekä lähettäjälle palautettavilla kuittauksilla. Jos lähettäjä ei saa segmentistä kuittausta tietyn ajan sisällä lähettämisestä, se olettaa segmentin hävinneen matkalla ja lähettää sen uudelleen. Jos tarkistussumma ei täsmää, vastaanottaja tietää segmentin vaurioituneen matkalla ja jättää sen käsittelemättä [87; 88], jolloin lähettäjä ei saa siitä kuittausta ja lähettää sen uudelleen.

Jos toinen osapuoli ei saa vastapuolelta kuittauksia tietyn viiveen sisällä, se tulkitsee yhteyden katkenneeksi. Tämä katkos ilmoitetaan ylemmälle sovelluskerrokselle. Tällöin sovelluskerroksen on lähetystä jatkaakseen muodostettava uusi TCP-yhteys. Yhteyden katketessa sovelluskerros ei saa tietoa siitä ehtikö TCP toimittaa kaiken sovelluksen sille antaman tiedon ennen katkosta, eli mille kaikille lähetetyille TCP-segmenteille saatiin kuittaus vastaanottajalta. Kun lähettävä sovellus saa yhteyden uudelleen muodostetuksi, se ei tiedä mistä lähetystä pitäisi jatkaa. Tämä puoltaa kuittausmekanismin toteutusta myös sovellustason protokollaan.

Koska TCP on tietovirtaorientoitunut protokolla, se kohtelee välitettävää informaatiota jatkuvana virtana, ei viesteinä. Jos lähettävä sovellus antaa TCP-kerrokselle toimitettavaksi paketit D1 ja D2 (kuva 5.13), voi TCP pilkkoa nämä toimituksen ajaksi esimerkiksi paketeiksi D1.1 ... D2.2. Kun vastaanottavan isännän TCP-kerros saa kyseiset paketit, järjestää ne oikeaan järjestykseen ja toimittaa ne sovellukselle, ei lähettäneen sovelluksen pakettiraja välttämättä säily. Tällöin vastaanottava sovellus voi saada esimerkiksi ensin vain osan paketista D1 ja vasta tämän jälkeen loput yhdistettynä paketin D2 sisältöön. Paketit D1 ja D2 täytyy siis erottaa toisistaan sovellustason protokollalla; lähettävä sovellus voi esimerkiksi merkitä paketin alkuun kuinka monta tavua paketti kokonaisuudessaan vie. Kun vastaanottava sovellus saa paketin, se voi lukea pituuden tästä kentästä ja odottaa kunnes saa luettua verkkoyhteydestä koko paketin. [82, luku 20.1]



Kuva 5.13. *TCP:n toimittama tietovirta.*

Pienikokoisilla sovelluksen lähettämillä paketeilla saattaa sama ilmetä jo lähettävällä isännällä; TCP voi viivästää pakettien lähetystä, puskuroiden lähetettävät tavut yhteen suurempaan TCP-segmenttiin. Kyseessä on *Naglen algoritmi* [89], jonka tarkoituksena on säästää verkon kapasiteettia. Kuvitellaan esimerkiksi tilannetta, jossa sovellus lähettää yhden oktetin dataa kerrallaan: jos jokainen lähetetään erikseen, syntyy tuloksena jokaista lähetettävää tavua kohti vähintään 41 tavun pituinen IP-datagrammi (kuva 5.12), josta 20 tavua on IP:n otsikkotietoja ja toiset 20 TCP:n [82, s. 578 ja 678]. Hyötykuorman osuus tällaisessa paketissa on hyvin alhainen. Huonona puolena algoritmin käyttö voi johtaa lähetysviiveisiin ja pahimmassa tapauksessa hetkellisin lukkotilanteisiin lähettäjän ja vastaanottajan välillä [90]. Moderneissa TCP/IP-protokollapinoissa tästä ei kuitenkaan pitäisi seurata ongelmaa; Naglen algoritmi voidaan poistaa käytöstä yhteyskohtaisesti. Lisäksi (mittauksesta riippuen) lähetettävää mittaustietoa voi kertyä niin paljon, että TCP:n lähetyspuskuri täyttyy ja lähetetään niin usein ettei lähetys aiheuta suurta viivettä mittaussignaalien toimitusketjussa.

Tietovirran myötä TCP:aan liittyy myös toinen ongelma: jos jokin TCP-segmentti tai sen osa (IP-fragmentti) katoaa matkallaan kohteeseen *(packet loss)*, pysähtyy vastaanottajalla koko yhteyden tietovirran käsittely kunnes kadonneen segmentin uudelleenlähetetty kopio on otettu vastaan (*head-of-line blocking*, HOLB). Koska TCP takaa järjestyksessä toimituksen, se ei voi antaa sovellukselle esimerkiksi segmentin D2.1 sisältöä ennen kuin se on antanut segmentin D1.2. Tämän vuoksi TCP ei sovellu kovin hyvin reaaliaikasovelluksiin, joissa kaikkien pakettien ei tarvitse saapua perille tai pakettien sisältämä informaatio menettää arvonsa myöhästyessään. Ominaisuudesta voi olla haittaa esimerkiksi mittaustiedon reaaliaikaisessa katselussa. [91]

User Datagram Protocol (UDP)

Toisin kuin TCP, ei UDP takaa tiedon luotettavaa eikä järjestyksessä perille toimitusta. Se on viestiorientoitunut: lähetetty viesti (paketillinen tavuja) joko päätyy vastaanottavalle sovellukselle kokonaan tai ei ollenkaan. Myös UDP on hyvin yleisesti käytetty; sen yleisimpiä käyttökohteita ovat reaaliaikaiset mediapalvelut (Internet-televisio (IPTV) ja Internet-puhelut (VoIP)) sekä online-tietokonepelit ja vertaisverkot. [92]

UDP on minimaalinen protokolla; se sisältää toiminnallisuutta vain murto-osan TCP:n vastaavasta. Periaatteessa UDP lisää IP-protokollan toiminnallisuuteen vain portit, joiden avulla isäntien välillä kulkevat paketit voidaan ohjata oikealle sovellukselle. Kommunikoivien sovellusten välille ei muodosteta loogista yhteyttä – viestit vain lähetetään matkaan ja ne joko päätyvät perille tai eivät. UDP:n otsikkotiedot sisältävät myös tarkistussumman, joka lasketaan otsikkotietojen ja datan yli, mutta kyseinen tarkistus-

summa ei ole pakollinen. [82, s. 694; 93]

UDP ei pilko lähetettäviä paketteja pienemmiksi, eikä siinä ole TCP:n tapaista kuittaustoiminnallisuutta. Matkallaan verkon läpi paketteja voi kadota tai monistua. Jos UDP-paketti on liian suuri lähetettäväksi verkkolinkin yli, IP-protokolla pilkkoo sen verkkotasolla IP-fragmenteiksi. Jos yksikin fragmentti jää saapumatta verkon kautta vastaanottajalle, ei vastaanottaja voi koota alkuperäistä IP-datagrammia ja jättää muut vastaavan datagrammin fragmentit huomiotta. Monistuminen (duplikaatio) voi tapahtua esimerkiksi jos lähettävä osapuoli jostain syystä luulee ettei paketin lähetys onnistunut ja lähettää sen siksi uudelleen. Lisäksi vialliset verkon toimilaitteet voivat monistaa paketteja. Saman paketin kopioita ei voi erottaa toisistaan UDP-tasolla, koska UDP:n otsikkotiedoissa ei ole tämän mahdollistavia tunnisteita. Kuittausten puutteen vuoksi UDP ei myöskään kykene vuonohjaukseen: jos tietoa lähetetään nopeammin kuin sitä ehditään vastaanottaa, paketteja jää käsittelemättä. UDP voi myös jättää lähettämättä paketteja, jos se ei ehdi käsitellä kaikkia sovelluskerrokselta tulevia lähetyspyyntöjä. [92]

Jos halutaan havaita puuttuvat ja mahdollisesti monistuneet paketit täytyy näille toteuttaa jonkinlainen sovellustason identifiointi- ja kuittausmekanismi. Vastaanottavan sovelluksen on myös järjestettävä paketit oikeaan järjestykseen. Lisäksi sovellustasolla täytyy toteuttaa jonkinlainen tarkistussumma, jonka perusteella voidaan todeta vastaanotetun tiedon eheys käytettyä UDP-protokollapinoa varmemmin.

Hyvänä puolenaan UDP on nopea: yhteyttä ei tarvitse erikseen avata, eikä matkalla katoava paketti estä vastaanottajaa saamasta sen jälkeen lähetettyjä paketteja (HOLB). Lisäksi UDP mahdollistaa multi- ja broadcast-tyyppisten pakettien lähetyksen. Multicast-paketissa voi olla useita vastaanottajia ja broadcast paketti välitetään kaikille. Tämän avulla voisi olla mahdollista jakaa reaaliaikaista mittausinformaatiota usealle vastaanottajalle verkkoa mahdollisimman vähän kuormittaen.

UDP soveltuu erityisen hyvin tilanteisiin, joissa yksittäinen viesti täytyy toimittaa paljon viivettä omaavan verkkoyhteyden yli mahdollisimman nopeasti, sekä tilanteisiin, joissa lähetetään paljon samanlaista tietoa usealle vastaanottajalle, eikä jokaisen paketin ole pakollista päätyä perille. [82, s. 694]

Stream Control Transmission Protocol (SCTP)

Eräs suhteellisen uusi protokolla, *Stream Control Transmission Protocol* (SCTP), vaikuttaa soveltuvan suhteellisen hyvin tässä työssä toteutettavaan sovellukseen. Se sai alkunsa, kun yritettiin sovittaa tiukat ajoitus- ja luotettavuusvaatimukset omaavaa puhelinverkon signalointiliikennettä kulkemaan TCP-protokollan päällä ja huomattiin, ettei TCP sovellu tarkoitukseen. [91; 94]

Lyhyesti ilmaistuna SCTP yhdistää TCP:n ja UDP:n hyvät puolet. Luonteeltaan se on UDP:n tavoin viestiorientoitunutta ja haluttaessa toimittaa vastaanotetut viestit sovelluskerrokselle TCP:n tapaan järjestyksessä. Lisäksi vastaanotetuista viesteistä lähtee TCP:n tavoin kuittaus lähettäneelle osapuolelle. Tässä kuittauksessa vastaanottaja voi myös erikseen ilmaista jos se havaitsee välistä tippuneita paketteja. SCTP:ssa käytetään samoja ruuhkautumisen tarkkailu- ja estoalgoritmeja kuin TCP:ssa. Ennen varsinaista

tiedonsiirtoa protokolla edellyttää että lähettäjän ja vastaanottajan välille muodostetaan looginen yhteys (SCTP:ssa tästä käytetään termiä assosiaatio). [91; 94]

SCTP tukee myös useita isännän verkkorajapintoja *(multi-home)*: samalle isännälle voi olla asennettuna useita verkkosovittimia, joista jokaisella on oma IP-osoitteensa. SCTP-assosiaatiolle valitaan näistä ensisijainen ja jos tämä vikaantuu (alemman protokollatason yhteys katkeaa), siirtyy SCTP käyttämään toisia sovittimia – sovellukselle täysin läpinäkyvästi. Eri verkkosovittimet voidaan isännät ja reitittimet konfiguroimalla asettaa käyttämään erillisiä fyysisiä verkkopolkuja, jolloin myös fyysinen vikasietoisuus paranee. [91; 94]

Valitettavasti SCTP-protokolla ei ole vielä laajalti tuettu. Isäntäkoneille on saatavissa useita ilmaisia ja maksullisia ajuri- ja käyttäjätason toteutuksia (Unix-, Linux-, FreeBSD- ja Windows-järjestelmät), mutta tällaisen asentaminen esimerkiksi sairaalassa sijaitsevalle palvelimelle on kyseenalaista. Lisäksi ongelmia voi ilmetä verkon reitittimissä, joiden täytyy ymmärtää kuljetuskerroksen protokollaa (TCP, UDP, SCTP) esimerkiksi verkon osoitemuunnosten (*Network Address Translation*, NAT) yhteydessä. Jos reititin ei tue SCTP-protokollaa, ei se välttämättä pysty käsittelemään SCTP:n multi-home-liikennettä. Lisäksi reitittimet saattavat joutua muuttamaan kuljetuskerroksen PDU:n informaatiota, jolloin kyseisen PDU:n tarkistussummat on laskettava uudestaan. TCP ja UDP -protokollissa tarkistussummien laskentaan käytetään kahden komplementtiin perustuvaa algoritmia; SCTP:ssa taasen raskaampaa CRC-32C-algoritmia. Tarkistussummat lasketaan koko PDU:n yli, eli mukana ovat sekä protokollan otsikkotiedot että ylemmältä protokollatasolta saatu data. Jos dataa on paljon, SCTP:n varmistussumman laskenta työllistää reititintä huomattavasti TCP- tai UDP-protokollan vastaavaa laskentaa enemmän. [95; 96]

Real-Time Transport Protocol (RTP)

Luodaan lopuksi katsaus hieman edellisistä poikkeavaan protokollaan, RTP:aan *(Real-Time Transport Protocol)*. Ensimmäinen ehdotus protokollasta julkaistiin vuonna 1996 [97] ja tämän korvaava uudempi versio vuonna 2003 [98]. RTP on suunnattu sovelluksiin joissa välitetään tietoa tasaiseen tahtiin yhdelle tai useammalle vastaanottajalle. Käyttökohteita ovat esimerkiksi videokonferenssit, live-videon jakelu ja toisto, puhelinliikenne, pelit, järjestelmien etävalvonta ja hallinta, sekä lääketieteellinen etädiagnoosi. [82, s. 821]

RTP ei noudata perinteistä OSI- tai TCP-kerrosmallin mukaista jakoa, vaan toimii sekä sovellus- että kuljetuskerroksella. Se ei sisällä kummankaan täydellistä toteutusta; tiedon toimituksessa RTP voi käyttää esimerkiksi UDP-protokollaa, joka puolestaan voi toimia IP-verkon päällä. Tämän vuoksi RTP ei pysty takaamaan tiedon ajoissa perille toimitusta, ja jättääkin sen alempien kerrosten vastuulle. RTP riippuu hyvin paljon sovelluskerroksen toteutuksesta (esim. onko kyseessä peli- tai etävalvontasovellus). Se ei ole valmis protokolla, joka voidaan ottaa suoraan käyttöön sovelluksessa, vaan pikemminkin toteutuskehys, joka koostuu edellä mainituille sovellusalueille yhteisestä toiminnallisuudesta (ajoitusinformaatio, sisältötyypin merkintä ja sisältövirtojen kuljetus). [82,

luku 24.4; 98]

RTP:n kumppanina toimii myös toinen protokolla RTCP *(RTP Control Protocol)*, jota käytetään ohjaukseen. Sen tärkeimpinä tehtävinä on välittää kaikille osapuolille tietoa RTP-tiedonsiirron laadusta (esim. ruuhkautumisen välttämiseksi) ja auttaa tunnistamaan osapuolet. [98]

RTP ja RTCP eivät sisällä yhteyden muodostamiseen tai purkamiseen tarvittavaa toiminnallisuutta, vaan tämä on toteutettava erillisellä protokollalla. Lisäksi RTP ei takaa luotettavaa järjestyksessä toimitusta; sen PDU-rakenteissa jokaisella paketilla on järjestysnumero, jonka perusteella vastaanottava sovellus voi järjestää paketit itse. Myös luotettava toimitus on sovelluksen vastuulla – tosin hyvästä syystä: esimerkiksi videoneuvottelusovelluksissa voi olla haluttavaa jättää myöhästyneet videosegmentit toimittamatta jos näitä jo uudempaa videota on lähetettävissä. [82, s. 822; 98]

Mutkikkaan arkkitehtuurin (useita protokollia ja kuljetus/sovellustason ylittävät kehysrakenteet) vuoksi RTP-protokollan käyttöönotto vaatii paljon resursseja eikä sitä tämän vuoksi käsitellä tarkemmin. Tulevaisuudessa voi kuitenkin olla haluttavaa tutkia RTP:n soveltuvuutta biosignaalien välitykseen tai mahdollisesti ottaa siitä vaikutteita erikseen toteutettavaan sovellustason protokollaan.

Taulukko 5.1. *TCP, UDP, SCTP ja RTP -protokollien ominaisuudet [82; 98; 99].*

Ominaisuus	**TCP**	**UDP**	**SCTP**	**RTP/RTCP**
Yhteydellinen	K	E	K	K
Kaksisuuntainen	K	K	K	K
Takaa luotettavan tiedonsiirron	K	E	K	K/E
Takaa osittain luotettavan tiedonsiirron	E	E	K/E	K/E
Toimitus järjestyksessä	K	E	K	E
Toimitus epäjärjestyksessä	E	K	K	K
Vuonohjaus	K	E	K	K
Ruuhkautumisen hallinta	K	E	K	K
Joustavampi kuittaus (ACK)	K/E	E	K	K/E
Säilyttää viestirajat	E	K	K	K
Sovellustason PDU-pirstoutuminen	K	E	K	K/E
Sovellustason PDU-kasaaminen	K	E	K	K/E
Monikanavointi	E	E	K	K
Monilähetys (multicast)	E	E	K	K

5.4.2 Sovelluskerroksen protokolla

Myös sovellustason protokollalla voi olla huomattava vaikutus tiedonsiirron kaistavaatimuksiin. Tällä tasolla päätetään mm. siitä, mitä tietoja välitetään ja millä tarkkuudella tieto ilmaistaan. Käytettävästä kuljetusprotokollasta riippuen sovelluskerros voi myös

mahdollisesti joutua huolehtimaan kadonneiden pakettien uudelleenlähetyksestä sekä vastaanotettujen pakettien järjestämisestä oikeaan järjestykseen. Sovelluskerroksen protokolla on jaettavissa formaattiin ja toimintoihin. Formaatilla tarkoitetaan mittaustiedon välitykseen liittyviä tietorakenteita ja toiminnoilla protokollan toteuttamaa toiminnallisuutta ja signalointia, kuten uudelleenlähetys. Tarkastellaan aluksi tiedonsiirrossa käytettävää formaattia.

Mitä tarkemmin mittaussignaalin arvo (näyte) halutaan kuvata, sitä enemmän sen kuvaamiseen vaaditaan bittejä (bittitarkkuus). 16 bitillä voidaan kuvata $2^{16} \approx 66\,000$ eri arvoa, 32 bitillä vastaavasti $2^{32} \approx 4{,}3 \cdot 10^9$. Tällä voi olla hyvinkin suuri merkitys paljon kaistaa vaativissa mittauksissa: bittitarkkuuden puolittaminen voi teoriassa puolittaa kaistavaatimukset. Haittapuolena on dynamiikan väheneminen; samalla joudutaan joko kaventamaan kuvattavaa arvoaluetta tai huonontamaan esitystarkkuutta. Kullekin signaalille sopiva arvoalue ja tarkkuus riippuvat signaalin luonteesta; esimerkiksi päänahalta mitatut häiriöttömät EEG-signaalit ovat amplitudiltaan muutamista satoihin mikrovoltteihin [32, s. 33], kun sydänkäyrän amplitudi voi olla jopa voltin luokkaa [32, s. 419]. Toisaalta sydänkäyrän suurempi arvoalue voidaan esittää samalla määrällä bittejä kuin EEG, sillä sydänkäyrässä näytteen arvoa ei välttämättä (käyttötarkoituksesta riippuen) tarvitse kuvata alle mikrovoltin tarkkuudella. Sörnmo ja Laguna toteavat biosähköisten signaalien tarkkuudeksi riittävän yleensä 12–14 bittiä, olettaen että hyvin matalat taajuudet ja tasakomponentti on poistettu signaalista [32, s. 15].

Tieto voidaan myös pakata siirron ja/tai varastoinnin ajaksi. Lääketieteellisessä käytössä suositaan häviöttömiä pakkausmetodeja, sillä usein halutaan säilyttää kaikki alkuperäinen informaatio. EEG:n pakkaamiseksi on tutkittu ja ehdotettu useita sekä häviöttömiä että häviöllisiä menetelmiä; jo yksinkertaisella häviöttömällä huffman-koodauksella voidaan päästä jopa 50 % pakkaussuhteeseen [100]. Huonona puolena tiedon pakkaaminen vaatii suoritinaikaa, joka on pois käsittelysovelluksen algoritmeilta. [101; 102]

Formaatin voidaan haluta olevan mahdollisimman helposti laajennettavissa sekä tukevan useita ominaisuuksia, kuten mittaukseen liittyviä metatietoja (esim. potilaan nimi) ja tapahtumia. Formaatti voi tukea useita bittitarkkuuksia, joista käyttäjä voi valita haluamansa tietoverkon kapasiteetista riippuen – tai vaihtoehtoisesti resoluutio voidaan automaattisesti sovittaa käytettävissä olevaan verkon kapasiteettiin. Esitystarkkuuden automaattinen muuttaminen voi toki johtaa ongelmiin palvelimella suoritettavissa algoritmeissä.

Yksi lähestymistapa on siirtää tieto jossakin jo-olemassa olevassa laajalti käytetyssä formaatissa. Tämän hyviä puolia ovat helppo integroitavuus muihin sovelluksiin ja mahdollisesti saatavilla olevat valmiit kolmansien osapuolien ohjelmistolliset toteutukset (esim. kyseisen formaatin käsittelyyn soveltuvat ohjelmistokomponentit). Huonona puolena ollaan sidottuja valitussa formaatissa mahdollisesti oleviin rajoitteisiin.

Eräs tällainen formaatti on nimeltään *European Data Format* (EDF). Se on pyritty suunnittelemaan yksinkertaiseksi ja joustavaksi monikanavaisten fysiologisten signaalien siirtoon ja varastointiin. Ensimmäinen versio EDF-formaatista julkaistiin vuonna

1992. Vuonna 2003 julkaistiin uudempi versio EDF+, jolla kyetään ilmaisemaan myös mittaukseen kuuluvia tekstuaalisia annotaatioita. Uudempi versio tukee myös ajallisesti ei-jatkuvia mittauksia, eli mittauksissa voi olla taukoja. EDF+ on taaksepäin yhteensopiva, joten sitä noudattavia tiedostoja voidaan avata myös vanhoilla, vain EDF:n ominaisuuksia tukevilla ohjelmilla. Sekä EDF että EDF+ ovat hyvin laajalti eri ohjelmistovalmistajien tukemia [103]. EDF-tiedosto sisältää alkeelliset metatiedot potilaasta (mm. nimi, syntymäaika, sukupuoli) sekä mittauksesta (esim. päivämäärä ja aika). Lisäksi EDF-tiedostojen lukemiseen ja kirjoittamiseen on saatavilla ilmainen .NET-luokkakirjasto [104]. [75; 76]

Yksittäisen näytteen tarkkuus on EDF-formaateissa 16 bittiä – tämän suuremmalla bittiresoluutiolla signaalia ei voida EDF:ssa kuvata. EDF- ja EDF+-formaateille onkin kehitetty 24-bittiset vastineet: BDF ja BDF+ *(Biosemi Data Format)* [105], joka tosin ei ole saavuttanut laajaa hyväksyntää.

Toinen EDF:n rajoite liittyy annotaatioihin: EDF-tiedostossa signaalit on jaettu ajallisesti lohkoihin *(data record)*. Kaikki lohkot ovat samanpituisia ja sisältävät ajallisesti saman verran näytteitä joka kanavasta. Lohkon pituus ja jokaisen siinä olevan kanavan näytteiden määrä ilmaistaan EDF-tiedoston alussa olevissa otsikkotiedoissa *(header)*. Taaksepäin yhteensopivuuden vuoksi annotaatiot on toteutettu yhtenä mittaussignaalina, joten myös ne sijaitsevat lohkon sisällä. EDF+-spesifikaation mukaan annotaatioiden on oltava juuri siinä lohkossa, jonka aika-alueeseen annotaatio kuuluu. Reaaliaikaisen mittauksen siirtäminen EDF-formaatissa on ongelmallista juuri tämän vuoksi; EDF-tiedoston otsikkotiedot luodaan mittauksen alussa ja tällöin on asetettava myös yksittäiseen lohkoon tulevan annotaatiokanavan koko. Tällöin ei ole tietoa, kuinka tiheään annotaatioita voi tulla. Voi ilmetä tilanne, jolloin yksittäisen lohkon ajanjakson aikana luodut annotaatiot eivät mahdu lohkoon. Ongelman esiintymistodennäköisyyttä voidaan pienentää kasvattamalla annotaatiokanavan kokoa lohkossa. Tämä puolestaan kasvattaa jokaista lohkoa – myös niitä joissa ei ole yhtäkään annotaatiota. Ilman erityiskäsittelyä myös tämä tyhjä tila joudutaan siirtämään tietoverkon yli, jolloin osa verkkokapasiteetista kuluu sen siirtämiseen. [75; 76]

Kyseinen ongelma voitaisiin ratkaista esimerkiksi kuljetus- ja EDF-kerroksen väliin asetettavalla protokollalla, joka on tietoinen välittämänsä EDF:n sisällöstä. Tämä protokolla voisi kutistaa jokaisen lohkon sisällä olevan annotaatiokanavan pienimpään mahdolliseen kokoon sen siirron ajaksi. Palvelimella vastaava kerros laajentaisi kanavan takaisin ennalleen ennen kuin se tallennetaan EDF-tiedostoon. Vaihtoehtoisesti kyseinen protokolla voisi myös pakata koko lohkon.

Myös oman, esimerkiksi EBML-pohjaisen *(Extensible Binary Meta Language)* [106], formaatin luomista EEG-mittaustiedon reaaliaikaiseen siirtoon olisi kannattavaa tutkia. Täten voitaisiin päästä tehokkaampaan tilankäyttöön sekä tiedon siirrossa että varastoinnissa. Huonona puolena menetettäisiin yhteensopivuus olemassa olevien sovellusten kanssa.

Riippumatta siitä, mihin formaattiin päädytään, voi sovelluskerroksen protokolla toimia esimerkiksi seuraavanlaisesti: mittauksen käynnistyessä lähettäjä aloittaa istunnon

palvelimen kanssa. Tämä tapahtuu tarkoitukseen sopivalla viestillä, jossa lähettäjä kertoo mm. potilaan nimen ja mitattavien kanavien ominaisuudet (nimi, näytteenottotaajuus, bittitarkkuus). Palvelin luo tämän perusteella säilön, johon mittaustiedot tullaan tallentamaan. Säilönä voi toimia esimerkiksi yksittäinen tiedosto. Näyteikkunan saatuaan lähettäjä paketoi sen yhdeksi tai useammaksi viestiksi, numeroi nämä viestit niiden järjestyksen mukaan ja lähettää ne palvelimelle. Palvelin vastaanottaa viestit, järjestää ne oikeaan järjestykseen, purkaa niistä mittaustiedon ja kirjoittaa tämän istuntoon kuuluvaan säilöön. Lisäksi palvelin toimittaa lähettäjälle kuittauksen, jossa se ilmoittaa vastaanottamiensa viestien numerot. Lähettäjä pitää kirjaa lähettämiensä viestien numeroista ja uudelleenlähettää ne viestit, joihin ei ole saatu kuittausta tietyn ajan sisällä. Kun mittaus on päättynyt ja lähettäjä on toimittanut kaiken mittaustiedon se ilmaisee palvelimelle, että enempää tietoa ei tule. Palvelin sulkee säilön, merkitsee mittauksen päättyneeksi ja sulkee yhteyden.

5.4.3 Tietoturva

Vaikka tässä työssä toteutettava sovellus on suunnattu vain tutkimuskäyttöön, on silti syytä pohtia sen tietoturvaa. Myös tutkimuskäytön sovelluksissa tietoturvallisuudella on merkitystä; ei haluta, että tutkimuksessa potilaalta kerättyä mahdollisesti arkaluontoista tietoa päätyy vääriin käsiin. Lisäksi halutaan säilyttää varmuus siitä, ettei tieto ole muuttunut sitä siirrettäessä – tämä vähentäisi tutkimuksessa saatujen tulosten luotettavuutta. Potilasturvallisuutta tiedon muuttuminen ei kuitenkaan pysty vaarantamaan, sillä kuten johdannossa alleviivattiin, tässä työssä toteutetun sovelluksen läpi kulkenutta tietoa ei voida käyttää päätöksenteon tukena potilaan hoitoon liittyvissä asioissa.

Matkallaan mittauspaikalta tilaajille (kuva 1.1) mittaustietoa siirretään useaan otteeseen ja säilötään useisiin paikkoihin. Tässä luvussa käsitellään tietoturvaa vain tiedonsiirron näkökulmasta; esimerkiksi palvelimella varastoidun tiedon tai käsittelysovelluksen puskurien suojaukseen ei oteta kantaa. Luvussa oletetaan, että mittauspaikalla toimivan käsittelysovelluksen ympäristö on turvallinen. Palvelintoteutus tai sen tietoturva eivät kuulu tämän työn aihepiiriin.

Verkon tietoturvaa voidaan tarkastella esimerkiksi Stallingsin mallin mukaan. Se jakautuu neljään osa-alueeseen: luottamuksellisuus *(confidentiality)*, eheys *(integrity)*, saatavuus *(availability)* ja tunnistettavuus *(authenticity)*. Luottamuksellisuudella tarkoitetaan, että vain sallitut tahot pääsevät tietoon käsiksi – tai edes tietävät sen olemassaolosta. Eheys merkitsee että vain sallitut tahot voivat muokata (esim. muuttaa, kirjoittaa, poistaa tai luoda) tietoa. Saatavuus puolestaan kuvaa sitä, että tieto on saatavilla oikeaan aikaan ja tunnistettavuudella varmistetaan että osapuolet ovat niitä joita väittävät olevansa. Saatavuus koskee osittain reaaliaikavaatimuksia, joita käsiteltiin luvussa 5.2. [82, s. 703]

Luottamuksellisuuden ja eheyden tukemiseksi tieto voidaan salata; yhdessä tarkistussummien kanssa tällä voidaan varmistaa ettei kolmas osapuoli voi lukea tai muokata tietoa. Yksin salaus ei täytä täysin kaikkia luottamuksellisuuden vaatimuksia; kolmas

osapuoli voi edelleen tehdä johtopäätöksiä tietoliikenteen määrän ja laadun perusteella (esim. milloin mittaus alkaa tai päättyy, sekä missä mitataan). Tunnistettavuus voidaan toteuttaa hyödyntäen asymmetrisiä salausmenetelmiä (digitaaliset allekirjoitukset) [82, luku 21.4]. Saatavuuden takaamiseksi on turvattava riittävät tiedonkäsittelyresurssit (verkkoyhteydet, prosessoriaika, muistin määrä) sekä kyettävä näitä säännöstelemään – tähän kuuluu myös suojautuminen mahdollisilta palvelunestohyökkäyksiltä.

Salaukseen on ainakin kaksi lähestymistapaa; se voidaan esimerkiksi toteuttaa sovelluksen protokollapinossa TLS-protokollalla *(Transport Layer Security)* [82, luku 21.5; 107]. Toinen vaihtoehto on yhdistää mittauspaikan lähiverkko sairaalan lähiverkkoon tunneloimalla koko verkkoyhteys VPN-yhteyden *(Virtual Private Networking)* kautta. Jos sairaalalla on käytössään VPN-ratkaisu, on se näistä kahdesta vaihtoehdosta vaivattomampi, ollen myös yhteensopivampi sairaalan palomuurien kanssa. Myös VPN-toteutus saattaa hyödyntää salauksessaan TLS-protokollaa. Huonona puolena VPN-yhteyttä hyödyntävässä ratkaisussa saattaa jäädä enemmän hyökkäyspinta-alaa; jos pahansuopa taho pääsee mittauspaikan lähiverkkoon, voi tästä olla VPN-yhteyden yli esteetön pääsy sairaalan lähiverkkoon.

Salatun tiedonsiirtoyhteyden tuoma turva ei ole ilmaista; lähetettävän mittaustiedon määrästä ja käytetystä salausalgoritmista riippuen salaus voi vaatia hyvinkin paljon prosessoritehoa. Tämä teho on luonnollisesti pois mittauspaikalla toimivien NeurOne- tai käsittelysovelluksen käytöstä.

Jos sekä lähettävä sovellus että mittaustietoa vastaanottava palvelin sijaitsevat luotetussa verkossa, kuten esimerkiksi sairaalan lähiverkko, ei tietoa tarvinne erikseen suojata. Toisaalta esimerkiksi laajakaista- tai 3G-yhteyden yli tietoa välitettäessä salaus voi olla tarpeen.

6 KÄSITTELYSOVELLUKSEN TOTEUTUS

Aiemmissa luvuissa muodostettiin kuva työtä ympäröivästä sovellusalueesta ja ympäristöstä, olemassa olevista vastaavanlaisista ratkaisuista sekä käsiteltiin sovellusalueen problematiikkaa. Tässä luvussa työ konkretisoituu sovelluksen ohjelmistollisen toteutuksen muodossa. Koko sovelluksen toimintaa ja toteutusta ei käsitellä. Esimerkiksi toiminnallisuus, jolla käyttäjä voi suunnitella luvussa 5.1 määriteltyjä käsittelyprotokollia on jätetty kuvaamatta, koska se ei ole olennainen osa sovelluksen toimintaa, vaan pikemminkin aputyökalu. Luku alkaa perustelemalla miksi toteutuksessa päädyttiin käyttämään .NET-ohjelmistoalustaa (6.1). Seuraavaksi kuvataan sovelluksen kokonaisarkkitehtuuri luvussa 6.2 ja edetään käsittelemään kukin arkkitehtuurin osa omana lukunaan (luvut 6.3–6.6).

Osa tässä luvussa käytetyistä kuvista sisältää englanninkielisiä termejä. Esimerkiksi ohjelmistokomponenttien nimet on esitetty sekaannuksen välttämiseksi sellaisina, kuin ne ilmenevät toteutetussa sovelluksessa. Lisäksi luku sisältää tietokonetekniikan termejä, jotka lukijan oletetaan tuntevan. Tekstissä ohjelmiston osiin (muuttujat, tyypit, luokat, attribuutit, metodit jne.) tehdyt viittaukset on tekstin selkeyttämiseksi esitetty eri fontilla.

6.1 Ohjelmistoalustan valinta

Työssä käytetty NeurOne-sovellusohjelmisto on toteutettu Microsoft .NET -sovelluskehyksellä; täten myös NeurOne-liitännäiset (luku 4.3) on toteutettava kyseistä alustaa käyttäen. Eräs ensimmäisenä esitettävistä kysymyksistä on se, käytetäänkö samaa alustaa myös käsittelysovelluksen toteutuksessa.

Hyviä puolia .NET-sovellusalustassa ovat automatisoitu muistinhallinta ja roskienkeruu *(garbage collection)*, helppo käyttöliittymäsuunnittelu, ajonaikainen tyypitys sekä alustariippumattomuus (sama lähdekoodi kääntyy 32- ja 64-bittisille prosessoriarkkitehtuureille). Roskienkeruun ja ajonaikaisten turvatarkastuksien ansiosta muistivuodot ja virheelliset osoittimet *(pointer)* ovat harvinaisia. Tämä parantaa ohjelman luotettavuutta etenkin pitkillä suoritusajoilla. Ajonaikainen tyypitys ja kirjastoihin sisäänrakennetut tyyppien metatiedot mahdollistavat jo käynnissä olevien ohjelmien helpon ajonaikaisen muuntelun. Sovelluksen ulkoisten luokkakirjastojen lataaminen sekä niissä olevien tyyppien tarkastelu ja käyttäminen ovat ohjelmoijalle vaivattomia toteuttaa. Tämän ansiosta .NET-sovelluksista on erittäin helppo tehdä modulaarisia, joka puolestaan helpottaa niiden kehittämistä, laajentamista, räätälöintiä ja ylläpitoa. Yksinkertaisuuden ja helppouden ansiosta tuotettu ohjelmakoodi sisältää vähemmän virheitä ja sitä on helpompi ylläpitää. Lisäksi sama koodi voidaan mahdollisesti siirtää toimimaan Mono-so-

vellusalustalla, jolloin sovellusta voidaan käyttää esimerkiksi Linux-ympäristössä pelkän Windowsin sijaan.

Erinäisillä turva- ja hallintamekanismeilla on hintansa; ne voivat haitata suorituskykyä. Esimerkiksi taulukon alkioon viitatessa alusta tarkastaa, onko viitattu alkio olemassa. Kääntäjä tosin osaa yleensä optimoida kyseisen turvatarkastuksen siten, ettei tarkastusta suoriteta silmukoissa jokaisen viittauksen yhteydessä. Tarvittaessa tarkastukset voidaan myös kiertää. [108, s. 387]

Ehkä suurin ongelma johon törmätään liittyy tietotyyppimuunnoksiin. Tarkastellaan esimerkiksi seuraavaa tilannetta:

> .NET-ohjelmassa käsitellään näytteitä taulukkona 64-bittisiä liukulukuja (double[ ]). Kyseinen taulukko halutaan lähettää verkkoyhteyden yli, mutta lähetysfunktio hyväksyy syötteen vain taulukkona tavuja (byte[ ]). Tämä on hyvin yleistä, sillä lähettävän funktion on käytännöllistä käsitellä sille lähetettäväksi annettua tietoa ottamatta kantaa tiedon luonteeseen. Monissa muissa ohjelmointikielissä voitaisiin luoda tyhjä tavutaulukon osoitin ja asettaa se osoittamaan liukulukuja sisältävään muistialueeseen *(pointer casting)*. Tämä ei kuitenkaan ole mahdollista .NET-ympäristössä. Liukulukuja sisältävään muistialueeseen ei voida suoraan viitata tavutaulukkona (koska .NET-taulukko ei ole pelkkä osoitin [108, s. 386]), vaan on varattava erillinen tavutaulukko ja kopioitava näytteet siihen.

Näytteisiin ei siis voida viitata toisena tietotyyppinä, vaan niistä on tehtävä kopio. Tätä on havainnollistettu koodiesimerkillä liitteessä 3. On hyvin todennäköistä, että signaaleja käsittelevässä sovelluksessa signaaleja käsitellään taulukoina. Edellä esitetyn kaltaisia tietotyyppimuunnoksia voidaan tarvita esimerkiksi siirrettäessä tietoa sovelluksesta toiseen, tallennettaessa tietoa tiedostoon tai lähetettäessä sitä verkkoyhteyden yli, eli prosessointiverkon alku- ja loppupäissä, jossa tietoa välitetään käsittelyistuntoon tai sieltä ulos. Mitä suurempia määriä tietoa käsittelyistunnon läpi kulkee, sitä suurempi on edellä mainitun kopioinnin aiheuttama kustannus.

Tämän projektin yhteydessä kuitenkin tulkittiin edellä esitettyjen hyötyjen voittavan haitat. Tärkeintä oli valita ohjelmistoympäristö, jossa sovelluskehitys on nopeaa ja vaivatonta mahdollistaen käsittelysovelluksen toteutuksen tämän diplomityön aikakehyksessä. Helpon käyttöliittymäsuunnittelun ja modulaarisuuden vuoksi .NET-alusta sopii hyvin tämän työn tyyppiseen suhteellisen laajaan projektiin, jossa rakennetaan myöhemmin mahdollisesti laajennettava prototyyppi. Lisäksi Mega Elektroniikka Oy voi halutessaan ottaa sovelluksesta yksittäisiä komponentteja ja integroida ne NeurOne-sovellukseen.

6.2 Sovelluksen arkkitehtuuri

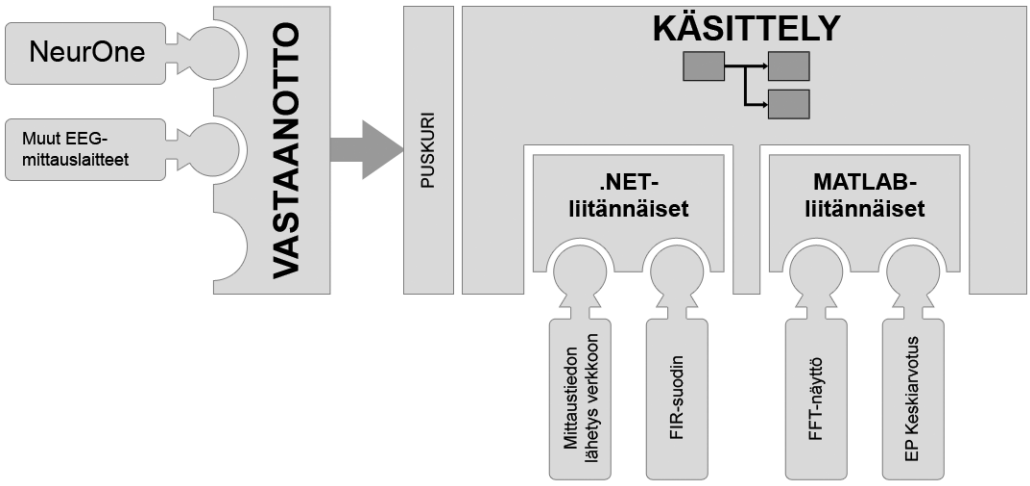
Luvussa 5.2 signaalien kulku sovelluksen läpi jaettiin kolmeen osaan ongelma-alueiden mukaisesti. Näitä olivat mittaustiedon vastaanotto, käsittely ja lähetys. Työssä toteutetun

käsittelysovelluksen arkkitehtuuri (kuva 6.1) noudattaa osittain näitä aluejakoja; siinä on erillinen mittaustietoa vastaanottava komponentti sekä erillinen tietoa käsittelevä komponentti. Tiedon lähetystä kohdellaan arkkitehtuurissa eräänlaisena käsittelynä, eli tietoa eteenpäin lähettävät komponentit ovat käsittelyistunnossa toimivia algoritmeja (luku 5.1). Algoritmi voi käytännössä olla mikä tahansa ohjelmistokomponentti; sen kehittäjällä on täysi valta toiminnallisuuden suhteen. Tästedes näistä komponenteista käytetään nimitystä *prosessointilohko*.

Tässä työssä tiedon vastaanotto tapahtuu NeurOne-sovellukselta. Tiedon vastaanottovaihe on kuitenkin suunniteltu sellaiseksi, että se voidaan sovittaa helposti muihinkin mittaussovelluksiin. Sovitus edellyttää, että mittaustietoa tuottavaan ohjelmistoon lisätään käsittelysovelluksen kanssa kommunikointiin tarvittava toiminnallisuus (esim. liitännäisenä). Kuten luvussa 3.1 ilmaistiin, voidaan monen muunkin valmistajan mittausohjelmistoja tällä tavoin laajentaa. NeurOne- ja käsittelysovelluksen välinen yhteys kuvataan luvussa 6.3.

Koska käsittelysovellukselle voidaan syöttää tietoa nopeammin kuin se ehtii käsitellä, täytyy tämä tieto säilöä jonnekin odottamaan myöhempää käsittelyä (kuva 6.1, puskuri). Kyseisen puskurin toiminta on kuvattu luvussa 6.4.

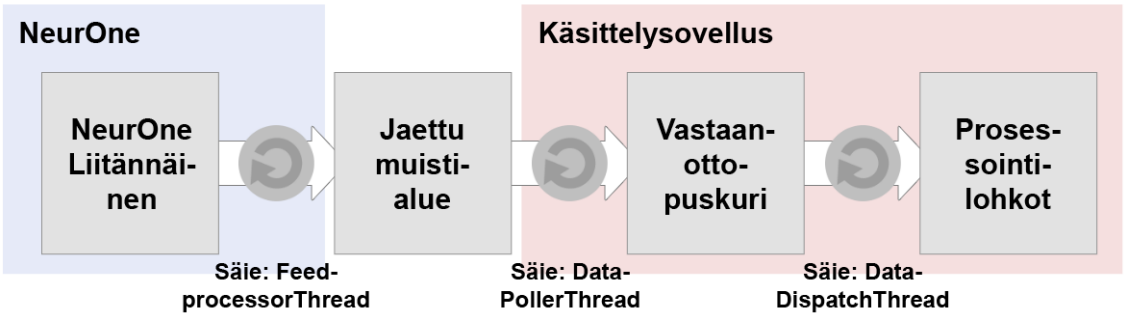
Sovelluksen tulee tukea MATLAB-algoritmeja, joita käyttäjän täytyy kyetä lisäämään ja poistamaan. Käsittelyalgoritmeja ei voi toteuttaa kiinteästi osana käsittelysovellusta, koska tällöin näiden lisäys ja poisto vaatisi sovelluksen uudelleen kääntämisen. Hyvin toteutetulla abstraktiolla sovellusta on helpompi laajentaa myöhemmin tukemaan myös muilla tekniikoilla toteutettuja algoritmeja (luku 6.5.1).



Kuva 6.1. *Käsittelysovelluksen arkkitehtuuri.*

Mittaustiedon kulku NeurOne-sovellukselta prosessointilohkoille on esitetty kuvassa 6.2. Tiedonsiirtoon osallistuu kolme säiettä; FeedProcessorThread toimii NeurOne-liitännäisen sisällä, siirtäen mittaustietoa NeurOne-sovelluksesta jaetulle muistialueelle. Käsittelysovelluksen sisällä toimiva DataPollerThread puolestaan siirtää tietoa jaetulta muistialueelta vastaanottopuskuriin, josta DataDispatchThread syöttää sitä prosessointilohkoille.

Kaikkia signaaleja käsitellään sovelluksen sisällä 64-bittisinä liukulukuina (double). Täten pyritään kuvaamaan yhdellä tietotyypillä mahdollisimman suuri arvoalue. Pitäytymällä yhdessä tietotyypissä on sovelluksen ja algoritmien kehitys yksinkertaisempaa. Haittapuolena näytteiden säilömisen ja käsittelyn muistivaatimukset kaksinkertaistuvat NeurOnen 32-bittiseen näytetarkkuuteen verrattuna.



Kuva 6.2. *Mittaustiedon kulkeminen NeurOne- ja käsittelysovelluksen välillä.*

Kuvassa esitetyn lisäksi käsittelyyn liittyy muitakin säikeitä ja puskureita. NeurOne-sovelluksen sisällä toimii säikeitä, jotka toimittavat tietoa liitännäiselle. Lisäksi prosessointilohkoissa voi niiden toteutuksesta riippuen olla puskureita ja käsittelysäikeitä. Jaetun muistialueen toiminta on esitetty luvussa 6.3.2, vastaanottopuskurin luvussa 6.4 ja prosessointilohkojen luvussa 6.5.

6.3 NeurOne- ja käsittelysovellusten eristys

Eräs tärkeimmistä arkkitehtonisista päätöksistä oli, kuinka läheisesti työssä toteutettava mittaustietoa käsittelevä komponentti integroidaan NeurOne-sovellukseen. Työssä päädyttiin eristämään käsittely erilliseen sovellukseen kahdesta syystä. Näistä tärkein on tietojenkäsittelyn turvallisuus; jos käsittelyssä ilmenee vikatilanne josta ei kyetä toipumaan, halutaan minimoida vaikutus NeurOne-mittauslaitteiston ja -ohjelmiston toimintaan. Vaikka tuotettu ohjelmistokomponentti testataan huolella ennen sen käyttöönottoa oikeissa mittauksissa, ei pystytä takaamaan ettei vikatilanteita ilmene. Tilannetta pahentaa se, ettei komponentti ole suljettu järjestelmä: käyttäjät voivat laajentaa sen toimintaa lisäämällä siihen omia käsittelyalgoritmejaan.

Toinen eristystä puoltava seikka juontaa juurensa NeurOne-sovelluksen liitännäisten arkkitehtuurista (luvut 4.2 ja 4.3). Liitännäisten asetuksille ei ole mahdollista tuottaa omaa käyttöliittymää, vaan kaikki asetukset syötetään taulukkomuotoisen käyttöliittymän kautta (liite 1). Tämä käyttöliittymä soveltuu vain tekstuaalisesti kuvattavissa olevan atomaarisen tiedon ilmaisuun. Lisäksi NeurOne-liitännäisen toiminta on sidottu mittaustapahtumaan; kun käyttäjä lopettaa mittauksen NeurOne-sovelluksessa, myös kyseiseen mittaukseen kuuluvat liitännäiset sammutetaan. Tästä seuraa ongelmia esimerkiksi tilanteessa, jossa liitännäinen ei ole ehtinyt lähettää kaikkea mitattua tietoa verkkoyhteyden kautta.

Sovellustason eristys tarkoittaa käytännössä sitä, että käsittely tapahtuu erillisessä

prosessissa. Windows-käyttöjärjestelmässä käsite prosessi tarkoittaa ohjelman ilmentymää – esimerkiksi muistio-sovellus toimii omana prosessinaan (notepad.exe). Muistioita voi käynnistää kaksi kappaletta, josta tuloksena on kaksi prosessia joilla kummallakin on oma tilansa. Se, mitä käyttäjä kirjoittaa toiseen muistioon ei millään tavalla vaikuta toiseen. Prosessi koostuu sovelluksen resursseista (muisti, tiedostokahvat, ynnä muut) ja säikeistä (aktiviteetti, suorituksen tila). [79, s. 5]

Windows-käyttöjärjestelmässä ohjelmat on eristetty toisistaan prosessitasolla. Prosessi ei voi esimerkiksi käsitellä toisen prosessin muistia ilman erillistä lupaa. Tällä taataan se, etteivät prosessit voi vahingossa muokata toisten prosessien muistia ja/tai resursseja esimerkiksi virheellisen osoittimen tai kahvan *(handle)* johdosta, ja täten korruptoida tai kaataa niitä *(crash)*. Juuri tämän ominaisuuden johdosta NeurOne- ja käsittelysovellus päätettiin eristää toisistaan. [79, s. 654]

Sovellustason eristys NeurOnesta tarjoaa myös olennaisen edun: käsittelysovellus ei ole sidottu NeurOne-ohjelmistoon, vaan sitä voidaan käyttää muidenkin vastaavien ohjelmistojen kanssa. Lisäksi käsittelysovelluksen testaaminen helpottuu huomattavasti, sillä siihen ei eristyksen johdosta tarvita NeurOne-laitteistoa ja -sovellusta. Haittapuolena eristys huonontaa suorituskykyä: mittaustietoa täytyy kopioida NeurOne-pääohjelmasta käsittelysovellukseen erillisellä mekanismilla, sillä prosessit eivät edellä mainittujen suojausmekanismien johdosta voi viitata toistensa muistialueisiin[33].

Kuten aiemmin linjattiin, eristyksen ensisijainen tehtävä on suojata NeurOne-sovelluksen toimintaa. Vakavissa virhetilanteissa tietojen käsittelyn turvallisuus pyritään varmistamaan nopealla irroittautumisella; jos käsittelysovellus lakkaa vastaamasta, purkaa liitännäinen mittaus- ja käsittelysovelluksen välisen kytköksen ja passivoi itsensä. Tämä ei vaikuta NeurOne-mittaukseen, vaan se jatkuu normaalisti. Passivoitumisen jälkeen NeurOne ei ole enää millään tapaa kytköksissä käsittelysovellukseen, eikä käsittelysovellus voi saada enempää mittaustietoa.

Tiedon saamiseksi NeurOne-sovelluksesta on olemassa useita eri lähestymistapoja: NeurOnen kirjoittaman datatiedoston lukeminen, mittaustiedon kaappaaminen NeurOne-sovelluksen ja laitteiston välisestä tietoliikenteestä, sekä NeurOne-liitännäisen (plugin) kehittäminen. NeurOnen kirjoittaman datatiedoston lukeminen hylättiin, koska tämä olisi tehnyt mittaustiedon saamisesta epävarmaa: käyttöjärjestelmä ei välttämättä kirjoita mitattua tietoa välittömästi tiedostoon, vaan se voi puskuroitua muistiin – tämä olisi myös lisännyt viivettä. Myös NeurOne-sovelluksen ja laitteiston välisen tietoliikenteen kaappaaminen hylättiin, koska tämä olisi vaatinut erillisen verkkokaappain-ajurin kehittämistä, sekä mahdollisesti häirinnyt NeurOne-sovelluksen ja laitteiston toimintaa. Kahdessa edellä kuvatussa vaihtoehdossa ongelmaksi olisi muodostunut myös yhteensopivuus; NeurOnen oma tiedostoformaatti tai tiedonsiirtoprotokolla voisi muuttua, estäen ohjelmaa lukemasta mittaustietoa näillä menetelmillä. Ainoaksi realistiseksi vaihtoehdoksi osoittautui NeurOne-liitännäisen kehittäminen, mikä oli myös Mega Elektro-

33. Jos prosessi omaa riittävät oikeudet, se voi lukea ja kirjoittaa toisten prosessien muistialueille käyttäen Windowsin tarjoamaa ReadProcessMemory ja WriteProcessMemory API:a. Tämä ei tosin ole järin siisti tapa sovellusten välisen kommunikaation toteuttamiseksi.

niikka Oy:n tarjoama ja tukema tapa toteuttaa haluttu toiminnallisuus. Tässä luvussa ei käsitellä liitännäisen rakennetta tai toimintaa, vaan keskitytään kuvaamaan sovellusten väliset ohjaus- ja mittaustiedon siirtokanavat.

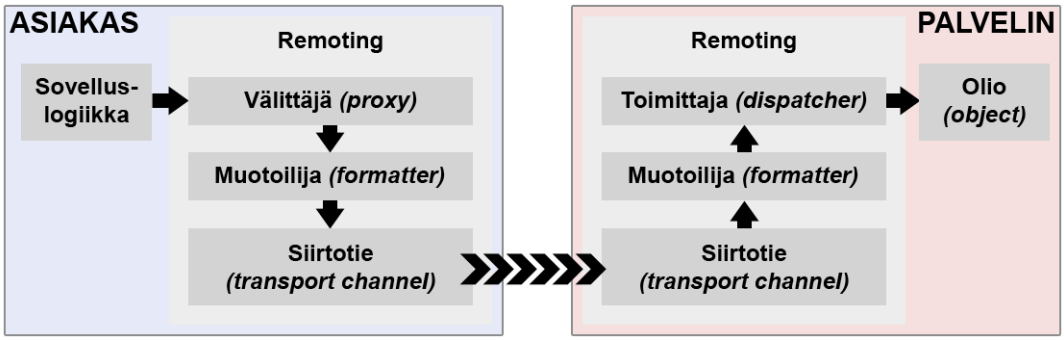
NeurOne- ja käsittelysovelluksen välinen kommunikaatio jaettiin kahteen kanavaan, joista yksi välittää ohjaus- ja toinen mittaustietoa. Näistä mittaustieto kulkee vain yhteen suuntaan (NeurOne-sovellukselta käsittelysovellukselle), on aikakriittistä ja voi vaatia huomattavan paljon tiedonsiirtokapasiteettia. Suhteessa tähän ohjaustietoa on hyvin vähän, mutta se on kaksisuuntaista muodostuen kutsuista, joihin kutsuja joutuu odottamaan vastausta. Ohjauskanavan toiminta käsitellään luvussa 6.3.1 ja mittaustietoa välittävän tiedonsiirtokanavan toiminta luvussa 6.3.2.

6.3.1 Ohjauskanava

Ohjauskanavan pohjaratkaisuksi oli luontevaa valita .NET Remoting sillä se on Microsoftin tukema, .NET-sovelluskehykselle ominainen, erittäin helppo toteuttaa, käyttää ja laajentaa sekä teollisuudessa laajalti käytössä [109, luku 2]. Lisäksi kyseinen tekniikka oli työn toteuttajalle ennestään tuttu. Ohjaustietoa kulkee prosessien välillä hyvin vähän, eikä se ole aikakriittistä. Toisaalta tämä asettaa vaatimuksen mittaustiedon lähteelle, johon käsittelysovellus halutaan liittää: myös sen täytyy tukea .NET remoting -teknologiaa. NeurOnen tapauksessa tämä ei ole ongelma, sillä kyseinen sovellus on toteutettu .NET-alustalla. Tulevaisuudessa tämä ohjauskanava voidaan abstrahoida siten, että se tukee .NET remotingin lisäksi esimerkiksi jotakin TCP/IP-protokollan tai nimettyjen putkien *(named pipes)* päälle rakennettua sovellustason protokollaa.

Ohjauskanavan perusideana on, että liitännäinen kutsuu käsittelysovelluksessa sijatsevan olion palveluita. Remoting-tekniikan ansiosta tämä kaikki on ohjelmoijalle täysin läpinäkyvää. Myös monimutkaisten tietorakenteiden välittäminen kutsujen yhteydessä on ohjelmoijan näkökulmasta vaivatonta.

NeurOne-mittausistunnon käynnistyessä liitännäinen alustetaan (Initialize-operaatio, luku 4.3). Tämän operaation aikana liitännäinen luo MeasurementSession-luokan ilmentymän. Kyseinen luokka on määritetty toimimaan remoting-kanavan yli: sen luominen avaa yhteyden paikalliseen TCP/IP-porttiin 8900, jossa käsittelysovelluksen remoting-rajapinta odottaa palvelupyyntöjä. Remoting tuottaa MeasurementSession-luokan CAO-ilmentymän *(Client Activated Object)* käsittelysovelluksen sisälle (kuva 6.3, olio) ja palauttaa liitännäiselle kyseiseen ilmentymään osoittavan välittäjän (kuva 6.3, välittäjä). Liitännäisen (kuva 6.3, sovelluslogiikka) kutsuessa välittäjän operaatioita ohjaa remoting kyseiset kutsut käsittelysovellukseen, jossa kutsu suoritetaan. Tämä yhteys toimii kaksisuuntaisena kommunikaatiokanavana liitännäisen ja käsittelysovelluksen välillä.

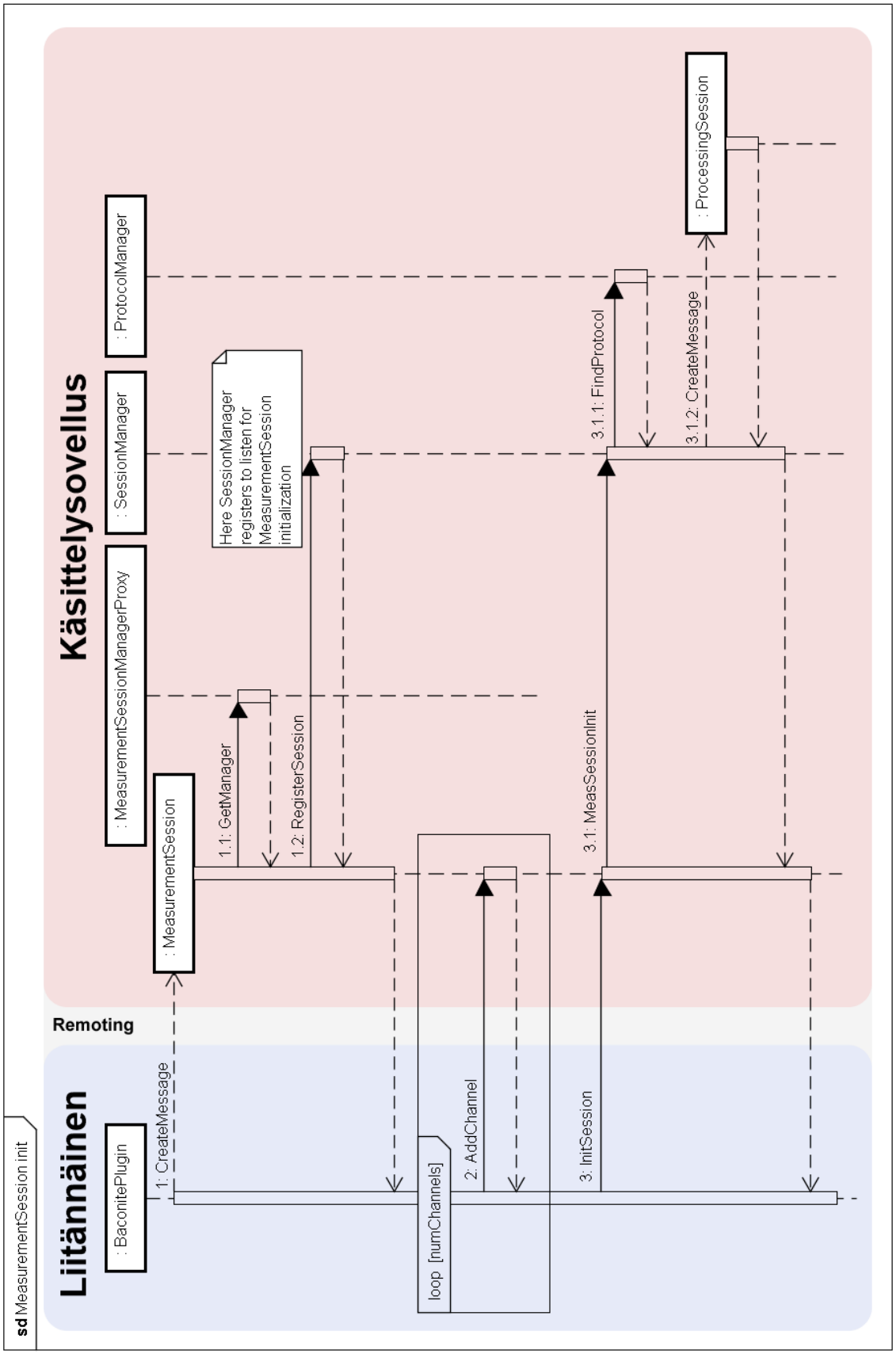


Kuva 6.3. *Remotingin arkkitehtuuri (yksinkertaistettu) [109].*

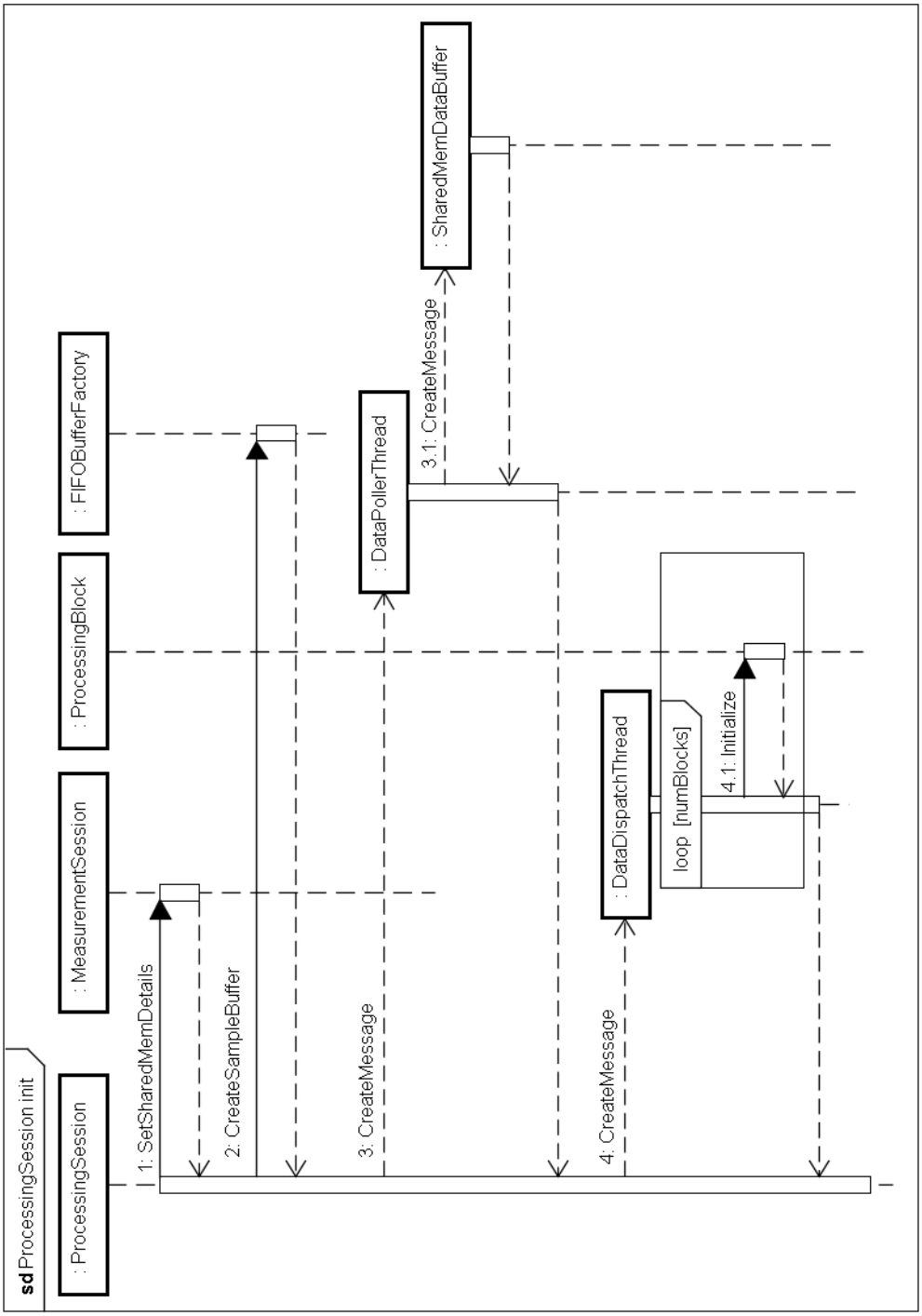
MeasurementSession-välittäjän saatuaan liitännäinen kokoaa kanavien sekä potilaan tiedot mittauksesta ja kutsuu välittäjän InitSession-operaatiota, antaen sille kokoamansa tiedot (kuva 6.4). Kutsu välittyy käsittelysovellukselle, joka valitsee mittauksen tietoihin sopivan käsittelyprotokollan ja käynnistää tähän pohjautuvan käsittelyistunnon (ProcessingSession). Käsittelyprotokollan valinta tapahtuu mittauskanavien sekä liitännäisen asetuksissa olevan tunnisteen perusteella.

Aluksi käsittelyistunto luo tiedon vastaanottopuskurin (luku 6.4) kutsumalla FIFOBufferFactory-luokan CreateSampleBuffer-operaatiota (kuva 6.5) ja käynnistää tietoa vastaanottavan säikeen (DataPollerThread). DataPollerThread puolestaan luo NeurOne-liitännäisen ja käsittelysovelluksen välillä käytettävän mittaustiedon siirtokanavan SharedMemDataBuffer (kuvattu luvussa 6.3.2) ja jää odottamaan, että liitännäinen syöttää kyseiseen kanavaan tietoa. Tämän jälkeen käsittelyistunto luo tietoa prosessointilohkoille syöttävän säikeen (DataDispatchThread).

Kun käyttäjä päättää NeurOne-mittausistunnon (luku 4.3, StoppingSession-viesti), kutsuu liitännäinen välittimen FinishSession-operaatiota. Tämä kutsu välittyy Remoting-kanavan kautta käsittelysovellukseen, joka merkitsee käsittelyistunnon tilaan “päättyvä” (kuvattu tarkemmin luvussa 6.5), ja irroittautuu Neurone-liitännäisestä. Tämän jälkeen kontrolli palaa liitännäiselle, joka palauttaa sen edelleen NeurOnelle ja mittausistunto päättyy. Samalla käsittelysovellus jatkaa mahdollisesti puskuroituneen tiedon käsittelyä itsenäisesti kunnes kaikki tieto on käsitelty tai käyttäjä keskeyttää käsittelyn.



Kuva 6.4. *Mittausistunnon alkaminen.*



Kuva 6.5. *Käsittelyistunnon käynnistyminen.*

6.3.2 Mittaustiedon siirtokanava

Microsoft listaa sivuillaan useita tekniikoita prosessien välisen kommunikaatiokanavan toteuttamiseksi [110]; näistä vain osa on riittävän nopeita tässä työssä vaadittavaan mittaustiedon siirtämiseen. Toteutustekniikkoina harkittiin sovellusten välillä jaettua tiedostoa, .NET Remoting-tekniikkaa, TCP/IP-pohjaista kommunikaatiota, nimettyjä putkia sekä jaettua muistialuetta.

Jaetun tiedoston käyttäminen hylättiin, koska se on menetelmänä toteutukseltaan lä-

hes samanlainen jaetun muistin kanssa; kummassakin tietoa siirretään jaetun virtuaalimuistialueen kautta. Jaetun muistin menetelmässä käyttöjärjestelmälle kerrotaan tarkemmin kuinka ko. virtuaalimuistialuetta käsitellään; siinä koko tiedosto avataan yhdeksi suureksi virtuaalimuistialueeksi. Jaetuissa tiedostoissa tätä ei määritetä, jolloin kumpikin tiedostoa käsittelevä sovellus liittää tiedostosta virtuaalimuistiinsa vain pienen ikkunan, jota se joutuu kuljettamaan tiedostoa käsitellessään. Tämä puolestaan voi aiheuttaa kiintolevyn luku- ja kirjoitusoperaatioita, huonontaen suorituskykyä. [79, luvut 9.2.5 ja 9.11]

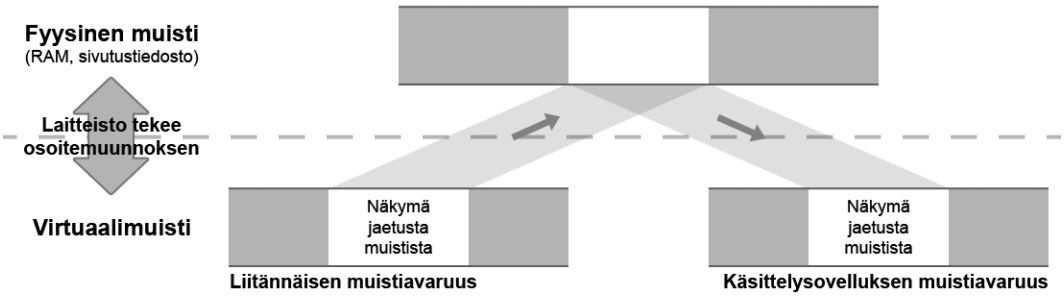
Edellä mainituista menetelmistä Remoting on korkeimman abstraktiotason omaava ja sen avulla mittaustiedon siirtäminen on helppo toteuttaa. Se ei itsessään toteuta varsinaista tiedonsiirtomenetelmää, vaan käyttää kommunikaatiokanavana käyttöjärjestelmän tarjoamaa TCP/IP-yhteyttä tai nimettyjä putkia. Sen tueksi voi rakentaa myös oman kommunikaatiokanavan. Remoting-pohjaisella menetelmällä suorituskyky on siis suoraa tiedonsiirtokanavaa huonompi, mutta helpompi toteuttaa.

Remoting ja TCP/IP-pohjaiset ratkaisut mahdollistavat käsittelysovelluksen siirtämisen erilliselle tietokoneelle. Tämä kuitenkin siirtää tiedon lähetysvarmuutta koskevat ongelmat NeurOne-liitännäiseen, laajentaen kyseisen ohjelmistokomponentin toiminnallisuutta.

Nimetyt putket ja jaettu muisti ovat keskenään siinä mielessä samankaltaisia, että myös nimetyt putket toimivat jaetun muistialueen kautta; niiden sisään- ja ulostulopuskurit varataan käyttöjärjestelmäytimen *(kernel)* muistista, joka on suojattu sivutukselta *(nonpaged pool)*. Tämä muisti ei siis voi päätyä sivutustiedostoon ja sen käyttö on siksi aina nopeaa. Kyseistä muistia on rajallisesti, joten liian suurien puskurien käyttö kuluttaa järjestelmäytimen resursseja. Nimetyt putket voivat toimia myös verkkoyhteyden yli, joskin tällä on huomattava negatiivinen vaikutus suorituskykyyn. [79, luku 12.2.5; 111]

Testattavaksi valittiin viisi eri menetelmää: Remoting-tekniikoista testattiin sekä TCP- että IPC-kuljetuskanavan yllä toimivat. Lisäksi testattiin TCP/IP-kommunikaatiota, nimettyjä putkia ja jaettua muistia. Testit suoritettiin aluksi tekijän kotitietokoneella; näissä jaettu muisti kykeni siirtämään tietoa noin 4-kertaisella nopeudella TCP/IP-siirtoon verrattuna ja 1,5-kertaisella nimettyihin putkiin verrattuna (liite 4). Lisäksi testit tehtiin myöhemmin TTY:n kannettavalla tietokoneella, jossa nimetyt putket osoittautuivat noin sata prosenttia jaettua muistia nopeammiksi.

Alunperin kotitietokoneella tehtyjen mittausten perusteella päädyttiin toteuttamaan tiedonsiirto jaetun muistialueen välityksellä. Tämä mekanismi rajoittaa käsittelysovelluksen fyysisesti samalle tietokoneelle, mutta mahdollistaa erittäin nopean tiedonsiirron sovellusten välillä. Windows-käyttöjärjestelmässä jaettu muisti toimii siten, että kaksi tai useampi sovellusta voivat nähdä samat fyysisen keskusmuistin sivut omassa osoiteavaruudessaan (kuva 6.6). Kun toinen sovellus muokkaa kyseistä muistia, heijastuvat muutokset välittömästi myös toisen sovelluksen virtuaalimuistiin [79, s. 728]. Ongelmaksi tässä menetelmässä muodostuu kuitenkin synkronointi. [79, luvut 1.2.4, 9.2.5 ja 9.11]



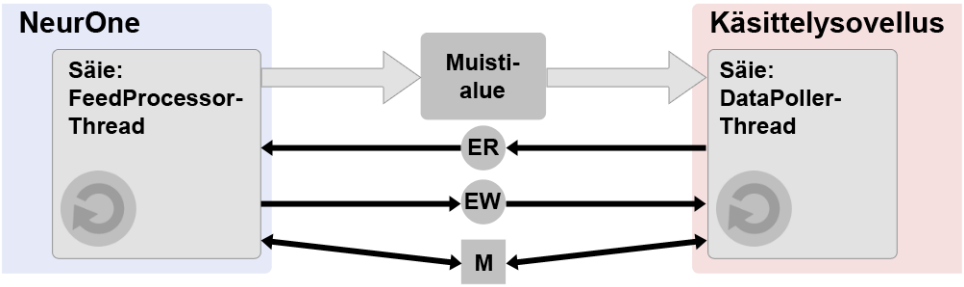
Kuva 6.6. *Prosessien sisäisten osoiteavaruuksien ja fyysisen muistin suhde.*

Moniajokäyttöjärjestelmässä toimiakseen jaettu muistialue tarvitsee rinnalleen mekanismin, joka synkronoi eri toimijoiden pääsyn kyseiselle muistialueelle – ilman tätä voi syntyä pahoja yhteiskäyttöongelmia. Tarkastellaan tyypillistä ongelmatilannetta:

> Säikeeltä A siirretään yhteisen muistialueen kautta tietoa säikeelle B. Säie A on kirjoittamassa tietopakettia jaetulle muistialueelle. Se ehtii kirjoittaa puolet tietopaketista kun käyttöjärjestelmä vaihtaa suoritettavaksi toisen säikeen. Nyt muistialue sisältää puolet tietopaketista sekä jotain muuta, edellisiltä kirjoituskerroilta muistialueelle jäänyttä tietoa. Jos muistialueelta lukeva säie B päätyy suoritukseen ennen kuin säie A ehtii kirjoittaa loput tietopaketista, se lukee virheellisen tietopaketin.

Moniajokäyttöjärjestelmät tarjoavat ongelman ratkaisemiseksi useita synkronointimekanismeja (esim. semaforit). Tästä kokonaisuudesta (jaettu muistialue ja sen synkronointimekanismi) käytetään työssä nimitystä *muistisilta*.

Tässä työssä toteutettu mustisilta on yksisuuntainen; tieto siirtyy ainoastaan NeurOne-liitännäiseltä käsittelysovellukselle. Muistisilta (esitetty kuvassa 6.7) koostuu jaetun muistin alueesta, sen lukitsemisprimitiivistä *(mutex)* M sekä kahdesta järjestelmänlaajuisesta tapahtumaprimitiivistä *(event)* ER ja EW. Tapahtumaprimitiivin EW semanttinen merkitys on “muistialueella on tietoa luettavaksi” ja primitiivin ER “tieto luettu muistialueelta”. Jaettu muisti tarvitsee lisäksi kaksi säiettä; muistialueelle kirjoittava säie (FeedProcessorThread) sijaitsee NeurOne-liitännäisessä ja muistialueelta lukeva säie (DataPollerThread) käsittelysovelluksessa.



Kuva 6.7. *Muistisillan komponentit.*

Lukitsemisprimitiivin M tehtävänä on ehkäistä tilanne, jossa muistialueelta luetaan

ja siihen kirjoitetaan yhtä aikaa. Tietoa siirtävät säikeet signaloivat toisilleen tapahtumien ER ja EW kautta. Kun liitännäisessä toimiva säie alkaa kirjoittamaan muistialueelle, se varaa aluksi lukon M, odottaen ensin tarvittaessa sen vapautumista. Saatuaan lukon säie kirjoittaa tietopaketin muistialueelle, vapauttaa lukon M ja ilmoittaa tästä tietoa lukevalle säikeelle tapahtuman EW avulla (kuva 6.7). Tietoa lukeva säie aktivoituu, varaa lukon M (odottaen tarvittaessa että toinen säie vapauttaa sen), lukee tiedon muistialueelta, vapauttaa lukon M ja ilmoittaa lopuksi kirjoittavalle säikeelle, että muistialueelle voi taas kirjoittaa signaloimalla tapahtuman ER.

Itse muistialue noudattaa liitteessä 5 kuvattua rakennetta; se koostuu yhdestä päälohkosta joka sisältää otsikkotiedot (SMHeader), siirrettävät näytteet (SMSamples) sekä nolla tai useampia tapahtumia (SMEvent). Otsikkolohkossa on muun muassa tieto siitä, montako kanavaa näytelohkossa on sekä kuinka monta tapahtumalohkoa muistialueella on. Jokaisen kanavan yhteydessä kulkee myös ensimmäisen näytteen aikakoodi (SMTimeCode), joka ilmaisee ajanhetken rationaalilukuna. Jokaisen näytteen aikakoodi voidaan ilmaista häviöttömästi rationaalilukuna, sillä luvussa 5.2.1 rajoitettiin näytteenottotaajuudet kokonaisluvuiksi (AP1).

6.4 Puskurointi

Kuten reaaliaikaisuutta käsittelevässä luvussa (luku 5.2) esitettiin, voi ilmetä tilanne jossa laskentaresurssit eivät riitä kaiken sovellukseen saatavan mittaustiedon reaaliaikaiseen käsittelyyn. Tieto täytyy käytännössä puskuroida odottamaan myöhempää käsittelyä, sillä välistä ei saa jättää pois näytteitä (AP6). Luontainen paikka kyseiselle puskurille on vastaanotto- ja käsittelyvaiheen välissä (kuva 5.2). Mittaustiedon kulkureitillä se sijaitsee käsittelysovelluksen sisällä DataPollerThread ja DataDispatchTread -säikeiden välissä (kuva 6.2).

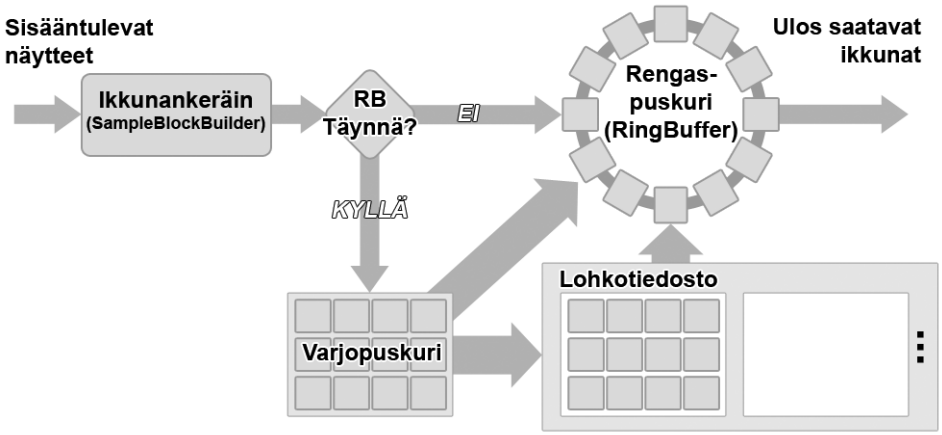
Mittaukset voivat olla ajallisesti hyvinkin pitkäkestoisia ja niissä voi kertyä paljon mittaustietoa (laskelma 4.3). Ei ole kannattavaa sijoittaa kyseistä puskuria täysin virtuaalimuistin varaiseksi, sillä tämän määrä on rajallinen; Windows-käyttöjärjestelmässä prosessin virtuaalimuistiavaruus voi olla 32-bittisellä alustalla korkeintaan kolme ja 64-bittisellä kahdeksan gigatavua [79, s. 643]. Lisäksi sovelluksen virtuaalimuistissa kasvava puskuri voi aiheuttaa käsittelysovelluksen ja muiden samalla tietokoneella toimivien tärkeiden ohjelmien – esimerkiksi NeurOnen – siirtämisen keskusmuistista (*Random Access Memory,* RAM) sivutustiedostoon. Tämä puolestaan lisää levyn luku- ja kirjoitustoimenpiteitä, jotka hidastavat järjestelmän toimintaa. [79, luku 1.2.4]

Puskuri ei myöskään voi sijaita täysin kiintolevyllä sillä tällöin mittaukset, joissa kaikki tieto ehditään käsitellä, voivat aiheuttaa turhaan levyn luku- ja kirjoitustoimenpiteitä. Pahimmassa tapauksessa kaikki käsiteltävä tieto voi kulkea kiintolevyn kautta.

Ratkaisuiksi näihin ongelmiin työssä kehitettiin *first-in-first-out -säännön*[34] toteutta-

34. Tiedon luku puskurista tapahtuu samassa järjestyksessä kuin sen sinne säilöntä. Tämä takaa, että säilöttävien tietoelementtien järjestys säilyy.

va puskuri FIFOBuffer (kuva 6.8), joka säilöö tiedon keskusmuistiin ja tarvittaessa kiintolevyllä sijaitsevaan tiedostoon. Se koostuu keskusmuistissa sijaitsevasta rengas- ja varjopuskurista sekä kiintolevyllä sijaitsevasta lohkotiedostosta. Puskurin luokkakaavio on esitetty kuvassa 6.9. Rengaspuskuri ja lohkotiedosto ovat toteutettu omina luokkinaan (RingBuffer ja SlotFile). Näitä käyttää abstrakti pääluokka FIFOBuffer, josta näytteiden varastointiin erikoistunut luokka SampleFIFOBuffer on periytetty. Koska kyseisen kaltaisia puskureita voidaan tarvita myös muualla sovelluksessa (esimerkiksi tiedon lähetyksessä), eriytettiin puskurista näytteitä käsittelevä toiminnallisuus omiin alaluokkiinsa (kuvassa 6.9 punaisella korostetut). Näiden rinnalle voidaan myöhemmin toteuttaa muunlaista tietoa säilöviä sisarluokkia. Tästedes luvussa viitataan näytteitä säilövään puskurikokonaisuuteen termillä *FIFO-puskuri*.



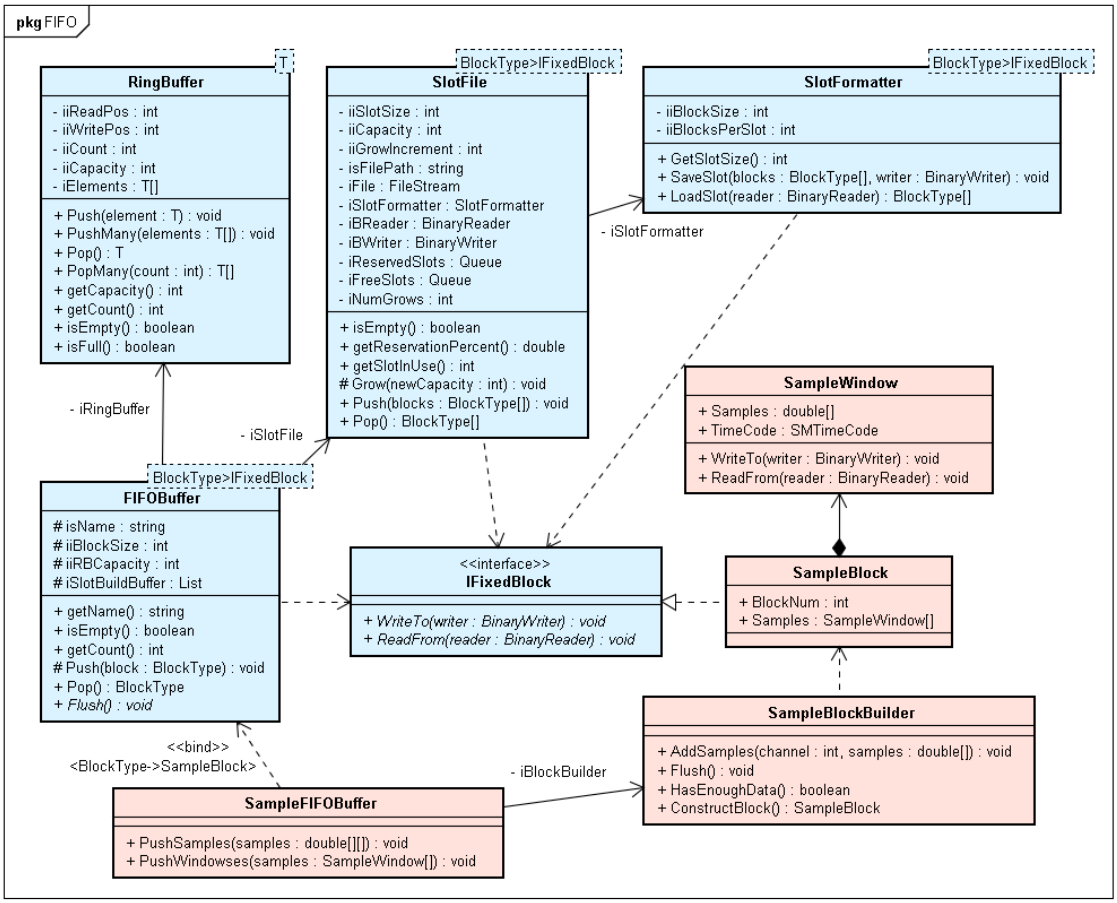
Kuva 6.8. *Kehitetyn FIFO-puskurin arkkitehtuuri.*

FIFO-puskuriin syötetään näytteitä vaihtelevan kokoisina ikkunoina (SampleFIFOBuffer-luokan PushWindowses ja PushSamples -operaatiot). Puskurin sisällä näytteet ajetaan aluksi ikkunankeräimeen (kuva 6.8), joka tuottaa säännöllisen kokoisia ikkunoita ((AP2) ja (AP3)). Yksi ikkuna sisältää jokaisen käsittelysovellukselle syötettävän kanavan mittausinformaation tietyltä ajanjaksolta. Jos ikkunan pituus on yksi sekunti, sisältää ensimmäinen ikkuna jokaisen kanavan mittausinformaation ajalta $[0, 1)$ sekuntia, toinen ikkuna ajalta $[1, 2)$ sekuntia ja niin edelleen. Säännöllisen kokoisella ikkunalla tarkoitetaan, että jokainen ikkuna on ajallisesti saman pituinen; kanavilla voi olla eri näytteenottotaajuudet, joten kanavassa 1 ei välttämättä ole saman verran näytteitä kuin kanavassa 2. Säännöllisellä pituudella varmistetaan kanavien keskinäinen synkronisaatio. Samalla rajoitetaan käsittelyikkunan minimikooksi yksi sekunti (AP1).

Jos rengaspuskurissa on tilaa, säilötään tuotettu ikkuna sinne. Jos ei, säilötään se taustalla toimivaan varjopuskuriin. Varjopuskurissa on aina tilaa yhtä monelle ikkunalle kuin rengaspuskurissa. Jos varjopuskuri on täynnä, tallennetaan sen sisältö kiintolevyllä sijaitsevaan lohkotiedostoon. Lohkotiedosto on jaettu lohkoihin – yksi varjopuskurillinen varastoidaan aina yhteen lohkoon. Jos lohkotiedosto on täynnä, sen kokoa kasvatetaan varaten tilaa tietylle määrälle uusia lohkoja.

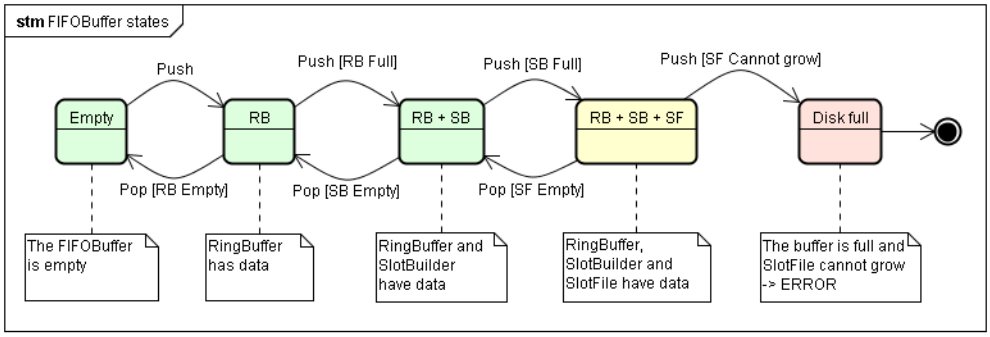
Luettaessa tietoa puskurista (SampleFIFOBuffer-luokan Pop-operaatio) palautetaan vanhin ikkuna rengaspuskurista. Jos rengaspuskuri on tyhjä, sen sisältö ladataan lohkotiedostossa olevasta vanhimmasta lohkosta. Tämä täyttää koko rengaspuskurin. Samalla kyseinen lohko poistetaan lohkotiedostosta, jolloin sen tilan voi käyttää uudelleen säilöttäessä seuraavaa varjopuskurillista. Tämä säästää tilaa kiintolevyllä, sillä lohkotiedosto ei kasva lineaarisesti käytön myötä. Jos lohkotiedosto on tyhjä, siirretään rengaspuskuriin varjopuskurin sisältö.

FIFO-puskuri voi olla jossakin viidestä eri tilasta (kuva 6.10): tyhjä (Empty), tietoa vain rengaspuskurissa (RB), tietoa sekä rengas- että varjopuskurissa (RB+SB), tietoa rengas- ja varjopuskurissa sekä lohkotiedostossa (RB+SB+SF) sekä levy täynnä (Disk full). Näistä viimeisin on virhetila johon saapuminen aiheuttaa aina poikkeuksen.



Kuva 6.9. *FIFO-puskuriin liittyvät luokat.*

Optimaalisessa toimintatilassa puskuri ei käytä tiedon varastointiin kiintolevyä (kolme vasemmanpuolimmaista tilaa kuvassa 6.10). Jos lohkotiedostossa on tietoa, matkaa kaikki puskurin läpi kulkeva tieto kiintolevyn kautta first-in-first-out -säännön takia.



Kuva 6.10. *FIFOBuffer-luokan tilat ja siirtymät niiden välillä.*

Kehitetyn FIFO-puskurin toimintaa voidaan optimoida sovelluskohtaisesti vaihtamalla käytettävän puskurin asetuksia; näitä ovat rengaspuskurin koko, alustava lohkotiedostoon varattujen tyhjien lohkojen määrä, sekä lohkotiedoston kasvatuksessa käytettävä lohkojen määrä. Jos esimerkiksi käsittelyistunnon kuorma ei ole vakio, vaan mittaustiedolle tapahtuu harvakseltaan suhteellisen raskaita operaatioita, voi olla hyödyllistä kasvattaa rengaspuskurin kokoa. Tämä toki lisää käsittelysovelluksen muistivaatimuksia.

6.5 Mittaustiedon käsittely

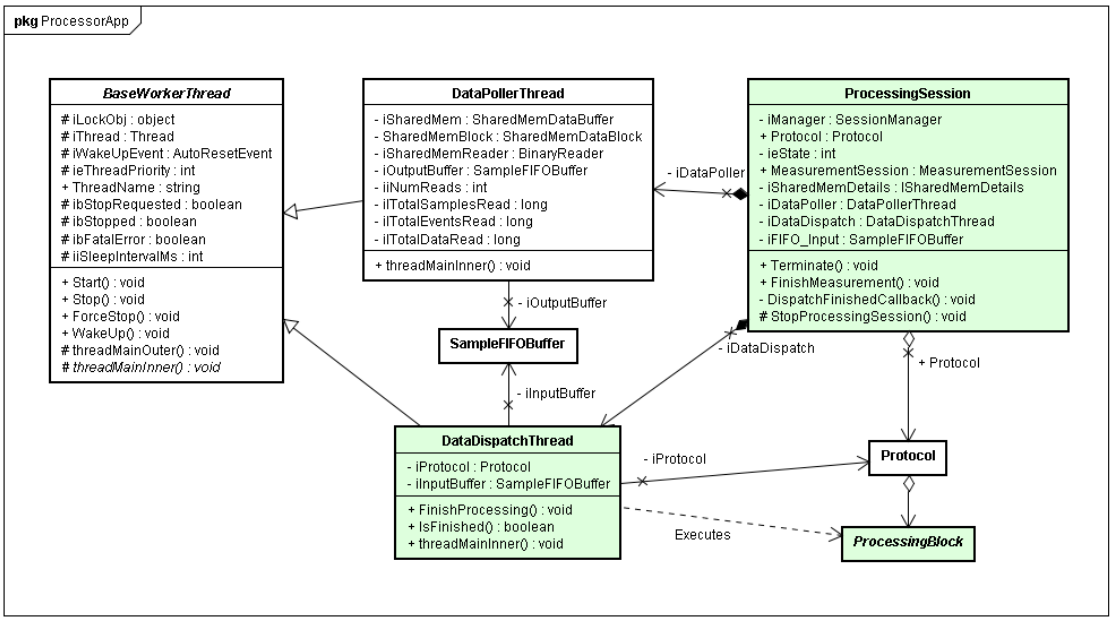
Toteutetussa käsittelysovelluksessa mittaustiedon käsittelyyn liittyy kolme luokkaa: ProcessingSession, DataDispatchThread ja ProcessingBlock (vihreällä kuvassa 6.11). ProcessingSession-luokka kuvaa yksittäistä tiedonkäsittelyyn liittyvää istuntoa käsittelysovelluksen sisällä (käsittelyistunto, luku 5.1). Se liitetään aina mittalaitteen luomaan mittausistuntoon MeasurementSession (kuva 6.4), mutta voi olla olemassa myös mittausistunnon päätyttyä. Tällainen tilanne ilmenee, jos käsittelysovellus ei ole ehtinyt käsitellä kaikkea saamaansa mittaustietoa mittausistunnon päättyessä.

Käsittelyistunto pohjautuu aina tiettyyn käsittelyprotokollaan (Protocol-luokka), joka sisältää ohjeet mittaustiedon käsittelyyn. Ohjeet koostuvat listasta prosessointilohkoja (ProcessingBlock), kunkin prosessointilohkon sisäisistä asetuksista ja tiedosta kuinka lohkot on kytketty toisiinsa. Protokolla huolehtii lohkojen ryhmittelystä peräkkäisiin käsittelyvaiheisiin (AP8), eikä lohkojen välisiä takaisinkytkentöjä sallita (AP7).

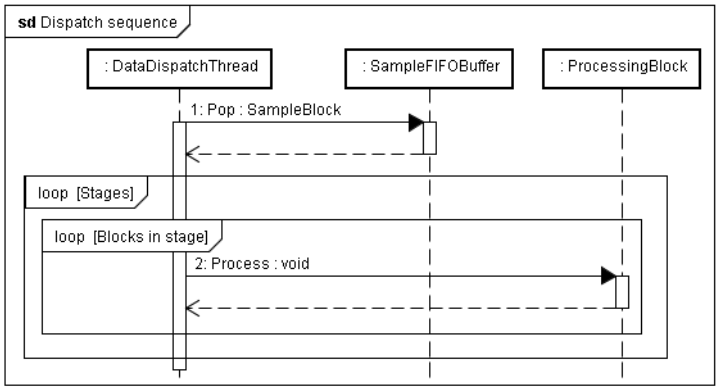
Luvussa 5.3.3 pohdittiin rinnakkaista käsittelyä. Tässä työssä päädyttiin kuitenkin toteuttamaan käsittely yhdellä säikeellä (kuvat 6.2 ja 6.11, DataDispatchThread). Käsittelyalgoritmit suoritetaan siis ajallisesti peräkkäin (kuva 5.7, tapaukset (i) ja (iii)), eikä moniydinjärjestelmien resursseja hyödynnetä täysin. Päätöstä puolsi usea syy; ensinnäkin MATLAB-algoritmeja ei voida ajaa rinnakkain [112], toiseksi samalla alustalla on muitakin prosessoriaikaa tarvitsevia säikeitä (NeurOne, käsittelysovelluksen muut säikeet) ja kolmanneksi työn toteutukseen käytettävissä olevaa aikaa oli rajallisesti. Sovellus voidaan tulevaisuudessa helposti laajentaa tukemaan useita käsittelysäikeitä tai algoritminkehittäjät voivat toteuttaa prosessointilohkojen sisälle useita säikeitä – jos toteu-

tettava algoritmi tämän mahdollistaa.

DataDispatchThread lukee edellä kuvatusta FIFO-puskurista käsittelyikkunan ja siirtää sen sisältämien kanavien mittausinformaation käsittelyprotokollan sisääntuloihin. Tämän jälkeen se käy läpi jokaisen käsittelyvaiheen kutsuen vaiheen sisältämiä prosessointilohkoja (kuva 6.12). Ensimmäisessä vaiheessa olevien prosessointilohkojen sisääntulot on kytketty käsittelyprotokollan sisääntuloihin, joten kyseiset lohkot saavat syötteekseen juuri FIFO-puskurista luetun ikkunan. Toisessa vaiheessa olevat lohkot lukevat syötteensä lisäksi ensimmäisen vaiheen lohkojen ulostuloista ja niin edelleen.



Kuva 6.11. *Käsittelyyn liittyvät toimijat.*



Kuva 6.12. *DataDispatchThreadin toiminta.*

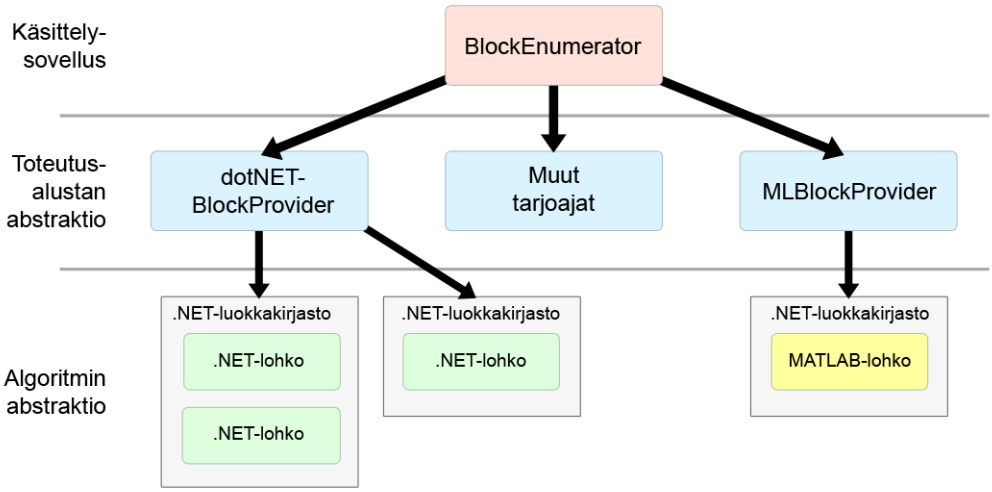
Säie toimii edellä kuvatulla tavalla silmukassa, kunnes käsittelyistunto keskeytetään (ProcessingSession::Terminate-operaatio, kuva 6.11) tai signaloidaan päättyväksi ja FIFO-puskurista loppuu tieto (DataDispatchThread::FinishProcessing-operaatio, kuva 6.11). Käsittelyistunnon päättyväksi signaloinnin aiheuttaa luvussa 6.3.1 kuvattu NeurOnessa päättyvä mittausistunto.

6.5.1 Käsittelyalgoritmien abstraktio

Eräs työlle asetetuista tavoitteista oli helppo laajennettavuus. Jos käsittelyalgoritmit rakennettaisiin kiinteästi sovellukseen, tarvitsisi sovellus kääntää lähdekoodista ajettavaan muotoon aina kun algoritmeja muokataan, poistetaan tai lisätään. Tämä edellyttäisi, että muokkaajalla on käytössään maksullinen Microsoft .NET -sovelluskehitysalusta. Lisäksi muokkaaminen vaatisi kokemusta .NET-sovelluskehityksestä.

Työssä päädyttiin erottamaan lohkojen toteutus käsittelysovelluksesta kahdella eri tasolla; ensinnäkin haluttiin erottaa lohkon toteutuksessa käytetty ohjelmistoalusta kuten .NET tai MATLAB (toteutusalustan abstrahointi). Toisaalta haluttiin erottaa lohkojen toteutus itse sovelluksesta (algoritmin abstrahointi), jotta lohkoja voitaisiin helposti lisätä ja poistaa käytöstä.

Abstraktioratkaisussa päädyttiin kolmikerroksiseen malliin (kuva 6.13). Ylimmällä tasolla toimii luokka BlockEnumerator, joka sovelluksen käynnistyessä kartoittaa käytettävissä olevat prosessointilohkot. Aluksi se lataa toteutusalustakohtaiset lohkotarjoajat (BlockProvider), ja pyytää näitä etsimään lohkoja. Kukin lohkotarjoaja kartoittaa toteutusalustaansa (kuten .NET tai MATLAB) sopivat lohkot, pyytää niiltä lohkokuvaajia (BlockDescriptor, kuva 6.14) ja palauttaa lopulta listan löytämistään kuvaajista BlockEnumeratorille. Kun mittausistunto käynnistyy, aiheuttaa tämä vastaavan käsittelyistunnon käynnistymisen. Sen yhteydessä käsittelysovellus etsii oikean protokollan ja valmistelee siihen liittyvät prosessointilohkot, joita se pyytää BlockEnumerator-luokan ilmentymältä.



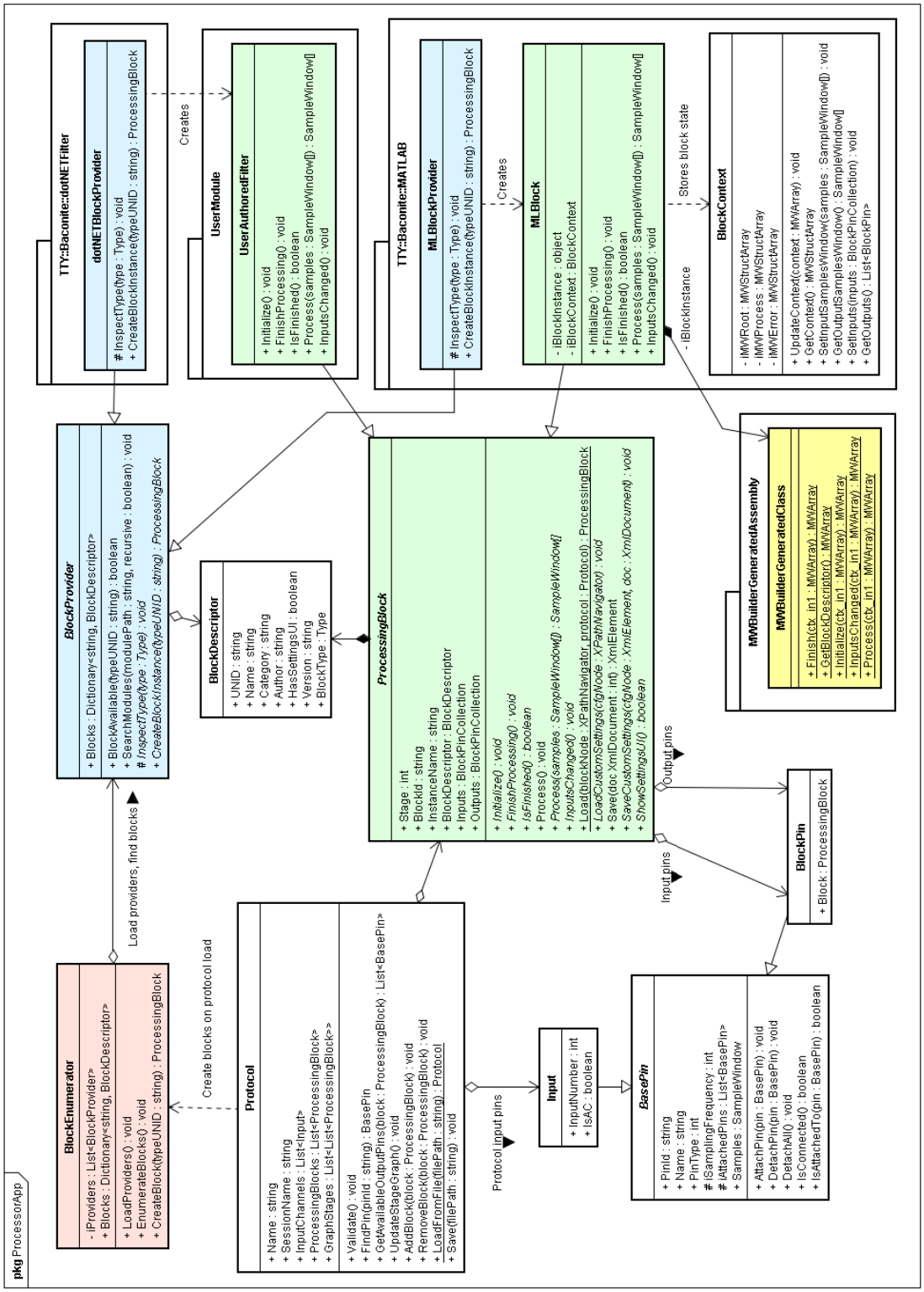
Kuva 6.13. *Lohkojen toteutuksen abstraktiokerrokset.*

Lohkokuvaaja sisältää prosessointilohkon tunnistetiedot (esimerkki liitteessä 6, L6.1). Näitä tietoja ovat mm. lohkototeutuksen yksilöllisesti tunnistava UUID, lohkon nimi, kiinteiden sisään- ja ulostulokanavien nimet sekä tieto siitä, onko lohkolla omaa asetuksiin liittyvää käyttöliittymää. Jokainen lohkotyyppi tunnistetaan 128-bittisen UUID-tunnisteen *(Universally Unique Identifier)* perusteella (BlockDescriptor-luokan

kenttä UNID). UUID voidaan esittää käyttäjälle tekstijonona, jossa on 5 ryhmää heksadesimaalisia lukuja, esimerkiksi “{F54AB597–80D8–46A8–95BE–7650DF85201E}” [113]. Jokainen prosessointilohkon toteuttaja joutuu valitsemaan lohkolleen tällaisen yksilöllisen tunnisteen. Pyrkimyksenä on ehkäistä tilanne, jossa kahdella eri prosessointilohkon toteutuksella on vahingossa sama yksilöivä tunniste. Tämä voisi johtaa tilanteeseen, jossa käsittelyistuntoa luotaessa siihen ladataan vääräntyyppinen lohko.

Jokainen prosessointilohko periytyy ProcessingBlock-luokasta. Lohkon täytyy toteuttaa muun muassa operaatiot Initialize, FinishProcessing ja Process. Initialize-operaatiota kutsutaan käsittelyistuntoa alustettaessa; sen aikana prosessointilohko varaa tarvitsemansa resurssit ja voi esimerkiksi laskea suodatinkertoimet valmiiksi käyttöä varten. FinishProcessing-operaatiota kutsutaan puolestaan käsittelyistunnon päättyessä. Tällöin prosessointilohkon on määrä vapauttaa käyttämänsä resurssit ja se voi esimerkiksi poistaa mahdollisesti käyttämänsä väliaikaistiedostot. Process-operaatio sisältää prosessointilohkon tietojenkäsittelylogiikan. Se saa parametrinään käsiteltävät näytteet ja vastaavasti palauttaa laskentansa tulokset. ProcessingBlock-luokka toteuttaa lisäksi parametrittömän Process-operaation, joka lukee näytteet prosessointilohkon sisääntuloista, kutsuu edellä mainittua parametrillistä versiota ja kutsun jälkeen tallentaa tulokset lohkon ulostuloihin.

Lisäksi prosessointilohkon täytyy tarjota toiminnallisuus lohkokohtaisten asetusten lataamiseen ja tallentamiseen (LoadCustomSettings- ja SaveCustomSettings-operaatiot) sekä logiikka lohkon ulostulojen luomiseksi sisääntulojen perusteella (InputsChanged-operaatio). Halutessaan ohjelmoija voi myös toteuttaa prosessointilohkon asetusten muokkaukseen soveltuvan käyttöliittymän (ShowSettingsUI-operaatio).



Kuva 6.14. *Prosessointilohkojen luokkakaavio.*

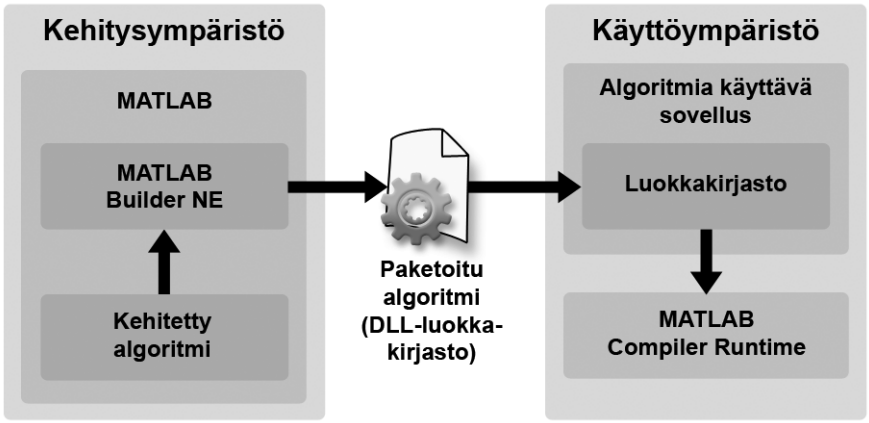
Työssä toteutettiin tuki .NET-alustalla toteutetuille lohkoille (dotNETBlockProvider), sekä MATLAB:illa tehdyille lohkoille (MLBlockProvider). DotNetBlockProvider saattaa tukea rajallisesti myös MONO-alustalla tuotettuja algoritmeja, mutta tätä ei ole testattu.

6.5.2 MATLAB-käsittelyalgoritmit

MATLAB-laskentaohjelmisto tarjoaa toiminnallisuuden enkapsuloida luotu MATLAB-koodi itsenäisesti ajettavaksi sovellukseksi tai toisessa sovelluksessa käytettäväksi komponentiksi [114]. Tämän johdosta käyttäjien on mahdollista tuoda käsittelysovellukseen algoritmeja suoraan MATLABista.

Tuotettaessa .NET-ympäristöön soveltuvaa komponenttia, käytetään MATLAB-työkalua nimeltä *MATLAB Builder NE*. Se tuottaa .NET-yhteensopivan luokkakirjaston, joka sisältää käyttäjän tuottamat algoritmit. Luokkakirjasto siirretään sitä käyttävän ulkoisen sovelluksen mukana kohdeympäristöön, johon täytyy olla asennettuna luokkakirjastoa tukeva MATLAB Compiler Runtime -sovelluskehys (MCR) (kuva 6.15). Eräs ongelmakohta kyseisessä toiminnallisuudessa on sidonnaisuus MATLAB-ohjelmiston eri versioihin. Jos luokkakirjasto tuotetaan esimerkiksi MATLAB-versiolla r2010b, vaatii se toimiakseen MCR-version 7.14 (taulukko 6.1). Jos halutaan käyttää eri MATLAB-versioilla tuotettuja luokkakirjastoja, täytyy kohdeympäristöön olla asennettuna kutakin versiota vastaava MCR. Tämä hankaloittaa MATLAB-pohjaisten komponenttien levitystä ja käyttöönottoa. Jokaisessa MCR:ssa on ajallinen käynnistyskustannus, joka on lähes yhtä suuri (10–30 sekuntia) kuin MATLAB-ohjelmistolla. Käytännössä tämä tarkoittaa sitä, että käsittelysovelluksen muistinkäyttö ja käynnistysaika kasvavat huomattavasti jokaisen käyttöön otetun MCR:n myötä.

Toinen MATLAB-algoritmien rajoite on, ettei MCR kykene rinnakkaissuoritukseen [112]. Jos MATLAB-algoritmeja halutaan suorittaa rinnakkain, täytyy olla käytössä useita MCR:eja, ja kunkin algoritmin täytyy olla juuri nimenomaiselle MCR:lle käännetty.



Kuva 6.15. *MATLAB-algoritmi käyttöympäristössä.*

Taulukko 6.1. *MATLABin sidonnaisuus eri MCR-versioihin [115].*

MATLAB-versio	**MCR-versio**
R2007a (7.4)	7.6
R2007b (7.5)	7.7

R2008a (7.6)	7.8
R2008b (7.7)	7.9
R2009a (7.8)	7.10
R2009b (7.9)	7.11
R2009bSP1 (7.9.1)	7.12
R2010a (7.10)	7.13
R2010b(7.11)	7.14
R2010bSP1 (7.11.1)	7.14.1
R2011a(7.12)	7.15
R2011b(7.13)	7.16

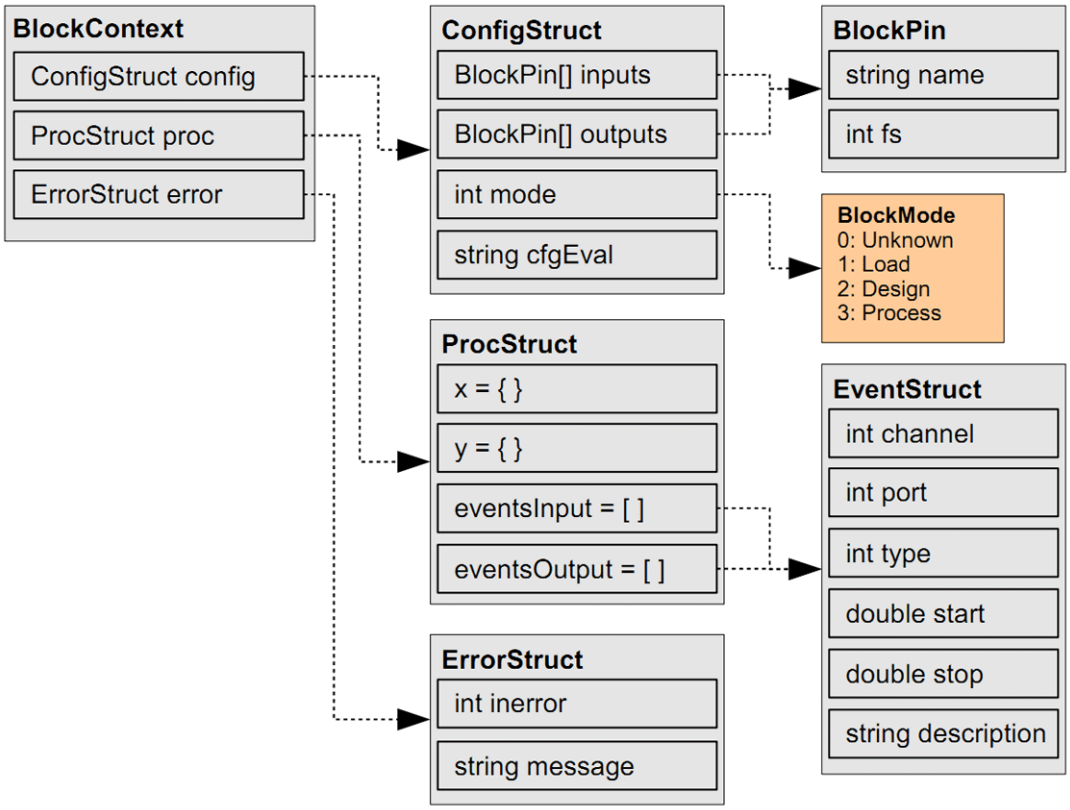
MATLAB Builder NE tuottaa vientiin *(export)* valituista funktioista .NET-luokan, jossa on kutakin funktiota vastaava staattinen metodi. Staattisesta kutsumistavasta seuraa, ettei funktioilla ole käytettävissä kyseisen luokan tilatietoa[35]. Tämä tarkoittaa, ettei MATLAB-algoritmeilla ole käytössään sisäistä muistia, johon algoritmin sisäinen tila voitaisiin varastoida kutsukertojen välillä, vaan kaikki niiden tarvitsema tieto täytyy välittää funktioiden kutsupyynnöissä. Toisin sanoen algoritmi “unohtaa” kaiken kutsukertojen välillä. Ratkaisuna ongelmaan tila voidaan säilöä algoritmin ulkopuolelle; algoritmille annetaan jokaisella kutsukerralla oma kontekstirakenne, johon se voi tallentaa tarvitsemiaan tietoja. Algoritmi tekee siihen muutoksia, ja palauttaa sen jälkeen muokatun kontekstin, joka annetaan algoritmille sen seuraavalla kutsukerralla.

Tässä työssä toteutettu käsittelysovellus käyttää kaikkia MATLAB-algoritmeja MLBlock-luokan ilmentymien kautta (kuva 6.14). Käsittelysovellus kutsuu kyseisen luokan toteuttamia prosessointilohkon rajapinnan (ProcessingBlock, kuva 6.14) operaatioita. MLBlock-luokka sovittaa nämä operaatiot (mm. Initialize, Process ja FinishProcessing) MATLAB Builder NE:n tuottaman luokan (MWBuilderGeneratedClass, kuva 6.14; tekstissä tästä lähtien termillä MATLAB-luokka) funktioihin, jotka puolestaan kutsuvat MATLAB-funktioita (näistä esimerkki liitteessä 6). MLBlock-luokkaa voidaan ajatella kerroksena kuvassa 6.15 esitetyn sovelluksen ja luokkakirjaston välissä (käyttöympäristö). Kyseinen luokka on myös vastuussa prosessointilohkon tilan säilömisestä algoritmin ulkopuolella sijaitsevaan kontekstiin (kuva 6.16). MATLAB-luokan staattiset metodit hyväksyvät syötteenään MCR:n tarjoaman MWArray-tietotyypin, joka vastaa liitteessä 6 esitettyjen funktioiden ctx-parametriä. Luokka BlockContext (kuva 6.14) tarjoaa tämän muuttujan käsittelyyn toiminnallisuuden, josta luokka MLBlock on riippuvainen.

Tarkastellaan lähemmin MATLAB-lohkon ulkoista kontekstia (kuva 6.16): se on kolmesta kentästä (config, proc ja error) koostuva MATLAB-struktuuri (MWStructArray). Config-kenttä on tietorakenne, joka sisältää lohkon asetukset (esim. inputs- ja out-

35. Olio-ohjelmointiparadigman mukaisesti puhutaan luokan kentistä tai attribuuteista. Nämä ovat olion ilmentymäkohtaisia sisäisiä muuttujia, jotka säilyttävät arvonsa luokan metodien kutsujen välissä.

puts-kentät sisältävät lohkon sisään- ja ulostulot). Proc-kenttä on tarkoitettu lohkon prosessoinnin aikaiseksi tietovarastoksi; MLBlock-luokka kopioi prosessointilohkon sisääntuloihin saapuneet näytteet x-kenttään ja vastaavasti lohkon suorittaman prosessoinnin tuottamat näytteet y-kentästä prosessointilohkon ulostuloihin. Error-kenttään lohko voi tallentaa tietoa mahdollisesti aiheutuneista virheistä sekä ilmoittaa käsittelysovellukselle ettei kykene jatkamaan tiedon käsittelyä.

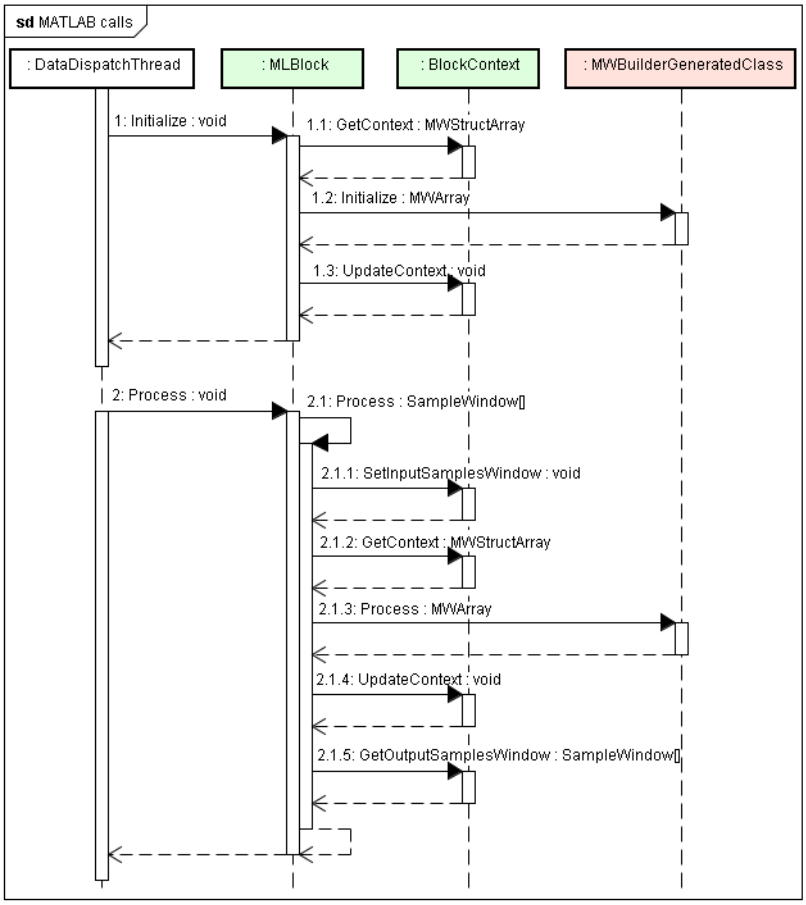


Kuva 6.16. *MATLAB-lohkon tilan säilövä konteksti.*

Tämä tietorakenne muodostaa rajapinnan MATLAB-funktioiden ja käsittelysovelluksen välille – kaikki käsiteltävä tieto kulkee sen läpi. Sen tulee sisältää vähintään kuvassa 6.16 esitetyt kentät. Lisäksi MATLAB-algoritmin toteuttaja voi vapaasti luoda sen sisälle muita algoritmin tarvitsemia kenttiä – esimerkiksi prosessointilohkon alustuksen aikana lasketut suodinkertoimet.

MLBlock- ja BlockContext-ilmentymien sekä MATLAB-luokan ajonaikainen yhteistoiminta on esitetty kuvassa 6.17. Käsittelyistunnon alkaessa DataDispatchThread luo prosessointilohkojen ilmentymät. MLBlock-luokan ilmentymän myötä luodaan myös kyseisen lohkon konteksti (BlockContext). Seuraavaksi DataDispatchThread alustaa prosessointilohkot kutsumalla jokaisen lohkon Initialize-operaatiota (kohta 1). MATLAB-prosessointilohkoilla MLBlock-ilmentymä pyytää aluksi BlockContext-ilmentymältä MATLAB-luokan tarvitseman MWArray-parametrin (GetContext-operaatio, kohta 1.1), ja antaa tämän sitten MATLAB-luokalle kutsuen sen Initialize-operaatiota (kohta 1.2). MATLAB-luokka kutsuu MCR:n kautta MATLAB-funktiota (Initialize, liite 6,

L6.1), joka alustaa prosessointilohkon MATLAB-toiminnallisuuden osalta, tallentaa nämä muutokset kontekstiin ja palauttaa sen kutsujalle. Lopuksi MLBlock-ilmentymä päivittää kontekstin takaisin BlockContext-ilmentymälle (kohta 1.3).



Kuva 6.17. *MATLAB-luokan kutsuminen ja kontekstin käyttö.*

Kuvasta nähdään myös signaalien käsittelyyn liittyvä toiminta (kohta 2, Process); DataDispatchThreadin kutsumassa MLBlock-lohkon Process-operaatiossa haetaan aluksi käsiteltävät näyteikkunat lohkon sisääntuloista, ja kutsutaan Process-operaation ylikuormitettua versiota, antaen sille parametrinä käsiteltävät näytteet (kohta 2.1). Tämä operaatio tallentaa näytteet lohkokontekstiin (kenttään proc.x) kutsumalla kontekstia ylläpitävän BlockContext-luokan operaatiota SetInputSamplesWindow. Seuraavaksi Process-operaatio pyytää kontekstin MCR:n ymmärtämänä MWStructArray:nä ja kutsuu MATLAB-luokan staattista Process-operaatiota, antaen sille parametrinä uudet näytteet sisältävän kontekstin (kohta 2.1.3). Tämä palauttaa muuttamansa kontekstin MWArray-tyyppisenä paluuarvona, joka tallennetaan takaisin BlockContext-luokan ilmentymään (operaatio UpdateContext, kohta 2.1.4). Lopuksi Process-operaatio pyytää suotimen palauttamat näytteet BlockContext-luokan ilmentymältä kutsumalla sen GetOutputSamplesWindow-operaatiota (kohta 2.1.5) ja palauttaa nämä ensimmäiseksi kutsutulle Process-operaatiolle. Tämä puolestaan tallentaa tulokset kyseisen MLBlock-prosessoin-

tilohkon ulostuloihin, joista seuraavien vaiheiden prosessointilohkot voivat lukea ne.

6.5.3 Käsittelyalgoritmien toteutus

Työssä toteutettiin itse käsittelysovelluksen lisäksi myös useita käsittelyalgoritmeja. Näiden tuottamista ei vaadittu työn alussa, mutta tehdyistä arkkitehtonisista ratkaisuista (luku 6.2) johtuen mm. mittaustiedon lähetys palvelimelle edellytti prosessointilohkon toteuttamista. Lisäksi joitakin lohkoja tarvittiin sovelluksen testauksessa ja joitakin lohkoja tehtiin puhtaasti oman motivaation pohjalta.

Lohkoista tiedonsiirtoon ja tallennukseen liittyviä olivat Girf Oü:n palvelintoteutukselle tietoa lähettävä ‘GPB over TCP’-lohko (kuvattu luvussa 6.6), raakaa EDF-tietoa TCP/IP-yhteyden yli lähettävä ‘EDF over TCP’, sekä signaaleja EDF-tiedostoon tallentava ‘EDF File’.

Varsinaiseen signaalinkäsittelyyn toteutettiin yksinkertainen signaaleja viivästävä ‘Delay’, FIR-suodatin ‘FIR Filter’, signaaleja alasnäytteistävä ‘Decimator’, sekä EP-kokeisiin soveltuva määrättyjen tapahtumien yhteydessä signaalista ikkunoita leikkaava ‘Evoked Potential Windower’.

Mittaustiedon esitystä varten tehtiin MATLAB-lohkoina yksinkertaiset signaaleja aikatasossa esittävä ‘SigDisp’ sekä taajuusalueella esittävä ‘SigFFT’. Lisäksi toteutettiin reaaliajassa spektrogrammia piirtävä ‘Spectrogram’ ja leikattuja ikkunoita keskiarvoistava ‘EPAverager’.

6.6 Mittaustiedon lähetys

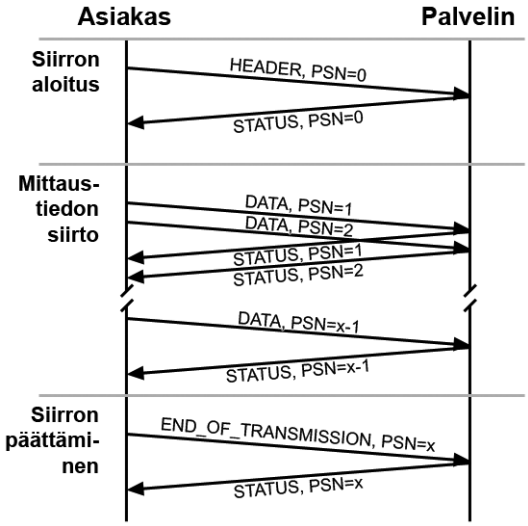
Luvun 5 alussa esitettiin vaatimus, jonka mukaan sovelluksen on kyettävä siirtämään mittaustietoa tietoverkon kautta palvelimelle. Tämä toiminnallisuus oli luontevaa toteuttaa prosessointilohkona luvussa 6.2 määritetyn käsittelysovelluksen arkkitehtuurin mukaisesti. Tässä luvussa kuvataan ensin tiedonsiirto asiakkaan (tämän työn yhteydessä toteutettu käsittelysovelluksesssa toimiva prosessointilohko) ja palvelimen (Girf Oü:n palvelinratkaisu) välillä sekä lopuksi toteutetun prosessointilohkon sisäinen arkkitehtuuri.

Sovellustason protokolla määritettiin yhteistyössä Girf Oü:n kanssa. Se on kaksikerroksinen: ylemmällä kerroksella mittaustieto esitetään EDF+-formaatin mukaisesti [76] ja alemmalla kerroksella se paketoidaan viesteiksi *Google Protocol Buffers* -tekniikkaa (GPB) [116] käyttäen. Alempi kerros puolestaan on yhteydessä luvussa 5.4.1 käsiteltyyn kuljetusprotokollaan, joka on tässä toteutuksessa TCP. GPB:lla on kolme tarkoitusta; ensinnäkin se eristää EDF-formaatin kuljetuskerroksesta, jolloin lähetettävän tiedon formaattia voidaan tulevaisuudessa helposti muuttaa. Toiseksi se määrittää paluuliikenteen formaatin (palvelimen lähettämät kuittaukset) ja kolmanneksi se muuntaa TCP-kuljetusprotokollan tietovirtaorientoituneen liikenteen viestiorientoituneeksi, eikä ylemmän sovelluslogiikan tarvitse täten huolehtia kuvassa 5.13 esitetystä skenaariosta. Viestien rakenne on kuvattu liitteessä 7.

EDF+ on tiedostoformaatti, mutta rakenteensa (liite 7, kuva L7.2) ansiosta sitä voidaan käyttää rajallisesti myös tiedon reaaliaikaiseen välitykseen. Tällä päästään kevyeen

palvelintoteutukseen: yksinkertaisimmillaan palvelimen tarvitsee vain vastaanottaa viestejä ja tallentaa näiden sisältämää EDF-tietoa palvelimella sijaitsevaan tiedostoon. EDF-formaatin hyviä ja huonoja puolia käsiteltiin luvussa 5.4.2. Hyvänä puolena EDF on laajalti tuettu, huonona puolena se rajoittaa näytteiden tarkkuuden 16 bittiin.

Asiakkaan ja palvelimen välinen sovellustason kommunikaatio on esitetty kuvassa 6.18. Jokainen sovellusten välinen viesti pohjautuu SdbMessage-rakenteeseen (kuva L7.1). Todellisuudessa sovellusten välillä kulkee enemmän paketteja, sillä TCP-taso voi pilkkoa esimerkiksi data-paketin useaksi TCP-segmentiksi, joihin palvelimen TCP-protokollapino vastaa TCP ACK-viesteillä.



Kuva 6.18. *Sovellustason protokollan kommunikointisekvenssi.*

Aluksi asiakas avaa TCP/IP-yhteyden palvelimeen ja lähettää tiedonsiirron otsikkotiedot (HEADER-viesti). Tämä aloittaa asiakkaan ja palvelimen välisen tiedonsiirtoistunnon. Otsikkotietojen perusteella palvelin valmistautuu vastaanottamaan mittaustietoa (luo uuden EDF-tiedoston sekä tallentaa otsikon sen alkuun) ja kuittaa asiakkaalle saaneensa otsikkotiedot. Tämän jälkeen asiakas alkaa lähettää mittaustietoa sisältäviä viestejä (DATA). Palvelin lähettää asiakkaalle kuittauksen (STATUS) jokaisesta saamastaan viestistä, kopioiden kuittaukseen kyseisen viestin sekvenssinumeron (kuva 6.18, PSN; kuva L7.1, kenttä packetSeqNo). Lähetettyään kaiken mittaustiedon asiakasohjelma päättää tiedonsiirtoistunnon lähettämällä palvelimelle END_OF_TRANSMISSION-viestin.

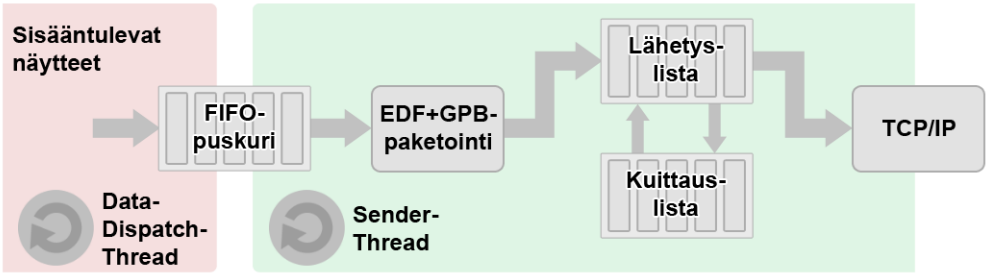
Palvelin voi käsitellä samanaikaisesti useita tiedonsiirtoistuntoja; nämä erotetaan toisistaan istunnon yksilöivän UUID-tunnisteen perusteella. Jokaisella samaan istuntoon kuuluvalla viestillä on sama UUID. Istunnon sisällä viesti yksilöidään juoksevan sekvenssinumeron (packetSeqNo, kuvassa PSN) perusteella. Sekvenssinumero alkaa istunnon sisällä nollasta.

Toteutetun prosessointilohkon arkkitehtuuri on esitetty kuvassa 6.19. Luvussa 6.5 kuvattu DataDispatchThread välittää mittaustiedon prosessointilohkolle kutsumalla loh-

kon Process-operaatiota, jossa tieto tallennetaan lohkon sisäiseen FIFO-puskuriin. Näin DataDispatchThread voi jatkaa muiden lohkojen kutsumista. Lohkon sisällä toimii erillinen tiedon paketoinnista ja lähetyksestä huolehtiva säie SenderThread.

Säikeen toiminta on seuraavanlaista: aluksi se tarkastaa, onko lähetyslistalla valmiita lähetyspaketteja. Jos on, se lähettää paketit ja siirtää nämä kuittauslistalle odottamaan palvelimelta saatua kuittausta. Jos lähetyslista on tyhjä, noutaa säie ikkunallisen mittaustietoa FIFO-puskurista, paketoi sen EDF-formaattiin ja tämän edelleen GPB-viestiksi, joka serialisoidaan jonoksi tavuja (lähetyspaketti) ja tallennetaan lähetyslistaan. Lähetyslista on järjestetty sekvenssinumeron perusteella: pienimmän sekvenssinumeron omaavilla paketeilla on korkeampi prioriteetti. Täten asiakas pyrkii takaamaan mittaustiedon lähetyksen ajallisesti jatkuvassa järjestyksessä.

Saatuaan palvelimelta kuittausviestin säie poistaa kuittauslistalta sekvenssinumeroa vastaavan lähetyspaketin. Jos kuittauslistalla odottava paketti ei saa tietyn ajan kuluessa kuittausta, se siirretään takaisin lähetyslistalle.



Kuva 6.19. *'GPB over TCP'-prosessointilohkon arkkitehtuuri.*

Joskus tiedonsiirtoyhteys saattaa katketa verkon olosuhteiden johdosta. Tämä voi johtua esimerkiksi reititysongelmasta tai fyysisestä siirtoyhteyden tai toimilaitteen pettämisestä. Tällöin SenderThread yrittää avata uuden yhteyden kunnes yhteys muodostuu tai lähetys keskeytetään (esim. käyttäjän terminoidessa käsittelyistunnon). Kun yhteys saadaan muodostettua kaikki kuittausta odottavat paketit siirretään lähetyslistalle. Tällöin voi ilmetä tilanne, jossa sama paketti toimitetaan palvelimelle kaksi kertaa. Palvelin on vastuussa ylimääräisten pakettien hylkäämisestä. Sen tosin täytyy kuitata myös tällaisen paketin onnistunut vastaanotto, sillä muutoin asiakas lähettää sen uudelleen.

Kuvatussa lähetysprotokollassa ei ole tietoturva-aspekteja. Tiedonsiirron oletetaan tapahtuvan turvallisessa verkossa. Tarvittaessa yhteys voidaan muodostaa VPN-tunnelin läpi tai käyttää TLS-protokollaa TCP- ja GPB-kerrosten välissä (luku 5.4.3).

Tiedonsiirtoa voidaan optimoida vielä siten, että palvelin lähettää useita kuittauksia samassa viestissä. Nykyisessä toteutuksessa jokaisesta palvelimen vastaanottamasta viestistä lähtee erillinen kuittausviesti.

7 KÄSITTELYSOVELLUKSEN TESTAUS

Teoksessaan Järvinen & Järvinen kehottavat konstruktiivisen tutkimuksen tuotosten huolelliseen evaluointiin. Arviointikriteereiksi he esittävät muun muassa tehokkuutta, hyödyllisyyttä, kustannuksia (esim. huolto- ja korjauskustannukset), sivuvaikutuksia, vaikuttavuutta, sekä vaikutuksia ympäristöön. [4, s.123]

Tässä työssä toteutettua sovellusta ei kyetä työn puitteissa arvioimaan kovin pitkällä aikavälillä, sillä projektina diplomityö ei saisi venyä useiden vuosien mittaiseksi. Niinpä sovelluksen arviointiin suhtauduttiin ohjelmistoprojektin näkökulmasta. Projektin alussa määritettyjä vaatimuksia kohtaan tehtiin hyväksymistestaus (luku 7.1). Lisäksi työssä toteutetun sovelluksen tehokkuutta arvioitiin erikseen nopeustestauksella (7.2).

7.1 Hyväksymistestaus

Ohjelmistoprojektissa hyväksymistestaus pohjautuu ennalta määrättyihin asiakasvaatimuksiin. Nämä ovat korkean tason kriteereitä, jotka määrittävät tilaajan näkökulmasta millainen ohjelmistotuotteen täytyy olla. Yleensä ne esitetään asiakasvaatimusmäärittelmässä, jollainen laadittiin myös tämän projektin alkuvaiheessa [117].

Testauksen tueksi luotiin erillinen testaussuunnitelma [118], jonka mukaisesti suoritettujen testitapausten tulokset kirjattiin erillisiin laskentataulukoihin. Jokaisen testitapauksen arvioitiin joko onnistuneen tai epäonnistuneen, ja näiden perusteella pääteltiin, täyttääkö ohjelma sille asetetut tavoitteet. Yhteenveto tuloksista on esitetty taulukossa 7.1.

Testaus jaettiin luvussa 5.2 tehdyn osittelun mukaisesti tiedon vastaanottoon, käsittelyyn ja lähetykseen. Vastaanotto-osiossa (testitapaukset 1.1–1.3) testattiin tiedon saantia NeurOne-sovellukselta sekä tilannetta, jossa käsittelysovellus kohtaa virheen josta se ei kykene toipumaan (luku 6.3). Käsittelyosiossa (testitapaukset 2.1–3.2) puolestaan tutkittiin onko .NET- ja MATLAB-algoritmien lisääminen ja käyttöönotto käsittelysovelluksessa vaivatonta (luku 6.5.1) sekä näiden välisten kytkentöjen toimintaa. Lähetysvaiheessa (testitapaus 4.1) testattiin käsittelysovelluksen ja Girf Oü:n toteuttaman palvelinsovelluksen välistä yhteistoimintaa.

Lisäksi suoritettiin kokonaisvaltainen testitapaus (5.1) jossa mittauslaitteistoon oli kytketty oikea henkilö. Tässä mitattuja signaaleja suodatettiin, esitettiin näytöllä ja tallennettiin paikalliselle kiintolevylle.

Taulukko 7.1. *Hyväksymistestauksen tulokset [119].*

Testi-tapaus	Kuvaus	Tulos
1.1	16 x 500 Hz kanavan vastaanotto NeurOnelta.	OK
1.2	16 x 5 kHz kanavan vastaanotto NeurOnelta.	OK
1.3	Vikatilanteen sieto: Käsittelysovelluksen kaatuminen ei kaada NeurOnea.	OK
2.1	.NET-prosessointilohkon asennus ja käyttöönotto.	OK
2.2	MATLAB-prosessointilohkon asennus ja käyttöönotto.	OK
2.3	Mittaustiedon välitys MATLAB-prosessointilohkoille.	OK
3.1	Prosessointiverkon rakenteet: sarjakytkentä, rinnakkaisuus, haarautuminen ja yhteenliittyminen sekä erisuuret näytteenottotaajuudet.	OK
3.2	Prosessointilohkon sisään- ja ulostulojen epäsymmetria.	OK
4.1	Mittaustiedon välitys Girf Oü-palvelimelle (16 x 5 kHz). Testaus suoritettiin ilman kuittaustoimintoa, sillä palvelin ei tätä vielä tuolloin tukenut.	OK
5.1	Oikea mittaustilanne. Mitattiin neljää kanavaa: O1, O2, F3 ja F4, referenssi-elektrodina Cz. Prosessointilohkoina .NET FIR-suodin (notch), EDF-tallennus, Spektrogrammi, sekä MATLAB plot ja fft.	OK

Toteutettu sovellus läpäisi kaikki testitilanteet. Tämän perusteella sen todettiin täyttävän työn alussa asetetut vaatimukset [117].

7.2 Nopeustestaus

Pelkän hyväksymistestauksen lisäksi haluttiin arvioida toteutetun sovelluksen toiminta-nopeutta. Tästä saadusta tiedosta on mahdollisesti hyötyä suunnitellessa oikeaa mittaustapahtumaa, jolloin halutaan tietää, kykeneekö sovellus käsittelemään mittaustapahtuman mukaista tietoa riittävän nopeasti.

Myös käsittelysovelluksen nopeustestit jaettiin luvussa 5.2 tehdyn osituksen mukaisesti vastaanottoon, käsittelyyn ja lähetykseen [118]. Jokaisessa osa-alueessa suoritettiin yksi tai useampi testi, joka puolestaan jakautui useaan testitapaukseen. Testitapaukset erosivat toisistaan vain mittausalustan parametrien (näytteenottotaajuus ja mittauskanavien lukumäärä) suhteen. Jokaisessa testissä pyrittiin määrittämään maksimi näytteenottotaajuus ja kanavien määrä, joilla käsittelysovellus kykenee reaaliaikaisuuteen. Kaikki testitapaukset suoritettiin käyttäen käsittelysovelluksen release-versiota[36] ja TTY:n tarjoamaa kannettavaa tietokonetta, jonka kokoonpano on kuvattu liitteessä 4 (taulukko L4.1).

Jokainen testi aloitettiin vaativimmalla testitapauksella (suurin määrä mittauskanavia ja suurin näytteenottotaajuus). Jos mittaus ei tyydyttänyt reaaliaikaisuudelle asetettuja kriteerejä, tulkittiin ettei käsittelysovellus kykene riittävään nopeuteen ja suoritettiin seuraava testitapaus, jossa oli pienemmät vaatimukset. Testien tulokset on esitetty taulu-

36. Release-versiosta puuttuu kaikki sovelluksen kehitykseen liittyvä toiminnallisuus, kuten sovelluksen mahdollisesti antamat debug-viestit. Lisäksi kääntäjä tekee enemmän optimointeja release-versioon tuottamaansa ohjelmakoodiin. Täten sovellus toimii nopeammin, mutta sen toimintaa ja virhetilanteita voi olla vaikeampi tutkia ja tulkita.

kossa 7.2.

Vastaanotto-osio sisälsi vain yhden testin (testitapaus 6.1), jossa haettiin maksimi näytteenottotaajuus ja kanavien määrä, jolla käsittelysovellus kykenee vastaanottamaan tietoa. Testissä tarkkailtiin NeurOne-liitännäisen ilmoittamaa tilatietoa, joka ilmaisee toimiiko liitännäisen ja käsittelysovelluksen välinen muistisilta (luku 6.3.2) riittävän nopeasti, vai karttuuko siirrettävää tietoa liitännäisen muistissa olevaan puskuriin.

Käsittelyosiossa testattiin algoritmin toteutusalustalle (.NET tai MATLAB) ominaisen kutsumiskustannuksen (luku 5.2.1) suuruutta (testitapaukset 6.2–6.3). Laskentakustannuksen osuutta ei tutkittu, koska se riippuu algoritmista, joka taasen riippuu täysin käyttötilanteesta ja käyttäjästä. Osio jaettiin kahteen testiin; näistä ensimmäisessä testattiin käsittelykustannusta .NET-prosessointilohkolla ja toisessa MATLAB-prosessointilohkolla. Kumpikin lohko toteutti yksinkertaisen kertolaskuun pohjautuvan skaalausoperaation. Käytettäväksi operaatioksi valittiin skaalaus, koska se on operaationa yksinkertainen ja käsittelee jokaista näytettä. Esimerkiksi konvoluution käyttäminen ei olisi ollut tarkoituksenmukaista, sillä valtaosa laskenta-algoritmeista ei perustu konvoluutioon.

Lähetysosiossa haettiin testitapausten avulla maksiminopeus, jolla käsittelysovellus kykenee välittämään tietoa Girf Oü:n toteuttamalle palvelinkomponentille (testitapaus 6.4).

Taulukko 7.2. *Nopeustestauksen tulokset [120].*

Testi-tapaus	**Kuvaus**	**Tulos**
6.1	Tiedon vastaanoton maksiminopeus	40 x 80 kHz *
6.2	.NET-prosessointilohkon maksimikäsittelynopeus	40 x 80 kHz *
6.3	MATLAB-prosessointilohkon maksimikäsittelynopeus	40 x 80 kHz *
6.4	Tiedon lähetyksen maksiminopeus (Girf Oü-palvelimelle)	20 x 80 kHz

* 40 x 80 kHz oli suurin kaistanleveys mitä käytettävissä olevalla NeurOne-järjestelmällä pystyttiin tuottamaan (käytössä oli vain yksi 40-kanavainen esivahvistin).

Toteutettu sovellus kykeni käsittelemään tietoa NeurOnen maksimikaistalla lähes kaikissa tapauksissa. Poikkeuksena tähän oli tiedon lähetys (testitapaus 6.4); sovellus kykeni välittämään reaaliaikaisesti vain puolet NeurOne-järjestelmän tarjoamasta maksimista. Mittaustietoa alkoi kerääntyä käsittelyistunnon vastaanottopuskuriin (kuva 6.2) sekä tietoa lähettävän prosessointilohkon FIFO-puskuriin (kuva 6.19). Kumpikin näistä puskureista on luvussa 6.4 kuvattu FIFO-puskuri.

Syyksi tähän paljastui FIFO-puskurin toteutuksessa käytetty synkronointimekanismi: säieturvallisuuden vuoksi koko puskuri lukitaan luku- ja kirjoitusoperaatioiden ajaksi. Ilman lukitusta voi ilmetä tilanne, jossa kaksi säiettä käyttää puskuria yhtäaikaa korruptoiden puskurin tietorakenteita. Jos puskuri käyttää lohkotiedostoa, täytyy tietoa puskuriin tallentavan säikeen kirjoittaa kiintolevylle ja vastaavasti tietoa lukevan säikeen ladata kiintolevyltä. Kiintolevyltä luku ja sille kirjoitus ovat suoritusajallisesti hyvin kalliita operaatioita – jonka ajan koko puskuri on lukittuna ja toinen säie joutuu odotta-

maan.

Tätä pahentaa vielä se, että DataDispatchThreadin odottaessa vuoroaan kirjoittaa prosessointilohkon FIFO-puskuriin, ehtii DataPollerThread tuoda käsittelyistuntoon lisää tietoa. Tämän johdosta vastaanottopuskuri täyttyy ja alkaa myös käyttää lohkotiedostoa. Täten käsittelysovellus joutuu yhtä aikaa lukemaan ja kirjoittamaan kahta lohkotiedostoa, jotka sijaitsevat samalla kiintolevyllä – entisestään suorituskykyä huonontaen.

Ongelmaa voitaisiin lieventää jakamalla FIFO-puskurin lukitus pienempiin osiin, esimerkiksi toteuttamalla lohkotiedostolle oman lukitsemisprimitiivin. Tällöin puskurista lukeminen lukitsisi vain rengaspuskurin (ja lohkotiedoston silloin kun rengaspuskuri on tyhjä). Tämä vähentäisi säikeiden odottamista.

8 JOHTOPÄÄTÖKSET

Työn tuloksena syntyi helposti laajennettavissa oleva sovellus, joka ei ole sidottu NeurOne-laitteistoon – tai edes EEG-signaaleihin. Sovellusta voidaan käyttää myös muiden yksiulotteisten signaalien (esim. EMG tai EKG) käsittelyyn. Lisäksi se on mahdollista sovittaa muidenkin valmistajien mittausjärjestelmiin, jolloin samoilla käsittelyalgoritmeilla voidaan laajentaa usean eri mittausjärjestelmän ohjelmistollista toteutusta.

Hyväksymis- ja nopeustestauksen tulosten perusteella toteutettu järjestelmä täyttää asetetut tavoitteet; käyttäjä voi helposti lisätä järjestelmään itse tekemiään käsittelyalgoritmeja sekä pystyy määrittelemään käsittelyprotokollan, jonka perusteella järjestelmä käsittelee ja lähettää tietoa mittauksen aikana. Tämän lisäksi järjestelmä kykenee edellä mainittuun protokollaan perustuen käsittelemään ja välittämään mittaustietoa reaaliaikaisesti samassa lähiverkossa sijaitsevalle palvelimelle.

FIFO-puskurin toteutuksessa olevan rajoitteen vuoksi mittaustiedon reaaliaikainen lähetys on mahdollista vain puolella NeurOne-järjestelmän maksimikapasiteetista. Tämä on kuitenkin kaikilta puolin riittävä kapasiteetti TTY:n Porin yksikön tekemien tutkimusten kannalta, eikä täyden kapasiteetin tukeminen kuulunut työn tavoitteisiin.

Myös johdannossa kuvattu toissijainen tavoite tuottaa Mega Elektroniikka Oy:n käyttöön valmiita ohjelmistokomponentteja vaikuttaa toteutuvan: työn valmistuessa kuultiin, että NeurOne-järjestelmän kehittänyt Mega Elektroniikka Oy aikoo integroida tässä työssä toteutetun sovelluksen (tai ainakin sen osia) osaksi NeurOne-järjestelmää.

Laajemmassa merkityksessä tarkastellen tuloksilla on huomattavia vaikutuksia: toteutettu käsittelysovellus eristää mittausjärjestelmän algoritmin toteutusalustasta ja itse algoritmista. Tämä mahdollistaa algoritmien jakamisen ja käytön yli organisaatio- ja mittausjärjestelmärajojen. Toteutuksessa voidaan käyttää .NET- tai MATLAB-tekniikoita, mutta käsittelysovelluksen voi helposti laajentaa tukemaan muitakin toteutusalustoja (esim. C++-lähdekoodista käännetty x86- tai x64-natiivikoodi).

Toteutettua sovellusta ei ole lääketieteellisesti hyväksytty, joten se soveltuu ainoastaan tutkimuskäyttöön. Sitä ei voida käyttää potilaan hoitoon vaikuttavassa päätöksenteossa. Aivan kaikkiin tutkimuksiin sovellus ei välttämättä sovi: käytetyn puskuroinnin johdosta reaaliaikaisuus ei välttämättä riitä esimerkiksi biopalautteen sovelluksiin, joissa viiveen pitäisi olla alle sekunnin.

Myös käsittelyprotokollien keskitetty hallinta on mahdollista: käsittelysovellus voidaan konfiguroida lataamaan käsittelyprotokollat ja algoritmit verkkolevyltä. Täten esimerkiksi sairaalafyysikko voi helposti hallita useilla eri mittauspaikoilla olevien käsittelysovellusten toimintaa.

Jatkokehitys

Projektin edetessä syntyi runsaasti ideoita, joita ei ollut rajallisesta aikakehyksestä johtuen realistista lähteä toteuttamaan. Työssä toteutetun ohjelmiston sovellusaluetta olisi mahdollista laajentaa sekä optimoinnilla että lisäominaisuuksilla.

Tehokkuutta voitaisiin parantaa usealla osa-alueella; ensimmäinen näistä on testauksessa (luku 7.2) havaittu FIFO-puskurin rajoite. Toiseksi tavoitteeksi kannattaisi ottaa algoritmien toiminnan tehostaminen, esimerkiksi rinnakkaistamalla algoritmit ajettaviksi useissa säikeissä. Lisäksi joidenkin algoritmien laskenta voitaisiin mahdollisesti siirtää näytönohjaimen suorittimelle (*Graphics Processing Unit*, GPU), esimerkiksi Nvidian CUDA-arkkitehtuuria *(Compute Unified Device Architecture)* käyttäen.

Kolmanneksi olisi kannattavaa kehittää mittaustiedon tallennus- ja siirtoformaattia. Työn yhteydessä sovellukseen toteutetut tietoa lähettävät ja tallentavat prosessointilohkot käyttävät (mahdollisimman laajan yhteensopivuuden takaamiseksi) EDF+-pohjaisia formaatteja, jolloin yksittäinen näyte kuvataan vain 16 bitin tarkkuudella. Uudella formaatilla mittaustieto kyettäisiin tarvittaessa esittämään tarkemmin sekä eliminoimaan annotaatiolohkojen hukkatila (luku 5.4.2).

Neljänneksi mittaus- ja käsittelysovelluksen välinen mittaustiedon siirtokanava voitaisiin abstrahoida, jolloin mittaustietoa voitaisiin siirtää jaetun muistin lisäksi myös TCP/IP-yhteyden kautta. Tämä mahdollistaisi kaksi asiaa; a) Käsittelysovellus voisi sijaita NeurOnesta erillisellä tietokoneella, sekä b) olisi mahdollista ketjuttaa useita eri tietokoneilla sijaitsevia käsittelysovelluksia, hajauttaen näin laskentaa.

Vaihtoehtoisesti toteutettuun käsittelysovellukseen voitaisiin toteuttaa FieldTrip-pohjainen lohko, joka mahdollistaisi online-integraation FieldTrip-ratkaisujen kanssa [2]. Myös tämä mahdollistaisi useille tietokoneille hajautetun laskennan.

Lisäksi MATLAB-suodinten käyttöliittymää voitaisiin parantaa entisestään, esimerkiksi lisäämällä mahdollisuus muokata suodinten parametrejä tai koodia käsittelyn aikana. Tämä sallisi nopeamman prototyypityksen sekä algoritmien testailun mittauksen aikana.

Vaikka tässä työssä toteutettu sovellus täyttääkin työn alussa asetetut kriteerit, voi sen jatkokehitys kantaa hedelmää usealla eri rintamalla. Tämä tarjoaakin hyvät lähtökohdat seuraaville tässä työssä toteutettuun sovellusalustaan pohjaaville diplomitöille tai jatko-opinnoille.

LÄHTEET

[1] Jollyon Smith, N., van Gils, M., Prior, P. Neurophysiological Monitoring during Intensive Care and Surgery. Amsterdam 2006, Elsevier. 408 p.

[2] Oostenveld, R., Fries, P., Maris, E., Schoffelen, J-M. 2011. FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data. Computational Intelligence and Neuroscience [WWW-sivu]. Hindawi Publishing Corporation. [viitattu 5.12.2011]. Saatavissa: http://www.hindawi.com/journals/cin/2011/156869/.

[3] The OpenEEG Project. 2011. Welcome to the OpenEEG project. [WWW-sivu]. The OpenEEG project. [viitattu 5.12.2011]. Saatavissa: http://openeeg.sourceforge.net/doc/index.html.

[4] Järvinen, P. & Järvinen, A. Tutkimustyön metodeista. Tampere 2000, Opinpajan kirja. 221 p.

[5] Ahmadlou, M., Adeli, H. Wavelet-Synchronization Methodology: A New Approach for EEG-Based Diagnosis of ADHD. Clinical EEG and Neuroscience 41(2010)1, pp. 1 - 10.

[6] Malmivuo, J. Theoretical Limits of the EEG Method are not Yet Reached. International Journal of Bioelectromagnetism 1(1999)1, pp. 2 - 3.

[7] Sanei, S., Chambers, J.A. EEG Signal Processing. West Sussex, England 2007, John Wiley & Sons. 312 p.

[8] Guérit, J.-M., Amantini, A., Amodio, P., Andersen, K.V., Butler, S., de Weerd, A., Facco, E., Fischer, C., Hantson, P., Jäntti, V., Lamblin, M.-D., Litscher, G., Péréon, Y. Consensus on the use of neurophysiological tests in the intensive care unit (ICU): Electroencephalogram (EEG), evoked potentials (EP), and electroneuromyography (ENMG). Clinical Neurophysiology 39(2009)2, pp. 71 - 83.

[9] Jäntti, V., Nikko, K., Kulkas, A., Heikkilä, H.T. Miten EEG auttaa tehohoidon hoitajaa työssään?. F.A.N.N. Jäsenlehti 23(2010), pp. 6 - 7.

[10] Najarian, K., Splinter, R. Biomedical Signal and Image Processing. Boca Raton, Florida 2005, The CRC Press. 448 p.

[11] Tong, S., Thakor, N.V. Quantitative EEG Analysis Methods and Clinical Applications. Norwood, Massachusetts 2009, Artech House Publishers. 421 p.

[12] Lokal_Profil, Dhp1080. 2006. Neuron. [SVG]. Wikimedia Commons. [viitattu 19.11.2011]. Saatavissa: http://commons.wikimedia.org/wiki/File:Neuron,_LangNeutral.svg.

[13] Stern, J.M., Engel, J. Atlas of EEG Patterns. Riverwoods, Illinois 2005, Lippincott Williams & Wilkins. 307 p.

[14] Nieminen, K. Neurofysiologisten mittausten tarve teho-osastolla. Finnanest, Suomen Anestesiologiyhdistyksen lehti 35(2002)2, pp. 137 - 139.

[15] Geocadin, R.G., Koenig, M.A., Jia, X., Stevens, R.D., Peberdy, M.A. Management of Brain Injury After Resuscitation From Cardiac Arrest. Neurologic Clinics 26(2008)2, pp. 487 - 506.

[16] Varsinais-Suomen Sairaanhoitopiiri 2011. Aivokuume - enkefaliitti. [WWW-sivu]. Varsinais-Suomen Sairaanhoitopiiri. [viitattu 21.10.2011]. Saatavissa:

http://ohjepankki.vsshp.fi/fi/2882/36787/.

[17] Neiman, E., Hanna, P. 2009. EEG in Dementia and Encephalopathy. Medscape [WWW-sivu]. Medscape. [viitattu 21.10.2011]. Saatavissa: http://emedicine.medscape.com/article/1138235-overview.

[18] Illis, L.S., Taylor, F.M. The Electroencephalogram in Herpes-simplex Encephalitis. The Lancet 299(1972)1, pp. 718 - 721.

[19] Rantala, H., Uhari, M. Lasten enkefaliitit. Lääketieteellinen Aikakausikirja Duodecim 108(1992)10, pp. 947 - 954.

[20] AUTH EEG Analysis group. 2006. Epileptic EEG records. [WWW-sivu]. AUTH EEG Analysis group. [viitattu 21.10.2011]. Saatavissa: http://eeganalysis.web.auth.gr/dataen.htm.

[21] AUTH EEG Analysis group. 2006. EDF Recording: ARS02NOV. [ZIP/EDF]. AUTH EEG Analysis group. [viitattu 21.10.2011]. Saatavissa: http://eeganalysis.web.auth.gr/Data/A/A.zip.

[22] Laureys, S., Berré, J., Goldman, S. Cerebral Function in Coma, Vegetative State, Minimally Conscious State, Locked-in Syndrome and Brain Death. Yearbook of Intensive Care and Emergency Medicine 22(2001), pp. 386 - 396.

[23] Laureys, S., Owen, A.M., Schiff, N.D. Brain function in coma, vegetative state, and related disorders. The Lancet: Neurology 3(2004)9, pp. 537 - 546.

[24] Synek, V.M. Prognostically important EEG coma patterns in diffuse anoxic and traumatic encephalopathies in adults. Clinical Neurophysiology 5(1988)2, pp. 161 - 174.

[25] Scollo-Lavizzari, G., Bassetti, C. Prognostic value of EEG in post-anoxic coma after cardiac arrest. European Neurology 26(1987)3, pp. 161 - 170.

[26] Hockaday, J.M., Potts, F., Epstein, E., Bonzzi, A., Schwab, R.S. Electroencephalograpic changes in acute cerebral anoxia from cardiac or respiratory arrest. Electroencephalography and Clinical Neurophysiology 18(1965)6, pp. 575 - 586.

[27] Matousek, M., Takeuchi, E., Starmark, J.E., Stalhammar, D. Quantitative EEG analysis as a supplement to the clinical coma scale RLS85. Acta Anaesthesiologica Scandinavica 40(1996)7, pp. 824 - 831.

[28] Söderholm, S., Meinander, M., Kahela, K., Alaranta, H. Suun motoriikan häiriöt ja korvaavat kommunikaatiomenetelmät “locked-in”-oireyhtymässä. Lääketieteellinen Aikakausikirja Duodecim 114(1998)9, pp. 879 - 886.

[29] Cruse, D., Chennu, S., Chatelle, C., Bekinschtein, T.A., Fernández-Espejo, D., Pickard, J.D., Laureys, S., Owen, A.M. Bedside detection of awareness in the vegetative state: a cohort study. The Lancet 378(2011)9809, pp. 2088 - 2094.

[30] Sackey, P.V. Frontal EEG for intensive care unit sedation: treating numbers or patients?. Critical Care 12(2008)5, pp. 128.

[31] Aho, A.J., Lyytikäinen, L.-P., Yli-Hankala, A., Kamata, K., Jäntti, V. Explaining Entropy responses after a noxious stimulus, with or without neuromuscular blocking agents, by means of the raw electroencephalographic and electromyographic characteristics. British Journal of Anaesthesia 106(2010)1, pp. 69 - 76.

[32] Sörnmo, L., Laguna, P. Bioelectrical Signal Processing in Cardiac and Neurolo-

gical Applications. Burlington, Massachusetts 2005, Elsevier Academic Press. 688 p.

[33] Rugg, M.D., Coles, M.G.H. Electrophysiology of Mind: Event-related Brain Potentials and Cognition. New York, USA 1996, Oxford University Press. 240 p.

[34] GE Healthcare 2009. EEG Module, EEG-E For continuous integrated neuromonitoring. [PDF]. GE Healthcare. [viitattu 6.11.2011]. Saatavissa: http://www.gehealthcare.com/euen/patient_monitoring/docs/E-EEG_M1034894_eng.pdf.

[35] Brain Products GmbH 2011. Analyzer 2: Innovation based on experience. [PDF]. Brain Products GmbH. [viitattu 6.11.2011]. Saatavissa: http://www.brainproducts.com/files/public/products/brochures_material/BV_Analyzer2.pdf.

[36] Pascual-Marqui, R.D., Michel, C.M., Lehmann, D. Low Resolution Electromagnetic Tomography: A New Method for Localizing Electrical Activity in the Brain. International Journal of Psychophysiology 18(1994)1, pp. 49 - 65.

[37] Brain Products GmbH 2011. BrainVision Recorder, Easy-to-use multifunctional recording software / BrainVision RecView, Software for realtime data analysis. [PDF]. Brain Products GmbH. [viitattu 6.11.2011]. Saatavissa: http://www.brainproducts.com/files/public/products/brochures_material/BV_Recorder-RecView.pdf.

[38] BrainVision RecView V 1.4 User Manual (version 003). 14.6.2010. Brain Products GmbH. Julkaisematon käyttöohje. 140 p.

[39] BrainVision Recorder 1.20 User Manual (version 005). 5.10.2010. Brain Products GmbH. Julkaisematon käyttöohje. 163 p.

[40] Janke, A. 2010. 4 Models of anatomy developed at the BIC. [PNG]. Wikimedia Commons. [viitattu 10.11.2011]. Saatavissa: http://commons.wikimedia.org/wiki/File:BIC-brainmodels.png.

[41] Neuroscan. 2006. Access SDK - Compumedics Neuroscan Real-Time SDK. [PDF]. Neuroscan. [viitattu 11.11.2011]. Saatavissa: http://www.neuroscan.com/AC147_2%20Neuro%20SDK%20LR.pdf?size=434250.

[42] Neuroscan. 2006. Scan 4.3 - Comprehensive EEG/ERP Acquisition and Analysis Software for Clinical and Research Applications in Neuroscience. [PDF]. Neuroscan. [viitattu 11.11.2011]. Saatavissa: http://www.neuroscan.com/3491A%20SCAN%204_3%20BROCHURE.pdf.

[43] Neuroscan. 2008. SynAmps RT. [PDF]. Neuroscan. [viitattu 11.11.2011]. Saatavissa: http://www.neuroscan.com/images/AE074_1%20SynAmps%20RT%20specs%20sheetfinal%20(3).pdf.

[44] Neuroscan. 2006. SCAN NuAmps Express. [PDF]. Neuroscan. [viitattu 11.11.2011]. Saatavissa: http://www.neuroscan.com/Express%201-24-06-05b.pdf.

[45] Neuroscan. 2007. CURRY 6 - True Multi-Modal Neuroimaging. [PDF]. Neuroscan. [viitattu 11.11.2011]. Saatavissa: http://www.neuroscan.com/AD326-01%20CURRY6%20brochureLR.pdf.

[46] Compumedics Ltd. 2011. Product details: ProFusion PSG. [WWW-sivu]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.compumedics.com/product_detail.asp?id=1&item=software.

[47] Compumedics Ltd. 2011. Product detail: ProFusion EEG. [WWW-sivu]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.compumedics.com/product_detail.asp?id=2&item=software.

[48] Grunwald, S. 2009. Siesta 802: Revolutionary Diagnostics for a Wireless World. [PDF]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.intelimed.com.mx/Brochures/Siestabrochure.pdf.

[49] Compumedics Ltd. 2011. Product details: E-Series. [WWW-sivu]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.compumedics.com/product_detail.asp?id=2&item=product.

[50] Compumedics Ltd. 2011. Product details: P-Series. [WWW-sivu]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.compumedics.com/product_detail.asp?id=10&item=product.

[51] Compumedics Ltd. 2011. Product details: Scan LT. [WWW-sivu]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.compumedics.com/product_detail.asp?id=18&item=product.

[52] Compumedics Ltd. 2007. Safiro: Ambulatory EEG recorder. [PDF]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.intelimed.com.mx/Brochures/neuroscience/SafiroEEGbrochure.pdf.

[53] Grunwald, S. 2009. Grael HD PSG/EEG: World's first high definition PSG. [PDF]. Compumedics Ltd. [viitattu 16.11.2011]. Saatavissa: http://www.biolinkarg.com/grael/folleto/grael-folleto.pdf.

[54] CareFusion 2011. Sleep Diagnostics. [PDF]. CareFusion. [viitattu 18.2.2012]. Saatavissa: http://www.carefusion.com/pdf/Respiratory/Sleep_Diagnostics_and_Therapy/Sleep_Diagnostic_Brochure_RC2606.pdf.

[55] CareFusion 2010. CareFusion Nicolet - Neurodiagnostics and Monitoring. [PDF]. CareFusion. [viitattu 18.2.2012]. Saatavissa: http://www.carefusion.com/pdf/Neurology/NeuroCare_Brochure_Revised_8-2011.pdf.

[56] Micromed S.p.A. 2010. BRAIN QUICK LTM. [PDF]. Micromed S.p.A. [viitattu 18.2.2012]. Saatavissa: http://www.micromed.eu/pdf/11-1-BRAIN_QUICK_LTM_ENG_2.02_Low_Res.pdf.

[57] Micromed S.p.A. 2009. MYOQUICK Matrix Line. [PDF]. Micromed S.p.A. [viitattu 18.2.2012]. Saatavissa: http://www.micromed.eu/pdf/6-1-MYOQUICK_Matrix_Line_ENG_2.02_Low_Res.pdf.

[58] Micromed S.p.A. 2009. BRAIN QUICK: VideoEEG for Networked Neurophysiology. [PDF]. Micromed S.p.A.. [viitattu 18.2.2012]. Saatavissa: http://www.micromed.eu/pdf/5-1-BRAIN_QUICK_ENG_2.02_Low_Res.pdf.

[59] Cadwell Laboratories, Inc. 2010. EASY Ambulatory PSG/EEG. [PDF]. Cadwell Laboratories, Inc. [viitattu 21.2.2012]. Saatavissa: http://www.cadwell.com/products/psg/easyambulatorypsg/ambulatorybrochure.pdf.

[60] Cadwell Laboratories, Inc. 2011. Easy III EEG. [PDF]. Cadwell Laboratories, Inc. [viitattu 21.2.2012]. Saatavissa: http://www.cadwell.com/products/eeg/easy3eeg/easy3eegbrochure.pdf.

[61] Cadwell Laboratories, Inc. 2009. Easy III PSG. [PDF]. Cadwell Laboratories, Inc. [viitattu 21.2.2012]. Saatavissa: http://www.cadwell.com/products/psg/easy3psg/easy3psgbrochure.pdf.

[62] Cadwell Laboratories, Inc. 2011. CASCADE Pro. [PDF]. Cadwell Laboratories, Inc. [viitattu 21.2.2012]. Saatavissa: http://www.cadwell.com/products/ionm/cascadepro/cascadeprobrochure.pdf.

[63] Kremer, J.M. 2012. MP System Hardware Guide. [PDF]. BIOPAC Systems Inc. [viitattu 21.2.2012]. Saatavissa: http://www.biopac.com/Manuals/mp_hardware_guide.pdf.

[64] BIOPAC Systems Inc. 2012. MP150 Data Acquisition System GLP. [WWW-sivu]. BIOPAC Systems Inc. [viitattu 21.2.2012]. Saatavissa: http://www.biopac.com/data-acquisition-system-mp150-system-glp-win-specifications#LowerTab.

[65] BIOPAC Systems Inc. 2012. Biopac Science Lab. Intro system - SCILAB-I. [WWW-sivu]. BIOPAC Systems Inc. [viitattu 21.2.2012]. Saatavissa: http://www.biopac.com/biopac-science-lab-teaching-system-specifications#LowerTab.

[66] Kremer, J.M., Macy, A.J., Peterlin, E. 2008. AcqKnowledge 4 Software Guide. [PDF]. BIOPAC Systems Inc. [viitattu 21.2.2012]. Saatavissa: http://www.biopac.com/Manuals/acqknowledge_software_guide.pdf.

[67] Kremer, J.M., Macy, A.J. 2009. AcqKnowledge: Analysis Scoring & Automation. [PDF]. BIOPAC Systems Inc. [viitattu 21.2.2012]. Saatavissa: http://www.biopac.com/manuals/AcqKnowledge-Specialized.pdf.

[68] Oostenveld, R. 2012. Start - FieldTrip. [WWW-sivu]. Radboud Universiteit Nijmegen, Donders Institute. [viitattu 17.2.2012]. Saatavissa: http://www.ru.nl/donders/fieldtrip.

[69] Oostenveld, R. 2012. Publications - FieldTrip. [WWW-sivu]. Radboud Universiteit Nijmegen, Donders Institute. [viitattu 17.2.2012]. Saatavissa: http://fieldtrip.fcdonders.nl/publications.

[70] Oostenveld, R. 2012. Development:realtime:buffer_overview - FieldTrip. [WWW-sivu]. Radboud Universiteit Nijmegen, Donders Institute. [viitattu 17.2.2012]. Saatavissa: http://fieldtrip.fcdonders.nl/development/realtime/buffer_overview.

[71] Oostenveld, R. 2012. Development:realtime:implementation - FieldTrip. [WWW-sivu]. Radboud Universiteit Nijmegen, Donders Institute. [viitattu 17.2.2012]. Saatavissa: http://fieldtrip.fcdonders.nl/development/realtime/implementation.

[72] Cieślik, M., Mura, C. 2011. A lightweight, flow-based toolkit for parallel and distributed bioinformatics pipelines. BMC Bioinformatics [online journal]. BioMed Central. [viitattu 22. syyskuuta 2011]. Saatavissa: http://www.biomedcentral.com/1471-2105/12/61.

[73] Cieślik, M., Mura, C. 2010. PaPy: Documentation and user's guide for the pac-

kages PaPy, NuMap & NuBio/NuBox. [PDF]. University of Virginia, Mura Lab. [viitattu 17.2.2012]. Saatavissa: http://muralab.org/PaPy/docs/PaPy_toolkit_doc.pdf.

[74] NeurOne System User Manual. 23.12.2010. Mega Elektroniikka Oy. Julkaisematon käyttöohje. 94 p.

[75] Kemp, B., Värri, A., Rosa, A.C., Nielsen, K.D. & Gade, J. A simple format for exchange of digitized polygraphic recordings. Electroencephalography and Clinical Neurophysiology 82(1992)5, pp. 391-393.

[76] Kemp, B. & Olivan, J. European data format 'plus' (EDF+), an EDF alike standard format for the exchange of physiological data. Clinical Neurophysiology 114(2003)9, pp. 1755-1761.

[77] Buttazzo, G., Lipari, L.A., Caccamo, M. Soft Real-Time Systems: Predictability vs. Efficiency. New York 2005, Springer Science + Business Media. 275 p.

[78] Kuo, S.M., Lee, B.H., Tian, W. Real-Time Digital Signal Processing: Implementations and Applications, 2nd Edition. West Sussex, England 2006, John Wiley & Sons. 664 p.

[79] Russinovich, M.E., Solomon, D.A., Ionescu, A. Windows® Internals, 5th Edition. Redmond, Washington 2009, Microsoft Press. 1264 p.

[80] Microsoft. 2011. Copying and Pinning. [WWW-sivu]. Microsoft. [viitattu 13.10.2011]. Saatavissa: http://msdn.microsoft.com/en-us/library/23acw07k(v=VS.90).aspx.

[81] Oppenheim, A.V., Schafer, R.W. Discrete-Time Signal Processing: International Version, 3rd Edition. Upper Saddle River, New Jersey 2009, Pearson Higher Education Inc. 1120 p.

[82] Stallings, W. Data and Computer Communications, 8th Edition. New Jersey 2007, Pearson Prentice Hall. 896 p.

[83] Bang-Jensen, J., Gutin, G. Digraphs: Theory, Algorithms and Applications. London, U.K. 2007, Springer-Verlag. 776 p.

[84] Gebali, F. Algorithms and Parallel Computing. Hoboken, New Jersey 2011, John Wiley & Sons, Inc. 364 p.

[85] Campbell, C., Miller, A. Parallel programming with Microsoft Visual C++. Redmond, Washington 2011, Microsoft Press. 208 p.

[86] Microsoft 2011. About BITS. [WWW-sivu]. Microsoft. [viitattu 25.2.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/windows/desktop/aa362708%28v=vs.85%29.aspx.

[87] University of Southern California, Information Sciences Institute 1981. RFC 793 - Transmission Control Protocol. [WWW-sivu]. Internet Engineering Task Force. [viitattu 12.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc793.

[88] Braden, R. 1989. RFC 1122 - Requirements for Internet Hosts - Communication Layers. [WWW-sivu]. Internet Engineering Task Force. [viitattu 12.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc1122.

[89] Nagle, J. 1984. RFC 896 - Congestion Control in IP/TCP Internetworks. [WWW-sivu]. Internet Engineering Task Force. [viitattu 12.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc896.

[90] Mogul, J., Minshall, G. Rethinking the TCP Nagle algorithm. ACM SIGCOMM Computer Communication Review 31(2001)1, pp. 6 - 20.

[91] Lindfors, H. SCTP - kehittynyt kuljetusprotokolla. Diplomityö. Tampere 2005. Tampereen teknillinen yliopisto. Tietoliikennetekniikan laitos. 91 p.

[92] Pessach, Y. 2006. Take Total Control Of Your Networking With .NET And UDP. [WWW-sivu]. Microsoft. [viitattu 14.2.2012]. Saatavissa: http://msdn.microsoft.com/en-us/magazine/cc163648.aspx.

[93] Postel, J. 1980. User Datagram Protocol. [WWW-sivu]. The Internet Society. [viitattu 14.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc768.

[94] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., Paxson, V. 2000. Stream Control Transmission Protocol. [WWW-sivu]. The Internet Society. [viitattu 11.2.2012]. Saatavissa: http://www.ietf.org/rfc/rfc2960.txt.

[95] Stewart, R. 2005. SCTP Implementations. [WWW-sivu]. SCTP.ORG. [viitattu 12.2.2012]. Saatavissa: http://www.sctp.org/implementations.html.

[96] Hayes, D.A., But, J., Armitage, G. Issues with Network Address Translation for SCTP. ACM SIGCOMM Computer Communication Review 39(2009)1, pp. 24 - 33.

[97] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. 1996. RTP: A Transport Protocol for Real-Time Applications. [WWW-sivu]. The Internet Society. [viitattu 25.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc1889.

[98] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. 2003. RTP: A Transport Protocol for Real-Time Applications. [WWW-sivu]. The Internet Society. [viitattu 25.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc3550.

[99] Amer, P.D., Stewart, R. 2007. Why is SCTP needed given TCP and UDP are widely available?. [WWW-sivu]. Internet Society. [viitattu 27.1.2012]. Saatavissa: http://www.isoc.org/briefings/017/.

[100] Antoniol, G., Tonella, P. EEG Data Compression Techniques. IEEE Transactions on Biomedical Engineering 44(1997)2, pp. 105 - 114.

[101] Wongsawat, Y., Oraintara, S., Tanaka, T., Rao, K.R. Lossless multi-channel EEG compression. IEEE International Symposium on Circuits and Systems (2006), pp. 1611 - 1614.

[102] Memon, N., Kong, X., Cinkler, J. Context-based lossless and near-lossless compression of EEG signals. IEEE Transactions on Information Technology in Biomedicine 3(1999)3, pp. 231 - 238.

[103] Kemp, B. 2012. EDF(+) compatible companies. [WWW-sivu]. Bob Kemp. [viitattu 25.2.2012]. Saatavissa: http://www.edfplus.info/companies/companies.html.

[104] Gordon, P. 2008. EDF/EDF+ (European Data Format). [WWW-sivu]. CodePlex.com, Microsoft. [viitattu 25.2.2012]. Saatavissa: http://edf.codeplex.com/.

[105] BioSemi. 2006. Which file format does BioSemi use?. [WWW-sivu]. BioSemi. [viitattu 25.2.2012]. Saatavissa: http://www.biosemi.com/faq/file_format.htm.

[106] Ebml.sourceforge.net. 2004. EBML - the opensource Extensible Binary Meta-Language. [WWW-sivu]. Sourceforge.net. [viitattu 25.2.2012]. Saatavissa:

http://ebml.sourceforge.net/.

[107] Dierks, T., Rescorla, E. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. [WWW-sivu]. The Internet Society. [viitattu 24.2.2012]. Saatavissa: http://tools.ietf.org/html/rfc5246.

[108] Richter, J. CLR via C#, 3rd Edition. Redmond, Washington 2010, Microsoft Press. 896 p.

[109] Rammer, I., Szpuszta, M. Advanced .NET Remoting, Second Edition. New York 2005, APress. 608 p.

[110] Microsoft 2012. Interprocess Communications. [WWW-sivu]. Microsoft. [viitattu 8.3.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx.

[111] Microsoft 2012. CreateNamedPipe function. [WWW-sivu]. Microsoft. [viitattu 1.3.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/windows/desktop/aa365150(v=vs.85).aspx.

[112] MathWorks. 2012. Working with the MCR :: Deployment Process (MATLAB® Compiler™). [WWW-sivu]. MathWorks. [viitattu 10.3.2012]. Saatavissa: http://www.mathworks.se/help/toolbox/compiler/f12-999353.html.

[113] International Telecommunication Union. 2009. ITU-T X.667: Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components. [PDF]. International Telecommunication Union. [viitattu 12.10.2011]. Saatavissa: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.667-200808-I!!PDF-E&type=items.

[114] MathWorks. 2011. MATLAB Builder NE 4.0 for Microsoft .NET Framework. [PDF]. MathWorks. [viitattu 8.10.2011]. Saatavissa: http://www.mathworks.com/products/datasheets/pdf/matlab-builder-ne.pdf.

[115] MathWorks. 2011. What is the version of the MATLAB Compiler Runtime (MCR) that corresponds to the version of MATLAB Compiler I am using?. [WWW-sivu]. MathWorks. [viitattu 8.10.2011]. Saatavissa: http://www.mathworks.se/support/solutions/en/data/1-4GSNCF/index.html?solution=1-4GSNCF.

[116] Google. 2012. Developer Guide - Protocol Buffers - Google Code. [WWW-sivu]. Google Inc. [viitattu 16.3.2012]. Saatavissa: http://code.google.com/apis/protocolbuffers/docs/overview.html.

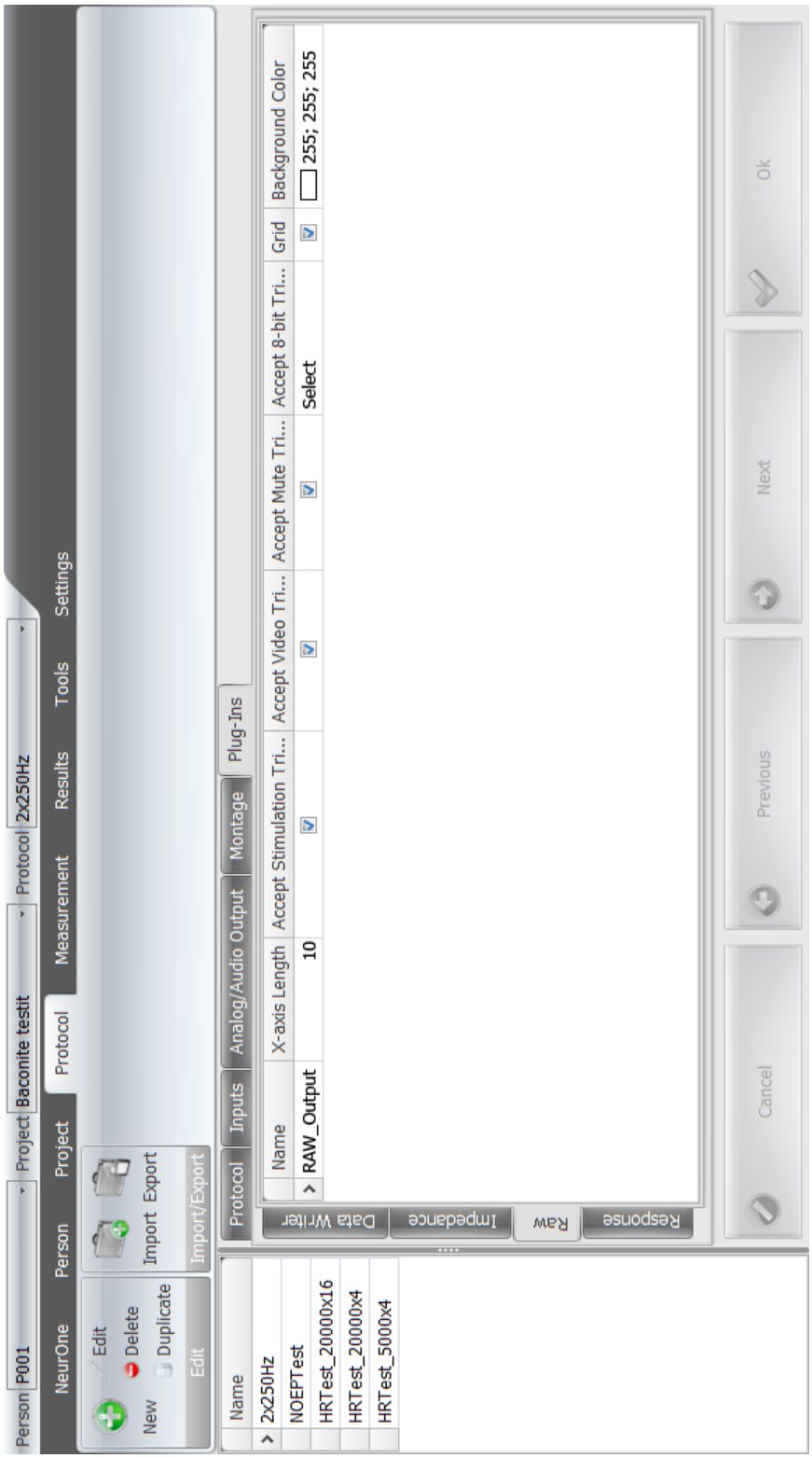
[117] Pitkänen, P., Lipping, T. NeurOne network interface "Baconite" - Requirement specification. 2011. Tampereen teknillinen yliopisto. Vaatimusmääritelmä. 11 p.

[118] Pitkänen, P., Lipping, T. NeurOne network interface "Baconite" - Functionality and performance test plan. 2012. Tampereen teknillinen yliopisto. Testaussuunnitelma. 17 p.

[119] Pitkänen, P., Lipping, T. Baconite Functionality test report. 2012. Tampereen teknillinen yliopisto. Testausraportti (laskentataulukko). 4 p.

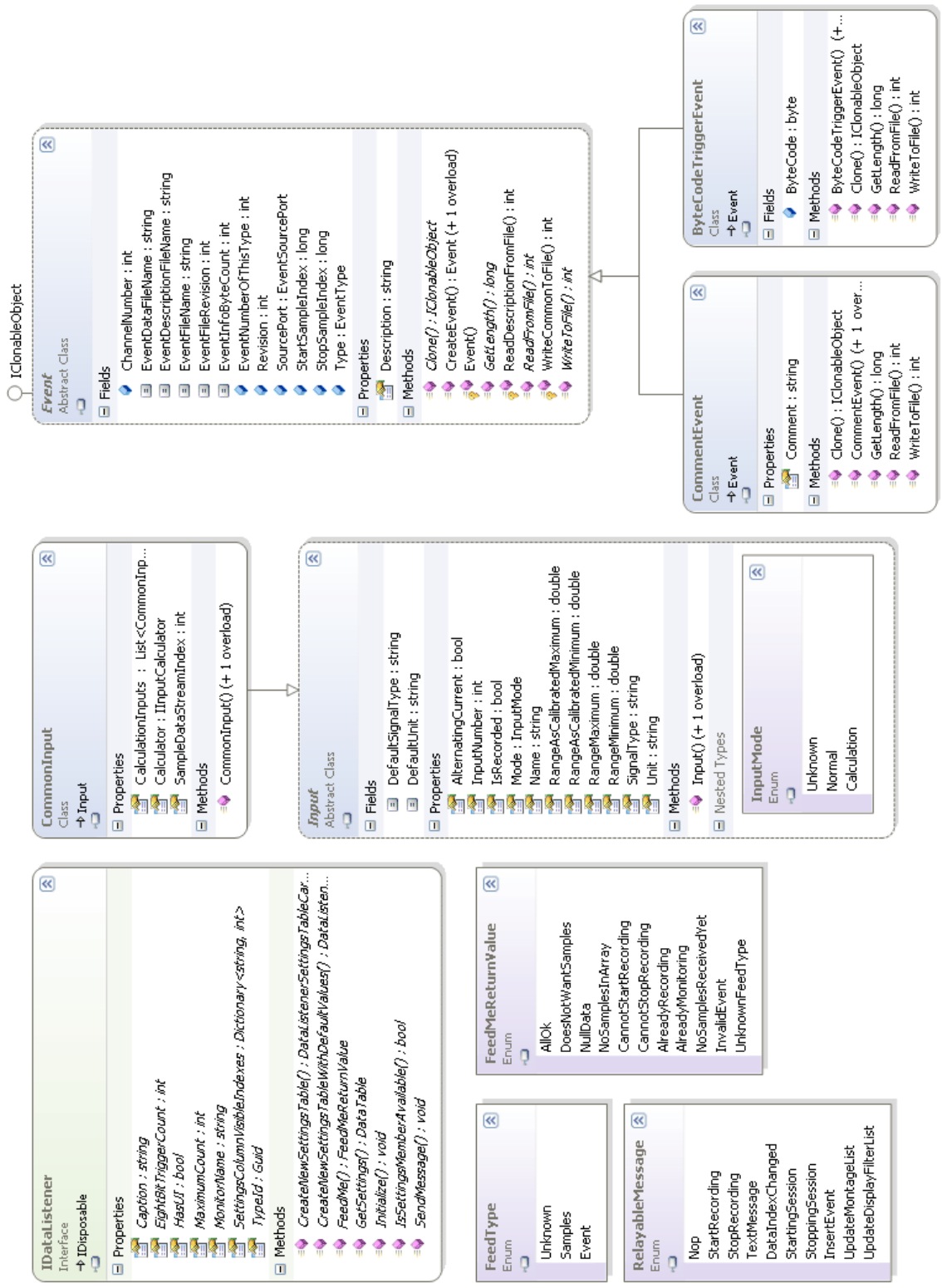
[120] Pitkänen, P. Baconite Performance test report. 2012. Tampereen teknillinen yliopisto. Testausraportti (laskentataulukko). 1 p.

LIITE 1: LIITÄNNÄISTEN KONFIGUROINTI NEURONE-SOVELLUKSESSA



Kuva L1.1. *Liitännäisten konfigurointi tapahtuu NeurOne-sovelluksessa taulukkomuotoisen käyttöliittymän kautta.*

LIITE 2: NEURONE-LIITÄNNÄISTEN OHJELMOINTIRAJAPINTA



Kuva L2.1. *NeurOne-liitännäisten ohjelmointirajapinnan keskeisimmät osat.*

LIITE 3: ESIMERKKI .NET-TAULUKON TYYPPIMUUNNOKSESTA

Alla kuvattu esimerkki (L3.1) havainnollistaa, kuinka .NET-ohjelmassa voidaan serialisoida taulukollinen liukulukuja. Kyseessä siis ei ole tyyppimuunnos, missä yksi liukuluku (double) muunnetaan yhdeksi tavuksi (byte) vaan operaatio, jossa liukuluvun muistissa oleva esitys (64 bittiä) tulkitaan joukoksi tavuja.

```
// Send- funktion prototyyppi, joka vaatii taulukon tavuja
static void Send(byte[] data);

// Tietoa käsittelevä funktio, jota kutsutaan muualta sovelluksesta
unsafe static void ProcessAndSend(double[] data)
{
      // Käsittele tietoa
      // ...
      // Seuraavaksi pitäisi välittää tieto lähetysfunktiolle, joka
      // odottaa saavansa taulukon tavuja.

      // C++:ssa voitaisiin käsitellä data-muuttujaa osoittimena
      // taulukkoon tavuja:
      unsigned char* dataBytes = (unsigned char*)data;
      Send(dataBytes);
      // tämä ei kuitenkaan onnistu managed.NET maailmassa, koska
      // data-muuttuja ei ole pelkkä osoitin vaan olio

      // Alla on esitetty kaksi kopiointiin perustuvaa vaihtoehtoa:
      // Varaa muistia uudelle tavutaulukolle, johon data kopioidaan
      byte[] byteBuf = new byte[data.Length * 8];

      // VAIHTOEHTO 1: kopioi tieto lohkona tavuja
      Buffer.BlockCopy(data, 0, byteBuf, 0, byteBuf.Length);

      // VAIHTOEHTO 2: Kopioi näytteet yksi kerrallaan
      // käy läpi jokainen näyte
      for (int i=0; i<data.Length; i++)
      {
            // muunna jokainen double -> byte[8] ja varastoi se
            // taulukkoon. Tämä on myös alustaturvallinen tapa
            // (little / big endianess)
            Array.Copy( BitConverter.GetBytes(data[i]), 0, byteBuf,
                  i * sizeof(double), sizeof(double) );
      }

      // kutsu lopuksi lähetysfunktiota - tämä vaatii byte[]-
      // parametrin
      Send(byteBuf);
}
```

Ohjelma L3.1. *Liukulukutaulukon käsittely raakadatana (byte[ ]).*

LIITE 4: MITTAUSTIEDON SIIRTOKANAVAN NOPEUSTESTIT

Tässä liitteessä kuvataan lyhyesti metodi, jota käytettiin arvioitaessa eri menetelmien soveltuvuutta NeurOne- ja käsittelysovelluksen väliseen mittaustiedon siirtoon. Lisäksi ohessa esitetään saadut mittaustulokset sekä näistä tehdyt johtopäätökset.

Soveltuvuuden arviointi tapahtui yhden kriteerin (tiedonsiirtonopeus) perusteella. Muita kriteerejä olisivat voineet olla esimerkiksi toteutuskustanne (aika ja raha) ja toimintavarmuus. Mitattavaksi valittiin viisi eri menetelmää:

- Jaettu muisti (shared memory)
- Nimetyt putket (named pipes)
- TCP/IP-yhteys
- Remoting, kommunikaatiokanavana System.Runtime.Remoting.Channels.Tcp.TcpChannel
- Remoting, kommunikaatiokanavana System.Runtime.Remoting.Channels.Ipc.IpcChannel

Koska remotingin tiedonsiirto pohjautuu TCP/IP-protokollaan (TcpChannel) tai nimettyihin putkiin (IpcChannel) odotettiin tämän olevan huomattavasti hitaampi. Lisäksi remoting-menetelmissä tieto kulkee sekä lähetyksessä että vastaanotossa erillisen serialisointimekanismin läpi, joka huonontaa suorituskykyä entisestään. Se otettiin testattavaksi lähinnä vertailupohjan saamiseksi. Myös TCP/IP-yhteyden odotettiin suoriutuvan huonommin, sillä mekanismeiltaan se on nimettyjä putkia ja jaettua muistia monimutkaisempi.

Aluksi toteutettiin .NET-alustalla toimiva sovelluspohja, joka sisälsi tarvittavat ajastus- ja ohjausmekanismit mittaustulosten saamiseksi. Sovelluspohjaan toteutettiin komentolinjaparametrit, joilla valitaan mm. siirrossa käytettävän lohkon koko tavuina, siirrettävien lohkojen määrä, sekä toimiiko sovellus tiedon vastaanottajana (palvelin) vai lähettäjänä (asiakas). Palvelimen ja asiakkaan pseudokoodi on esitetty ohjelmalistauksissa L4.1 ja L4.2. Seuraavaksi sovelluspohjasta tehtiin neljä kopiota (yksi jokaista testattavaa menetelmää varten, kumpikin remoting-menetelmä samassa sovelluksessa) ja toteutettiin kuhunkin pohjaan oma siirtomenetelmänsä. Näistä jokainen pyrittiin tekemään lohko-orientoituneeksi siten, että lähettäjän toimittama lohko otetaan vastaan kokonaisena. Tämä ei ollut mahdollista TCP/IP-yhdeyden tapauksessa, sillä se on luonteeltaan tietovirtaorientoitunut. Lohkon sisältö määritettiin taulukoksi tavuja (C-kielellä ilmaistuna lohkon tietotyyppi oli siis *unsigned char [ ]*).

```
Palvelin()
{
      Odota yhteyttä
      Vastaanota N kappaletta B tavun kokoisia lohkoja
      Sulje yhteys
}
```

Ohjelma L4.1. *Palvelinohjelman pseudokoodi.*

```
Asiakas()
{
      Yhdistä palvelinohjelmaan
      Alusta lähetettävä B tavun kokoinen lohko satunnaistiedolla
      Aloita ajan mittaus
      Lähetä N kappaletta B tavun kokoisia lohkoja
      Lopeta ajan mittaus
      Sulje yhteys
}
```

Ohjelma L4.2. *Asiakasohjelman pseudokoodi.*

Seuraavaksi testialustana toimivan tietokoneen palomuuri kytkettiin pois päältä ja tietokone käynnistettiin uudelleen, antaen sen asettua käynnistymisen jälkeen noin viiden minuutin ajan. Tällä pyrittiin varmistamaan, että myös kaikki käyttöjärjestelmän taustapalvelut ovat käynnistyneet eivätkä kuormita järjestelmää. Tämän jälkeen avattiin windowsin tehtävienhallinta sekä varmistettiin prosessorin- ja levynkäyttöä tarkkailemalla ettei järjestelmässä ilmene aktiviteettia.

Jokaisella siirtomenetelmällä ajettiin kuusi testitapausta, kukin erisuuruista lohkon kokoa käyttäen. Useita eri lohkojen kokoja testattiin, sillä haluttiin nähdä onko tällä vaikutusta menetelmän tehokkuuteen. Kussakin testitapauksessa siirrettiin sata tuhatta (100 000) lohkoa. TCP/IP-protokollapinoa hyödyntävissä menetelmissä (TCP/IP, Remoting) yhteys muodostettiin paikalliseen loopback-osoitteeseen 127.0.0.1.

Aluksi testit suoritettiin tekijän omalla kotitietokoneella, jonka kokoonpano on esitetty taulukossa L4.1 (kyseiset tiedot on koottu ohjelmalla nimeltä CPU-Z). Saadut tulokset on esitetty taulukossa L4.2 ja kuvassa L4.1. Taulukossa on esitetty tilan säästämiseksi vain eri menetelmien saavuttamat nopeudet S, jotka on laskettu kertomalla lohkon koko B siirrettyjen lohkojen määrällä N ja jakamalla tämä siirtoon kuluneella ajalla T:

$$S = \frac{B \cdot N}{T} \tag{L4.1}$$

Tuloksista nähdään, että jaettu muisti, nimetyt putket ja TCP/IP ovat tasaväkisiä pienellä lohkon koolla. Kommunikaatiokanavasta riippumatta remoting on selkeästi hitain menetelmä. Jaettu muisti ja nimetyt putket suoriutuvat ylivoimaisesti verrattuna muihin menetelmiin. Näiden välille alkaa muodostua eroa yli 16 kilotavun kokoisia lohkoja kerrallaan siirrettäessä. Suurella 1 megatavun lohkokoolla jaettu muisti toimii noin 50 % nimettyjä putkia nopeammin, yltäen noin 1,4 gigatavun sekuntivauhtiin.

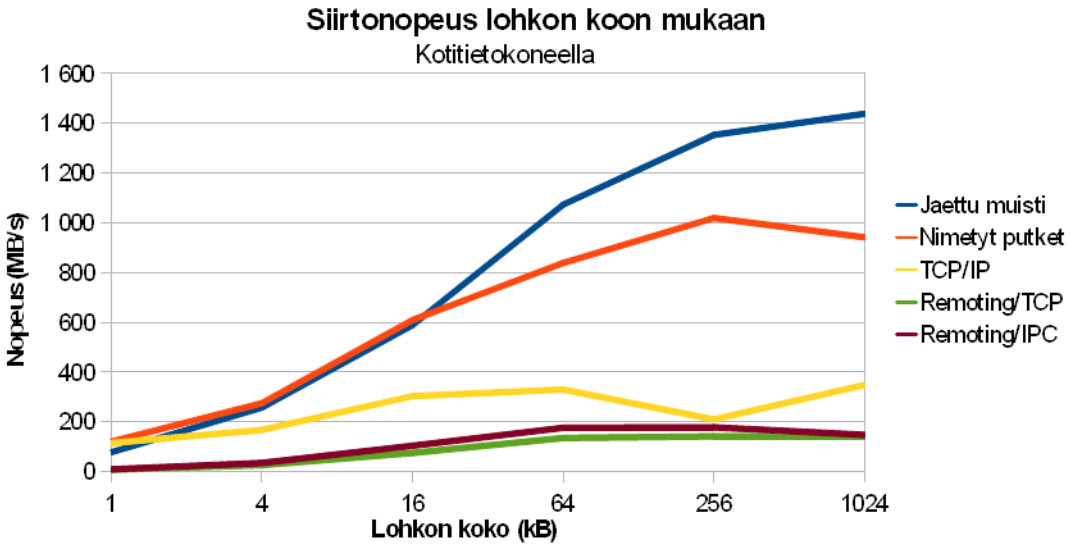
Taulukko L4.1. *Testeissä käytetyjen tietokoneiden kokoonpanot (CPU-Z).*

	Kotitietokone	**TTY:n kannettava**
Prosessori	Intel Core2 Duo E6600 (2,4 GHz)	Intel Mobile Core2 Duo T9600 (2,9 GHz)
Keskusmuisti	2048 MB DDR2 / 333,3 MHz	2520 MB DDR3 / 532,0 MHz
Väylätaajuus (FSB)	266,7 MHz	266,0 MHz
L2 välimuisti	4 MB	6 MB

Käyttöjärjestelmä	Windows XP Professional SP2, 32-bit	Windows 7 Enterprise SP1, 32-bit

Taulukko L4.2. *Eri menetelmien siirtonopeus kotitietokoneella.*

Lohkon koko N (kB)	**Jaettu muisti (MB/s)**	**Nimetyt putket (MB/s)**	**TCP/IP (MB/s)**	**Remoting/TCP (MB/s)**	**Remoting/IPC (MB/s)**
1	77,9	120,5	115,4	8,1	9,7
4	257,7	275,1	167,8	26,3	35,1
16	589,2	609,6	303,6	75,3	104,4
64	1 073,4	839,1	330,2	135,6	176,6
256	1 353,5	1 020,0	210,0	142,2	178,2
1024	1 438,6	941,8	349,0	140,0	148,2

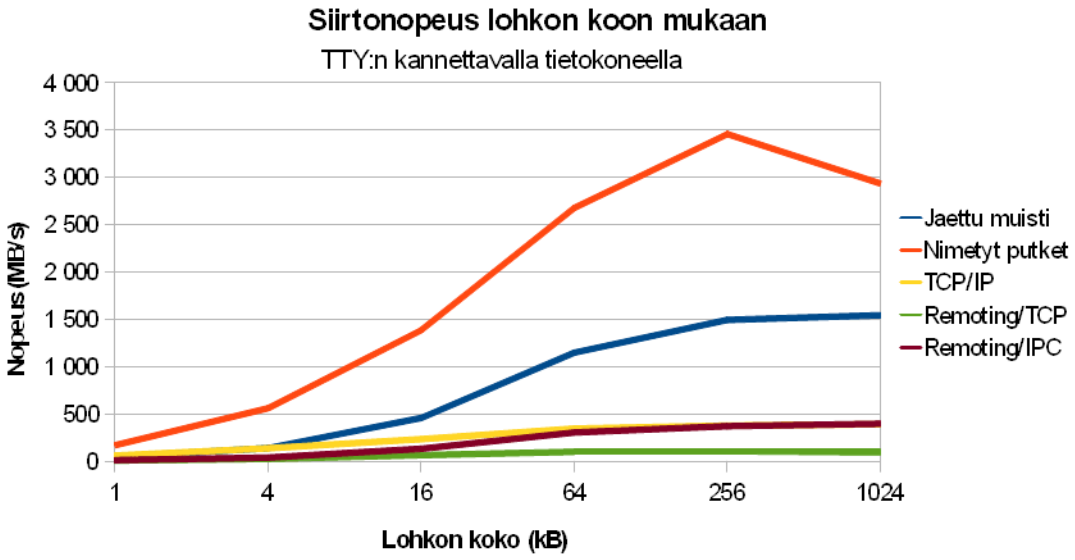


Kuva L4.1. *Siirtonopeus lohkon mukaan kotitietokoneella.*

Myöhemmin testit suoritettiin myös TTY:n kannettavalla tietokoneella (kokoonpano taulukossa L4.1). Näistä saadut tulokset on esitetty taulukossa L4.3 ja kuvassa L4.2. Yllätykseksi nähdään, että tällä kokoonpanolla nimetyt putket ovat huomattavasti jaettua muistia tehokkaampia. Jaetun muistin nopeus on lähes sama kuin kotikoneella tehdyissä testeissä jääden noin 1,5 gigatavuun, mutta nimettyjen putkien nopeus on kotikoneeseen verrattuna noin kolminkertainen, yltäen jopa kolmeen gigatavuun sekunnissa. Syynä voivat olla käyttöjärjestelmän ytimessä tapahtuneet muutokset nimettyjen putkien – tai vaihtoehtoisesti jaetun muistin – käsittelyyn. Myös nimettyihin putkiin nojaava Remoting (IPC) suoriutuu yli kaksinkertaisella nopeudella kotikoneeseen verrattuna.

Taulukko L4.3. *Eri menetelmien siirtonopeus TTY:n kannettavalla tietokoneella.*

Lohkon koko N (kB)	Jaettu muisti (MB/s)	Nimetyt putket (MB/s)	TCP/IP (MB/s)	Remoting/TCP (MB/s)	Remoting/IPC (MB/s)
1	45,6	168,9	59,6	7,5	11,0
4	138,9	560,1	137,5	25,0	38,3
16	457,5	1 385,9	233,9	63,5	131,1
64	1 147,3	2 678,4	345,4	99,9	303,8
256	1 493,8	3 458,5	376,5	106,6	370,7
1024	1 544,0	2 936,3	390,4	94,1	396,7



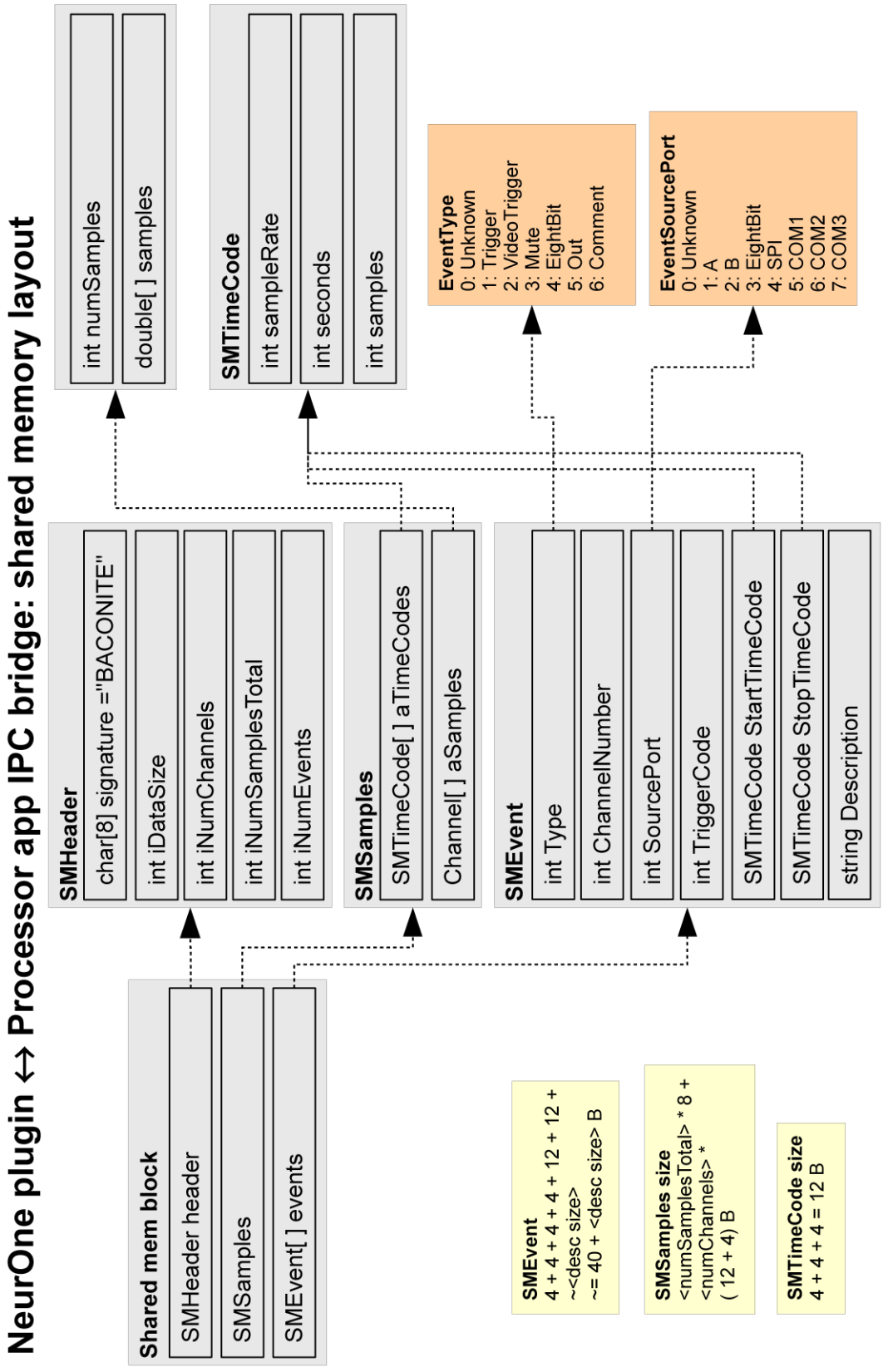
Kuva L4.2. *Siirtonopeus lohkon mukaan TTY:n kannettavalla tietokoneella.*

Saatujen tulosten perusteella nimetyt putket ja jaettu muisti vaikuttavat selkeästi olevan testatun joukon nopeimpia menetelmiä. Niiden nopeusero muihin menetelmiin selittynee osin sillä, että molemmat ovat mekanismeina suhteellisen yksinkertaisia. Lisäksi vaikuttaa sille, että testialustalla (laitteisto ja käyttöjärjestelmä) on suuri vaikutus näiden suorituskykyjen keskinäiseen suhteeseen.

Kummassakin testimittauksessa nimetyillä putkilla suorituskyky on parhaimmillaan 256 kilotavun lohkoilla. Tämä voi johtua esimerkiksi käyttöjärjestelmän käyttämien nimetyille putkille varattujen puskurien koosta.

Testausprotokolla on siltä osin puutteellinen, että mittaukset tehtiin vain kerran. Täten saaduissa mittaustuloksissa ei ole tietoa mahdollisesta tarkkuudesta tai vaihteluvälistä. Menetelmien suhteellinen nopeus toisiinsa verrattuna on sama kummallakin mittausalustalla – nimettyihin putkiin nojaavia menetelmiä lukuun ottamatta. Tämän nojalla voidaan kuitenkin todeta, että testatuista menetelmistä nimetyt putket ja jaettu muisti soveltuvat lohkotyyppisen tiedon siirtoon muita paremmin.

LIITE 5: JAETUN MUISTIALUEEN RAKENNE



Kuva L5.1. *Jaetun muistialueen rakenne.*

LIITE 6: YKSINKERTAINEN MATLAB-SUODIN

```
% Yksinkertainen MATLAB-suodin, lohkokuvaaaja
function BDesc = GetBlockDescriptor
      % Lohkon yksilöllinen tunniste
      BDesc.UNID = '{F54AB597-80D8-46A8-95BE-7650DF85201E}';
      % Lohkon nimi (esillä käsittelysovelluksessa)
      BDesc.Name = 'R2010b Test block';
      % Lohkon kategoria (esillä käsittelysovelluksessa)
      BDesc.Category = 'Tests';
      % Tekijä ja versio
      BDesc.Author = 'Paavo Pitkänen';
      BDesc.Version = '1.0.0.0';
      % Asetuksia voi muokata
      BDesc.HasSettingsUI = true;
      % Yksi kiinteä kanava X1, jota vastaa ulostulo Y1
      BDesc.Inputs = { 'X1' };
      BDesc.Outputs = { 'Y1' };
      % loput kanavista ovat käyttäjän muutettavisssa
      BDesc.VarInputPrefix = 'chan';
end
```

Ohjelma L6.1. *MATLAB-lohkon lohkokuvaaja.*

```
% Yksinkertainen MATLAB-suodin, alustusfunktio
function ctx = Initialize(ctx)
      % Aseta skaalaustekijäksi 2
      ctx.config.scalingFactor = 2;
      % aja mukautetut asetukset (cfgEval on MATLAB-koodia)
      if (isfield(ctx.config, 'cfgEval'))
            eval(ctx.config.cfgEval);
      end
end
```

Ohjelma L6.2. *MATLAB-lohkon alustusfunktio.*

```
% Yksinkertainen MATLAB-suodin, skaalaa jokaisen sisääntulevan
% näytteen asetuksissa määrätyllä luvulla
function ctx = Process(ctx)
      % hae kanavien lukumäärä apumuuttujaan
      numChans = length(ctx.proc.x);

      % skaalaa näytteet
      if (ctx.config.scalingFactor ~= 1)
            % käsittele jokainen kanava
            for i = 1 : numChans
                  % skaalaa sisääntulvien näytteiden vektori
                  % (proc.x) kontekstin ulostuloihin (proc.y)
                  ctx.proc.y{i} = ctx.proc.x{i} .* ...
                  ctx.config.scalingFactor;
            end
      end
end
```

Ohjelma L6.3. *MATLAB-lohkon käsittelyfunktio.*

LIITE 7: TIEDONSIIRTOPROTOKOLLAN VIESTIRAKENNE

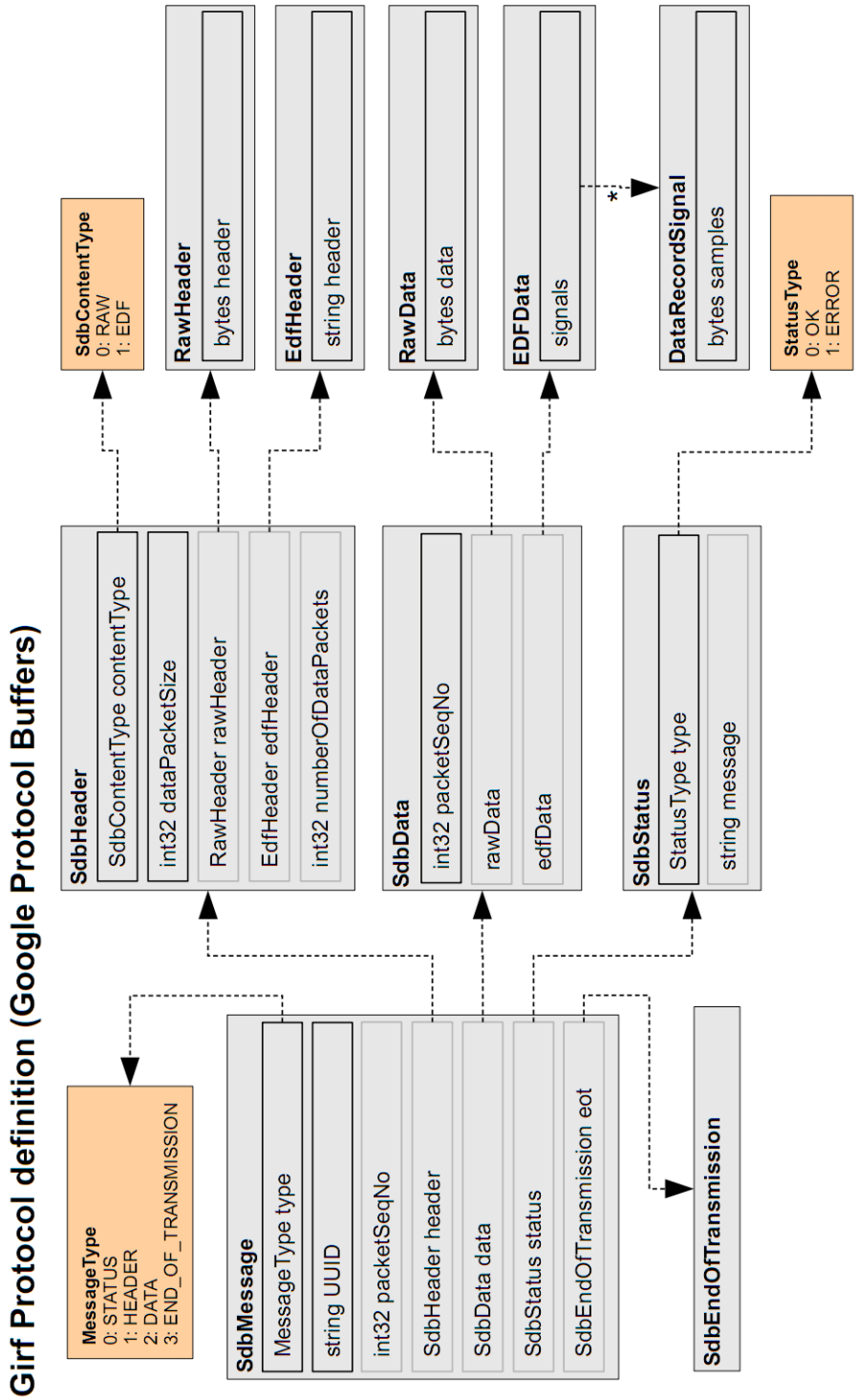
Tässä liitteessä kuvataan työssä toteutetun tiedonsiirtoprotokollan viestien rakenne. Jokainen viesti on .NET-luokan (SdbMessage, kuva L7.1) instanssi, joka serialisoidaan *Google Protocol Buffers* -tekniikalla. Se sisältää ensimmäisenä kenttänään enumeraation (MessageType), joka kertoo minkä tyyppinen viesti on kyseessä. Tätä seuraavat tiedonsiirtoistunnon yksilöivä UUID-tunniste ja viestin yksilöivä packetSeqNo. Kenttien header, data, status ja eot sisältö riippuu viestityypistä. Huomattakoon, että kuvassa pakolliset kentät on korostettu mustalla reunuksella, ja optionaaliset kentät puolestaan harmaalla (esim. SdbMessage-rakenteen kenttä UUID on pakollinen, kun puolestaan header-kenttä on optionaalinen).

Jos kyseessä on HEADER-tyyppinen viesti, sisältää header-kenttä SdbHeader-rakenteen, joka kuvaa alkavan tiedonsiirtoistunnon tiedot (mm. siirrettävän sisällön tyypin (contentType) ja datapakettien koon (dataPacketSize)). Jos contentType osoittaa kyseessä olevan EDF-tyyppisen tiedon siirron, sisältää edfHeader-kenttä siirrettävän EDF-tietovuon otsikkotiedot (kuva L7.2, EDFHeader-rakenne).

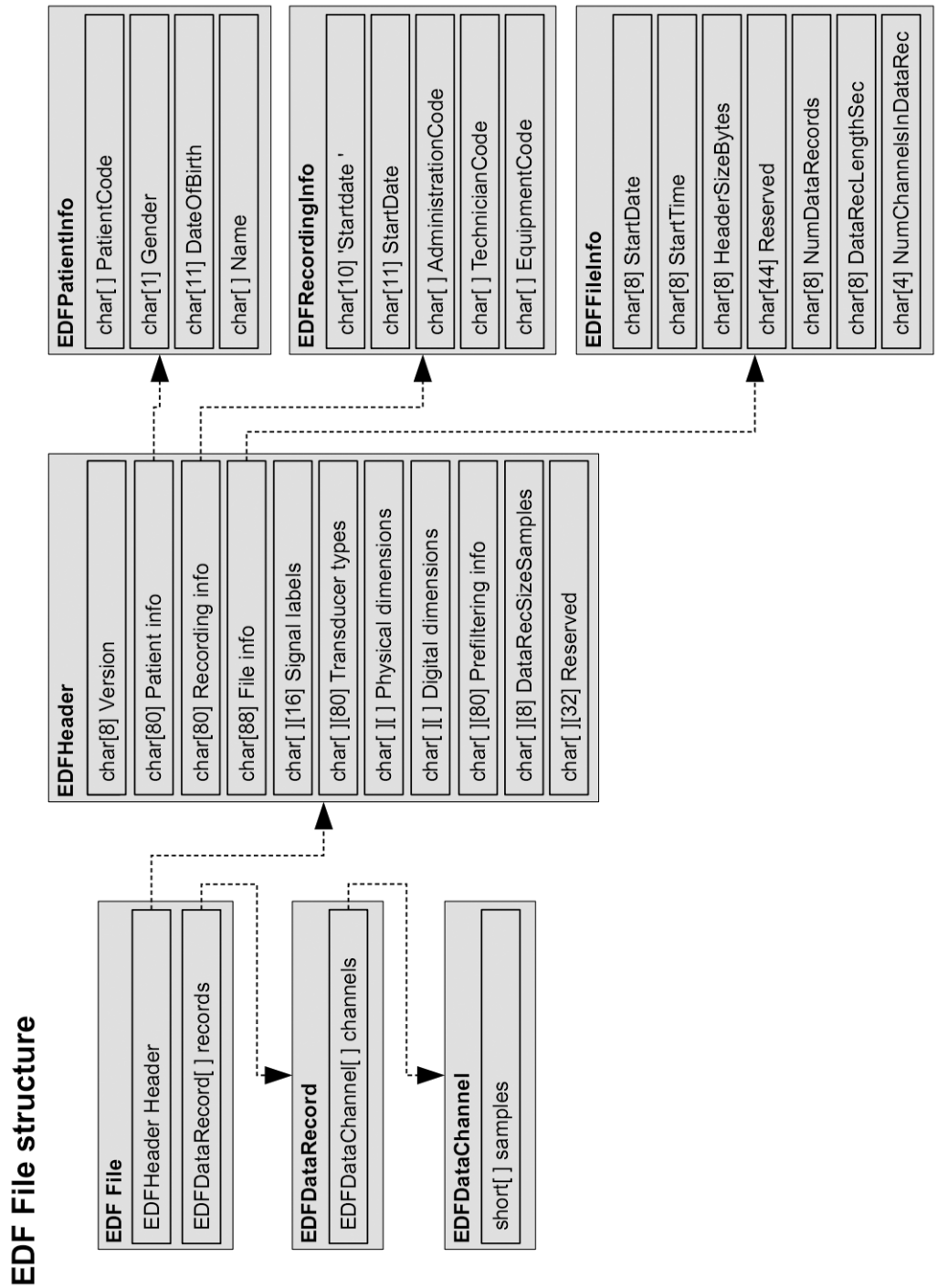
DATA-tyyppiset viestit pitävät sisällään mittaustietoa. Data-kenttä sisältää SdbData-rakenteen, joka sisältää joko tuntemattomaksi luokiteltavaa raakadataa tai EDF-formaatin mukaisen lohkon (kuva L7.2, EDFDataRecord-rakenne) – tiedonsiirtoistunnon tyypistä (contentType) riippuen.

Palvelin vastaa jokaiseen saamaansa viestiin STATUS-viestillä. Tällöin SdbMessage-rakenteen status-kenttä sisältää SdbStatus-rakenteen, joka koostuu tilakoodista (type) sekä optionaalisesta tekstikuvauksesta (message).

Kun mittaustietoa lähettävä osapuoli on saanut kaiken tiedon lähetettyä ja haluaa päättää tiedonsiirtoistunnon, se lähettää END_OF_TRANSMISSION-tyyppisen viestin. Tämä viesti ei sisällä muuta hyötyinformaatiota. Sen saatuaan palvelin vapauttaa tiedon vastaanottamiseen tarvitsemansa resurssit ja sulkee yhteyden.



Kuva L7.1. *Tiedonsiirtoprotokollan rakenne.*



Kuva L7.2. *EDF-tiedoston rakenne.*