TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALEKSI MARTIKAINEN

CANOPEN-OHJELMOINTIRAJAPINNAT

Kandidaatintyö

Tarkastaja: Sakari Lahti
Tammikuu 2018

TIIVISTELMÄ

ALEKSI MARTIKAINEN: CANopen-ohjelmointirajapinnat
Tampereen teknillinen yliopisto
Kandidaatintyö, 17 sivua, 6 liitesivua
Tammikuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Sakari Lahti

Avainsanat: CAN, CAN-väylä, CANopen, ohjelmointirajapinta

Työssä perehdyttiin yleisellä tasolla CANopeniin ja siihen tarjolla olevien ohjelmointirajapintojen käyttöön. Työn tarkoituksena oli selvittää, mitä CANopen ohjelmointirajapintoja on saatavilla ja kuinka niitä käytetään käytännössä. CANopen ohjelmointirajapintoja etsittiin google hauilla, alan kirjallisuudesta ja kysymällä alalla työskenteleviltä henkilöiltä.

Tuloksena tuotettiin tutkielma CANopenin keskeisistä ominaisuuksista ja löydettiin neljä CANopen-ohjelmointirajapintaa. Näistä rajapinnoista esiteltiin esimerkkikoodia, ohjelmointirajapintojen käyttöä ja ominaisuuksia vertailtiin päällisin puolin. Vertailun tarkoituksena on tarjota nopea katsaus ohjelmointirajapintojen syntaksiin ja auttaa pääsemään alkuun kunkin rajapinnan käytössä. Työn tarkoituksena on nopeuttaa ja helpottaa ohjelmistokehityksen aloittamista kyseisellä ohjelmointirajapinnalla ja yleisesti auttaa CANopeniin perehtymisessä.

Englanninkielinen nimi: CANopen APIs

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. CAN-VÄYLÄ 2
3. CANOPEN 5
   3.1 Yleistä tietoa 6
   3.2 Viestin tunnisteosa (COB-ID) 8
   3.3 Objektikirjastot 8
   3.4 Network Management (NMT) protokolla 9
   3.5 Tiedonsiirtoprotokollat SDO ja PDO 11
   3.6 SYNC, TIME ja EMCY protokollat 11
   3.7 Electronic Data Sheet (EDS) 12
4. OHJELMOINTIRAJAPINNAT 13
   4.1 CANopenNode/CANOpenSocket 13
   4.2 CanFestival 14
   4.3 CANopen for Python 15
   4.4 Bosch Rexrothin BODAS-ohjainten rajapinta 16
5. YHTEENVETO 17
LÄHTEET 18

LIITE A: OSA KUVITTEELLISESTA EDS TIEDOSTOSTA

LIITE B: SUOMENNETTU CANOPENNODE ALOITUSOPAS

LIITE C: KOODIESIMERKKI BOSCHIN RAJAPINNASTA

LIITE D: CANOPEN FOR PYTHON KOODIESIMERKKI

LYHENTEET JA MERKINNÄT

API	Application Program Interface, ohjelmointirajapinta
C	Ohjelmointikieli. Uusin versio vuodelta 2011, C11
CAN	Controller Area Network, väylätekniikka
CANopen	CiA:n kehittämä ja ylläpitämä CAN-väylän korkeamman tason protokolla
CiA	CAN in Automation, CAN-käyttäjien muodostama yhdistys, joka vastaa CANopenin kehityksestä.
COB-ID	Communication Object Identifier, CANopen viestikehyksen tunnisteosa
EDS	Electronic Data Sheet, tiedostomuoto, joka kuvaa CANopen laitteen objektikirjaston
EMCY	Emergency Object, CANopen aliprotokolla
HAL	Hardware Abstraction Layer
NMT	Network Management, CANopen aliprotokolla
OSI-malli	Open Systems Interconnection Reference Model. Tiedonsiirtoprotokollien muodostama malli.
PDO	Process Data Object, CANopen aliprotokolla
Python	Ohjelmointikieli. Tässä työssä viitataan aina Python 3 versioon
SAE J1939	SAE:n kehittämä ja ylläpitämä CAN-väylän korkeamman tason protokolla
SAE	Society of Automotive Engineers
SDO	Service Data Object, CANopen aliprotokolla
Solmu	Node, CAN-verkkoon kytketty laite
SYNC	Synchronization Object, CANopen aliprotokolla
Tiedonsiirtonopeus	Nopeus jolla itse hyötykuormaa siirretään väylällä jättäen huomioimatta CAN-viestikehyksen muut osat kuten viestin prioriteetti.
TIME	Time Stamp Object, CANopen aliprotokolla
Väylä	CAN-väylä

1. JOHDANTO

Nykyajan tekniikka sisältää koko ajan entistä enemmän elektroniikkaa. Esimerkiksi autoissa tämä näkyy jatkuvasti lisääntyvinä elektronisina järjestelminä kuljettajan kojelaudalla. Näiden elektronisten järjestelmien on kyettävä kommunikoimaan keskenään, esimerkiksi renkaiden paineantureiden mittausdata on oltava kuljettajan saatavilla kojelaudalla. Autoala kehitti 70- ja 80-luvuilla useita erilaisia väyläratkaisuja tämän ongelman ratkaisemiseen. Nämä järjestelmät eivät kuitenkaan olleet keskenään yhteensopivia. Yhteensopivuuden parantamiseksi Robert Bosch GmbH kehitti niin kutsutun CAN-väylän, jonka tarkoituksena oli tarjota standardoitu ratkaisu ajoneuvon sisäisten järjestelmien keskinäiseen viestintään. [1, 2]

CAN-väylän määrittelydokumentit jättivät monia käytännön toteutuksen yksityiskohtia kuitenkin avoimiksi, eivätkä tarjonneet korkeamman tason ominaisuuksia. Tästä syystä CAN:iin pohjautuvia protokollia kehitettiin standardisoimaan CAN-viestiliikennettä. Yksi näistä protokollista on CANopen. [1]

CANopen tarjoaa useita korkeamman tason ominaisuuksia ja aliprotokollia väylälle liitetyille laitteille. CANopen ei kuitenkaan ota kantaa käytännön ohjelmointitoteutuksiin eikä tarjoa valmista ohjelmointirajapintaa. CANopeniin on kuitenkin toteutettu muutamia avoimen lähdekoodin rajapintoja. Lisäksi CANopenia hyödyntävillä yrityksillä on omia toteutuksiaan CANopenin ohjelmointirajapinnoista. [3]

Tämän työn tarkoituksena on selvittää, mitä ohjelmointirajapintoja CANopeniin on saatavilla ja kuinka CANopenin aliprotokollia käytetään niiden avulla. Aiheesta löytyy hyvin vähän käytännön ohjeistusta, joten tämän työn on tarkoitus osaltaan vastata tähän ongelmaan. Tämän työn avulla aloittelijan on helpompi päästä alkuun CANopenin kanssa.

Luvussa 2 käsitellään CAN-väylää yleisesti. CAN-väylästä kerrotaan muun muassa missä kaikkialla sitä yleisesti käytetään ja arvioidaan väylän luotettavuutta ja viiveitä. Luvussa 3 käsitellään CANopen protokollaa. Protokollan tärkeimmät korkeamman tason ominaisuudet ja aliprotokollat kuvataan luvun aliluvuissa. Luvussa 4 päästään CANopenin ohjelmointirajapintoihin. Käsiteltävät ohjelmointirajapinnat ovat kukin omina alilukuinaan. Aliluvissa käsitellään rajapintojen ominaisuuksia lyhyiden koodiesimerkkien avulla. Viimeisessä luvussa tehdään lyhyt yhteenveto löydetyistä rajapinnoista. Työn liitteistä löytyy laajempia koodiesimerkkejä ohjelmointirajapinnoista.

2. CAN-VÄYLÄ

CAN-väylä (engl. Controller Area Network bus) on automaatioväylä, jota käytetään muun muassa ajoneuvoissa, sairaalalaitteissa ja teollisuuslaitteissa niiden alijärjestelmien välisen kommunikaation mahdollistamiseksi. Esimerkiksi autossa jarrut voivat lähettää virheviestin ajotietokoneelle CAN-väylää pitkin ja ajotietokone voi varoittaa kuljettajaa kojelaudan merkkivalolla. CAN-väylä on toimintavarma ja rakenteeltaan yksinkertainen. [2, 4]

CAN-väylän suunnittelu alkoi vuonna 1983, kun Robert Bosch GmbH ryhtyi kehittämään uutta väyläjärjestelmää henkilöautoihin. Helmikuussa 1986 CAN-väylä esiteltiin SAE(Society of Automotive Engineers)-kongressissa. Vaikka CAN-väylä suunniteltiin alun perin käytettäväksi henkilöautoissa, ryhdyttiin sitä aluksi soveltamaan muilla aloilla, kuten hisseissä, sairaalalaitteissa ja tekstiiliteollisuudessa. Suomalainen hissivalmistaja Kone oli yksi ensimmäisistä CAN-väylän soveltajista. Ensimmäiset henkilöautot, jotka käyttivät CAN-väylää, valmistuivat vasta vuonna 1992. CAN-väylä nousi muutamassa vuosikymmenessä yhdeksi merkittävimmistä väylätekniikoista. [1]

CAN-väylän tiedonsiirtonopeudet vaihtelevat 1000 kb/s – 10 kb/s välillä. Hitaammat tiedonsiirtonopeudet ovat virhesietoisempia ja mahdollistavat fyysisiltä mitoiltaan pidemmän väylän. CAN-väylän maksimipituus riippuu valitusta tiedonsiirtonopeudesta: mitä suurempi tiedonsiirtonopeus, sitä lyhyempi väylän maksimipituus on. Väylän maksimipituus määrittyy väylällä aiheutuvista viiveistä. Jos väylän pituus kasvaa liian suureksi, tieto jännitteen muuttumisesta ei ennätä koko väylälle ennen kuin jokin päätelaitteista yrittää muuttaa jännitettä taas uudelleen. Oheinen taulukko 1 kuvaa tyypillisiä CAN-väylällä käytettyjä bittinopeuksia (bit rate) ja niitä vastaavia väylän maksimipituuksia. [2]

Taulukko 1: *CAN-väylän bittinopeuden yhteys väylänpituuteen.*

Bittinopeus (kb/s)	**Väylänpituus (m)**
1000	30
800	50
500	100
250	250
125	500
62,5	1000
20	2500
10	5000

Väylän bittinopeus vaikuttaa myös väylällä esiintyviin viiveisiin. Korkeampi bittinopeus madaltaa viestien lähettämiseen kuluvaa aikaa. Oheinen taulukko 2 kuvaa bittinopeuden yhteyttä viestin lähetysaikaan ja väylän teoreettista tiedonsiirtomaksimia. Viestin lähettämiseen kuluvan ajan lisäksi väylälle voi syntyä muita huomattavia viiveitä, jos viesti turmeltuu ja vaatii uudelleenlähetyksen tai väylä on kuormittunut korkeamman prioriteetin viesteillä. Lisäksi vastaanotetun viestin käsitteleminen vie aikaa. Nämä viiveet on otettava huomioon kriittisten järjestelmien suunnittelussa. [2]

Taulukko 2: *CAN-väylän bittinopeuden vaikutus viestien kestoon ja tiedonsiirtonopeuteen 29-bittisellä ID tunnisteella. [5]*

Bittinopeus (kb/s)	Bittiaika (µs)	8-tavuisen viestin pisin mahdollinen viestiaika (µs)	tiedonsiirtonopeusmaksimi (kbps)
1000	1	152	496,12
800	1,25	190	396,90
500	2	304	248,06
250	4	608	124,03
125	8	1216	62,02
62,5	16	2432	31,01
20	50	7600	9,92
10	100	15200	4,96

Myös CAN-väylälaitteen sisäisistä komponenteista tulee viivettä komponenttien laadusta riippuen 210 – 452 µs. Lisäksi jokainen metri väylällä lisää viivettä 5 ns/m. [2]

CAN-väylää käytetään erityisesti järjestelmissä, joiden virheellinen toiminta johtaisi vaaratilanteisiin. Esimerkiksi autoissa järjestelmien on ehdottomasti toimittava kaikissa tilanteissa turvallisuuden takaamiseksi.

CAN-väylän toimintavarmuus on saavutettu hyvällä suunnittelulla. CAN-väylä on parikaapelina fyysiseltä rakenteeltaan hyvin yksinkertainen. Parikaapelissa johtimet kumoavat keskenään tehokkaasti ympäröivää sähkömagneettista säteilyä. Lisäksi CAN-väylällä tulkitaan vain parikaapelin johtimien välistä jännite-eroa, joten edes suuret indusoituneet virrat eivät juurikaan häiritse CAN-väylää induktion tapahtuessa yleensä tasaisesti molempien kaapelien välille. CAN-väylän protokollat myös aktiivisesti tarkistavat viestien oikeellisen siirtymisen ja tarvittaessa korjaavat havaitut bittivirheet. [2]

CAN-väylän luotettavuudesta on vain muutamia tutkimuksia vapaasti saatavilla. Kaikista vapaasti saatavilla olevista tutkimuksista vain yksi [6] mittasi suoraan virheiden määrää. Tutkimuksen kokeessa käytettiin 1 Mb/s:n tiedonsiirtonopeutta ja 30 metriä pitkää kaapelia. Koe toistettiin kolmessa eri ympäristössä: yliopiston laboratoriossa, tehtaan tuotantolinjalla ja kahden metrin päässä korkeataajuuksisesta hitsikoneesta. Tutkimuksen tulokset ovat taulukossa 3. [2]

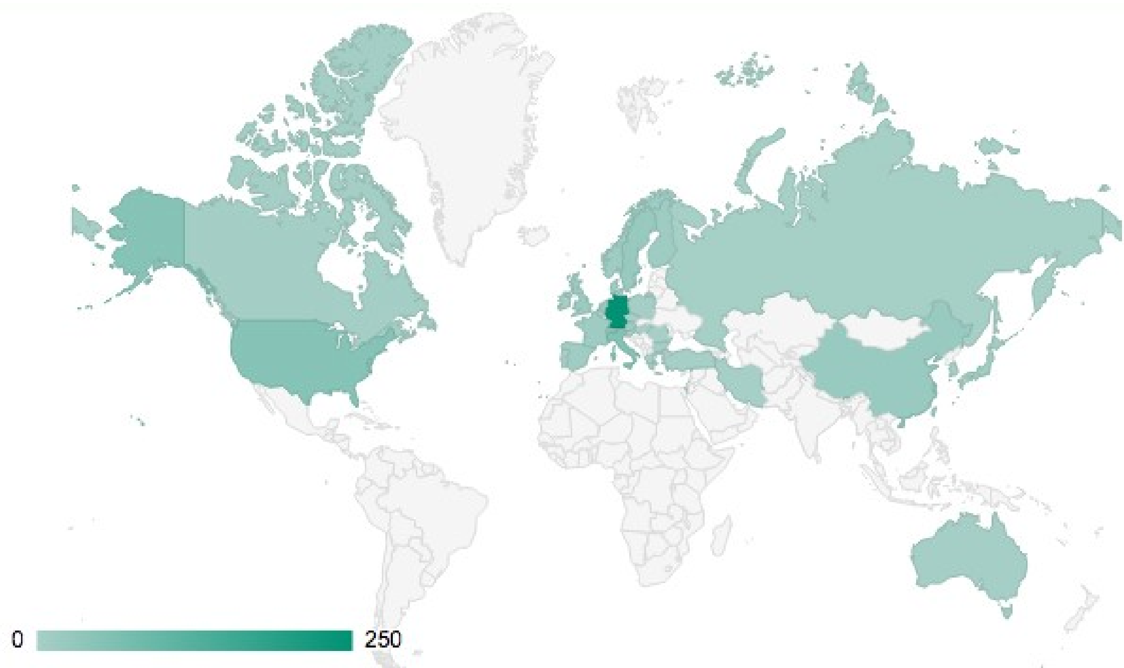
Taulukko 3: *Tutkimuksessa [6] havaittujen bittivirheiden määrä.*

	Yliopiston laboratorio	**Tehtaan tuotantolinja**	**Hitsauskoneen vieressä**
Lähetetyt bitit	$2,02 * 10^{11}$	$1,98 * 10^{11}$	$9,79 * 10^{10}$
Bittivirheet	6	609	25 239
Bittivirheet / lähetetyt bitit	$3 * 10^{-11}$	$3,1 * 10^{-9}$	$2,6 * 10^{-7}$

Taulukon 3 virheet ovat kuitenkin vasta vaihtuneiden bittien määrä jotka eivät johda aina virheisiin. CAN-väylän virheentarkastusprotokollat kykenevät havaitsemaan suurimman osan vaihtuneista biteistä ja korjaamaan tilanteen esimerkiksi lähettämällä viestin uudelleen. CAN-väylän virheentarkastusprotokolla havaitsee kaikki yhden bitin virheet yksittäisessä viestikehyksessä ja suuren osan useamman vaihtuneen bitin tapauksista. Virhe pääsee virheentarkastusprotokollan ohi todennäköisimmin silloin, kun vaihtuneiden bittien määrä on tasan kuusi bittiä. Tällöin todennäköisyys sille, että virhettä ei havaita ollenkaan, on $1,4 * 10^{-6}$. Näiden lukujen perusteella Di Natale et al. arvioivat, että Yhdysvaltojen koko autokannalla havaitsemattomia CAN-väyläviestejä tapahtuisi noin 0,05 viestiä päivittäin; yksi virheellinen havaitsematon väyläviesti pääsee virheentarkastusprotokollan läpi keskimäärin joka kahdeskymmenes päivä. [2] Luku ei ole järin suuri, kun se suhteutetaan siihen, että USA:ssa on 250 miljoonaa rekisteröityä moottoroitua ajoneuvoja, joista valtaosassa on CAN-väylä. [2, 7]

3. CANOPEN

CAN-väylän määritelmät ja standardit jättävät monia protokollien yksityiskohtia määrittelemättä. CAN-väylä ei myöskään tarjonnut mitään korkeamman tason ominaisuuksia. CAN-väylän standardi tarjoaa vain siirtokerroksen, joka jättää sovellustasolle paljon vastuuta. Jokainen CAN-väylää käyttävä yritys joutui kehittämään omat ratkaisunsa. Tämä johti lopulta CANopen-nimisen protokollan kehittämiseen. Protokollaa kehittämään perustettiin CAN-väylän käyttäjien muodostama yhdistys CiA (CAN in Automation). CiA:n päämaja sijaitsee Saksassa Nürnbergissä, ja CiA:n jäsenistä 45 % on saksalaisia. Kuva 3.1 esittää CiA:n jäsenten jakautumisen maailmassa. [1, 4]

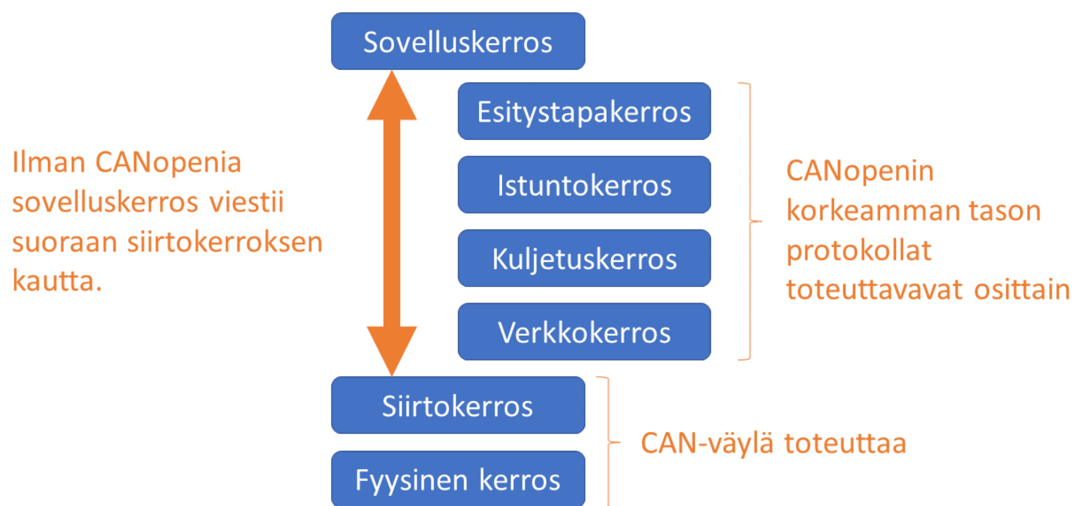


Kuva 3.1: *CiA:n jäsenten jakautuminen maailmankartalle.[1]*

CiA julkaisi CANopenin ensimmäisen version vuonna 1995, ja uusia versioita on julkaistu tasaisesti vuosien varrella. CANopenia kehitetään edelleen, ja siitä on useita eri aliversioita eri tarkoituksiin. Tällä hetkellä työn alla on myös CANopen FD, joka mahdollistaa aiempaa suuremman tiedonsiirtonopeuden. [1] 90-luvulla kehitettiin myös muita CAN-väylään pohjautuvia korkeamman tason protokollia, kuten DeviceNET, CAN Kingdom, Smart Distributed System ja SAE J1939. [2, 3]

3.1 Yleistä tietoa

Tiedonsiirtoa ja siihen käytettäviä protokollia kuvataan usein OSI-mallilla (Open Systems Interconnection Reference Model). OSI-malli kuvaa tietoverkkokommunikaation seitsemänä tasona. OSI-mallin seitsemän tasoa ovat fyysinen kerros, siirtokerros, verkkokerros, kuljetuskerros, istuntokerros, esitystapakerros ja sovelluskerros. [8] Jos CANopenia verrataan OSI-malliin, toteuttaa se ainakin osittain esitystapa-, istunto-, kuljetus- ja verkkokerrokset. [9, 10] Alla oleva Kuva 3.2 esittää CANopenin asettumisen OSI-malliin.



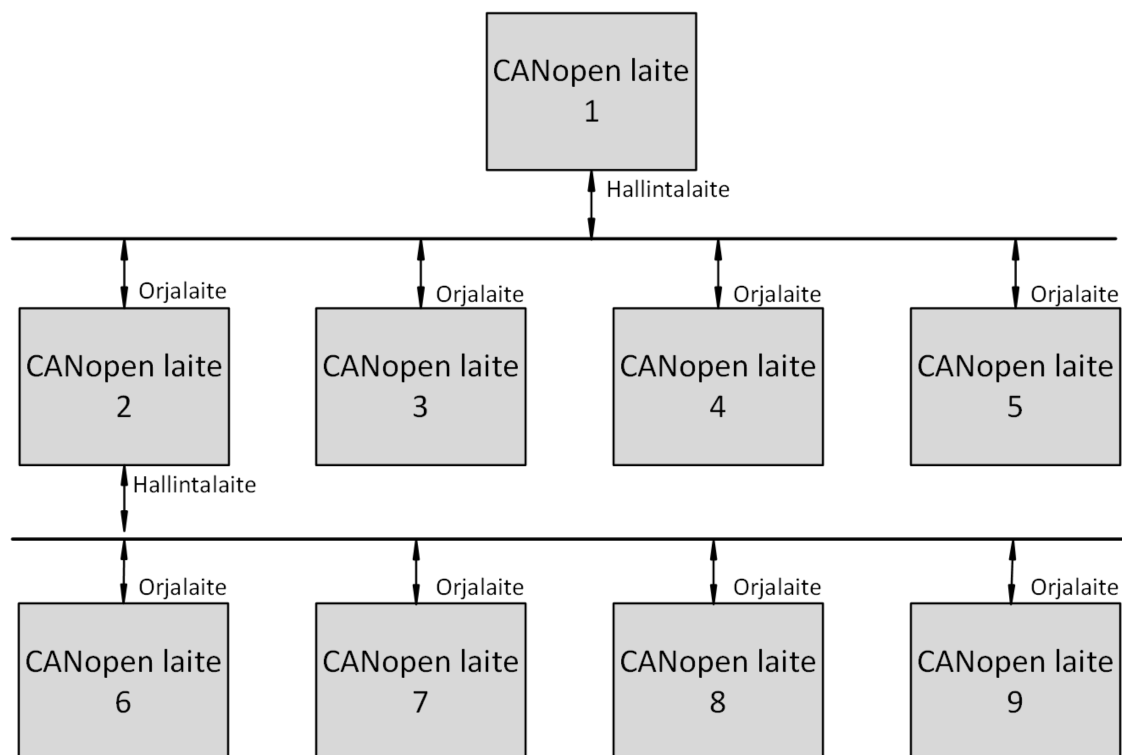
Kuva 3.2: *CAN- ja CANopenin sijoittuminen OSI-malliin. [9, 10]*

CANopen on nimensä mukaisesti avoin. CiA:n nettisivuilla on ilmaiseksi saatavilla määritelmät, suositukset, ohjeistukset ja tekniset dokumentit CANopenista. Dokumenttien lataaminen vaatii ilmaisen rekisteröitymisen joka ei velvoita mihinkään. Joitakin CiA:n nettisivuilla olevia dokumentteja tarjotaan vain CiA:n jäsenille, kuten esimerkiksi keskeneräiset määritelmädokumentit. [1]

Vaikka CANopen on kaikille ilmaiseksi avoin, se ei ole helposti lähestyttävä. CANopenin dokumentaatio on aloittelijalle vaikeaselkoinen ja vaatii pitkällistä perehtymistä. CANopenista löytyy niukasti käytännön ohjeistusta ja opetusmateriaalia ilmaiseksi verrattuna yleisisiin ohjelmointikieliin kuten C++ ja Python. CiA ei myöskään tarjoa minkäänlaista CANopen ohjelmointirajapintaa. Nämä seikat tekevät CANopenin itseopiskelusta vaikeaa. Harrastelijat ovatkin törmänneet ongelmiin kohdatessaan CANopenin[11, 12].

CANopeniin on kuitenkin saatavilla harrastelijoiden tekemiä avoimen lähdekoodin ohjelmointirajapintoja. Näihin rajapintoihin liittyy kuitenkin rajoitteita esimerkiksi ajureiden suhteen. Näitä avoimen lähdekoodin rajapintoja käsitellään tarkemmin luvussa neljä.

CANopen ei ole yksittäinen protokolla, vaan TCP/IP:n tavoin muodostuu useista protokollista muodostaen protokollaperheen. Tässä työssä tietoverkon hallintaisännällä tarkoitetaan CANopen-manager laitetta, joka hallitsee orjalaitteita NMT ja SDO protokollilla. Edellä mainituista protokollista kerrotaan tarkemmin myöhemmin. CANopenin perusajatuksena on, että väylällä on aina yksi tietoverkon hallintaisäntä (NMT Master, Network Management Master) joka hallitsee verkon muita laitteita. Muita laitteita kutsutaan orjalaitteiksi. Kuitenkin poikkeustapauksissa hallintaisäntää ei tarvita. Kuva 3.3 esittää esimerkin mahdollisesta CANopen verkon rakenteesta, jossa kaksi erillistä CANopen verkkoa on yhdistetty yhdeksi suuremmaksi kokonaisuudeksi yhdistämällä yksi väylän laitteista kahdelle väylälle. [1, 3, 4]



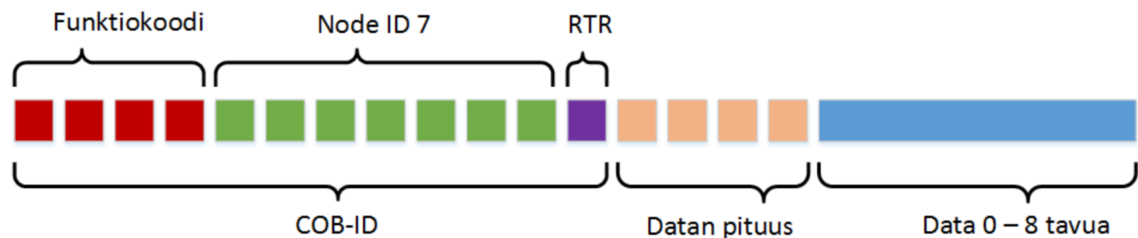
Kuva 3.3: *Esimerkki mahdollisesta CANopen verkon rakenteesta. Laite 2 toimii sekä orjalaitteena, että hallintalaitteena eri verkoille. Laite 1 hallitsee koko verkkoa laitteen 2 kautta.*

Kuva 3.3 esittää esimerkin, minkälaisilla ratkaisuilla CANopenista saadaan hyvin skaalautuvan ja joustavan. Verkko tukee tarvittaessa jopa sadoista väylistä ja tuhansista laitteista koostuvia järjestelmiä. [13]

CANopeniin sisältyy seuraavia ominaisuuksia ja aliprotokollia: objektikirjastot, COB-ID, NMT-, SDO-, PDO-, SYNC-, TIME-, ja EMCY-protokollat ja EDS:ät. Nämä ominaisuudet on esitelty seuraavissa aliluvuissa tarkemmin. [1]

3.2 Viestin tunnisteosa (COB-ID)

Viestin tunnisteosa eli COB-ID (Communication Object Identifier) kertoo, kuka kyseisen viestin on lähettänyt ja mitä tyyppiä kyseinen viesti on. Viestin tunnisteosa on standardin mukaan aina 11- tai 29-bittiä pitkä. 11-bitin tunnisteessa 4 eniten merkitsevää bittiä kertovat viestin tyypin ja loput 7-bittiä kertovat lähettäjän ID:n. ID:tä kutsutaan myös solmunumeroksi. [4, 14] Kuva 3.4 havainnollistaa CANopen viestikehyksen rakennetta



Kuva 3.4: *CANopen viestikehys*

Koska COB-ID:ssä olevat 7-bittiä on varattu solmunumerolle, on CANopen-laitteiden maksimimäärä yhdessä verkossa 127 (2^7). Verkkoon voidaan kuitenkin liittää useampia laitteita esimerkiksi kuvan Kuva 3.3 tavalla, jossa laite 2 on yhdistettynä useaan verkkoon. Laite 2 näkyy kuvan ylemmälle väylälle vain yhtenä laitteena, mutta se voi välittää myös laitteiden 6 – 9 tietoja. Kuvan 3.3 mukaisen järjestelyn lisäksi on mahdollista käyttää 29-bitin mittaista viestin tunnisteosaa jolloin mahdollisten laitteiden lukumäärä yhdessä verkossa kasvaa huomattavasti. [4]

3.3 Objektikirjastot

Jokaisella CANopen laitteella on niin sanottu objektikirjasto (Object Library). Objektikirjasto on laitteen tietorakenne, jonne tallennetaan laitteen tiedot ja asetukset. Objektikirjasto toimii rajapintana laitteen oman sovelluskerroksen ja väylän välillä. Jos laite on esimerkiksi jonkinlainen anturi, tallentuu anturidata objektikirjastoon. Objektikirjastosta tiedot voidaan lähettää väylälle muiden laitteiden käytettäväksi. CANopen laitteiden välistä kommunikaatiota käsitellään tarkemmin myöhemmissä luvuissa. Väylän hallintaisäntä myös asettaa laitteen asetukset tämän objektikirjastoon silloin, kun väylä ja sille liitetyt laitteet alustetaan. [14]

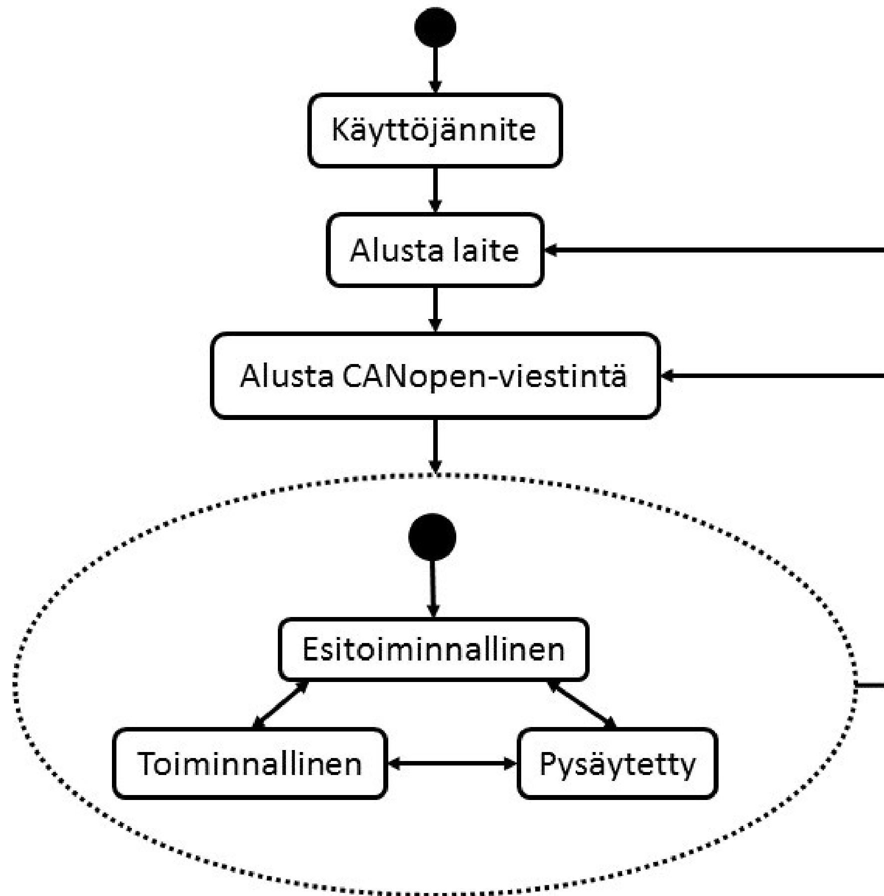
Taulukko 4: *Objektikirjaston sisältö. [14]*

Indeksialue	Sisältö
0x0000	Ei käytössä
0x0001-0x001F	Staattiset tietotyypit
0x0020-0x003F	Moniosaiset tietotyypit
0x0040-0x005F	Valmistajakohtaiset moniosaiset tietotyypit
0x0060-0x025F	Laiteprofiilikohtaiset tietotyypit
0x0260-0x03FF	Varattu
0x0400-0x0FFF	Varattu
0x1000-0x1FFF	Kommunikaatioprofiilin alue
0x2000-0x5FFF	Valmistajakohtainen alue
0x6000-0x67FF	Standardisoidun profiilin alue, 1. looginen laite
0x6800-0x6FFF	Standardisoidun profiilin alue, 2. looginen laite
0x7000-0x77FF	Standardisoidun profiilin alue, 3. looginen laite
0x7800-0x7FFF	Standardisoidun profiilin alue, 4. looginen laite
0x8000-0x87FF	Standardisoidun profiilin alue, 5. looginen laite
0x8800-0x8FFF	Standardisoidun profiilin alue, 6. looginen laite
0x9000-0x97FF	Standardisoidun profiilin alue, 7. looginen laite
0x9800-0x9FFF	Standardisoidun profiilin alue, 8. looginen laite
0xA000-0xAFFF	Standardisoidun verkon muuttujien alue
0xB000-0xBFFF	Standardisoidun järjestelmämuuttujien alue
0xC000-0xFFFF	Varattu

Objektikirjaston alussa välillä 0x001-0x025F määritellään käytössä olevat tietotyypit, kuten totuusarvomuuttuja, kokonaisluku, liukuluku ja merkkijono. Nämä tietotyypit ovat kaikille CANopen laitteille yhteisiä. Valmistajakohtaiset moniosaiset tietotyypit ovat laitekohtaisia yhdistelmiä aiemmin mainituista tietotyypeistä. [14]

3.4 Network Management (NMT) protokolla

CANopen väylällä on niin sanottu hallintaisäntä, joka hallitsee muiden laitteiden tiloja NMT-protokollalla (Network Management). Verkon käynnistyessä hallintaisäntä alustaa väylän ja lähettää väylälle liitetyille laitteiden käskyn siirtyä toiminnalliseen tilaan. Hallintaisällä on myös oikeudet tarkkailla kaikkia muita verkon laitteita ja niiden tilaan, sekä tarvittaessa esimerkiksi oikeudet kytkeä orjalaitteita pois käytöstä esimerkiksi kriittisen virheen sattuessa. [13] Kuva 3.5 esittää CANopen laitteen sisäistä tilakonetta. CANopen orjalaite vaihtaa tilakoneensa tiloja hallintaisännän käskyjen perusteella.



Kuva 3.5: *CANopen laitteen tilakone. Kun laitteeseen kytketään virrat päälle, siirtyy se alustusten kautta esitoiminnalliseen tilaan. Toiminnallinen tila on laitteen normaalitila. [1, 13]*

Kun laitteeseen kytketään virrat, laite alustaa itsensä ja CANopen rajapintansa itsenäisesti. Tämän jälkeen laite lähettää verkon hallintaisännälle tiedon, että alustus on valmis ja jää odottamaan esitoiminnalliseen tilaan. Esitoiminnallisessa tilassa laite odottaa verkonhallintaisännältä käskyä siirtyä toiminnalliseen tilaan. Ollessaan esitoiminnallisessa, toiminnallisessa tai pysäytetyssä tilassa verkon hallintaisäntä voi käskeä laitetta siirtymään mihin tahansa tilaan. [13]

Perinteisesti CANopen verkko vaatii aina yhden ja vain yhden hallintaisännän joka hallinnoi muiden laitteiden tiloja. Poikkeustilanteissa verkon hallintaisäntää ei kuitenkaan tarvita. Jos verkon kaikki solmut siirtyvät itsenäisesti toiminnalliseen tilaan ja niiden PDO-viestintä on määritetty etukäteen, ei hallintaisäntää tarvita. [15]

3.5 Tiedonsiirtoprotokollat SDO ja PDO

SDO:n (service data object) avulla voidaan lukea ja kirjoittaa solmun objektikirjastoon. SDO-viesteillä hallintaisäntä voi suoraan kirjoittaa orjalaitteen objektikirjastossa oleviin asetusparametreihin. SDO-protokollan viesteillä voidaan siirtää myös mittaus- ja ohjausdataa. Tätä ei kuitenkaan suositella, koska SDO-viestit kuormittavat väylää. SDO-protokollassa palvelimena toimiva laite kuittaa aina viestin vastaanotetuksi, josta syntyy ylimääräistä väylää kuormittavaa viestiliikennettä. Mittaus- ja ohjausdataan käytetään yleensä PDO-viestejä, jotka on esitelty seuraavassa luvussa. Oletusarvoisesti vain verkon hallintaisäntä voi aloittaa viestinnän SDO-protokollalla orjien vain vastatessa hallintaisännän SDO-viesteihin. Useampia SDO-isäntälaitteita voidaan asettaa väylälle, mutta tämä laskee väylälaitteiden enimmäismäärää. Enimmäislaitteiden määrä puolittuu SDO-isäntälaitteiden määrän kaksinkertaistuessa. [2]

Reaaliaikainen tiedonsiirto suoritetaan yleensä PDO-protokollalla (process data object). Lähetettäviä ja vastaanotettavia PDO-viestejä säädetään niin sanotulla PDO-mappingilla. PDO-mappingilla määritetään, mitä objektikirjaston viestejä lähetetään (1-8 tavua eli 64 bittiä dataa) ja milloin. PDO-viestejä voidaan lähettää esimerkiksi tasavälein 50 ms välein, tai vaikkapa vain jonkin tietyn tapahtuman ilmetessä. PDO-viestit soveltuvat siis nimenomaan esimerkiksi säännölliseen anturidatan lähettämiseen. PDO-viestit jaetaan TPDO ja RPDO viesteihin (lähetys ja vastaanotto). Oletuksena on määritelty 4 TPDO ja 4 RPDO viestiä, mutta manuaalisesti säätämällä jopa 512 PDO:ta on mahdollista asettaa päätelaitteeseen. Joillekin CANopen laitteille ei ole tarjolla minkäänlaista PDO-mappingia vaan laitteisiin on kovakoodattu etukäteen TPDO ja RPDO viestit eikä niitä ole mahdollista muokata. [16]

3.6 SYNC, TIME ja EMCY protokollat

SYNC-protokollaa (syncronization object protocol) käytetään nimensä mukaisesti solmujen väliseen synkronointiin. Esimerkiksi PDO-viestien lähetys voidaan ajastaa SYNC-viestien kanssa. Näin voidaan hallita tarkemmin muun muassa järjestystä, jossa solmut lähettävät tietoa väylälle. SYNC-tuottaja lähettää synkronointiobjektin määräajoin. Aika SYNC-viestien välillä määritellään mikrosekunteina objektikirjaston objektissa 0x1006. Jos arvo on nolla, synkronisointi ei ole käytössä ja SYNC-viestejä ei lähetetä. [9, 14]

TIME-protokolla välittää aikatietoa. Väylässä voi olla vain yksi TIME-viestin lähettäjä. Viestin datana kulkee 6-tavuinen aikatieto. Neljä ensimmäistä tavua kertovat kuluneen ajan keskiyöstä ja kaksi jälkimmäistä tavua kertovat kuluneiden päivien määrän tammikuun ensimmäisestä päivästä 1984. [14]

EMCY protokollalla solmut voivat lähettää virheilmoituksia sisäisistä virheistään ja ilmoituksia virhetilojen poistumisesta. Lähettäjänä on solmu, jossa virhe tapahtui. Solmujen reagointia viestiin ei ole määritelty CANopen protokollassa. Virhetilaan siirtymisestä ilmoitetaan vain kerran ja virhetilan poistuessa lähetetään uusi EMCY-viesti. EMCY-protokolla on valinnainen ominaisuus CANopen standardissa, eli laitevalmistaja voi itse päättää, toteutetaanko ominaisuutta. Näin ollen CANopen ei ota kantaa, mitä solmussa tulisi tapahtua EMCY-viestin saapuessa. [1, 9]

3.7 Electronic Data Sheet (EDS)

EDS on ini-tiedoston mallinen tietolehti, joka kuvailee CANopen laitteen objektikirjaston. EDS-kuvaa siis laitteen koko rajapinnan CANopen-verkon kanssa. EDS-datalehteen on kirjattu muun muassa tiedot, minne objektikirjastossa orjalaitteen anturidatan tiedot kerätään. Näin EDS:ää tutkimalla voidaan esimerkiksi verkon hallintaisäntä asettaa muokkaamaan haluttuja tietoja verkkoon liitetystä orjalaitteesta. EDS tiedostot on tarkoitus tulevaisuudessa muuttaa .xml muotoisiksi. [14]

Liitteessä A on katkelma kuvitteellisesta EDS tiedostosta. Liitteessä on vain pieni osa EDS tiedostosta, koska EDS tiedostot voivat olla pituudeltaan yli kymmenentuhatta riviä pitkiä, sillä ne kuvaavat solmun koko objektikirjaston jokaisen muistiosoitteen.

4. OHJELMOINTIRAJAPINNAT

Tässä työssä keskitytttiin tutkimaan neljää mielivaltaisesti valittua ohjelmointirajapintaa. Valintaan vaikutti kuitenkin rajapintojen saatavuus. Kaikki rajapinnat Boschin rajapintaa lukuun ottamatta ovat avointa lähdekoodia ja saatavilla ilmaiseksi esimerkiksi githubista. CANopeniin löytyy rajapintatoteutuksia, mutta monet niistä ovat maksullisia tai salaisia. Harrastelijat ovat toteuttaneet itse muutamia avoimen lähdekoodin ohjelmointirajapintoja. [3, 4]

Seuraavissa alaluvuissa on tarkemmin esiteltynä neljä rajapintaa CANopen for python, CanFestival, CANopenNode ja Boschin rajapinta RC-30-sarjan ohjaimille. Rajapintojen esittelyt sisältävät lyhyen esimerkin jostain yksinkertaisesta operaatiosta, esimerkiksi heartbeat-viestin lähettämistaajuuden asettaminen SDO-protokollalla.

4.1 CANopenNode/CANOpenSocket

CANopenNode on ilmainen ja avoimen lähdekoodin CANopen protokollapino. CANopenNode on julkaistu vähän muokatun GNU v2 lisenssin. CANopenNode on kirjoitettu ANSI C:llä oliopohjaisesti, joten se on mahdollista kääntää mille vain alustalle jolle on olemassa C-kääntäjä. CANopenNode on kuitenkin suunniteltu toimimaan tietyillä mikrokontrollereilla, Linuxilla ja reaaliaikaisissa käyttöjärjestelmissä. Tuetuille mikrokontrollereille tarjotaan tarvittavat ajurit. CANopen toteuttaa yksinkertaisen NMT-isännän, PDO, SDO, EMCY ja SYNC protokollat sekä muita ominaisuuksia. Objektikirjasto on käytettävissä sekä C-koodista, että CAN-verkon kautta. CANOpenSocket on Linux versio CANopenNodesta.

Liitteestä B löytyy suomennettu versio CANopenSocketin verkkosivuilta löytyvästä ohjeistuksesta, joka on kirjoitettu auttamaan uusia käyttäjiä pääsemään alkuun CANopenSocketin kanssa. Ohjeistus kuvaa vaihe vaiheelta väylän ja laitteiden alustuksen ja yksinkertaisen SDO-viestin lähettämisen orjalaitteelle Linuxin komentoriviltä.

CANopenSockettia voidaan käyttää suoraan Linuxin terminaalista. Tällöin Linux-tietokone toimii verkon hallintaisäntänä. CANopenSockettia ei ole kuitenkaan pakko käyttää Linux-terminaalista. Ohjelma 1 kuvaa käytännön esimerkin CANopenSocketin käytöstä Linuxin komentoriviltä.

```
$ ./canopencomm [1] 4 read  0x1017 0 i16
$ ./canopencomm [1] 4 write 0x1017 0 i16 5000
```

Ohjelma 1. *SDO-protokollalla tiedon lukeminen ja kirjoittaminen solmun objektikirjastosta.*

Ohjelmassa 1 ensin luetaan solmun 4 objektikirjaston muistiosoitteesta 0x1017 (alaindeksi 0) 16-bittinen kokonaisluku ja tämän jälkeen samaan muistiosoitteeseen kirjoitetaan arvo 5000. Dokumentaatiota tarkastelemalla selviää, että muistiosoitteesta 0x1017 löytyy tieto laitteen heartbeat-viestin lähetystaajuudesta. Toisin sanoen ohjelmassa 1 asetetaan solmun 4 heartbeat-viestin lähetys lähetettäväksi joka viides sekunti.

CANopenNode/CANopenSocket on syntaksiltaan verrattain selkeä ja tarjoaa riittävän dokumentaation ja opetusmateriaalia itsestään. Koska CANopenNode on tehty standardi C-kielellä, on sitä mahdollista ajaa vaihtelevissa ympäristöissä. Rajapinnan käyttö vaatii myös käyttäjältä huomattavaa tarkkuutta bittien, tavujen ja heksadesimaalien kanssa.

4.2 CanFestival

CanFestival on ilmainen avoimen lähdekoodin LGPLv2 lisenssin alla julkaistu ANSI-C:llä toteutettu CANopen protokollapino. Standardi C-kielen ansiota CanFestival toimii niin PC:llä, kuin mikrokontrollereilla. CanFestivalin rajapintadokumentaatio löytyy lähdekoodin mukana Doxygen-muodossa. Ohjelma 2 kuvaa SDO-protokollalla tiedon kirjoittamista solmuun.

```
err = writeNetworkDict (d          /* CO_Data* d          */,
                        4          /* UNS8  nodeId        */,
                        0x1017     /* UNS16 index         */,
                        0          /* UNS8  subindex      */,
                        2          /* UNS8  count         */,
                        0          /* UNS8  datatype      */,
                        data       /* void  *data         */,
                        0          /* UNS8  useBlockMode  */)
```

Ohjelma 2. *SDO-viestin lähettäminen CanFestivalin ohjelmointirajapinnalla*

SDO-protokollan käyttö CanFestivalissa eroaa hieman CANopenSocketista. CanFestivalin SDO-protokollan funktio *writeNetworkDict* vastaanottaa useampia parametreja. Molemmat rajapinnat vaativat parametreiksi solmun ID:n, muistiosoitteen ja tämän aliosoitteen, kirjoitettavan tietotyypin ja kirjoitettavan arvon. Näiden lisäksi CanFestivalin funktio tarvitsee syötteekseen, kuinka monta tavua tietoa kirjoitetaan (count), käytetäänkö block-tilaa joka mahdollistaa normaalia suurempien datamäärien siirron useissa osissa ja viitteen CAN-objektin tietorakenteeseen (CO_Data). *writeNetworkDict* palauttaa arvon 0, jos kaikki onnistui tai virhekoodin virheen sattuessa. [17]

CanFestival sisältää myös muita funktioita SDO-protokollan käyttämiseen. Yksi näistä on *writeNetworkDictCallBack*, joka eroaa aiemmin esitellystä siltä osin, että funktion yhtenä parametrina voidaan määritellä takaisinkutsufunktio, jota kutsutaan aina silloin kun päätesolmu kuittaa viestin vastaanotetuksi. [17]

4.3 CANopen for Python

CANopen for Python on ilmainen avoimen lähdekoodin CANopen protokollapino. Nimensä mukaisesti CANopen for Python on toteutettu Pythonille ja tukee Pythonin versioita 2.7 ja 3.3+. CANopen for Python on julkaistu MIT-lisenssin alla. [18]

Ohjelma 3 kuvaa, kuinka aiemmin määriteltyyn solmua kuvaavaan olion *node* objektikirjaston kohtaan ’Producer heartbeat time’ kirjoitetaan SDO-protokollalla arvo 5000. Tämä tarkoittaa käytännössä sitä, että solmun *node* heartbeat-viestin lähetys asetetaan lähetettäväksi 5 sekunnin välein.

```
# Kirjoita solmun objektikirjastoon SDO-protokollalla
node.sdo['Producer heartbeat time'].raw = 5000
```

Ohjelma 3. *SDO-protokollalla tiedon lukeminen ja kirjoittaminen*

CANopen for Python ei kuitenkaan toteuta kaikkia CANopen-protokollaperheen protokollia. CANopen for Python tarjoaa esimerkiksi vain verkonhallintaisännän aliprotokollat, eikä toteuta esimerkiksi SDO-protokollaa orjalaitteen näkökulmasta.

CANopen for Python on tehty Pythonille tyypilliseen tapaan lähemmäs luonnollista kieltä ja tarjoaa palveluitaan varsin korkealla abstraktiotasolla, jolloin varsinaisten koodirivien määrä jää suhteellisen pieneksi. Rajapinnan käyttäjän ei tarvitse huolehtia yksityiskohtaisesti kaikista operaatioista bittien tarkkuudella. Ohjelma 2:n koodista myös havaitaan, että objektikirjaston elementit on kuvattu luonnollisella kielellä (tässä tapauksessa ’Producer heartbeat time’) pelkkien muistiosoitteiden sijaan. Muut tässä työssä käsiteltävät rajapinnat eivät tarjoa vastaavanlaista ominaisuutta automaattisesti. CANopen for Python vaatii kuitenkin aina mukaansa Python tulkin eikä näin ollen sovi kaikkiin mikrokontrollereihin. [18]

4.4 Bosch Rexrothin BODAS-ohjainten rajapinta

Bosch Rexroth tarjoaa omille BODAS-sarjan ohjaimilleen C-ohjelmointirajapintaa. Ohjelmointirajapinta sisältää CANopen kirjaston. Rajapinta on suljettua lähdekoodia eikä se ole vapaasti saatavilla internetistä. Myöskään dokumentaatiota ei ole vapaasti saatavilla. Liitteissä C on kuitenkin rajapinnan dokumentaatiosta löytyvää esimerkkikoodia CANopen kirjaston käytöstä. [19]

Ohjelma 4 esittää, miten data_au8 niminen muuttuja kirjoitetaan SDO protokollalla solmun 4 objektikirjastoon. Tässä tapauksessa data_au8, muuttujaan voisi olla tallentuneena esimerkiksi kokonaisluku 5000. Ohjelma 4 tällöin asettaa solmun 4 heartbeat-viestin lähettämisen viiden sekunnin välein.

```
CIA405_SDO_WRITE4(4,              // Node-ID (0 = local)
                  0x1017,         // CANopen index
                  0x00,           // CANopen subindex
                  TRUE,           // Activate query
                  data_au8,       // Data array for the data to be
                                  // written
                  2,              // Length of the data to be written
                  &confirm_l,     // Status flag
                  &error_u16,     // Error code
                  &errorinfo_u32);// CANopen error code
```

Ohjelma 4. *SDO-protokollalla tiedon kirjoittaminen solmuun 4*

Rajapinnan komentojen rakenne on selkeä ja systemaattinen, mutta ne vievät huomattavan määrän rivejä. Funktiokutsun parametrit muistuttavat hyvin paljon CanFestivalin vastaavia. Boschin rajapinta vaatii tarkkuutta bittien, tavujen ja heksadesimaalilukujen kanssa.

Vaikka rajapinta on suljettu eikä näin esimerkiksi muiden käyttäjien kysymiä kysymyksiä ole käytettävissä, on rajapinta erinomaisesti dokumentoitu ja jokaisen käytössä olevan funktion käytöstä löytyy ammattimaisesti toteutetut esimerkit.

5. YHTEENVETO

CANopen tarjoaa perinteisen CAN-väylän päälle korkeamman tason omaisuuksia. Nämä korkeamman tason ominaisuudet mahdollistavat aiempaa laajempien ja monimutkaisten verkkojen rakentamisen useiden toimijoiden yhteistyönä. CANopen kuitenkin vain määrittelee käytettävät protokollat, eikä ota kantaa käytännön toteutukseen. CANopenin käytännön toteutuksia eli ohjelmointirajapintoja on kuitenkin tehty useita niin harrastelijoiden, kuin alan yritysten toimesta.

Kaikki tässä työssä käsitellyt rajapinnat toteuttivat kaikki keskeiset CANopenin ominaisuudet. Kaikki rajapinnat toteuttivat NMT, SDO ja PDO-protokollat verkonhallintaisäntänä toimimiseksi. Eräs huomattava puute oli kuitenkin esimerkiksi CANopen for Pythonissa: se ei mahdollistanut orjalaitteena toimimista. CanFestival tarjosi hyvin monipuolisesti useita eri funktioita SDO-protokollan käytölle ja mahdollisti SDO-protokollassa niin sanotun block-tilan käytön joka ei ollut mahdollista muissa rajapinnoissa.

Syntaksiltaan rajapinnat vaihtelivat keskenään. Rexroth ja CanFestival muistuttivat toisiaan läheisesti, kun taas CANOpenSocket ja CANopen for Python olivat molemmat omalaatuisia. CANOpenSocketin dokumentaatio esitteli esimerkissään käytön Linuxin terminaalien kautta ja CANopen for Python taasen toteutti CANopenin ominaisuudet suuremman abstraktiotason kautta. Myös CANopen for Pythonia olisi luultavasti mahdollista käyttää suoraan Python tulkin kautta. CANOpenSocket ja CANopen for Python mahdollistavatkin, että väylään voidaan liittyä suoraan ilman tarvetta kääntää ohjelmaa ensin. Tämä voisi nopeuttaa esimerkiksi ohjelmistojen prototyyppien kokeilua ja väylän testaamista, kun komentoja voidaan suorittaa suoraan tulkin avulla ilman käännöksen tekemistä.

Kaikki ohjelmointirajapinnat oli dokumentoitu hyvin. Kaikkien ohjelmistorajapintojen dokumentaatiot tarjosivat esimerkkikoodia. Tätä esimerkkikoodia on esitelty tämän työn liitteissä. CanFestival käytännössä vaati kuitenkin Github-repositoryn kloonaamisen omalle tietokoneelleen, koska dokumentaatio oltiin tuotettu Doxygenin avulla.

Kandityötä voisi jatkokehittää tutustumalla entistä läheisemmin käsiteltyihin rajapintoihin. Ohjelmointirajapinnoista voisi tehdä muun muassa ominaisuustaulukon, josta selviäisi mitä ominaisuuksia mikäkin ohjelmointirajapinta toteuttaa, jotta rajapintoja olisi helpompi vertailla keskenään. Lisäksi rajapintojen suorituskykyä ja laitteistovaatimuksia voisi tutkia syvällisemmin. Myös käytännön kokeilut rajapintojen kanssa paljastaisivat varmasti uusia asioita kuten vaikkapa puutteita dokumentaatiossa.

LÄHTEET

[1] CAN in Automation (CiA): Controller Area Network (CAN), CAN in Automation, verkkosivu. Saatavissa (viitattu 10.10.2017): https://www.can-cia.org/.

[2] M. Di Natale, H. Zeng, P. Giusto, A. Ghosal, I. Books24x7, Understanding and Using the Controller Area Network Communication Protocol : Theory and Practice, Springer, New York, NY, 2012, 223 p.

[3] M. Korkee, Etähallintalaitteen liittäminen osaksi CANopenjärjestelmää, Tampereen teknillinen yliopisto, 2013, 56 p.

[4] J.J. Hänninen, Kaivoslastauskoneiden CANopen-ohjausjärjestelmän simulointisovellus, Tampereen teknillinen yliopisto, 2016, 55 p.

[5] CAN Best and Worst Case Calculator - EmSA, EmSA, verkkosivu. Saatavissa (viitattu 10.11.2017): http://www.esacademy.com/en/library/calculators/can-best-and-worst-case-calculator.html.

[6] J. Ferreira, A. Oliveira, P. Fonseca, J. Fonseca, An Experiment to Assess Bit Error Rate in CAN, in: Anonymous (ed.), In Proceedings of 3rd International Workshop of Real-Time Networks (RTN2004}, ACM New York, NY, USA ©2017, Pittsburgh, PA, USA, 2004, pp. 15-18.

[7] P. Koopman, E. Tran, G. Hendrey, Toward middleware fault injection for automotive networks, 1998, pp. 78-79.

[8] Microsoft The OSI Model's Seven Layers Defined and Functions Explained, verkkosivu, viitattu 13.11.2017, saatavissa https://support.microsoft.com/fi-fi/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained.

[9] O. Pfeiffer, A. Ayre, C. Keydel, Embedded networking with CAN and CANopen, Copperhill Media, San Clemente, CA, USA, 2008, 533 p.

[10] National Instruments The Basics of CANopen - National Instruments, verkkosivu, viitattu 15.10.2017, saatavissa http://www.ni.com/white-paper/14162/en/.

[11] linuxgeek81 What is it about CANopen, video. Saatavissa (viitattu 15.10.2017): https://www.youtube.com/watch?v=k35wlcQs0Qw.

[12] T. Casagrande, Raspberry Pi CAN-väyläohjaimena, Metropolia Ammattikorkeakoulu, 2015, 19 p.

[13] H. Saha, CANopenin perusteet, FLUID Finland, Iss. 4, 2006, pp. 6.

[14] CiA 301 Version 4.2.0, CAN in Automation, saatavissa (viitattu 10.11.2017): http://www.can-cia.org/index.php?id=specifications&no_cache=1.

[15] CANopen Quickstart Guide for the Renesas CAN Development Kit RCDK8C, ESD-Electronics, Inc. Saatavilla (viitattu 11.11.2017): http://www.esd-electronics-usa.com/Shared/Handbooks/Renesas/CANopen-Quickstart-Guide.pdf.

[16] H. Zeltwanger, Good to know: PDO re-mapping procedure, CANopen newsletter, 2016, pp. 28-29. Saatavilla (viitattu 23.11.2017): can-newsletter.org/uploads/media/raw/e522172be523845be799af1e5d6dd1f7.pdf.

[17] Dupin Francis, Bossard Camille, Romieux Laurent, Tisserant Edouard, Bessard Laurent, Zulliger Raphael, Duminy David, Belamari Zakaria, CanFestival v3.0 Manual, https://hg.beremiz.org/CanFestival-3/raw-file/tip/objdictgen/doc/manual_en.pdf.

[18] Sandberg Christian, CANopen for python Github, 2017, verkkosivu, viitattu 1.12.2017, saatavissa https://github.com/christiansandberg/canopen.

[19] P. Lehmusoksa, Suunnittelupäällikkö, Patria Land Systems Oy, sähköpostikeskustelu 10.10.2017.

LIITE A: OSA KUVITTEELLISESTA EDS TIEDOSTOSTA

```
[DeviceInfo]
VendorName=Valmistaja Oy
VendorNumber=0x123
ProductName=Tuote123
ProductNumber=0xA0000001
RevisionNumber=1
OrderCode=0
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1
SimpleBootUpMaster=0
SimpleBootUpSlave=1
Granularity=8
DynamicChannelsSupported=0
CompactPDO=0
GroupMessaging=0
NrOfRXPDO=8
NrOfTXPDO=8
LSS_Supported=0

[DummyUsage]
Dummy0001=0
Dummy0002=1
Dummy0003=1
Dummy0004=1
Dummy0005=1
Dummy0006=1
Dummy0007=1

[MandatoryObjects]
SupportedObjects=3
1=0x1000
2=0x1001
3=0x1018

[1000]
ParameterName=Device type
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=131474
PDOMapping=0
```

LIITE B: SUOMENNETTU CANOPENNODE ALOITUSOPAS

Avaa terminaali väylän tarkkailua varten.

Alusta CAN laite seuraaville terminaalikomennoilla:

```
$ sudo modprobe vcan
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
```

Suorita candump can-utilsista:

```
$ sudo apt-get install can-utils
$ candump vcan0
```

Terminaali näyttää nyt kaiken liikenteen vcan0 väylällä. Toiseen terminaaliin avataan canopend. Käännä ja käynnistä canopend:

```
$ cd CANopenSocket/canopend
$ make
$ app/canopend --help
$ app/canopend vcan0 -i 4 -s od4_storage -a od4_storage_auto
```

Ensimmäiseen terminaaliin pitäisi nyt ilmestyä CAN-viestejä. Odota muutama sekunti ja paina Ctrl-C.

```
vcan0  704   [1]  00                        # Käynnistysviesti.
vcan0  084   [8]  00 50 01 2F F3 FF FF FF   # EMCY-viesti.
vcan0  704   [1]  7F                        # Heartbeat-viesti
vcan0  704   [1]  7F                        # toinen Heartbeat-viesti.
```

Heartbeat-viestit ilmoittavat laitteen olevan esitoiminnallisessa tilassa (0x7F). EMCY-viestin neljännen tavun 0x2F tieto ilmoittaa ”CO_EM_NON_VOLATILE_MEMORY”, johon on CANopenNode/stack/CO_Emergency.h kirjattuna lisätieto “0x2F, generic, critical, Error with access to non volatile device memory”. Toisin sanoen laitteella ei ole pääsyä laitteen haihtumattomaan muistiin. Ensimmäiset kaksi tavua näyttävät virheviestiä 0x5000 (Device Hardware) ja kolmas tavu virherekisteriä. Jos virherekisterin arvo on muu kuin nolla, laite ei voi siirtyä toiminnalliseen tilaan ja PDO-viestejä ei voida lähettää eikä vastaanottaa.

Ongelma voidaan jäljittää lähdekoodista. Ohjelmasta puuttuu kuitenkin vakioarvosta poikkeavia säiliötiedostoja. Lisäämällä oikeat säiliötiedostot ja ajamalla canopend uudelleen kaiken pitäisi toimia:

```
# Toinen terminaali
$ echo - > od4_storage
$ echo - > od4_storage_auto
$ app/canopend vcan0 -i 4 -s od4_storage -a od4_storage_auto

# Ensimmäisessä terminaalissa näkyvät tulosteet
vcan0  704   [1]  00                  # Käynnistysviesti.
vcan0  184   [2]  00 00               # PDO-viesti
vcan0  704   [1]  05                  # Heartbeat-viesti
```

Käynnistä kolmas terminaali ja käännä ja käynnistä canopencomm.

```
$ cd CANopenSocket/canopencomm
$ make
$ ./canopencomm --help
```

Nyt kaikki on valmista SDO-protokollaa varten. Seuraavilla komennoilla voimme muokata solmun 4 objektikirjastoa ja muuttaa heartbeat-viestin lähettämistaajuutta.

```
$ ./canopencomm [1] 4 read 0x1017 0 i16
$ ./canopencomm [1] 4 write 0x1017 0 i16 5000
```

Komennon parametrit viittaavat solmuun 4, objektikirjaston osoitteeseen 0x1017 (heartbeat), alaindeksi 0, tietotyyppiä 16-bittinen kokonaisluku ja uusi kirjoitettava tieto 5000 millisekuntia.

Ensimmäisessä terminaalissa on nyt näkyvillä SDO-viestintää. Nyt solmun 4 heartbeat-viestit tulevat viiden sekunnin välein aiemman yhden sekunnin sijasta. Tallenna nyt objektikirjasto, jotta muuttujien arvot säilyvät ohjelman seuraavaan käynnistyskertaan:

```
$ ./canopencomm 4 w 0x1010 1 u32 0x65766173
```

CANopenNoden objektikirjaston muuttujista voi lukea lisää [canopend/CANopenSocket.html].

Jos solmu on toiminnallisessa tilassa, se voi viestiä kaikilla protokollilla (PDO, SDO ja niin edespäin). Esitoiminnallisessa tilassa PDO-viestit eivät toimi, mutta SDO-viestit toimivat. Pysäytetyssä tilassa hyväksytään vain NMT-viestejä.

```
$ ./canopencomm 4 preop
$ ./canopencomm 4 start
$ ./canopencomm 4 stop
$ ./canopencomm 4 r 0x1017 0 i16             # time out
$ ./canopencomm 4 reset communication
$ ./canopencomm 4 reset node
$ ./canopencomm 3 reset node
```

CANopend terminaalista nähdään, että molemmat laitteet ovat valmiita eivätkä laitteet voi enää vastaanottaa viestejä. Asetuksista voidaan myös asettaa siten, että reset komento nollaa myös tietokoneen.

LIITE C: KOODIESIMERKKI BOSCHIN RAJAPINNASTA

```
// reserve memory for send queue and databox
#define CAN2_TX_BUF_LEN   50
#define CAN2_TX_BOX_NBR    2
static can_Message_ts   can2_tx_buf[CAN2_TX_BUF_LEN];
static can_TxDatabox_ts can2_tx_box[CAN2_TX_BOX_NBR];

...

// register send queue, databox and activate CANopen for CAN 2
can_registerTxBuf(CAN_2, can2_tx_buf, CAN2_TX_BUF_LEN);
can_registerTxDataboxes(CAN_2, can2_tx_box, CAN2_TX_BOX_NBR);
canopen_init(CAN_2, 200, 1);
// The variables must absolutely be static or global, because they
// must be located at the same address on each cyclic call!
// data_au8[4] contains heartbeat producer every 500 ms
static uint8 data_au8[4] = {0xF4, 0x01, 0x00, 0x00};
static boolean confirm_l;
static CIA405_CANOpen_Kernel_Error_tu16 error_u16;
static CIA405_SDO_Error_tu32 errorinfo_u32;

// Process has not yet completed, i.e. start or continue
CIA405_SDO_WRITE4(100,           // Node-ID (0 = local)
                  0x1017,        // CANopen index
                  0x00,          // CANopen subindex
                  TRUE,          // Activate query
                  data_au8,      // Data array for the data to be
                                 // written
                  2,             // Length of the data to be written
                  &confirm_l,    // Status flag
                  &error_u16,    // Error code
                  &errorinfo_u32);// CANopen error code


// Process ended successfully?
if (FALSE != confirm_l)
{
  // Write query successfully completed

  <individual code>

// Release used resources!
  CIA405_SDO_WRITE4(100,
                    0x1017,
                    0x00,
                    FALSE,             // Deactivate query
                    data_au8,
                    2,
                    &confirm_l,
                    &error_u16,
                    &errorinfo_u32);
}
```

```
// Process terminated with an error?
else if (FALSE == confirm_l && 0 != error_u16)
{
  // Error case (error code is in error_u16 or errorinfo_u32)

  // Release used resources!
  CIA405_SDO_WRITE4(100,
                    0x1017,
                    0x00,
                    FALSE,            // Deactivate query
                    data_au8,
                    2,
                    &confirm_l,
                    &error_u16,
                    &errorinfo_u32);
}
else
{
  // Don't do anything: request has not yet been completed
}
```

LIITE D: CANOPEN FOR PYTHON KOODIESIMERKKI

```
import canopen

# Aloita luomalla network kuvaamaan yhtä CAN-verkkoa
network = canopen.Network()

# Lisää solmuja vastaavilla EDS-datalehdillä
node = network.add_node(6, '/path/to/object_dictionary.eds')
network.add_node(7, '/path/to/object_dictionary.eds')

# Yhdistä CAN-väylään
# Syötteet välitetään python-canin can.interface.Bus() rakentajalle
# (katso https://python-can.readthedocs.io/en/latest/bus.html).
network.connect(bustype='socketcan', channel='can0')

# Kirjoita solmun objektikirjastoon SDO-protokollalla
node.sdo['Producer heartbeat time'].raw = 5000

# Vaihda solmun tila toiminnalliseksi
node.nmt.state = 'OPERATIONAL'
```