TAMPEREEN TEKNILLINEN YLIOPISTO

MATTI HELMINEN

KONELUETTAVAN CANOPEN-SUUNNITTELUAINEISTON MONITEKNINEN HYÖDYNTÄMINEN

Diplomityö

Tarkastajat:
professori Tommi Mikkonen
professori Kari T. Koskinen
Tarkastajat ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 9.1.2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HELMINEN, MATTI: Koneluettavan CANopen-suunnitteluaineiston monitekninen hyödyntäminen
Diplomityö, 56 sivua
Kesäkuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastajat: professori Tommi Mikkonen ja professori Kari T. Koskinen
Avainsanat: CANopen, koneluettava data, semanttinen tietomalli, RDF, OWL, simulointi, virtuaalinen konelaboratorio, koneen yhteissuunnittelun tuki

Suurin osa suunnittelutyöstä tehdään tänäpäivänä tietokoneella ja siitä syntyy valtavia määriä dataa. Oikein käsiteltynä ja rikastettuna tätä dataa on mahdollista hyödyntää uusissa sovelluskohteissa. Eräs suunnittelutiedon uudelleenkäytön kohde on virtuaalinen konelaboratorio. Virtuaalinen konelaboratorio antaa mahdollisuuden nähdä koneen sisäinen rakenne ja tarkastella, testata ja diagnosoida koneen osia. Näin konejärjestelmän toiminta on helpommin ymmärrettävissä.

Aikaisemmissa virtuaalisen konelaboratorion toteutuksissa materiaali on tuotettu käsin. Virtuaalinen konelaboratorio sisältää usean tekniikanalan tietoa ja näiden tietojen tuottaminen on vaatinut runsaasti käsityötä eikä ole ollut erityisen tehokasta. Näistä tarpeista syntyi idea olemassaolevan aidon suunnitteluaineiston käyttämiseen osana virtuaalista konelaboratoriota. Eri tekniikanalojen suunnitteluaineistot osoittautuivat kuitenkin keskenään irrallisiksi ja ne oli saatava linkittymään keskenään ennen kuin niitä voitaisiin hyödyntää. Virtuaalista konelaboratoriota varten luotiin semanttinen malli, johon koottiin eri lähteistä peräinsin olevat aineistot ja linkitettiin ne keskenään.

Tässä diplomityössä käsitellään CANopen-suunnitteluaineiston yhdistämistä muuhun suunnitteluaineistoon semanttisen mallin avulla, CANopen- ja sähkösuunnitteluaineiston yhdistämistä keskenään, sekä esitellään sovelluskohteita tälle linkitetylle aineistolle. Suunnitteluaineiston yhdistäminen vaati komponenttien yksikäsitteistä tunnistamista koko aineistosta, jotta useamman tekniikanalan aineistoista löytyviä, samaa komponenttia koskevia, tietoja osattiin liittää yhteen. Yksikäsitteistä tunnistamista varten alkuperäistä suunnitteluaineistoa piti rikastaa ja tämä onnistui CANopen- ja sähkösuunnitteluaineiston osalta alkuperäisillä suunnitteluohjelmilla suunnitteluprosessia sopivasti ohjaamalla. Työssä CANopen-aineisto liitettiin osaksi moniteknisen aineiston kattavaa semanttista mallia ja linkittyvää CANopen-aineistoa hyödynnetiin virtuaalisen konelaboratorion osana väyläkaavio- ja datalehtinäkymissä. Aineistoa käytettiin myös vianhakumonitorissa ja simulointimallin generoinnin apuna.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HELMINEN, MATTI: Multi-technical utilization of machine readable CANopen design material
Master of Science Thesis, 56 pages
June 2013
Major: Software Systems
Examiners: professor Tommi Mikkonen and professor Kari T. Koskinen
Keywords: CANopen, machine-readable data, semantic data model, RDF, OWL, simulation, virtual machine laboratory

Most of today's design work is done by computer and it creates massive amounts of data. With the right design process and enrichment this data can be used in new application areas. One of these areas is virtual machine laboratory. In the virtual machine laboratory user has a chance to view inside the machine and examine, measure and diagnose different parts of the machine. This way machines functionality is more easily understood.

Previous creation of virtual machine laboratories has required enormous amount of manual labour. Because virtual machine laboratory includes multitechnical data, creation of this material was very time consuming and not very efficient. Within these needs raised the idea of using existing design material as part of the virtual machine laboratory. Multitechnical design material was in separated pieces and had to be combined befoce it could be used. Semantic data model was created for the virtual machine laboratory and different design materials were linked to it.

This thesis focuses on combining the CANopen-design material as part of the semantic model, linking CANopen and electrical engineering design material together and utilizing these combined materials in different scenarios. Unique identifiers are required for linking these design materials from different sources together. These identifiers had to be added to the original design material and this worked out with original CANopen- and electric engineering design programs by following a proper design process. In this thesis CANopen design material was combined with multidisiplinary semantic model and this combined data was used as part of the virtual machine laboratorys diagram view and datacheet view. Combined data was also used in CANopen troubleshooting monitor and for generating simulation model.

ALKUSANAT

Diplomityö käsittelee Tampereen teknillisessä yliopistossa Semogen- ja Semogen II -projekteissa syntyneitä tuloksia ja havaintoja liittyen CANopen-suunnitteluaineiston hyödyntämiseen osana semanttista mallia, virtuaalista konelaboratoriota ja muita käyttökohteita. Projektit toteutettiin yhdessä Hydrauliikan ja automatiikan laitoksen, Matematiikan laitoksen yhteydessä toimivan Intelligent Information Systems Laboratoryn, sekä yrityskumppaneiden kesken.

Diplomityön parissa ja projekteissa työskennellessäni sain oppia paljon uutta liittyen väylätekniikoihin, erityisesti CANopen-väylää, mutta sain myös lisää ohjelmistopuolen kokemusta erilaisten työkalujen ja ohjelmointikielten parissa.

Haluan kiittää kaikkia projekteissa työskennelleitä henkilöitä, erityisesti työkavereitani Jaakko Salosta ja Juha Nurmea hyvästä tiimihengestä ja yhteisestä osaamisen kartuttamisesta erilaisten ja hyvin vaihtelevien työtehtävien parissa. Kiitokset myös projektin johtohenkilöille, sekä erityisesti Heikki Sahalle CANopen-tekniikkaan liittyvästä asiantuntija-avusta. Lisäksi kiitokset myös diplomityön ohjaajille professori Kari T. Koskiselle ja professori Tommi Mikkoselle tarkastustyöstä ja vinkeistä.

Tampereella 18.5.2013

Matti Helminen

SISÄLLYS

LYHENTEET JA TERMIT

Apache Ant Työkalu ohjelmiston automaattiseen käännösprosessiin. Käyttää XML-pohjaista tiedostoformaattia käännösprosessin ja riippuvuuksien kuvailuun.

CAN *Controller Area Network.*
Koneissa, ajoneuvoissa ja teollisuuslaitteissa käytetty vikasietoinen automaatioväylä. Määritelty ISO 11898-1- ja ISO 11898-2 -standardeissa.

CANopen Korkeamman tason integraatioalusta, kommunikaatioprotokolla- ja sovellusprofiilimäärityserhe pääasiassa CAN-laitteille. Määrittelee myös tiedostot, prosessin ja työkalujen ohjelmointirajapinnat.

CiA *CAN in Automation.*
Kansainvälinen käyttäjien ja valmistajien organisaatio, joka aktiivisesti kehittää ja ylläpitää CANopen-integrointialustaa.

CAN-ID 11-bittisen CAN Message ID-kentän päälle CANopen-protokollassa määritelty viestin tunniste. Sisältää 4-bittisen funktiokoodin ja 7-bittisen Node ID:n.

COB-ID *Communication object identifier.*
CANopen-viestille objektikirjastossa määritelty 32-bittinen tunniste, joka sisältää CAN-ID:n lisäksi joitakin asetusbittejä.

COBD *CANopen profile database.*
Tietokanta joka sisältää laiteprofiilin mukaisia määrityksiä CANopen-laitteelle.

CPD-tiedosto COBD profiilitietokannan tallennusformaatti. Tiedostopäätteeltään CPD.

CSV *Comma-separated values.*
Tiedostomuoto taulukko muotoisen tiedon tallennukseen tekstinä. Jokainen tietoalkio on erotettu pilkulla, puolipisteellä, tabulaattorilla tai muulla sovitulla erotinmerkillä. Taulukon rivit on eroteltu rivinvaihdolla.

Datalehti Dokumentti, joka kuvaa tuotteen, koneen tai komponentin tekniset ominaisuudet.

DCF *Device Configuration File.*
Kuten EDS, mutta sisältää myös CAN-laitteen verkkokohtaiset asetukset, kuten node ID:n, siirtonopeuden ja konfiguroitujen parametrien arvot. Tarkoitettu laitekonfiguraation tallentamiseen. Määritelty CiA 306-1- ja CiA 306-3 -standardeissa.

DIP-kykin *Dual in-line package -kytkin*
Heikkoja virtoja ohjaava rakenteeltaan pieni kytkin. Usein termillä tarkoitetaan myös useampia pieniä kytkimiä sisältävää yksikköä. Voidaan käyttää elektronisen laitteen ominaisuuksien tai asetusten muuttamiseen.

Eclipse Alustariippumaton ohjelmointiympäristö, joka on laajennettavissa lisäosien avulla. Tuettuja ohjelmointikieliä ovat mm. Java, C, C++, PHP ja Python.

EDS *Electronic Data Sheet.*
CANopen-laitteen koneluettava datalehti. Tallennettu INI-tyyppiseen tiedostoformaattiin, joka kuvaa laitteen ominaisuudet, parametrit ja niiden oletusarvot. Määritelty CiA 306-1- ja CiA 306-3 -standardeissa.

EMCY *Emergency protocol*
Protokolla, jolla siirretään virheilmoituksia CANopen-verkossa. Määritelty CiA 301 -standardissa.

INI *INI-tiedostoformaatti.*
Standardi tapa kuvata konfigurointitietoa Windows-ympäristössä. INI tulee sanasta initialization. Korvattiin myöhemmin Windows Registerillä.

ISO *International standardisation organisation.*
Kansainvälinen standardisoimisjärjestä. Standardoinut mm. CAN-väylän.

LSS *Layer Setting Services*
Protokolla, jolla CANopen-laitteen solmutunnisteen ja siirtonopeuden voi muuttaa suoraan CANopen-väylän välityksellä ilman manuaalisia DIP-kytkimiä tai muita ratkaisuita.

NMT *Network management*
Isäntä-orja-arkkitehtuurin mukainen protokolla CANopen-laitteiden hallintaan. Määritelty CiA 301 -standardissa.

Net ID *Network ID*
CANopen-verkosta käytettävä tunniste. Yksilöi CANopen-verkon useamman verkon kokoonpanosta. Tallennettu DCF-tiedostoon 32-bittisenä positiivisena kokonaislukuna nimellä NetNumber.

Node CAN-verkossa olevaa laitetta kutsutaan node:ksi. Suomenkielessä puhutaan solmuista.

Node ID CANopen-verkossa laitteen eli solmun yksilöivä tunniste. Tunniste on kokonaisluku väliltä 1-127.

OWL *Web Ontology Language.*
W3C:n standardoima tiedon kuvailukielien perhe. Suunniteltu koneluettavan tiedon kuvailuun niin, että kuvailun perusteella voidaan myös tehdä koneellista päättelyä.

PDO *Process Data Object.*
CANopen-protokolla sykliseen signaalien arvojen päivitykseen solmujen välillä. Siirrettävä tieto voi olla esimerkiksi pyörimisnopeus, jännite, taajuus, virta, paine, jne.

PLC *Programmable Logic Controller.*
Ohjelmoitava logiikka. Toteuttaa usein laitteen ohjauksen ja hallinnoi CANopen-verkkoa.

RDF *Resource Description Framework.*
W3C:n standardoima kuvailukieli tiedon tallennukseen ja vaihtoon sovellusten välillä. Voidaan esittää useilla formaateilla mm. RDF/XML muodossa XML-pohjaisena sarjallistuksena.

RDFS *RDF Schema.*
W3C:n standardoima ontologioiden ja luokkien määrittelytapa RDF-kielellä. Mahdollistaa OWL-ontologioiden käytön osana RDF-kuvailukieltä.

RPDO *Receive PDO.*
CANopen-laitteen vastaanottama PDO-viestikehys.

SDO *Service Data Object.*
CANopen-protokolla laitteiden konfigurointiin, kuten Node ID:n, siirtonopeuden ja muiden asetusten muuttamiseen.

Solmu CAN-verkossa olevaa laitetta kutsutaan solmuksi. Englanniksi node.

Solmutunniste Sama kuin Node ID.

SPARQL *SPARQL Protocol and RDF Query Language*
W3C:n standardoima RDF kyselykieli, jolla voi hakea ja muokata RDF-tietokantaan tallennettua tietoa.

SVG *Scalable Vector Graphics.*
XML-pohjainen kaksiulotteisen vektorigrafiikan kuvausformaatti, joka on määritelty W3C:n avoimessa standardissa.

SYNC *Synchronization*
Protokolla, jolla voidaan tahdistaa CANopen-laitteiden viestintä, esimerkiksi PDO-viestien lähetys tapahtumaan säännöllisesti. Määritelty CiA 301 -standardissa.

TIME TIME-protokolla, jolla välitetään aikatietoa CANopen-verkossa. Määritelty CiA 301 -standardissa.

TPDO *Transmit PDO*
CANopen-laitteen lähettämä PDO-viestikehys.

TTY *Tampereen teknillinen yliopisto.*

URI *Uniform Resource Identifier*
Merkkijono jolla tieto voidaan yksikäsitteisesti tunnistaa. Erityisesti verkkoresurssien tunnistuksessa käytetty tapa.

VML *Virtual Machine Laboratory.*
Virtuaalinen konelaboratorio.

W3C *World Wide Web Consortium*
Kansainvälinen yritysten ja yhteisöjen yhteenliittymä, joka ylläpitää ja kehittää WWW:n standardeja ja suosituksia.

XML *Extensible Markup Language*
W3C:n määrittelemä merkkauskieli, jolla voidaan tuottaa samanaikaisesti sekä koneen että ihmisen luettavia dokumentteja tai datarakenteita.

XSLT *Extensible Stylesheet Language Transformations*
XML-pohjainen merkintäkieli XML-tiedostojen muunnoksiin. Valmiita työkaluja XSLT-muutostiedoston ajamiseen on saatavilla useisiin ympäristöihin myös Eclipseen.

1. JOHDANTO

Tampereen teknillisen yliopiston SmartSimulators -tutkimusryhmä on kehittänyt virtuaalisia konelaboratorioita teollisuuden ja koulutusorganisaatioiden kanssa liikkuvan työkoneasentajan osaamisen kehittämisen tueksi. Virtuaalisessa konelaboratoriossa käyttäjällä on mahdollisuus nähdä koneen sisäistä toimintaa ja tarkastella sitä eri näkökulmista dynaamisen reaaliaikasimulaation, mittaustyövälineiden ja erilaisten 3D- ja kaavionäkymien avulla.

Virtuaalisten konelaboratorioiden kehitys on vaatinut runsaasti käsityötä eri aineistojen osalta. Tuotantomenetelmien automatisoinnille, informaation uudelleenkäytölle, tuotantoprosessin tehokkuudelle ja moniteknisen tiedon yhdistämiselle ilmeni suuri tarve. Näistä tarpeista syntyivät Semogen- ja Semogen II -hankkeet, joissa tutkittiin moniteknisen suunnitteluaineiston hyödyntämistä osana virtuaalista konelaboratoriota, sekä suunnittelumenetelmien kehitystä koneluettavuuden kannalta. Tämä diplomityö on tehty osana näitä hankkeita. Hankkeessa käytettiin aitoja suunnitteluaineistoja ja tutkittiin niiden koneellista käsittelyä, yhdistämistä ja rikastamista, sekä suunnitteluprosessien kehittämistä. Semogen-hankkeiden ja tämän diplomityön tulosaineistot ja tuotetut työkalut on saatavilla Smart Simulators -tutkimusryhmän wiki-sivuilta (http://wiki.tut.fi/SmartSimulators/).

Tässä diplomityössä keskitytään CANopen-suunnitteluaineiston linkittämiseen muuhun suunnitteluaineistoon ja tämän moniteknisen aineiston hyödyntämistä osana virtuaalista konelaboratoriota ja muita sovelluskohteita, joissa monitekninen aineisto antaa uutta lisäarvoa. CANopen-aineisto on jo valmiiksi koneluettavaa ja hyvin standardoitua, joten se on erinomainen esimerkki koneluettavasta aineistosta. Linkittyvyyttä muihin aineistoihin täytyy kuitenkin parantaa ja se on mahdollista tietyin muutoksin CANopen-suunnitteluprosessiin, sekä suunnitteluohjelmien käyttöön. Hyvin linkittyneestä aineistosta voidaan koostaa semanttinen malli ja tätä mallia voidaan hyödyntää edelleen esimerkiksi virtuaalisen konelaboratorion tietolähteenä tai siitä voidaan generoida uusia näkymiä monitekniseen tietoon.

Työn alussa luvussa kaksi esitellään CAN-väylä sekä väylän fyysinen rakenne, että viestikehysten tyypit ja sisällöt. CAN toteuttaa vain OSI-mallin alimpien kerrosten toiminnot ja sen päällä voidaan käyttää korkeamman tason protokollia kuten CANopen:ia. Viestikehysten rajoitteet ja ominaisuudet periytyvät siten CAN:sta myös CANopen:iin.

Luvussa kolme käydään CANopen-teknologiaperhe tarkemmin läpi ja kuvataan sen sisältämät eri protokollat, objektikirjaston määrittelyt, sekä laitekohtaiset profiilit. Myös CANopen-laitekonfigurointien tallennusformaatit käsitellään, koska tallennettu suunnittelutieto on tärkeässä osassa tässä diplomityössä.

Luvussa neljä esitellään käytettyyn CANopen-järjestelmään liittyvä suunnitteluaineisto, sekä suunnitteluohjelmat. Myös aineiston linkittämisen vaatimat muutokset suunnitteluprosessiin käydään läpi.

Luvussa viisi kuvataan semanttinen malli, sekä miten aineisto linkittyy semanttiseen malliin. Aineiston lukemiseksi semanttiseen malliin tehtiin useita työkaluja. Nämä työkalut ja koko prosessi semanttisen mallin tuottamiseksi esitellään myös tässä luvussa.

Luvussa kuusi kuvataan miten semanttista mallia tai linkittyvää suunnitteluaineistoa voidaan hyödyntää eri sovelluskohteissa, kuten virtuaalisessa konelaboratoriossa, vianhaussa tai simulointimallin generoinnin apuna.

Luvussa seitsemän esitetään yhteenveto tuloksista ja pohditaan jatkokehitysmahdollisuuksia ja tulevaisuuden näkymiä semanttisen mallinnuksen ja CANopen-suunnittelun osalta.

2. CAN-AUTOMAATIOVÄYLÄ

Controller Area Network (CAN) on automaatioväylä, jota käytetään ajoneuvoissa, junissa, laivoissa, lentokoneissa, satelliiteissa ja teollisuuslaitteissa. Sen avulla voidaan siirtää digitaalista tietoa antureiden, toimilaitteiden ja ohjelmoitavan logiikan välillä. CAN esiteltiin alunperin ajoneuvokäyttöön Society of Automotive Engineers (SAE) kokouksessa 1986. CAN on osoittautunut yksinkertaiseksi, monipuoliseksi ja vikasietoiseksi, joten sen käyttö on levinnyt kaikille aloille, joilla tarvitaan viestintää mikroprosessoreiden välillä. Ajoneuvojen ohella CAN-väylää käytetään teollisuusautomaatiossa, mutta myös muualla, jossa hajautetusta ohjauksesta saavutetaan hyötyä tai väyläratkaisulla voidaan vähentää erilliskaapeloinnin tarvetta. CAN-väylän avulla on saavutettu muihin väyläratkaisuihin verrattuna matalat kustannukset, hyvä toimivuus vaikeissa ympäristöissä, hyvät reaaliaikavalmiudet, sekä erinomainen virheenhavaitsemis- ja sietokyky, mutta se on silti säilynyt helppokäyttöisenä. [1]

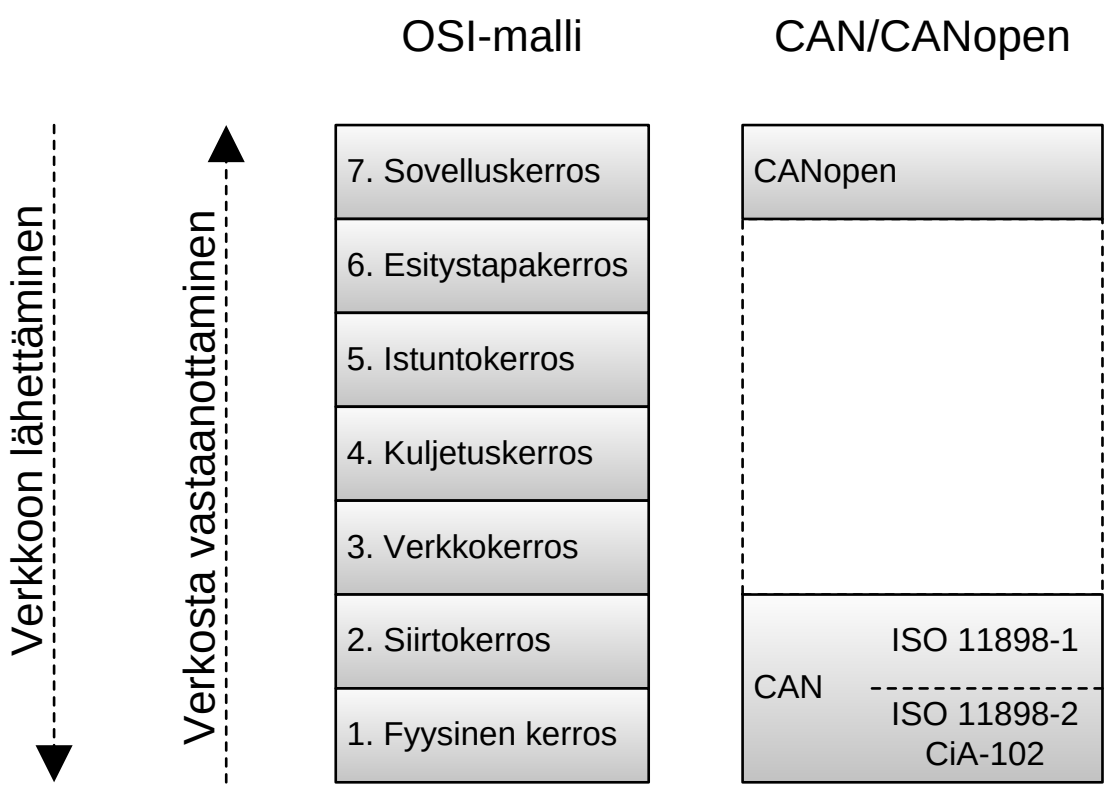
Digitaalinen tiedonsiirto ja virheentarkistussummat sekä kiinteät kentät tekevät havaitsemattomista virheistä harvinaisia. CAN-verkossa on hyvin pienet viiveet esimerkiksi verrattuna ethernet-verkkoon ja tämä antaa hyvät mahdollisuudet CAN:n käyttöön reaaliaikasovelluksissa. CAN on matalan tason protokolla (vain fyysinen ja siirtokerros, kts. kuva 2.1) ja siksi kevyt toteuttaa. Tästä johtuen CAN-laitteiden toiminnallisuus on yleensä integroitu suoraan laitetason piirille. [1, s.24]

2.1 Fyysinen kerros (CAN High-Speed)

OSI-malli eli Open Systems Interconnection Reference Model kuvaa tiedonsiirtoprotokollien 7 kerrosta [2]. Kukin kerros käyttää aina seuraavan kerroksen palveluita. Näistä kerroksista alimpana on fyysinen kerros (kts. kuva 2.1). Kaikki muu on rakennettu sen päälle.

CAN-väylälle on määritelty useita erilaisia fyysisiä kerroksia. Yleisin ja tässä diplomityössä esitelty on CAN High-Speed, joka on määritelty ISO 11898-2 -standardissa. Muita fyysisen kerroksen standardeja ovat esimerkiksi CAN Low-Speed (ISO 11898-3) ja Single-wire CAN (SAE J2411). [3]

CAN High-Speed -standardi ISO 11898-2 [4] keskittyy pääasiassa fyysisiin signaaleihin. Sallitut tiedonsiirtonopeudet ja kaapelointi sekä liittimet on määritelty CiA 303-1-standardissa [5]. Fyysinen kerros siis määrittelee verkon rakenteen ja sen miten yksittäiset bitit lähetetään jännitetasoina johtimia pitkin.

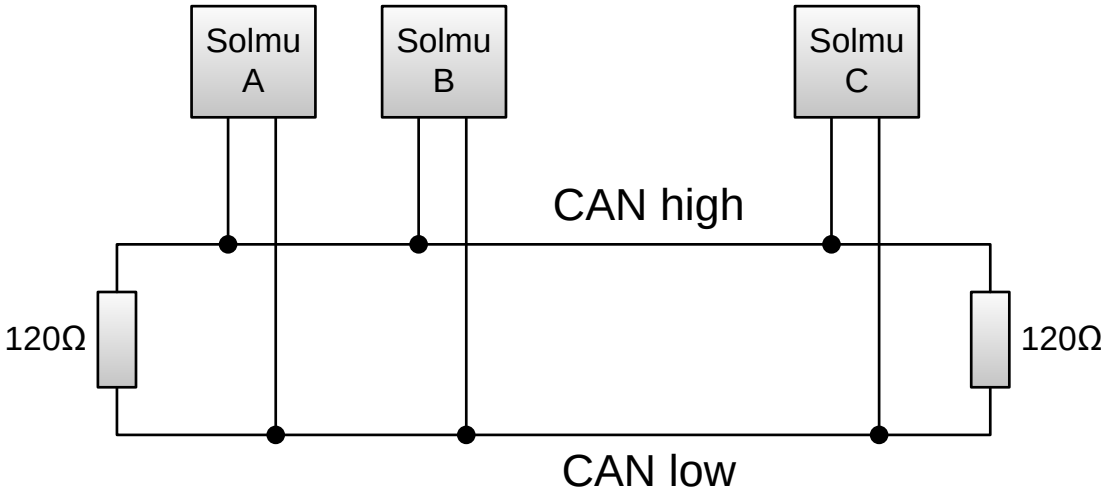


Kuva 2.1: OSI-mallin kerrokset suhteessa CAN- ja CANopen-standardeihin. OSI-malli mukailtu lähteestä [2].

2.1.1 Fyysisen kerroksen rakenne

CAN-verkossa olevia laitteita kutsutaan solmuiksi (nodes). CAN High-Speed -verkossa kaikki solmut on kytketty toisiinsa kahdella johtimella, *CAN high* ja *CAN low* [1, s.132]. Väylän rakenteen tulee olla aina lineaarinen, ja väylä on terminoitu molemmista päistä 120 ohmin vastuksilla, kuten kuvasta 2.2 nähdään. Terminoinnilla estetään signaalien heijastumiset väylällä. Väylä tulee aina terminoida molemmista päistä erillisellä terminaattorilla. Jos terminaattori on fyysisesti CAN-laitteessa kiinni, terminointi menetetään, jos laite irrotetaan väylästä.

CAN-väylän maksimipituus riippuu tiedonsiirtonopeudesta. Kun väylälle lähetetään bitti on sen ehdittävä väylän toiseen päähän ja takaisin ennen seuraavan bitin lähettämistä. Näin kaikki laitteet pystyvät priorisoimaan viestinsä suoraan dominoivan ja resessiivisen bitin avulla viestin lähetyksen alusta alkaen. Tästä johtuen nopeudella 1MBit/s väylän maksimipituus on noin 25m. Hitaammilla nopeuksilla voidaan käyttää pitempiä väyliä. Esimerkiksi 125kBit/s nopeudella voidaan käyttää 500m pitkää väylää ja 10kBit/s nopeudella jopa 5km väyläpituutta [3,6].



Kuva 2.2: Fyysinen CAN-väylä.

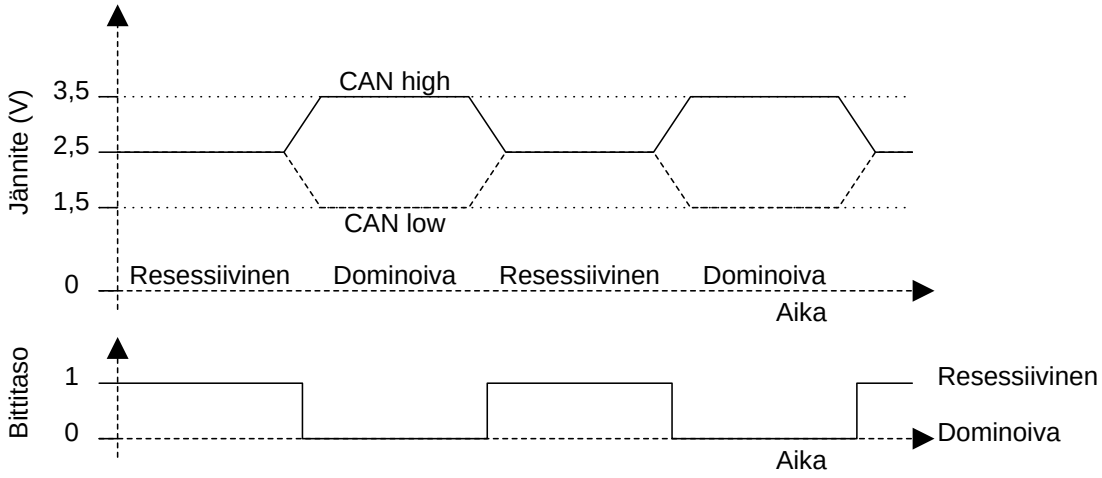
2.1.2 Fyysisen kerroksen signaalitasot

CAN-väylällä 0-bitti on dominoiva ja 1-bitti on resessiivinen. Tämä tarkoittaa sitä, että jos useampi laite lähettää väylälle samaan aikaan ja mikä tahansa niistä lähettää dominoivaa 0-bittiä, näkyy väylällä kaikille 0-bitti. 1-bitti näkyy väylällä vain silloin, kun kukaan ei lähetä 0-bittiä. Kaikki väylälle lähettävät solmut tarkkailevat samalla väylän tilaa, ja jos ne huomaavat resessiivistä bittiä lähettäessään väylällä jo olevan dominantin bitin, ne osaavat lopettaa lähettämisen. Näin laite, jonka viestin alussa on eniten dominoivia bittejä saa lähetysoikeuden. Tätä käytetään myös väylän viestien priorisoinnissa siirtokerroksessa.

Resessiivisen bitin aikana molemmissa johtimissa on 2,5V jännite. Dominantin bitin aikana *CAN high* on 3,5V ja *CAN low* on 1,5V. Näin saadaan 2V jännite-ero johtimien välille (kts. kuva 2.3). Väyläsignaalin tulkinta biteiksi tapahtuu *CAN high*- ja *CAN low*-johtimien jännite-eron avulla. Ulkoiset häiriöt eivät vaikuta signaalin tulkintaan, koska ne muuttavat molempien johtimien jännitettä saman verran ja näin ollen jännite-ero pysyy vakiona [1, s.137].

2.2 Siirtokerros

OSI-mallin mukainen siirtokerros määritellään CAN-standardissa ISO 11898-1 [7]. CAN-siirtokerros tarjoaa OSI-mallin datasiirtomahdollisuudet sekä virheenhavaitsemiskeinoja [8]. Data siirretään data-kehyksillä, joissa on virheentarkistussumma ja oikein vastaanotetun sanoman kuittausbitti. Virheellisestä tarkistussummasta ja muista siirtovirheistä lähetetään kuitenkin aina väittömästi myös error-kehys ja näin tieto virheestä välittyy kaikille verkon solmuille. Solmujen määrää kasvattamalla jäännösvirhetodennäköisyys pienenee, koska riittää että yksi solmu havaitsee vir-



Kuva 2.3: Bittiesitys väylällä.

heen.

CAN-verkossa kaikki viestit näkyvät kaikille laitteille. Verkossa ei myöskään ole isäntä- tai orja-laitteita, vaan kaikki solmut ovat saman arvoisia. Jos useampi laite yrittää lähettää väylälle viestiä samaan, viesti jonka alussa on eniten 0-bittejä peräkkäin lähetetään. Data-kehykset alkavat tunnistenumerolla (*Message ID*), joten pienimmän tunnistenumeron omaava viesti saa aina lähetysoikeuden.

CAN-verkossa yli 5 peräkkäistä samaa bittiä katsotaan virheeksi. Jos normaalisti datakehyksen sisällä olevaan dataan kuuluu yli 5 saman merkkistä bittiä lisää lähettävä solmu vastakkaisen merkkisen täytebitin viidennen saman merkkisen bitin jälkeen. Tätä kutsutaan bit stuffingiksi. Bit stuffingilla saadaan vastaanottajan bittitahdistusta helpotettua kun vaihtelevia bittejä tulee tarpeeksi usein.

2.2.1 Siirtokerroksen viestikehystyypit

CAN-standardissa kaikkia CAN-viestejä kutsutaan kehyksiksi (frames). Kaiken CAN-väylälle välitettävän tiedon on oltava yhteensopivaa viestikehyksien määrittelyn kanssa. [1]

CAN määrittelee seuraavat neljä viestikehystyyppiä:

Datakehys. Datakehys välittää viestin, joka sisältää varsinaisen datan, CAN-väylälle [1, s.37]. Datakehys lähetetään yleensä mitattavan arvon muuttuessa, ajastetun toiminnon mukaan tai vastauksena viestipyyntöön. Datakehys tunnistetaan kehyksen alussa olevan *Message ID*:n perusteella. Kehyksen voi vastaanottaa yksi tai useampi väylällä oleva solmu sovelluksen tarpeesta riippuen. Kuitenkin vain yksi solmu voi lähettää tietyn *Message ID*:n omaavaa viestiä.

Remote-kehys. Remote-kehyksellä mikä vain solmu voi pyytää dataa joltain

muulta solmulta, jolloin remote-kehystä seuraa datakehys sisältäen pyydetyn datan. Remote-kehys tunnistetaan RTR-bitistä (Remote Transmission Request). Remote-kehys vastaa muutoin datakehystä, mutta siltä puuttuu data-kenttä kokonaan ja remote-kehyksen RTR-bitti on 1, kun datakehyksellä se on 0 [1, s.40]. Remote-kehyksessä määriteltävän datan pituuden (data length code) on vastattava pyydetyn datakehyksen datan pituutta, joten solmun on tiedettävä ennalta montako bittiä kysyttävän datan pituus on. Remote-kehys ja siihen vastauksena saatava datakehys käyttävät myös samaa Message ID:tä [1, s.39]. Remote-kehyksen käyttö ei ole suositeltavaa [9]. Koska CAN-standardi ISO 11898-1 ei määrittele remote-kehysten toteutusta, toteutus on valmistajasta riippuvainen [1, s.41]. Lisäksi solmut, joissa ohjelmisto on kaatunut voivat vastata vanhentuneilla tiedoilla remote-kehykseen, koska sovelluslogiikka toimii yleensä eri piirillä ja CAN-tason kommunikointi on toteutettu suoraan latteistotasolla.

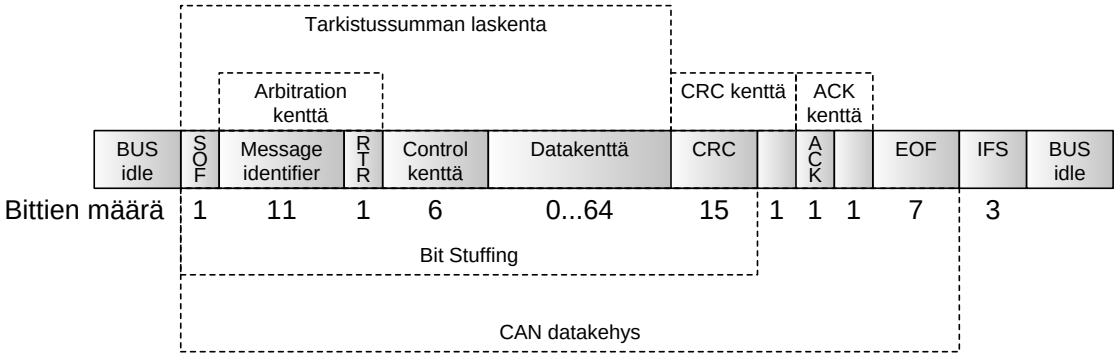
Error-kehys. Error-kehys koostuu virhelipusta ja virhe-erottimesta. Virhelippu on 6-bittinen ja koostuu pelkästään dominoivista 0-biteistä. Sitä seuraa 8-bittinen virhe-erotin joka koostuu pelkästään resessiivisistä 1-biteistä. Error-kehys lähetetään aina kun väylällä havaitaan virhe. Error-kehyksen voi lähettää mikä tahansa väylällä oleva solmu, joko lähettävä tai vastaanottava [1, s.57]. Virheeseen reagoidaan aina välittömästi lähettämällä error-kehys tai CRC-virheen tapauksessa odotetaan 2 bitin verran, jotta Acknowledgement-kenttä CRC-kentän jälkeen saadaan myös lähetettyä. CAN-standardi sallii vain 5 samaa peräkkäistä bittiä normaalin datan joukossa. Suurempi määrä samoja bittejä peräkkäin, joko dominantteja tai resessiivisiä, katsotaan virheeksi. Sääntö ei koske EOF-lopetuskenttää tai tyhjää väylää. Error-kehys sisältää aina vähintään 6 dominanttia bittiä peräkkäin joten muut solmut katsovat sen myös aina virheeksi ja aloittavat sopivan reagoinnin. Tämä tahallinen standardin rikkominen takaa virheellisen datan varman tuhoutumisen. Jokainen virheen tai error-kehyksen vastaanottanut solmu vastaa siihen myös error-kehyksellä. Tällöin error-kehys saattaa koostua useista osittain päällekkäisistä error-kehyksistä. Error-kehyksen jälkeen seuraa normaali 3 bitin viive, jonka jälkeen virheellinen kehys yritetään lähettää uudelleen. Tämä lyhyt virheestä toipumiseen vaadittava aika on yksi CAN-standardin eduista verrattuna muihin kenttäväylästandardeihin.

Overload-kehys. Overload-kehys on erikoistapaus error-kehyksestä, mutta toisin kuin error-kehys overload-kehys ei aiheuta edellisen kehyksen uudelleenlähetystä [1, s.66]. Overload-kehys koostuu 6 dominantista ja 8 resessiivisestä bitistä kuten error-kehyskin. Overload-kehys voidaan lähettää data- tai remote-kehyksen EOF-kentän jälkeen, kun taas error-kehys voidaan lähettää kesken data- tai remote-kehyksen. Overload-kehyksen tarkoituksena on aiheuttaa viivettä seuraavan viestikehyksen lähetykseen, jos vastaanottava solmu on ylikuormittunut. Enintään kaksi overload-kehystä voidaan lähettää peräkkäin. Haittapuolena overload-kehysten käy-

töstä on väylän kuormittuminen ja myös kaikkien muiden solmujen keskinäisen viestinnän estyminen hetkeksi. Overload-kehystä ei nykyään käytetä ja se on vanhentunut tekniikan kehittyessä ja CAN-piirien suorituskyvyn kasvaessa.

2.2.2 Data- ja remote-kehyksen rakenne bittitasolla

Data- ja remote-kehysten rakenne on muutoin kuvan 2.4 mukainen, mutta remote-kehykseltä puuttuu data-kenttä. Kehykset koostuvat seuraavista kentistä:



Kuva 2.4: Datakehyksen rakenne. Määritelty ISO 11898-1-standardissa. Mukailtu lähteestä [1].

SOF. Start of Frame (SOF) bitti on dominoiva 0-bitti, joka aloittaa Data- tai Remote-kehyksen [1, s.45]. CAN-solmun on odotettava kunnes väylä on vapaana, ennen kuin se voi yrittää lähettää uutta kehystä väylälle. Väylä todetaan vapaaksi, kun vähintään 11 resessiivistä bittiä on tullut peräkkäin, ja näin ollen aloitetaan lähetys dominoivalla bitillä.

Arbitration-kenttä. Arbitration-kenttä on 12- tai 32-bittinen, riippuen siitä käytetäänkö lyhyttä (standard format) 11-bitin *Message ID*:tä vai pidempää (extended format) 29-bittistä ID:tä [1, s.45]. Viestien priorisointi hoidetaan *Message ID*:n avulla. Koska nolla-bitti on dominoiva saa pienimmän Message ID:n omaava viesti aina korkeimman prioriteetin ja siten lähetysoikeuden väylälle. Arbitration kenttä päättyy RTR-bittiin. RTR (Remote Transmission Request) määrittää onko kyseessä data- vai remote-kehys. Data-kehyksessä RTR saa arvon 0 ja remote-kehyksessä RTR on 1. Tästä johtuen datakehyksellä on myös korkeampi prioriteetti kuin remote-kehyksellä.

Control-kenttä. 6-bittisen control-kentän ensimmäinen bitti kertoo onko käytössä lyhyt (bit = 0) vai pitkä (bit = 1) *Message ID*. Toinen bitti on varattu tulevaa käyttöä varten. Neljä viimeistä bittiä kertovat data-kentän pituuden tavuina. Data-kentän pituus voi kuitenkin olla nollasta maksimissaan vain kahdeksaan tavua [1, s.47]. Remote-kehyksen tapauksessa data-kentän pituudella kuvataan vastauksena tulevan viestin data-kentän pituutta, koska lähetettävällä remote-kehyksellä ei ole data-kenttää.

Data-kenttä. Datakehyksellä data-kenttä sisältää enintään kahdeksan tavua eli 64 bittiä tietoa, mutta voi olla myös puuttua kokonaan. Remote-kehyksellä ei koskaan ole data-kenttää.

CRC-kenttä. CRC-kenttä (Cyclical Recovery Checking) sisältää 15-bittisen tarkistussumman ja yhden resessiivisen erotinbitin. Tarkistussumma lasketaan kehyksessä ennen CRC-kenttää olevista biteistä, lukuun ottamatta täytebittejä (bit stuffing) [1, s.48].

ACK-kenttä. ACK-kenttä, yhteensä 2 bittiä, sisältää 1 bitin acknowledgement slotin ja resessiivisen erotinbitin. ACK-kentällä varmistetaan onko tarkistussumma oikein [1, s.49]. ACK-kentän aikana lähettävä solmu siirtyy vastaanottavaan tilaan lähettämällä resessiivistä bittiä verkkoon. Samaan aikaan kaikki muut solmut varmistavat tarkistussumman oikeellisuuden ja lähettävät dominantin bitin jos tarkistussumma täsmää. Alkuperäinen lähettäjä tarkkailee väylää ja jos kukaan ei lähetä dominanttia bittiä, eli hyväksy viestin tarkistussummaa, voidaan päätellä, että virhe on lähettäjässä. Koska virhe kuitataan dominoivalla bitillä riittää, että yksi vastaanottaja saa viestin oikein perille. Virhetapaus, jossa esimerkiksi yksi solmu saa viestin virheellisenä ei kuitenkaan jää havaitsematta, koska jokainen solmu lähettää aina virheellisen viestin tapauksessa error-kehyksen ja virhe havaitaan tämän avulla. ACK-kentän avulla voidaan kuitenkin erotella se aiheutuuko virhe vastaanottajan vai lähettäjän päässä. CAN-standardi mahdollistaa myös solmun irtaantumisen verkosta (self-removal) jos se havaitsee itse aiheuttavansa liikaa virheitä.

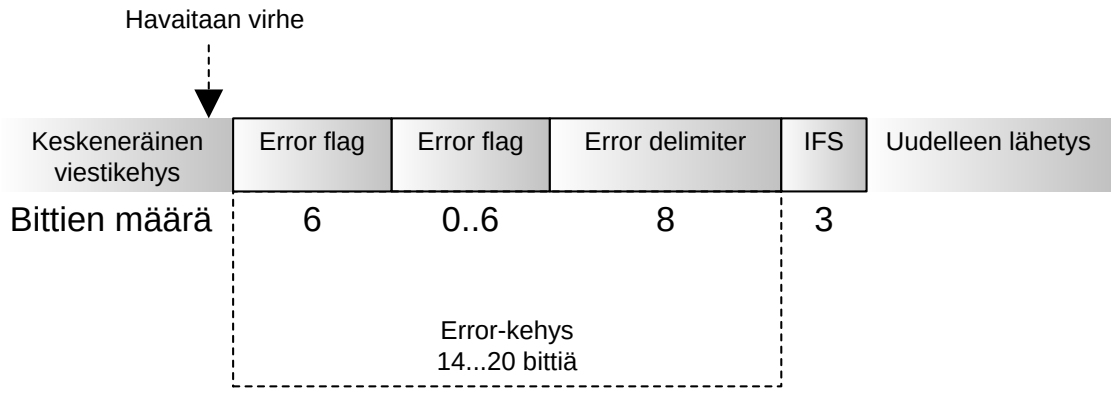
EOF. Jokainen data- tai remote-kehys lopetetaan 7 bitin resessiivisellä End-of-Frame-jaksolla.

IFS. Interframe Space tarkoittaa minimiväliä eri kehysten välillä, joka on 3 resessiivistä bittiä. IFS:n aikana mikään solmu ei voi aloittaa data- tai remote-kehyksen lähetystä [1, s.70]. IFS:n jälkeen väylä palaa idle-tilaan, jos mikään solmu ei lähetä kehystä. Idle-tila on myös resessiivinen tila.

2.2.3 Error- ja overload-kehyksen rakenne bittitasolla

Error- ja overload-kehysten rakenne (kuva 2.5) on sama. Error-kehys lähetetään aina heti virheen sattuessa, kesken vastaanotettavaa kehystä. Overload-kehys lähetetään vasta edellisen kehyksen loputtua ja sillä pyritään aiheuttamaan viivettä, jos vastaanottava solmu on kuormittunut.

Kehys koostuu 6-bittisestä virhe-lippusta (error flag), joka sisältää pelkästään dominoivia 0-bittejä. Sitä seuraa 8-bittinen virhe-erotin (error delimiter), joka puolestaan koostuu pelkästään resessiivisistä 1-biteistä. Koska yli 5 samaa bittiä aiheuttaa aina virheen, vastaavat muut solmut tähän aina error- tai overload-kehyksellä. Vastaus tapahtuu välittömästi 5 peräkkäisen bitin jälkeen. Tästä johtuen siis error- ja overload kehys voi koostua useista osittain päällekkäisistä samanlaisista kehyksistä,



Kuva 2.5: Error-kehyksen rakenne.

joten sen pituus voi venyä 14-bitistä enintään 20-bittiin.

2.3 Korkeamman tason kerrokset

CAN ei ota kantaa korkeamman tason toteutuksiin vaan toteuttaa vain alimmat kerrokset. Toteuttamalla vain alimmat kerrokset säästetään laitteiden muistinkuormituksessa ja pienennetään verkon overheadia, jonka seurauksena suorituskyky paranee [1, s.132]. Korkeamman tason protokollia, jotka toimivat CAN-protokollan päällä on useita esimerkiksi J1939, ISO Bus, DeviceNet ja CANopen [10].

3. CANOPEN-KOMMUNIKAATIOPROTOKOLLA JA LAITEPROFIILIT

CANopen on kansainvälisesti standardoitu usein CAN-pohjainen korkean tason protokollaperhe, jota käytetään sulautetuissa oh jausjärjestelmissä. CANopen määrittelee sovelluskerroksen sekä viestintäprofiilin, että sovellus-, laite- ja liityntäprofiilit [11]. CANopen-verkkoja käytetään laajasti eri sovellusalueilla, kuten autoissa, työkoneissa, teollisuudessa, laivoilla ja lentokoneissa erilaisten laitteiden ja toimintojen ohjaukseen, sekä anturointiin.

CAN siirtokerroksena mahdollistaa vikasietoisen redundantin verkon rakentamisen. CAN sisältää useita standardeja erilaisia fyysisiä kerroksia varten eri sovelluskohteille. Yleisin ja käytetyin näistä on CAN high-speed medium access unit (MAU), joka on määritetty ISO 11898-2 standardissa [11]. CANopenia voidaan kuitenkin käyttää myös muiden kuin CAN-verkon päällä, esimerkiksi Ethernet-verkossa [12].

CANopen on määritelty CAN-datakehysten päälle, joissa on 11-bittinen tunnistekenttä (Message ID). 29-bittistä laajennettua tunnistekenttää ei ole pakko tukea, mutta sitä voidaan myös käyttää, jos kaikki solmut sitä tukevat [13, s.23]. CANopen:n yhteydessä tunnistekentästä käytetään nimitystä CAN-ID. Yleensä CAN-ID koostuu 4-bittisestä funktiokoodista ja 7-bittisestä Node ID:stä. Funktiokoodilla tunnistetaan mitä protokollaa kyseinen viesti edustaa. [14] Protokollat on esitelty myöhemmin tässä luvussa. Node ID on yleensä palveluun liittyvän solmun Node ID. Tunnistekentän lisäksi CANopen hyödyntää CAN-datakehyksen data-kenttää. CANopen-viesteissä dataa voidaan lähettää 0 - 8 tavua (kuten CAN-viesteissäkin) riippuen protokollasta ja tarpeesta.

Jotkin CANopen-protokollat käyttävät lisäksi CAN remote-kehyksiä datan pyytämiseen toiselta solmulta, mutta näiden käyttöä ei suositella. Syitä tähän ovat mm. data-kentän puuttuminen, remote-kehyksen häviäminen datakehykselle, jos molemmat lähetetään samaan aikaan, sekä erilaiset valmistajakohtaiset toteutukset remote-kehyksen käsittelyyn CAN-ohjaimessa. Lisää tietoa remote-kehyksen ongelmista löytyy CiA 802 -sovellusmuistiosta. [9]

3.1 Objektikirjasto

Objektikirjasto on tärkein osa CANopen-laiteprofiilia. Objektikirjasto on ryhmä objekteja, joihin voi päästä käsiksi ennalta määritellyllä tavalla CANopen verkon kaut-

ta. Kirjasto toimii CANopen-solmussa keskitettynä tietovarastona [15]. Objektit pitävät sisällään CANopen-laitteen liikennöintiin ja sovelluksiin liittyvät parametrit. Myös laitteen signaalien arvot päivitetään sovelluksen ja väylän välillä objektikirjaston kautta.

3.1.1 Objektien indeksit

Jokainen objektikirjaston objekti on numeroitu 16-bittisellä indeksillä, joten objekteja voi olla enintään 65536 kappaletta. Indeksit on jaettu taulukossa 3.1 esitettyihin alueisiin. Tämä jako muistuttaa muita kenttäväylästandardeja. [13, s.17-18] Lisäksi jokaisella indeksillä voi olla myös 8-bittinen ali-indeksi [16, s.88]. Tämä mahdollistaa objektien kokonaismäärän kasvattamisen yli 16 miljoonaan.

Indeksialue heksadesimaalina	**Sisältö**
0000	Ei käytössä
0001-001F	Staattiset tietotyypit
0020-003F	Moniosaiset tietotyypit
0040-005F	Valmistajakohtaiset moniosaiset tietotyypit
0060-025F	Laiteprofiilikohtaiset tietotyypit
0260-0FFF	Varattu tulevaisuuden laajennuksiin
1000-1FFF	Solmukohtaiset tunnisteet ja liikennöintiparametrit
2000-5FFF	Valmistaja- tai sovelluskohtainen alue
6000-67FF	Laiteprofiilikohtainen alue 1
6800-6FFF	Laiteprofiilikohtainen alue 2
7000-77FF	Laiteprofiilikohtainen alue 3
7800-7FFF	Laiteprofiilikohtainen alue 4
8000-87FF	Laiteprofiilikohtainen alue 5
8800-8FFF	Laiteprofiilikohtainen alue 6
9000-97FF	Laiteprofiilikohtainen alue 7
9800-9FFF	Laiteprofiilikohtainen alue 8
A000-AFFF	Prosessimuuttujat
B000-BFFF	Prosessimuuttujat
C000-FFFF	Varattu tulevaisuuden laajennuksiin

Taulukko 3.1: Objektikirjaston indeksien karkea aluejako. Mukailtu lähteistä [16, s.87] ja [15].

Staattiset tietotyypit indekseillä 0001-001F pitävät sisällään tyyppimäärittelyt standardeille tietotyypeille, joita ovat esimerkiksi totuusarvo, kokonaisluku, positiivinen kokonaisluku, reaaliluku, merkkijono, jne. Moniosaiset tietotyypit indekseillä 0020-003F ovat ennalta määrättyjä yhdistelmiä standardeista tietotyypeistä ja ovat yhteisiä kaikille CANopen laitteille. Valmistajakohtaiset moniosaiset tietotyypit indekseillä 0040-005F ovat myös yhdistelmiä standardeista tietotyypeistä, mutta ovat

laitekohtaisia. Laiteprofiilit voivat määritellä lisäksi staattisia tai moniosaisia laitekohtaisia tietotyyppejä. Nämä on määritelty indekseillä 0060-025F. [16, s.88]

Indeksit 1000-1FFF sisältävät solmukohtaiset tunnisteet ja liikennöintiparametrit eli kommunikaatioon liittyvät parametrit. Nämä ovat yhteisiä kaikille CANopen laitteille. Indeksien 2000-5FFF välinen alue on varattu sovellusparametreille ja valmistajakohtaisille arvoille, joihin valmistaja voi tehdä sovelluksesta riippuen omaa toiminnallisuuttaan. [16, s.88]

Laiteprofiilikohtainen alue 6000-9FFF sisältää kaikki CANopenin määrittelemät dataobjektit joita voidaan lukea tai joihin voidaan kirjoittaa verkon kautta. Nämä objektit voivat olla parametreja tai kuvata toiminnallisuutta. Yksi fyysinen CANopen laite voi sisältää useamman loogisen CANopen laitteen. Laiteprofiilikohtainen alue on jaettu kahdeksaan osaan kuten taulukossa 3.1 on esitetty. Jos yksi CANopen laite sisältää useamman erillisen loogisen laitteen jokainen näistä laitteista käyttää omaa osaansa laiteprofiilikohtaisesta alueesta. [16, s.88]

Indeksialue A000-BFFF on varattu ohjelmoitavan logiikan ja CANopen siltojen prosessimuuttujille [15, s.8].

3.1.2 Objektien objektityypit

Kaikilla objekteilla on objektikoodi (CPD-, DCF- ja EDS-tiedostoissa *ObjectType*), joka kuvaa objektin tyypin. Objektikoodi määrittää onko objekti yksi muuttuja vai taulukko tai tallenne, joka voi sisältää useita ali-indeksejä, joissa on toisia objekteja. [16, s.89] Eri objektikoodien tyypit on esitelty taulukossa 3.2.

Kuten edellä mainittiin objektikirjaston objekteihin viitataan 16-bittisellä indeksillä. Jos objekti sisältää yksinkertaisen muuttujan indeksillä viitataan suoraan siihen. Jos taas indeksi sisältää monimutkaisen tietorakenteen tai taulukon viitataan indeksillä koko rakenteeseen. Tietorakenteen osiin voidaan taas viitata 8-bittisellä ali-indeksillä. [16, s.88] Ali-indeksi 0 kertoo suurimman ali-indeksin numeron ja vasta seuraavat ali-indeksit sisältävät varsinaisen datan [13, s.18].

3.1.3 Objektien datatyypit

Objektin datatyyppi voi olla jokin indekseillä 0001-025F määritelty tietotyyppi (kts. taulukko 3.1. Datatyyppi voi olla joko standardin määrittelemä tietotyyppi tai valmistajan määrittelemä tietotyyppi. Standardi määrittelee perustietotyypit totuusarvoille, 8-64 bittisille etumerkillisille ja etumerkittömille kokonaisluvuille ja 32-64 bittisille reaaliluvuille. Lisäksi on ajan ja merkkijonon esitykseen määritelty datatyypit. Tarkemmat datatyyppimääritykset löytyvät CiA 301 -standardista [16].

Asetustiedostoissa datatyypiksi merkitään suoraan heksadesimaali-indeksi, joka viittaa objektikirjaston kohtaan, jossa kyseinen datatyyppi on määritelty. Esimer-

Objektikoodi heksana	Nimi	Merkitys
00	NULL	Objekti ilman datakenttää.
02	DOMAIN	Iso vaihteleva määrä dataa. Esim. ohjelmakoodi.
05	DEFTYPE	Tyyppimäärittely kuten BOOLEAN, FLOAT, jne.
06	DEFSTRUCT	Määrittelee uuden RECORD tyypin.
07	VAR	Yksi arvo kuten UNSIGNED8, FLOAT, VISIBLE STRING, jne.
08	ARRAY	Taulukko samantyyppisiä arvoja kuten esimerkiksi taulukko UNSIGNED16 lukuja. Ali-indeksissä 0 on aina UNSIGNED8 joka määrittelee taulukon koon.
09	RECORD	Taulukko arvoja, jotka voivat olla keskenään eri tyyppisiä. Ali-indeksissä 0 on aina UNSIGNED8 joka määrittelee taulukon koon, sekä ali-indeksissä 255 on aina UNSIGNED32.

Taulukko 3.2: Objektikirjaston objektien tyyppimääritykset. Mukailtu lähteestä [16, s.89].

kiksi datatyyppi UNSIGNED8 määritellään standardin mukaan indeksissä 0005. Jos objektikirjaston objekti on tyypiltään UNSIGNED8, merkitään objektin datatyypiksi silloin 0005.

3.1.4 Objektien käyttöoikeudet

Objektille voi määrittää luku- ja kirjoitusoikeuksia. Oikeudet on määritelty verkosta CANopen-laitteelle päin, ja ne voivat olla taulukon 3.3 mukaiset. [16, s.89]

Attribuutti	Merkitys
rw	Luku- ja kirjoitusoikeus.
wo	Vain kirjoitusoikeus.
ro	Vain lukuoikeus.
const	Vain lukuoikeus. Arvo pysyy vakiona.

Taulukko 3.3: Objektikirjaston objektien käyttöoikeudet. Mukailtu lähteestä [16, s.90].

Const-tyyppisen objektin arvo voi muuttua ainoastaan jos laite on initialisation-tilassa. Muissa NMT-tiloissa (pre-operational, operationa ja stopped) arvo ei voi muuttua.

3.2 NMT-tilakone ja -protokolla

Network management eli NMT on CANopen laitteille suunnattu isäntä-orja-arkkitehtuurin mukainen protokolla. NMT-objekteja käytetään NMT-palveluiden

suorittamiseen. NMT-palveluiden kautta CANopen-laitteet alustetaan, käynnistetään, resetoidaan ja pysäytetään. Myös niiden tilaa voidaan tarkkailla NMT-error control-palvelun Node guarding- ja heartbeat-protokollien avulla. Kaikki CANopen-laitteet ovat NMT-orjia, ja ne tunnistetaan verkossa Node ID -tunnisteen avulla, joka on kokonaisluku väliltä 1-127. NMT vaatii, että yksi laite toimii myös NMT-isäntänä. [16, s.73]

Jokaisella laitteella on sisäinen NMT-tilakone, joka koostuu tiloista: initialisation, pre-operational, operational ja stopped. NMT-isäntä hallitsee orjien tilakoneen tiloja node control -protokollan avulla CANopen verkon yli. Omaa tilaansa isäntä hallitsee paikallisesti. Tilakone ja siirtymät on esitelty kuvassa 3.2. NMT-tilasta riippuen laitteella on käytössä eri protokollia ja sen toiminta voi olla rajattua. Kuvassa 3.1 on esitelty mitkä protokollat missäkin NMT-tilassa ovat käytettävissä.

Protokolla	**Pre-operational**	**Operational**	**Stopped**
PDO		X	
SDO	X	X	
SYNC	X	X	
TIME	X	X	
EMCY	X	X	
Node control and error control	X	X	X

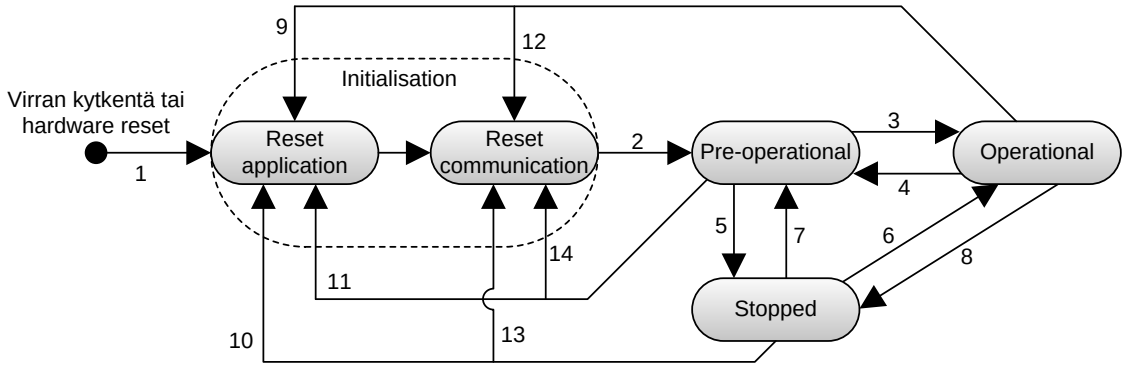
Kuva 3.1: CANopen laitteen tarjoamat palvelut eri NMT-tiloissa [16, s.85].

Network management (NMT) määrittelee node control-, error control- ja boot-up-palvelut ja protokollat. Nämä on kuvattu seuraavissa alakohdissa.

3.2.1 Node control -palvelut

Node control -palveluiden avulla NMT-isäntä voi vaihtaa orjalaitteiden NMT-tiloja CANopen-verkossa node control -protokollan kautta. Palvelu ja protokolla sisältävät seuraavat komennot: start remote node, stop remote node, enter pre-operational, reset application ja reset communication [16, s.77-78]. Näillä orjalaitteen NMT-tiloja voidaan vaihtaa kuvan 3.2 mukaisesti. Reset application- ja reset communication -tiloissa viivytään vain hetki ja niistä siirrytään automaattisesti eteenpäin. Reset application käynnistää koko solmun ohjelmiston ja reset communication alustaa ja käynnistää CANopen-protokollapinon. Solmu voidaan myös nollata uudelleen joko reset application- tai reset communication -tasolle vastaavilla NTM-protokollan komennoilla. Käynnistyksen tai nollauksen jälkeen solmun käytyä läpi reset application- ja reset communication -tilat se siirtyy automaattisesti ensimmäiseen pysyvään tilaansa joka on pre-operational tila. Solmu jää odottamaan pre-operation tilaan NMT-masterin käynnistymistä. Jos kaikki sujuu hyvin, eikä konfi-

gurointivirheitä havaita siirtää NMT-master seuraavaksi solmut operational-tilaan. Stopped tilaan siirrytään yleensä vakavan virheen seurauksena. [15]



Siirtymä	Tapahtuma
1	Virrat kytkettäessä automaattinen siirtyminen initialisation -tilaan.
2	Kun initialisation on valmis automaattinen siirtyminen pre-operational -tilaan.
3	NMT protokolla: start remote node tai paikallinen ohjaus.
4,7	NMT protokolla: enter pre-operational.
5,8	NMT protokolla: stop remote node.
6	NMT protokolla: start remote node.
9,10,11	NMT protokolla: reset application.
12,13,14	NMT protokolla: reset communication.

Kuva 3.2: CANopen laitteen NMT-tilakone ja tilasiirtymät. Mukailtu lähteestä [16, s.83].

3.2.2 Error control -palvelut

Error control -palveluita käytetään virheiden havaitsemiseen CAN-verkoissa. Ne on toteutettu lähettämällä määräajoin viestejä CAN-verkkoon. Standardi määrittelee kolme protokollaa valvoa CANopen-laitteiden toimintaa. Node guarding -protokollassa NMT-isäntä lähettää guarding-pyyntöjä NMT-orjille. Pyynnöt on toteutettu remote-kehyksillä. Jos orja ei vastaa tietyn ajan kuluessa huomataan tämä ja tieto välittyy edelleen NMT-isännän sovellukselle. Life guarding -protokollassa NMT-orja tarkkailee sille tulevia guarding-pyyntöjä. Jos pyyntöä ei saavu tietyn ajan kuluessa NMT-orja tiedottaa tästä omalle sovellukselleen. Heartbeat-protokollassa laite tuottaa määräajoin heartbeat-viestejä ja yksi tai useampi laite voi tarkkailla niitä. Jos tarkkailija ei saa viestiä sopivan ajan kuluessa tarkkailijan ilmoittaa tästä omalle sovellukselleen. [16, s.75] Heartbeat-protokolla on uudempi ja suositeltu tapa toteuttaa virheiden havaitseminen. Node guarding -protokollan käyttöä tulisi välttää, koska siinä käytetään remote-kehyksiä, joiden käyttöä ei suositella [9].

3.2.3 Boot-up-palvelu

Boot-up-palvelun avulla todetaan, että NMT-orja on vaihtanut NMT-tilaansa initialisation-tilasta pre-operational -tilaan [16, s.81]. Tilamuutoksen tapahtuessa laite lähettää verkkoon boot-up -protokollan mukaisen viestin.

3.3 Protokollat ja palvelut

CANopen-standardi kuvaa kommunikointiobjektit, jotka on määritelty palveluiden ja protokollien avulla. Protokollilla on eri tapoja virheiden tunnistamiseen vielä CAN-kerroksen virheentunnistuksen lisäksi. Näitä ovat mm. signaalien alueen valvonta ja erilaiset timeoutit. Jokainen protokolla määrittelee omat viestityyppinsä, joiden avulla kommunikointi kyseisellä protokollalla tapahtuu. Protokollien viestit toimivat pääosin CAN-datakehysten päällä, mutta jotkin toiminnot käyttävät remote-kehystä. Tärkeimmät CANopen-protokollat PDO, SDO, SYNC, TIME, EMCY ja NMT on esitelty alla. Ne löytyvät myös CiA 301 -standardista [16]. Näiden lisäksi on myös esitelty CiA 305 -standardista löytyvä LSS-protokolla, jota käytetään solmujen liikennöintiasetusten konfigurointiin.

3.3.1 PDO-protokolla

Reaaliaikainen tiedonsiirto suoritetaan PDO eli Process Data Object -protokollalla, PDO-viestien avulla. PDO-viestit on toteutettu siten, että niiden lähettäminen ei aiheuta ylimääräistä protokollasta johtuvaa lisäkuormaa, joten koko CAN-viestin 8 tavun datakenttä on käytössä datalle [14]. PDO-viestit peilautuvat suoraan objektikirjaston arvoihin ja tarjoavat siten rajapinnan objektikirjastoon. [16, s.33] PDO-viestien avulla päivitetään prosessisignaalien tiloja eri laitteiden objektikirjastojen välillä [15]. Jos laite tukee muutettavia PDO-määrityksiä (PDO mapping) voidaan PDO-viestien kokoa ja määrää ja niihin kytkettyjä objektikirjaston objekteja vaihtaa SDO-palveluiden avulla muuttamalla oikeita objektikirjaston arvoja [16, s.33].

PDO-viestit on jaettu väylälle lähetettäviin Transmit-PDO-viesteihin (TPDO), sekä väylältä vastaanotettaviin Receive-PDO-viesteihin (RPDO). TPDO-viestejä tuottavia CANopen-laitteita sanotaan PDO-tuottajiksi ja RPDO-viestejä vastaanottavia laitteita PDO-kuluttajiksi. PDO:t määritellään objektikirjaston PDO-kommunikointiparametreissa ja PDO-määrityksissä. CANopen-laitteessa voi olla 512 eri PDO-viestiä määriteltynä. [16, s.33]

PDO-viestejä voidaan lähettää synkronisesti tai tapahtumapohjaisesti. Synkroniset PDO-viestit tahdistetaan SYNC-viestien avulla (kts. alakohta 3.3.3). Synkroniset viestit voidaan lähettää jokaisen SYNC-viestin jälkeen tai vain joka n:nnen SYNC-viestin jälkeen riippuen PDO-viestin lähetysasetuksista. Tapahtumapohjaisien PDO-viestien lähetys ei ole sidoksissa SYNC-viesteihin. PDO-viestejä voidaan

myös pyytää lähettämään remote-pyynnöillä. [16, s.34]

3.3.2 SDO-protokolla

Service data object eli SDO-protokolla tarjoaa suoran pääsyn CANopen-laitteen objektikirjaston merkintöihin [16, s.39]. SDO-protokollassa SDO-viesteillä voidaan lukea tai kirjoittaa objektikirjastojen objekteihin. SDO-viesti sisältää objektikirjaston indeksin, ali-indeksin ja datan. Koska objektikirjaston objektit voivat sisältää mielivaltaisen määrän dataa ja CAN-viestikehykseen mahtuu vain 8 tavua dataa, on SDO-protokollaan määritelty segmentointi. Segmentoinnissa iso datamäärä jaetaan pieniin palasiin ja siirretään usean eri SDO-viestin avulla hallitusti. Segmentointia varten on määritelty kaksi protokollaa, SDO download/upload ja SDO Block download/upload. SDO Block download/upload on uudempi standardi ja sallii suuren datamäärän siirtämisen hieman pienemmällä protokollasta aiheutuvalla kuormalla. [14]

Ensimmäisessä SDO-viestissä voidaan siirtää 4 tavua ja seuraavissa enintään 7 tavua dataa. Sekä lähettäjä että vastaanottaja voivat milloin vain halutessaan keskeyttää segmentoidun tiedonsiirron [16, s.39]. Ensimmäisen SDO-viestin rakenne on esitetty kuvassa 3.3. Sitä seuraavat segmentoidut viestit ovat melko samanlaisia ja eivät sisällä indeksiä ja ali-indeksiä. Siten niihin on saatu mahtumaan 7 tavua dataa.

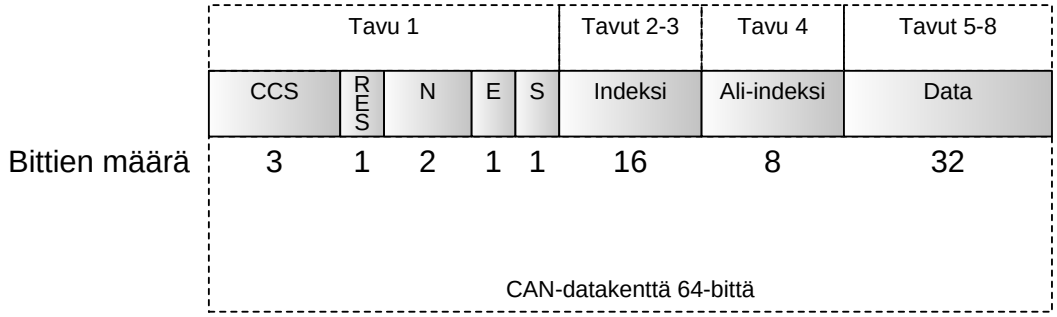
3.3.3 SYNC-protokolla

SYNC-tuottaja lähettää verkkoon määräajoin synchronization-viestejä. Määräaikaa voidaan säätää SYNC-tuottajan objektikirjastosta. SYNC-viesteille on myös annettu hyvin korkea prioriteetti, jotta ne lähtevät ajallaan. [16, s.67] CANopen-verkossa voi olla vain yksi SYNC-tuottaja kerrallaan [15, s.8]. SYNC-kuluttajat voivat tahdistaa esimerkiksi mittauksensa ja PDO-viestien lähetyksensä SYNC-viestin saapumiseen [14].

SYNC-viesti voi myös sisältää 8-bittisen laskurin, jolla solmut voivat ajoittaa toimintansa tapahtumaan esimerkiksi joka viidennellä SYNC-viestillä. Osa laitteista voi myös synkronoitua eri laskurin arvoon kuin muut joten verkon kuormaa voidaan tasoittaa eri SYNC-viestien välille.

3.3.4 TIME-protokolla

TIME-tuottaja lähettää verkkoon time stamp -viestejä. Myös TIME-viestit on priorisoitu korkealle, jotta niissä olisi mahdollisimman vähän viivettä. TIME-tuottaja lähettää verkkoon time stamp -viestin, jonka datana kulkee 6-tavuinen aikatieto. Aikatieto on kuvattu millisekuntteina keskiyöstä ja kuluneina päivinä vuoden 1984



Lyhenne	Selitys
CCS	Client command specifier, jolla määritetään SDO siirron tyyppi: 0 on SDO segmentoitu lataus, 1 on lataamisen aloituskehys, 2 on lähettämisen aloituskehys, 3 on SDO segmentoitu lähetys ja 4 on SDO siirron keskeytys.
RES	Varattu tulevaisuutta varten. Aina 0.
N	Tavujen määrä viestin dataosassa, jotka eivät sisällä dataa. Vain voimassa silloin kun E ja S on asetettu arvoon 1.
E	Jos 1 niin siirto on tyyppiä expedited transfer eli kaikki data on ensimmäisessä viestissä. Jos 0 niin kyseessä on segmentoitu siirto ja data jatkuu seuraavissa viesteissä.
S	Jos 1 niin datan koko on ilmoitettu. Jos E on 1 niin datan koko kerrottu viestin osuudessa N ja on 4-N, jos E=0 on datan koko kerrottu viestin data osuudessa.
Indeksi	Objektikirjaston indeksi datalle jota käsitellään.
Ali-indeksi	Objektikirjaston ali-indeksi datalle jota käsitellään.
Data	Data joka lähetetään expedited transfer (E=1) siirrossa tai datan koko (S=1, E=0).

Kuva 3.3: SDO-download aloituskehys. Mukailtu lähteistä [14] ja [16, s.52].

alusta lähtien. TIME-kuluttajat kuuntelevat näitä viestejä ja voivat tahdistaa kellonsa niiden mukaan. [16, s.68-69]

Normaalisti aikatieto on millisekunnin tarkkuudella, mutta aikakriittisissä sovelluksissa voidaan käyttää korkearesoluutioista ajan esitystä. Tällöin aikatieto on 4-tavuinen, mutta esittää mikrosekunteja, ja laskuri pyörähtää ympäri takaisin nollaan aina 72 minuutin välein. [14]

3.3.5 EMCY-protokolla

Emergency eli EMCY-protokollan avulla siirretään CANopen-verkossa virheilmoituksia, varoituksia ja ilmoituksia virhetilojen poistumisesta [15]. EMCY-viesti lähetetään kun CANopen-laitteessa havaitaan sisäinen virhetilanne. EMCY-viesteillä on korkea prioriteetti ja yhden laitteen tuottamia viestejä voivat vastaanottaa kaikki niistä kiinnostuneet. Tämä tekee EMCY-viesteistä sopivia keskeytys-tyyppisten virhehälytysten antamiseen. [14]

EMCY-viesti lähetetään vain kerran per virhetilanne, eikä niitä toisteta jos sama virhetilanne jatkuu [16, s.69]. EMCY-viestissä lähetetään virhekoodi, virherekisterin numero ja valmistajakohtainen virhekoodi. CANopen standardi määrittelee virhekoodit ja virherekisterit, mutta valmistaja voi määrittää valmistajakohtaiset virhekoodit laiteprofiilissa [14].

3.3.6 LSS-palvelut

LSS-palvelut (Layer Setting Services) määrittelevät kuinka solmun liikennöintinopeutta ja solmutunnistetta muutetaan CANopen-verkossa [17]. Tyypillisesti LSS-palveluita käytetään solmun asetusten muuttamiseen ennen solmun asentamista kohdejärjestelmään, mutta LSS-palveluita voidaan käyttää myös kohdejärjestelmässä milloin tahansa [15]. LSS on kuvattu CiA 305 -standardissa [18].

LSS toimii isäntä-orja -arkkitehtuurin mukaisesti ja CANopen-verkossa voi olla vain yksi LSS-isäntä. Orjia voi olla useita. Isäntälaitteena toimii sama laite kuin NMT-isäntänäkin. LSS-protokollalla voidaan myös korvata kaikki käsikäyttöiset ratkaisut, kuten DIP-kytkimien käyttö Node ID:n tai bittinopeuden asettamiseen ja konfiguroida CANopen laite pelkästään CANopen-verkon avulla. [18]

LSS-protokollan avulla voidaan siis hakea CANopen verkossa olevat laitteet ja konfiguroida niille sopivat Node ID:t ja bittinopeudet. Protokolla sallii solmun Node ID -tunnisteen, bittinopeuden ja LSS-osoiteen lukemisen, muuttamisen ja muutosten tallentamisen. Node ID on CANopen-verkossa laitteen eli solmun yksilöivä tunniste. Node ID on kokonaisluku väliltä 1-127. Jos Node ID on asetettu arvoksi 255 tarkoittaa se sitä, ettei sitä vielä ole konfiguroitu. Bittinopeus voi olla jokin seuraavista: 1000, 800, 500, 250, 125, 50, 20 tai 10kbit/s. LSS-osoite on laitteen yksikäsitteinen tunniste, jolla se voidaan yksilöidä ennen kuin Node ID tunnistetta on asetettu. Node ID yksilöi laitteen yhdessä CANopen-verkossa, kun taas LSS-osoite yksilöi laitteen maailmanlaajuisesti kuten MAC-osoite ethernet-verkossa. LSS-osoite on 128-bittinen ja koostuu myyjän ja tuotteen tunnisteesta, sekä sarja- ja versionumerosta. Jokainen näistä neljästä tunnisteesta on 32-bittinen. LSS-osoite löytyy myös laitteen objektikirjastosta indeksistä 0x1018. [18]

3.4 Laiteprofiilit

CiA on määritellyt eri tyyppisille laitteille laiteprofiileita, joita laitevalmistajien tulee noudattaa. Laiteprofiili kuvaa objektikirjaston objektit yhdelle CANopen laitteelle. Kuvaus sisältää toiminnallisen ja muodollisen kuvauksen objekteista. Toiminnallinen kuvaus määrittää objektin käyttäytymisen objektikirjastossa. Muodollinen kuvaus taas määrittää onko objekti pakko toteuttaa vai onko se valinnaista. Muodollinen kuvaus määrittää myös sen voidaanko objektiin päästä käsiksi CANopen-

verkosta ja miten. [16, s.22]

Laiteprofiili on siis yleiskuvaus tietyn tyyppisestä laitteesta ja laitevalmistaja voi halutessaan tarjota tarkemman kuvauksen, joka toteuttaa laiteprofiilin pakolliset objektit ja mahdollisesti myös valinnaisia objekteja. Esimerkiksi kaikkien lineaaristen CANopen absoluuttianturien tulee toteuttaa CiA 406 -standardin määrittelemät pakolliset objektit ja jos sovelluskohteessa käytetään vain pakollisia objekteja on anturi vaihdettavissa periaatteessa minkä tahansa valmistajan lineaariseen absoluuttianturiin.

Laiteprofiileita ovat esimerkiksi CiA 401, CiA 402, CiA 404, CiA 406, CiA 408, CiA 410. Nämä ja muutama muu oleellinen CANopen-standardi on kuvattu taulukossa 3.4.

Standardi	**Sisältö**
CiA 301	Sovelluskerroksen ja liikennöintiprofiilin määrittely. Sisältää mm. protokollien määrittelyn.
CiA 305	LSS-palveluiden määrittely.
CiA 306	EDS- ja DCF-tiedostojen määrittelyt.
CiA 311	Uudempien XML-muotoisten tiedostojen määrittelyt.
CiA 401	Laiteprofiili I/O-laitteille.
CiA 402	Laiteprofiili taajuusmuuttajille ja servo-ohjaimille.
CiA 404	Laiteprofiili mittalaitteile ja yksikkösäätimille.
CiA 406	Laiteprofiili lineaarisille ja pyöriville pulssi- ja absoluuttiantureille.
CiA 408	Laiteprofiili hydraulikäytöille ja hydrostaattisille voimansiirroille, sekä proportionaaliventtiileille.
CiA 410	Laiteprofiili painovoimaan perustuville, 16- ja 32-bittisille kallistusantureille.

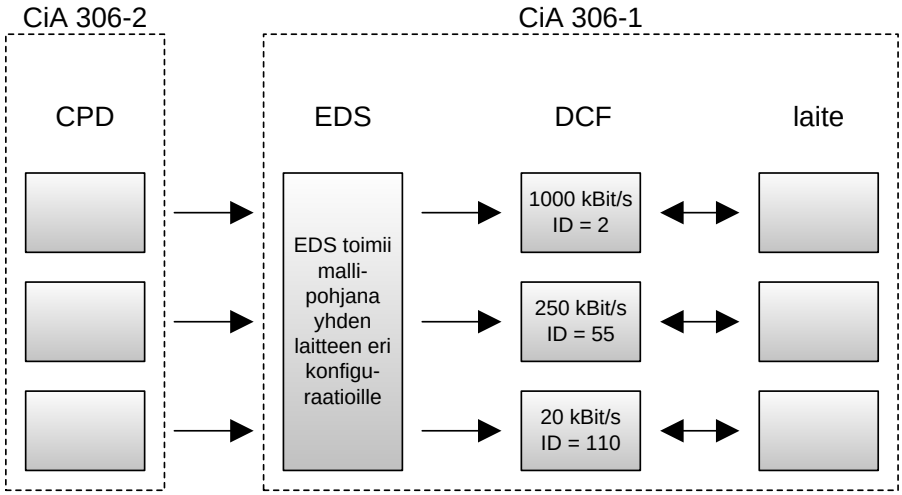
Taulukko 3.4: Osa CiA standardeista ja laiteprofiileista. Poimittu lähteestä [17].

3.5 Laitekonfiguraation hallinta

CAN-laitteiden käyttö verkossa vaatii laitteiden kommunikointi- ja konfigurointiparametrien muokkaamista. CANopen tarjoaa standardin tavan näiden parametrien säätöön objektikirjaston avulla. Monimutkaisten CANopen järjestelmien konfigurointiin on olemassa CANopen ohjelmistoja. Niiden avulla suunnittelu-, konfigurointi- ja analysointiprosesseja saadaan yksinkertaistettua ja luotettavuutta parannettua [19, s.4].

Ohjelmistotyökalut tarvitsevat kuitenkin sähköisen kuvauksen laitteista. Jotta voitaisiin käyttää valmistajariippumatonta konfigurointityökalua on käytettävä yhteensopivaa tiedostoformaattia laitteiden oletusarvojen ja parametrien kuvaamiseen. CAN in Automation (CiA) on määritellyt CiA 306-1 -standardin [20], joka kuvaa

sähköisen datalehden (EDS) tallennusformaatin. Sama standardi kuvaa myös laitekonfiguraation tallentamiseen käytettävän DCF-formaatin, joka sisältää datalehden tietojen lisäksi kyseiselle verkolle konfiguroidut asetukset (kts. kuva 3.4). Toisin sanoen EDS kuvaa yleisesti laitteen objektikirjaston rakenteen ja DCF kuvaa yksittäisen solmun konfiguroinnin yhden verkon yhteen positioon [15].



Kuva 3.4: CPD-, EDS- ja DCF-tiedostojen käyttö konfigurointiprosessissa. Mukailtu lähteistä [19, s.8] ja [21, s.5].

3.5.1 CPD-tiedostot ja CANopen-profiilitietokanta

CAN in Automation on määritellyt standardissaan CiA 306-2 CANopen-profiilitietokannat, joita kutsutaan myös lyhenteellä CODB. Nämä profiilitietokannat voidaan tallentaa tekstimuotoiseen CPD-tiedostoon ja ne sisältävät laiteprofiilien (esim. CiA 401) kaikki objektikirjaston määrittelyt koneluettavassa muodossa. CPD-tiedostoja voidaan käyttää esimerkiksi EDS-tiedostojen koostamiseen ja validointiin laiteprofiileja vastaan. [21]

CPD-tiedosto noudattaa CSV-formaatista mukailtua rakennetta ja sisältää kaksoispisteellä eriteltyjä kenttiä. Ensimmäisenä kenttänä toimii objektikirjaston indeksi, seuraavana ali-indeksi jonka jälkeen on määritelty kyseisen objektin ominaisuuksia. Kuvassa 3.5 on esitetty CiA 401 laiteprofiilin mukainen kuvaus objektikirjaston objektista 0x6000.

```
6000::Read Input 8-Bit: conditional::ARRAY:m::m::n:1:m:255:m::m::m::m::m:ValOf(0x1000)&0x10000
6000:00:Number of Elements:optional::VAR:m:UNSIGNED8:m:ro:m::m::m:::::d[0...254]:n:m
6000:01:DigInput8_1:conditional::VAR:m:UNSIGNED8:m:ro:d[ro,rwr]::m::m:::::::y:m: ValOf(0x1000)&0x10000
6000:02:DigInput8_2:optional::VAR:m:UNSIGNED8:m:ro:d[ro,rwr]::m::m:::::::y:m
```

Kuva 3.5: Esimerkki CPD-tiedoston sisällöstä. Lähde [21, s.23].

CiA 306-2 määrittelee myös tavan linkittää samansisältöiset objektit keskenään.

Kuvassa 3.6 on esitetty kuinka linkitys tapahtuu. Ensin on määritelty objektin 0x1A00 sisältö ja sen aliobjektion 0x1A00:00 ja 0x1A00:01 sisältö. Jos objektit 0x1A01, 0x1A02, 0x1A03 ja 0x1A04 sisältävät täysin samat määrittelyt voidaan ne merkitä kuten kuvassa eli dollarimerkillä alkavassa indeksissä sisältö on sama kuin yhtäsuuruusmerkin jälkeisessä indeksissä. Myös näiden objektien ali-indekseillä olevat objektit vastaavat nyt automaattisesti toisiaan.

```
1A00::Transmit PDO Mapping Parameter:optional::RECORD:d[RECORD,ARRAY,DEFSTRUCT]::n::n:1:m:65:m::n::n::n::n
1A00:00:Number of Entries:optional::VAR:m:UNSIGNED8:m:rw:d[rw,ro]::n::n:0:d[,0]:64:d[1...64]::d[1...64]:n:m
1A00:01:PDO Mapping Entry:optional::VAR:m:UNSIGNED32:m:rw:d[rw,ro]::n::n:::::::n:m
$1A01=1A00
$1A02=1A00
$1A03=1A00
$1A04=1A00
```

Kuva 3.6: Esimerkki CPD-tiedoston sisällöstä. Lähde [21, s.24].

Laajempi kuvaus CPD-formaatista eli CODB-tallennusformaatista ja eri kenttien merkityksestä löytyy CiA 306-2 -standardista [21].

3.5.2 EDS-tiedosto sähköisen datalehden tallennukseen

EDS eli Electronic data sheet on sähköinen kuvaus CANopen-laitteesta. Monet laitevalmistajat tarjoavat laitteidensa EDS-tiedostot ladattavaksi suoraan nettisivuiltaan. Jos valmistaja ei tarjoa EDS-tiedostoa, voidaan käyttää laiteprofiilin mukaista oletus EDS-tiedostoa. Oletuslaiteprofiilin EDS saattaa kuitenkin erota laitteeseen konkreettisesti toteutetuista ominaisuuksista, ja tästä voi seurata vakavia ongelmia [19, s.8]. Standardoitu laitekuvaus (EDS) ja konfiguraatio (DCF) mahdollistavat yleiskäyttöisten valmistajariippumattomien työkalusovellusten käytön CANopen-laitteiden konfigurointiin, CANopen-verkkojen suunnitteluun ja CANopen-projektien tiedonhallintaan.

EDS sisältää tuetut kommunikointiasetukset ja CiA 301 -standardissa ja kuvatut objektikirjaston objektit. Näiden lisäksi EDS sisältää myös laitetyyppikohtaiset objektit, jotka on kuvattu kyseisen laitetyypin laiteprofiilissa [19, s.8], sekä valmistajan määrittelemät objektit. EDS siis toimii mallipohjana laitteen konfiguraatille ja sen perusteella voidaan luoda verkkokohtaisia konfiguraatioita, jotka voidaan tallentaa DCF tiedostoiksi, kuten kuvassa 3.4 on esitetty.

EDS:n tallennusformaatti perustuu Windows INI -tiedoston rakenteeseen eli koostuu lohkoista ja avain-arvo-pareista. Lohkon nimi on hakasulkeiden sisällä, kuten kuvan 3.7 *[FileInfo]*. Lohkoon kuuluvat sen alapuolelle kirjoitetut avain-arvo -parit aina seuraavaan lohkonimeen asti. Formaattia on myös rajoitettu CiA 306 -standardissa [19, s.10] määrittämällä merkistöksi ISO646 sekä rajoittamalla rivien pituus 255 merkkiin. Lohkon nimet, kuten avaintenkin nimet, esimerkiksi *FileName* (kts. kuva 3.7), voidaan kirjoittaa isolla tai pienellä kirjainkoolla. Arvot voivat olla

merkkijonoja tai numeroita. Merkkijonoja ei kirjoiteta lainausmerkkeihin. Numeeriset arvot voivat olla kuvattuina desimaaleina tai hexadesimaaleina, jolloin ne alkavat merkeillä 0x, esimerkiksi 0x0a tai 0xFF3C. EDS ja DCF sisältävät versiotiedot itsestään, tiedot laitteesta ja valmistajasta, sekä tiedot laitteen objektikirjastosta.

Versiotiedot. EDS:n itsestään sisältämät versiotiedot on kuvattu *FileInfo*-lohkon alle. Esimerkki *FileInfo*-lohkon sisällöstä on esitetty kuvassa 3.7. *FileName* kuvaa tiedostonimen. *FileVersion* ja *FileRevision* kuvaavat versionumerot 8-bittisellä kokonaisluvuilla. *EDSVersion* kuvaa käytettävän CANopen-standardin version formaatissa "x.y". *Description* sisältää lyhyen kuvauksen (max 243 merkkiä) tiedostosta. *CreationTime* sisältää tiedoston luontiajan muodossa "hh:mm(AM|PM)" ja *CreationDate* luontipäivän muodossa "mm-dd-yyyy". *CreatedBy* avaimeen voidaan tallentaa tiedoston luoneen henkilön nimi. *ModificationTime*, *ModificationDate* ja *ModifiedBy* sisältävät samat tiedot viimeisen muokkauksen osalta. [19, s.11]

```
[FileInfo]
FileName=vendor1.eds
FileVersion=1
FileRevision=2
EDSVersion=4.0
Description=EDS for simple I/O-device
CreationTime=09:45AM
CreationDate=05-15-1995
CreatedBy=Zaphod Beeblebrox
ModificationTime=11:30PM
ModificationDate=08-21-1995
ModifiedBy=Zaphod Beeblebrox
```

Kuva 3.7: Esimerkki EDS-tiedoston FileInfo-lohkon sisällöstä. Lähde [19, s.12].

Laitetiedot. EDS:n laitteesta sisältämät yleiset tiedot on kuvattu *DeviceInfo*-lohkon alle. Esimerkki *DeviceInfo*-lohkon sisällöstä on esitetty kuvassa 3.8. *VendorName* kuvaa laitteen myyjän nimen ja *ProductName* laitteen nimen. *VendorNumber* kuvaa myyjän tunnisteen ja *ProductNumber* laitteen tunnisteen. *RevisionNumber* kuvaa laitteen versionumeron. Tunnisteet ja versionumero on kuvattu 32-bittisillä kokonaisluvuilla. *OrderCode* on tilauskoodi, joka on kuvattu merkkijonolla. Tuetut ominaisuudet on kerrottu booal arvoin (1=tuettu, 0=ei tuettu). *LSS_Support* kuvaa onko LSS (Layer Settings Services) tuettu (kts. alakohta 3.3.6). *BaudRate* kuvaa tuetut bittinopeudet. Kuvan 3.8 esimerkissä laite tukee bittinopeuksia 50, 250, 500 ja 1000kbit/s. *SimpleBootUpSlave* ja *SimpleBootUpMaster* kuvaavat tukeeko laite boot-up-protokollaa (kts. kohta 3.2). *NrOfRxPdo* ja *NrOfTxPdo* kuvaavat laitteen tukemien RPDO- ja TPDO-viestien määrän (kts. alakohta 3.3.1). [19, s.12-13]

Objektikirjasto. EDS-tiedostossa kuvataan mitä objektikirjaston objekteja laite tukee sekä minimi- ja maksimi-arvot, oletusarvot ja datatyypit objektien arvoille. Tuetut objektit on jaettu kolmeen ryhmään: pakolliset objektit, valinnaiset objektit ja valmistajan määrittelemät objektit. Pakolliset objektit on kirjattu *Mandatory-*

```
[DeviceInfo]
VendorName=Nepp Ltd.
VendorNumber=156678
ProductName=E/A 64
ProductNumber=45570
RevisionNumber=1
OrderCode=BUY ME - 177/65/0815
LSS_Supported=0
BaudRate_50=1
BaudRate_250=1
BaudRate_500=1
BaudRate_1000=1
SimpleBootUpSlave=1
SimpleBootUpMaster=0
NrOfRxPdo=1
NrOfTxPdo=2
```

Kuva 3.8: Esimerkki EDS-tiedoston DeviceInfo-lohkon sisällöstä. Lähde [19, s.13].

Object-lohkon alle. Listan alussa on *SupportedObjects*-avain jolla kuvataan montako objektia on yhteensä ja sen alle on listattu objektit kuten kuvassa 3.9 on esitetty. Valinnaiset objektit on listattu samaan tapaan *OptionalObjects*-lohkoon ja valmistajan määrittelemät objektit *ManufacturerObjects*-lohkoon. CANopen määrittelee pakollisiksi objekteiksi objektikirjaston indekseissä 0x1000 ja 0x1001 olevat objektit, sekä CANopen 4.0 mukaan 0x1018. Valinnaiset objektit ovat objektikirjaston indekseillä välillä 0x1000 - 0x1FFF ja 0x6000 - 0xBFFF. Valmistajan määrittelemät objektit ovat aina indekseillä 0x2000 - 0x5FFF. [19, s.14-15]

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

Kuva 3.9: Esimerkki EDS-tiedoston objektien listauksesta.

Jokaisen listatun objektin tiedot on vielä erikseen kuvattu tarkemmin omissa lohkoissaan. Lohkon nimenä toimii suoraan objektin indeksi. Esimerkki objektin 0x1003 kuvauksesta on kuvassa 3.10. *SubNumber* kuvaa objektin todellisten aliobjektien määrän. Aliobjektit on kuvattu myös omissa lohkoissaan. Ensimmäinen aliobjekti on tässä tapauksessa lohkossa 1003sub0 ja seuraava lohkossa 1003sub1 jne. Vaikka *SubNumber* kuvaakin aliobjektien määrän, se ei välttämättä ole suurin ali-indeksi, koska ali-indeksejä voi puuttua välistä. Niinpä suurin ali-indeksi on kuvattu ensimmäisen aliobjektin 1003sub0 *DefaultValue*-arvossa. Tässä tapauksessa sen arvo on 1.

Objektit voivat esiintyä myös ilman aliobjekteja. Objektit ja niiden mahdolliset aliobjektit sisältävät saman rakenteen mukaiset tiedot. *ParamenterName* kuvaa objektin eli parametrin nimen. *ObjectType* kuvaa objektin tyypin objektikirjaston määritelmän mukaisesti (kts. alakohta 3.1.2). *DataType* kertoo datan tyypin objektikirjaston määritelmän mukaisesti (kts. alakohta 3.1.3). *LowLimit* ja *HighLimit*

määrittelevät minimi- ja maksimirajat objektin arvolle. *AccessType* määrittää objektin käyttöoikeudet objektikirjaston määritelmän mukaisesti (kts. alakohta 3.1.4). *DefaultValue* määrittelee oletusarvon kyseiselle objektille. *PDOMapping* kuvaa voiko objektin sitoa PDO-viestiin (1=kyllä, 0=ei). [19, s.15-16] Itse objektin arvoa ei ole EDS-tiedostossa määritelty, koska se riippuu laitteen sovelluskohteesta ja EDS määrittelee vain laitteen yleiset ominaisuudet. Verkko tai sovelluskohteesta riippuvat asetukset määritellään vasta DCF-tiedostossa.

```
[1003]
SubNumber=2
ParameterName=Pre-defined Error Field
ObjectType=8

[1003sub0]
ParameterName=Number of Errors
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=0x1
PDOMapping=0

[1003sub1]
ParameterName=Standard Error Field
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=0x0
PDOMapping=0
```

Kuva 3.10: Esimerkki EDS-tiedoston objektikirjastosta ja indeksin sekä ali-indeksin käytöstä. Lähde [19, s.17].

Työkalujen linkitys. EDS sisältää valinnaisen *Tools*-lohkon, johon laitevalmistaja voi halutessaan linkittää laitekohtaisia työkaluja tai ohjelmia. *Tools*-lohko kopioidaan myös DCF-tiedostoon, kun se luodaan EDS-tiedostosta. *Tools*-lohko sisältää *Items*-avaimen jonka arvo kertoo työkalujen määrän. Jokainen työkalu on vuorostaan kuvattu omassa numeroidussa lohkossaan *Tool1*, *Tool2*, *Tool3*, jne. Jokaisella työkalulla on kaksi avainta. *Name*-avain kertoo työkalun nimen ja *Command*-avain sisältää ajettavan komennon nimen ja parametrit. [22, s.11] Yhden työkalun sisältävä esimerkki *Tools*-lohkosta on esitetty kuvassa 3.11.

```
[Tools]
Items=1

[Tool1]
Name=convert DCF to DTY
Command=DCF2DTY $DCF $NAME.DTY
```

Kuva 3.11: Esimerkki työkalujen linkityksestä EDS-tiedostoon. Lähde [22, s.12].

Työkalun *Command*-avaimessa voidaan myös viitata tietyllä avainsanoilla tiedostojen nimiin ja solmun tietoihin. Avainsanat ja niiden merkitys on esitetty taulukossa 3.5. Näin esimerkiksi DCF- tai EDS-tiedoston polku voidaan välittää ajettavalle

työkalulle, ja työkalu pääsee näin käsiksi näiden tiedostojen sisältöön.

Avainsana	**Kuvaus**
$DCF	DCF-tiedoston polku.
$EDS	EDS-tiedoston polku.
$NODEID	Solmun Node ID.
$NAME	Valmistajan määrittelemä laitenimi solmulle. Kuvattu myös objektikirjaston indeksissä 0x1008.
$NETID	Solmun Net ID.
$NETNAME	Solmun CANopen-verkon nimi.

Taulukko 3.5: Avainsanat EDS-tiedoston [Tools]-lohkossa. Mukailtu lähteestä [22, s.12].

3.5.3 DCF-tiedosto laitekonfiguraation tallennukseen

DCF eli Device configuration file kuvaa EDS-tiedoston esittämän laitekonfiguraation objektit, mutta lisäksi myös niiden arvot. Näiden ohella se sisältää tiedon verkossa käytettävästä bittinopeudesta ja laitteen Node ID -tunnisteen. [19, s.8] Tiedostoformaatiltaan DCF noudattaa samaa INI-tiedoston rakennetta kuin EDS-tiedostokin. DCF siis sisältää laitteen konfigurointitiedon kyseiseen käyttökohteesen, kun taas EDS kuvaa laitteen ominaisuudet.

DCF sisältää kaikki konfiguroidun laitteen objektit. Rakenteeltaan tiedosto vastaa EDS-tiedostoa, mutta sisältää muutaman lisäparametrin, joilla kuvataan laiteen konfiguraatiota. *FileInfo*-lohkoon (kts. kuva 3.7) on lisätty *LastEDS*-parametri joka sisältää EDS-tiedoston nimen, jota on käytetty laitteen DCF-tiedoston luomisen pohjana. Objektien lohkoihin (kts. kuva 3.10) on lisätty *ParameterValue*, joka kuvaa objektin arvon objektityypin ja datatyypin mukaisessa formaatissa. [19, s.22]

DCF sisältää *Device Comissioning* -lohkon, jota EDS-tiedostosta ei löydy. Lohko määrittelee CANopen-verkkoon liittyviä solmun asetuksia. Kuvassa 3.12 on esitetty esimerkki *Device Commissioning* -lohkon sisällöstä. *NodeID* määrittelee solmun tunnisteen CANopen-verkossa. *NodeName* määrittelee solmulle tekstimuotoisen nimen. *Baudrate* kuvaa verkossa käytetyn bittinopeuden. *NetNumber* on 32-bittinen tunnistenumero CANopen-verkolle. *NetworkName* kertoo verkon tekstimuotoisen nimen. *CANopenManager*-parametri on valinnainen ja määrittelee onko kyseinen CANopen laite CANopen-manageri (1=kyllä, 0=ei). Jos parametri puuttuu kokonaan silloin laite ei ole CANopen-manageri. [19, s.24] CANopen-managerilla tarkoitetaan solmua missä on vähintään seuraavat ominaisuudet: NMT-master, SDO-master ja somujen konfiguraation hallinta. CANopen-master voi myös olla SYNC-tuottaja, TIME-tuottaja tai LSS-master. Yksinkertaisissa järjestelmissä voidaan käyttää pelkkää NMT-masteria, eikä niissä välttämättä tarvita CANopen-manageria. [15] *LSS_SerialNumber* kuvaa 128-bittisen LSS-sarjanumeron (esitelty

tarkemmin alakohdassa 3.3.6), joka löytyy myös objektikirjastosta objektin 0x1018 ali-indekseiltä 1-4.

```
[DeviceComissioning]
NodeID=2
NodeName=DEVICE2
Baudrate=1000
NetNumber=42
NetworkName=very important subnet in a big network
LSS_SerialNumber=9912345
```

Kuva 3.12: Esimerkki CANopen-laitteen DCF-tiedoston verkkokonfiguraatiosta. Lähde [19, s.25].

3.5.4 CPJ-tiedosto laitekonfiguraatioden linkittämiseksi projekteiksi

Nodelist.cpj-tiedosto sisältää kaikki verkon solmut ja viittauksen niiden konfiguraatiotiedostoon. Kun solmujen DCF-konfiguraatiotiedostot ja Nodelist.cpj ovat samassa hakemistossa ja tarpeelliset tiedot on määritelty Nodelist.cpj-tiedostoon, on konfiguraatiotyökalujen helppo automaattisesti käsitellä verkon solmujen EDS- ja DCF-tiedostoja. [22, s.10]

Nodelist.cpj sisältää *Topology*-lohkon, jossa määritellään yhteen verkkoon kuuluvat solmut. Kuvassa 3.13 on esitetty esimerkki *Topology*-lohkon sisällöstä. *EDSBaseName* kuvaa hakemiston jossa laitteiden EDS-tiedostot sijaitsevat. Tämän tiedon ja DCF-tiedostoissa olevan tiedon perusteella voidaan löytää laitteen EDS-tiedosto. *NetName* kertoo verkon nimen. *Nodes* kuvaa verkossa olevien solmujen määrän. Jokaisesta solmusta on kolme tietoa *NodeXName*, *NodeXPresent* ja *NodeXDCFName*. Näissä X kuvaa solmun Node ID -tunnusta. *NodeXName* kertoo solmun nimen. *NodeXPresent* kuvaa onko solmu läsnä (0x01) verkossa vai ei (0x00). *NodeXDCFName* kertoo solmun DCF-tiedoston nimen ilman hakemistopolkua. [22, s.10]

```
[Topology]
EDSBaseName=C:\MyEDS\DefaultNetEDS
NetName=DefaultNet
Nodes=0x03
Node2Present=0x01
Node2DCFName=demo_plc.dcf
Node2Name=DemoNode_A
Node3Present=0x01
Node3DCFName=demodeva.dcf
Node3Name=DemoNode_B
Node4Present=0x01
Node4DCFName=demodevb.dcf
Node4Name=DemoNode_C
```

Kuva 3.13: Esimerkki Nodelist.cpj-tiedoston sisällöstä. Lähde [22, s.11].

4. CANOPEN-ESIMERKKIAINEISTO, -SUUNNITTELUOHJELMAT JA SEMANTTINEN SUUNNITTELUPROSESSI

Esimerkkiaineistona toimi hydraulisen porapuomin suunnitteluaineisto. Aineisto on hyvin moniteknistä ja liittyy moneen tieteenalaan. Eri suunnitteluohjelmia on käytetty aineiston eri osien tuotantoon ja tarkoitus oli yhdistellä näitä aineistoja linkittämällä niitä semanttisen mallin avulla keskenään. Aineiston linkittämistä ja semanttista prosessia on käsitelty myös julkaisussa [23]. Käytetyt suunnitteluohjelmat ja -aineistot on esitetty taulukossa 4.1.

Osa-alue	**Aineisto**	**Ohjelma**
Hydraulisuunnittelu	Hydraulikaaviot (2D)	Vertex HD
Mekaniikkasuunnittelu	3D mallit	Vertex 4G ja SolidWorks
Sähkösuunnittelu	Sähkökaaviot (2D)	Vertex ED
Väyläsuunnittelu	CANopen laitekonfiguraatiot	ProCANopen
Simulointi	Simulaatiomalli	MATLAB/Simulink
Käsitteidenhallinta	OWL-ontologia	Tekstieditori ja Protégé
Projektinhallinta	Muutostiedot	Trac
Aineiston hallinta	Projektin tiedostot	Vertex PDM

Taulukko 4.1: Lähdeaineistot ja niihin liittyvät ohjelmistot Semogen- ja Semogen II -projekteissa. Mukailtu osittain lähteestä [23].

Aineistosta CANopen-suunnitteluun liittyy suoraan ProCANopen ja sillä tehdyt väyläsuunnittelut, mutta myös Vertex ED-ohjelmalla tehdyt sähkökaaviot, koska ne sisältävät myös tietoa CANopen-laitteista.

Esimerkkiaineisto sisältää useamman väylän CANopen-järjestelmän, mutta CANopen-suunnittelu on tehty väyläkohtaisesti, johtuen mm. suunnitteluohjelman rajoituksista. Esimerkkiaineistossa on tarkalleen kolme eri väylää eli kolme eri ProCANopen-projektia. Vertex ED -ohjelmalla tehdyssä sähkökaaviossa näkyvät CANopen-laitteiden fyysinen kytkentä väylään, sekä miten CANopen-laitteet ja analogiset laitteet on kytketty toisiinsa.

Kaikki suunnitteluaineisto tallennettiin Vertex PDM -järjestelmään, joka toimi projektissa tiedostovarastona. Samalla PDM generoi uusille nimikkeille yksikäsitteiset tunnisteet automaattisesti. Nimikkeillä tarkoitetaan komponentteja tai ko-

koonpanoja, joita voivat olla esimerkiksi puomi, sylinteri tai venttiili. CANopenin tapauksessa nimikkeenä toimi komponenttitasolla yksi solmu ja kokoonpanotasolla itse väylä. Näiden tunnisteiden avulla pystyimme linkittää aineistoa eri lähteistä yhteen semanttiseen malliin. PDM:ään voitiin myös luoda hierarkkisia rakenteita, joilla komponenteista voitiin koostaa kokoonpanoja eli liittää nimikkeitä kuulumaan toisiin nimikkeisiin.

Aineistossa olevat komponentit pitää järjestelmän laajuisesti tunnistaa, jotta aineiston myöhempi yhdistäminen olisi mahdollista. Eri ohjelmat käyttävät eri sisäisesti eri tunnistetyyppejä. Vertex PDM käyttää nimikekoodia esim. *VX3000316*, ProCANopen Node- ja Net ID:tä esim. *12* ja *3* ja Vertex ED -viitetunnusta esim. *Y12*. Yhdistäminen päätettiin tehdä lisäämällä PDM-järjestelmän generoima tunniste päätettiin CANopen-suunnitteluaineistossa oleviin solmuihin ja CANopen-laitteiden Node ID- ja Net ID -tunnisteet päätettiin liittää sähkökaavioihin. Tämä vaati tiettyjä muutoksia sekä CANopen- että sähkösuunnitteluprosessiin. Näitä muutoksia on esitelty seuraavissa kohdissa.

4.1 CANopen-suunnitteaineiston rikastus ProCANopen-ohjelmalla

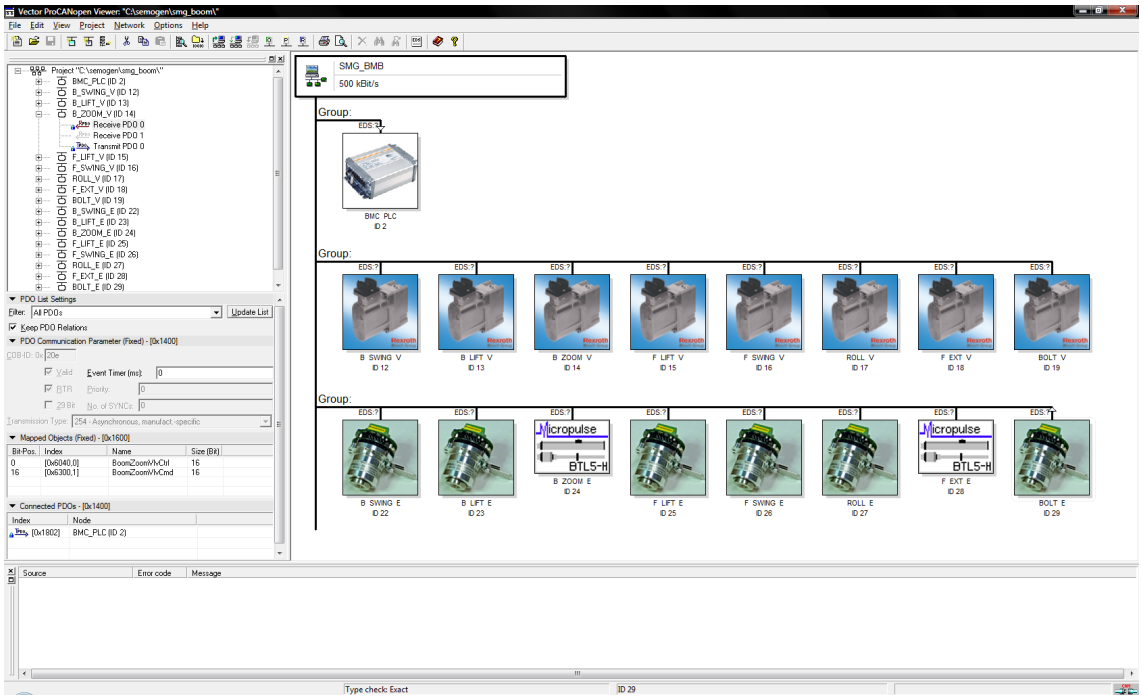
CANopen-suunnittelu on toteutettu tässä esimerkkitapauksessa ProCANopen-ohjelmalla. ProCANopen on Vector-yrityksen kehittämä ohjelma, jota käytetään CANopen-verkkojen suunnitteluun, testaukseen, diagnosointiin ja CANopen-laitteiden ohjelmointiin. Ohjelmalla voidaan suunnitella koko CANopen-verkko ja konfiguroida sen yksittäiset solmut, niiden parametrit ja objektikirjastot. [24]

ProCANopen-projektit koostuvat pco.ini-tiedostosta, Nodelist.pco-tiedostosta, solmujen DCF- ja EDS-tiedostoista ja prjdb.dbc-tiedostosta [25]. Pco.ini sisältää projektin asetukset ja verkon somujen Node ID -tunnisteet, nimet ja EDS- ja DCF-tiedostojen nimet. Myös Nodelist.pco sisältää solmujen Node ID -tunnisteet ja DCF-tiedostojen nimet. Samat tiedot kuin standardin mukaisessa Nodelist.cpj-tiedosto olisi saatavilla. Nodelist.pco-tiedostosta onkin mahdollista generoida standardin mukainen Nodelist.cpj-tiedosto suhteellisen helposti. Solmujen EDS- ja DCF-tiedostot ovat CANopen-standardin mukaisia. Prjdb.dbc on verkon signaalitietokanta, joka voidaan generoida DCF-tiedostojen perusteella. Se sisältää solmujen nimet ja tietoa verkossa liikkuvista viesteistä.

Oleellinen aineisto tämän työn kannalta on Nodelist.pco-tiedosto ja solmujen DCF-tiedostot. Nodelist.pco-tiedoston perusteella saadaan selville verkkoon kuuluvat solmut ja niiden DCF-tiedostot ja DCF-tiedostojen perusteella voidaan selvittää solmujen kaikki asetukset. Aineisto sisältää eri tyyppisiä solmuja, kuten venttiileitä, antureita ja PLC-laitteita (ohjelmoitavia logiikoita). Osa PLC-solmuista toimii

myös siltaavina laitteina kahden CANopen-väylän välillä ja niiden avulla kommunikaatio eri väylien välillä onnistuu. Nämä laitteet löytyvät siis kaksi kertaa molempien väylien suunnitteluaineistosta, ja tämä on huomioitava myöhemmin aineistoa linkitettäessä.

Rakenteeltaan CANopen-aineisto koostuu kolmesta väylästä: puomiväylästä, ohjausväylästä ja pääväylästä. Puomiväylällä on puomiin liittyvät anturit ja venttiilit, ohjausväylällä ohjaamoon liittyvät joystickit ja ohjaimet, sekä hydrauliikan tehoyksikkö ja pääväylä yhdistää nämä kaksi väylää ja sisältää myös ohjelmoitavaa logiikkaa. Kuvassa 4.1 näkyy puomiväylän suunnitteluaineisto avattuna ProCANopen-ohjelmaan.



Kuva 4.1: Puomiväylän suunnitteluaineisto ProCANopen-ohjelmassa.

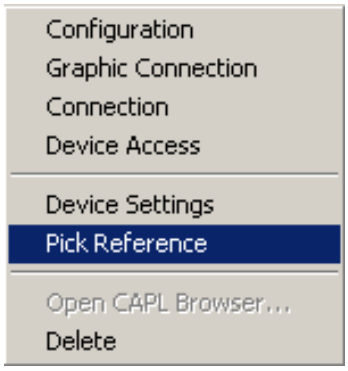
Solmujen DCF-tiedostojen linkitys muuhun aineistoon ratkaistiin lisäämällä DCF-tiedoston *DeviceCommissioning*-lohkoon *RefDesignator*-avain, jonka arvoksi laitettiin solmun PDM-järjestelmässä saama tunniste. Kuvassa 4.2 on esitetty muokattu DCF-tiedoston osa. *RefDesignator*-avainta ei löydy vielä CANopen-standardeista, mutta sitä on esitetty osaksi CiA 306 -standardia.

RefDesignator-avain voidaan lisätä käsin INI-muotoiseen DCF-tiedostoon tai sitä varten voidaan tehdä erillinen työkalu, jolla tunnus voidaan poimia olemassaolevien tunnusten joukosta suoraan ProCANopen-ohjelmassa. Tällainen työkalu on myös toteutettu. Ulkopuolisten työkalujen linkitys on mahdollista lisäämällä DCF-tiedoston *Tools*-lohkoon työkalun nimi ja suoritettava tiedosto, kuten kuvassa 3.11 on esitetty. ProCANopen-ohjelmassa DCF-tiedostoihin lisätyt työkalut näkyvät kätevästi solmun kohdalla oikealla hiiren näppäimellä avautuvassa ponnahdusvalikossa (kuvassa

```
[DeviceComissioning]
NodeID=12
NodeName=B_SWING_V
Baudrate=500
NetNumber=6
NetworkName=SMG_BMB
RefDesignator=VX3000326
```

Kuva 4.2: Esimerkki RefDesignator-arvon lisäämisestä osaksi CANopen-laitteen DCF-tiedoston verkkokonfiguraatiota.

4.3) ja työkalu voidaan ajetaan sen nimeä klikkaamalla.



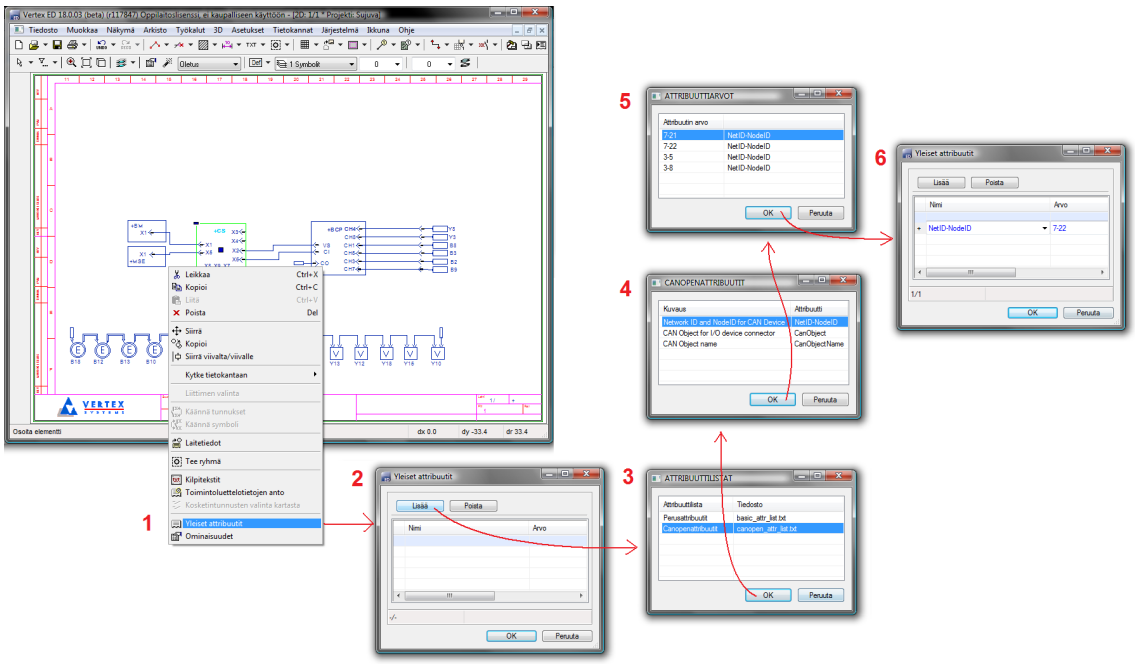
Kuva 4.3: ProCANopen-ohjelman solmukohtainen ponnahdusvalikko.

4.2 Sähkösuunnitteluaineiston rikastus Vertex ED -ohjelmassa

Vertex ED on Vertex-yrityksen tekemä sähkösuunnitteluohjelma, jolla voi tehdä sähkökaavioita. Ohjelmasta on mahdollista saada export-toiminnon avulla SVG-muotoinen kuva sähkökaaviosta, joka sisältää rikastettua tietoa laitteista ja niiden kytköksistä toisiinsa. Tätä tietoa voidaan myöhemmin hyödyntää linkitettäessä aineistoa semanttiseen malliin tai muuhun aineistoon.

Esimerkkiaineistona toimii puomiväylän sähkökaavio, joka sisältää CANopen-laitteet, sekä muutaman analogilaitteen. Kuvassa 4.4 on esitetty kyseinen sähkökaavio avattuna Vertex ED -ohjelmaan. Jotta kaaviossa olevat laitteet voitaisiin linkittää muuhun aineistoon on niihin lisättävä jonkinlaiset tunnisteet. Tunnistamista varten sähkökaavioon lisättiin CANopen-laitteiden Node ID- ja Net ID -tunnisteet. Tiedot lisättiin Vertex ED -ohjelman *Yleiset attribuutit* -toiminnon avulla kuvassa 4.4 esitettyjen vaiheiden mukaan. Ensin valitaan oikea laite ja valitaan hiiren oikealla näppäimellä pikavalikosta *Yleiset attribuutit*. Vaiheessa kaksi lisätään uusi attribuutti. Vaiheessa kolme valitaaan attribuuttilistaksi *Canopenattribuutit*. Neljännessä vaiheessa aukeaa valintaikkuna jossa näkyy *Network ID ja Node ID for CAN Device*. Tämä on automaattisesti generoitu lista CANopen-aineiston laitteiden Node ID- ja Net ID -tunnuksista. Vaiheessa viisi näkyvä lista sisältää vain aineistosta löytyvät laitteet, joten linkitystä olemattomiin laitteisiin ei voi vahingossa

tehdä. Vaiheessa kuusi näkyy kuinka valitut Node ID- ja Net ID -tunnukset näkyvät laitteen yleisissä attribuuteissa.



Kuva 4.4: Yksikäsitteisen laitteen tunnistetiedon lisääminen Vertex ED-ohjelmassa osaksi sähkökaaviota [26].

Valintaikkunassa näkyvä Node ID- ja Net ID -tunnusparilista generoituu automaattisesti ajamalla putkilinjassa (kts. alakohta 5.2.1) Python-ohjelmakoodi, joka käy CANopen-aineiston läpi ja poimii tekstimuotoiseen tiedostoon kaikki sieltä löytyvien laitteiden tunnisteet ja tiedot. Node ID- ja Net ID -tunnusparin lisäksi vaiheeseen neljä lisättiin *CAN Object for I/O device connector* -lista, josta voidaan valita CANopen I/O -laitteen liittimeen linkittyvä objektikirjaston objekti. *CAN Object name* -attribuutin avulla voidaan tällä objektille antaa myös kuvaava nimi. Linkityksen avulla voidaan myöhemmin semanttisessa mallissa seurata signaaleiden kulkua analogisten laitteiden ja CANopen-väylässä olevien laitteiden välillä.

Node ID- ja Net ID -tunnusparin sijaan sähkökaaviossa olevat CANopen-laitteet olisi voitu linkittää PDM-tunnusten avulla muuhun aineistoon, mutta Node ID ja Net ID -pari valittiin käyttöön siksi koska nyt CANopen- ja sähkökaavioaineisto voidaan halutessa yhdistää keskenään ilman PDM-ohjelmistoa. Tätä hyödynnetään esimerkiksi topologian generoinnissa (kts. alakohta 5.2.4), jossa PDM-ohjelmiston käytöstä ei saada lisäarvoa.

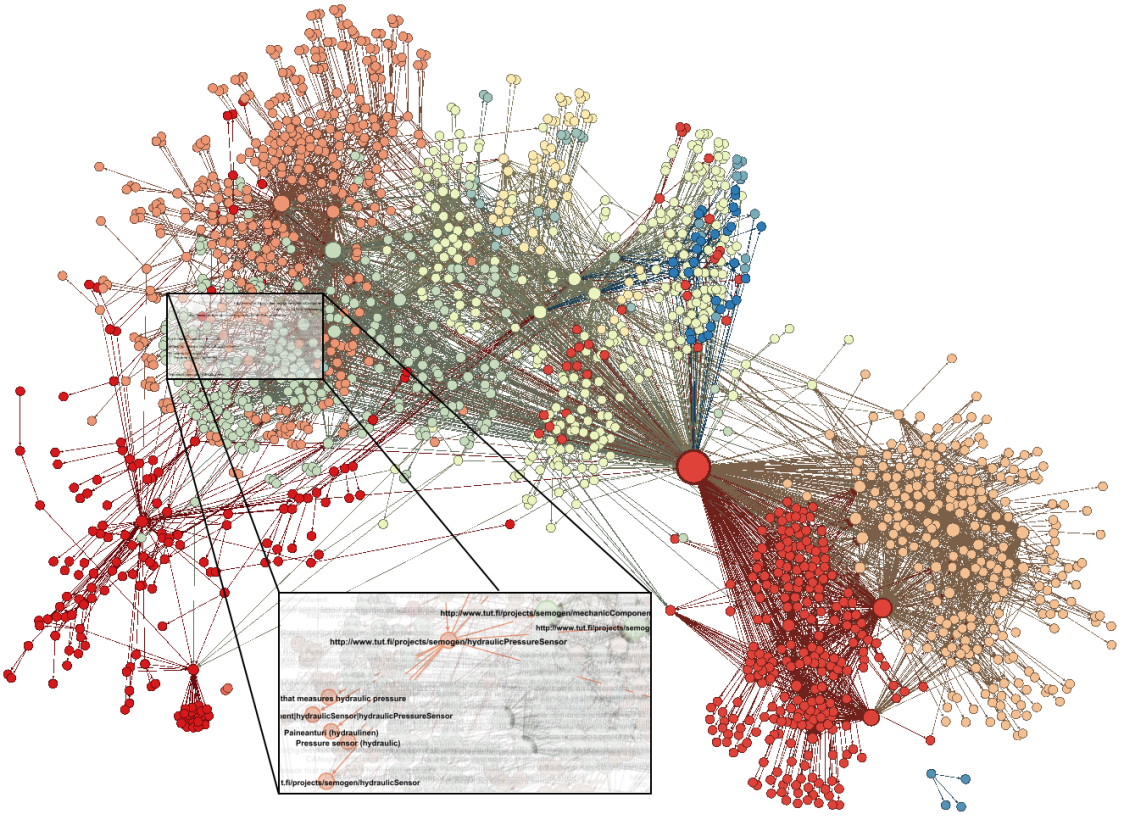
5. CANOPEN-AINEISTON LINKITTÄMINEN SEMANTTISEEN MALLIIN JA MUUHUN SUUNNITTELUAINEISTOON

Alunperin suunnitteluaineisto on hajallaan ja tuotettu useassa keskenään yhteensopimattomassa ohjelmassa. Jotta suunnitteluaineiston kokonaisvaltainen hyödyntäminen olisi mahdollista on siihen päästävä helposti käsiksi yhden yhtenäisen rajapinnan kautta. Semanttisen mallin tarkoituksena on tarjota yksi rajapinta kaikkeen suunnittelutietoon ja mahdollistaa eri lähteistä saatavan suunnittelutiedon keskinäinen linkitys. Semanttinen malli mahdollistaa myös edelleen tiedon rikastamisen, luokittelun ja liittämisen ulkoisiin tietolähteisiin.

Automaattinen tiedon linkittäminen semanttiseen malliin asettaa aineistolle ja suunnittelulle vaatimuksia. Aineiston on oltava koneluettavaa ja sen on sisällettävä koko aineiston laajuisesti yksikäsitteiset tunnisteet yksittäisille komponenteille, jotta nämä voidaan tunnistaa eri lähteistä tulevien aineistojen välillä. Diplomityössä käytetyt suunnitteluohjelmat tuottivat tai niistä sai generoitua koneluettavaa dataa ja sitä rikastettiin tunnisteilla CANopen- ja sähkösuunnitteluaineiston osalta luvussa 4 esitetyllä tavoilla. Koneluettavasta aineistosta edelleen generoitiin semanttinen malli putkilinjan ja eri aineistoihin räätälöityjen scriptien avulla.

5.1 Konejärjestelmän semanttinen malli

Konejärjestelmän koneluettava semanttinen tietomalli tuotetaan automatisoidusti lähdeaineistoista. Semanttisella mallinnuksella tarkoitetaan tietosisällön jäsentämistä ja kuvailua ohjelmallisesti tulkittavassa muodossa [26]. Semantiikka lisää tallennetulle tiedolle merkityksen. Käytännössä tämä on toteutettu metatiedon avulla. Semanttinen tietomalli kasvaa automaattisen generoinnin seurauksena hyvin suureksi kuten kuvasta 5.1 voidaan nähdä. Tietomalli on kuvailtu RDF-muotoisena tietona, joka on mahdollista luokitella OWL-ontologian avulla. RDF itsessään mahdollistaa rikkaan tiedon kuvailun ja ontologian avulla myös kuvaileva metatieto voidaan tarkasti määritellä. Ontologian määritteleminen lisää tiedon yhteensopivuutta ja parantaa sen integroitavuutta muihin järjestelmiin. Näin semanttisesta mallista saadaan koneluettavaa ja helpommin hyödynnettävää ja siitä voidaan etsiä vain tietyn tyyppistä tietoa.



Kuva 5.1: Gephi-ohjelmalla generoitu esitys Semogen II -projektin semanttisesta mallista [26].

5.1.1 RDF-kuvailukieli ja semanttisen mallin rakenne

Resource Description Framework eli RDF on World-Wide Web Consortiumin (W3C) standardoima, erityisesti web-ympäristöihin soveltuva malli semanttisen tiedon mallinnukseen ja vaihtoon. Tiedon mallinnus RDF:ssä pohjautuu kolmikkoihin (engl. triples). Kolmikot sisältävät subjektin, predikaatin ja objektin. Esimerkiksi tieto "Taivas on väriltään sininen" voidaan tallentaa RDF-tietomalliin kolmikkona: subjekti("taivas"), predikaatti("on väriltään"), objekti("sininen") [27].

RDF-kielessä subjektien, predikaattien ja objektien tunnistamiseen käytetään URI-tunnisteita tai vakioita (engl. literals) [28]. Kuvassa 5.2 esitetyssä RDF-muotoisessa semanttisen mallin osassa CANopen-venttiilin yksilöi URI-tunniste *http://smartsimu.rd.tut.fi:8080/item/VX3000323*. CANopen-laitteen Node ID -tunniste on taas esitetty vakioarvona *13*.

RDF-tietomallin käsittelyyn on olemassa paljon valmiita työkaluja eri alustoille. SPARQL-kyselykielen avulla RDF-tietokantaan voidaan tehdä hakuja kuten muihinkin tietokantoihin. RDF-tietomallia ei ole sidottu esitysmuotoon, mutta se voidaan sarjallistaa ja tallentaa useampaan eri formaattiin valmiiden työkalujen avulla. Työssä käytettiin XML-muotoista tiedostoa semanttisen RDF-tietomallin tallennusformaattia eli RDF/XML-sarjallistusta [29]. Muita tunnettuja RDF:n sarjallistuksia

ovat mm. N3 [30], Turtle [31] ja JSON-LD [32]. Esimerkki semanttisessa mallissa olevasta CANopen-venttiilistä XML-muotoisena tiedostona on esitetty kuvassa 5.2.

```
<rdf:RDF
   xmlns:ns1="http://www.can-cia.org/semogen/objectDictionary/"
   xmlns:ns2="http://www.vertex.fi/semogen/"
   xmlns:ns21="http://www.w3.org/2002/07/owl#"
   xmlns:ns23="http://purl.org/dc/elements/1.1/"
   xmlns:ns24="http://www.can-cia.org/semogen/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:semogen="http://www.tut.fi/projects/semogen/">
<!-- ... -->
<rdf:Description rdf:about="http://smartsimu.rd.tut.fi:8080/item/VX3000323">
  <rdf:type rdf:resource="http://smartsimu.rd.tut.fi:8080/classification/component/
  canopenComponent/canopenValve"/>
  <rdf:type rdf:resource="http://smartsimu.rd.tut.fi:8080/classification/component/
  hydraulicComponent/valve"/>
  <ns2:pdmId>1332399809681</ns2:pdmId>
  <semogen:deviceType rdf:resource="http://www.can-cia.org/semogen/408"/>
  <ns23:title>valve</ns23:title>
  <semogen:refDesignator>VX3000323</semogen:refDesignator>
  <ns2:pdmObjectType>item</ns2:pdmObjectType>
  <semogen:NodeName>B_LIFT_V</semogen:NodeName>
  <ns24:hasObjectDictionary rdf:resource="http://smartsimu.rd.tut.fi:8080/item/
  VX3000323/objectDictionary"/>
  <semogen:url rdf:resource="http://smartsimu.rd.tut.fi:8080/vx/pdm/
  pf.jsp?ID=1332399809681.1L,item,null,t"/>
  <ns21:sameAs rdf:resource="http://smartsimu.rd.tut.fi:8080/objects/1332399809681"/>
  <semogen:NodeId>13</semogen:NodeId>
  <semogen:BusId>6</semogen:BusId>
  <semogen:hasParent rdf:resource="http://smartsimu.rd.tut.fi:8080/item/VX3000318"/>
  <semogen:hasParent rdf:resource="http://smartsimu.rd.tut.fi:8080/item/VX134568"/>
</rdf:Description>
<!-- ... -->
</rdf:RDF>
```

Kuva 5.2: Esimerkki CANopen-venttiilin tiedoista RDF-mallissa.

5.1.2 Tiedon luokittelu OWL-ontologian avulla

OWL eli Web Ontology Language on W3C:n kehittämä standardiperhe, jolla on mahdollista tehdä sovelluskohtaisia sanastoja. Sitä pidetään yhtenä Semanttisen Webin keskeisimmistä teknologioista. [33] RDFS eli RDF Schema puolestaan määrittelee tiettyjä luokkia RDF-kielen avulla mahdollistaen ontologioiden kuvaamisen. [34]

Ontologian avulla on myös mahdollista kuvata luokkien ja niiden ominaisuuksien välisiä suhteita ja toisiinsa liittymistä. Nykyisistä suhteista on mahdollista myös generoida automaattisesti uusia suhteita, jos tarpeeksi tietoa on saatavilla. Esimerkkinä tästä projektissa käytettiin Pellet-päättelijää [35] generoimaan käänteisia suhteita luokkien välille. Käänteisiä suhteita ovat esimerkiksi *hasParent*, joka generoitiin *hasChild*-ominaisuudesta automaattisesti lapsiluokalle.

Kuvassa 5.3 on esitetty CANopen-venttiilille määritelty yksinkertainen OWL-luokka XML-muodossa. Luokka sisältää komponenttityypin suomen- ja englanninkielisen nimen sekä tiedon miten luokka sijoittuu luokkahierarkkiassa. Hierarkkinen asema on kuvattu käyttämällä RDFS:n määrittelemää *subClassOf*-attribuuttia.

```
<Class rdf:about="&semogen;canopenValve">
  <rdfs:label>CANopen valve</rdfs:label>
  <rdfs:label xml:lang="fi">CANopen-venttiili</rdfs:label>
  <dc:description></dc:description>
  <semogen:canopenDeviceType></semogen:canopenDeviceType>
  <rdfs:subClassOf rdf:resource="&semogen;canopenComponent"/>
</Class>
```

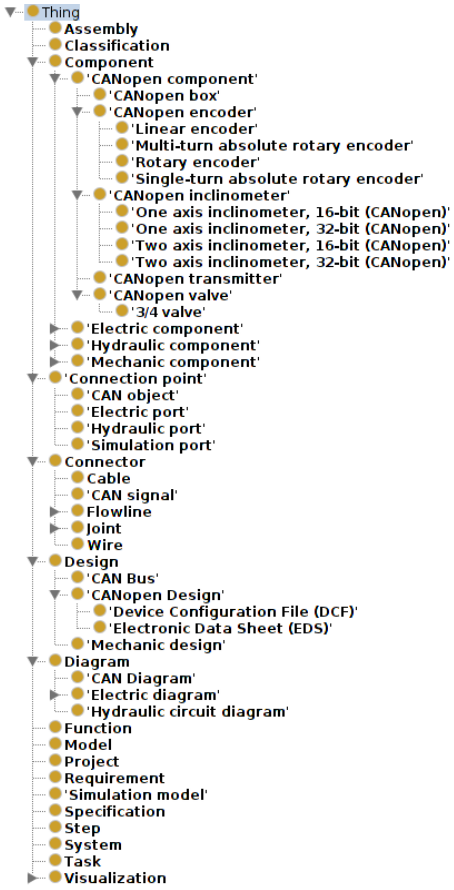
Kuva 5.3: Esimerkki CANopen-venttiilin OWL-luokituksesta.

Komponenttien luokitteluhierarkkiaa ei ollut valmiina nykyisessä aineistossa, joten se piti tuottaa käsin. Kuvassa 5.4 on esitetty työssä tuotettu OWL-luokkahierarkkia. Hierarkkiassa juuritasona toimii aina OWL:n määrittelemä *owl:Thing*. CANopen-aineistoon liittyvät luokat sijoittuvat hajalleen *Component*, *Connection point*, *Connector*, *Design* ja *Diagrams* luokkien alle. Moni näistä luokista on jaettu edelleen CANopen-, sähkö-, hydrauli- ja mekaniikka-tyyppinsä perusteella omiin aliluokkiinsa. Itse CANopen-laitteet kuuluvat *Component/CANopen component* -luokkaan. *Connection point* -luokka käsittää laitteiden liitäntännän toisiin laitteisiin. CANopen-laitteen tapauksessa linkittäminen tapahtuu sitomalla objektikirjaston objekteja PDO-signaaleihin, joten *Connection point* -luokan aliluokaksi on valittu *CAN object*. Itse PDO-signaalit on luokiteltu *Connector/CAN Signal* -luokan alle. Suunnitteluaineisto on ryhmitelty *Design*-luokan alle ja CANopen:n osalta jaettu EDS- ja DCF-tiedostojen perusteella kahteen aliluokkaan. Visualisoitavat kaaviot ovat *Diagram*-luokassa.

5.2 Kehitetyt menetelmät ja työkalut CANopen-aineiston linkittämiseen osaksi semanttista mallia

Semogen- ja Semogen II -projekteissa kehitettiin erilaisia työkaluja lähdeaineiston rikastamiseen ja liittämiseen osaksi semanttista tietomallia. Työkalut tehtiin pääasiassa Python-ohjelmointikielellä ja niitä ajettiin Eclipse-ohjelmointiympäristössä. Alla on esitelty CANopen-väylään liittyvän tiedon koostamiseen tehtyjä työkaluja. Työkaluilla tuotetaa semanttinen malli, joka kokoaa ja linkittää eri tieteenalojen lähdeaineiston yhteen. Myöhemmin on esitetty myös mihin CANopen-väylään liittyvää tietoa käytetään.

Python-pohjaiset työkalut on tehty toimimaan Python version 2.7 kanssa ja käyttävät XML-muotoisten tiedostojen, esimerkiksi SVG-tiedostojen, jäsentämiseen *minidom*-kirjastoa. INI-muotoiset tiedostot, kuten DCF-tiedostot, saadaan jäsennettyä *ConfigParser*-kirjaston avulla ja RDF-muotoisen semanttisen mallin tuottamiseen käytetään *rdflib*-kirjastoa. Työkalujen avulla poimitaan tarvittavia tietoja semanttiseen malliin, sekä linkitetään se olemassaolevaan dataan yksikäsitteisten tunnisteiden avulla.

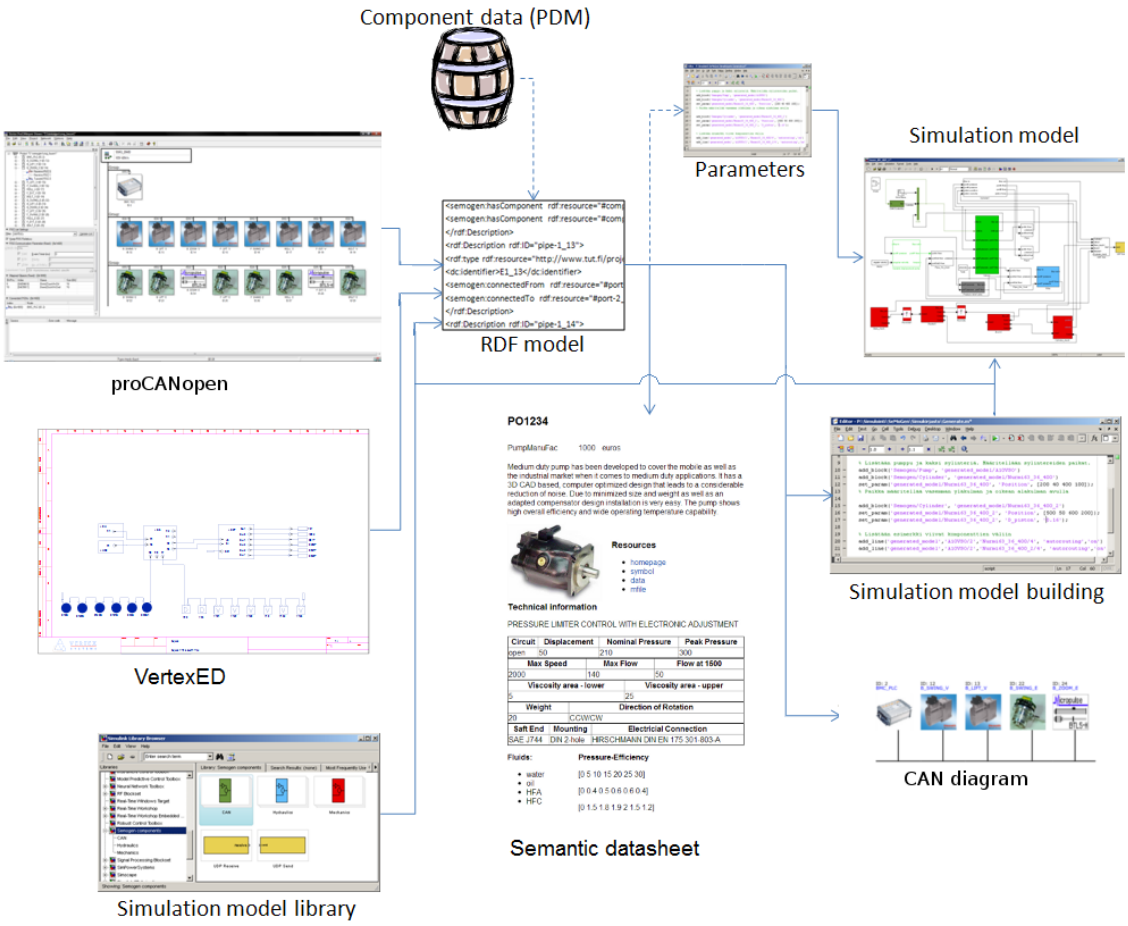


Kuva 5.4: Protégé-ohjelmalla luotu näkymä Semogen II -projektin OWL-luokkahierarkkiasta.

5.2.1 Putkilinja

Semanttisen mallin generoinnin ytimenä toimii niin sanottu putkilinja. Putkilinjalla tarkoitetaan generointiprosessia, jossa eri työkaluilla tuotetusta aineistosta koostetaan semanttinen malli hyödyntämällä useita eri aineistokohtaisesti räätälöityjä scriptejä. Scriptit ajetaan Eclipse-ohjelmointiympäristössä Apache Ant -käännöstyökalun avulla. Ant mahdollistaa ohjelmistokomponenttien keskinäisten riippuvuuksien kuvailun ja osaa ajaa komponentit oikeassa järjestyksessä. Komponentit ja riippuvuuden voidaan Ant-työkalua käyttäen kuvailla helposti luettavaan XML-muotoiseen tiedostoon. [36]

Semanttisen mallin generointi etenee CANopen-aineiston osalta kuvan 5.5 osoittamalla tavalla. Ensin putkilinja kerää CANopen- ja Vertex ED sähkösuunniteluaineistosta tarvittavat tiedot semanttiseen RDF-muotoiseen malliin sekä linkittää simulointimallikirjaston komponentit mallin komponentteihin. Tämän jälkeen putkilinja tuottaa CANopen-kaavion, CANopen-simulointimallipohjan ja parametrit simulointimallille. Semanttinen malli myös tarjoaa tiedot virtuaalisen konelaboratorion semanttisen datalehden näkymään.



Kuva 5.5: Semanttisen prosessin vaiheet CANopen-aineiston osalta [26].

Putkilinjan Ant-käännöstiedosto koostuu projektista, joka sisältää kohteita (engl. targets). Projektille annetaan nimi, oletuskohde ja sille voidaan myös määritellä vakioita, joita voidaan käyttää kohteiden sisällä. Kohteille voidaan määritellä riippuvuuksia toisista kohteista ja kun projekti "käännetään" ajetaan ensin valitun kohteen vaatimat riippuvuudet ja sen jälkeen itse kohde. Kuvassa 5.6 on esitetty *dcf2rdfdatasheet*-kohde, joka suorittaa Python-scriptin, jonka avulla kerätään semanttiseen datalehteen liittyvää materiaalia CANopen-aineistosta semanttiseen malliin. Kuvasta voidaan nähdä kuinka kohteessa ajetaan Python-scripti *exec*-ominaisuuden avulla. Python-scriptin sijainti ja sille välitettävät parametrit on määritelty *arg*-parametrien avulla. Parametreilla saadaan tässä tapauksessa välitettyä CANopen-projektien PCO-tiedostojen sijainnit Python-scriptille.

Putkilinja on rakennettu niin, että jokainen putkilinjan komponentti tuottaa ulostulonaan oman RDF-tiedoston. Putkilinjan viimeisenä ajettavassa kohteessa nämä erilliset RDF-tiedostot validoidaan ja jos kaikkien validointi onnistuu yhdistetään ne yhdeksi RDF-tiedostoksi, joka toimii yhtenäisenä semanttisena mallina.

```
<target name="dcf2rdfdatasheet">
  <exec executable="${python}" failonerror="true">
    <arg value="tools/can.dcf2rdf/dcf2rdfdatasheet.py" />
    <arg value="activities/ontology-design/resources/generated/rdf/dcf-datasheet.rdf" />
    <arg value="activities/canopen-design/resources/external/smg-bbb/NODELIST.PCO" />
    <arg value="activities/canopen-design/resources/external/smg-boom/NODELIST.PCO" />
    <arg value="activities/canopen-design/resources/external/smg-hmi_updated/NODELIST.PCO" />
  </exec>
</target>
```

Kuva 5.6: Esimerkki Ant-käännöstyökalulle tehdystä putkilinjastan osasta.

5.2.2 DCF-tiedostojen lukeminen osaksi mallia

DCF-tiedostoista luetaan CANopen-laitteen objektikirjasto ja tärkeimmät tiedot kuten Node ID, Net ID, laitteen nimi ja laitetyyppi osaksi semanttista mallia. Tiedot linkitetään yhteen PDM-järjestelmästä saatujen tietojen kanssa. Linkittämiseen käytetään aiemmin (kohdassa 4.1) CANopen-aineistoon lisättyä *RefDesignator*-avainta, jonka arvona toimii PDM-järjestelmässä oleva laitteen tunniste. Näin DCF-tiedoston objektikirjasto voidaan liittää semanttisessa mallissa osaksi CANopen-laitteen muita tietoja.

Objektikirjaston lukeminen toteutettiin erillisenä Python-scriptinä. Scripti lukee kaikki sille annetut CANopen-projektitiedostot läpi ja hakee niistä kaikkien verkon laitteiden DCF-tiedostot. Jokaisesta DCF-tiedostosta puolestaan luetaan koko objektikirjasto ja *RefDesignator*-arvo, jonka jälkeen nämä tallennetaan RDF/XML-muotoon.

Lisäksi tehtiin toinen skripti, joka käsittelee tarkemmin laitteen tietoja, kuten Node ID-, NodeName-, Bus ID- ja Device type -arvoja ja jäsentää nämä paremmin semanttiseen malliin sopivaan muotoon. Näin semanttisesta mallista nähdään helposti mitkä laitteet kuuluvat mihinkin verkkoon ja miten eri tyyppiset laitteet liittyvät eri standardeihin. Myös laitteen objektikirjastosta löytyvät PDO-signaalin tiedot muokattiin sriptan avulla semanttiseen malliin sopivaan muotoon, jolloin signaaleiden kulkua laitteiden välillä voitiin visualisoida suoraan semanttisen mallin avulla.

5.2.3 CANopen-standardien linkittäminen malliin

Jokaisen CANopen-laitteen objektikirjaston indeksin 0x1000 arvona on laitteen tyyppi (device type), joka kertoo suoraan heksana laitteeseen liittyvän laiteprofiilistandardin numeron. CiA tarjoaa sivuillaan CSV-muotoista tiedostoa, jossa on laiteprofiilistandardien numerot ja lyhyt kuvaus jokaisesta laiteprofiilista. Lisäksi myös itse standardit on ladattavissa PDF-muodossa.

CSV-tiedosto luettiin osaksi semanttista mallia tähän tarkoitukseen tehdyllä skriptillä. Näin semanttisessa mallissa laitteista olevat standardinumerot saatiin lin-

kitettyä laiteprofiilien nimiin. Myös PDF-muotoiset standardi-tiedostot linkitettiin standardinumeroon. CSV-tiedostosta tuotettiin kuvan 5.7 mukainen RDF-tiedosto, jossa jokaisesta laiteprofiilista on kuvaus, nimi ja tyyppi. Kaikki laiteprofiilit on tyypitetty URI-tunnisteen avulla *http://www.tut.fi/projects/semogen/specification*-tyyppisiksi. Näin semanttisesta mallista on myöhemmin helppo hakea tyypin perusteella esimerkiksi kaikki laiteprofiilit.

```
<rdf:Description rdf:about="http://www.can-cia.org/semogen/408">
  <ns1:description>CANopen device profile for fluid power technology proportional valves and
  hydraulic transmissions</ns1:description>
  <rdf:type rdf:resource="http://www.tut.fi/projects/semogen/specification"/>
  <rdfs:Label>CiA-408</rdfs:Label>
</rdf:Description>
<rdf:Description rdf:about="&semogen;canopenValve">
  <semogen:conformsTo rdf:resource="&semogen_cia;408"></semogen:conformsTo>
</rdf:Description>
```

Kuva 5.7: Katkelma generoidusta RDF-muotoisesta laiteprofiilien kuvauksesta.

Kuvasta 5.7 nähdään myös miten laiteprofiili voidaan linkittää OWL-ontologiassa määriteltyyn luokkaan. *semogen:conformsTo*-elementin avulla CiA 408 -laiteprofiili linkitetään kuvassa 5.3 esitettyyn *&semogen;canopenValve*-luokkaan.

5.2.4 Sähkökaavion linkittäminen CANopen-dataan

Sähkökaavion tietojen tuonti Vertex ED -ohjelmistosta toteutettiin ohjelman SVG-export toiminnon avulla. SVG-export tuottaa SVG-muotoisen kuvan sähkökaaviosta, joka sisältää myös laitteille määritettyjä attribuutteja. SVG-standardi sallii ylimääräisten attribuuttien lisäämisen, joten saatu kuva on edelleen XML-pohjaisen SVG-standardin mukainen ja sitä voi katsella vaikka selaimessa. Attribuutit on lisätty kuvassa 5.8 näkyvän *http://www.vertex.fi/svg*-nimiavaruuden avulla. Nimiavaruudesta käytetään lyhennettä *vx*. Lisättyjen attribuuttien avulla yksilöidään SVG-kuvassa olevat laitteet ja tunnistetaan laitteiden liittyminen toisiinsa.

Kuvassa 5.8 nähdään osa Vertex ED -ohjelmiston tuottaman SVG-kuvan lähdekoodista. Formaatissa sähkökomponentit on kuvattu SVG:n ryhmien (g-elementti) avulla. Jos ryhmältä löytyy attribuutti *vx:NetID-NodeID* tiedämme, että kyseessä on CANopen-laite, jolle on aineiston rikastusvaiheessa lisätty tunniste. Tämän attribuutin arvon perusteella voimme tunnistaa saman laitteen CANopen-aineistosta.

Jos komponentilla on attribuutti *vx:vmt_linkends*, kuten kuvassa 5.8 viimeisen ryhmän esittämällä komponentilla, tiedämme että komponentti linkittää kaksi muuta komponenttia yhteen joiden *vx:ELEM*-tunnisteet on kuvattu attribuutin arvossa. Kuvan esimerkissä alin komponentti linkittää kaksi ylempää komponenttia yhteen. Näiden tietojen perusteella on mahdollista rakentaa verkkotopologia CANopen-verkosta ja näin tehdäänkin. Topologia tallennetaan nodelist.graphml-formaattiin.

```
<svg viewBox="0 0 768 1024" version="1.1"
xmlns="http://www.w3.org/2000/svg"
xmlns:vx="http://www.vertex.fi/svg"
stroke-linecap="round" stroke-linejoin="round" fill-rule="evenodd" xml:space="preserve" >
<!-- ... -->
<g vx:VxGroupId="MACRO_27_5ee98979-9f8c-4d41-9b8e-2c4ef68c7a91" vx:ELEM="3,23" vx:ELEMID="3_23"
  vx:NetID-NodeID="6-13 (B_LIFT_V)" vx:vmt_symbolid="1010000066" vx:vmt_symboltype="150"
  vx:vmt_vxtransform="|0.500000,0.000000,186.250000 0.000000,0.500000,41.250000|"
  vx:vmt_fulldevlabel="Y26" vx:vmt_vxlayer="1" >
<polyline points="377.475,662.361 377.475,673.709 " stroke-width="0.35" fill="none"
  stroke="rgb(0,36,170)" vx:linesort="1" vx:linescale="3" />
<!-- ... -->
<ellipse rx="0.472834" ry="0.472834" cx="377.002" cy="658.106" stroke-width="0.35"
  fill="rgb(0,36,170)" stroke="none" vx:linesort="1" vx:linescale="3" />
</g>

<g vx:VxGroupId="MACRO_3_1bc4d760-a0b2-416f-a150-6ff08f06bf74" vx:ELEM="3,6" vx:ELEMID="3_6"
  vx:NetID-NodeID="6-14 (B_ZOOM_V)" vx:vmt_symbolid="1010000067" vx:vmt_symboltype="150"
  vx:vmt_vxtransform="|0.500000,0.000000,205.500000|0.000000,0.500000,41.064129|"
  vx:vmt_fulldevlabel="Y101" vx:vmt_vxlayer="1" >
<polyline points="413.41,664.604 417.193,673.588 " stroke-width="0.35" fill="none"
  stroke="rgb(0,36,170)" vx:linesort="1" vx:linescale="3" />
<!-- ... -->
<path d="M420.503,656.093  A2.83701,2.83701 0 0,0 426.177,656.093 " stroke-width="0.35"
  fill="none" stroke="rgb(0,36,170)" vx:linesort="1" vx:linescale="3" />
</g>

<g vx:ELEM="1,415" vx:ELEMID="1_415" vx:link_refio="|3,95|3,69|" vx:vmt_subtype="LINK"
  vx:vmt_linkends="|3,23|3,6|" vx:vmt_netno="4" vx:vmt_vxlayer="3" >
<polyline points="386.932,655.742 386.932,641.557 412.465,641.557 412.465,656.093 "
  stroke-width="0.35" fill="none" stroke="rgb(0,36,170)" vx:linesort="6" vx:linescale="1" />
</g>
<!-- ... -->
</svg>
```

Kuva 5.8: Katkelma Vertex ED -ohjelmiston tuottamasta SVG-tiedostosta.

Nodelist.graphml-formaatti on XML-pohjainen GraphML-formaatin päälle GraphML-laajennusten avulla määritelty tiedostoformaatti, jolla voidaan tallentaa CANopen-verkon topologia ja tietoja verkon solmuista ja kaapeleista. Formaatti on yhteensopiva GraphML-määrityksen ja työkalujen kanssa. Tarkempi kuvaus formaatista löytyy julkaisusta [37]. Formaatissa solmujen tiedot tallennetaan *node*-elementtien sisään ja kytkentätiedot *edge*-elementtien sisään.

Yksi *node*-elementti sisältää yhden fyysisen CANopen-laitteen tiedon, kuten kuvassa 5.9 on esitetty. Koska yhdessä CANopen-laitteessa voi olla useita loogisia laitteita, sisältää *node*-elementti yhden laitekohtaisen tieto-osion ja yhden tai useamman loogisen laitteen verkkokohtaisen osion. Sähkökaavion SVG-tiedoston perusteella voidaan päätellä esiintyykö solmu yhdessä vai useammassa CANopen-verkossa. Päättely ja tiedon koostaminen tehdään tarkoitukseen räätälöidyllä Python-scriptillä. Solmun piirroskoordinaatit X ja Y luetaan SVG-tiedostosta. Loput tiedot saadaan generoitua CANopen-aineistoa lukemalla.

Varsinainen verkkotopologia eli tiedot solmujen liittymisestä toisiinsa tallennetaan nodelist.graphml-tiedostoon *edge*-elementtien sisään. Kuvassa 5.10 on esitetty esimerkki *edge*-elementistä, joka yhdistää kaksi solmua toisiinsa. Yllä mainittu

```
<node id="N003">
  <!-- Solmun laitekohtaiset tiedot -->
  <data key="NodeName">DemoNode_C</data>
  <data key="NodeType">device</data>
  <data key="NodeFig">testnode.bmp</data>
  <data key="X">220</data>
  <data key="Y">20</data>
  <data key="NumOfNets">1</data>

  <!-- Solmun verkkokohtaiset tiedot -->
  <data key="NetNumberN1">1</data>
  <data key="NetworkNameN1">DefaultNet</data>
  <data key="NodeIDN1">4</data>
  <data key="NodeDCFNameN1">demodevb.dcf</data>
  <data key="SupplyDomainN1">Primary</data>
  <data key="SupplyPointN1">0</data>
</node>
```

Kuva 5.9: Solmun tiedot nodelist.graphml-formaatissa [37].

Python-scripti käy myös läpi SVG-tiedostosta löytyvät *vx:vmt_ linkends*-attribuuttit ja muodostaa *edge*-elementtejä niiden perusteella. Myös kaapelin kulkuväylä on mahdollista päätellä SVG-tiedostossa olevan datan perusteella. Tämä tieto voidaan edelleen tallentaa *LineParams*-avaimeen. Kaapelin verkon tunniste (engl. NetNumber) päätellään siihen liittyvien solmujen verkkotunnisteista.

```
<edge id="E002" source="N001" target="N003">
  <data key="NetNumber">7</data>
  <data key="CableName">W5002</data>
  <data key="Length">500</data>
  <data key="LineType">line</data>
  <data key="LineParams"></data>
</edge>
```

Kuva 5.10: Kaapelin tai kytkennän tiedot nodelist.graphml-formaatissa [37].

6. LINKITETYN CANOPEN-AINEISTON HYÖDYNTÄMINEN

Muuhun suunnitteluaineistoon linkitettyä CANopen-aineistoa voidaan hyödyntää monessa eri käyttötarkoituksessa koulutuksesta ja vianhausta aina ohjekirjojen ja muun materiaalin generointiin. Tässä luvussa esitellään semanttisen mallin avulla linkitettyn aineiston hyödyntämistä virtuaalisessa konelaboratoriossa, sekä virtuaalisen konelaboratorion tarjoamia näkymiä semanttisessa mallissa olevaan CANopen-aineistoon. Myös CANopen- ja sähkösuunnitteluaineiston linkittämisestä syntyneen topologian hyödyntäminen vianhaussa, sekä simulointimallin generointi CANopen-aineistosta esitellään.

6.1 Virtuaalisen konelaboratorion osana

Virtuaalisella konelaboratoriolla tarkoitetaan simuloitua suunnittelu- ja oppimisympäristöä, joka kuvaa laitteen tai koneen rakennetta ja toimintaa [23]. Perinteisesti tällaisten järjestelmien kehittäminen vaatii paljon manuaalista työtä erityisesti aineiston tuottamisen osalta. Tässä diplomityössä pyrittiin hyödyntämään valmista suunnittelutietoa eri lähteistä ja generoimaan siitä suoraan käyttökelpoista materiaalia osaksi virtuaalista konelaboratoriota. Lähdemateriaaliksi valittiin eri tekniikanalojen suunnitteluaineistoa. Näitä olivat hydrauliikka-, mekaniikka-, sähkö- ja väylätekniikan aineistot. Lähdeaineistot ja niihin liittyvät suunnitteluohjelmat on esitetty tarkemmin luvussa 4.

Aineistoista koostettiin semanttinen tietomalli, joka sisältää aineistoista poimitun datan, sekä linkittää yhteen eri tekniikan alueiden aineistot. Semanttinen malli on kuvattu tarkemmin kohdassa 5.1. Semanttista mallia hyödynnettiin edelleen luomalla siihen erilaisia näkymiä virtuaalisen konelaboratorion avulla.

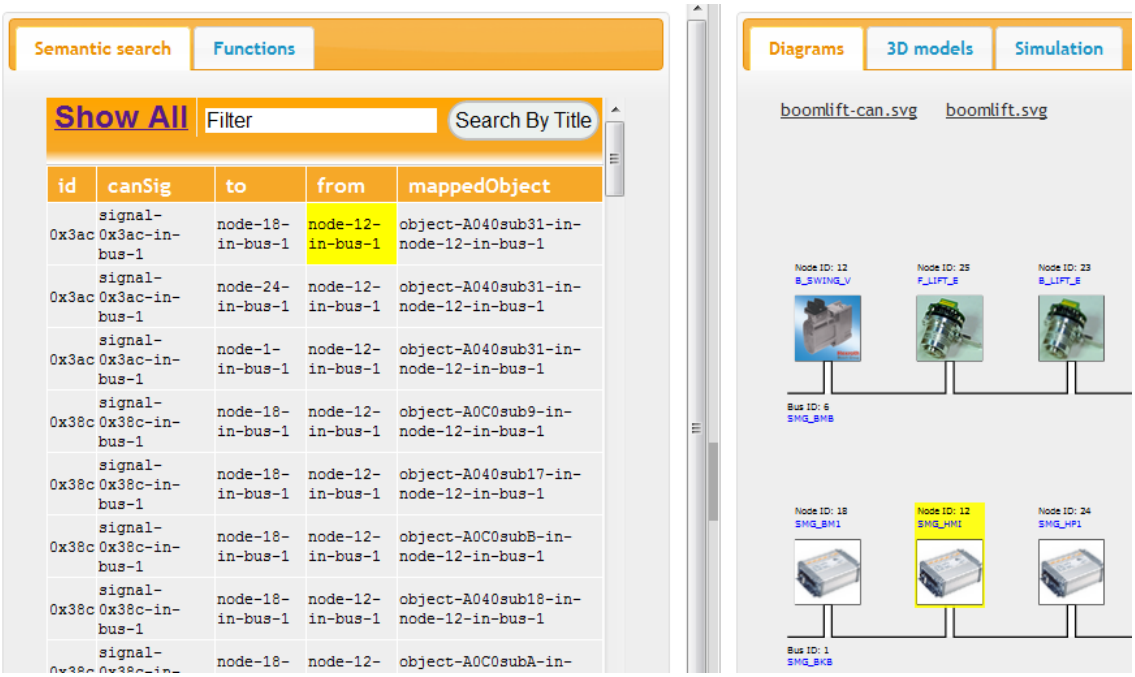
Työn aikana havaittiin, että kaikki suunnitteluaineisto ei sovellu suoraan koneluettavaksi, vaan aineistoa pitää rikastaa, jotta se linkittyy paremmin. Rikastaminen voidaan tehdä joko suoraan suunnitteluohjelmassa tai jälkeenpäin suoraan aineistoon. Suunnitteluvaiheessa rikastaminen onnistuu ohjeistamalla suunnittelua tekemään tarvittavat lisäykset aineistoon. Jälkeenpäin rikastaminen voi olla hankalaa, koska suunnittelijalla oleva hiljainen tieto ei välttämättä ole saatavilla.

Datan yhdistämisestä syntyneeseen semanttiseen malliin voidaan luoda useita näkymiä tai generoida siitä edelleen uutta dataa. Virtuaalisessa konelabo-

ratoriossa on myös useita CANopeniin liittyviä näkymiä, esimerkiksi semanttinen datalehti, CAN-kaavio sekä simulointimallin generointi, kuten kuvassa 5.5 on esitetty. Yhdistetyt aineistot luovat lisäarvoa toisilleen. Semanttinen datalehti esittää komponenttiin liittyvää tietoa ja sallii tiedon muokkaamisen. CANopen-komponentin tapauksessa CANopen-konfigurointitiedon (ProCANopen) lisäksi datalehdessä voi näkyä esimerkiksi komponentin mekaanisia, sähköisiä tai hydraulisia tietoja. Generoitu CANopen-kaavio voi sisältää CANopen-konfigurointitiedon lisäksi topologiatietoa sähkökaaviosta tai komponenttien hierarkkiatietoa PDM-järjestelmästä. Simulointimalli taas hyödyntää osittain sähkökaaviosta ja osittain CANopen-konfiguraatiotiedostoista peräisin olevaa tietoa.

6.1.1 CANopen-väylänäkymä

Virtuaalinen konelaboratorio tarjoaa kuvan 6.1 mukaisen näkymän semanttisessa mallissa olevaan CANopen-aineistoon. Kuvassa oikealla näkyy CANopen-väyläkaavio, jossa on visualisoitu aineistossa olevat CANopen-väylät ja niihin kuuluvat solmut. Vasemmalla on hakunäkymä solmujen objektikirjastoiden tietoihin.



Kuva 6.1: Virtuaalisen konelaboratorion tarjoama näkymä CANopen-verkon tietoihin [38].

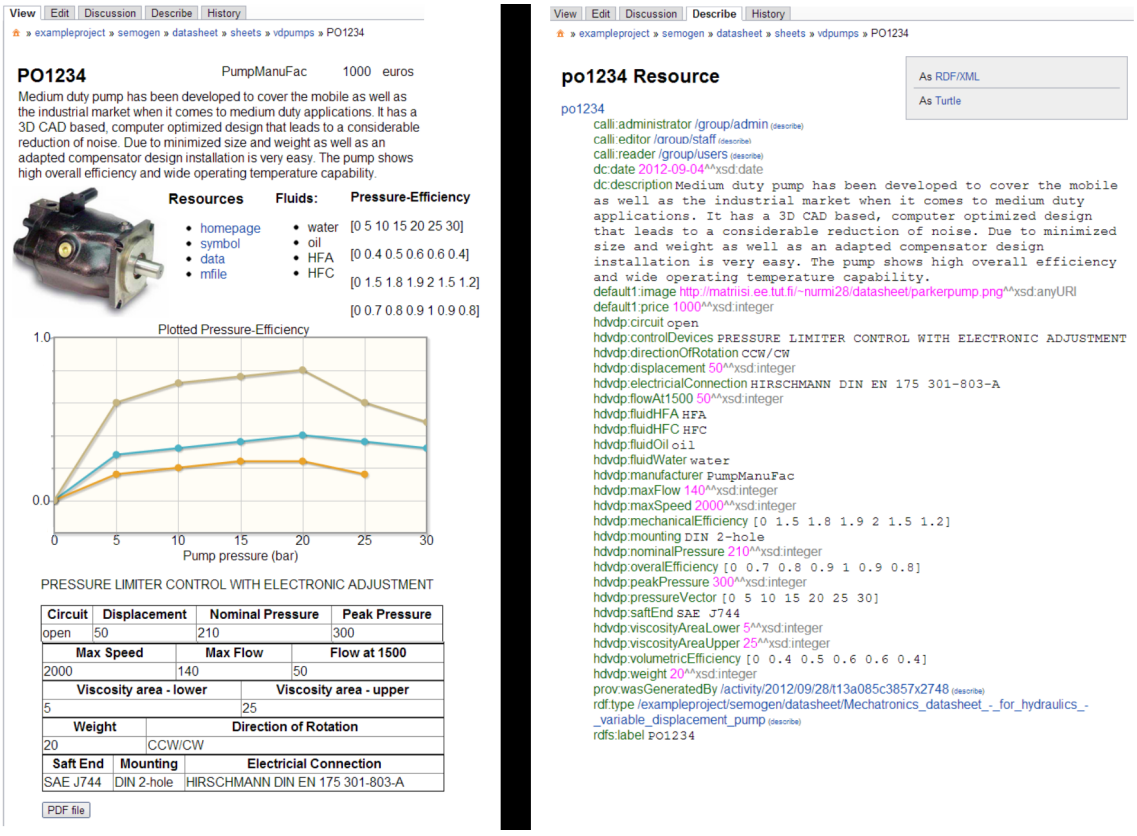
Hakunäkymän avulla on myös mahdollista nähdä PDO-signaalien kulkureitti solmusta toiseen. PDO-viesteillä välitetään suurin osa reaaliaikaisesta tiedonsiirrosta CANopen-laitteen ollessa päällä. Esimerkiksi venttiilin ohjaus tai anturin asento välittyvät PDO-viestien avulla. Molemmissa näkymissä olevat tiedot tulevat semanttisesta mallista ja ovat siten linkittyneitä keskenään. Tämä näkyy käyttäjälle muunmuassa siten, että valittaessa hakunäkymästä jokin objektikirjaston objekti tai sig-

naali, käyttöliittymä valitsee automaattisesti myös väyläkaavionäkymästä solmun, johon tämä signaali tai objekti kuuluu. Näin voimme virtuaalisessa konelaboratoriossa helposti tarkastella mitkä viestit välittyvät millekkin laitteelle.

Väyläkaavionäkymän pohjana toimii SVG-kuva, joka generoidaan automaattisesti semanttisesta mallista. Generointi tehdään Python-scriptillä, joka tuottaa standardin SVG-kuvan, jossa CANopen-laitteisiin on *id*-attribuutin avulla liitetty Node ID ja väylää kuvaavaan viivaan Net ID. Näin aineiston linkitys virtuaalisessa olevaan hakunäkymään onnistuu.

6.1.2 CANopen-laitteen semanttinen datalehti

Semanttinen datalehti on toinen virtuaalisen konelaboratorion CANopen-aineistoon tarjoamista näkymistä. Datalehti-näkymän pohjana toimii Callimachus-ympäristö. Callimachus on avoimen lähdekoodin projekti, joka tarjoaa valmiin käyttöliittymän semanttisen mallin visualisointiin ja muokkaamiseen selaimessa [39]. Käyttöliittymiä voi muokata omiin tarpeisiin sopivaksi ja virtuaalisessa konelaboratoriossa näkymästä tehtiin kuvassa 6.2 vasemmalla esitetyn näköinen datalehti-näkymä. Datalehti pystyy visualisoimaan minkä tahansa semanttisesta mallista löytyvän komponentin tiedot, kunhan ne on kuvattu sovitussa formaatissa.



Kuva 6.2: Semanttisen datalehden näkymä virtuaalisessa konelaboratoriossa [26].

Kuvassa 6.2 oikealla näkyy samaisen datalehden semanttisessa mallissa olevat tiedot RDF-muodossa esitettynä. Siitä nähdään myös datalehden rakenne semanttisessa mallissa.

6.2 CANopen-verkon topologiatiedon hyödyntäminen vianhaussa

CANopen-verkon topologiainformaatiota hyödynnetään myös vianhakumonitorissa. Vianhakumonitori on CANopen-verkkoon kytketty laite, joka tarkkailee verkon ja verkossa olevien solmujen tilaa. Ilman topologia tai projektitietoa voidaan verkosta havaita vialliset tai väärin toimivat laitteet, mutta topologiatiedon avulla voidaan virhetilanteiden aiheuttaja sijainti ja syy päätellä tarkemmin.

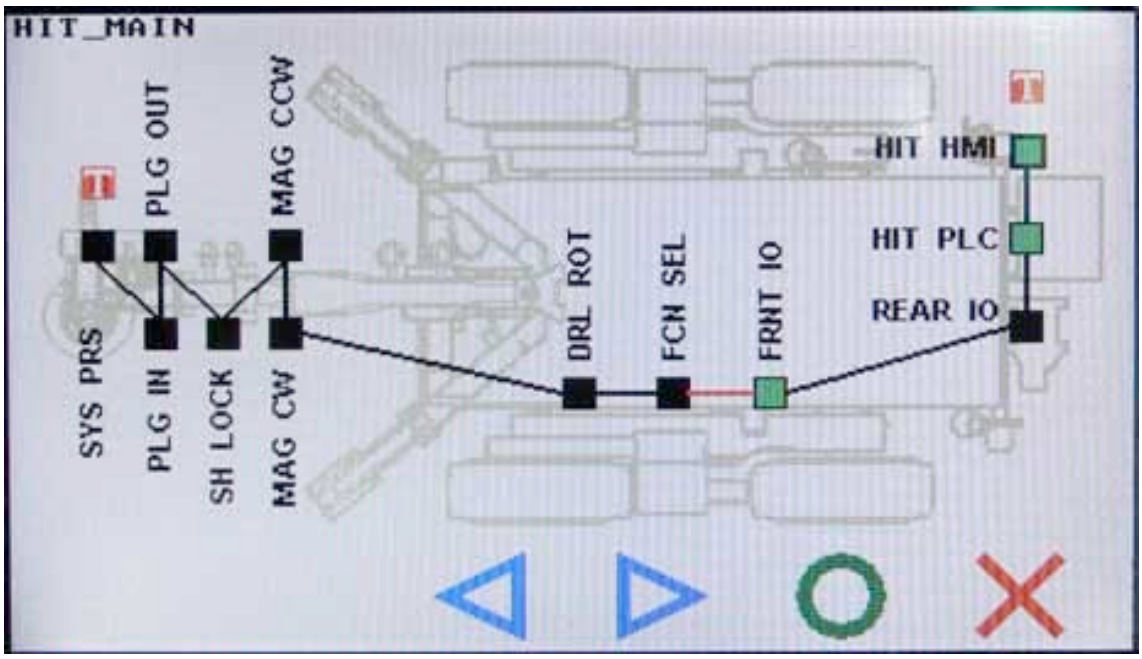
Vastoin yleistä käsitystä CANopen-verkossa laitteiden määrää lisäämällä havaitsemattomien virheiden todennäköisyys laskee. Tämä johtuu siitä, että jokainen laite tarkistaa kaikki verkossa menevät viestit ja raportoi aina virheellisestä viestistä. Näin myös virheen paikallistaminen helpottuu laitemäärän kasvaessa. [40]

Alakohdassa 5.2.4 esitetyn sähkökaavion ja CANopen-aineiston yhteenlinkityksen tuloksena syntynyneen, Nodelist.graphml-formaatissa olevan, verkkotopologiatiedon avulla voidaan muodostaa kuvan 6.3 mukainen vianhakunäkymä. Nodelist.graphml mahdollistaa solmujen sijaintitiedon, väyläkaapelin reititystiedon, sekä taustakuvan tallentamisen. Näin ollen kaikki vianhakumonitorin tarvitsema tieto voidaan poimia suoraan nodelist.graphml-formaatista, eikä erillistä omaa tallennusformaattia tarvita. Vianhakumonitorista voidaan siis tehdä geneerinen, missä tahansa CANopen-verkossa toimiva monitori, joka saa tietonsa suoraan verkon CANopen-projektin topologiatiedoista.

Topologiatiedon perusteella vikaantunut solmu voidaan paikallistaa laitteessa. Kuvassa 6.3 solmu *REAR IO* on vikaantunut ja näkyy mustana. Toimivat solmut ovat vihreällä. Esimerkiksi huoltomies näkee monitorin kuvan perusteella suoraan solmun fyysisen sijainnin laitteessa ja tämä tehostaa laitteen huoltotoimia.

Topologiatieto mahdollistaa myös kaapeli- tai liitinvian päättelyn. Kuvassa 6.3 *FCN SEL*-solmu ja kaikki siitä vasemmalle olevat solmut näkyvät viallisina. Tästä monitori päättelee, että kyseessä on todennäköisesti kaapeli- tai liitinvika solmujen *FCN SEL* ja *FRNT IO* välisessä kaapelissa ja värittää tämän kaapelin punaisella. [40] Näin huoltomies tietää etsiä vikaa ensin kyseisestä kaapelista.

Terminaattoreiden virheellisestä asentamisesta tai puuttumisesta johtuvat virheet ovat hankalampia päätellä, koska terminaattorit ovat passiivisia laitteita, eivätkä raportoi mitään tietoja itsestään. Myös verkon reagointi terminointivikoihin riippuu topologiasta ja kaapelin impedanssista. Näin terminointivirheet on pääteltävä epäsuorasti solmujen toiminnan perusteella. Kuvassa 6.3 terminaattorit on mer-



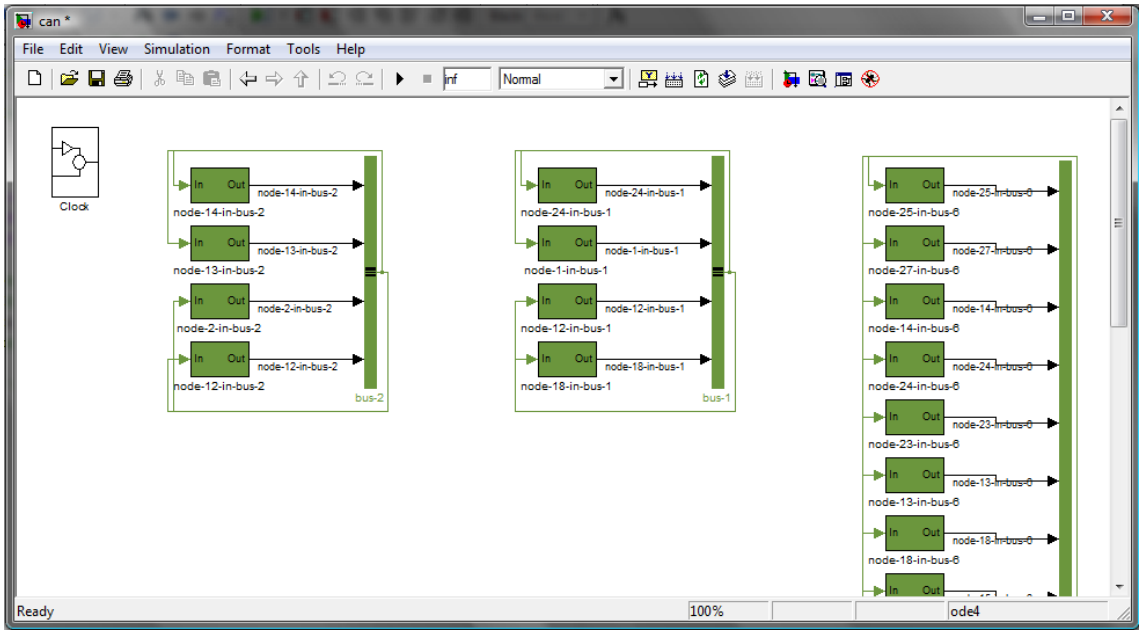
Kuva 6.3: Topologiatietoa hyödyntävä näkymä vianhakumonitorissa [40].

kitty punaisella T-kirjaimella väylän kumpaankin päähän. Punainen väri tarkoittaa terminointivirhettä, joka tässä tapauksessa on päätelty olevan mahdollinen, koska *REAR IO*-solmu toimii virheellisesti vaikka sen jälkeen olevat *HIT PLC*- ja *HIT HMI*-somut toimivat normaalisti. [40]

6.3 CANopen-simulointimallipohjan generointi semanttisesta mallista

Semanttisessa mallissa olevan tiedon perusteella voidaan myös generoida simulointimallipohjia. Työssä tehtiin semanttista mallia lukeva Python-skripti, jonka avulla generoidaan lopulta kuvassa 6.4 esitetty CANopen-simulointimallipohja. Semanttinen malli ei sisällä CANopen-laitteiden toiminnallista kuvausta, joten sen perusteella ei voida generoida täydellistä simulointimallia vaan mallipohjaan on vielä käsin rakennettava laitteiden toiminnallinen logiikka, jotta simulointi onnistuisi. Laitteiden tyypit, kytkennät, konfiguraatioparametrit ja väylän topologiatieto löytyvät kuitenkin semanttisesta mallista. Näiden perusteella voidaan muodostaa simulointimallipohja, joka koostuu käsin luoduista simulointikirjastokomponenteista. Erilliseen simulointikirjastoon on siis koottu peruskomponentit kuten CANopen-laite ja CANopen-väylä ja semanttisen mallin perusteella näistä kirjastokomponenteista voidaan generoida mallipohja, jossa CANopen-laitteet on kytketty oikean aineiston mukaisesti väyliin, sekä niille on määritelty oikeat Node ID -tunnisteet.

Simulointimallipohjan generointi etenee vaiheittain. Ensin generoidaan semanttisesta mallista Python-scriptin avulla MATLAB m-tiedosto. M-tiedosto on MATLAB-



Kuva 6.4: CANopen-aineistosta generoitu simulointimallipohja MATLAB/Simulink-ohjelmassa.

ympäristössä ajettavaa koodia, joka avulla tässä tapauksessa koostetaan simulointimallipohja tyhjään simulointimalliin. M-tiedostoon generoituu tiedot lisättävistä laitteista, niiden paikoista, kytkennöistä ja parametreista. Kuvassa 6.5 on esitetty osa generoidusta m-tiedostosta. Generoidussa m-tiedostossa *add_bloc*-funktion avulla lisätään kirjastokomponenteina CANopen-väylä ja -solmut mallipohjaan, sekä *set_param*-funktiolla asetetaan näille sopivat parametrit. *Add_line*-funktiolla väylä ja solmut liitetään simulointimallissa yhteen. Ajamalla m-tiedosto saadaan kuvan 6.4 mukainen simulointimallipohja, johon voidaan alkaa käsin lisäämään solmujen toiminnallista logiikkaa.

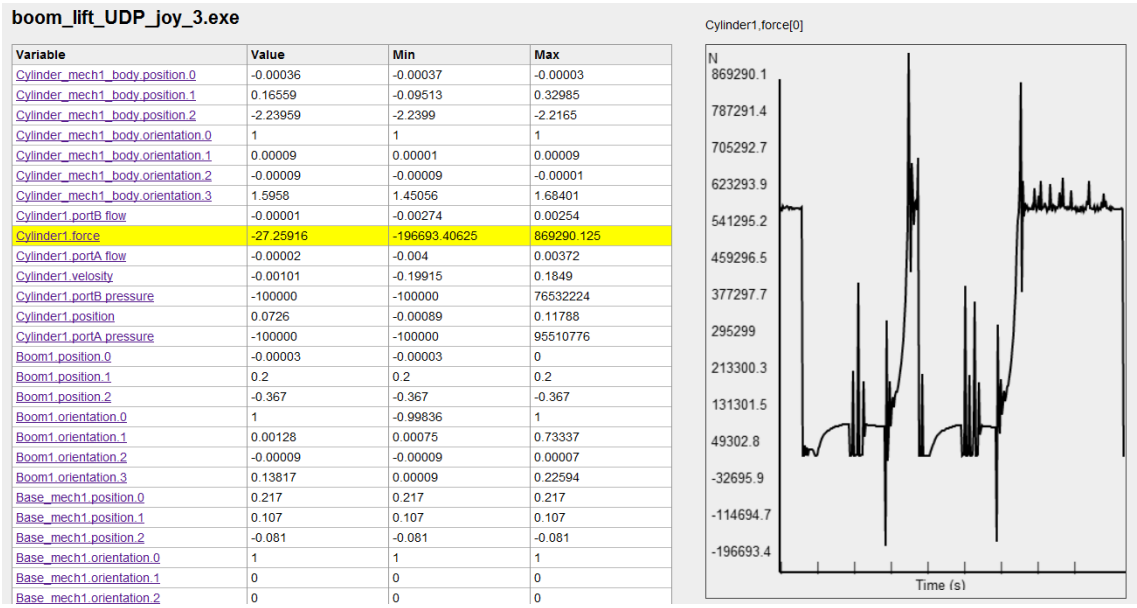
```
%Create bus
%bus 2
add_block('Semogen/CAN/CAN', 'can/bus-2');
set_param('can/bus-2','Position',[300 50 310 250]);
set_param('can/bus-2','Inputs','4');

%Create nodes
%node HMI_PWP, eds:canopen_IO_node_v00070003.eds
add_block('Semogen/CAN/CAN Controller', 'can/node-14-in-bus-2');
set_param('can/node-14-in-bus-2','Position',[150 60 200 90]);
%connect node to bus
line=add_line('can','node-14-in-bus-2/1','bus-2/1','autorouting','on');
set_param(line,'Name','node-14-in-bus-2');
add_line('can','bus-2/1','node-14-in-bus-2/1','autorouting','on');
```

Kuva 6.5: Katkelma generoidusta m-tiedostosta kirjastokomponenttien lisäämiseksi simulointimallipohjaan.

Simulointimallipohjan generointia voisi edelleen kehittää laajentamalla simulointikirjastoa erityyppisillä komponenttien alaluokilla kuten eri tyyppisillä CANopen-

venttiileillä ja antureilla. Semanttisen mallin tietojen perusteella mallipohjaa generoitaessa on mahdollista valita juuri oikean tyyppinen komponentti oikeaan kohtaan. Jopa komponentin konfiguraatio olisi mahdollista lukea simulointimalliin. Tämä vaatii kuitenkin, että simulointikirjastoon on tehty tarpeeksi kattavat mallit jokaisesta komponenttityypistä.

Samaa ideaa simulointimallipohjan generoinnista voitaisiin soveltaa myös hydrauliikan tai mekaniikan simulointimallien generointiin semanttisesta mallista. Hydrauliikan generoinnista toteutettiinkin vastaavasti toimiva generointi-scripti. Myös virtuaalisessa konelaboratoriossa käytettiin laitteen simulointimallia apuna erilaisen mittausdatan tuottamiseen, sekä hydraulikaavion ja koneen 3d-mallin animoinnin apuna. Simulointimallin generointi aineistosta nopeutti siten myös virtuaalisen konelaboratorion kehitystä. Kuvassa 6.6 näkyy simulointimallista saatavaa mittausdataa virtuaalisessa konelaboratoriossa.



boom_lift_UDP_joy_3.exe

Variable	Value	Min	Max
Cylinder_mech1_body.position.0	-0.00036	-0.00037	-0.00003
Cylinder_mech1_body.position.1	0.16559	-0.09513	0.32985
Cylinder_mech1_body.position.2	-2.23959	-2.2399	-2.2165
Cylinder_mech1_body.orientation.0	1	1	1
Cylinder_mech1_body.orientation.1	0.00009	0.00001	0.00009
Cylinder_mech1_body.orientation.2	-0.00009	-0.00009	-0.00001
Cylinder_mech1_body.orientation.3	1.5958	1.45056	1.68401
Cylinder1.portB flow	-0.00001	-0.00274	0.00254
Cylinder1.force	-27.25916	-196693.40625	869290.125
Cylinder1.portA flow	-0.00002	-0.004	0.00372
Cylinder1.velosity	-0.00101	-0.19915	0.1849
Cylinder1.portB pressure	-100000	-100000	76532224
Cylinder1.position	0.0726	-0.00089	0.11788
Cylinder1.portA pressure	-100000	-100000	95510776
Boom1.position.0	-0.00003	-0.00003	0
Boom1.position.1	0.2	0.2	0.2
Boom1.position.2	-0.367	-0.367	-0.367
Boom1.orientation.0	1	-0.99836	1
Boom1.orientation.1	0.00128	0.00075	0.73337
Boom1.orientation.2	-0.00009	-0.00009	0.00007
Boom1.orientation.3	0.13817	0.00009	0.22594
Base_mech1.position.0	0.217	0.217	0.217
Base_mech1.position.1	0.107	0.107	0.107
Base_mech1.position.2	-0.081	-0.081	-0.081
Base_mech1.orientation.0	1	1	1
Base_mech1.orientation.1	0	0	0
Base_mech1.orientation.2	0	0	0

Kuva 6.6: Simulointimallin tuottamaa dataa virtuaalisessa konelaboratoriossa [38].

Simulointimallin generointi suunnitteluaineistosta ei kuitenkaan suunnitteluprosessin kannalta ole välttämättä järkevää. Joissain tapauksissa laitteita simuloidaan jo suunnitteluvaiheen alussa ja silloin prosessia kannattaisi viedä toiseen suuntaaan eli generoida laitteille konfigurointitiedostoja simulointimallin perusteella. CANopen-laitteiden osalta tätä on myös käytännössä kokeiltu ja todettu toimivaksi, kunhan simulointimalli luodaan tietyllä tavalla ja sitä rikastetaan sopivasti solmujen ja solmujen objektikirjastoiden objektien tiedoilla.

7. YHTEENVETO

CANopen-suunnitteluaineiston ja suunnitteluprosessin huomattiin jo nykyisellään tukevan hyvin koneluettavan aineiston tuottamista. Kuitenkin aineiston linkittäminen ei ilman aineiston rikastamista onnistu. CANopen-aineiston solmut on pystyttävä yksilöimään koko suunniteltavan järjestelmän tasolla, eikä pelkästään yhden CANopen-väylän tasolla. On myös pystyttävä yhdistämään solmun tiedot muuhun suunnitteluaineistoon.

Diplomityössä käsitelty CANopen-aineisto koostui kolmesta ProCANopen-projektista, jotka kuvasivat kallioporalaitteen yhden puomin CANopen-väyliä. Puomin ohjauksen kolme väylää hoitivat seuraavia tehtäviä: ohjausväylällä olivat käyttäjän ohjainlaitteet, puomiväylällä puomia ohjaavat venttiilit ja anturit ja pääväylä yhdistää nämä kaksi väylää. Väyläaineiston lisäksi käsiteltiin sähkökaavioita, joista nähdään myös CANopen-laitteiden verkkotopologia ja liitäntäpaikka väylälle.

Nämä kaksi eri tekniikanalan aineistoa linkitettiin keskenään lisäämällä sähkökaavioon CANopen-solmujen Node ID ja Net ID -tunnisteet, joiden avulla samat solmut voitiin tunnistaa ProCANopen-projekteista. Linkittäminen tapahtui suoraan Vertex ED -sähkösuunnitteluohjelmassa, siihen generoitujen attribuuttivalintojen avulla. CANopen-aineistosta voidaan generoida vain olemassa olevat Node ID ja Net ID -parit, jotka näytetään Vertex ED:n valintaikkunassa. Näin käyttäjä voi linkittää sähköaineiston ainoastaan valideihin CANopen-solmujen tunnisteisiin.

Pelkästään näiden kahden aineiston keskinäinen linkittäminen mahdollistaa jo yksinään uusia käyttökohteita suunnittelutiedolle. Sähkökaavion topologiatiedon perusteella voidaan generoida kattava CANopen-verkon topologiamalli, jossa on myös solmujen tiedot mukana. Tätä topologiamallia on käytetty jo käytännön esimerkissä poralaiteen vikanäytössä, jossa CANopen-väylän liikenteen ja topologiatiedon perusteella voidaan automaattisesti päätellä CANopen-väylän kaapelivikojen sijainti tai rikkoontuneen solmun paikka. Vikatieto ja vikakohde voidaan esittää havainnollisesti näytössä näkyvän koneen kuvan ja väyläkaavion avulla.

Työssä oli käytössä myös Vertex PDM -järjestelmä tiedon varastointia varten. Järjestelmä antoi jokaiselle tietoalkiolle yksilöllisen tunnisteen, joka lisättiin myös CANopen-aineiston jokaiselle solmulle. Näin aineisto saatiin linkitettyä yhteen myös PDM-järjestelmässä olevan tiedon kanssa. Koko tietomassasta koottiin semanttinen malli jota voitiin edelleen hyödyntää virtuaalisessa konelaboratioriossa. Siellä

malli tarjosi taustalla olevan informaation useisiin eri näkymiin, kuten kaavioihin ja datalehtiin. Semanttisesta mallista voitiin myös generoida simulointimallipohjia CANopen-simulaatioon MATLAB/Simulink-ympäristöön.

Semanttisen mallin automaattista generointia varten luotiin putkilinjaprosessi. Putkilinja toteutettiin Eclipse-ohjelmointiympäristöön Ant-skriptikielen avulla. Putkilinjassa voidaan ajaa yksittäisiä muunnosprosesseja, joissa tietoa rikastetaan, linkitetään ja tuodaan semanttiseen malliin. Ant tarjoaa myös mahdollisuuden kuvailla riippuvuuksia ja osaa ajaa prosessit automaattisesti oikeassa järjestyksessä.

Tulevaisuudessa koneluettavuutta voitaisiin parantaa muilla tekniikan alueilla esimerkiksi sopimalla yhteisiä formaatteja sähkökaavioiden tai mekaniikkamallien esitykseen. CANopen on jo nykyään hyvin pitkällä koneluettavuudessa mm. Can in Automation -organisaation standardoimien tiedostoformaattien ansiosta. Jopa standardeista tuotettu koneluettava tieto löytyy CPD-tiedostoista, jolloin suunnitteluprosessissa suunnitteluaineisto (EDS-tiedostot) voidaan validoida standardeja vastaan.

Itse CANopen-suunnittelun kehitys voidaan tulevaisuudessa tehdä simulointimallipohjaisesti niin, että ensin tehdään toimiva simulointimalli, josta voidaan edelleen generoida suunnitteluaineisto, jota sopivasti täydentämällä saadaan suoraan CANopen-laitteissa toimivat konfiguraatiotiedostot.

LÄHTEET

[1] W. Voss, *A Comprehensible Guide to Controller Area Network*, Copperhill Media Corporation, 2. painos, s. 152 (2008).

[2] Wikipedia, *OSI-malli*, http://fi.wikipedia.org/wiki/OSI-malli, [WWW] [viitattu 03.09.2012] (2012).

[3] Wikipedia, *CAN bus*, http://en.wikipedia.org/wiki/CAN_bus, [WWW] [viitattu 27.11.2012] (2012).

[4] ISO, *ISO 11898-2, Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit*, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=33423, [WWW] [viitattu 31.10.2012] (2003).

[5] CiA, *CiA 303 Version 1.8.0, Part 1: Cabling and connector pin assignment*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 28.03.2013] (2012).

[6] softing, *CAN/CANopen/DeviceNet*, http://www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus/bit-timing/practical-bus-length.php?navanchor=3010538, [WWW] [viitattu 27.11.2012] (2012).

[7] ISO, *ISO 11898-1, Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=33422, [WWW] [viitattu 31.10.2012] (2003).

[8] Wikipedia, *Siirtokerros*, http://en.wikipedia.org/wiki/Data_link_layer, [WWW] [viitattu 22.11.2012] (2012).

[9] CiA, *CiA 802 Version 1.1.0, CANopen application note - CAN remote frames: Avoiding of usage*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 8.11.2012] (2010).

[10] *The CAN wiki page*, http://www.can-wiki.info, [WWW] [viitattu 31.10.2012] (2012).

[11] CiA, *CAN in Automation (CiA)*, http://www.can-cia.org/, [WWW] [viitattu 31.08.2012] (2012).

[12] T. Shelley, *CANopen expands over Ethernet*, http://www.eurekamagazine.co.uk/design-engineering-features/canopen-expands-over-ethernet/18716/, [WWW] [viitattu 31.08.2012] (2009).

[13] CiA, *CiA 301 Version 4.02, Application Layer and Communication Profile*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 30.11.2012] (2002).

[14] Wikipedia, *CANopen*, http://en.wikipedia.org/wiki/CANopen, [WWW] [viitattu 13.02.2012] (2013).

[15] H. Saha, *CANopen perusteet*, http://www.canopen.fi/artikkelit/CANopen.pdf, [WWW] [viitattu 10.12.2012] (2006).

[16] CiA, *CiA 301 Version 4.2.0, Application Layer and Communication Profile*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 30.11.2012] (2011).

[17] CiA, *CANdictionary in Finnish*, http://www.can-cia.org/index.php?id=candictionary, [WWW] [viitattu 21.03.2013] (2008).

[18] CiA, *CiA Draft Standard Proposal 305 Version 2.2, Layer setting services (LSS) and protocols*, http://www.can-cia.org/index.php?id=915&no_cache=1, [WWW] [viitattu 21.02.2013] (2008).

[19] CiA, *CiA 306 Version 1.3.0, Electronic data sheet specification*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 30.11.2012] (2005).

[20] CiA, *CiA 306 Work Draft Version 1.3.5, Electronic device description Part 1: Electronic Data Sheet and Device Configuration File*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 02.04.2013] (2012).

[21] CiA, *CiA 306 Work Draft Version 0.0.4, Electronic device description Part 2: Profile database specification*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 02.04.2013] (2011).

[22] CiA, *CiA 306 Work Draft Version 0.0.6, Electronic device description Part 3: Network variable handling and tool integration*, http://www.can-cia.org/index.php?id=specifications&no_cache=1, [WWW] [viitattu 02.04.2013] (2012).

[23] J. Salonen, O. Nykänen, P. Ranta, J. Nurmi, M. Helminen, M. Rokala, T. Palonen, V. Alarotu, K. Koskinen ja S. Pohjolainen, *An Implementation of a Semantic, Web-Based Virtual Machine Laboratory Prototyping Environment*, The Semantic Web - ISWC **7032**, 221–236 (2011).

[24] Vector, *Solutions for your CANopen Networking*, `http://canopen-solutions.com/canopen_procanopen_en.html/`, [WWW] [viitattu 27.04.2013] (2013).

[25] Vector, *ProCANopen manual, Version 6.3 English*, `http://wwwtbl.bagn.obs-mip.fr/TblDoc/Documentation/Materiel/can/Softs/Vector%20ProCANopen/ProCANopen_Manual.pdf/`, [WWW] [viitattu 27.04.2013] (2013).

[26] O. Nykänen, K. Koskinen, P. Ranta, J. Salonen, J. Aaltonen, J. Nurmi, M. Helminen, V. Alarotu, T. Salomaa ja S. Pohjolainen, *Semogen II - hanke. Loppuraportti. Tampereen teknillinen yliopisto.*, `http://wiki.tut.fi/SmartSimulators/Semogen2`, [WWW] [viitattu 20.04.2013] (2013).

[27] Wikipedia, *Resource Description Framework*, `https://en.wikipedia.org/wiki/Resource_Description_Framework`, [WWW] [viitattu 12.03.2013] (2013).

[28] D. Brickley ja R. V. Guha, *RDF Primer. W3C, Recommendation 10 February 2004*, `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`, [WWW] [viitattu 30.06.2012] (2004).

[29] F. Manola ja E. Miller, *RDF/XML Syntax Specification, WRecommendation 10 February 2004*, `http://www.w3.org/TR/REC-rdf-syntax/`, [WWW] [viitattu 15.05.2013] (2004).

[30] T. Berners-Lee ja D. Connolly, *Notation3 (N3): A readable RDF syntax, Team Submission 28 March 2011*, `http://www.w3.org/TeamSubmission/n3/`, [WWW] [viitattu 15.05.2013] (2011).

[31] D. Beckett, T. Berners-Lee, E. Prud'hommeaux ja G. Carothers, *Turtle, Terse RDF Triple Language, Candidate Recommendation 19 February 2013*, `http://www.w3.org/TR/turtle/`, [WWW] [viitattu 15.05.2013] (2013).

[32] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler ja N. Lindström, *JSON-LD 1.0, A JSON-based Serialization for Linked Data, Editor's Draft 14 May 2013*, `http://json-ld.org/spec/latest/json-ld/`, [WWW] [viitattu 15.05.2013] (2013).

[33] Wikipedia, *Web Ontology Language*, http://en.wikipedia.org/wiki/Web_Ontology_Language, [WWW] [viitattu 06.05.2013] (2013).

[34] Wikipedia, *RDF Schema*, http://en.wikipedia.org/wiki/RDFS, [WWW] [viitattu 06.05.2013] (2013).

[35] *Pellet: OWL 2 Reasoner for Java*, http://clarkparsia.com/pellet/, [WWW] [viitattu 15.05.2013] (2013).

[36] Wikipedia, *Apache Ant*, http://en.wikipedia.org/wiki/Apache_Ant, [WWW] [viitattu 08.05.2013] (2013).

[37] M. Helminen, J. Salonen, H. Saha, O. Nykänen, K. T. Koskinen, P. Ranta ja S. Pohjolainen, *A New Method and Format for Describing CANopen System Topology*, 13th International CAN Conference, Hambach Castle, Germany 06–11 – 06–18 (2012).

[38] P. Ranta, O. Nykänen, J. Salonen, S. Pohjolainen ja K. Koskinen, *Teollisuuden älykkäiden ja virtuaalisten konelaboratorioiden tuotantomenetelmien kehitys semanttisen kuvauksen avulla Semogen-hanke. Loppuraportti. Tampereen teknillinen yliopisto.*, http://wiki.tut.fi/pub/SmartSimulators/Semogen/Semogen_loppuraportti_TTY_SmartSimulators_Final_211111.pdf, [WWW] [viitattu 28.06.2012] s. 205 (2011).

[39] 3RoundStonesInc., *Callimachus Project. Project's homepage*, http://callimachusproject.org/, [WWW] [viitattu 13.05.2013] (2013).

[40] H. Saha, *Exception management in CANopen systems*, CAN Newsletter 2/2013 12–17 (2013).