TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANNE-JOONAS MANTSINEN

PROSESSI- JA AUTOMAATIOSUUNNITTELUN TIEDONSIIRTO STANDARDIEN AVULLA

Diplomityö

Tarkastajat: Prof. Matti Vilkko ja
TkT David Hästbacka

Tarkastajat ja aihe hyväksytty
1. maaliskuuta 2017

TIIVISTELMÄ

JANNE-JOONAS MANTSINEN: Prosessi- ja automaatiosuunnittelun tiedonsiirto standardien avulla
Tampereen teknillinen yliopisto
Diplomityö, 84 sivua, 10 liitesivua
Lokakuu 2017
Automaatiotekniikan koulutusohjelma
Pääaine: Prosessien hallinta
Tarkastajat: Prof. Matti Vilkko ja TkT David Hästbacka
Avainsanat: metamallintaminen, mallimuunnos, suunnittelutiedon siirto, XML, standardit, DEXPI, OPC UA

Laitossuunnittelutiedon siirto on murroksessa. Laitossuunniteluprojektien osapuolien on kyettävä vaihtamaan suunnittelutietoa keskenään vaivattomasti ja häviöttä. Jos kaikki suunnitteluosapuolet eri suunnitteluosa-alueilta käyttäisivät samaa laitossuunnitteluohjelmistoa, ongelmaa ei olisi, mutta käytäntö on toinen. Laitossuunnitteluohjelmistojen välinen tiedonsiirto voidaan yhdenmukaistaa siten, että ohjelmistoihin toteutetaan standardin mukainen rajapinta. Standardin mukaisten suunnitteludokumenttien arvoa voidaan lisätä ja suunnittelutyötä automatisoida tarjoamalla suunnittelutieto tunnetun ja avoimen rajapinnan kautta. Ongelman ratkaisemiseksi käytettiin toimintatutkimusmenetelmää, jossa teoria ja käytäntö kulkevat käsikädessä.

Aluksi selvitettiin suunnittelutiedonsiirtoon soveltuvat standardit. Standardeista valittiin ISO 15926 standardiin perustuva DEXPI-spesifikaatio (Data Exchange in the Process Industry). Tähän spesifikaatioon päädyttiin, koska sitä kehitetään aktiivisesti ja sen toteuttaminen on aloitettu useammassa laitossuunnitteluohjelmistossa. Lisäksi tämä spesifikaatio soveltuu prosessi- ja automaatiosuunnittelmien graafisen tiedon ja metatiedon siirtämiseen. Tässä työssä kehitettiin metriikka laitossuunnitteluohjelmistoista tuotujen suunnitteludokumenttien rakenteellisen kypsyyden arviointiin. Tässä työssä kypsyydellä tarkoitetaan miten dokumentin rakenne noudattaa spesifikaatiossa määritettyä rakennetta. Spesifikaation toteutumista laitossuunnitteluohjelmistoissa seurattiin aktiivisuustaulukon avulla. Standardin mukaista suunnitteludokumenttia voidaan hyödyntää myös laitteistorajapinnassa OPC UA -standardin (OPC Unified Architecture) avulla. Muunnos standardien mukaisten dokumenttien välillä voidaan automatisoida metamallinnus- ja mallimuunnostekniikoiden avulla.

ABSTRACT

JANNE-JOONAS MANTSINEN: Standard based Information Exchange in Process and Automation Engineering
Tampere University of Technology
Master of Science thesis, 84 pages, 10 Appendix pages
October 2017
Master's Degree Programme in Automation Engineering
Major: Process Automation
Examiners: Prof. Matti Vilkko and D.Sc. (Tech.) David Hästbacka
Keywords: metamodeling, model transformation, engineering infromation exchange, XML, standards, DEXPI, OPC UA

The exchange of plant engineering information is in transition. The parties involved in plant design projects must be able to exchange design information easily and without losing data. If all design partners from different design areas would use the same plant design software, the problem would not be, but the practice is the other. The data transfer between the plant design softwares can be harmonized in such a way that the standard interface is implemented for the software. The value of the standard design documents can be increased and the design work automated by providing design information through a well-known and open interface. An action research method was used to solve this problem, where theory and practice go hand in hand.

At first, the standards applicable to the design data transfer were investigated. From the standards, the ISO 15926 based DEXPI (Data Exchange in Process Industry) spesification was selected. This specification was chosen because it is being developed actively and its implementation has been started in several plant design programs. In addition, this specification is suitable for process and automation planning for transferring graphic information and metadata. In this work, a metric for the evaluation of the maturity of the engineering document structure exported from plant desing program was developed. In this context maturity means how the structure of a document conforms to a structure defined in spesification. Implementation of specification in plant design software was monitored using the activity matrix. The standard design document can also be used in the hardware interface using OPC UA (OPC Unified Architecture). The transformation of standard documents can be automated with meta-model and model transformation techniques.

ALKUSANAT

Näin ensialkuun haluan kiittää tarkastajia professori Matti Vilkkoa ja tutkijatohtori David Hästbackaa, sekä ohjaajia professori Tommi Karhelaa ja tohtori Nikolaos Papakonstantinouta heidän tarjomastaan tuesta diplomityöni kirjoitusprosessissa.

Vuolaat kiitokset kuuluvat Systeemisuunnittelu ja simulointi -tiimille, niin sen vetäjälle tohtori Juha Kortelaiselle puitteista ja motivoinnista, kuin myös Tuomas Miettiselle, Marko Luukkaiselle ja Reino Ruusulle teknisen osaamisen pyytteettömästä jakamisesta.

Viimeisenä tärkeimmät kiitokset vaimolle ja kotijoukoille äidille, isälle ja siskolle kannustuksesta koko opiskeluputken ajan.

Espoossa, 20.9.2017

Janne-Joonas Mantsinen

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

OHJELMALUETTELO

LYHENTEET

CAEX	Computer Aided Engineering Exchange on IEC 62424 -standardin määrittelemä neutraali tietomalli ja formaatti hierarkisen laitossuunnitelutiedon tallentamiseen ja siirtämiseen.
CDD	Common Data Dictionary on eräänlainen standardin määrittelemä yleinen tesaurus, johon koottu toimialakohtaiset konseptit menetelmien ja tietomallin perusteella.
CHR	Constraint Handling Rules on ylemmän tason ohjelmointikieli, jota voidaan käyttää muun muassa mallimuunnoksiin. Sitä voidaan käyttää myös laajennoksena toisessa ohjelmointikielessä.
CSV	Comma-Separated Values on tekstitiedosto, jossa taulukkomuotoista tietoa tallennetaan pilkuilla erotellen ja rivinvaihdoin.
DEXPI	Data Exchange for the Process Industry on hanke, jonka tavoitteena on kehittää tiedonsiirtostandardia prosessiteollisuuden tarpeisiin laitoksen elinkaaren näkökulmasta.
EPC	Engineering-Procurement-Construction on yhteisnimitys laitoksen toimitusprojektin suunnittelu-, hankinta- ja rakennusosapuolille. Muita toimitusprojektin osapuolia ovat muun muassa järjestelmätoimittaja ja omistaja-operaattori.
EPL	Eclipse Public License on oikeudellinen sopimus, jolla hallinnoidaan sen alle lisenssioitua materiaalia, kuten esimerkiksi ohjelmistoja.
HTML	Hyper Text Markup Language on standardi kuvauskieli verkkosivujen ja -sovellusten kehittämiseen.
IEC	International Electrotechnical Commission on voittoa tavoittelematon sähköalan standardardointi oraganisaatio.
ISO	International Organization for Standardization on kansainvälinen standardointi organisaatio.
MOF	Model Object Facility on OMG:n standardi mallipohjaiselle suunnittelulle. Se määrittää nelitasoisen arkkitehtuurin metamallintamiselle. Mallintamisessa voidaan käyttää UML-mallinnuskieltä.
OCL	Object Constraint Language on OMG:n määrittelemä mallinnuskieli.
OMG	Object Management Group on kansainvälinen voittoa tavoittelematon teknologiakonsortio, joka kehittää standardeja mallinnuksen tarpeisiin.

OPC	OPC Foundation on teollisuuskonsortio, joka kehittää ja ylläpitää standardeja teollisuusautomaation järjestelmien ja laitteiden yhteensopivuuden näkökulmasta.
OWL	Web Ontology Language on kuvauskieli tiedon esittämiselle ontologioiden avulla.
PandIX	Piping and Instrumation Diagram Exchange on metamalli laitoksen prosessiautomaation funktionaalisen rakenteen kuvaamiseen.
PCA	POSC Caesar Association on voittoa tavoittelematon jäsenorganisaatio, joka pyrkii edistämään ohjelmistojen ja tiedon yhteensopivuutta kehittämällä avoimia spesifikaatioita käytettäväksi osana standardeja.
PI-kaavio	Putki- ja instrumentointikaavio on prosessiteollisuuden toimitusprojektien tärkein suunnitteludokumentti. Se muun muassa esittää prosessilaitteiston, materiaalivirrat sekä miten ja millä näihin virtoihin voidaan vaikuttaa.
PLC	Programmable Logic Controller eli ohjelmoitava logiikka on teollisuusprosessin hallintaan käytetty tietokone.
QVT	Query, View and Transformation on OMG:n määrittelemä joukko standardeja mallimuunnoksille.
RDF	Resource Desciption Framework on W3C:n spesifikaation määrittelemä semanttinen tietomalli.
RDL	Reference Data Library on eräänlainen kirjasto, joka koostuu ISO 15926 standardin osassa 4 määritellyistä luokista ja osassa 7 määritellyistä luokkapohjista.
RDS	Reference Data Service on palvelu, jonka avulla päästään käsiksi RDL:ään
SCADA	Supervisory Control And Data Acquisition on valvomo-ohjelmisto, jolla voidaan seurata ja ohjata prosessia.
SCL	Simantics Constraint Language on Haskellin kaltainen funktionaalinen ohjelmointikieli. Kehitetty Simantics-alustaa varten.
STEP	Standard for the Exchange of Product model data on akronyymi standardille ISO 10303, jota käytetään alusta- ja ohjelmistoriippumattomaan tuotetiedon tiedonsiirtoon ja esittämiseen.
STL	Simantics Transformation Language on Simantics-alustan mallimuunnoskieli.
SOA	Service-Oriented Architecture eli palvelukeskeinen arkkitehtuuri
SPARQL	SPARQL Protocol And RDF Query Language

UA	Unified Architecture on alusta riippumaton palvelukeskeiseen arkkitehtuuriin perustuva viestintäprotokolla.
UML	Unified Modelling Language on yleisluonnollinen, mutta standardoitu, graafinen mallinnuskieli eritoten ohjelmistokehityksen tarpeisiin.
URI	Uniform Resource Identifier on merkkijono resurssin yksilöintiin.
URL	Uniform Resource Locator on URI, mutta sitä käytetään web-resurssien yksilöintiin. Esimerkiksi verkkosivujen osoitteet ovat URL:eja
W3C	World Wide Web Consortium on kansainvälinen järjestö, joka kehittää web-standardeja.
XMI	XML Metadata Interchange on MOF-pohjaisten mallien XML-serialisaatio tiedonsiirtoa varten.
XML	Extensible Markup Language on W3C:n standardoima rakenteellinen kuvauskieli. Käytetään muun muassa tiedon tallentamisen ja tiedonsiirron formaattina.
XSD	XML Schema Definition on XML skeemakieli, jota voidaan käyttää esimerkiksi XML-dokumentin rakenteen määrittämiseen.
XSLT	Extensible Stylesheet Language Transformation on muunnoskieli XML-dokumenttien rakenteen tai formaatin muuttamiseksi.

1. JOHDANTO

1.1 Motivointi

Prosessiteollisuuden suunnitteluohjelmistot ja toimitusprojektin osapuolet ovat vailla yhteisesti hyväksyttyä tapaa vaihtaa suunnittelutietoa keskenään. Laitossuunnitteluprojektiin osallistuu suunnittelijoita eri suunnittelun osa-alueilta. Näistä jokaisella voi olla käytössään toisistaan eriävät laitossuunnitteluohjelmistot ja suurimmalla osalla suunnitteluohjelmistotoimittajista ei ole tarjota kaikkia projektin osa-alueita kattavaa työkalua tai kokoelmaa työkaluja. Ongelman ytimessä ovat ohjelmistokehittäjien toisistaan eriävät tavat toteuttaa tietorakenteita ja -formaatteja, sekä kypsän ja hyväksi todetun standardin puute.

Eräs keino lähestyä edellä esitettyä ongelmaa on standardisointi eli sovitaan yhdenmukaisista toimintatavoista ja edistetään yhteensopivuutta. Suunnitteluohjelmistojen näkökulmasta tämä tarkoittaa avoimen rajapinnan, esimerkiksi yhdenmukaisen tiedonsiirtoformaatin, määrittelyä. Standardi yksistään ei voi pakottaa ketään noudattamaan sitä, sillä standardit eivät ole lakeja, mutta laki voi velvoittaa noudattamaan standardeja. Laitossuunnittelutiedonsiirrolle on esitetty standardeja, mutta niitä ei ole otettu yleisesti käyttöön. Syynä voivat olla standardin puutteellisuus, monimutkaisuus tai ohjelmistokehittäjillä ei ole kiinnostusta implementoida niitä järjestelmiinsä edellä esitetyistä syistä tai kilpailullisista syistä. Ohjelmistotoimittajat voivat tarkoituksenmukaisesti suojata heidän ohjelmistoilla tehdyt suunnitelmat ja ohjelmistorajapinnat, ja siten sitouttaa asiakkaat käyttämään omia tuotteita (vendor lock-in).

Sitouttamisen seurauksena suunnitteluohjelmistojen käyttäjien on hankala siirtyä käyttämään eri ohjelmistoja esimerkiksi yritysostojen yhteydessä. Siirtyminen on hankalaa, koska suunnittelutietokannat eivät ole avoimia ja muiden ohjelmistokehittäjien suunnitteluohjelmistot eivät välttämättä tue vieraita tallennusformaatteja. Avoin pääsy suunnittelutietoon avaa uusia ovia innovaatioille. Uusia ratkaisuja on jo kehitteillä sunnittelutietoon perustuen. Suunnittelutietokantaa voidaan käyttää muun muassa koneoppimissovelluksissa suunnittelun automatisointiin, suunnitelmien virhetarkasteluihin, käytettävän lattia pinta-alan optimointiin sekä materi-

aalien käytön minimointiin.

Eräs mahdollinen motivaatiotekijä yhdenmukaisen formaatin kehittämiselle ja käytölle on ajan säästäminen, mikä puolestaan tarkoittaa rahallista säästöä. Teollisuuden prosessisuunnittelun tuotoksena syntyvä putkisto- ja instrumentointikaavio (PI-kaavio) on keskeinen dokumentti laitossuunnittelun eri vaiheissa. PI-kaaviota käytetään perustana sähkö- ja automaatiosuunnittelussa sekä putkisto- ja laitesuunnittelussa. Jos suunnittelun eri osapuolilla ei ole käytössään työkaluja, jotka kykenevät siirtämään tietoa keskenään, tämä tarkoittaa käytännössä suunnittelutiedon käsin kopioimista. Yhdenmukaisuus ja avoimuus ovat askel eteenpäin suunnittelutiedon siirtämisen ja suunnitteluprosessin automatisoinnille, ja siten myös eräs mahdollisuus säästää laitosprojektin kustannuksissa, niin säästettyjen suunnittelutyötuntien kuin pienentyneiden materiaalimäärien muodossa. Saavutetuilla säästöillä on myös vaikutus ympäristöä kuormittaviin tekijöihin.

1.2 Tavoitteet ja rajaus

Tämä diplomityö on osa VTT:n koordinoimaa Engineering Rulez -hankekokonaisuutta, johon osallistuvia osapuolia ovat PSK Standardisointi, Pöyry, Fortum, Outotec, Fennovoima, Prosys, Semantum, Masinotek, EQUA sekä Aalto-yliopisto. Lisää tietoa hankkeesta voi lukea lähteistä [99, 114].

Tässä diplomityössä tutustutaan prosessi- ja automaatiosuunnittelun tiedonsiirron standardeihin. Käsiteltävät standardit määrittelevät suunnittelutiedolle metamallin sekä tallennusformaatin. Tarkastelu on rajattu PI-suunnittelutiedon siirtoon suunnitteluohjelmistosta toiseen sekä suunnitteluohjelmistosta laitteistorajanpintaan.

DEXPI (Data Exchange for the Process Industry) [18] on saksalaisen teknisen tutkimuslaitoksen DECHEMA:n ProcessNet-hankkeen työryhmä, joka kehittää DEXPI-spesifikaatiota kansainvälisen ISO (International Organization for Standardization) 15926-standardin ja Proteus-skeeman pohjalta. Proteus on Fiatech:n [25] IIMM-projektin [26] (Information models for Process and Instrumentation Diagrams) XML-skeematoteutus[1] (Extensible Markup Language) standardin ISO 15926 laatimasta tietomallista. Työssä paneudutaan seuraaviin kysymyksiin:

- Mitkä ovat PI-suunnittelutiedonsiirtoon soveltuvat standardit?
- Mikä on suunnitteluohjelmistojen valmius tukea DEXPI-spesifikaatiota?
- Miten puolirakenteellisten dokumenttien kypsyyttä voidaan arvioida?

[1] XML-skeemalla määritetään XML-dokumentin rakenne, sisältö ja semantiikka.

- Miten Proteus/DEXPI-malli muunnetaan OPC UA -malliksi?

Suunnitteluohjelmistojen valmiutta tutkitaan avoimesti saatavilla olevien esimerkkitapauksien avulla. Valmiuden tarkastelu tehdään seuraamalla ohjelmistotoimittajien aktiivisuutta ja tutkimalla suunnitteluohjelmistoista tuotujen tiedostojen rakennetta työssä kehitettävän rakennemittarin avulla. Lisäksi toteutetaan muunnostyökalu, jolla voidaan automaattisesti muuntaa Proteus/DEXPI-malli OPC UA -tietomallin mukaiseksi AdressSpace-malliksi. DEXPI-tietomalli on laadittu silmällä pitäen suunnittelutiedon siirtoa eri ohjelmistojen välillä, pääasiassa laitossuunnitteluohjelmistojen kesken. Muuntamalla DEXPI-tietomallin mukainen Proteus-tiedosto (Proteus/DEXPI-malli) OPC UA -tietomallin mukaiseen muotoon, voidaan avata pääsy suunnittelutietoon myös kolmansien osapuolien sovelluksille. Muunnettua tiedostoa voidaan käyttää OPC UA -palvelimen alustamiseen ja näin tarjota avoimen standardin mukainen rajapinta suunnittelutietoon, johon on mahdollista päästä käsiksi OPC UA -asiakaskoneella. Näin ollen laitoksen mallia (PI-kaaviota) voidaan käyttää eräänlaisena tiedonlouhinnan, visualisoinnin, koneoppimisen ja päätöksen teon mahdollistajana. Edellä esitetyt ajatukset ovat lisäksi relevantteja digitalisaation näkökulmasta, kuten esineiden internetin (Internet of Things, IoT) ja Industry 4.0 näkökulmasta.

1.3 Aihepiirin aikaisempi tutkimus

Siltanen ja Pärnänen [97] vertailevat standardien määrittelemiä tietomalleja liiketoiminnan, tekniikan ja käytettävyyden näkökulmasta. Holm et al. [40] aukaisevat lukijalle standardien ISO 15926 ja IEC (International Electrotechnical Commission) 62424 mallikonseptit, sekä vertailevat niiden rakenteellisen ja ajallisen tiedon toteutuksia. Lisäksi he vertailevat konseptien joustavuutta käyttökelpoisuuden ja laajennettavuuden kannalta. Mahnke et al. [58] vertailevat mallien laajennettavuuden lisäksi standardien mallinnusfilosofioita ja konkreettisuutta. Mahnke et al. listaavat sekä laitteistorajapinnan kannalta oleellisia tietomallintamisen konsepteja että millä laajuudella laitteistorajapinnan määrittelevät standardit näitä tukevat.

Yhteenvetona edellä esitellyistä julkaisuista voidaan sanoa, että ISO 15926 on kestänyt hyvin ajan hammasta. Se esiintyy vertailusta toiseen ja saa kiitosta laajasta osaalue soveltuvuudesta sekä elinkaaritiedon mallinnusmahdollisuudesta. Heikkoutena sillä on mallin kompleksisuus ja hankala laajennettavuus. IEC 62424 on tuoreempi standardi, jonka tavoitteena on kuvata tehtaan hierarkkinen rakenne. Sinällään IEC 62424 kalpenee ISO 15926:lle semantiikan ja elinkaaritiedon implementoinnin puutteen vuoksi. Semantiikan puute on korjattu AutomationML:ssa (Auto-

mation Markup Language), joka käyttää topologian esittämiseen standardin IEC 62424 määrittämää XML-skeemaa CAEX:a (Computer Aided Engineering Exchange). PandIX (Piping and Instrumentation Exchange) on otettu myös osaksi AutomationML:a, mikä tuo mukanaan PI-kaavioihin liittyvän semantiikan.

Suunnittelutietoa voidaan hyödyntää laitteistorajapinnassa OPC UA:n avulla. Tavallisesti tietomalli luodaan OPC UA -palvelimelle käsin, mutta jos käytössä on lähde- ja tavoitemalli, voidaan muunnos mallien välillä automatisoida. Standardin IEC 62424 määrittelemä tietomalli ja XML-skeema CAEX on kuvattu OPC UA -tietomalliksi ja kuvantamisen pohjalta on tehty myös muunnostyökalu [91]. Edellä mainittuun standardiin pohjautuva AutomationML on kuvattu OPC UA -tietomalliksi julkaisussa [39], jonka pohjalta AutomationML e.V. on tehnyt OPC Foundationin kanssa liittostandardin (companion specification) [9]. DEXPI- ja OPC UA -tietomallin välille on tämän työn kirjoitushetkellä meneillään liittostandardin laadintaprosessi. Papakonstantinoun ja Karhelan [76] laatimaa alustavaa kuvausta mallien välille tullaan hyödyntämään tämän työn muunnostyökalutoteutuksessa.

1.4 Tutkimusmenetelmä

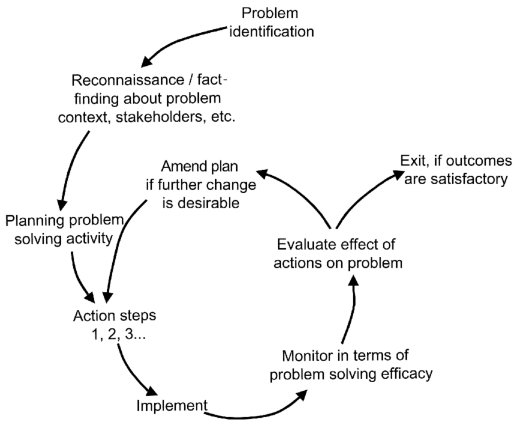
Lähestymistapana edellä esitettyjen kysymysten ratkaisemiseksi käytetään toimintatutkimusmenetelmää (action research). Yksistään jo menetelmän nimi, *toimintatutkimus*, kertoo mistä tässä menetelmässä on kyse; lyhykäisyydessään toiminnasta ja tutkimuksesta, eli käytännöstä ja teoriasta. Menetelmänä toimintatutkimus auttaa tutkijaa kehittymään tutkimuksensa aihealueessa, koska se vaatii tutkijalta aktiivista ja tietoista itsensä sitouttamista tutkimuksen kontekstiin niin teorian kuin käytännön näkökulmasta. [60]

Toimintatutkimukseen kuuluu yhteistyö niin sanotun ongelman omistajan ja tutkijan kesken. Ongelman omistajia voivat olla tutkimusongelman primääriosapuolet joihin ongelma vaikuttaa suoraan, ja/tai sekundääriosapuolet joihin ongelma vaikuttaa välillisesti. Jotta tutkimuksessa saavutettaisiin toimintatutkimuksen kaksinainen päämäärä, on osapuolille eduksi toimia yhteistyössä. Käytännön ongelman ratkaisun ohessa kasvatetaan ymmärrystä kohdejärjestelmästä ja ammennetaan uutta tietoa. [60] Tutkijan näkökulmasta ongelman omistajia voidaan pitää eräänlaisina toimintatutkimuksessa kehitettyjen käytännön menetelmien ja ratkaisujen validaattoreina. Kuten arvata saattaa, prosessina toimintatutkimus on iteratiivinen (ks. kuva 1.1(a)). Jos kehitetty ratkaisu ei läpäise validointia, eli ratkaisu ei tyydyttävästi korreloi ongelman omistajan tarpeisiin, on askelmerkkiä siirrettävä ja aloitettava uusintakierros. Uusintakierrokselle voidaan poiketa myös tapauksessa, jossa toimivaa ratkaisua

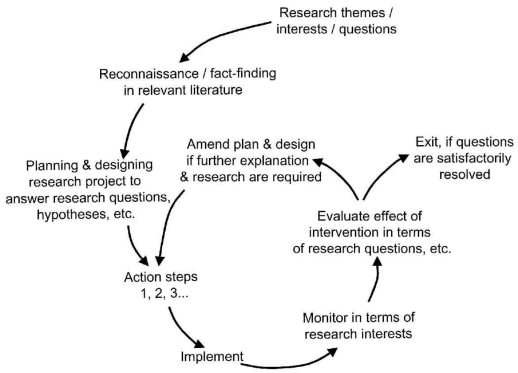
halutaan jatkokehittää. Sama pätee toimintatutkimuksen teoria osuuteen, jos tutkimuskysymyksiin ei olla nykyisellä iteraatiokierroksella saatu vastauksia, edetään uudelle kierrokselle (ks. kuva 1.1(b)).

Kuten MacKay ja Marshall [60] julkaisussaan toteavat, että toimintatapatutkimuksessa käytäntö ja teoria ovat rinnakkaisia, mutta toisistaan riippuvia prosesseja (ks. kuva 1.1(c)). Kiteytettynä toimintatutkimuksessa osallistuvat osapuolet pyrkivät ratkaisemaan tieteelliseltä pohjalta oikean elämän ongelmia. Teoriaan pohjautuvia ratkaisuja testataan kohdejärjestelmää vasten. Tarvittaessa ratkaisuja jatkokehitetään joko potentiaalisen lisäarvon toivossa, tai jos testatulla ratkaisulla ei ollut toivottua impaktia käsillä olevaan ongelmaan. Toimintatutkimus yhdistää oikean elämän ongelmanratkaisumenetelmät ja tieteelliset tutkimuskeinot.

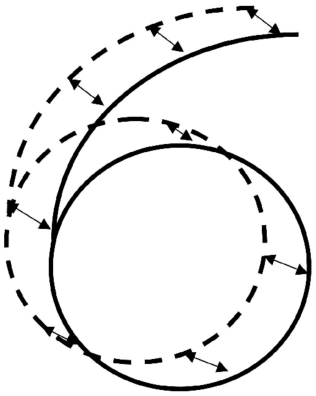
Tässä työssä kehitettävien ratkaisujen validoinnissa hyödynnetään hankeosapuolia ja organisaatioita, jotka ovat ajan tasalla käsiteltävien mallien kehityksestä. Toimintatutkimuksen ajatusta tämän työn kannalta vielä tiivistäen; käytännössä kehitetään mallien välinen kuvaus niiden taustalla vaikuttavien mallien ja konseptien pohjalta, ja työkalu muunnoksen automatisoimiseksi. Työn eteneminen jaetaan toimintavaiheisiin (action steps), joissa kehitystä viedään eteenpäin siten, että teoria ja käytäntö kulkevat rinnakkain. Toimintavaiheet rakentuvat mallien eri konseptien välisistä kuvauksista teorian ja standardien pohjalta, ja kuvausten kanssa rinnalla kehittyvästä muunnostyökalu toteutuksesta. Aikaan saadusta kehitystyöstä pyydetään osapuolilta palautetta ja tehdään tarvittaessa korjausliikkeet, ja aloitetaan palautekierros alusta.



(a) Käytännön näkökulmat toimintatutkimuksessa.



(b) Tutkimuksen näkökulmat toimintatutkimuksessa.



(c) Käytännön ja tutkimuksen rinnakkaisuus toimintatutkimuksen iteratiivisessa prosessissa.

Kuva 1.1 *Toimintatutkimuksen käytännön ja teorian rinnakkaisuus ja riippuvuus.[60]*

1.5 Työn rakenne

Tämä diplomityö on jaettu osiin seuraavanlaisesti:

Luku 2 esittelee metamallintamisen tavoitteet sekä yleisesti käytetyt mallintamisen työkalut ja tekniikat.

Luku 3 esittelee ja vertailee prosessi- ja automaatiosuunnittelun tiedonsiirron standardeja.

Luku 4 tarkastelee suunnitteluohjelmistotoimittajien kehitysprosessin aktiivisuutta ja määrittelee suunnittelutiedonsiirrossa käytetylle formaatille rakennemittarin.

Luku 5 määrittelee luvun 2 tietojen avulla, miten metamallit eri ontologioista voidaan linkittää keskenään. Lisäksi käydään läpi muunnostyökalutoteutus ja käytetyt tekniikat.

Luku 6 kertaa alussa esitetyt laitossuunnittelutiedonhallinnan ongelmia ja työssä esitetyn ratkaisun. Pohdintaa ratkaisun hyödyllisyydestä ja jatkokehitysmahdollisuuksista.

2. METAMALLINTAMISEN TEKNIIKKA

Tässä luvussa esitellään, miten metamallintamista voidaan hyödyntää sunnittelutiedon mallintamiseen ja miten tietomalleja voidaan kuvata mallista toiseen.

Moni on kuullut keskustelua keskustelusta eli metakeskustelusta. Metakeskustelua voi olla esimerkiksi keskustelu aiheesta, miten keskustelu aloitetaan tai miten keskustelua johdetaan. Kun jokin asia suoritetaan kahdesti, voidaan käyttää etuliitettä "meta". Ennen varsinaista keskustelun aloittamista käydään (meta)keskustelu, jossa keskustellaan, miten keskustelu aloitetaan. Sama päättelyketju pätee metamallintamiseen, eli jotakin (meta)mallinnetaan ennen varsinaista mallintamista. Näin ollen voidaan todeta, että malli on metamallin instanssi [49].

Metamallien yhteydessä esiintyvät ontologian- ja semantiikan käsitteet. Ontologian avulla kuvataan metamallin semantiikka. Ontologia on eräänlainen tesaurus, eli käsitesanakirja. Se rakentuu erilaisista konsepteista, jotka muodostuvat sanoista, käsitteistä ja ilmaisuista. Jotta tesauruksesta olisi hyötyä metamallintamisessa, on konseptien välille määritettävä relaatioita, sillä konseptit yksistään eivät muodosta mallia. Metamallin ontologia määrittelee joukon konsepteja ja päättelysääntöjä niiden välisten yhteyksien muodostamiseen. Kun ontologian konsepteille määritetään ongelman aihealueelta merkityksiä, on kyse semantiikasta [90]. Eli metamallin semantiikka määräytyy sen käyttötarkoituksen mukaan.

Metamallintamista voidaan hyödyntää suunnitteluohjelmistojen välisessä tiedonsiirrossa. Metamalli ei ohjaa varsinaista laitossuunnittelutyötä, vaan tässä yhteydessä se esittää miten ohjelmistotoimittaja toteuttaa suunnittelutiedon tallentamisen. Jos ohjelmistotoimittajat käyttäisivät samoja metamalleja tietorakenteidensa toteuttamisessa, ei tiedonsiirto järjestelmien välillä olisi ongelma. On myös ymmärrettävää, että ohjelmistotoimittajat haluavat räätälöidä ohjelmistonsa vastaamaan omia tarpeitaan ja näkemyksiään sillä suunnittelun osa-alueella jolla he toimivat.

Tämänhetkinen trendi laitossuunnittelutiedon siirtämisessä käsin kopioinnin lisäksi on kuvata suunnitelutieto standardin määrittelemän metamallin mukaisesti. Kuvaamisella (mapping) tarkoitetaan tässä yhteydessä eri (meta)mallien semanttisten

vastaavuuksien esittämistä. Kun metamallien semanttiset vastaavuudet ovat selvillä, voidaan mallien muuntaminen automatisoida. Suunnittelutieto esitetään standardin esittämän rakenteen mukaisesti ja vastaanottava osapuoli tietää, että tiedon rakenne noudattaa tiettyä standardia.

2.1 UML-mallinnus

UML (Unified Modeling Language) on OMG:n (Object Management Group) [66] kehittämä yleiskäyttöinen mallinnuskieli. UML:n versio 2.4.1 on hyväksytty ISO/IEC-standardi (ISO/IEC 19505-1:2012 ja 19505-2:2012). [70] Standardin pohjalta on laadittu uudempi UML spesifikaatioversio 2.5, joka yhdistää ja yhdenmukaistaa version 2.4.1 kaksi osiota (Infrastructure ja Superstructure). UML tarjoaa järjestelmä- ja ohjelmistosuunnittelijoille työkalut tietokonepohjaisten järjestelmien analysointiin, suunnitteluun ja toteuttamiseen.[71]

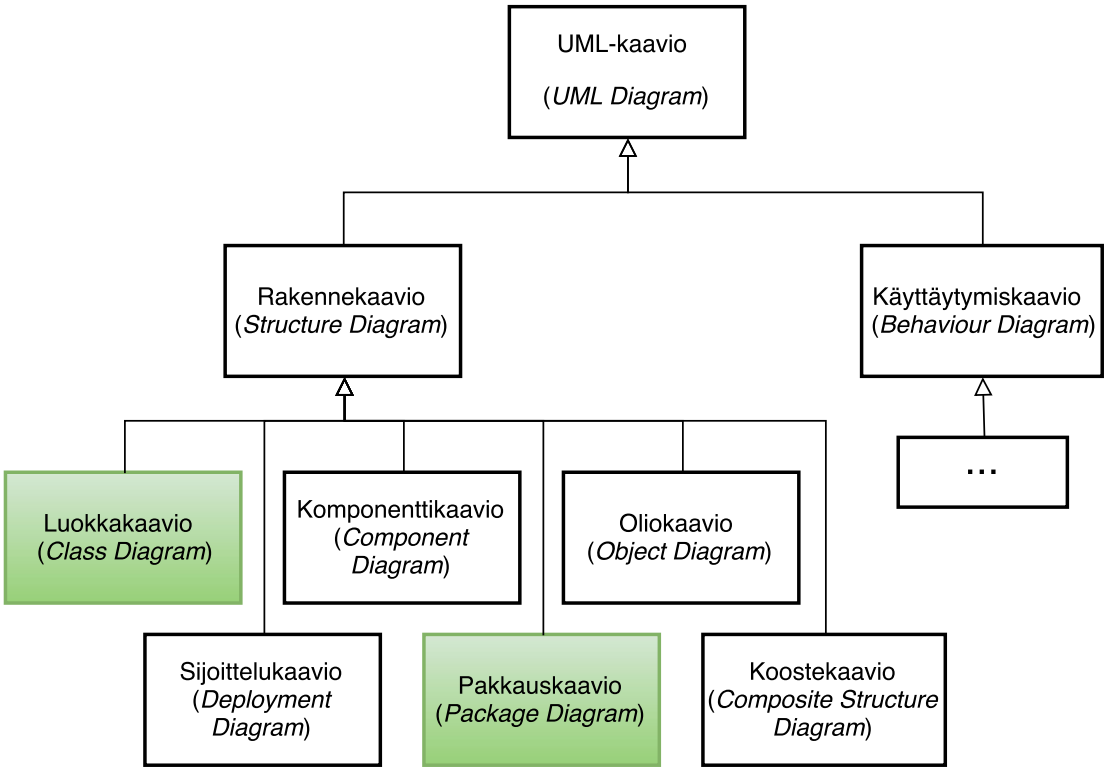
UML määrittelee 14 eri graafista menetelmää järjestelmän mallintamiseen. Nämä menetelmät jakautuvat kahteen pääryhmään: rakennetta ja käyttäytymistä mallintaviin kaavioihin. Rakennekaavioilla kuvataan järjestelmän staattista- eli ajasta riippumatonta rakennetta. Järjestelmän dynaamiset- eli ajasta riippuvaiset ominaisuudet puolestaan kuvataan käyttäytymiskaavioilla. Kuvassa 2.1 on esitetty rakennekaavioiden taksonomia[1], missä kaksi kaaviotyyppiä, luokka- ja profiilikaavio, on korostettu vihreällä. Nämä kaksi kaaviotyyppiä ovat oleellisia suunnittelutiedon metamallintamisessa. Käyttäytymiskaaviot on esitetty toisaalla.

Luokkakaaviolla voidaan mallintaa järjestelmän elementit ja niiden väliset yhteydet. OMG määrittelee luokille semantiikan ja niiden väliset suhteet. [71] Luokkien välillä käytettävät suhteet on esitetty taulukossa 2.1.

UML:n määrittelemä käsitteistö on laajennettavissa ja erikostettavissa mallinnustarpeen mukaisesti laajennustekniikan avulla. UML:n käsitteistöt ovat yleisiä, ja ne eivät ole aina sellaisenaan käytettävissä sovelluskohteisiin tai sitten tyydyttävän lopputuloksen saavuttamiseksi on nähtävä kohtuuton määrä työtä. Määriteltyä yleistä käsitteistöä voidaan käyttää pohjana mallille tarpeellisen käsitteistön laadinnassa.

Stereotyyppi (stereotype) on UML:n mekanismi yleisen käsitteistön erikoistamiseksi. Stereotyypeillä voidaan erikoistaa edellä mainittujen luokkien ja suhteiden lisäksi attribuutteja ja palveluita. Erityistä tarkoitusta varten määritetyt stereotyypit voidaan ryhmitellä *profiilien* (profile) avulla. Profiili on samantyyppinen ryhmittelymekanismi kuin *paketti* (package), jolla voidaan ryhmitellä esimerkiksi luokkia. Oi-

[1] Asioiden hierarkkinen luokittelu.



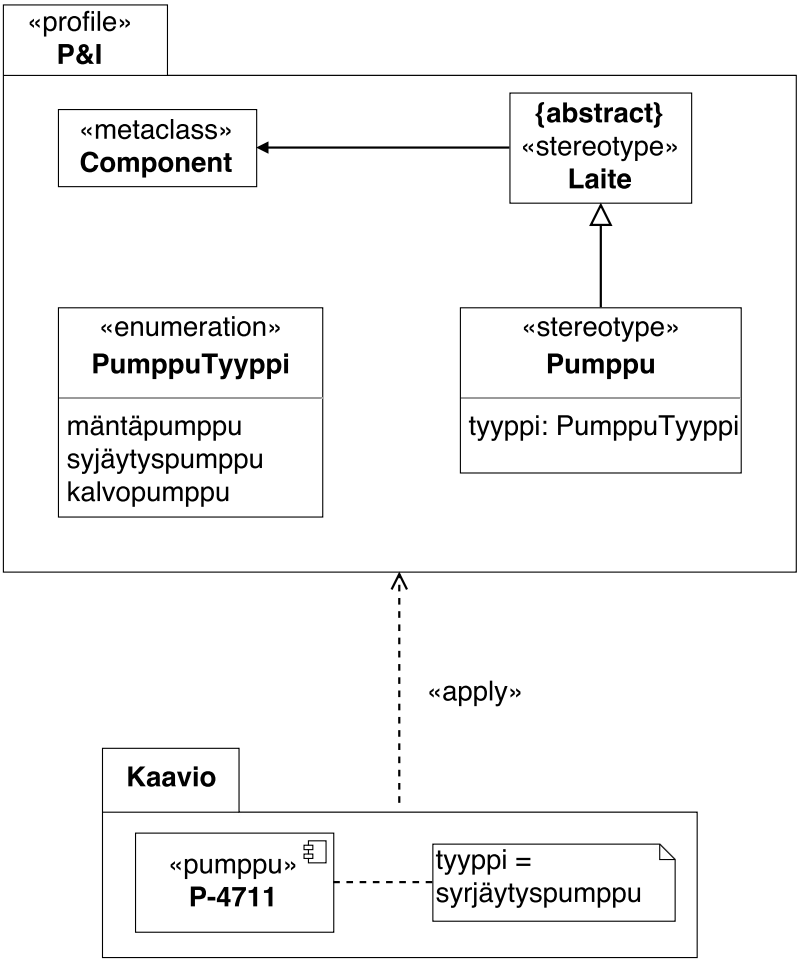
Kuva 2.1 *UML:n kaaviotyypien taksonomia [71].*

keastaan profiili on erikoistus paketista ja sen vuoksi muun muassa profiilin nimeämiskäytännöt vastaavat paketille määriteltyjä [95].

Kuvassa 2.2 on esitetty esimerkki profiilin (≪profile≫) ja stereotyypin (≪stereotype≫) käytöstä. Esimerkin profiili *P&I* voitaisiin kuvitella olevan otos PI-kaavion mallintamiseen tarvittavista metaluokista kaavion pumppujen mallintamiseen. Profiilissa P&I on esitetty kaksi stereotyyppiä *Laite* ja *Pumppu*, joista ensimmäinen on tyypiltään abstrakti ({abstract}). Tämä abstrakti stereotyyppi on yhteydessä laajennussuhteen (extension relationship, ei ole esitetty taulukossa 2.1) kautta metaluokkaan *Component*. Näin määritettynä stereotyypin käyttö metaluokan jatkeena on vapaaehtoista, mutta stereotyypin käyttö on pakotettavissa lisäämällä laajennussuhteelle vaade ≪required≫. Abstraktista Laite-stereotyypistä on edelleen periytyssuhteen kautta johdettu Pumppu-stereotyyppi, joka perii kaikki abstraktin stereotyypin ominaisuudet. Pumppu-stereotyypille on asetettu meta-attribuutti *tyyppi*, jonka arvo voi olla joku *PumppuTyyppi*-listalla (≪enumeration≫) esitetyistä. Profiili voidaan haalia sovelluskäyttöön määrittelemällä profiilin ja sovelluspaketin välille riippuvuus ≪apply≫. Näin esimerkin paketin *Kaavio* nimiavaruuteen (namespace) saadaan tuotua profiilissa P&I määritetyt stereotyypit. Paketissa on komponentti *P-4711*, jolle on määritetty kommentilla *tyyppi*-attribuutti literaaliarvolla *syrjäytyspumppu*. Komponentin attribuutit voitaisiin määritellä myös itse komponenttiin.

Taulukko 2.1 *UML-luokkien välisten suhteiden notaatio. Luokat vaikuttavat toisiinsa tietyllä tavalla. Nämä suhteet kuvataan erilaisilla loogisilla yhteyksillä.*

Nimi	Graafinen notaatio
Kooste (Aggregation)	
Kompositio (Composition)	
Riippuvuus (Dependency)	
Assosiaatio (Association)	
Periytyminen (Inheritance)	
Toteutus (Realization)	



Kuva 2.2 *Esimerkki UML-profiilista ja -stereotyyppistä, sekä stereotyypin käytöstä sovelluksessa. Mukailtu lähteestä [95].*

2.2 Semanttinen mallinnus

RDF (Resource Description Framework) on W3C:n laatima graafitietomalli tiedonsiirtoon semanttisessa web:ssä (Semantic Web) [120]. Tämä tietomalli rakentuu tripleteistä, missä yksi tripletti on yksi väittämä, joka kuvaa kahta resurssia ja niiden välistä yhteyttä. RDF:in graafitietomalli on visualisoitu kuvassa 2.3. RDF:lla tieto voidaan esittää puolirakenteellisena (semi-structured), eli tieto itsessään määrittää oman rakenteensa, toisin kuin rakenteellisissa ratkaisuissa, joissa tiedon rakenne määritetään esimerkiksi erillisellä skeemalla.

Kuva 2.3 *RDF:n graafitietomalli, jossa kahta solmua yhdistää suunnattu viiva. Mukailtu lähteestä [120].*

OWL (Web Ontology Language) on ontologiakieli, joka on kehitetty W3C:n toimesta semanttisen web:in tarpeisiin sillä ajatuksella, että dokumentin sisältämää tietoa on tarkoitus jatkoprosessoida pelkän esittämisen sijaan (vrt. XML) [119]. Puhuttaessa ontologioista viitataan tieteeseen, joka kuvailee elävän elämän entiteettejä ja niiden välisiä yhteyksiä filosofisesta näkökulmasta. Oikeastaan OWL tekee juuri saman. Lisäksi se määrittelee oman tietokoneymmärrettävän kielen ontologioiden esittämiselle, minkä avulla voidaan rakentaa omia alakohtaisia sanakirjoja, joissa termien/entiteettien tarkoitukset on esitetty yksiselitteisesti, mukaan lukien niiden väliset suhteet [119]. OWL:a voidaan käyttää lisäämään muiden XML-pohjaisten tekniikoiden (XML, XML-skeema, RDF, RDF-skeema) semanttista kyvykkyyttä. Koska OWL perustuu RDF:iin, on myös OWL-ontologia puolirakenteellinen. Graafipohjaisten kuvausten lisäksi ensimmäisen kertaluvun logiikkaan perustuvat kuvaukset noudattavat puolirakenteellisuutta [46].

Edellä kuvattua graafitietomallia, joka rakentuu yksittäisistä tripleteistä, voidaan hyödyntää niin tiedon, metatiedon kuin ontologioiden määrittämiseen ja kuvailemiseen. Toisaalta yksittäisistä binaarisuhteista muodostuva tietomalli on hyvinkin yksinkertainen, mutta toisaalta sillä voidaan rakentaa myös monimutkaisia rakenteita. Tripletti-ideaan pohjautuvia tietokantoja kutsutaan triplettivarastoiksi ja RDF-tietomalliin pohjautuvia puolestaan RDF-varastoiksi. [53] Tiedon hakeminen puolirakenteellisista tietokannoista (querying) on vaativampaa verrattuna rakenteellisiin, koska kyselyä tehtäessä ei ole välttämättä tietoa tietokannan rakenteesta, joka voi olla epäsäännöllinen, ja sen rakenne voi erittäin sisäkkäinen tai jopa jaksollinen [2].

Eräs olemassa oleva kyselykieli (query language) triplettivarastojen tietokantakyselyihin on SPARQL (SPARQL Protocol And RDF Query Language) [121]. Tietokantakysely voidaan tehdä myös käymällä tietokanta läpi viittaus viitaukselta [53]. Molemmat edellä esitetyt vaihtoehdot on triplettivaraston näkökulmasta esitetty lähteessä [53] ja yleisesti puolirakenteisen tietokannan kyselyihin liittyviä pohdintoja on esitetty lähteessä [2].

2.3 XML-mallinnus

XML (Extensible Markup Language) on Wide Web Consortiumin (W3C) [116] määrittelemä joustava ja yksinkertainen kuvauskieli. Se on osajoukko standardissa ISO 8879 määritetystä metakielestä SGML:stä (Standard Generalize Markup Language). Toisin sanoen XML on periytetty SGML:stä ja siten XML:lla kuvatut dokumentit vastaavat edellä mainittua standardia. XML:lla on mahdollista esittää dokumentin sisältö tekstipohjaisesti siten, että se on ihmisen ja tietokoneen luettavissa. Se soveltuu hyvin puuhierarkkisten tietorakenteiden esittämiseen ja on siten myös soveltuva muun muassa sunnittelutiedon tallentamiseen ja siirtämiseen [13].

Ohjelmassa 2.1 on esimerkki XML-dokumentista, jossa listataan pumppuja ja niiden tietoja. Ohjelma alkaa prologilla, joka määrittelee dokumentissa käytettävän merkistön ja mitä W3C:n XML-suositusversiota se noudattaa. Prologia kutsutaan joskus myös prosessointikäskyksi ja se on muotoa *<? prosessointikäsky ?>*. Jokaisella XML-dokumentilla on oltava juurielementti, joka on kaikkien muiden elementtien vanhempi (parent). Rivillä kaksi on juurielementin aloitustagi *<PumppuLista>* ja rivillä 13 lopetustagi *</PumppuLista>*. Elementtien sulkeminen on tapahduttava avaamista vastaavassa järjestyksessä, toisin kuin esimerkiksi HTML:ssä (Hyper Text Markup Language) lopetustagin voi jättää pois. Prologilla ei ole lopetustagia, sillä se ei varsinaisesti ole osa XML-dokumenttia. Pumppulistalla on kaksi attribuutein varustettua pumppua ja niillä on edelleen lapsielementtejä (child elements), jotka kuvaavat pumppujen ominaisuuksia. Tässä esimerkissä pumppujen attribuutit määrittävät niiden tyypit. Listan pumput on määritelty eri kielillä ja niiden ominaisuuksia kuvaavat elementit ovat vastaavia, mutta eri järjestyksessä. Tästä huolimatta dokumentti on oikein muotoiltu (well-formed), sillä se noudattaa W3C:n XML 1.0 suosituksen mukaista loogista- ja fyysistä rakennetta.

Yksistään oikean syntaksin noudattaminen XML-dokumenttia laadittaessa, antaa tietosisällön määrittämiselle varsin vapaat kädet. Edeltä voidaan huomata, että esimerkiksi käytettävä kieli ja elementtien järjestys voivat muuttua kesken dokumentin. Jos kieli voi vaihtua kesken dokumentin, niin pumppujen ominaisuuksia kuvaavien elementtien yksikötkin voivat muuttua esimerkiksi metreistä jaloiksi. Tämä

tekee dokumentista hankalasti luettavan niin ihmiselle kuin koneelle ja sisällöstä epäluotettavan. Tämä ongelma on ratkaistavissa määrittelemällä XML-dokumentin rakenne, sisältö ja semantiikka XML-skeemalla. Skeemalla voidaan rajata missä elementit ja attribuutit voivat esiintyä sekä mitä elementeissä voi esiintyä, toisin sanoen määritetään dokumentin metamalli. Myös skeeman tulee olla oikein muotoiltu. Nyt XML-dokumentin syntaksin lisäksi voidaan tarkistaa rakenteen ja sisällön oikeellisuus skeemaa vastaan. [106] Kun XML-dokumentti vastaa skeemaa, se on lisäksi kelvollinen (valid).

```
<?xml version="1.0" encoding="utf-8"?>
<PumppuLista>
   <Pumppu Tyyppi="Keskipakopumppu">
      <Tuotto>1680</Tuotto>
      <MaxPaine>27</MaxPaine>
      <MinLämpötila>-75</MinLämpötila>
   </Pumppu>
   <Pump Type="Reciprocating piston">
      <MaxPressure>315</MaxPressure>
      <Capacity>210</Capacity>
      <MinTemp>-15</MinTemp>
   </Pump>
</PumppuLista>
```

Ohjelma 2.1 *Esimerkki XML-dokumentti.*

Ohjelmassa 2.2 on määritetty pumppulistalle skeema. Ohjelma alkaa ohjelmasta 2.1 tutulla prologilla, jota seuraa skeeman nimiavaruuden (namespace) määrittely. *<xsd:schema ...>* aloittaa skeeman määrittelyn, jossa nimiavaruuden etuliitteksi määritetään *xsd*, sekä itse nimiavaruuden määrittävä URI-osoite (Uniform Resource Identifier) on *http://www.w3.org/2001/XMLSchema*. Osoitteen takaa löytyy W3C:n skeemakielen määrittely, joka sisältää muun muassa dokumentin rakenteen ja primitiivitietotyyppien (kuten string, integer, double) määrittelyt. Riviltä neljä alkaa elementin *PumppuLista* määrittely, joka on monimutkainen tietotyyppi (complex type). Monimutkaiset tietotyypit sisältävät toisia elementtejä, attribuutteja tai molempia. Koska listassa voi olla useita pumppuja, ei pumppujen esiintymisien määrää haluta rajoittaa, joten määritetään elementille *Pumppu* esiintymiskertoja rajoittamaton attribuutti *maxOccurs="unbounded"*. Ympäröidään elementti *Pumppu* elementillä *xsd:sequence*, joka kertoo, että *Pumppu*-elementtejä voi esiintyä useampia tai jos *PumppuLista*-elementille olisi määritelty useampi lapsielementti, olisi niiden esiinnyttävä skeemassa esitetyssä järjestyksessä. Tässä tapauksessa elementin *xsd:sequence* paikalle sopisi myös elementti *xsd:choice*. Seuraavaksi määritetään mitä tietoa *Pumppu*-elementti voi sisältää. *Pumppu*-elementin tietotyyppi on moni-

mutkainen tietotyyppi, koska se sisältää muita elementtejä. Elementit *Tuotto*, *MaxPaine* ja *MinLämpötila* määrittävät pumpun ominaisuudet. Koska ne eivät sisällä muita elementtejä tai attribuutteja, ovat ne tyypiltään yksinkertaista tietotyyppiä (simple type). Nämä kolme elementtiä on ympäröity elementillä *xsd:sequence*, koska elementtien halutaan esiintyvän dokumentissa juuri tässä järjestyksessä. Lopuksi *Pumppu*-elementille määritetään pumpun tyyppiä kuvaava attribuutti *Tyyppi*. Attribuuttimäärittelyt tulevat aina lopuksi.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3
  <xsd:element name="PumppuLista">
5    <xsd:complexType>
       <xsd:sequence>
7      <xsd:element name="Pumppu" maxOccurs="unbounded">
         <xsd:complexType>
9          <xsd:sequence>
             <xsd:element name="Tuotto" type="xsd:integer"/>
11           <xsd:element name="MaxPaine" type="xsd:double"/>
             <xsd:element name="MinLämpötila" type="xsd:double"/>
13         </xsd:sequence>
           <xsd:attribute name="Tyyppi" type="xsd:string"/>
15       </xsd:complexType>
       </xsd:element>
17     </xsd:sequence>
     </xsd:complexType>
19 </xsd:element>

21 </xsd:schema>
```

Ohjelma 2.2 *Esimerkki XML-skeema, joka kuvaa dokumentin rakenteen.*

Ohjelmassa 2.1 esitetty vapaasti muotoiltu XML-dokumentti on muokattu vastaamaan ohjelman 2.2 kuvaamaa skeemaa ohjelmassa 2.3, joka on oikein muotoiltu ja kelvollinen XML-dokumentti. Näin ollen ohjelman 2.2 skeeman mukaiseen dokumenttiin ei *PumppuListaan* voida enää lisätä ohjelmassa 2.1 esitettyä elementtiä Pump. Kuten voidaan huomata, XML-skeeman avulla XML-dokumentit voivat sisältää kuvauksen kantamastaan tiedosta omassa formaatissaan. Edellä esitetyistä XML-dokumenteista saataisiin entistä ilmaisuvoimaisempia määrittämällä oma XML-skeema käytettäville tietotyypeille. Tämä tarkoittaisi sitä, että esimerkiksi pumpun tuottoa kuvaava tietotyyppi ei olisi paljas kokonaisluku, vaan tuottoa oikeasti kuvaava yksikkö, kuten m^3/h (vrt. *<...type="xsd:integer"/>* vs. *<...type="xsö:m3/h"/>*). XML-skeema on yksi tapa sopia tiedonsiirrosta tiedonsiirron osapuolien kesken. Laitossuunnittelutiedon siirrossa XML on vakiinnuttanut ase-

mansa tiedonsiirtoformaattina, kuten luvussa 3 huomataan.

```
<?xml version="1.0" encoding="utf-8"?>
<PumppuLista
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="PumppuLista.xsd">
    <Pumppu Tyyppi="Keskipakopumppu">
        <Tuotto>1680</Tuotto>
        <MaxPaine>27</MaxPaine>
        <MinLämpötila>-75</MinLämpötila>
    </Pumppu>
    <Pumppu Tyyppi="Mäntäpumppu">
        <Tuotto>210</Tuotto>
        <MaxPaine>315</MaxPaine>
        <MinLämpötila>-15</MinLämpötila>
    </Pumppu>
</PumppuLista>
```

Ohjelma 2.3 *Ohjelmassa 2.1 esitetty XML-dokumentti on muokattu vastaamaan ohjelmassa 2.2 määritettyä skeemaa.*

2.4 Mallimuunnos

Mallin muuntamisen (model transformation) määritelmä on ehtinyt muuttua monia kertoja ajan saatossa. Ensimmäisinä lähdemallin (source model) muuntamisesta automaattisesti kohdemalliksi (target model) pohtivat Kleppe et al. [47], joiden mukaan "mallimuunnoksessa kohdemalli generoidaan automaattisesti lähdemallista siten, että se noudattaa muunnosmäärityksiä". Määritelmä on hyvä lähtökohta mallimuunnokselle, mutta se ei ota huomioon useamman mallin muunnosmahdollisuutta. Armani et al. [3] ja Mens et al. [61] ovat tulleet johtopäätökseen, jossa muunnoksen määritelmän olisi oltava vielä yleisemmällä tasolla ja sen olisi otettava huomioon useamman lähde- ja kohdemallin mahdollisuus mallimuunnoksessa. Armani et al. määrittelevät mallimuunnoksen seuraavasti: "lähdemallin (-mallien) automaattinen manipulointi, joka vastaa kuvausta, tuottaa kohdemallin (-malleja) tietyn aikomuksen mukaisesti." [3] Edellä esitetty muunnoksen määritelmä vastaa tämän työn aikeita:

- Muunnos on automaattinen toimenpide
- Muunnoksen kuvaus on erotettavissa toteutuksesta
- Muunnoksella on tietty tarkoitus
- Muunnos tuottaa lähdemallista kuvauksen mukaisen kohdemallin

Itse mallimuunnos perustuu mallien väliseen kuvaukseen, eli mallien keskinäisten vastaavuuksien esittämiseen valitun teknisen avaruuden mukaisesti (technological space, technical space, domain language), ja näiden kuvauksien perusteella laadittuihin muunnossääntöihin (transformation rules). Teknisellä avaruudella [11, 50] tarkoitetaan tässä yhteydessä mallimuunnoksen hallintaan käytettävää ohjelmistokehystä. Esimerkkinä teknisestä avaruudesta mainittakoon XML, joka määrittelee kuvaamiselle ja muuntamiselle tarvittavat kielet (XML, XSLT, XMI), ja tarvittavan (meta)mallin (XML-skeema).[61] Muunnossäännöt ovat muunnostoteutuksen pienimpiä palasia. Kun lähdemallista löydetään halutunlainen rakenne, muunnetaan se vastaamaan kohdemallin mukaista rakennetta muunnossääntöjen avulla. [16] Valittu mallimuunnoksen määritelmä sallii yksi-yhdeksi -muunnoksen lisäksi yksi-moneksi- (one-to-many) ja monta-yhdeksi- (many-to-one) muunnokset, mutta näitä kahta viimeiseksi mainittua mallimuuntamisen lähestymistapoja ei käsitellä tässä työssä.

Mallimuunnoksella on suunta (directionality), joka kuvaa onko muunnos tehtävissä vain yhteen suuntaan, esimerkiksi lähdemallista kohdemalliksi (yksisuuntainen muunnos, unidirectionality), vai myös muihin suuntiin (monisuuntainen muunnos, multidirectionality). Monisuuntainen muunnos on tehtävissä käyttämällä monisuuntaisen muunnoksen muunnossääntöjä tai määrittelemällä joka munnossuunnalle omat erilliset muunnossäännöt. [16] Yksistään kaksisuuntaisen (bidirectional) mallimuunnoksen vaatimuksia on esitetty lähteessä [104]. Tässä työssä toteutettava mallimuunnos on yksisuuntainen.

Eräs hyödyllinen mallimuunnoksen ominaisuus on muunnoksen inkrementaallisuus, eli jos lähdemalli muuttuu, miten muutokset viedään kohdemalleihin. Czarnecki ja Helsen [16] esittelevät inkrementaalisuudelle kolme käsitettä: Kohdeinkrementaalisuus (target incrementality), lähdeinkrementaalisuus (source incementality) ja käyttäjän muokkausten säilyttäminen kohdemallissa (preservation of user edits in the target). Kohdeinkrementaalisessa muunnoksessa kohdemalliin päivitetään lähdemalliin tehdyt muutokset. Lähdeinkrementaallisessa muunnoksessa pyritään jäljittämään lähdemalliin tehdyt muutokset ja muuntamaan vain tehdyt muutokset. Näin voidaan välttää koko lähdemallin läpikäyminen. Käyttäjän muokkausten säilyttäminen kohdemallissa -tapauksessa käyttäjä on tehnyt kohdemalliin muutoksia ja lähdemalliin tehdyt muutokset halutaan muuntaa lähdemalliin siten, että käyttäjän tekemät muutokset säilyvät.[16]

Extensible Stylesheet Language Transformation (XSLT) on W3C:n XML-kuvauskielelle määrittelemä muunnoskieli, jonka avulla voidaan XML-dokumentteja muuntaa muiksi XML-pohjaisiksi dokumenteiksi, kuten esimerkiksi toiseksi XML-dokumentiksi, HTML-dokumentiksi tai vaikkapa CSV-tekstitiedostoksi (comma-separated va-

lues). [115] XSLT:n syntaksi ja semantiikka on määritetty lähteessä [118]. Muunnos XSLT-kielellä esitetään eräänlaisena tyyliohjeena (stylesheet), jonka tarkoituksena on muotoilla XML-dokumentista (lähdemalli) muunnossääntöjen avulla halutunlainen dokumentti (kohdemalli). Tyyliohje noudattaa aikaisemmin kappaleessa 2.3 esitettyjä oikein muotoillun XML-dokumentin vaatimuksia. XSLT-spesifikaatio käyttää syötedokumenteista nimitystä lähdepuu (source tree) ja muunnoksen tuotoksista nimitystä kohdepuu (result tree). Varsinaisen muunnoksen lähdedokumenttien ja XSLT:n pohjalta toteuttavasta ohjelmistotyökalusta käytetään nimitystä XSLT-prosessori (XSLT processor).[118]

QVT (Queries, Views, Transformations) [67] on OMG:n laatima spesifikaatio mallimuunnoksille. QVT määrittelee kolme muunnoskieltä, joilla muunnokset voidaan toteuttaa: Core (QVTc), Relational (QVTr) ja Operational Mappings (QVTo). QVT:n muunnoskielistä QVTr on mielekkäin tämän työn kannalta, sillä se soveltuu käyttäjän määrittämien muunnoksien mallintamiseen, tukee inkrementaalisuutta, muunnokset voivat olla yksi- tai monisuuntaisia ja kieli tukee myös rakenteiden yhteensovittamista (pattern matching) muunnettavien objektien välillä. [67] Kieli soveltuu MOF-konseptein (Meta Object Facility) mallinnettujen mallien muuntamiseen, ja se hyödyntää OCL:a (Object Constraint Language) muunnoskielen vakiokirjastona (standard library). OCL on OMG:n laatima rajoitekieli, joka on määritetty lähteessä [68]. [67] MOF on OMG:n määrittelemä avoin ja alusta riippumaton ohjelmistokehys metamallintamiseen ja metatiedon käsittelyyn [69]. MOF:n spesifikaatioversio 2.4.1 on julkaistu ISO-standardina ISO/IEC 19508 "*Information technology - Object Management Group Meta Object Facility (MOF) Core*", mutta ajantasaisin spesifikaatioversio on 2.5.1. OMG käyttää MOF:ia eri tekniikoidensa rakenteiden määrittelyyn. Muun muassa QVT, XMI (XML Metadata Interchange), UML ja itse MOF on määritetty MOF:n konseptein.

3. PROSESSI- JA AUTOMAATISOUUNNITTELUTIEDON TIEDONSIIRRON STANDARDIT

Tässä luvussa esitellään prosessi- ja automaatiosuunnittelun tiedonsiirron standardeja ja niiden pohjalta tehtyjä spesifikaatioita ja toteutuksia. Esiteltävillä standardeilla on yhteinen tavoite, mutta erilaiset lähestymistavat. Nämä standardit pyrkivät määrittämään geneerisen tietomallin laitossuunnittelutiedolle ja avoimen tavan siirtää sitä. Käyttämällä yhdenmukaisia formaatteja laitossuunnittelun eri vaiheessa voidaan välttyä redundantilta työltä.

3.1 IEC 62424

IEC 62424 "*Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*" [42] on nimensä mukaisesti standardi laitossuunnittelutiedon tiedonsiirtoon. Standardi määrittää miten säätösuunnittelussa (process control engineering) tarvittava tieto esitetään PI-kaavioissa siten, että laitossuunnittelutiedon tiedonsiirto voidaan automatisoida toimitusprojektin osapuolien kesken. Päästäkseen tähän tavoitteeseen IEC 62424 määrittelee tietomallin nimeltä CAEX (Computer Aided Engineering Exchange), joka on kuvattu XML-skeemana. Näin tiedon rakenne ja semantiikka saadaan yhdenmukaistettua. CAEX:lla ei voi siirtää suunnitelmien piirrosmerkkejä. [42]

CAEX on hyvä lähtökohta järjestelmä- ja ohjelmistotoimittajariippumattoman laitossuunnittelutiedon tiedonsiirtoformaatiksi. CAEX keskittyy siirtämään PI-kaavioihin liittyvää laitossuunnitelutietoa, mutta sillä on mahdollista siirtää myös muuta relevanttia tietoa toimitusprojektin osapuolien kesken [40] varsinkin, jos tiedon rakenne on PI-kaavioiden tavoin hierarkkinen. CAEX:n vahvuutena ja toisaalta heikkoutena on sen tietomallin laajennettavuus. Malli on laajennettavissa määrittelemällä uusia rooleja, mutta samaa roolia kuvaava objekti on mahdollista määrittää eri tavoin. [94] Esimerkiksi rooli *MänttäPumppu* voidaan kuvata myös roolilla *Pumppu*, jolla on attribuutti *Tyyppi*, jonka arvoksi asetetaan *Mäntä*. Kolmas

vaihtoehto on esitetty aikaisemmin ohjelmassa 2.3. Roolit ovat CAEX:in tapa kuvata mallinnettavan järjestelmän objekteja ja semantiikkaa. Roolit määritetään ja kootaan yhteen rooliluokkakirjastoihin (role class libraries) [92].

3.1.1 PandIX

PandIX [14] (Piping And Instrumation Diagram Exhange) on metamalli PI-kaaviotiedon toiminnallisen rakenteen esittämiseen. Sen voidaan ajatella olevan laajennus edellä esitettyyn CAEX-tietomalliin, jota voidaan käyttää monenlaisiin hierarkkisiin ja modulaarisiin järjestelmiin. PandIX:n puolestaan keskittyy prosessilaitoksen elementtien ja niiden välisien yhteyksien esittämiseen prosessiautomaation näkökulmasta. Elementtejä ovat muun muassa tankit ja toimilaitteet, ja yhteyksiä putket ja säätösilmukat. Toisin sanoen PandIX jäsentää CAEX-tietomallista toimilaite- ja instrumentointisuunnittelun kannalta oleellisen tiedon omaksi mallikseen.

PandIX ei mallinna tuote- ja prosessikohtaista tietoa, kuten pumppujen tuottoa tai prosessin virtausnopeuksia, vaan mitä mittaustietoa tarvitaan toimilaitteen ohjaamiseen [94]. PandIX:lla voi tarvittaessa siirtää myös tietoa suunnitelman piirrosmerkkeistä viittaamalla standardissa ISO 10628 määriteltyihin piirrosmerkkityyppeihin standardin ISO 1416 nimeämiskäytännön mukaisesti [109]. Tarvittaessa PandIX-mallia on mahdollista tutkia ja selata laitoksen toiminnan aikana [40]. PandIX:lla on lähinnä akateemista tukea ja sitä ei ole sinällään otettu osaksi suunnitteluohjelmistoja.

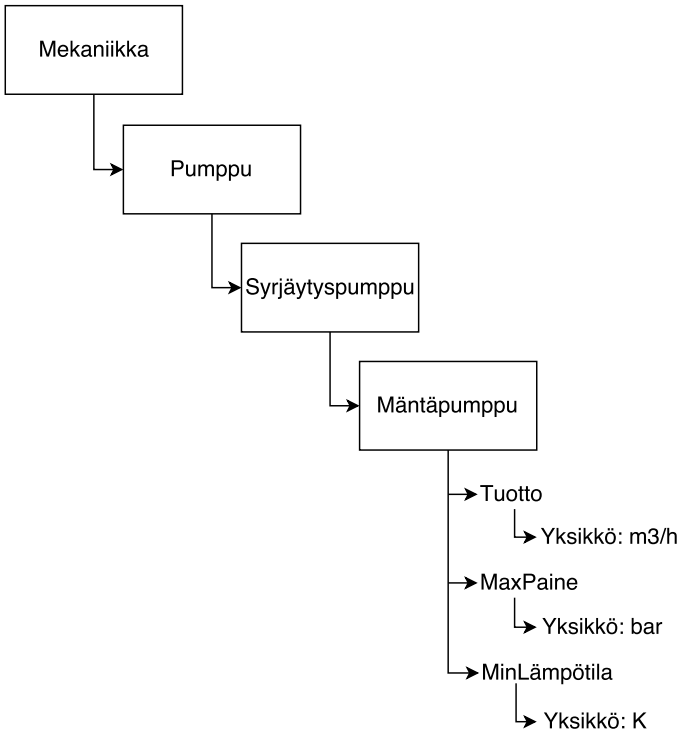
3.1.2 Automation Markup Language

Automation Markup Language [6] (AutomationML, AML) on CAEX:n tavoin laatinut tiedonsiirtoformaatin XML-skeemana, jonka tavoitteena on yhdenmukaistaa automaatiosuunnittelun tiedonsiirto toimitusprojektin osapuolien käyttämien suunnitteluohjelmistojen välillä usealla eri suunnittelun osa-alueilla. Mikä erottaa AutomationML:n CAEX:sta on se, että AutomationML ottaa CAEX:in osaksi itseään suunnittelun eri osa-alueiden formaattien väliseksi integraattoriksi sekä hierarkisen laitostopologian esittämiseksi. AutomationML käyttää siis topologian formaattina CAEX:ia ja lisäksi COLLADA^{TM} kinematiikan sekä geometrian formaattina, ja PLCOpen XML:a logiikkatiedon formaattina. Edellä mainitut formaatit perustuvat XML-kuvauskieleen ja ne yhdistetään käyttäen CAEX:a. AutomationML määrittää mitä CAEX:n elementtejä käytetään sen missäkin rakenteessa [57].

Koska AutomationML käyttää CAEX:a tiedonsiirtoformaattinsa ytimenä, on sen laajennusmekanismi myös sama. Toisin kuin CAEX:ssa, AutomatonML:ssa on valmiiksi määriteltyjä rooliluokkakirjastoja. “Roolit ovat luokkia, jotka kuvaavat abstraktin toiminnallisuuden ilman varsinaista teknistä implementaatiota.”[6] Tällöin muun muassa dokumentin rakenne on tiukemmin tyypitetty ja samalla se ohjaa suunnittelijan työtä. Jos valmiiksi laadituista kirjastoista ei löydy tarvetta vastaavaa ratkaisua, on edelleen mahdollista laatia omat räätälöidyt kirjastonsa. Esimerkiksi PI-kaavion putkisto, instrumentointi ja prosessin säätöön liittyvän tiedon mallintaminen onnistuu PandIX:sta tehdyllä käyttäjämääritetyllä rooliluokkakirjastolla. [4]

Toimitusprojektin edetessä tulee tarve määrittää suunnitelman eri elementeille tietosisältö eli semantiikka. Esimerkiksi suunnitelmassa olevalla pumpulla on valmistaja, enimmäistuotto ja muita teknisiä ominaisuuksia, ja näille tiedoille on tietomallissa valittu attribuutit kuvaamaan tuota tietoa. Jotta suunnittelutieto saadaan siirrettyä toimitusprojektin osapuolien kesken häviöttä ja ilman väärää tulkintaa, on projektin etenemisen kannalta edukasta sopia käytettävistä nimeämiskäytännöistä. Teollisuuden käyttötarkoituksiin on olemassa useita elementtien nimeämiseen, määrittämiseen ja niiden tietorakenteeseen liittyviä katalogimaisia standardeja. Teollisuudessa tähän ongelmaan pureutuvia standardeja ovat IEC 61360 “*Standard data element types with associated classification scheme for electric components*” ISO 13584 “*Industrial automation systems and integration* ” ja IEC 61987 “*Industrial-process measurement and control - Data structures and elements in process equipment catalogues*”, jonka tietomalli noudattaa kahdessa ensimmäisessä mainitussa standardissa kuvattua [44].

AutomationML on määritellyt tapoja [5], joilla tällaisia palveluja, tuotteita ja niiden materiaaleja ryhmitteleviä katalogeja voidaan valjastaa elementtien semantiikan erikoistamiseksi. eCl@ss on eräs IEC 61360 standardiin perustuva maksullinen palvelu, jonka määrittelemä katalogi on mahdollista valjastaa osaksi AutomationML:a. Katalogi on saatavissa CSV- ja XML-formaatissa. eCl@ss:in tuotteiden ja palveluiden luokkarakenne on nelitasoinen ja esimerkki tästä on esitetty kuvassa 3.1. Kuvassa eCl@ssin ensimmäiseltä luokkatasolta (segmentti, segment) on valittu luokka *Mekaniikka*, jonka alapuolella on toiselta luokkatasolta (pääryhmä, main group) valittu *Pumppu*. Tämän luokan alapuolella on kolmas luokkataso (ryhmä, group) *Syrjäytyspumppu* ja viimeiseltä neljänneltä luokkatasolta (hyödykeluokka, commodity class) *Mäntäpumppu*. Itse hyödykkeelle on määritetty joukko ominaisuuksia yksiköineen. Vertaa luokittelua ohjelman 2.2 latteampaan pumppuluokittelutoteutukseen. eCl@ssin luokitusjärjestelmä on esitettävissä myös numeerisesti. Kuvan 3.1 mäntäpumppu on tunnistettavissa myös numerosarjalla 36-41-05-04, joka on eCl@ssin numerojärjestelmän mukainen. Tarkalleen ottaen kyseessä on radiaalimäntäpumppu. Numerosarjan jokainen segmentti kuvaa yhtä luokkatasoa, jossa



Kuva 3.1 *Esimerkki eCl@ss:in mukaisesta nelitasoisesta tuoteluokittelusta.*

ensimmäinen kuvaa ylintä tasoa ja viimeinen alinta eli tarkinta luokkatasoa. Lisäksi laitteilla on lista ominaisuuksia kuvaavia attribuutteja, esimerkiksi mäntäpumpulla on attribuutti *Tuotto*, jonka yksikkö on m^3/h.

AutomationML:n tukijoita ovat muun muassa ABB, KUKA ja Siemens. Myötävaikuttajia ovat muun muassa Airbus Group, Festo ja EKS. Akateemisia tukijoita muun muassa Ifak, RWTH ja Fortiss. Täydellinen listaus AutomationML:n kotisivuilta [7]. Sivustolta ei käy ilmi onko kukaan AutomationML e.V.:n jäsenistöstä ottanut AutomationML:n ohjelmistoihinsa.

3.2 ISO 15926

ISO 15926 "*Industrial automation systems and integration – Integration of lifecycle data for process plants including oil and gas production facilities*" [43] on kansainvälinen standardi teollisuuden mallintamisen ja tiedonsiirron tarpeisiin. Nimestään huolimatta, standardi soveltuu myös muille teollisuuden osa-aluille, kuin öljy- ja kaasulaitoksille. Se määrittää yleisen tason tietomallin, joka ottaa huomioon [38] muun muassa laitoksen toimitusprojektin ja käyttövaiheen osapuolien tarpeet elinkaaritiedon näkökulmasta.

ISO 15926 on jatkotarina standardille ISO 10303 "*Automation systems and integration — Product data representation and exchange*", joka tunnetaan paremmin nimellä STEP (Standard for the Exchange of Product model data). Kuten STEP:n akronyymikin kertoo, sen tavoitteena on mallintaa ja siirtää tuotetietoa. [52]

Standardin IEC 61360 tavoin ISO 15926-4:ssa on määritetty katalogimainen kirjasto teollisuudessa käytetyistä termeistä, mitä kutsutaan standardissa myös nimellä RDL (Reference Data Library). Termit on kuvattu RDL-luokkina, jotka ovat instansseja standardissa ISO 15926-2 määritellyistä tietotyypeistä. Eli standardin toinen osa määrittää miten standardin mukaisia termejä kuvataan ja standardin neljäs osa kuvaa termien semantiikan. [32, 48] Kuvassa 3.2 on esitetty kuvaa 3.1 vastaava laiteluokittelu ISO 15926-4:n mukaisesti. Kuvasta on nähtävissä, että ISO 15926:n tietomalli on määritelty hyvin abstraktilla tasolla. ISO 15926-4:n mahdollisia yksilöitä ovat asiat, jotka ovat olemassa ajassa ja paikassa.

Standardin alullepanija PCA [78] (POSC Caesar Association) ylläpitää RDL-kirjastoa ja tarjoaa avoimen pääsyn kirjastoon RDS-palvelun (Reference Data Service) avulla. Kun ohjelmistotoimittaja haluaa tietorakenteessaan määritellä ja kertoa selvästi minkälaisesta elementistä on kyse, voi hän viitata RDS-palvelun RDL-kirjastoon. Esimerkiksi mäntäpumppu voidaan määrittää viittaamalla osoitteeseen

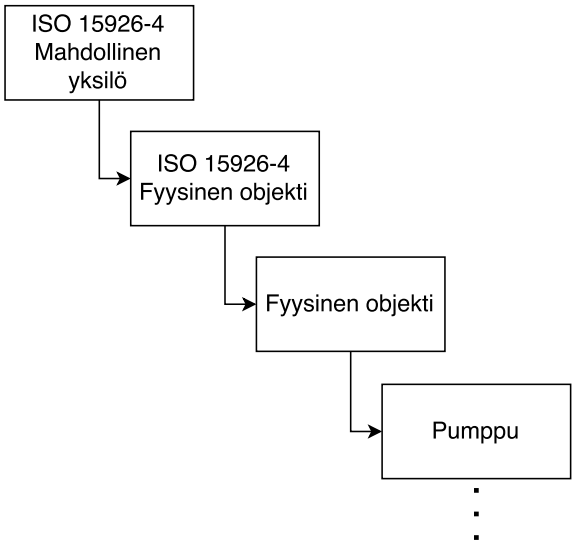
http://data.posccaesar.org/rdl/RDS393992051

ja jonka tuoton yksikkö on m^3/h viittaamalla osoitteeseen

http://data.posccaesar.org/rdl/RDS8128553985.

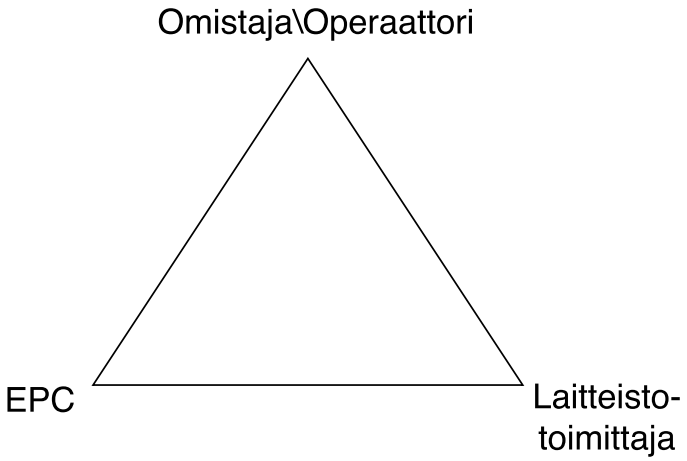
3.2.1 Proteus/DEXPI

DEXPI (Data Exchange in the Process Industry) [18] on Decheman johtaman ProcessNet-hankkeen alainen työryhmä, jonka tavoitteena on edistää ja kehittää tiedonsiirtomenetelmää petrokemianteollisuuden tarpeisiin. DEXPI:n tiedonsiirtomalli perustuu edellä esiteltyyn standardiin ISO 15926 ja sen pohjalta tehtyyn Proteus-skeematoteutukseen. Malli on esitetty lähteessä [105] ja sen tavoitteena on valjastaa PI-kaavioiden graafinen-, topologinen- ja laitekohtainen tieto konseptuaalisen mallin avulla. Proteus-skeema, johon DEXPI-tietomalli perustuu, on jatkoa Noumenonin XMpLant-skeemalle. Tämän hetkinen Proteus-skeeman luonnosversio 4.0.1 on tarkoitettu vain PI-kaaviotiedon tiedonsiirtoon, mutta aikaisemmissa skeemaversioissa on tuettuna 3D-objektien tiedonsiirto.



Kuva 3.2 *Esimerkki standardin ISO 15926 mukaisesta laiteluokittelusta*

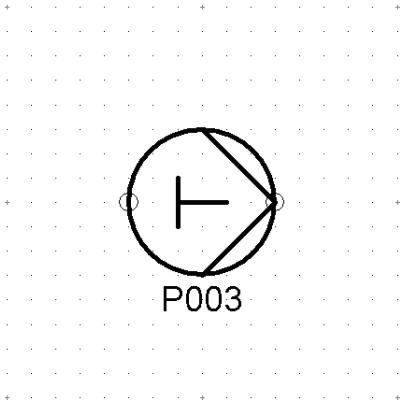
DEXPI-työryhmän tavoitteena on kehittää tiedonsiirtoformaattia laitoksen toimitusprojektin kolminaisuuden tarpeisiin. Kuvassa 3.3 esitetty kolminaisuus kuvaa toimitusprojektin osapuolet. Toimitusprojektiin kuuluvat omistaja-operaattori, joka tilaa laitoksen ja käyttää sitä, laitteistotoimittaja(t) ja EPC (Engineering-Procurement-Construction), joka vastaa laitoksen suunnittelusta, hankinnasta ja rakentamisesta. Näiden osapuolien kesken on tarvetta siirtää tietoa. Esimerkiksi ECP toimittaa laitoksen dokumentaation ja ohjaus- ja kunnossapitojärjestelmät omistaja-operaattorille sekä hankintatarpeet laitteistotoimittajalle. Jo pelkästään EPC:n alla on useita eri osa-alueiden toimijoita joiden on vaihdettava tietoa keskenään; suunnittelun osa-alueella esimerkiksi PI-, sähkö- ja automaatiosuunnittelu.



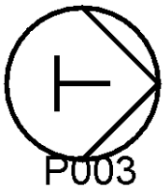
Kuva 3.3 *Toimitusprojektin osapuolet.*

DEXPI hyödyntää PCA/RDS RDL -palvelua tietomallinsa elementtien määrittelyyn. Ohjelmassa 3.1 on esitetty ohjelman 2.3 mäntäpumpun implementaatio Proteus-skeeman (versio 4.0.1) mukaisesti. Varsinainen mäntäpumppuun liittyvä tieto on elementtien *<Equipment...>* ja *</Equipment>* välissä, jonka attribuutit *ID* yksilöi kyseisen pumpun; *ComponentClass* kertoo mihin luokkaan laite kuuluu ja *ComponentClassURI* on viittaus osoitteeseen, jossa laiteluokka on määritelty. Laite-elementillä on edelleen lapsielementti *<GenericAttributes>* (yleiset attribuutit), jolla on useampia *<GenericAttribute>*-lapsielementtejä (yleinen attribuutti). Yleinen attribuuttielementti määrittää kokoelman yksittäisiä yleisiä attribuutteja. Laite-elementillä voi olla myös useampia attribuuttikokoelmia ja nämä kokoelmat erotellaan toisistaan määrittelemällä kokoelman attribuutilla *Set*. DEXPI-spesifikaation mukaiselle attribuuttikokoelmalle annetaan attribuutille *Set* arvo *DexpiAttributes*. Yleisen attribuuttielementin attribuuteilla voidaan määritellä sille esimerkiksi nimi, arvo ja yksikkö sekä viittaukset osoitteisiin, joissa kyseinen yleinen attribuuttielementin luokka ja yksikkö on määritetty (ks. esimerkiksi ohjelman 3.1 rivit 23-26). DEXPI-tietomallille muuta oleellista tietoa on esitetty ohjelman riveillä 5-13. Rivit sisältävät tietoa esimerkiksi mitä skeemaversiota dokumentti noudattaa, mistä järjestelmästä XML-dokumentti luotu ja milloin.

Kuvassa 3.4 on esimerkki, jossa laitossuunnitteluohjelmistolla on piirretty mäntäpumppu (a), joka on seuraavaksi viety Proteus-skeeman mukaiseen Proteus-tiedostoon. Tämän jälkeen tiedosto on luettu toisessa ohjelmistossa, jossa grafiikkaobjekti on rekonstruktuoitu (b). Laitetiedot ja laitteen graafinen esitys sidotaan toisiinsa laitenimen perusteella (ohjelmassa 3.1 rivillä 14: *ComponentName*).



(a) Mäntäpumpun graafinen esitys COMOS-laitossuunniteluohjelmistossa.



(b) Simantics-alustalle COMOS-laitossuunnitteluohjelmistosta tuotu mäntäpumppu-grafiikkaobjekti.

Kuva 3.4 *Esimerkki Proteus-tiedostolla Simantics-alustalle tuodusta Simenesin COMOS-laitossuunnitteluohjelmiston mäntäpumppu-grafiikkaobjektista.*

```
<?xml version="1.0" encoding="utf-8"?>
<PlantModel
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <PlantInformation SchemaVersion="4.0.1" OriginatingSystem="xXx"
    Date = "2017-05-16" Time = "10:00:00+02:00"
    Is3D = "no" Units = "mm" Discipline="PID">
    <UnitsOfMeasure/>
  </PlantInformation>
  <Extent>
    <Min X="0" Y="0"/>
    <Max X="100" Y="100"/>
  </Extent>
  <Equipment ID="Dippa1" ComponentName="Pumppu1"
    ComponentClass="PistonPump"
    ComponentClassURI="http://data.posccaesar.org/rdl/RDS393992051">
    <GenericAttributes Number = "3" Set="DexpiAttributes">
      <GenericAttribute Name="FlowRate" Value="210" Units="m3/h"
        AttributeURI="http://data.posccaesar.org/rdl/RDS1059668431"
        UnitsURI="http://data.posccaesar.org/rdl/RDS393992051"/>
      <GenericAttribute Name="MaximumPressure" Value="315" Units="bar"
        AttributeURI="http://data.posccaesar.org/rdl/RDS7344775"
        UnitsURI="http://data.posccaesar.org/rdl/RDS1314539"/>
      <GenericAttribute Name="LowerLimitTemperature" Value="-15"
        Units="DegreeCelsius"
        AttributeURI="http://data.posccaesar.org/rdl/RDS400229"
        UnitsURI="http://data.posccaesar.org/rdl/RDS1322684"/>
    </GenericAttributes>
  </Equipment>
</PlantModel>
```

Ohjelma 3.1 *Proteus-skeeman mukainen toteutus ohjelmassa 2.3 esitetystä mäntäpumpun määrittelystä.*

DEXPI-spesifikaatiota tukevia yrityksiä omistaja-operaattori -näkökulmasta ovat BASF, Bayer ja EVONIK. Tutkimustyötä spesifikaation eteen tekevät RWTH Aachen, AixCAPE ja VTT. Ohjelmistokehittäjistä Proteus/DEXPI-tietomallin käyttöön ottaneita ovat muun muassa Autodesk, Aveva, Integraph, Siemens ja X-visual. [19]

Tässä työssä selvitetään Autodeskin AutoCAD P&ID:n, Aveva P&ID:n, Integraph SmartPlantin sekä Siemens COMOS:in valmiutta tukea DEXPI-spesifikaatiota.

3.3 PSK Standardisointi

Tässä luvussa esitellään PSK Standardisoinnin toimintaa, sekä sen laatimia suunnittelutiedonsiirron kannalta oleellisia standardeja.

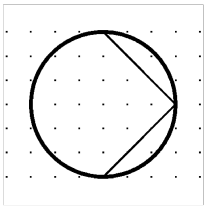
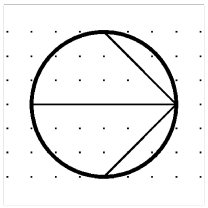
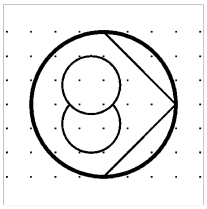
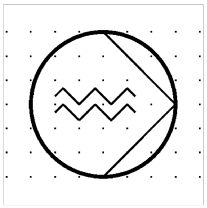
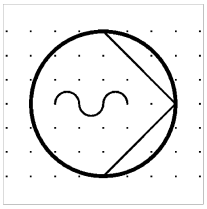
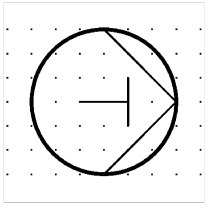
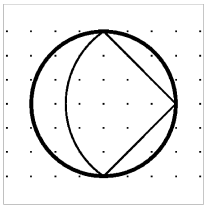
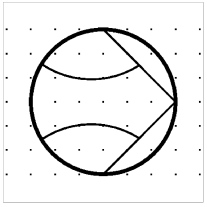
PSK Standardisointi (PSK Standardisointi ry, PSK) [81] on riippumaton kansallisen prosessiteollisuuden standardisointielin, joka tuottaa sekä ylläpitää PSK- ja SFS-standardeja. Teollisuutta tukeva standardisointi vaikuttaa sitä palveleviin yrityksiin, niin kotimaisessa kuin kansanvälisessä liiketoiminnassa. Kansainvälinen aspekti tulee huomioitua myös PSK:n standardeissa, koska se käyttää viitestandardeina kansainvälisiä ja eurooppalaisia tuotestandardeja, joita tarvittaessa täydennetään vastaamaan kansallisen teollisuuden tarpeita. Suurin osa PSK:n standardeista on kirjoitettu suomeksi ja englanniksi, ja lisäksi ne on laadittu käytännön näkökulmasta siten, että ne ovat helposti otettavissa käyttöön käytännön työkaluiksi [80].

3.3.1 PSK 3605

PSK 3605 "*Prosessiteollisuuden virtaus- ja PI-kaavioiden symbolit*" [87] määrittelee nimensä mukaisesti prosessiteollisuuden virtaus- ja PI-kaavioiden symbolit. Standardi käyttää kehyksenä standardia SFS-EN ISO 10628-2 symbolien ja niiden nimeämisen määrittämiseen, ja standardeja ISO 3511-5 ja SFS-ISO 14617-6 instrumentoinnin viivatyyppien määrittämiseen. PSK-standardi täydentää standardia SFS-EN ISO 10628-2 lisäämällä itse määriteltyjä ryhmiä.

Kuvassa 3.5 on esimerkki kyseisen standardin määrittelemästä pumppuryhmän symboleista ja nimeämiskäytännöistä. Jokainen tämän standardin mukainen symboli alkaa kirjaimella A, seuraavat kaksi numeroa muodostavat ryhmänumeron ja viimeiset kaksi numeroa määrittelevät ryhmään kuuluvan symbolin numeron. Esimerkiksi A1502 määrittelee keskipakopumpulle käytettävän symbolin, joka noudattaa standardin PSK 3605 symboliikkaa (A), kuuluu pumppuryhmään (15) ja se on ryhmän sisällä toinen symboli (02).

Kyseinen standardi on relevantti suunnittelutiedonsiirrolle siinä mielessä, että esimerkiksi Proteus-tiedostolla on mahdollista siirtää PI-suunnittelutiedon sisältämää standardin mukaista grafiikkaa komponenttitietojen ja niiden välisten yhteyksien lisäksi.

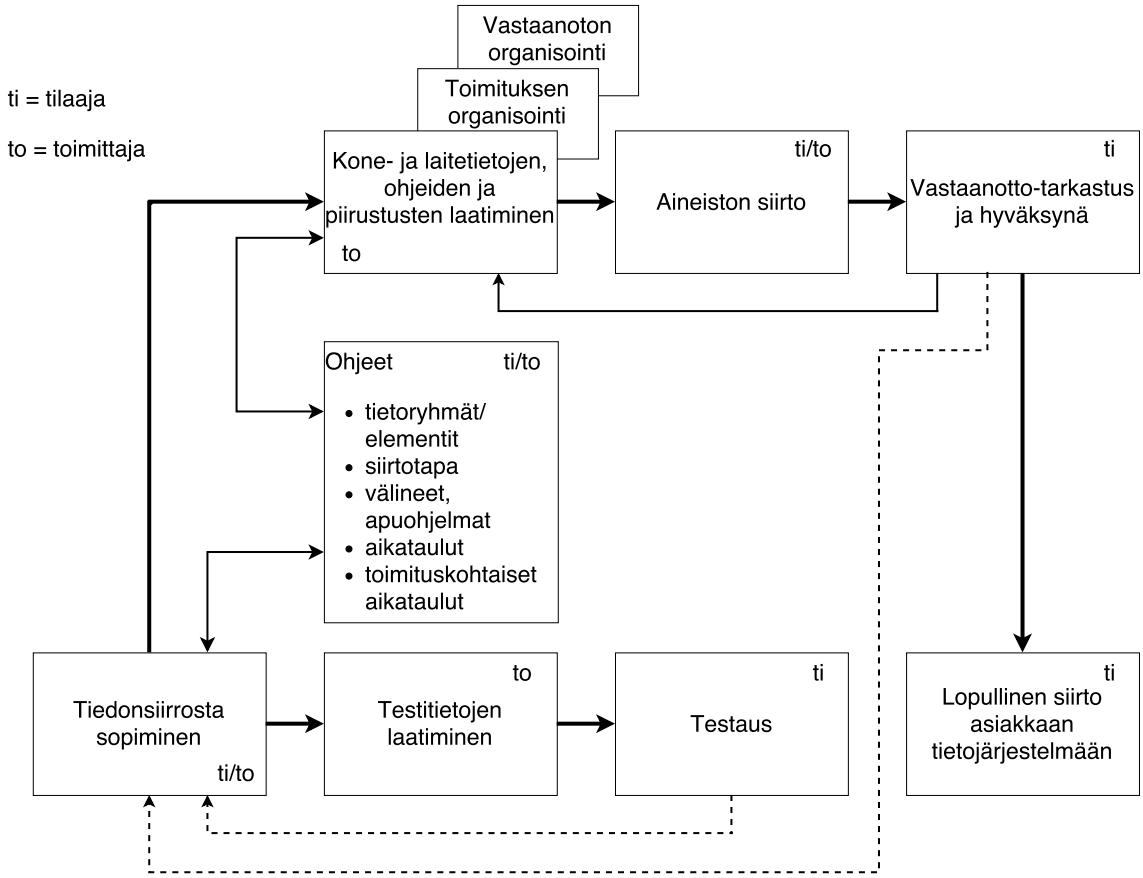
Aiheryhmä 15 Subject group 15		Pumput Liquid pumps	
 A1501 Pumppu, (yleinen) Pump, liquid type (general)	 A1502 Keskipakopumppu Pump, centrifugal type	 A1503 Hammasrataspumppu Pump, gear type	 A1504 Ruuvipumppu Pump, screw type
 A1505 Epäkeskoruuvipumppu Pump, progressive cavity type	 A1506 Mäntäpumppu Pump, reciprocating piston type	 A1507 Kalvopumppu Pump, diaphragm type	 A1508 Suihkupumppu Jet pump, liquid type ejector pump

Kuva 3.5 *Otos PSK 3605:n pumppuryhmän mukaisista piirrossymboleista ja nimeämiskäytännöstä. [87]*

3.3.2 Tiedonsiirtostandardit

Tiedonsiirtostandardit yhdenmukaistavat miten tietoa esitetään ja mikä tieto on laitosprojektin osapuolien kannalta relevanttia. Varsinkin tilaajan kannattaa sopia tiedonsiirrosta toimittajan ja suunnittelun osapuolien kanssa. Kuvassa 3.6 on esitetty PSK:n suunnitelma, millaisen prosessin kautta tilaaja ja toimittaja voivat sopia tiedonsiirrosta. Prosessi soveltuu kuvassa 3.3 esitettyjen osapuolien keskinäiseen tiedonsiirtotarpeiden sopimiseen. Seuraavassa käsitellään PSK:n tiedonsiirtostandardeja.

PSK 5965 "*Tiedonsiirto. Laitteiden luokat ja alaluokat*" [85] soveltaa standardeja ISO 15926-4 ja SFS-EN 10628 teollisuuden laitteiden luokkien ja alaluokkien määrittelemiseen ja nimeämiseen. Luokkia ovat muun muassa pumppu ja venttiili. Esimerkiksi keskipakopumppu on pumpun alaluokka ja varoventtiili on venttiilin alaluokka. Määriteltyjä luokkia hyödynnetään järjestelmistä riippumattoman tiedonsiirron rajapintamäärittelyssä. Luokat ryhmitellään teknisen aloihin standardissa PSK 5930 "*Elektronisen suunnitteluaineiston siirto. Numeeristen, aakkosnumeeristen ja aakkosellisten tietoelementtien esitysmuodot. Peruskäsitteet ja esitystavat.*" [82] esitetyn nimeämiskäytännön mukaisesti.

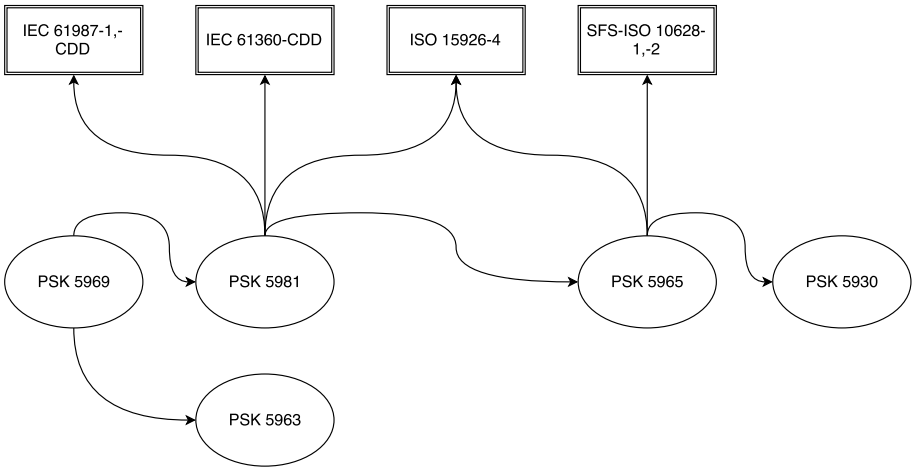


Kuva 3.6 *PSK tiedonsiirtosuunnitelma. [83]*

PSK 5981 "*Tiedonsiirto. Tietoelementtiluettelo*" määrittää PSK 5965:ssa määritellyille teollisuuden laiteluokille tietoelementit, jotka on soveltuvin osin yhtenäistetty standardien ISO 15926-4 ja IEC 61987-CDD (Common Data Dictionary) kanssa. Tietojen periytyminen luokille ja luokkakohtaisten tietojen periytyminen edelleen alaluokille noudattaa olio-ohjelmoinnin paradigmaa. Tässä standardissa esitellyt tietoelementit ovat teollisuuden instrumentoinnin hankinnassa oleellisia ja ne on otettava huomioon edellä esitellyn tiedonsiirtosuunnitelman laadintaprosessissa.

PSK 5969 "*Tiedonsiirto. Järjestelmien attribuuttien vastaavuus. Automaation kenttälaitteiden tiedot*" [86] poimii PSK 5981:ssä määritellyistä tietoelementeistä automaation kent-tälaitteiden kannalta oleelliset tietoalkiot. Standardi listaa myös eri suunnitteluohjelmistojen vastaavat suunnitteluattribuutit.

Kuvasta 3.7 on nähtävissä edellä esiteltyjen standardien väliset yhteydet. Kuvassa nuoli kuvaa mihin standardiin viitataan. Kuvasta nähdään, miten PSK-standardit tarkentuvat edettäessä oikealta vasemmalle.

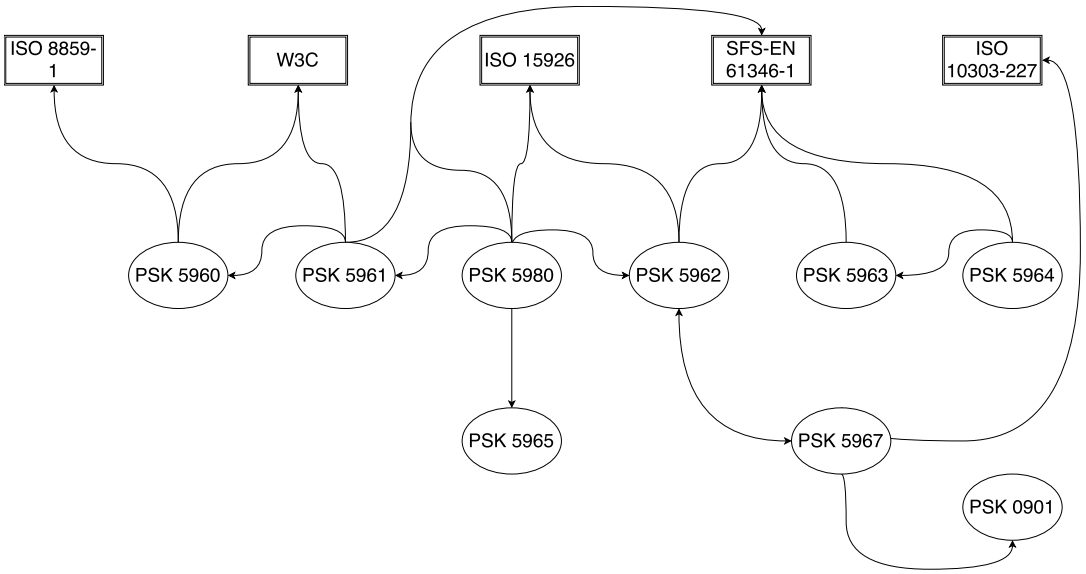


Kuva 3.7 *PSK tiedonsiirtostandardien relaatiot.*

3.3.3 XML-tiedonsiirtostandardit

PSK:n XML-tiedonsiirtostandardit määrittelevät XML-skeeman, joka on saman kaltainen kuin aikaisemmin esitellyillä tiedonsiirtostandardeilla. Kuvassa 3.8 on esitetty XML-standardisarjan standardinen keskinäiset yhteydet, sekä yhteydet kansainvälisiin standardeihin. Kuvassa nuoli kuvaa mihin standardiin viitataan.

Laitteiden määrittely on esitetty standardeissa PSK 5963 (automaatio), PSK 5964 (sähköjärjestelmät) ja PSK 5965 (laitteet). Putkistorakenne kuvataan standardissa PSK 5967, joka periyttää hierarkkiaa kuvaavat luokat standardista PSK 5962. Kantaluokkien määrittämisen lisäksi PSK 5962 määrittää tiedonsiirtomallin.



Kuva 3.8 *PSK XML-tiedonsiirtostandardien keskinäiset suhteet.*

3.4 Vertailu

Laitosprojektin PI-suunnittelutiedon siirtämiseen on tarjolla monia standardisoituja ratkaisuja. Edellä esitellyistä ratkaisuista jokaisella on mahdollista mallintaa ja/tai siirtää PI-suunnittelutietoa. Standardit IEC 62424 ja ISO 15926 määrittävät PI-suunnittelutiedon mallintamiseen soveltuvat metamallit, mutta ne ovat liian geneerisiä PI-suunnittelutiedon konsistenttisuuden kannalta. Tämä tarkoittaa muun muassa sitä, että standardien pohjalta tehdyt tietomallitoteutukset voivat poiketa paljonkin toisistaan. Tähän vertailuun on otettu mukaan standardien IEC 62424 ja ISO 15926 pohjalta toteutetut ratkaisut ja niiden ominaisuuksien vertailu on esitetty taulukossa 3.1.

Taulukko 3.1 *Tiedonsiirtomallien ominaisuuksien erittely kriteereittäin.*

Kriteeri	**DEXPI**	**PSK**	**PandIX**	**AutomationML**
Mitä tietoa siirretään	PI 2D/(3D)	PI Sähkö Automaatio	PI Automaatio	PI 3D Automaatio Kinematiikka
Liittyvät standardit	ISO 15926	Katso kuva 3.8	IEC 62424	IEC 62424 ISO/PAS 17506 IEC 61131-3
Tiedonsiirtotapa	XML(Proteus)	XML(PSK 5962)	XML(CAEX)	XML(AML)
Spesifikaatioehdotus	DEXPI-spesifikaatio	PSK standardi	-	IEC 62714
Ohjelmistokehittäjät	Autodesk AVEVA Bentely Systems Intergraph Siemens X-Visual	-	Implementoitavissa AML:ssä	ABB Daimler Siemens

PandIX ei sinällään sovellu PI-suunnittelutiedon siirtoon, sillä se rajoittuu tarkastelemaan laitosmallia automaation näkökulmasta. Sen tavoitteena on helpottaa tiedonvaihtoa prosessi- ja automaatiosuunnittelun välillä siten, että suunnittelutiedon käsin kopioimisen sijaan PI-kaaviosta saatava rakenne- ja funktionaalisuustieto käsitellään ja siirretään automaattisesti eri ohjelmistotoimittajien työkalujen välillä käyttäen CAEX-pohjaista formaattia. [109] PandIX:ia on mahdollista hyödyntää CAEX-pohjaisissa malleissa, mutta sinällään sitä ei ole toteutettuna ohjelmistotoimittajien järjestelmissä. Toisaalta PandIX:n funktionaalisuus on hyödynnettävissä esimerkiksi AutomationML:ssa.

PSK on määritellyt oman XML-pohjaisen tiedonsiirtomallinsa standardissaan PSK 5962. PSK käyttää kaiken kattavasti erilaisia standardeja järjestelmäriippumattoman

XML-rajapinnan määrittelyyn, kuten kuvista 3.7 ja 3.8 voidaan todeta. PSK käyttää rajapintamäärittelyssään niin kansainvälisiä standardeja, kuin itsensä laatimia kansallisia standardeja, jotka perustuvat kansainvälisiin standardeihin. Tietomallin määrittely perustuu standardin ISO 15926 kantaluokka lähestymistapaan [84]. Tietomallin avulla on siirrettävissä PI-, sähkö- ja automaatiosuunnittelutietoa. Toistaiseksi ei ole tiedossa, että PSK:n mallia olisi otettu käyttöön ohjelmistotoimittajien järjestelmissä, mutta lähestymistapa on vankasti standardisoituihin menetelmiin pohjautuva.

AutomationML on monen standardoidun XML-pohjaisen menetelmän integraattori. Sen sijaan, että se yrittäisi kehittää jotakin täysin uutta, se integroi yleisesti käytössä olevia menetelmiä yhdeksi kokonaisuudeksi käyttäen runkona standardia IEC 62424. AutomationML:lla on mahdollista siirtää PI-suunnittelutietoa, sillä se pohjautuu CAEX:een, jolla on mahdollista mallintaa erilaisia hierarkkisia järjestelmiä. SISO/PAS 17506 ja IEX 61131-3 standardituen ansiosta AutomationML tukee myös 3D-, kinematiikka- ja automaatiosuunnittelutiedon siirtoa [6]. AutomationML e.V.:ltä on saatavana joitakin pikkunäppäriä työkaluja [8] CAEX- ja AutomationML-tiedostojen käsittelyyn, mutta vielä harva suunnitteluohjelmisto tukee AutomationML-formaattia [1]. Saatavilla olevat sovellusesimerkit ja työkalut ovat robotiikka ja kappaletavara-automaatio-orientoituneita. Esimerkki hitsaus- ja liukuhihnaprosessin mallintamisesta AutomationML:lla on esitetty lähteessä [57]. AutomationML on standardisoitu kolmessa osassa standardissa IEC 62714 "*Engineering data exchange format for use in industrial automation systems engineering - Automation markup language*". Toisin kuin ISO 15926, AutomationML ei määrittele muun muassa laiteluokkia ja laitteita [1]. Sen sijaan, että ohjelmistotoimittaja voisi suoraan periyttää ja viitata standardoituihin laiteluokkiin, esimerkiksi RDS-palvelun kautta, on AutomationML:n tapauksessa ne määritettävä itse. Lisäksi ei ole standardisoitu menetelmää, jolla toimitusprojektin osapuolet voisivat sovittaa eri luokat keskenään yhteen [1].

DEXPI-työryhmän standardin ISO 15926 pohjalta laatima tiedonsiirtomalli on varteenotettava vaihtoehto laitossuunnittelutiedon tiedonsiirtoon. Malli antaa mahdollisuudet siirtää PI-kaaviotietoa metatietoineen, eli suunnitelmasta ei ainoastaan mallinneta hierarkkista rakennetta, vaan suunnitelman elementtikohtaiset tiedot siirretään myös (ks. ohjelma 3.1). Koska ISO 15926 pohjautuu standardiin ISO 10303, on Proteus-tiedostolla mahdollista siirtää myös suunnitelman graafista tietoa standardoidusti. Uusin Proteus-skeemaversio 4.0.1 ei tue 3D-objektien siirtoa toisin kuin edeltävä versio 3.6.0. Proteus-skeemat ovat taaksepäin yhteensopivia versioon 3.6.0 asti, sillä versiossa 4.0.1 instrumentointimalli on muutettu. Instrumentointimalli kuvaa minkä mittauksen ja säätöfunktion avulla prosessitoimilaitetta ohjataan, jot-

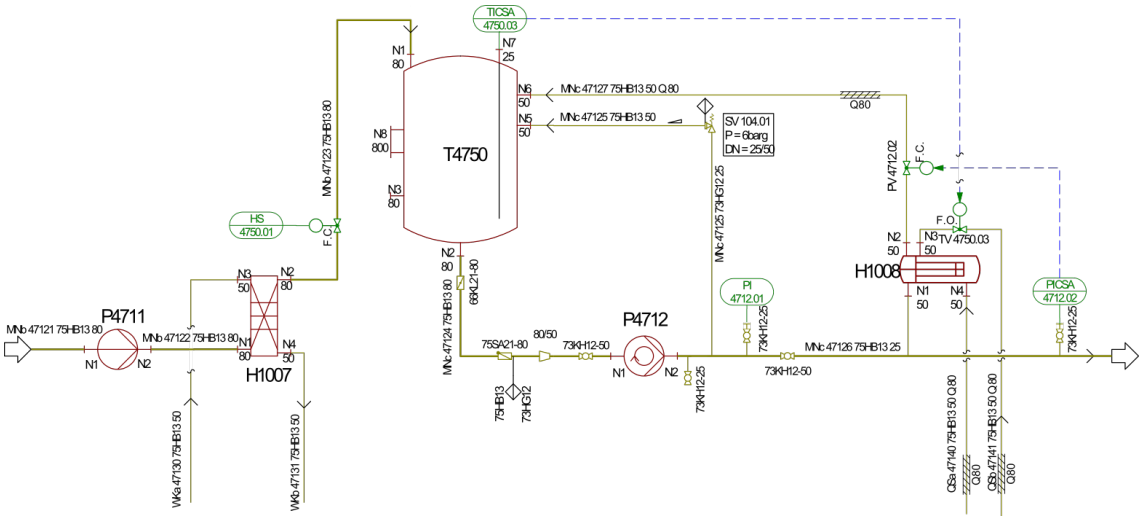
ta mitattuun prosessisuureeseen voidaan vaikuttaa. Toisaalta ISO 15926 kuvaamaa tietomallia on moitittu monimutkaiseksi [12, 40, 58], mutta DEXPI-spesifikaatio yrittää parhaansa mukaan muodostaa selkeän skeemakuvausen tuon mallin pohjalta. Tästä huolimatta spesifikaatio antaa kohtuullisen vapaat kädet rajapintatoteutukselle ja tämän vuoksi ohjelmistotoimittajien ratkaisut poikkeavat toisistaan. Tämä on nähtävissä vertailemalla suunnitteluohjelmistoista tuotuja Proteus-tiedostoja.

PandIX:n ja PSK:n tiedonsiirtomallit eivät tue PI-suunnittelutiedon graafista mallintamista ja siirtoa. PandIX:n tietomallia on mahdollista käyttää AutomationML:n avustuksella. PSK:n lähestymistapa perustuu vankkoihin kansainvälisiin standardeihin ja näiden pohjalta valmisteltuihin kansallisiin standardeihin. AutomationML on esitellyistä tiedonsiirtomalleista kattavin, mutta PI-suunnittelutiedon kannalta esimerkiksi kinematiikkatietomalli ei ole PI-suunnittelutiedon kannalta oleellista. PandIX:n, PSK:n ja AutomationML:n tiedonsiirtomallit kattavat tämän työn kannalta ylimääräistä tietoa ja lisäksi niiden ohjelmistotuki on puutteellinen. Esitellyistä tiedonsiirtomalleista Proteus-skeemaan perustuva DEXPI-spesifikaatio käy työn rajaukseen ja sillä on tiedettävästi usean ohjelmistotoimittajan [19] ja omistaja-operaattorin tuki taustalla. Muun muassa näistä lähtökodista voidaan olettaa, että omistaja-operaattorien mielenkiinto DEXPI-spesifikaatiota kohtaan ajaa tiedonsiirtomallin kehitystä eteenpäin ja motivoi ohjelmistotoimittajia enemmissä määrin kehittämään suunnitteluohjelmistojensa Proteus/DEXPI-tukea.

DEXPI:n ja OPC UA:n välille on kehitteillä liittostandardi DEXPI-työryhmän ja OPC Foundationin yhteistyönä. Liittostandardin tavoitteena on määrittää, miten DEXPI-tietomalli esitetään OPC UA -tietomallina. Vastaavanlainen liittostandardi on laadittu myös AutomationML:n ja OPC UA:n kesken [9]. Tässä työssä tehdään soveltuvuusselvitys (proof of concept) Proteus/DEXPI-mallin muuntaminen OPC UA -tietomallin mukaiseen XML-tiedostoon Simantics-alustaa hyödyntäen. OPC UA, mallien muunnos ja Simantics-alusta esitellään luvussa 5.

4. PI-SUUNNITTELUOHJELMISTOJEN PROTEUS/DEXPI-TUKI

Tässä luvussa esitetään katsaus suunnitteluohjelmistojen valmiuteen tukea Proteus/DEXPI-mallia. Tämä malli vastaa Proteus-skeemaa, joka noudattaa DEXPI-spesifikaation määrittämää tietomallia. DEXPI-työryhmän asettama tavoite PI-kaavio tiedonvaihtoa varten on kuvassa 4.1. Suunnitteluohjelmistokehittäjät kokoontuvat määräajoin saman pöydän ääreen hackathon-tapahtumissa[1], joissa kehittäjät testaavat ohjelmistojensa tuonti- ja vientiominaisuuksia testitapauksin. Testitapaukset ja työryhmän ohjelmistokehittäjien aktiivisuus on esitetty taulukossa 4.1. Edistyminen kahden edellisen hackathonin välillä on osoitettu taulukossa sinisillä soluilla. Taulukon ensimmäisessä sarakkeessa on testitapauksen nimi ja toisessa sarakkeessa suunnitteluohjelmiston tukemat tapaukset. Testitapauksien sisältö ja testausprosessi on määritetty avoimessa lähteessä [21].



Kuva 4.1 *DEXPI-työryhmän suunnittelutiedonsiirrolle asettama tavoite PI-kaavio. [20]*

Taulukon 4.1 sisältö ei kerro miten kypsiä suunnitteluohjelmistojen toteutukset ovat, ennemmin mitä toteutuksen osa-alueita on aloitettu kehittämään. Kaikki tiedostot

[1]Hackathon (a.k.a hack day/hackfest/sprint/codefest/...) on ICT-aiheinen työpaja, missä joukko ihmisiä yhteistuumin kehittää ratkaisuja annettuihin ongelmiin.

Taulukko 4.1 *DEXPI-työryhmän testitapaukset ja suunnitteluohjelmistojen Proteus/DEXPI-tuen tilanne. Sinisellä värjätyt solut ovat lisäyksiä kahden hackathon tapahtuman välillä.*

Testitapaus	**A**	**B**	**C**	**D**
C01 Pump, Tank, and Control loop	-	-	-	-
C02 DEXPI Example Complete E+P	-	✓	✓	-
C03 DEXPI Example Tank Displ Pump Pipe with Tee	-	✓	✓	-
C04 DEXPI Example Pump Plate Heat Exchanger	-	✓	✓	-
E01 Tank T100	✓	✓	✓	✓
E02 Tank with Nozzles	✓	✓	✓	✓
E03 Pump With Nozzles	✓	✓	-	-
E04 HeatExchanger With Nozzles	✓	✓	-	-
E05 Pump With Nozzles And HeatExchanger With Nozzles	✓	✓	-	-
E06 Pump, HeatExchanger, Nozzles Connected With PNS	✓	✓	-	-
E07 PressureVessel with ColumnSections	-	-	-	-
E08 ProcessColumn with ColumnSections	-	-	-	-
I01 Measurement	-	✓	✓	✓
I02 Control	-	✓	✓	✓
I03 Measurement and Control	-	✓	-	✓
I04 Measurement and Control	-	-	✓	✓
I05 Two flow indications and flow ratio control in CCR	-	-	-	✓
I06 CCR flow indication and high alarm, flow control, control valve	-	-	-	✓
I07 Local and CCR pressure indic., alarm and safety switch	-	-	-	✓
I08 Local pressure indication, CCR pressure indication, alarms and switches	-	-	-	✓
I09 Measurement	-	-	-	✓
I10 Measurement	-	-	-	✓
I11 Measurement	-	-	✓	✓
I12 Control	-	-	✓	✓
P01 Pipe FromTo Nozzles	✓	✓	✓	✓
P02 Pipe From OPC to Nozzle	-	✓	-	✓
P03 Pipe With Edge	✓	✓	-	-
P04 Pipe With Intersection	✓	✓	-	-

on validoitu ja verifioitu Proteus-skeemaa (luonnosversio 4.0.1) vasten. DEXPI:lla on käytössään työkalu suunnitteluohjelmistoista tuotettujen tiedostojen semantiikan verifioimiseksi DEXPI-spesifikaatiota vasten. Tiedoston semantiikan oikeellisuudesta ei voi tehdä päätelmiä rakenteellisesta osuvuudesta ja vice versa. Esimerkki semanttisesta oikeellisuudesta Proteus/DEXPI:n yhteydessä voisi olla mallin mukaisten attribuuttien, kuten *ComponentClass*:in ja *ComponentClassURI*:in, käyttö ja yhteensopivuus. Edellä mainittu yhteensopivuus Proteus/DEXPI-mallin kanssa tarkoittaa sitä, että *ComponentClass*:lle asetetulle arvolle on löydyttävä vastaa-

vuus PCA:n tai DEXPI:n RDL-kirjastosta. Joillekin testitapauksille on luotu erillinen XML-muotoinen verifiointitiedosto, joka kertoo mitä tietoa ja millaisella rakenteella DEXPI odottaa suunnitteluohjelmistoista tuotujen tiedostojen noudattavan. Tiedostoihin on sisällytetty myös metadataa. Nämä verifiointitiedostot, kuten ei myöskään työryhmän verifiointityökalu, eivät ota kantaa miten kaavioiden grafiikka on XML-dokumentissa esitettävä.

Koska käytettävissä on eräänlaisia verifiointitiedostoja, joiden mukaista rakennetta suunnitteluohjelmistoista tuotujen tiedostojen odotetaan noudattavan, on mahdollista tarkastella miten paljon nykyiset toteutukset poikkeavat niin sanotusta referenssistä. XML-dokumenttien keskinäisten rakenteellisten ja semanttisten ominaisuuksien vertailemista varten on kehitetty menetelmiä ja mittareita samankaltaisuuden määrittelemiseksi. XML-dokumenttien rakenteen samankaltaisuutta on tutkittu lähteissä [56, 28, 65, 88]. Rakenteellisen ja semanttisen samankaltaisuuden yhdistävä menetelmä on esitetty lähteessä [108]. Guerrini et al. [31] esittävät joukon erilaisia samankaltaisuusmittareita XML-dokumenttien ryhmittelyn (clustering) näkökulmasta. Samankaltaisuutta voidaan arvioida myös ryhmittelyn avulla, tarkastelemalla kuinka kaukana, tai lähellä, dokumentti on jotakin dokumentti ryhmittymää. Joukko erilaisia dokumentteja voidaan jakaa eri ryhmiin esimerkiksi XML-skeemojen avulla tai käyttämällä automaattisia menetelmiä dokumenttien rakenteiden tunnistamiseen. Erilaisia ryhmittelymenetelmiä on esitetty lähteissä [15, 17, 22, 51, 55, 77, 122]

Dokumenttien poikkeavuutta/päällekkäisyyttä/samankaltaisuutta referenssitiedostosta voidaan mitata Simpsonin[2] samankaltaisuuskertoimella [96] (Simpson similarity coefficient), joka on sukua Jaccardin [45] kertoimelle. Simpsonin samankaltaisuuskerroin voidaan joukko-opillisesti esittää muodossa

$$S(A,B) = \begin{cases} \frac{|A \cap B|}{|A \cap B| + |A - B|}, & \text{jos } |A \cap B| + |A - B| < |A \cap B| + |B - A| \\ \frac{|A \cap B|}{|A \cap B| + |B - A|} & \text{muuten} \end{cases} \tag{4.1}$$

tai muodossa

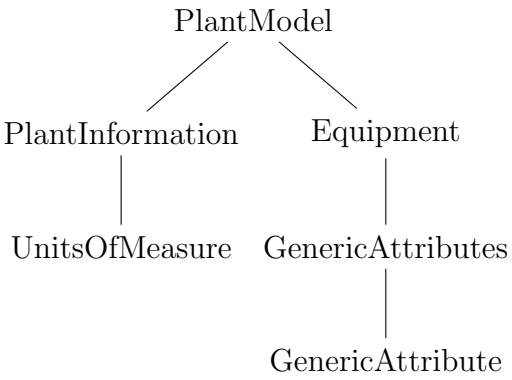
$$S(A,B) = \frac{|A \cap B|}{min(|A|, |B|)}, \tag{4.2}$$

[2]Simpsonin samankaltaisuuskerroin tunnetaan myös nimillä Szymkiewicz-Simpson -kerroin ja päällekkäisyyskerroin (overlap coefficient).

joissa A ja B ovat joukkoja. Yhtälöiden osoittajissa on joukkojen A ja B leikkauksien mahtavuus[3], ja nimittäjässä näistä kahdesta joukoista pienemmän joukon mahtavuus. Joukkojen mahtavuudet ovat kokonaislukuja ja kerroin on rationaaliluku. Saatu kertoimen arvo on sitä lähempänä arvoa yksi, mitä enemmän yhteisiä alkioita näillä kahdella joukolla on. Simpsonin samankaltaisuuskerrointa voidaan hyödyntää dokumenttien rakenteiden vertailussa, kun dokumenteista erotetaan joukko rakennetta kuvaavia piirteitä. Yksistään pelkkien XML-elementtien vertaileminen ei kerro rakenteesta mitään. Koska XML-dokumentit ovat rakenteiltaan puuhierarkkisia, voidaan dokumenttien juuripoluista [88] muodostaa piirrevektoreita [122]. Juuripolut ovat reittejä dokumentin juurielementistä jokaiseen lehteen. Esimerkiksi ohjelman 3.1 juurielementti on *PlantModel* ja eräs lehti *UnitsOfMeasure*, ja juuripolku muodostuu näiden kahden elementin välisestä reitistä

$$\{PlantModel/PlantInformation/UnitsOfMeasure\}.$$

Kuvassa 4.2 on esitetty ohjelman 3.1 XML-dokumentti puurakenteisena diagrammina.



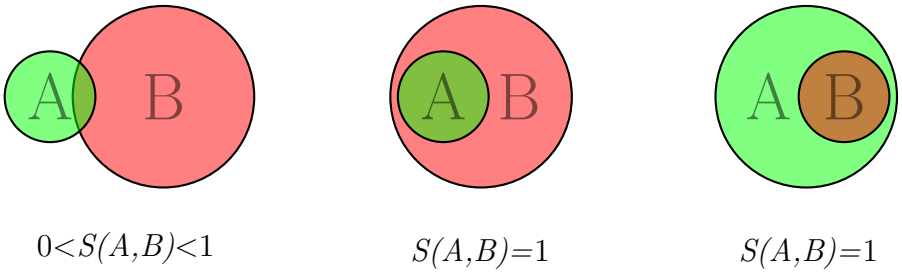
Kuva 4.2 *Ohjelman 3.1 XML-dokumenttia vastaava puurakenne.*

Kuten kuvasta havaitaan, dokumentilla on myös toinen juuripolku. Yhdessä edellisen polun kanssa muodostuu dokumentin juuripoluista koostuva piirevektorien joukko

$$\{PlantModel/Equipment/GenericAttributes/GenericAttribute,$$
$$PlantModel/PlantInformation/UnitsOfMeasure\}.$$

Lähtökohtaisesti verifiointitiedostot ovat kooltaan pienempiä, eli niissä on vähemmän

[3]Joukon mahtavuus, eli alkioiden lukumäärä joukossa.



Kuva 4.3 *Venn-diagrammiesimerkki Simpsonin samankaltaisuukertoimesta.*

juuripolkuja, ja siten myös niistä muodostuva joukko on pienempi kuin suunnitteluohjelmistoista tuodut tiedostot. Tämä oletus juontuu siitä, että suunnitteluohjelmistoista tuodut tiedostot sisältävät muun muassa suunnitelmien graafista tietoa. Kaavan (4.2) käyttö sellaisenaan dokumenttien samankaltaisuuden arvioinnissa johtaa harhaan tapauksissa, joissa verrattavan tiedoston juuripolkujen joukko on pienempi. Mielenkiintoista on tässä yhteydessä tietää, miten suuri osa verifiointitiedoston rakenteesta löytyy suunnitteluohjelmistosta tuodusta tiedostosta eikä päinvastoin. Tätä tilannetta on havainnollistettu kuvassa 4.3, jossa on kolme Venn-diagrammia. Jokaisessa tapauksessa vihreällä merkitty joukko A esittää verifiointitiedoston juuripolkujen joukkoa ja tarkasteltavan tiedoston juuripolkujen joukko B on esitetty punaisella. Ensimmäisessä tapauksessa joukko A ja B leikkaavat toisensa, eli tarkasteltavasta rakenteesta löytyy joitakin samankaltaisia rakenteellisia ominaisuuksia kuin verifiointitiedostossa. Keskimmäisessä tapauksessa joukko A on aito osajoukko joukosta B, koska A on osajoukko B:stä ja A ei ole sama kuin B. Tämä tarkoittaa sitä, että tarkasteltavasta tiedostosta löytyy verifiointitiedostoa vastaava rakenne. Viimeisessä tapauksessa, jossa joukko A on suurempi kuin joukko B, Simpsonin samankaltaisuuskertoimen arvo on sama kuin keskimmäisessä tapauksessa. Joukko-opillisesti tulkinta on oikea, mutta yhteys jossa sitä on tarkoitus käyttää, antaa kerroin väärää tietoa. Kertoimen pitäisi kertoa kohdetiedoston vastaavuudesta verifiointitiedostoon, mutta tässä valossa kerroin osoittaa rakenteiden olevan yhtäläiset, vaikka diagrammin pohjalta tulkinnan pitäisi olla samankaltainen kuin ensimmäisessä tapauksessa, eli kohdetiedoston rakenne ei täysin vastaa verifiointitiedoston rakennetta.

Simpsonin samankaltaisuuskertoimen kaavaa hieman muuttamalla saadaan se soveltumaan kaikkiin edellä esitettyihin tapauksiin muokkaamalla se muotoon

$$S(A, B) = \frac{|A \cap B|}{|A|}, \tag{4.3}$$

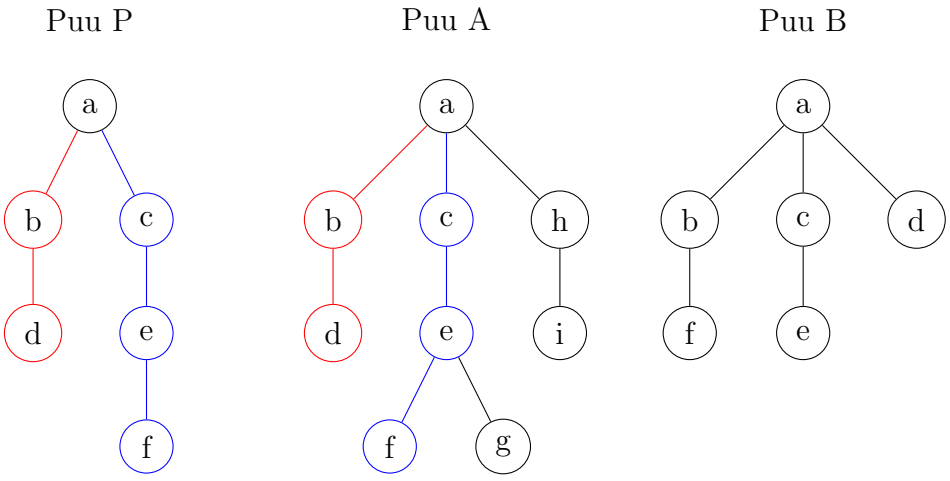
jossa A on verifiointitiedoston juuripolkujen joukko ja B on tarkasteltavan tiedoston

juuripolkujen joukko. Joukkojen leikkausjoukon mahtavuus jaetaan joukon A mahtavuudella. Näin muodostettu kerroin ilmaisee joukkojen samankaltaisuutta verifiointitiedoston näkökulmasta. Kuvan 4.3 viimeisessä tapauksessa, kaavan uudelleen määrittelemisen jälkeen, voi samankaltaisuuskerroin saada arvoja $0<S(A,B)<1$.

Kuvassa 4.4 on esitetty esimerkki juuripoluista puuhierarkkisessa rakenteessa. Kuten edellä todettiin, juuripolut ovat reittejä puun juurielementistä lehtielementteihin, ja näitä juuripolkuja käytetään puuta kuvaavina piirrevektoreina. Puun P juurielementti on a ja sillä on kaksi lehtielementtiä d ja f. Puun P vastaavat juuripolut löytyvät puusta A, jolla on lisäksi kaksi muuta juuri polkua. Puulla B ei ole yhteneviä juuripolkuja puiden P ja B kanssa. Puiden piirrevektorit on esitetty taulukossa 4.2, jossa

$$\begin{aligned}
\text{p1} &= \{a,b,c\},\\
\text{p2} &= \{a,c,e,f\},\\
\text{p3} &= \{a,c,e,g\},\\
\text{p4} &= \{a,h,i\},\\
\text{p5} &= \{a,b,f\},\\
\text{p6} &= \{a,c,e\},\\
\text{p7} &= \{a,d\}.
\end{aligned}$$

Taulukosta on helposti nähtävissä puiden A ($(P,A)=1$) ja B ($(P,B)=0$) samankaltaisuus verrattuna puuhun P.



Kuva 4.4 *Juuripolkuesimerkki puuhierarkisessa rakenteessa.*

Taulukko 4.2 *Juuripolut kuvasta 4.4*

Puu \ Polku	p1	p2	p3	p4	p5	p6	p7
P	1	1	0	0	0	0	0
A	1	1	1	1	0	0	0
B	0	0	0	0	1	1	1

Ohjelmassa B.1 on Python-koodilla toteutettu ohjelma juuripolkujen erottamiseksi syötetiedostoista. Ohjelma hakee ensimmäiseksi kaikki XML-dokumentin lehtielementit ja selvittää rekursiivisesti polun lehdestä juureen. Kuvan 4.4 puun *A* tapauksessa ensiksi haetaan elementit *d, f, g* ja *i*, ja seuraavaksi jokaisesta lehdestä erikseen reitti oksia pitkin juureen. Proteus-skeeman mukaan <*GenericAttributes*>-elementeillä voi olla useampia <*GenericAttribute*>-elementtejä, ja <*GenericAttributes*>-elementtejä voi esiintyä samalla hierarkiatasolla useampia. DEXPI-spesifikaation mukaiset attribuuttijoukot esitetään asettamalla elementin <*GenericAttributes*> attribuutin *Set* arvoksi *DexpiAttributes*. Dokumentin rakennetta tarkasteltaessa ei juuripolkuihin ole tarvetta sisällyttää tietoa jokaisesta semantiikkaa kuvaavasta <*GenericAttribute*>-elementistä. Esimerkiksi kuvan 4.4 puussa *A* elementtiä *e* voisi vastata elementti <*GenericAttributes*>, ja solmut *f* ja *g* <*GenericAttribute*>-elementtejä. Näistä kahdesta muodostuvasta juuripolusta (*p2* ja *p3*) piirrevektorien joukkoon valitaan ensimmäisenä vastaan tiedostoa läpikäytäessä tuleva juuripolku. Lehtien lukumäärä elementtien <*GenericAttribute*> tapauksessa ei ole mielenkiintoinen tutkittava seikka, koska ne kuvaavat ennemmin dokumentin metatietoa kuin rakennetta. Suunnitteluohjelmistotoimittajat voivat määritellä Proteus-skeeman ja DEXPI-tietomallin mukaisesti eriäviä määriä <*GenericAttribute*>-elementtejä.

Taulukossa 4.3 on esitetty testitapauksien samankaltaisuuskertoimet verifiointitiedostoa vasten prosentteina vertailun helpottamiseksi. Kaikille testitapauksille ei ole vielä työn kirjoitusvaiheessa verifiointitiedostoa saatavilla.

Taulukko 4.3 *Suunnitteluohjelmistoista tuotujen tiedostojen rakenteen samankaltaisuus verifiointitiedostoon nähden prosentteina.*

Testitapaus	**A**(%)	**B**(%)	**C**(%)	**D**(%)	**VER**
E01 Tank T100	50	50	100	50	4
E02 Tank with Nozzles	37	21	74	47	19
E03 Pump With Nozzles	50	33	-	-	6
I01 Measurement	-	55	65	15	20

VER - Piirrevektorien lukumäärä verifiointitiedostossa

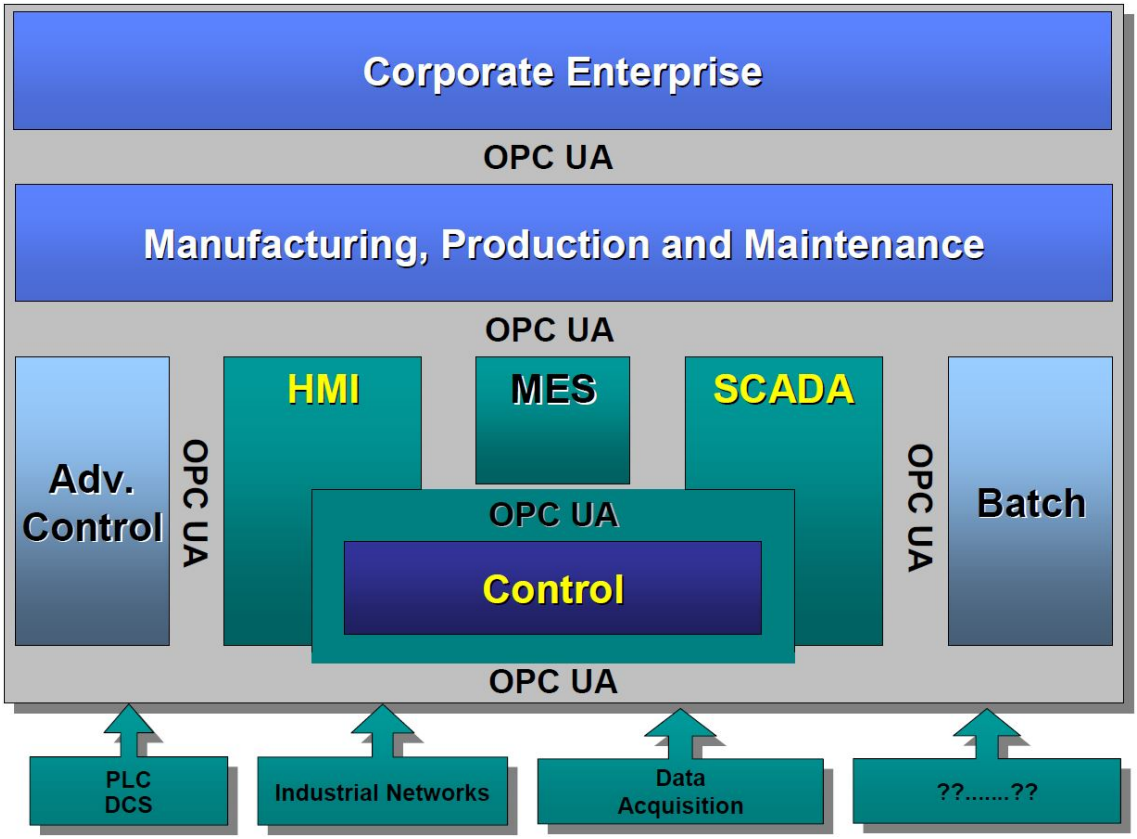
5. DEXPI/PROTEUS-TIETOMALLIN MUUNTAMINEN OPC UA -TIETOMALLIKSI

Tässä luvussa tutustutaan OPC Foundationin (OPC) [72] laatimaan OPC Unified Architecture (OPC UA) palvelukeskeiseen arkkitehtuuriin (service oriented architecture, SOA) tietomallintamisen näkökulmasta. Lisäksi määritetään miten Proteus/-DEXPI-malli kuvataan OPC UA:n tietomalliksi. Lopuksi esitellään mallimuunnostyökalu Proteus/DEXPI-OPC UA -muunnoksen automatisoimiseksi.

5.1 OPC Unified Architecture

OPC Unified Architecture on OPC:n laatima viestintäprotokolla automaatioteollisuuden laitteistojen ja ohjelmistojen väliseen tiedonsiirtoon; kenttälaitteista toiminnanohjausjärjestelmiin. OPC UA:n tavoitteena on helpottaa ja yhdenmukaistaa kuvassa 5.1 esitettyjen yrityksen funktioiden välistä tiedonsiirtoa. Päästäkseen tähän tavoitteeseen OPC on määrittänyt yleisen infrastruktuurimallin, joka rakentuu tieto-, sanoma-, viestintä ja yhdenmukaisuusosamallista. [73] OPC UA on standardisoitu 13 osassa standardissa IEC 62541 "*OPC Unified Architecture*".

OPC UA on alustariippumaton standardi, minkä avulla monenlaiset järjestelmät ja laitteet voivat kommunikoida keskenään erilaisissa verkoissa asiakaskoneen (client) ja palvelimen (server) kesken [73]. OPC UA:n aikaisempi toteutus OPC Classic toimii ainoastaan Microsoft-alustalla hyödyntäen viestintään sen COM/DCOM-rajapintaa. OPC UA on takaisin yhteensopiva OPC Classic:in kanssa. Aikaisemmin OPC määritteli omat erilliset mallit tiedonkäsittelylle (Data Access, DA), hälytyksille ja tapahtumille (Alarm & Events, AE) ja historiatiedolle (Historical Data Access, HDA). Esimerkiksi DA:sta luetulla arvolla ei ole yhtymäkohtaa HDA:n historiatietoihin. OPC UA muun muassa yhdenmukaistaa nämä osamallit yhdeksi malliksi samaan osoiteavaruuteen (address space). Nyt DA, HDA ja AE on integroitu samaan malliin ja niihin pääsee käsiksi palvelimen tarjoamien palvelujen kautta. [59, 73] Eri siirtymistapoja OPC Classic:sta OPC UA:han on esitetty lähteissä [33, 54, 59].

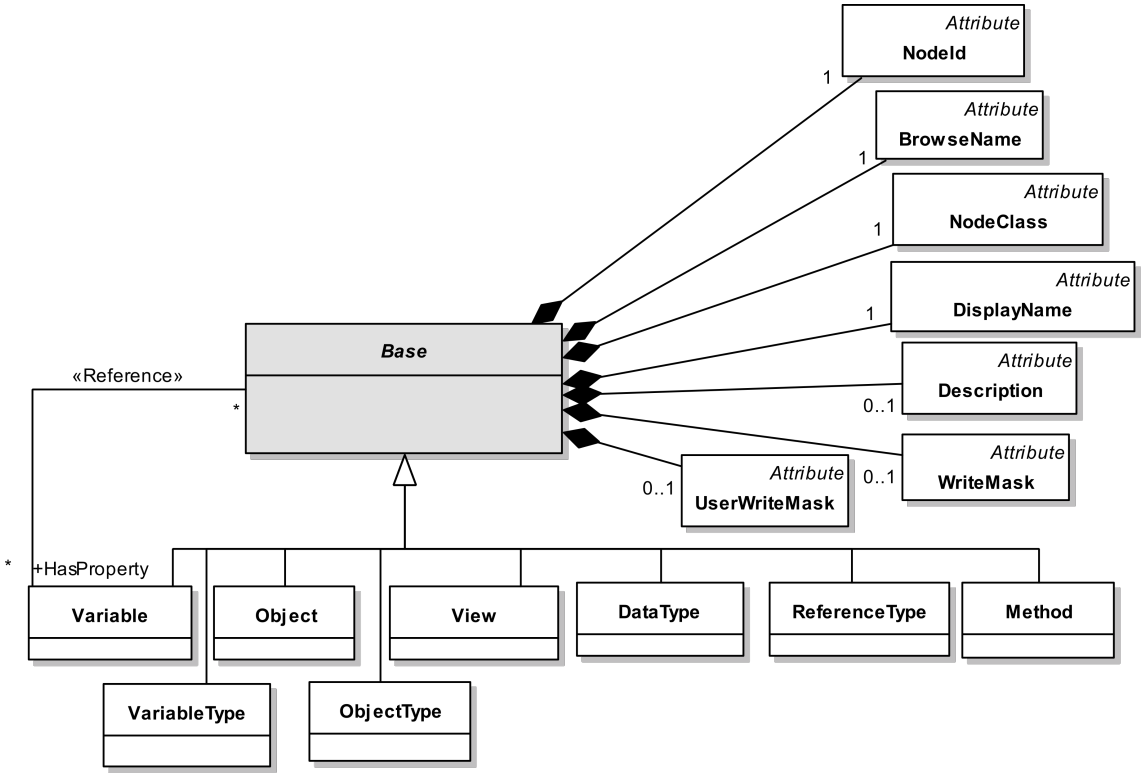


Kuva 5.1 *OPC UA:n soveltuvuus yrityksien eri funktioihin.[73]*

5.2 OPC UA -tietomalli ja AddressSpace-mallinnus

OPC UA:aan metamalli on määritetty standardissa IEC 62541-3 [75]. Mallia kutsutaan myös nimellä osoiteavaruusmalli (address space model, AddressSpace). Kuvassa 5.2 esitetyllä OPC UA:n metamallilla on mahdollista mallintaa muun muassa laitteita, toiminnallisuutta ja järjestelmätietoa. Lisäksi mallinnettujen objektien välillä voi olla erilaisia yhteyksiä viittauksien (Reference) avulla. Osoiteavaruusmalli määrittää rakennuspalikat instanssien ja tyyppitiedon esittämiselle. Metamallia käytetään tiedon kuvaamiseen ja kuinka OPC UA -palvelin voi tuoda mallinnetun tiedon esille osoiteavaruutensa kautta. [59]

Perusluokkalla (Base) on kompositio-relaation kautta seitsemän eri attribuuttia, joista *NodeId, BrowseName, NodeClass* ja *DisplayName* ovat pakollisia. Attribuuttien *Description, WriteMask* ja *UserWriteMask* määrittäminen on mallintajan oman harkinnan varassa. Perusluokasta voidaan periyttää luokkia, jolloin ne perivät edellä esitetyt perusluokan attribuutit. Standardisoituja luokkia ovat *Variable, VariableType, Object, ObjectType, View, DataType, RerefenceType* ja *Method*, ja ne on määritetty standardissa IEC 62541-5 [74]. Näille luokille on määritetty luokkakohtaisesti jouk-



Kuva 5.2 *OPC UA:n metamalli UML-notaatiolla esitettynä.[75]*

ko attribuutteja. IEC 62541-3 määrittää metamallin, eli peruskonseptit joihin OPC UA perustuu. Standardissa IEC 62541-5 määritetään perustietomalli, joka toimii mallinnettavien järjestelmien kehyksenä.

OPC UA -tietomalli tarjoaa palvelimille standardisoidun tavan esittää objekteja asiakaskoneille. Objektit on rakennettu muista objekteista, muuttujista ja metodeista, ja ne kuvaavat olemassa olevia entiteettejä, esimerkiksi PI-kaavion pumppua. Näillä objekteilla voi olla myös yhteyksiä muihin objekteihin, kuten esimerkiksi pumpulla on olla laippa ja laippaan kiinnittyy putki. Tällaiset suhteet objektien välillä kuvataan viittauksilla. [10]

OPC UA -palvelin rakentuu objekteista ja niihin liittyvästä tiedosta. Tieto minkä palvelin paljastaa asiakaskoneelle rakentuu osoiteavaruudesta, jota kutsutaan OPC UA:n termein AddressSpace:ksi. Nämä objektit ovat AddressSpace:ssa solmuja (Nodes), joilla on attribuutteja ja viittauksia, joilla solmut liittyvät muihin solmuihin. Edellä esiteltyjä kahdeksaa luokkaa käytetään kuvaaman osoiteavaruuden komponentteja. Nämä luokat ovat solmuluokkia (NodeClass) ja ne määrittelevät osoiteavaruudelle metatiedon ja ne on määritetty standardissa IEC 62541-3.

Riippuen solmun luokasta, toisin sanoen minkä solmuluokan (NodeClass:in) solmu

(Node) perii, sen mukainen on solmun graafinen esitys OPC UA AddressSpace:ssa. Solmuluokkien graafiset notaatiot on esitetty taulukossa 5.1. Varjostetut symbolit esittävät tyyppejä. *ObjectType*-solmuluokka on siis AddressSpace:n solmu, joka määrittelee *Object*-solmulle tyypin. *Reference* on nimettyosoitin (pointer) solmusta toiseen ja *ReferenceType* määrittää osoittimelle tyypin. Kaikilla viittauksilla on oltava tyyppi, mutta kaikilla objekteilla ei välttämättä ole määritettyä tyyppiä. [73]

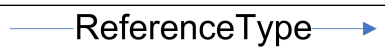
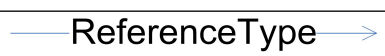
Taulukko 5.1 *OPC UA -solmuluokkien (NodeClass) notaatio.[75]*

Nimi	Notaatio
Object	Object
ObjectType	ObjectType
Variable	Variable
VariableType	VariableType
DataType	DataType
ReferenceType	ReferenceType
Method	Method
View	View

Kahden solmun välistä yhteyttä kuvataan viittauksella. Koska viittaus ei ole osoiteavaruuden solmu, niin siihen ei päästä suoraan käsiksi ja tämän vuoksi viittaukselle ei voida määrittää attribuutteja tai muita ominaisuuksia. Viittauksen semantiikka on määritettävissä viittaustyypin (ReferenceType) avulla. [59] Viittaustyypit on määritetty lähteessä [75] ja miten ne esitetään osoiteavaruudessa on määritetty lähteessä [74]. Viittaustyypit voidaan esittää osoiteavaruuden solmuina ja ne voidaan siten saattaa asiakaskoneen tietoisuuteen, vaikka palvelimelle olisi määritetty uusia, ei standardissa määritettyjä, viittaustyyppejä. Tämä on mahdollista, sillä viittaustyyppi on laajennettavissa oleva konsepti. [59]

Riippuen onko viittaus tyypiltään symmetrinen, asymmetrinen tai hierarkkinen, käytetään taulukon 5.2 mukaista notaatiota. Yleensä viitaukset osoittavat lähtösolmusta kohdesolmuun, mutta *HasSubtype*-viittaus on ainoa poikkeus sääntöön. Viittausnotaatioon liitetään myös viittaustyypin tai viittaustyyppijohdannaisen nimi. Taulukossa 5.3 on määritetty muutamille viittaustyypeille omat notaatiot. Huomaa

Taulukko 5.2 *OPC UA viittaustyyppien notaatio (ReferenceType). [75]*

Viitaustyyppi	Notaatio
Any symmetric ReferenceType	
Any asymmetric ReferenceType	
Any hierarchical ReferenceType	

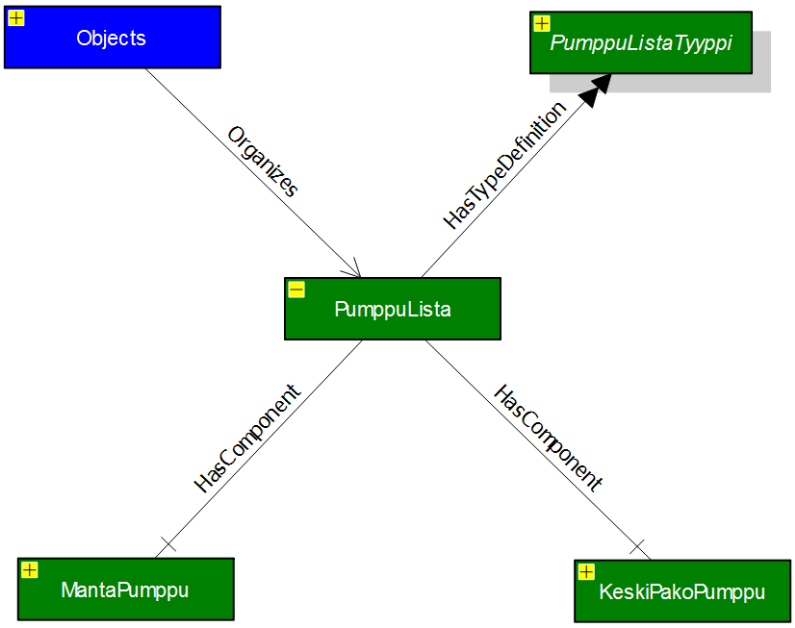
HasSubtype-viitauksen orientaatio taulukossa. Näitä voidaan käyttää taulukossa 5.2 esitettyjen notaatioiden ja viittaustyypin nimen yhdistelmien sijasta. [75]

Taulukko 5.3 *OPC UA viittauksien notaatio (Reference). [75]*

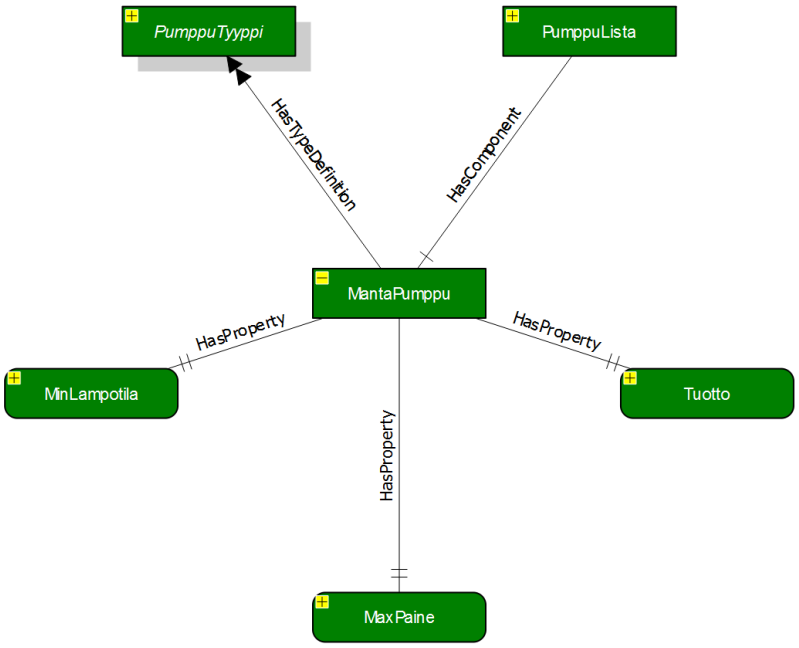
Viittaus	Notaatio
HasComponent	
HasProperty	
HasTypeDefinition	
HasSubtype	
HasEventSource	

Kuvassa 5.3 on mallinnettu ohjelmasta 2.3 tuttu pumppulista OPC UA:n konseptein. Solmulle *PumppuLista* on määritetty tyyppi *PumppuListaTyyppi* viittauksella *HasTypeDefinition*. *HasComponent*-viittauksilla solmulle *PumppuLista* on lisätty kaksi solmua, jotka mallintavat listalla olevia pumppuja *MantaPumppu* ja *KeskiPakoPumppu*. Viittaus *HasComponent* kertoo, että kohdesolmu on osa lähtösolmua (part-of). Solmu *Objects* on viittauksella *Organizes* kiinni *PumppuLista*:ssa. Kyseistä viittausta käytetään solmujen ryhmittelyyn.

Edettäessä AddressSpace:ssa syvemmälle *PumppuLista*:n komponenttiin *MantaPumppu*, löydetään mäntäpumpun tyyppimäärittely *HasTypeDefinition*-viittauksella *PumppuTyyppi* ja ominaisuudet *MinLampotila*, *MaxPaine* ja *Tuotto* *HasProperty*-viittauksin. *MantaPumpun* mallinnus on esitetty kuvassa 5.4.



Kuva 5.3 *Ohjelman 2.3 pumppulista OPC UA:n käsitteiden mukaisesti mallinnettuna. Kuvakaappaus UaModeler-ohjelmasta [111]*



Kuva 5.4 *Ohjelman 2.3 mäntäpumppu OPC UA:n käsitteiden mukaisesti mallinnettuna. Kuvakaappaus UaModeler-ohjelmasta [111]*

5.3 Simantics

Simantics [100] on avoin integraatioalusta, joka on kehitetty eritoten mallintamisen ja numeerisen simuloinnin tarpeisiin. Simantics-alustaa voidaan käyttää eri ohjelmistojen keskinäiseen integroimiseen sekä mallinnus- ja simulointityökalujen kehitysalustana. Alusta on julkaistu avoimena lähdekoodina ja se on lisensoitu EPL-lisenssillä (Eclipse Public License) [24]. Koska alusta halutaan pitää avoimena myös jatkossa, ja alustan jatkokehitys ja ylläpito halutaan turvata, on Teollisuuden hajautetun tiedonhallinnan yhdistys THTH ry:n [107] alaisuuteen perustettu Simantic-jaos. Simantics-alustan sovellustapauksia on esitetty lähteessä [46].

5.3.1 Simantics-alusta

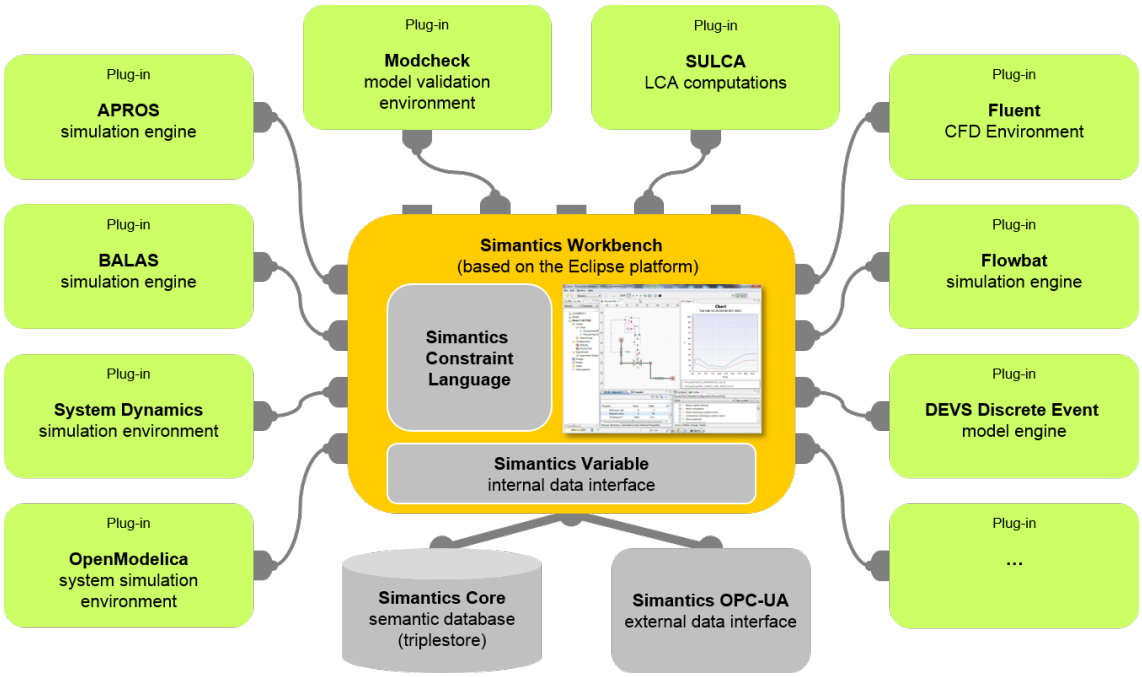
Simantics-alustan perustana on avoin ja laajennettava semanttinen tietomalli, jonka avulla on mahdollista esittää integroitujen ja implementoitujen järjestelmien tietoja. Integraattorialustalle tärkeä ominaisuus on pystyä käsittelemään eri järjestelmien tietomalleja. Tämä on toteutettu Simantics-alustassa ontologiaperusteisella muunnostoteutuksella, jonka avulla mallit kuvataan ja muunnetaan toisiksi malleiksi itse Simantics-alustassa. [46]

Laajennettavuuden ja joustavuuden nimissä Simantics-alustassa kaikki tieto esitetään semanttisen mallintamisen mukaisesti tripleteillä (triplet), jotka kuvaavat resurssien välisiä yhteyksiä. Resurssit ovat literaaleja tai viittauksia entiteetteihin. Tripletti on kuin luonnollisen kielen väittämä (statement), joka rakentuu subjektista, predikaatista ja objektista. [64] Esimerkiksi "*Pumpulla on ominaisuus tuotto*", jossa "*pumppu*" on subjekti, "*on ominaisuus*" on predikaatti ja "*tuotto*" on objekti ("*pump has property flow rate*"). Yhdessä resurssit ja väittämät muodostavat semanttisen verkon (semantic graph), jossa resurssit ovat solmuja ja väittämät suunnattuja viivoja (directed edges) [64].

Simantics-alustalle on kehitetty oma kuvauskieli Layer0 ontologioiden mallintamiseen, joka on määritetty lähteessä [64]. [46] Layer0 määrittelee säännöt triplettien ja resurssien käytölle, ja siten myös ontologioiden mallintamisen peruskonseptit [53], eli Simantics-alustan eri ontologiat perustuvat Layer0-konsepteihin. Layer0:n kaltaisia ontologian kuvauskieliä ovat muun muassa RDF [120] ja OWL [119]. Perusteluita, miksi Simantics-alustalle on päädytty kehittämään oma räätälöity kuvauskieli, on esitetty lähteissä [46, 53, 113].

Simantics-alusta perustuu asiakas-palvelin -konseptiin, missä palvelin (Simantics Core) toimii tietokantana, jota käsitellään asiakaskoneen (Simantics Workbench)

ja sen käyttöliittymän kautta. Alustan asiakaspuolen ohjelmistokehys (framework) on toteutettu Eclipsen [23] liitännäisarkkitehtuurin (plug-in) mukaisesti. [30, 46]



Kuva 5.5 *Simantics-alustan liitännäisarkkitehtuuri mahdollistaa eri simulointi- ja mallinnussovelluksien integraation.*

5.3.2 Simantics Constraint Language

Simantics Contstraint Language (SCL) [101] on funktionaalinen ohjelmointikieli, joka on kehitetty silmällä pitäen kielen soveltuvuutta semanttisten tietorakenteiden käsittelyyn. SCL:a käytetään Simantics-alustassa muun muassa mallien generointiin, parametrisointiin ja muunnoksiin, tietokanta kyselyihin (query) ja komentosarjoihin (scripting). [34]

SCL on eräänlainen murre funktionaalisesta ohjelmointikielestä nimeltä Haskell [37], sillä SCL omaksuu suurimman osan Haskellin ominaisuuksista; tärkeimpänä mainittakoon muuttujien staattinen tyypittäminen. Verrattuna imperatiivisiin ohjelmointikieliin, kuten C++ ja Java, funktionaaliset ohjelmointikielet pyrkivät välttämään tarpeettomia tiloja (state). Tilat vältetään käyttämällä laskennassa lausekkeita (expression), jotka muistuttavat matemaattisia funktioita. Tämä tarkoittaa sitä, että funktiot toimivat vain parametreissaan saamansa tiedon perusteella. Funktiot palauttavat vain tuloksen, joka riippuu vain funktion parametreista. Näin ollen funktionaalisen ohjelmointikielen funktiot palauttavat aina saman arvon samoilla parametreilla ja niillä ei ole sivuvaikutuksia (side-effect), kuten vaikutuksia muihin tiloihin. Tällöin puhutaan viittausten läpikuultavuudesta (referential transparency). [41]

SCL hieman rikkoo funktionaalisen ohjelmoinnin paradigmoja sallimalla hallitut sivuvaikutukset ja tilallisen ohjelmoinnin, eli ohjelmat voivat sisältää sijoituslauseita (assignment statement). Sivuvaikutuksien hallittavuus tarkoittaa sitä, että funktion sivuvaikutukset määritetään funktion rakentajassa, ja näin on mahdollista jäljittää missä ja mitä sivuvaikutuksia on käytetty. [34] SCL:n evaluointimalli on tiukka (strict, call by value), jossa kaikki lausekkeet ja funktiot evaluoidaan ennen suorittamista, toisin kuin Haskellissa. Haskellin evaluointimalli on puolestaan laiska (lazy), jossa funktioita pyritään suorittamaan niin vähän kuin mahdollista. [34, 41]

5.3.3 Muunnoskielet

Simantics-alustalla on kolme lähestymistapaa käsitellä mallimuunnoksia; joko rakentamalla muunnos lähdemallista kohdemalliksi resurssi ja relaatio kerrallaan käyttäen SCL-kieltä, tai hyödyntämällä SCL:n alikielitä (sublanguage) STL:a (Simantics Transformation Language) [36] tai CHR:ia (Constraint Handling Rules) [98]. Oikeuden mukaisuuden nimissä mainittakoon, jos SCL:n harkitaan olevan mallimuunnokselle soveltuva lähestymistapa, niin neljäs mahdollinen tapa toteuttaa mallimuunnokset on käyttää Simantics-alustan Java-sidonnaisuutta ja toteuttaa mallimuunnos Java-ohjelmointikielellä. Mutta hyödyntämällä mallimuunnokselle tarkoitettuja tekniikoita, kuten STL:a tai CHR:ia, aikaansaadaan luettavampaa ja rakenteellisempaa koodia, jonka modulaarisuuden ansiosta muunnokseen on mahdollista tehdä muutoksia koskematta aikaisempiin muunnossääntöihin [35].

STL muistuttaa muunnoskielenä kappaleessa 2.4 esitettyä QVTr:ia. STL:ssa muunnoksen inkrementaallisuus, yksisuuntaisuus ja kaksisuuntaisuus, on otettu huomioon. [35, 36] Itse muunnokset rakentuvat muunnossäännöistä, jotka määrittelevät lähde- ja kohdemallin elementtien välille riippuvuuuksia. Säännöt rakentuvat osista, jotka määrittelevät ehdot joiden on oltava voimassa muunnoksen jälkeen. Oleellisimmat säännön osaset ovat:

- @when: ehto lähde- ja kohdemallin välisille riippuvuuksille.
- @from: yhteensovittaa lähdemallin osaset.
- @to: yhteensovittaa kohdemallin osaset.
- @where: jälkiehto muiden sääntöjen kutsumiselle. [36]

Ohjelmassa 5.1 on esitetty esimerkki CreateEquipmentObject-nimisestä STL-säännöstä. Sääntö on sidoksissa MapEquipments-ehtoon, joka sitoo tässä tapauksessa lähde- ja kohdemallin toisiinsa. Muuttujat *?source* ja *?target* ovat tässä tapauksessa

lähde- ja kohdemallien juurielementit. Sääntö etsii lähdemallista (*?source*) relaatiolla *PRO.hasEquipment* yhteydessä olevia elementtejä (*?equipment*). *PRO* on Proteus-ontologialle asetettu nimiavaruus, ja *hasEquipment* on kyseisen ontologian mukainen relaatio eli predikaatti. Tämän jälkeen sääntö yhteensovittaa löydetyt elementit kohdemalliin ja luo tarvittavat elementit. Säännön suorittamisen jälkeen kaikki lähdemallin juurielementissä (-subjektissa) relaatiolla *hasEquipment* (-predikaatilla) kiinnitettyä *Equipment*-elementtiä (-objektia) kohden on luotu kohdemalliin *UAObject*-elementti *hasUAObject*-relaatiolla.

```
rule CreateEquipmentObjects where
    @when
    MapEquipments ?source ?target

    @from
    PRO.hasEquipment ?source ?equipment

    @to
    UA.UANodeSet.hasUAObject ?target ?equipmentObject
    LO.InstanceOf ?equipmentObject UA.ComplexTypes.UAObject
```

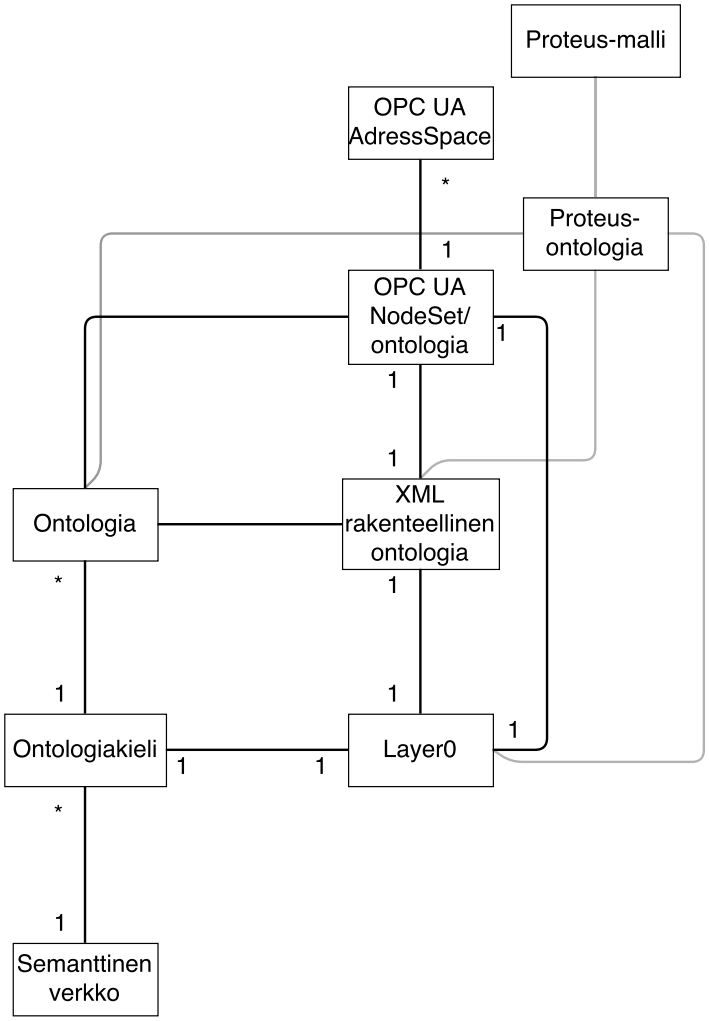
Ohjelma 5.1 *Esimerkki STL-muunnoskielen säännöstä, jolla tietokantaan tuodun Proteus-tiedoston Equipment-objektit muunnetaan OPC UA -objekteiksi.*

Ohjelmassa 5.2 on esitetty geneerisempi versio ohjelman 5.1 toteutuksesta. Edellä esitetty lähestymistapa on relevantti, jos tarkoituksena on muuntaa valikoituja osia lähdemallista. Jos muunnoksessa halutaan muuntaa kaikki tieto mallista toiseen, on jokaista relaatiota vastaavan säännön määrittely työlästä.

Kuvassa 5.6 on esitetty Simantics-alusta semanttisten mallien kerroksittainen rakenne, josta on nähtävissä miten tämän työn kannalta oleelliset ontologiat ovat yhteydessä toisiinsa. Proteus-ontologia ja OPC UA NodeSet-ontologia ovat keskenään samalla ontologian tasolla, vaikka kuvassa Proteus-ontologia on näistä kahdesta ylempänä. Samaa pätee molempien ontologioiden malli-instansseihin. Hyödyntämällä tätä tietoa mallimuunnoksessa, voidaan säännöissä käyttää myös ylemmäntason ontologioita. Proteus-ontologian *PRO.hasEquipment* on alirelaatio XML-ontologian relaatiolle *XML.hasElement*. Näin ollen käyttämällä muunnossäännössä relaatiota *XML.hasElement*, voidaan Proteus-tiedostossa XML-elementteinä esiintyneet elementit hakea Simantics-tietokannasta. Lisätään ohjelman 5.1 muunnossäännölle rekursiivinen jälkiehto-osa (*@where*), joka saa säännön kutsumaan itseään, jos lähdemalli sisältää sisäkkäisiä rakenteita.

```
rule CreateObjects where
    @when
    MapElements ?source ?target

    @from
    XML.hasElement ?source ?element

    @to
    UA.UANodeSet.hasUAObject ?target ?oject
    LO.InstanceOf ?object UA.ComplexTypes.UAObject

    @where
    MapElements ?element ?target
```

Ohjelma 5.2 *Ohjelman 5.1 muunnossäännön geneerisempi toteutus.*



Kuva 5.6 *Simantics-alustan semanttisten mallien kerroksittainen rakenne. Mukailtu lähteestä [46].*

Itse muunnoksen toteutukselle on määritetty kolme toteutustapaa, joista kaksi on inkrementaallisia. Yksinkertaisin toteutustapa on *kertatoteutus* (one-shot execution), joka yksinkertaisesti luo lähdemallista kohdemallin ilman minkäänlaista tukea inkrementaallisuudelle. Toinen toteutustapa on *kohdeinkrementaallinen toteutus* (target-incremental execution), joka mahdollistaa lähdemalliin tehtyjen muutoksien päivittämisen kohdemalliin ilman koko kohdemallin uudelleen generointia, mutta se käy läpi koko lähdemallin löytääkseen siihen tehdyt muutokset. Eli tämä toteutustapa on kohdeinkrementaallinen, muttei lähdeinkrementaallinen. Kolmas toteutustapa on *reaktiivinen toteutus* (reactive execution), joka on puolestaan lähdeinkrementaallinen. Toisin sanoen, se reagoi ainoastaan lähdemalliin tehtyihin muutoksiin ja päivittää kohdemallia muutoksien mukaisesti. [36] Muunnostoteutustapojen inkrementaallisuus on saavutettavissa Simantics-alustalla tallentamalla muunnoksen jäljet, eli mitä reittiä jokin lähdemallin elementti muuttui kohdemallin elementiksi, semanttiseen tietokantaan [35, 36].

Constraint Handling Rules [29] (CHR) on sääntö-/logiikkapohjainen ohjelmointikieli, joka voidaan upottaa osaksi isäntäkieltä. CHR-toiminnallisuus on toteutettu muun muassa seuraavissa ohjelmointikielissä: Haskell, Java, Prolog ja HAL [93]. CHR:in toteutus SCL:ssa on dokumentoitu lähteessä [98]. CHR taipuu myös graafimuunnoksiin [89], ja on siten myös sovellettavissa Simantics-alustan semanttisen graafitietokannan käsittelyyn, eli sitä voidaan hyödyntää mallimuunnoksiin. Ohjelmassa 5.3 on ohjelmaa 5.1 vastaavaa toteutus, joka käyttää SCL:n CHR-ominaisuutta.

```
1 when PRO.hasEquipment source ?Equipment
  then Equipment ?Equipment (newResource ())
3
  when Equipment ?Equipment ?EquipmentUAObject
5 then claim ?EquipmentUAObject L0.InstanceOf UA.ComplexTypes.UAObject
       claim target UA.UANodeSet.hasUAObject ?EquipmentUAObject
```

Ohjelma 5.3 *Ohjelman 5.1 muunnossääntöä vastaavaa toteutus CHR-muunnoskielellä.*

STL:n ja CHR:in tuottamat mallit voivat poiketa toisistaan, sillä muunnokset toteutetaan eritavoilla. STL:ssa ensiksi säännöt järjestetään riippuvuuksien perusteella, etsitään lähdemallista sääntöjä vastaavat resurssit ja lopuksi luodaan kohdemalli. CHR:ssa puolestaan säännöt suoritetaan siinä järjestyksessä kuin ne on määritelty. Jos säännölle ei löydy kohdemallista vastinetta tai kyseinen sääntö on jo suoritettu tarjotuilla lähdemallin resursseilla, otetaan järjestyksessä seuraava sääntö käsittelyyn.

5.3.4 Simupedia

Simupedia [103] on verkkosovelluskehityksen työkalu kaikille Simantics-alustalla kehitetyille simulointi- ja mallinnustuotteille. Varsinainen laskenta tapahtuu Simupedia-palvelimella ja itse Simupedia-sovellus suoritetaan verkkoselaimessa. Tämä mahdollistaa mallien, simulointien ja tuloksien jakamisen kohdehenkilöiden kesken siten, että loppukäyttäjän ei tarvitse asentaa verkkoselaimen lisäksi muita ohjelmistoja. [62]

Simupedia-sovellukset kehitetään samaisella editorilla, jota käytetään Simantics-alustalla mallintamiseen. Simupedian mukana tulee valmiita widgettejä (napit, tekstikentät, kaaviot jne.) ja komponentteja, joita voidaan käyttää sinällään tai rakennuspalikoina omille komponenteille. Kehittäjä voi jo näillä peruspalikoilla laatia oman käyttöliittymän. Varsinainen kehittäminen ei sinällään vaadi aikaisempaa ohjelmointikokemusta, koska käyttöliittymä rakennetaan vedä ja pudota -tekniikkaa (drag and drop) hyödyntäen. Simupedia käyttää taustalla Vaadin-ohjelmistokehystä [112], minkä vuoksi luodut sivustot toimivat lähestulkoon selaimella kuin selaimella ja niihin ei ole tarvetta asentaa erillisiä liitännäisiä. [62, 103] Tässä työssä kehitetään Simupedia-sovellus Proteus-tiedoston muuntamiseen OPC UA:n AddressSpace:een. Sovellus vie XML-formaatissa olevan Proteus-tiedoston Simantics-alustan tietokantaan, muuntaa tiedoston ja tarjoaa muunnettua tiedostoa käyttäjälle ladattavaksi. Ajatuksena on tarjota avoin rajapinta mallimuunnokselle DEXPI-tietomallista OPC UA -tietomalliin. Muunnos käyttää edellisessä kappaleessa kuvatulla SCL-kielellä ohjelmoituja funktioita.

5.4 Mallin muuntaminen

Tässä kappaleessa esitetään mallimuunnokset DEXPI-mallista Proteus/DEXPI-malliksi, ja Proteus/DEXPI-mallista edelleen OPC UA:n AddressSpace:een. Proteus/-DEXPI-tietomalli siis kuvaa siirtoformaatin rakenteen ja se perustuu puhtaasti Proteus-skeemaan. On hyvä muistaa, että DEXPI-malli on täysin yhteensopiva Proteus-mallin kanssa. DEXPI-malli muokkaa joitakin Proteus-skeemaan luokkia ja määrittelee joukon uusia attribuutteja, sekä muuttaa joidenkin Proteus-skeeman mukaisten attribuuttien kardinaalisuutta[1] (cardinality). Esimerkiksi Proteus-skeema ei vaadi *<GenericAttribute>*-elementeille attribuutin *AttributeURI* käyttöä, mutta DEXPI-spesifikaatio vaatii. *AttributeURI*:lla sidotaan mallinnettava elementti standardiesitykseen viittaamalla RDL-kirjaston vastaavaan URI:iin.

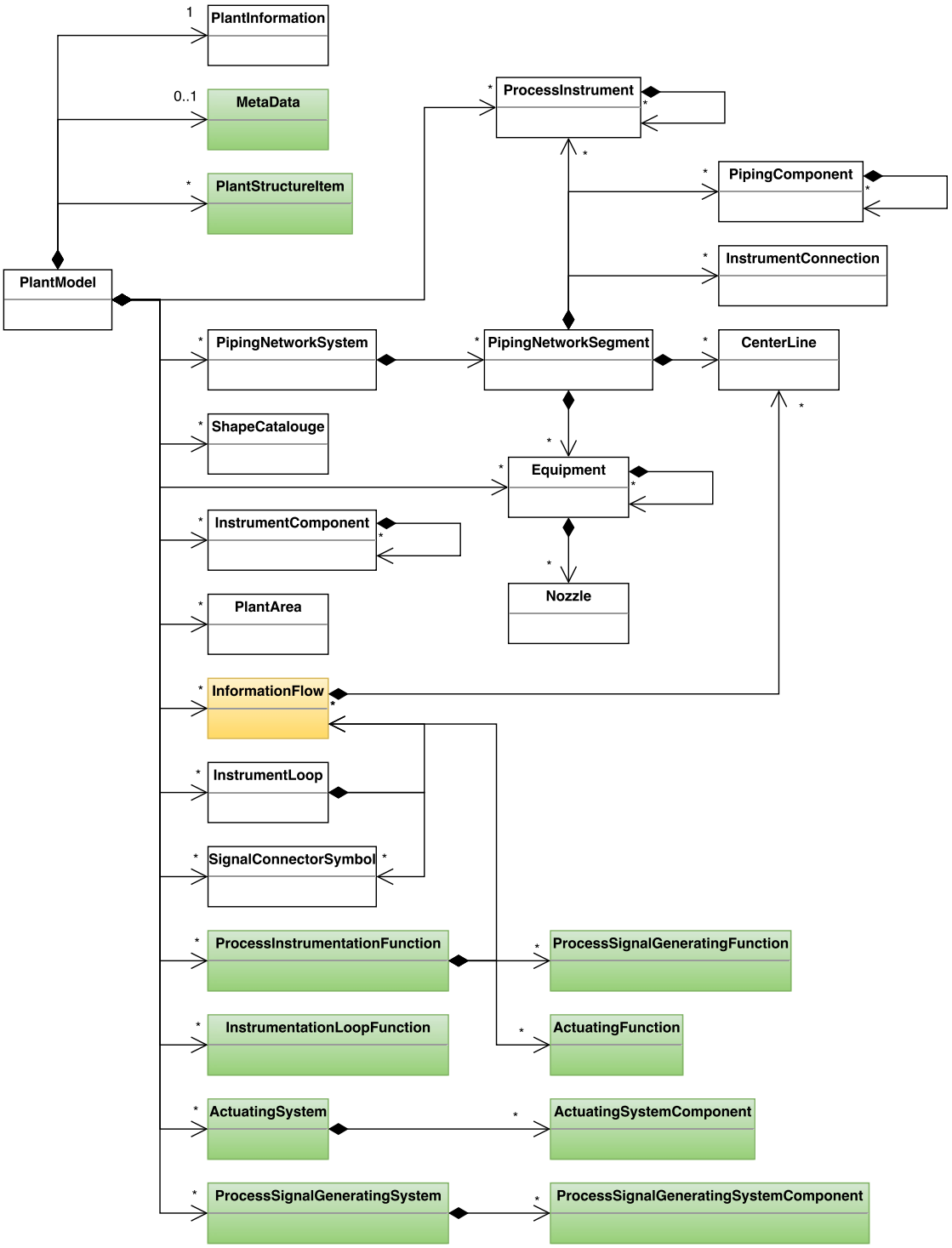
[1] Tässä kardinaalisuudella määritetään kuinka monta suhdetta elementillä voi olla toisiin elementteihin tai attribuutteihin. Valinnaisen yksi yhteen -kardinaalisuuden (one-to-one) määrittäminen XML-termein: elementillä *minOccur="0"* ja attribuutilla *use="optional"*.

5.4.1 DEXPI

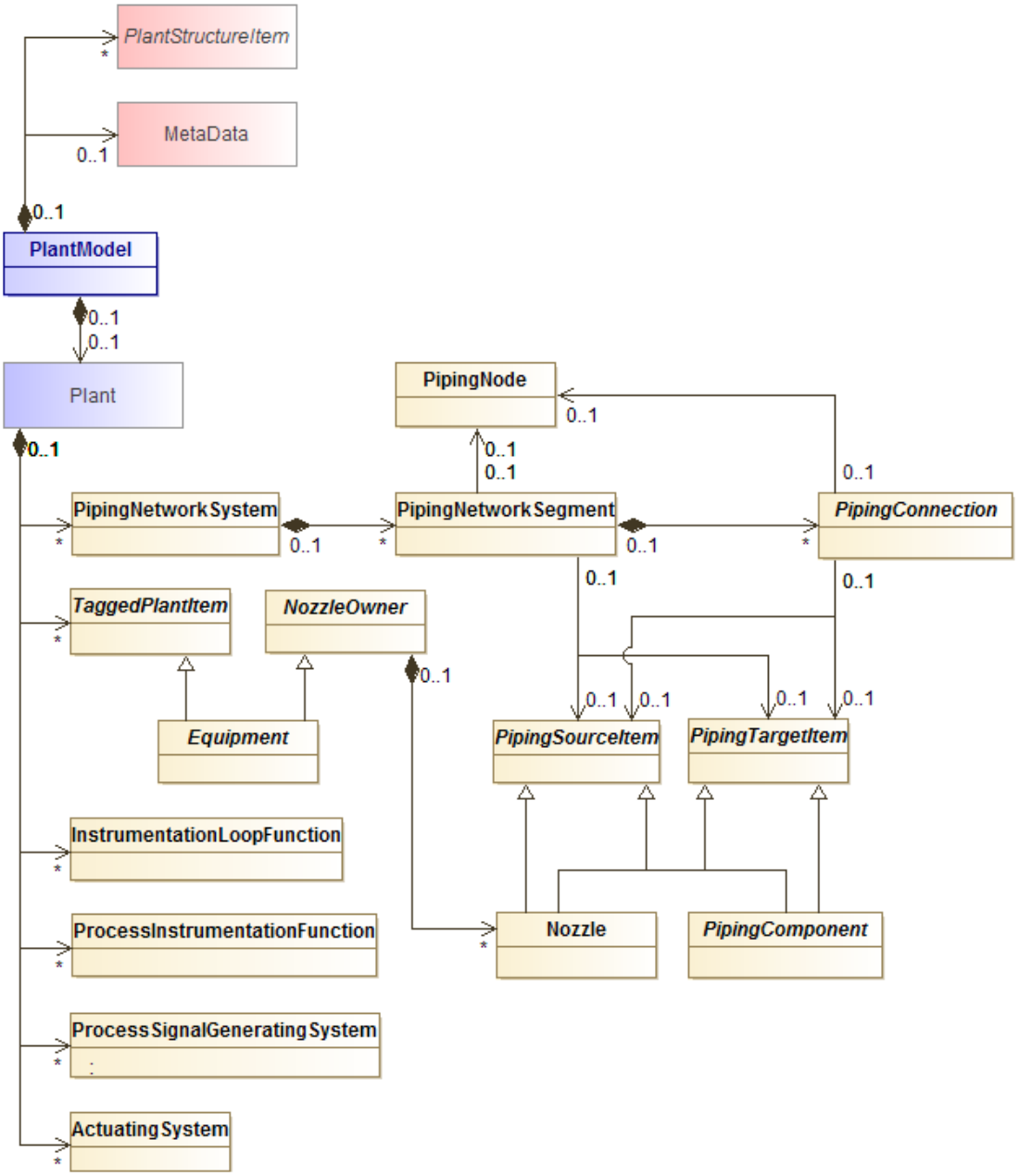
Kuten jo aikaisemmin on todettu, DEXPI-spesifikaatio ja siten myös DEXPI-tietomalli pohjautuu Proteus-skeemaan. Proteus-skeemaversion 4.0.1 instrumentoinnin toteutus on muuttunut versiosta 3.6.0. Tämän muutoksen myötä skeemaan on lisätty elementtejä. Lisäykset Proteus-skeemaan on esitetty kuvassa 5.7 vihreällä ja elementin *SignalLine* nimenmuutos *InformationFlow*:ksi on esitetty keltaisella. Kuvan kaavio on yksinkertaistus Proteus-skeeman rakenteesta, mutta se esittää oleellisimmat rakenteelliset elementit. Kuvan nuolet osoittavat vanhemmasta lapseen ja kompositio-relaatio osoittaa elementin vastuullisuudesta toiseen elementtiin nähden. Esimerkiksi, jos elementti *Nozzle* poistetaan mallista, sen poistaminen ei poista *Equipment*-elementtiä, mutta elementin *PlantModel* poistaminen poistaa kaikki muut mahdolliset elementit, koska *PlantModel* on tässä mallissa juurielementti.

Kuvassa 5.8 on yleiskatsaus DEXPI-tietomallista, jossa kursiivisella tekstillä nimetyt elementit ovat abstrakteja, eli ne eivät esiinny itse mallissa. Esimerkiksi DEXPI-mallissa ei esiinny *Equipment*-elementtejä kuten Proteus-mallissa. Kuvaan on otettu mukaan oleellisimmat rakenteelliset elementit. Toisin kuin Proteus-mallissa, DEXPI-mallin juurielementtinä voi esiintyä joko *PlantModel*- tai *Plant*-elementti. *PlantModel*-elementtiä ei ole välttämätöntä käyttää tapauksissa joissa ei käytetä elementin *PlanStructureItem* instansseja tai elementtiä *MetaData*.

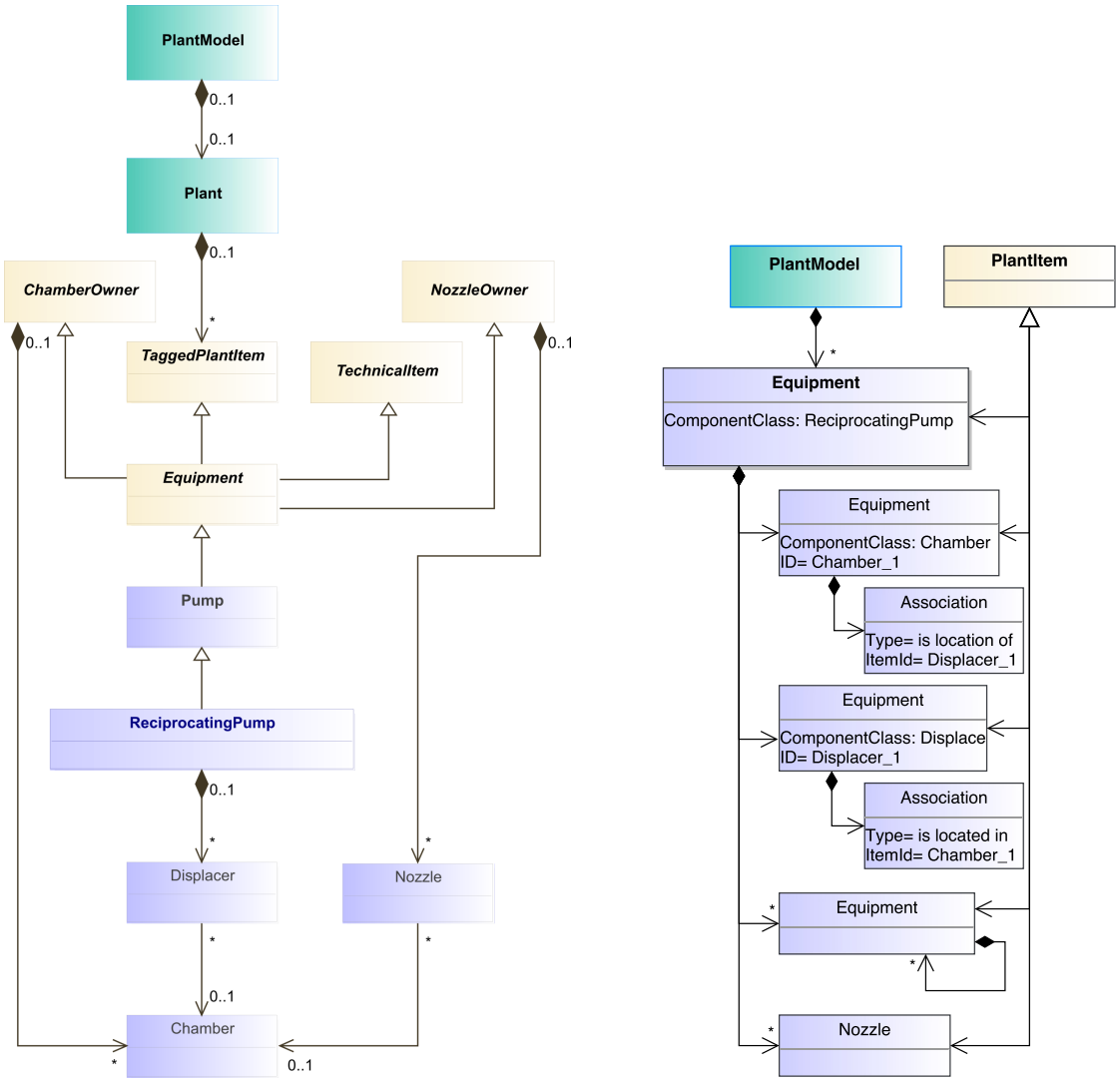
Kuvassa 5.9 on esimerkki, miten DEXPI-mallin mäntäpumppu mallinnetaan tiedonsiirtoa varten Proteus/DEXPI-malliksi. Kuten kuvasta 5.9(a) nähdään, mallin juuressa voi olla *PlantModel*-elementti tai *Plant*-elementti, mutta riippumatta kumpi edellä mainituista elementeistä DEXPI-mallin juuressa esiintyy, Proteus/DEXPI-mallin juuressa on aina *PlantModel*-elementti (ks. 5.9(b)). DEXPI-mallissa elementit perivät attribuutteja monelta eri abstraktilta-elementiltä. Esimerkiksi kuvassa 5.9(a) abstrakti elementti *Equipment* perii elementit *ChamberOwner*, *TaggedPlantItem*, *TechnicalItem* ja *NozzleOwner*, ja *Pump* perii edelleen *Equipment*-elementin ja lopuksi kaikki edellä periytetyt ominaisuudet kumuloituvat elementille *ReciprocatingPump* (mäntäpumppu) sen omien attribuuttien lisäksi. Proteus/DEXPI-mallissa kaikki rakenteelliset elementit perivät *PlantItem*-elementin, ja moni elementti laajentaa *PlantItem*-elementistä perittyjä ominaisuuksia ja määrittelevät omia elementtikohtaisia attribuutteja. Kuvassa 5.9(b) ei saa sekoitta kompositio-relaatiota assosiaatio-relaatioon. *Association* on Proteus-skeeman elementti, jolla viitataan toisiin elementteihin niiden attribuuttien *ID*, *Name* tai *Tag* perusteella.



Kuva 5.7 *Yleiskatsaus Proteus-skeeman (versio 4.0.1) rakenteellisesta mallista.*



Kuva 5.8 *Yleiskatsaus DEXPI-mallista. Kuvakaappaus Modelio-ohjelmasta [63].*



(a) DEXPI-tietomallin mukaisesti mallinnettu mäntäpumppu. Kuvakaappaus Modelio-ohjelmasta [63].

(b) Proteus/DEXPI-mallin mukainen esitys mäntäpumpusta.

Kuva 5.9 *Vertailuesimerkki DEXPI- ja Proteus/DEXPI-mallin välisistä eroista.*

5.4.2 Proteus/DEXPI

Proteus-skeeman oleellisimmat rakenteelliset elementit on esitetty luokkamallina kuvassa A.1. Nuolet osoittavat vanhemmasta lapseen (parent/child) ja esittävät siten Proteus-tiedoston hierarkkisen rakenteen. Mallimuunnoksessa jokaista kuvan luokkaa, eli Proteus-tiedostossa esiintyvää XML-elementtiä, vastaa OPC UA:n AdressSpace:ssa *UA*-objekti (NodeClass = Object) ja attribuutteja *UA*-muuttuja (NodeClass = Variable). Nämä esiintyvät AddressSpace:ssa solmuina, joille on määritelty solmuluokka (NodeClass). AddressSpace:n XML-serialisaatiossa solmuluokat esitetään XML-elementin nimessä. Solmuluokkaa Object vastaa XML-elementti $<$*UAObject...*$>$ ja solmuluokka *Variable*:a $<$*UAVariable...*$>$.

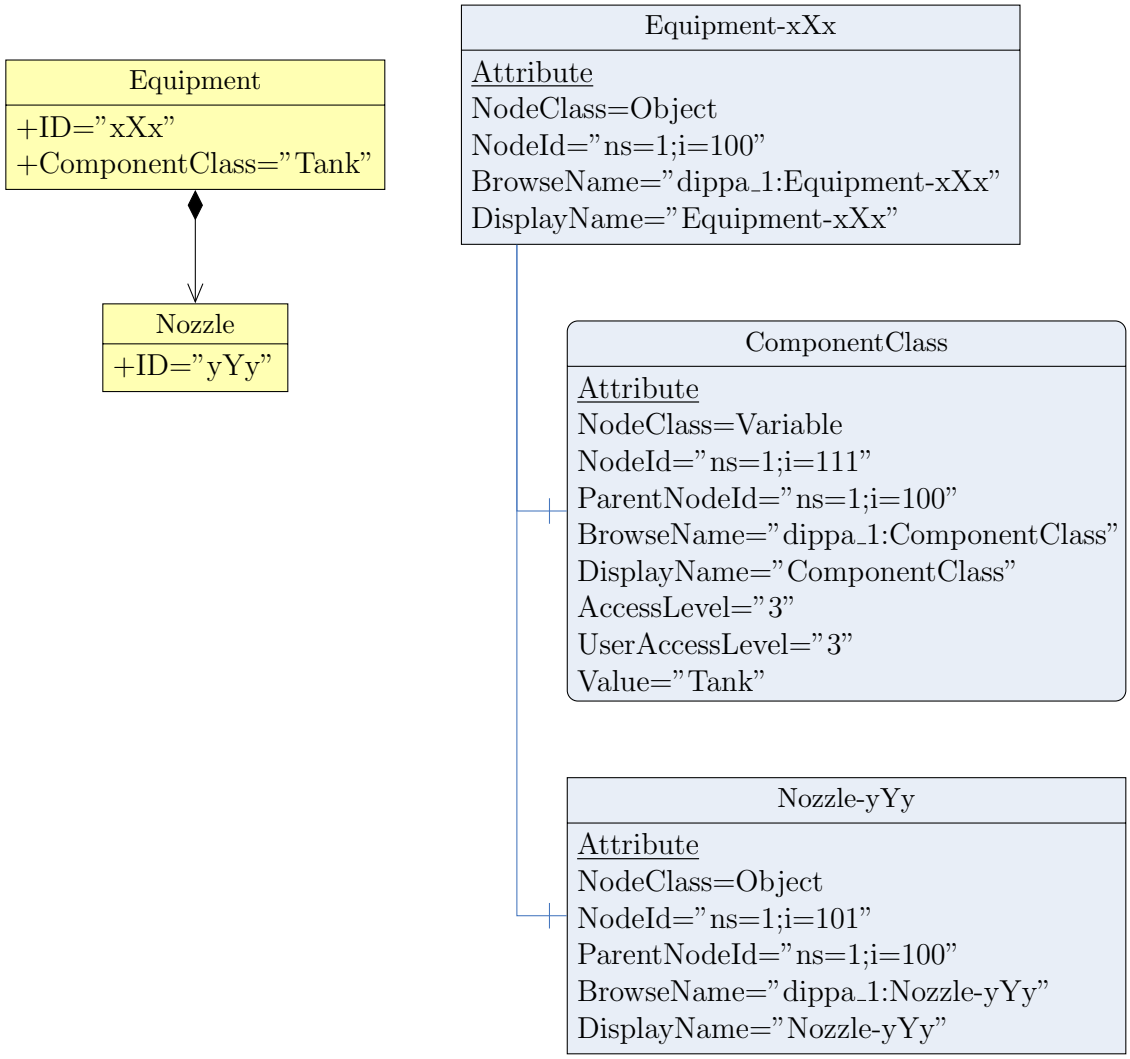
Kuvassa 5.10 on luokkamalliesimerkki Proteus-mallin esittämisestä OPC UA:n mallinnuskonseptein. Kuvassa vasemmalla keltaisella on Proteus-malli, jossa on *Equipment*-objekti *ID*- ja *ComponentClass*-attribuuttien kera. *ID*-attribuutti yksilöi objektin ja sitä voidaan käyttää ristiviittaamiseen objektista toiseen. *ComponentClass*-attribuutti määrittää *Equipment*-objektille laiteluokan. *Equipment*-objektilla on kompositio-relaatiolla yhdistetty *Nozzle*-objekti. Toisin sanoen tämä luokkamalli esittää tankkia, jolla on yksi liitäntälaippa.

Vastaava tankkimalli on esitetty kuvassa 5.10 oikealla puolella OPC UA AddressSpace-mallina. Notaatio vastaa kappaleessa 5.2 esitettyä. Kuva esittää perustavanlaisesti kuinka mallimuunnos Proteus-tiedostosta AddressSpacee:n tapahtuu. Jokaista Proteus-tiedoston XML-elementtiä (kuvassa objektia) vastaa UA-objekti ja attribuuttia vastaa UA-muuttuja. *Equipment*-objektin *ID*-attribuuttia ei ole kuvassa 5.10 esitetty UA-muuttujana, mutta sen käyttö UA-objektin ja UA-muuttujan nimeämisessä puolestaan on. *ID*-attribuuttia käytetään hyväksi AddressSpace:n solmujen nimeämisessä luettavuuden lisäämiseksi. Sen sijaan, että kaikki Proteus-tiedoston *Equipment*-elementit esiintyisivät AddressSpace:ssa *Equipment*-nimisinä solmuina, lisätään jokaisen muuntuvan UA-objektin attribuutteihin *BrowseName*:n nimiosion ja *DisplayName*:n jatkoksi *ID*-attribuutin arvo. Jos asikaskone haluaa esittää käyttäjälle palvelimen AddressSpace:n solmujen nimet, on niiden käytettävä *DisplayName*-attribuutin arvoa [75]. XML-attribuuteilla ei ole XML-elementtien *ID*-attribuutin kaltaista yksilöintiä, joten UA-muuttujien nimeämiseen liittyvien attribuuttien jatkoksi ei ole lisätä yksilöintitietoa. Jokaisella AddressSpace:n solmulla on yksilöllinen *NodeId*, joka erottelee solmut toisistaan AddressSpace:n näkökulmasta.

UA-muuttujilla on kaikilla solmuilla esiintyvien perusattribuuttien lisäksi attribuutit *AccessLevel* ja *UserAccessLevel*. *AccessLevel* ilmaisee muuttujan *Value*-attribuutin

arvon saavutettavuuden, eli onko arvoon luku- tai kirjoitusoikeutta (read/write), ottamatta kantaa käyttäjän luku/kirjoitusoikeuksiin [75]. Esimerkiksi *Value*-attribuutin arvolle on voitu sallia kirjoitusoikeudet, mutta pääsy tähän tietoon on voitu rajoittaa muulla tapaa. *UserAccessLevel*-attribuutti sisältää saman tiedon kuin *AccessLevel*-attribuutti, mutta ottaen huomioon käyttöoikeudet. Asettamalla näiden attribuuttien arvot kolmoseksi, annetaan käyttäjälle oikeudet *Value*-attribuutin arvon muuttamiselle lukemisen lisäksi.

Lapsisolmuille lisätään *ParentNodeId*-attribuutti, jonka arvoksi asetetaan solmun vanhemman *NodeId*. Näin solmujen välille saadaan UML:n kompositio-relaation kaltainen yhteys. Kun vanhempana esiintyvä solmu poistetaan AddressSpace:sta, niin lapsisolmut poistetaan myös.



Kuva 5.10 *Muunnosesimerkki Proteus-tiedostosta AddressSpace:en*

Taulukon 5.4 kohdassa 1 on esitetty esimerkki kuinka Proteus-tiedoston <*Equipment*>-elementti muunnetaan AddressSpace:n <*UAObject*>-elementiksi. UA-objek-

tilla on attribuutti *NodeId*, jota käytetään objektin yksilöintiin. Tässä se koostuu nimiavaruuden määrittelevästä osasta (*ns="..."*) ja solmun yksilöivästä osasta (*i="..."*), jonka pitää olla uniikki nimiavaruudessaan. Nimiavaruudella voidaan määrittää erillisiä näkymiä AddressSpace:ssa esitettyyn tietoon. Proteus-tiedoston XML-elementtien nimeä käytetään osana UA-objektin attribuuttia *BrowseName* ja sen elementtiä *DisplayName*. XML-elementin nimen jatkoksi lisätään elementin *ID*-attribuutin arvo, joka on kaikilla Proteus-tiedoston rakenteellisilla elementeillä. Esimerkiksi <*GenericAttribute*>-elementeillä ei ole *ID*-attribuuttia, tällöin käytetään vain XML-elementin nimeä. Jos objektin nimi halutaan näyttää käyttäjälle, asiakaskone käyttää *DisplayName*-elementin arvoa.

Proteus-tiedoston rakenteellisilla elementeillä voi olla myös lapsielementtejä. Esimerkiksi <*Equipment*>-elementillä on relaatio itseensä, kuten kuvasta A.1 voidaan nähdä, eli kyseisellä elementillä voi olla <*Equipment*>-elementtejä lapsielementteinä ja lapsielementeillä voi olla edelleen lapsielementtejä ja niin edelleen. AddressSpace:ssa jokaista elementtiä vastaa UA-objekti, ja sama pätee myös lapsielementteihin. AddressSpace:ssa lapsielementit esitetään viittauksin ja ainoastaan suoraan alenevassa polvessa oleviin lapsiin. Taulukon 5.4 rivillä 2 on esitetty esimerkki kuinka <*Equipment*>-elementin suhde lapsielementtiin <*Nozzle*> mallinnetaan AddressSpace:n XLM-serialisaatiossa. Koska Proteus-tiedoston rakenteellisilla elementeillä voi olla useampia lapsielementtejä ja attribuutteja, kootaan kaikki viitaukset UA-objektin <*References*>-elementin sisälle. Jokainen viittaus mallinnetaan erikseen <*Reference*>-elementeillä. Kaikki viittaukset niin lapsielementteihin kuin attribuutteihin tyypitetään *ReferenceType*-attribuutin arvolla *HasComponent*, joka vastaa semanttisesti osa-jotakin -relaatiota (part-of). Viittauksen arvoksi asetetaan viitattavan objektin *NodeId*. Viittaukset suuntautuvat oletusarvoisesti eteenpäin, eli rakenteellisessa hierarkiassa ylhäältä alas. Kun on tarve viitata objektin vanhempaan, asetetaan <*Reference*>-elementille *IsForward*-attribuutti, ja sen arvoksi *false*. Koska <*Equipment*>-elementti esiintyy suoraan juurielementin alla (*PlantModel*-juurielementti on jätetty pois selkeyden vuoksi), sillä ei ole AddressSpace:ssa vanhempana objektia, vaan objektit kokoava kansio. Tämä kansio on eräänlainen sisääntulopiste AddressSpace:ssa olevien objektien selaamiseen. Näin ollen tällaiset objektit viittaavat tuohon kansioon *Organizes*-viittauksella taaksepäin *NodeId*:llä *i=85*.

Taulukon 5.4 rivillä 3 on esitetty tapaus, joka vastaa rivin 2 esimerkkiä, mutta lapsielementin näkökulmasta. Lapsielementti muuntuu UA-objektiksi samoin kuin edellä, mutta sillä on lisänä *ParentNodeId*-attribuutti, jonka arvo on objektin vanhemman *NodeId*. Lisäksi objekti viittaa taaksepäin *HasComponent*-viittaustyypillä vanhempaansa sen *NodeId*:llä.

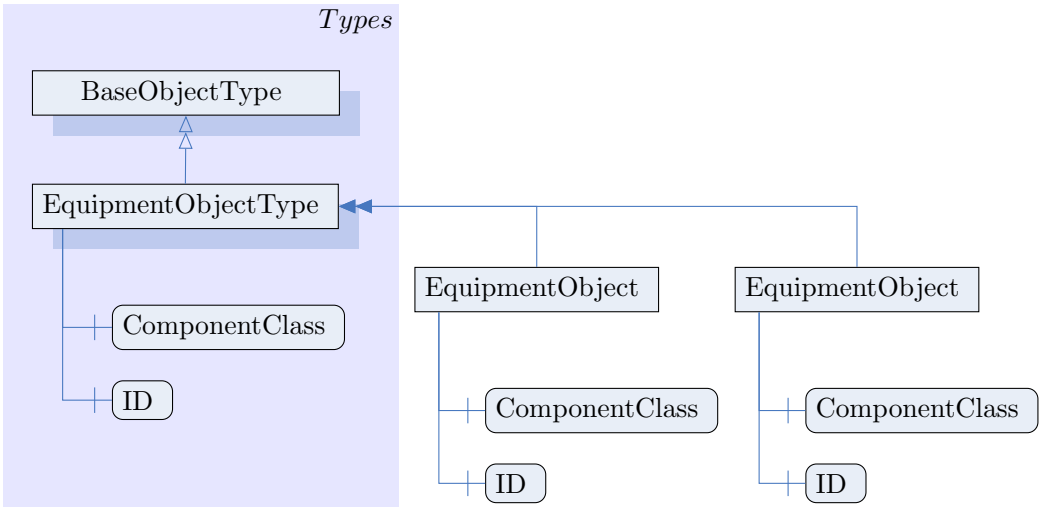
Taulukko 5.4 *Esimerkki, miten Proteus-tiedoston rakenteelliset elementit ja elementtien attribuutit ilmentyvät OPC UA:n NodeSet:issä.*

No.	Proteus	OPC UA
1.	<Equipment ID=”xXx”> ⋮ </Equipment>	<UAObject NodeId=”ns=1;i=100” BrowseName=”dippa_1:Equipment-xXx”> <DisplayName=”Equipment-xXx”/> </UAObject>
2.	<**Equipment** ...> <Nozzle .../> </**Equipment**>	<UAObject ...> <References> <Reference ReferenceType=”HasComponent”> ns=1;i=101</Reference> <Reference IsForward=”false” ReferenceType=”Organizes”> i=85</Reference> </References> </UAObject>
3.	<Equipment ...> <**Nozzle** .../> </Equipment>	<UAObject NodeId=”ns=1;i=101” ParentNodeId=”ns=1;i=100”...> <References> <Reference IsForward=”false” ReferenceType=”HasComponent”> ns=1;i=100</Reference> </References> </UAObject>
4.	<Equipment **ComponentClass=”Tank”**> </Equipment>	<UAVariable NodeId=”ns=1;i=111” BrowseName=”dippa_1:ComponentClass” ParentNodeId=”ns=1;i=100” AccessLevel=”3” UserAccessLevel=”3”...> <References> <Reference IsForward=”false” ReferenceType=”HasComponent”> ns=1;i=100</Reference> </References> <Value> <uax:String>Tank</uax:String> </Value> </UAVariable>

Proteus-tiedoston elementeillä on myös attribuutteja. Attribuutit muunnetaan AddressSpace:een UA-muuttujiksi, jotka muuntuvat lähes samantapaisesti kuin edellä esitetty lapsielementti-tapauksessa. Taulukon 5.4 OPC UA NodeSet-serialisaatio on esitetty kokonaisuudessaan ohjelmassa C.1. Nimitystä NodeSet käytetään AddressSpace-mallin XML-serialisaatiosta.

OPC UA tarjoaa mahdollisuuden objektien tyypittämiselle. Objektin tyyppi voi-

daan asettaa viittaamalla sen objektiin *TypeObject* viittaustyypillä *HasTypeDefinition* (attribuutin ReferenceType arvolla HasTypeDefinition). TypeObject-objekti on verrattavissa olio-ohjelmoinnin luokkaan. TypeObject-objektille voidaan määrittää muuttujia ja metodeja, ja periyttää ne UA-objektille. Määritetyt tyypit on nähtävissä myös AddressSpace:ssa. Kuvassa 5.11 on esitetty esimerkki OPC UA -objektien tyypittämisestä.



Kuva 5.11 *Esimerkki OPC UA -objektien tyyppimäärittelystä.*

Koska tässä työssä määritetään muunnos Proteus-skeeman mukaisesta Proteus-tiedostosta OPC UA AddressSpace:een, eli Proteus-tiedoston sisältö halutaan esittää OPC UA -palvelimen avulla, varsinainen objektien tyypittäminen ei palvele tarkoitusta. Tällöin muunnettaisiin myös tietomalli. Jos asiaa katsotaan työn alkupuolella esitettyjen mallintamisen konseptien näkökulmasta, niin Proteus-tiedosto on jo metamallinsa (Proteus-skeema) mukainen instanssi suunnittelutiedosta. Toisin sanoen Proteus-tiedostossa näytetään vain mitä on mallinnettu, eikä mitä olisi voitu mallintaa. Esimerkiksi käyttämättä jätettyjä attribuutteja ei instanssioida tiedostoon tyhjillä arvoilla. Toisaalta, jos joku haluaisi tarkastella muunnetun Proteus-tiedoston elementtien *mahdollisia* ominaisuuksia OPC UA -palvelimella tarjotun AddressSpace:n kautta, ja muokata ja/tai lisätä UA-objekteille Proteus-skeeman mukaisia ominaisuuksia, tällöin tyyppien määrittely ja niiden sitominen vastaaviin AddressSpace:n solmuihin olisi yksiselitteisesti edukasta.

Edellä esitetyt kuvaukset mallien välillä on vastaavien elementtien osalta koottu taulukkoon 5.5, josta nähdään, että Proteus-tiedoston elementtiä vastaa OPC UA -objekti, ja attribuuttia OPC UA -muuttuja. Relaatioiden osalta kuvaukset on esitetty taulukossa 5.6. Kun Proteus-tiedostossa elementeillä on lapsielementtejä ja/tai attribuutteja, käytetään AddressSpace:ssa HasComponent-relaatiota.

Taulukko 5.5 *Elementtien kuvaus.*

Proteus	OPC UA
Element	Object
Attribute	Variable

Taulukko 5.6 *Elementtien välisten relaatioiden kuvaus.*

Proteus Mistä	Proteus Mihin	OPC UA Relaatio	Selitys
Element	Element	HasComponent	Lapsielementti
Element	Attribute	HasComponent	Elementin attribuutti

Taulukossa 5.7 on kuvattu vain Proteus-skeemassa käytetyt tietotyypit OPC UA -tietotyypeiksi. XML-skeeman mahdolliset tietotyypit on määritetty lähteessä [117]. OPC UA:n tietotyypit on määritetty lähteessä [75] ja niiden esitys AddressSpace:ssa on määritetty lähteessä [74]. Tietotyyppien välillä on joitakin eroavaisuuksia. Esimerkiksi XML-skeeman totuusarvo-tietotyypillä (boolean) käytetään pienin kirjaimin arvoja true tai false, mutta OPC UA:ssa vastaavat arvot on kirjoitettu isoilla kirjaimilla TRUE tai FALSE.

Taulukko 5.7 *Tietotyyppien kuvaus Proteus-skeeman käyttämistä XML-tietotyypeistä OPC UA -tietotyypeiksi.*

XML	OPC UA	Kommentti
string	String	
double	Double	
boolean	Boolean	true/false → TRUE/FALSE
integer	Int64	OPC UA Integer on abstarkti → Int64
nonNegativeInteger	UInt64	OPC UA UInteger on abstarkti → UInt64
positiveInteger	Uint64	OPC UA ei määrittele omaa tietotyyppiä positiiviselle kokonaisluvulle.
date	DateString	
time	TimeString	
anyURI	String	
ID	String	
IDREF	String	
NMTOKEN	String	

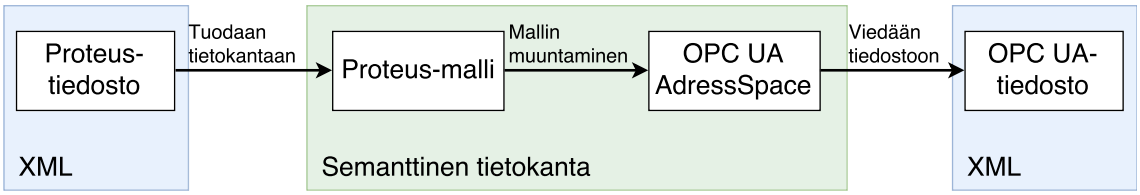
Edellä määritettyä kuvausta voidaan hyödyntää tapauksissa, joissa suunnittelutieto on esitetty Proteus-skeeman mukaisesti. Proteus-skeeman versiolla ei tässä tapauksessa ole merkitystä, sillä tarkoituksena on muuntaa kaikki Proteus-tiedostoon tallennettu tieto AddressSpace:een ottamatta kantaa Proteuksen tai DEXPI:n tietomalliin.

5.5 Muunnostyökalun toteutus

Tämä kappale esittelee Simantics-alustalla toteutetun mallimuunnosprosessin. Mallien välinen kuvaus poikkeaa edellisessä kappaleessa määritellystä tietotyyppien osalta. Lisäksi käydään läpi eräs muunnossääntö tutun esimerkin kautta ja esitellään mallimuunnosrajapinta.

5.5.1 Muunnosprosessi

Kuvassa 5.12 on kuvaus mallin muunnosprosessista. Simantics-alustan tietokantaan tuodaan suunnitteluohjelmistosta tuotu Proteus-skeeman mukainen XML-dokumentti. Tuodusta XML-dokumentista luodaan tietokantaan semanttinen malli, jota voidaan tarkastella ja muokata. Mallista voidaan luoda toinen malli hyödyntäen Simantics-alustan Java-sidonnaisuutta, SCL-, STL- tai CHR-kieliä, mitkä esiteltiin kappaleessa 5.3.3. Tässä työssä mallimuunnoksessa käytetään STL-kieltä ja paikoin myös SCL-kieltä. Kun malli muunnetaan, tietokantaan luodaan tuodusta lähdemallista kohdemalli. Muunnos noudattaa pääsääntöisesti kappaleessa 5.4.2 määritettyä kuvausta pienin muutoksin tietotyyppien kuvauksissa.



Kuva 5.12 *Prosessi Proteus-tiedoston muuntamisesta OPC UA -tiedostoksi.*

Taulukossa 5.8 on esitetty tietotyypit muunnoksen eri vaiheissa. Taulukon ensimmäinen sarake edustaa suunnitteluohjelmistosta tuodun XML-dokumentin tietotyyppejä, toinen sarake edustaa Simantics-alustan Layer0-ontologian tietotyyppejä, kolmas sarake esittää Layer0-tietotyyppien tulkintaa SCL-kielellä ja viimeisessä neljännessä sarakkeessa on vastaavat OPC UA -tietotyypit. Poiketen taulukon 5.7

määrityksistä, kaikki XML-tietotyypin desimaali (decimal) johdannaiset tietotyypit, kuten kokonaisluku (integer), muuntuvat loppujen lopuksi 32-bittiseksi OPC UA:n kokonaisluvuksi Int32. Tämä siitä syystä, että Simantics-alustassa käytetään 32-bittistä Java-arkkitehtuuria. Muunnoksessa menetetään tieto liittyen date- ja time-tietotyyppeihin, koska SCL:lla ei ole sidontaa vastaaviin Layer0-ontologian tietotyyppeihin vielä työn toteutus vaiheessa. Proteus-skeeman mukaan elementit <*PlantInformation*>, <*ShapeCatalogue*> ja <*Transaction*> voivat sisältää aikasidonnaisia tietotyyppejä.

Simantics-alustalle voidaan automaattisesti luoda uusia ontologioita XML-skeemoista [102]. Tämä skeema-ontologia -muunnin käsittelee esimerkiksi Proteus-skeeman *Coordinate*-elementin erillisistä X-, Y- ja Z-attribuuteista yhden kokoelman [X,Y,Z], joka on Simantics-alustassa tietotyyppiä DoubleArray. Tällainen tieto muunnetaan pilkulla erotelluksi merkkijonoksi “X,Y,Z” ja se esitetään OPC UA:ssa tietotyypillä String.

Taulukko 5.8 *Tietotyyppit mallin muunnosprosessin eri vaiheissa. Muunnoksessa menetetään tieto liittyen Proteus-skeeman käyttämiin XML-tietotyyppeihin date ja time.*

XML	**Simantics**	**SCL**	**OPC UA**
string	L0.String	String	String
double	L0.Double	Double	Double
boolean	L0.Boolean	Boolean	Boolean
integer	L0.Integer	Integer	Int32
nonNegativeInteger	L0.Integer	Integer	Int32
positiveInteger	L0.Integer	Integer	Int32
date	L0.Date	-	-
time	L0.Time	-	-
anyURI	L0.Uri	String	String
ID	L0.String	String	String
IDREF	L0.String	String	String
NMTOKEN	L0.String	String	String

Kun tietokantaan on luotu kohdemalli (AddressSpace), luodaan mallista XML-dokumentti (NodeSet). Toistaiseksi Simantics-alustan vientitoiminnallisuus ei tarkista luotua tiedostoa skeemaa vasten, joten tiedoston muotoilun oikeellisuudesta

ja kelvollisuudesta ei voida olla tässä vaiheessa vielä varmoja. Tämä selviää vasta, kun luodulla tiedostolla yritetään pystyttää OPC UA -palvelin tai viedä tiedosto OPC UA -mallinnusohjelmaan. Tässä työssä mallimuunnoksen ja tiedoston viennin tuloksena saatua tiedostoa testattiin yrittämällä pystyttää OPC UA -palvelin Prosys:n [79] ohjelmistokehityspaketilla OPC UA Java SDK v2.0.2-275 tai viemällä tiedosto UaModeler-ohjelmaan. Virheellisen tiedoston tapauksessa palvelimen pystyttäminen tai mallin muodostaminen mallinnusohjelmaan epäonnistuu. Kelvollisen tiedoston tapauksessa palvelimen pystyttäminen onnistuu ja palvelimen tarjoamaa tietoa voidaan tarkastella asiakasohjelmalla, kuten esimerkiksi UaExpert-ohjelmalla [110].

5.5.2 Muunnossäännöt

Muunnostoteutuksessa käytetään Simantics-alustalle määriteltyjä SCL- ja STL-kieliä. Simantics-alustan Proteus-tiedostojen tuontitoiminnallisuus ja NodeSet-tiedostojen vientitoiminnallisuus on valmiiksi toteutettuna, joten varsinainen toteutus koskee mallimuunnosta kahden eri mallin välillä. Muunnos noudattaa kappaleissa 5.4.2 ja 5.5.1 määritettyä kuvausta.

AddressSpace:ssa jokaisella solmulla pitää olla yksilöllinen NodeId. Sen sijaan, että toteutettaisiin erillinen funktio jokaisen AddressSpace:n solmun yksilöllisen NodeId:n allokoinnille, käytetään hyväksi Simantics-alusta semanttista tietokantaa. Jokaiselle semanttisen tietokannan entiteetille on määritetään oma yksilöllinen resurssitunniste kokonaislukuna. Muunnoksessa käytetään AddressSpace:n solmujen NodeId-attribuutin arvona tietokantaan tuodun Proteus-tiedoston entiteeteille allokoituja resurssitunnisteita.

Ohjelmassa 5.4 on esitetty XML-serialisaatio kuvan 5.10 Proteus-mallista. Ohjelmassa 5.5 on muunnossääntö *CreateObjects*, joka muuntaa kaikista XML-elementeistä UA-objekteja. Sääntö suoritetaan, kun ehtoa *ProteusToOPCUA* kutsutaan yhdessä lähde- ja kohderesurssin kanssa. Säännön *@from*-osiossa haetaan annetusta lähderesurssista kaikki XML-elementit ja AddressSpace:n objektien niemämiseen käytettäviä tietoja. *@to*-osiossa määritetään lähderesurssin vastaavuudet kohdemallissa, mikä lisätään lopuksi kohdemalliin. STL-säännön sisällä voidaan suorittaa myös SCL-kielen komentoja *Execute*-funktion avulla. Säännön *@where*-osiossa voidaan kutsua muita sääntöjä. Tämä sääntö kutsuu itseään, jos lähderesurssilla on muita XML-resursseja. Lisäksi sääntö kutsuu *MapUAVariables*-sääntöä, jos lähderesurssilla on attribuutteja, ja *MapReferences*-sääntöä OPC UA -tietomallin mukaisten referenssien muodostamiseksi. Kaksi edellä mainuttua sääntöä on määritelty muualla.

Ohjelmassa D.1 on esitetty ohjelmassa 5.5 esitetyn muunnossäännön tuotos, kun sille syötetään ohjelma 5.4. Muunnostuotos vastaa sisällöltään käsin laadittua muunnosta ohjelmassa C.1 pienin poikkeuksin UA-muuttujien lapsielementtien järjestyksessä ja arvojen tyypittämisessä. Muunnoksien testaamisessa käytetty OPC UA -ohjelmistokehityspaketti osaa käsitellä eri järjestyksessä olevat UA-muuttujien lapsielementit. UaModeler-ohjelma odottaa, että lapsielementit ovat järjestyksessä: *<DisplayName>*, *<References>*, *<Value>*. Molemmat muuttujan arvon tyypittämistavat ovat valideja molempien testaamisessa käytettyjen toteuksien näkökulmasta. Tietotyypeille voidaan määrittää nimiavaruus jo dokumentin alussa esimerkiksi muodossa

xmlns:uax=”tietotyypit-määrittelevän-skeeman-URI”

ja siten käyttää nimiavaruusmäärittelyä XML-elementeissä

<uax:Tyyppi>Arvo</uax:Tyyppi>.

Vaihtoehtoisesti tietotyyppi voidaan määritellä suoraan XML-elementin attribuuttina

<Tyyppi xmlns=”tietotyypit-määrittelevän-skeeman-URI”>Arvo</Tyyppi>.

Kuvassa 5.13 on graafinen esitys muunnostuotoksena saadusta NodeSet-tiedostosta. Kuvasta nähdään solmujen nimet ja millaisilla relaatiolla solmut ovat yhteydessä toisiinsa. Kuvassa 5.14 on näkymä, miten OPC UA -asiakaskone voi selata muunnostuotoksella alustettua OPC UA -palvelinta.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlantModel>
  <Equipment ID="xXx" ComponentClass="Tank">
    <Nozzle ID="yYy">
    </Nozzle>
  </Equipment>
</PlantModel>
```

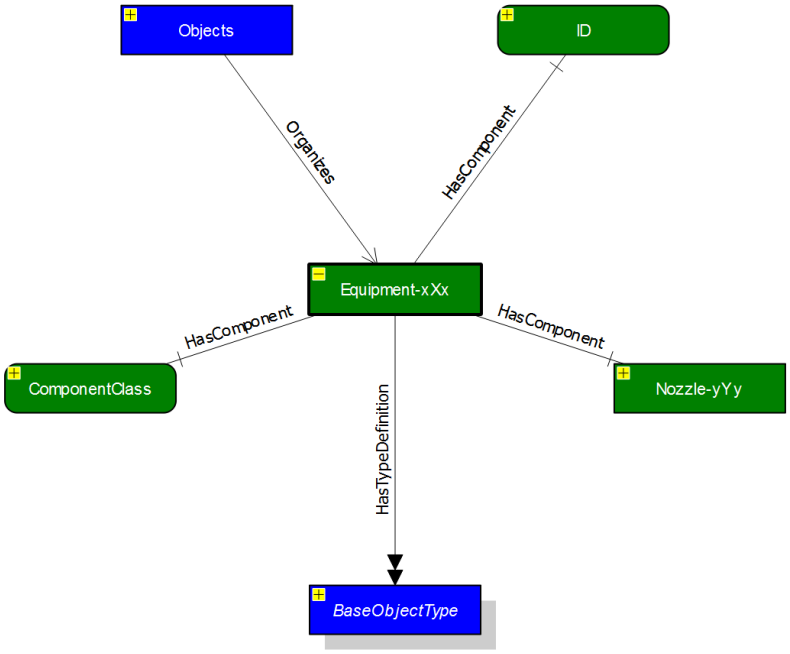
Ohjelma 5.4 *Kuvan 5.10 Proteus-mallia vastaava XML-serialisaatio.*

```
rule CreateObjects where
    @when
    ProteusToOPCUA ?source ?target

    @from
    XML.hasElement ?source ?element
    //is used as part of uaobject naming
    ?hasID = possibleRelatedValue ?element PI.hasID
    //used in ua namespace i=...
    ?resourceID = toString $ resourceId ?element
    L0.InstanceOf ?element ?instanceOf
    //used for uaobject name with id
    ?name = relatedValue ?instanceOf L0.HasName :: String

    @to
    UA.UANodeSet.hasUAObject ?target ?uaObject
    L0.InstanceOf ?uaObject UA.ComplexTypes.UAObject
    UA.ComplexTypes.UANode.hasReferences ?uaObject ?listOfRefs
    L0.InstanceOf ?listOfRefs UA.ComplexTypes.ListOfReferences
    Execute(claimRelatedValue ?uaObject \
        UA.ComplexTypes.UANode.hasNodeId ("ns=1;i="+?resourceID))
    Execute(claimRelatedValue ?uaObject \
        UA.ComplexTypes.UANode.hasBrowseName \
        ("dippa_1:"+?name+match ?hasID with \
        Just s -> "-"+s; Nothing -> ""))
    Execute(claimRelatedValue ?uaObject \
        UA.ComplexTypes.UANode.hasDisplayName \
        (?name+match ?hasID with Just s->"-"+s; Nothing->""))

    @where
    //if XML element has child elements
    ProteusToOPCUA ?element ?target
    //if XML element has attributes
    MapUAVariables ?element ?target
    //references to necessary elements(variables,objects..)
    MapReferences ?element (?listOfRefs,?uaObject)
```

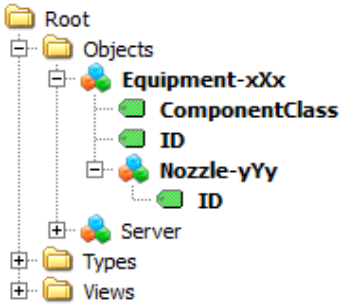
Ohjelma 5.5 *STL-sääntö, jota sovelletaan ohjelmaan 5.4.*

5.5.3 Suunnittelutiedon esittäminen ja mallimuunnosrajapinta

Kuvassa 5.15 on Simupedia-sovellus verkkosivulle, jossa voidaan esittää suunnitteluohjelmistosta tuodun Proteus-skeeman (versio 3.6.0) mukaisen XML-dokumentin graafinen- ja tietosisältö. Sovelluksen vasemmassa yläkulmassa on juurielementti, joka toimii sovelluksen, ja siten myös verkkosivudokumentin, rakenteellisena perus-



Kuva 5.13 *Esimerkkimuunnoksen tuotoksena syntyneellä NodeSet-tiedostolla alustettu mallinnusprojekti. Kuvakaappaus UaModeler-ohjelmasta [111].*



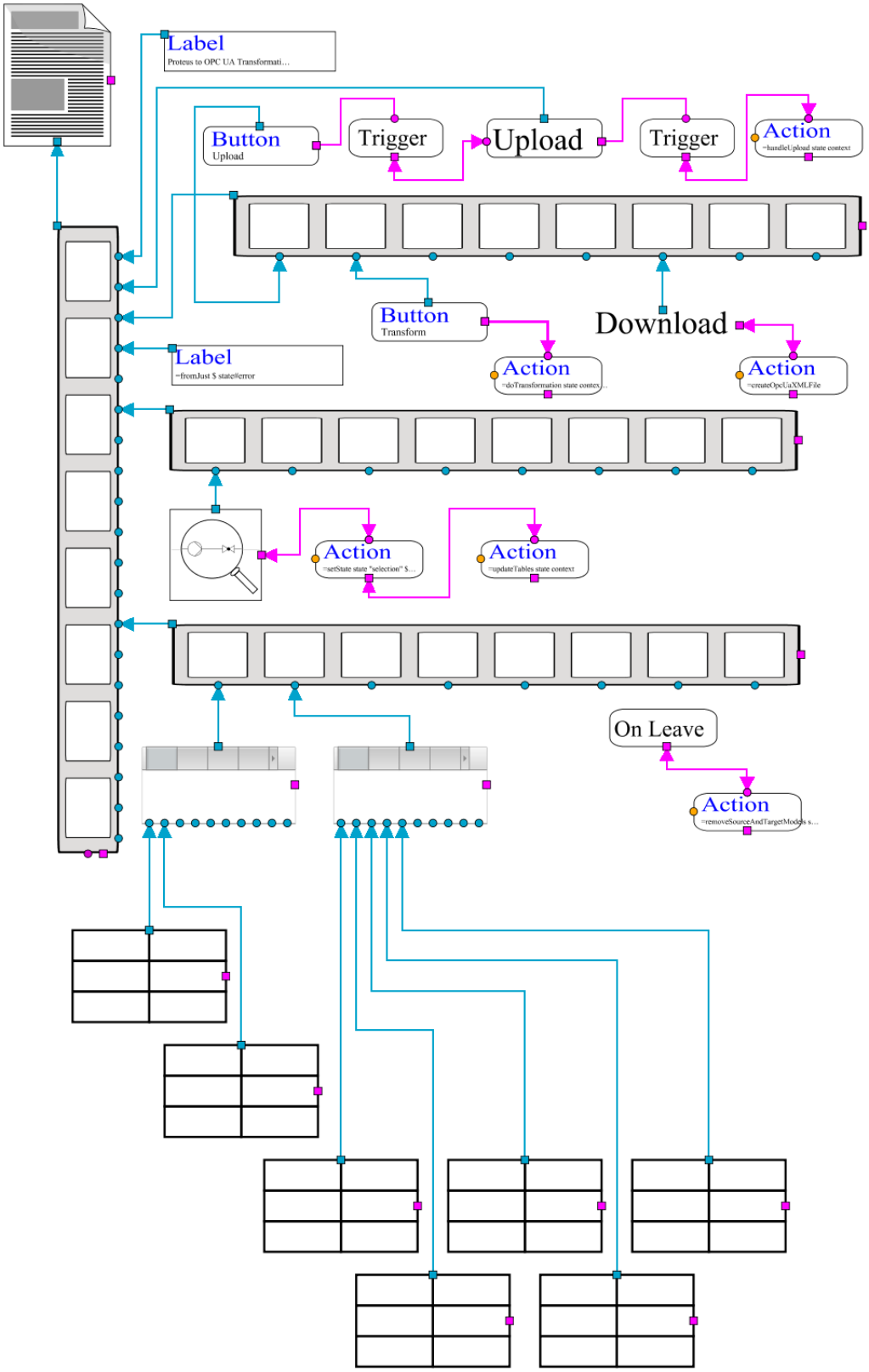
Kuva 5.14 *Esimerkkimuunnoksen tuotoksena syntyneellä NodeSet-tiedostolla on alustettu OPC UA -palvelin, jonka sisältöä selataan OPC UA -asiakasohjelmalla. Kuvakaappaus UaExpret-ohjelmasta [110].*

tana. Dokumentin sisältöä voidaan ryhmitellä pysty- ja vaakasäiliöillä (container), mitkä ovat kuvassa harmaita suorakaiteita, joiden sisällä on useita valkeita ruutuja. Dokumentin rakenne muodostuu sitomalla säiliöt ja widgetit haluttuihin paikkoihin dokumentissa turkooseilla viivoilla. Kuvassa juurielementtiin on liitetty ensiksi pystysäiliö ja siihen edelleen kolme vaakasäiliötä. Säiliöihin on lisäksi liitetty erinäinen määrä erilaisia widgettejä, kuten nimiöitä (label) ja painonappeja (button). Dokumentin rakenne jakaantuu karkeasti kolmeen osaan. Ensimmäisessä ylimmässä kolmanneksessa huolehditaan muunnettavan tiedoston siirtämisestä palvelimelle, tiedoston muuntamisesta ja muunnetun tiedoston lataamisesta. Keskellä on palvelimelle tuodun tiedoston sisällön graafinen esitys, jos tiedosto sisältää Proteus-skeeman

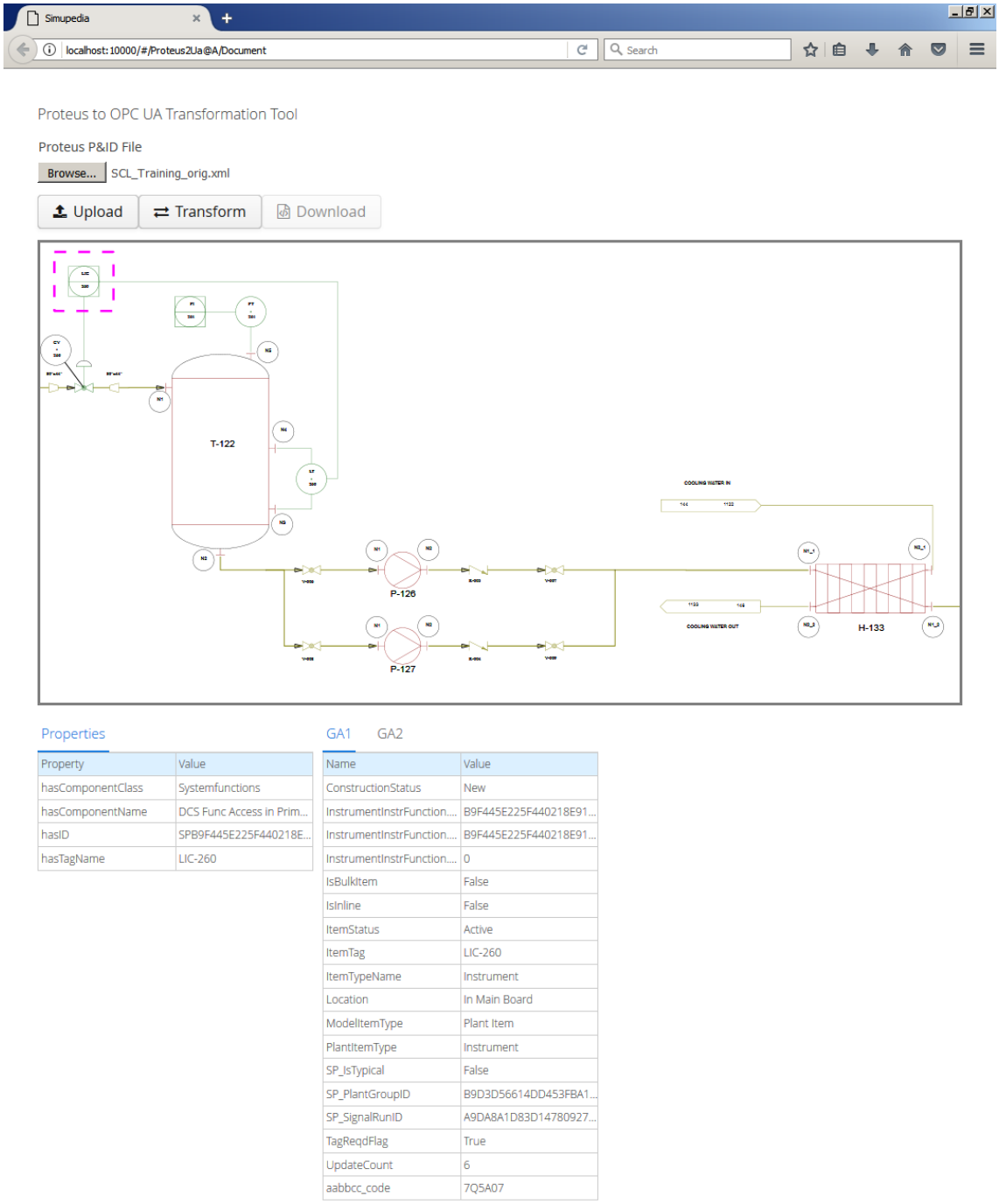
mukaisia graafisia elementtejä. Alimpana taulukoissa esitetään graafinäkymästä valitun PI-komponentin metatietoa.

Simupedia-sovelluksessa on pinkeillä viivoilla yhdistetty widgettejä toisiinsa. Widgetit voivat lähettää ja vastaanottaa tapahtumia (event), mutta vain määrätynlaisia. Esimerkiksi Button-widgetti voi lähettää vain “clicked”-tapahtumia ja Upload-widgetti voi vastaanottaa vain “submit”-tapahtumia. Saadakseen nämä widgetit ymmärtämään toisiaan, sijoitetaan widgettien väliin Trigger-widgetti, joka toimii tapahtumatulkkina. Action-widgettillä voidaan suorittaa SCL-funktioita.

Kuvassa 5.16 on esitetty näkymä kuvan 5.15 Simupedia-sovelluksen tuottamasta verkkosivusta. Palvelimelle on ladattu esimerkkitiedostona PI-kaavio, josta on valittu instrumenttisymboli. Valitun symbolin metatiedot on esitetty taulukoissa graafisen näkymän alapuolella.



Kuva 5.15 *Simupedia-sovellus suunnittelutiedon esittämiselle ja mallimuunnostoteutukselle.*



Property	Value
hasComponentClass	Systemfunctions
hasComponentName	DCS Func Access in Prim...
hasID	SPB9F445E225F440218E...
hasTagName	LIC-260

Name	Value
ConstructionStatus	New
InstrumentInstrFunction....	B9F445E225F440218E91...
InstrumentInstrFunction....	B9F445E225F440218E91...
InstrumentInstrFunction....	0
IsBulkItem	False
IsInline	False
ItemStatus	Active
ItemTag	LIC-260
ItemTypeName	Instrument
Location	In Main Board
ModelItemType	Plant Item
PlantItemType	Instrument
SP_IsTypical	False
SP_PlantGroupID	B9D3D56614DD453FBA1...
SP_SignalRunID	A9DA8A1D83D14780927...
TagReqdFlag	True
UpdateCount	6
aabbcc_code	7Q5A07

Kuva 5.16 *Simupedia-sovelluksella toteutettu verkkosivu Proteus/DEXPI-mallin muuntamiselle sekä graafisentiedon että metatiedon esittämiselle.*

6. TULOKSET JA JOHTOPÄÄTÖKSET

Tässä työssä tavoitteena oli selvittää PI-suunnittelutiedonsiirtoon soveltuvat standardit ja tutkia näistä potentiaalisimman standardin toteutusvalmiutta laitossuunnitteluohjelmistoissa. Jatkoon valikoituneen standardin ja laitteistorajapintastarandardin OPC UA:n metamallien välille määriteltiin kuvaus ja toteutettiin mallimuunnostyökalu käyttäen Simantics-alustaa. Muunnoksen tuotoksena saatavaa OPC UA AddressSpace-mallin XML-serialisaatiota NodeSet-tiedostoa voidaan edelleen käyttää OPC UA -palvelimen alustamiseen ja siten tarjota muun muassa avoin rajapinta suunnittelutietoon tunnetun standardin mukaisesti.

Vertailluista standardeista jatkoon valikoitui standardin ISO 15926 XML-serialisaation Proteus-skeemaan perustuva DEXPI-spesifikaatio. Tähän spesifikaatioon päädyttiin sen valmiudesta mallintaa PI-kaavioiden metatietoa ja graafista tietoa, se perustuu vertailusta toiseen kestäneeseen standardiin, ja ennen kaikkea se on jo otettu osaksi usean ohjelmistotoimittajan laitossuunnitteluohjelmistoa. Lisäksi spesifikaatiota ja sen toteutusta suunnitteluohjelmistoissa kehitetään aktiivisesti, ja sillä on usean omistaja-operaattorin tuki taustalla.

Eri suunnitteluohjelmistojen valmiutta tukea DEXPI-spesifikaatiota tarkasteltiin DEXPI-työryhmän avoimen versionhallinnan verkkosivuston avulla. Suunnitteluohjelmistojen DEXPI-tuen kehitys on jaettu osatavoitteisiin, jotka ovat askelia kohti kokonaisvaltaista PI-suunnittelutiedonsiirtoa. Versionhallintaan vietyjen tiedostojen perusteella voidaan päätellä, mitä toteutuksen osa-alueita suunnitteluohjelmistoissa on aloitettu kehittämään. Suunnitteluohjelmistojen kehitystä seurattiin versionhallintaan vietyjen tiedostojen perusteella puolen vuoden ajan aktiivisuustaulukolla. Se ei ota kantaa toteutuksen valmiuteen. Seurannan perusteella valmiisiin toteutuksiin on toisilla ohjelmistotoimittajilla enemmän matkaa kuin toisilla.

Ohjelmistotoimittajat voivat tarkastella suunnitteluohjelmistoista tuotujen tiedostojen graafisen tiedon mallinnustoteutusta ja semanttisen tiedon vastaavuutta DEXPI-spesifikaatioon DEXPI-työryhmän työkaluilla. Toistaiseksi työryhmällä ei ole työkaluja tai metriikkaa suunnitteluohjelmistoista tuotujen tiedostojen rakenteen vertailuun. Toisaalta työryhmällä on osatavoitekohtaisia verifiointitiedostoja, jotka ku-

vaavat millainen suunnitteluohjelmistoista tuotujen tiedostojen rakenne tulisi olla DEXPI-spesifikaation näkökulmasta. Jotta suunnitteluohjelmistojen valmiutta tukea DEXPI-spesifikaatiota voitaisiin arvioida, määritettiin mittari tiedostojen rakenteen samankaltaisuuden vertaamiseksi. Määritelty metriikka vertailee tiedostojen rakennetta piirrevektorien avulla, jotka muodostuvat tiedostojen juuripoluista. Suunnitteluohjelmistoista tuotujen tiedostojen oletettiin olevan verifioitu ja validoitu Proteus-skeemaa vasten jo suunnitteluohjelmistoissa. Kehitetyn metriikan perustellaa toteutettiin ohjelma, joka vertaili suunnitteluohjelmistoista tuotuja tiedostoja verifiointitiedostoihin automaattisesti. Metriikka ja ohjelma toimivat hyvin työryhmän versionhallinnan esimerkinomaisiin tiedostoihin. Saatu tulos varmistettiin myös silmämääräisesti.

Metriikkaa voisi jatkokehittää määrittelemällä lisää piirevektoreita. Puolirakenteellisen tiedoston juuripolkujen lisäksi voitaisiin vertailla myös alipolkuja ja näin parantaa rakenteiden vertailun luotettavuutta. Piirrevektoreihin voitaisiin lisätä tiedoston elementejä yksilöivästä tiedosta tai määrittelemällä metriikka uusiksi semanttiikkaan perustuen.

Proteus/DEXPI-tietomallin ja OPC UA -tietomallin välille määritettiin kuvaus, jonka perusteella toteutettiin muunnostyökalu web-käyttöliittymällä Proteus/DEXPI-mallin muuntamiseksi OPC UA -malliksi. Kuvauksessa määritettiin, mitkä Proteus/DEXPI-tietomallin elementit, relaatiot ja tietotyypit vastaavat OPC UA -tietomallissa määriteltyjä. Kuvauksessa käytettiin molempien tietomallien standardeja/spesifikaatioita ja XML-skeemoja. Varsinainen mallimuunnos toteutettiin hyödyntäen Simantics-alustan semanttista tietokantaa ja STL-muunnoskieltä. Web-käyttöliittymä suunniteltiin Simupedia-sovelluksella. Mallimuunnoksen toteutus mahdollisuus web-käyttöliittymän kautta palvelee ajatusta pilvipalvelujen taustalla.

Aikaansaatu mallimuunnostoteutus vastaa muunnokselle asetettuja tavoitteita. Muunnos lähdemallista kohdemalliksi voidaan suorittaa automaattisesti siten, että muunnostuotos eli kohdemalli noudattaa tietomallien välille määriteltyä kuvausta. Muunnoksen helpottaa suunnittelutiedon siirtoa laitossuunnitteluprojektin suunnitteluosapuolien kesken ja suunnittelutiedon käyttöä käytönaikaisissa prosesseissa. Näin ollen suunnittelutieto voidaan tarjota edelleen palveluna OPC UA -palvelimen avulla projektin muille osapuolille tai helpottaa esimerkiksi automaatioinsinöörin työtaakkaa laitteistorajapintatoteutuksissa, jos laitoksessa päädytään käyttämään OPC UA:ta hyödyntävää tekniikkaa.

LÄHTEET

[1] L. Abele and S. Grimm, "Knowledge-based integration of industrial plant models," in *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE.* IEEE, 2013, pp. 4392–4397.

[2] S. Abiteboul, "Querying semi-structured data," *Database Theory—ICDT'97*, pp. 1–18, 1997.

[3] M. Amrani, L. Lucio, G. Selim, B. Combemale, J. Dingel, H. Vangheluwe, Y. Le Traon, and J. R. Cordy, "A Tridimensional Approach for Studying the Formal Verification of Model Transformations." IEEE, 2012, pp. 921–928.

[4] AutomationML, *Whitepaper AutomationML Part 2–Role class libraries*, 2014.

[5] ——, *Whitepaper AutomationML and eCl@ss integration*, 2016.

[6] ——, *Whitepaper AutomationML Part 1–Architecture and General Requirements*, 2016.

[7] AutomationML e.V., "Members," Saatavissa (viitattu 21.04.2017): https://www.automationml.org/o.red.c/mitglieder.html.

[8] ——, "Tools," Saatavissa (viitattu: 17.5.2017): https://www.automationml.org/o.red.c/tools.html.

[9] AutomationML e.V. and OPC Foundation, *OPC UA Information Model for AutomationML*, 2016.

[10] ——, *OPC UA Information Model for AutomationML*, 2016.

[11] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui, "First experiments with the ATL model transformation language: Transforming XSLT into XQuery," in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, vol. 37, 2003.

[12] P. G. Bigvand, R. Drath, A. Scholz, and A. Schüller, "Agile standardization by means of pce requests," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, 2015, pp. 1–8.

[13] T. Bray, J. Paoli, C. Sperberg-McQueen, Y. Mailer, and F. Yergeau, "Extensible Markup Language (XML) 1.0 5th Edition, W3C recommendation, November 2008."

[14] Chair of Process Control Engineering, *PandIX*, RWTH Aachen University, 2015, Saatavissa: http://www.plt.rwth-aachen.de/cms/PLT/Forschung/Projekte2/~ejwz/PandIX/lidx/1/.

[15] G. Costa, G. Manco, R. Ortale, and A. Tagarelli, "A tree-based approach to clustering XML documents by structure," in *PKDD*, vol. 2004. Springer, 2004, pp. 137–148.

[16] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.

[17] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis, "Clustering XML documents using structural summaries," in *International Conference on Extending Database Technology*. Springer, 2004, pp. 547–556.

[18] DEXPI, "Data Exchange in the Process Industry," Saatavissa (viitattu 4.5.2017): http://www.dexpi.org.

[19] ——, "Members," Saatavissa (viitattu 4.5.2017): http://www.dexpi.org/?page_id=4.

[20] ——, "Specifications & Services," Saatavissa (viitattu 18.5.2017): http://www.dexpi.org/?page_id=26.

[21] ——, "TestCases/tests at master · DEXPI/TestCases · GitHub," Saatavissa (viitattu 4.8.2017): https://github.com/DEXPI/TestCases/tree/master/tests.

[22] A. Doucet and H. Ahonen-Myka, "Naive clustering of a large XML document collection," in *INEX 2002 Workshop Proceedings*, 2002, p. 84.

[23] Eclipse, "Eclipse home," Saatavissa (viitattu 30.5.2017): https://eclipse.org/.

[24] ——, "Eclipse Public License - v 1.0," Saatavissa (viitattu 29.5.2017): http://www.eclipse.org/legal/epl-v10.html.

[25] Fiatech, "Fiatech," Saatavissa (viitattu 6.7.2017): http://fiatech.org/.

[26] ——, "ISO 15926 Information Models and Proteus Mappings (IIMM)," Saatavissa (viitattu 6.7.2017): http://fiatech.org/information-management/projects/1161-iso-15926-information-models-and-proteus-mappings-iimm.

[27] ——, "ISO15926 P&ID Profile Schema Specification – Rev 1.6," Saatavissa (viitattu 24.7.2017): http://fiatech.org/images/stories/techprojects/ISO%2015926%20PID%20Profile%20Schema%20specification_draft.pdf.

[28] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese, "Fast detection of XML structural similarity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 2, pp. 160–175, 2005.

[29] T. Frühwirth, "Theory and practice of constraint handling rules," *The Journal of Logic Programming*, vol. 37, no. 1, pp. 95 – 138, 1998.

[30] M. Gayer, J. Kortelainen, and T. Karhela, "CFD modelling as an integrated part of multi-level simulation of process plants: semantic modelling approach," in *Proceedings of the 2010 summer computer simulation conference*. Society for Computer Simulation International, 2010, pp. 219–227.

[31] G. Guerrini, M. Mesiti, and I. Sanz, "An overview of similarity measures for clustering XML documents," *Information Systems*, 2006.

[32] J. A. Gulla, S. L. Tomassen, and D. Strasunskas, "Semantic Interoperability in the Norwegian Petroleum Industry," in *ISTA*, 2006, pp. 81–93.

[33] T. Hannelius, M. Salmenpera, and S. Kuikka, "Roadmap to adopting OPC UA," in *2008 6th IEEE International Conference on Industrial Informatics*, 2008, pp. 756–761.

[34] Hannu, Niemistö, "Introduction to scl," Saatavissa (viitattu 1.6.2017): http://www.simantics.org/~niemisto/Introduction%20to%20SCL.pptx.

[35] ——, "Simantics Transformation Language – Presentation," Saatavissa (viitattu 19.7.2017): http://www.simantics.org/~niemisto/Transformations_2015_01_23.pdf.

[36] ——, "Simantics Transformation Language – Spesification," Saatavissa (viitattu 19.7.2017): http://www.simantics.org/~niemisto/Transformations.pdf.

[37] Haskell, "Haskell Language," Saatavissa (viitattu 1.6.2017): https://www.haskell.org/.

[38] D. Hästbacka and T. Mätäsniemi, "Unifying process design with automation and control application development-an approach based on information integration and model-driven methods," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 1227–1232, 2009.

[39] R. Henssen and M. Schleipen, "Interoperability between OPC UA and AutomationML," *Procedia CIRP*, vol. 25, pp. 297–304, 2014.

[40] T. Holm, L. Christiansen, M. Göring, T. Jäger, and A. Fay, "ISO 15926 vs. IEC 62424; comparison of plant structure modeling concepts," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*.

[41] J. Hughes, "Why functional programming matters," *The computer journal*, vol. 32, no. 2, pp. 98–107, 1989.

[42] International Electrotechnical Commission *et al.*, "IEC 62424," *Representation of process control engineering-Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*, 2016.

[43] International Organization for Standardization, "ISO 15926-1:2004," *Industrial automation systems and integration–Integration of life-cycle data for process plants including oil and gas production facilities – Part 1: Overview and fundamental principles*, 2004.

[44] International Organization of Standardization , "ISO 13584-1:2001," *Industrial automation systems and integration – Parts library – Part 1: Overview and fundamental principles*, 2001.

[45] P. Jaccard, "The distribution of the flora in the alpine zone." *New phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[46] T. Karhela, A. Villberg, and H. Niemistö, "Open ontology-based integration platform for modeling and simulation in engineering," *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 3, no. 02, p. 1250004, 2012.

[47] A. G. Kleppe, J. B. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[48] J. W. Klüwer, M. G. Skjæveland, and M. Valen-Sendstad, "ISO 15926 templates and the Semantic Web," in *Position paper for W3C Workshop on Semantic Web in Energy Industries; Part I: Oil and Gas*, 2008.

[49] T. Kühne, "Matters of (meta-) modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[50] I. Kurtev, J. Bézivin, and M. Akşit, "Technological spaces: An initial appraisal," 2002.

[51] S. Kutty, T. Tran, R. Nayak, and Y. Li, "Clustering XML documents using frequent subtrees," in *International Workshop of the Initiative for the Evaluation of XML Retrieval*. Springer, 2008, pp. 436–445.

[52] D. Leal, "ISO 15926 "Life Cycle Data for Process Plant": an Overview," *Oil & gas science and technology*, vol. 60, no. 4, pp. 629–637, 2005.

[53] T. Lehtonen, "Ontology-based diagram methods in process modelling and simulation," Master's thesis, Helsinki University of Technology, 2007.

[54] S.-H. Leitner and W. Mahnke, "OPC UA–service-oriented architecture for industrial applications," *ABB Corporate Research Center*, 2006.

[55] W. Lian, N. Mamoulis, S.-M. Yiu *et al.*, "An efficient and scalable algorithm for clustering XML documents by structure," *IEEE transactions on Knowledge and Data Engineering*, vol. 16, no. 1, pp. 82–96, 2004.

[56] J. Long, D. G. Schwartz, and S. Stoecklin, "An XML Distance Measure," in *DMIN*, 2005, pp. 119–125.

[57] A. Lüder, L. Hundt, and A. Keibel, "Description of manufacturing processes using AutomationML," in *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*, 2010, pp. 1–8.

[58] W. Mahnke, A. Gössling, M. Graube, and L. Urbas, "Information modeling for middleware in automation," in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on.* IEEE, 2011, pp. 1–7.

[59] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture.* Springer Science & Business Media, 2009.

[60] J. McKay and P. Marshall, "The dual imperatives of action research," *Information Technology & People*, vol. 14, no. 1, pp. 46–59, 2001.

[61] T. Mens and P. Van Gorp, "A taxonomy of model transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, 2006.

[62] T. Miettinen, J. Salmi, K. Gupta, J. Koskela, J. Kauttio, T. Karhela, and S. Ruutu, "Applying Modelica Tools to System Dynamics Based Learning Games," *Modelling and Simulation in Engineering*, 2016.

[63] Modelio, "Modelio Open Source - UML and BPMN free modeling tool," Saatavissa (viitattu 14.8.2017): https://www.modelio.org/.

[64] H. Niemistö and A. Villberg, *Layer0 specification*, 2011.

[65] A. Nierman and H. Jagadish, "Evaluating Structural Similarity in XML Documents," in *webdb*, vol. 2, 2002, pp. 61–66.

[66] Object Management Group, "Home Page," Saatavissa (viitattu 26.6.2017): http://www.omg.org/.

[67] ——, "Meta Objet Facility (MOF) 2.0 Query/View/Transformation, v1.3," Saatavissa (viitattu 28.6.2017): http://www.omg.org/spec/QVT/1.3/.

[68] ——, "Object Constraint Language. Version 2.4," Saatavissa (viitattu 28.6.2017): http://www.omg.org/spec/OCL/2.4/.

[69] ——, "OMG Meta Object Facility (MOF) Core Specification. Version 2.5.1," Saatavissa (viitattu 28.6.2017): http://www.omg.org/spec/MOF/2.5.1/.

[70] ——, *OMG Unified Modeling Language^{TM}, Infrastructure*, 2011, version 2.4.1.

[71] ——, *OMG Unified Modeling Language^{TM}, Infrastructure*, 2015, version 2.5.

[72] OPC Foundation, "Home Page - OPC Foundation," Saatavissa (viitattu 26.6.2017): https://opcfoundation.org/.

[73] ——, *OPC Unified Architecture, Part 1: Overview and Concepts*, 2015, Industry Standard Specification, Release 1.03.

[74] ——, *OPC Unified Architecture, Part 3: Address Space Model*, 2015, Industry Standard Specification, Release 1.03.

[75] ——, *OPC Unified Architecture, Part 5: Information Model*, 2015, Industry Standard Specification, Release 1.03.

[76] N. Papakonstantinou and T. Karhela, "Towards a multidisciplinary platform based on OPC UA for accessing plant data: Open P & ID data access," 2017, Saatavissa (viitattu 20.6.2017): https://www.automaatioseura.fi/site/assets/files/1599/au-22_paper_37.pdf.

[77] M. Piernik, D. Brzezinski, and T. Morzy, "Clustering XML documents by patterns," *Knowledge and Information Systems*, vol. 46, no. 1, pp. 185–212, 2016.

[78] POSC Caesar Accociation, "POSC Caesar - Trac," Saatavissa (viitattu 6.7.2017): https://www.posccaesar.org/.

[79] Prosys OPC Ltd, "Prosys OPC," Saatavissa (viitattu 23.8.2017): https://www.prosysopc.com/.

[80] PSK Standardisointi, "PSK lyhyesti," Saatavissa (viitattu 20.2.2017): http://www.psk-standardisointi.fi/Alasivut/PSK_lyhyesti.htm.

[81] ——, "PSK-standardisointi," Saatavissa (viitattu 26.6.2017): http://www.psk-standardisointi.f.

[82] ——, *Elektronisen suunnitteluaineiston siirto. Numeeristen, aakkosnumeeristen ja aakkosellisten tietoelementtien esitysmuodot. Peruskäsitteet ja esitystavat*, 1997.

[83] ——, *Teollisuuden kone- ja laitehankinnat. Elektronisen aineiston siirto. Kone- ja laitetiedot*, 1997.

[84] ——, *XML-tiedonsiirto. Kantaluokat ja tiedonsiirtomalli*, 2006.

[85] ——, *Tiedonsiirto. Laitteiden luokat ja alaluokat*, 2010.

[86] ——, *Tiedonsiirto. Järjestelmien attribuuttien vastaavuus. Automaation kenttälaitteiden tiedot*, 2015.

[87] ——, *Prosessiteollisuuden virtaus- ja PI-kaavioiden symbolit*, 2016.

[88] D. Rafiei, D. L. Moise, and D. Sun, "Finding syntactic similarities between xml documents," in *Database and Expert Systems Applications, 2006. DEXA'06. 17th International Workshop on.* IEEE, 2006, pp. 512–516.

[89] F. Raiser, *Graph Transformation Systems in CHR.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 240–254.

[90] M. Saeki and H. Kaiya, "On relationships among models, meta models and ontologies," in *Proceedings of the Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM 2006)*, 2006.

[91] M. Schleipen, O. Sauer, and J. Wang, *Semantic integration by means of a graphical OPC Unified Architecture (OPC-UA) information model designer for Manufacturing Execution Systems*, 2010.

[92] N. Schmidt and A. Lüder, "AutomationML in a Nutshell AutomationML eV Office," Saatavissa: https://www.automationml.org/o.red/uploads/dateien/1447420977-AutomationML%20in%20a%20Nutshell_151104.pdf.

[93] T. Schrijvers, *Analyses, Optimizations and Extensions of Constraint Handling Rules: Ph.D. Summary.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 435–436.

[94] A. Schüller and U. Epple, "PandIX; exchanging p amp;i diagram model data," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, 2012, pp. 1–8.

[95] M. Seidl, M. Scholz, C. Huemer, and G. Kappel, *UML@ classroom: An introduction to object-oriented modeling*. Springer, 2015.

[96] S. Sharma and M. Singh, "Generalized similarity measure for categorical data clustering," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016, pp. 765–769.

[97] P. Siltanen and A. Pärnänen, "Comparison of data models for plant lifecycle information management." Intuition Network of Excellence, 2006.

[98] Simantics, "CHRGuide," Saatavissa (viitattu 19.7.2017): http://www.simantics.org/~niemisto/CHRGuide.html.

[99] ——, "Engineering rulez - a new r&d project based on simantics technology has started," Saatavissa (viitattu 2017-06-22): https://www.simantics.org/news/engineering-rulez-new-rd-project-based-simantics-technology-has-started.

[100] ——, "Open operating system for modeling and simulation," Saatavissa (viitattu 2017-05-27): https://www.simantics.org/.

[101] ——, "Simantics End User Documentation," Saatavissa (viitattu 2017-06-01): https://www.simantics.org/end_user_wiki/index.php/Main_Page.

[102] ——, "XML Schema Conversion - Developer Documents," Saatavissa (viitattu 2017-08-02): http://dev.simantics.org/index.php/XML_Schema_Conversion.

[103] Simupedia, "Harnessing Collective Intelligence," Saatavissa (viitattu 2017-06-01): http://www.simupedia.com/.

[104] P. Stevens, "Bidirectional model transformations in qvt: Semantic issues and open questions," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2007, pp. 1–15.

[105] M. Theißen and M. Wiedau, "P&ID Specification," *Data Exchange in Process Industry (DEXPI)*, 2017, Saatavissa: https://github.com/DEXPI/DEXPIdev/tree/master/specification.

[106] H. S. Thompson, N. Mendelsohn, D. Beech, and M. Maloney, "W3C XML schema definition language (XSD) 1.1 part 1: Structures," *The World Wide Web Consortium (W3C), W3C Working Draft Dec*, vol. 3, 2009.

[107] THTH, "Teollisuuden hajautetun tiedonhallinnan yhdistys thth ry," Saatavissa (viitattu 29.5.2017): http://www.ththry.org/.

[108] T. Tran, R. Nayak, and P. Bruza, "Combining structure and content similarities for xml document clustering," in *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*. Australian Computer Society, Inc., 2008, pp. 219–225.

[109] E. Ulrich, R. Markus, and D. Oliver, *PandIX – PPE-Musterbibliothek*, 2010, version 5.01.

[110] Unified Automation, "UaExpert - Unified Automation," Saatavissa (viitattu 2017-08-23): https://www.unified-automation.com/products/development-tools/uaexpert.html.

[111] ——, "UaModeler "Turns Design into Code"-Unified Automation," Saatavissa (viitattu 2017-06-06): https://www.unified-automation.com/products/development-tools/uamodeler.html.

[112] Vaadin, "User interface components for web apps," Saatavissa (viitattu 1.6.2017): https://vaadin.com/home.

[113] A. Villberg, "Desing challenges of an ontology-based modelling and simulation environment," Master's thesis, Helsinki University of Technology, 2007.

[114] VTT, "Suomalaisen laitossuunnittelun automaatioaste ja laatu korkeammaksi," Saatavissa (viitattu 27.6.2017): http://www.vtt.fi/medialle/uutiset/suomalaisen-laitossuunnittelun-automaatioaste-ja-laatu-korkeammaksi.

[115] World Wide Web Consortium, "Transformation," Saatavissa (viitattu 27.6.2017): https://www.w3.org/standards/xml/transformation.

[116] ——, "World Wide Web Consortium," Saatavissa (viitattu 26.6.2017): https://www.w3.org/.

[117] ——, "XML Schema Part 2: Datatypes Second Edition," Saatavissa (viitattu 11.8.2017): https://www.w3.org/TR/xmlschema-2/, 2004.

[118] ——, "XSL Transformations (XSLT) Version 3.0," Saatavissa (viitattu 27.6.2017): https://www.w3.org/TR/xslt-30/, 2012.

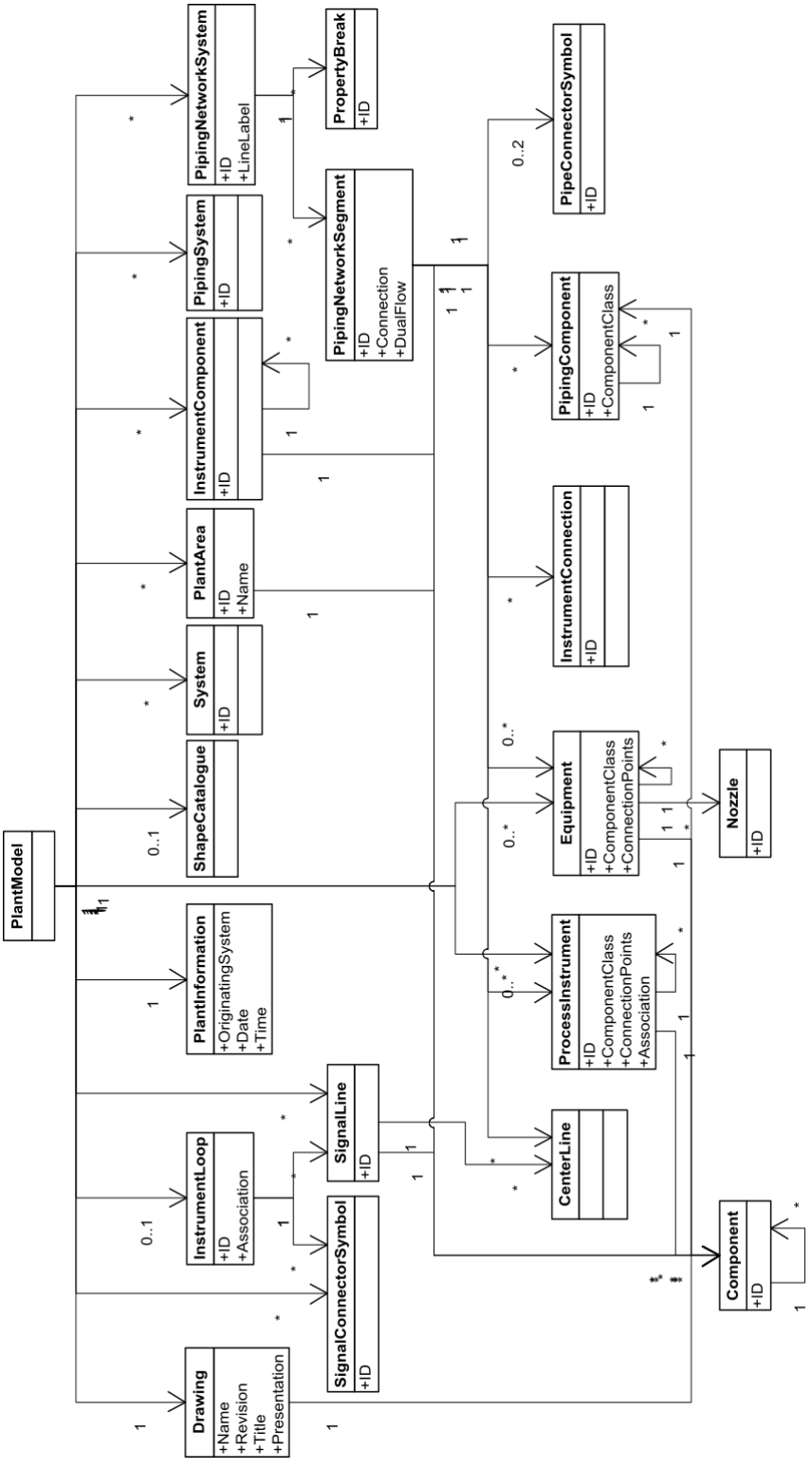
[119] ——, "OWL Web Ontology Language Overview," Saatavissa (viitattu 28.6.2017): https://www.w3.org/TR/owl-features/, 2014.

[120] ——, "RDF 1.1 Concepts and Abstract Syntax," Saatavissa (viitattu 28.6.2017): https://www.w3.org/TR/rdf11-concepts/, 2014.

[121] ——, "SPARQL 1.1 Overview," Saatavissa (viitattu: 29.6.2017): https://www.w3.org/TR/sparql11-overview/, 2014.

[122] J.-s. Yuan, X.-y. Li, and L.-n. Ma, "An improved xml document clustering using path feature," in *Fuzzy Systems and Knowledge Discovery, 2008. FSKD'08. Fifth International Conference on*, vol. 2. IEEE, 2008, pp. 400–404.

LIITE A. YLEISKUVA PROTEUS-SKEEMASTA



Kuva A.1 *Luokkakaavio Proteus-skeeman (versio 3.6.0) oleellisimmista rakenteellisista elementeistä. [27]*

LIITE B. OHJELMA XML-DOKUMENTTIEN RAKENTEIDEN SAMANKALTAISUUDEN VERTAILUUN

```
"""
/----------------------------------------------------------------------/
/----------------------------------------------------------------------/
|    /_\  _   _| |_| |__   ___  _ __ _
|   //_\\| | | | __| '_ \ / _ \| '__(_)
|  /  _  \ |_| | |_| | | | (_) | |   _
|  \_/ \_/\__,_|\__|_| |_|\___/|_|  (_)
|
|    \ \  __ _ _ __  _ __   ___         \ \  ___   ___  _ __   __ _ ___
|     \ \/ _` | '_ \| '_ \ / _ \_____    \ \/ _ \ / _ \| '_ \ / _` / __|
|  /\_/ / (_| | | | | | | |  __/_____/\_/ / (_) | (_) | | | | (_| \__ \
|  \___/ \__,_|_| |_|_| |_|\___|     \___/ \___/ \___/|_| |_|\__,_|___/
|                          _       _
|    /\/\   __ _ _ __ | |_ ___(_)_ __   ___ _ __
|   /    \ / _` | '_ \| __/ __| | '_ \ / _ \ '_ \
|  / /\/\ \ (_| | | | | |_\__ \ | | | |  __/ | | |
|  \/    \/\__,_|_| |_|\__|___/_|_| |_|\___|_| |_|
/----------------------------------------------------------------------/
/----------------------------------------------------------------------/
"""
import glob,os #modulit hakemistojen ja tiedostojen käsittelyyn
import lxml.etree as ET  #moduli XML-prosessointiin
import re #regex-moduli

'''Hakee rekursiivisesti annettun XML-elementin
('solmun') vanhemman, isovanhemman jne. muodostaen
juuripolun.

**Parameters:
-solmu (lxml.etree._Element)
-lista ([lxml.etree._Element])
**Return:
ei varsinaisesti palauta mitään; lisää juuripolun
listana funktiolle parametrina annetulle listalle
'lista'
'''
def getRootPath(solmu,lista):
    if solmu.getparent() is None:
        None
    else:
```

```
            lista.append(solmu.getparent())
42          getRootPath(solmu.getparent(),lista)

44  '''Funktio hakee kaikki XML-tiedoston "lehdet", eli elementit
    joilla ei ole lapsielementtejä, ja palauttaa ne listana.
46
    **Parameters:
48  -elementTree (lxml.etree._ElementTree)
    **Returns:
50  -leafNodes ([lxml.etree._Element])
    '''
52  def getLeafNodes(elementTree):
        leafNodes = []
54      for node in elementTree.iter('*'):
            if 0 ==  len(node):
56              leafNodes.append(node)
        leafNodes = filterAttributeNodes(leafNodes)
58      return leafNodes

60  '''Funktio poistaa parametrinaan saadusta listasta ei-halutut
    elementit ja palauttaa sen.
62
    **Parameters:
64  -leafList ([lxml.etree._Element])
    **Retruns:
66  -filteredList ([lxml.etree._Element])
    '''
68  def filterAttributeNodes(leafList):
        dexpiAttributeElementList = []
70      filteredList = []
        for leaf in leafList:
72          if leaf.tag == "GenericAttribute":
                parent = leaf.getparent()
74              if parent.get("Set") == "DexpiAttributes" and \
                not parent in dexpiAttributeElementList:
76                  dexpiAttributeElementList.append(parent)
                    filteredList.append(leaf)
78              else:
                    None
80          else:
                filteredList.append(leaf)
82      return filteredList

84  '''Funktio etsii XML-dokumentista juuripolut.

86  **Paramenters:
    -elementTree (lxml.etree._ElementTree)
```

```
88 **Returns:
   -rootPaths ([[lxml.etree._Element]])
90 '''
   def getRootPaths(elementTree):
92     rootPaths = []
       leafNodes = getLeafNodes(elementTree)
94     while len(rootPaths) <= len(leafNodes)-1:
           rootPaths.append([])
96     for node in leafNodes:
           if node.getparent() is None:
98             None
           else:
100            rootPaths[leafNodes.index(node)].append(node)
               getRootPath(node,rootPaths[leafNodes.index(node)])
102    return rootPaths

104 '''Funktio vertaa kahden XML-tiedoston juuripolkuja keskenään. Jos
   ensimmäisenä parametrina annetun listan juuripolku löytyy
106 toisena parametrina annetusta listasta, lisätään polun
   indeksi listaan 'pathIndexMatch'.
108
   **Paramenters:
110 -ver ([[lxml.etree._Element]])
   -comp ([[lxml.etree._Element]])
112 **Returns:
   -pathIndexMatch ([int])
114 '''
   def matchSearch(ver,comp):
116    addressBucket = []
       pathIndexMatch = []
118    for rootPathVer in ver:
           pathVer = [element.tag for element in rootPathVer]
120        for rootPathVend in comp:
               pathVend = [element.tag for element in rootPathVend]
122            if pathVer == pathVend and \
               len([item for item in addressBucket if
124                 hex(id(rootPathVer)) in item])==0 and \
               len([item for item in addressBucket if
126                 hex(id(rootPathVend)) in item])==0:
                   pathIndexMatch.append(ver.index(rootPathVer))
128                verAdd = hex(id(rootPathVer))
                   vendAdd = hex(id(rootPathVend))
130                addressBucket.append(tuple((verAdd,vendAdd)))
       return pathIndexMatch
132
   if __name__ == '__main__':
134
```

```
path=('C:/data/Desktop/xml vert/')
verRootPaths = [[]]
vendRootPaths =[[]]

r = re.compile('^[A-Z][0-9][0-9]')
subFolders = filter(r.match , [x[1] for x in os.walk(path)][0])
for sub in subFolders:
    print(sub)
    os.chdir(os.path.join(path+sub))
    for verificatorFile in glob.glob('*VER*.xml'):
        verEt = ET.parse(verificatorFile)
        verRootPaths = getRootPaths(verEt)
        print('Number of rootPaths: {0}'.format(len(verRootPaths)))
        for vendorFile in glob.glob('*-*[!V][!E][!R].*.xml'):
            vendEt = ET.parse(vendorFile)
            vendRootPaths =  getRootPaths(vendEt)
            equalPaths = matchSearch(verRootPaths, vendRootPaths)
            print('{0} - {1}'.format(vendorFile, len(equalPaths)))
```

Ohjelma B.1 *Python-koodi Proteus/DEXPI-tiedostojen juuripolkujen vertailuun referenssitiedostoa vasten.*

LIITE C. OPC UA NODESET -ESIMERKKI

```
<?xml version="1.0" encoding="UTF-8"?>
<UANodeSet
 xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd">
<UAObject NodeId="ns=1;i=100" BrowseName="dippa_1:Equipment-xXx">
  <DisplayName>Equipment-xXx</DisplayName>
  <References>
    <Reference IsForward="false" ReferenceType="Organizes">i=85</Reference>
    <Reference ReferenceType="HasComponent">ns=1;i=101</Reference>
    <Reference ReferenceType="HasComponent">ns=1;i=111</Reference>
    <Reference ReferenceType="HasComponent">ns=1;i=112</Reference>
  </References>
</UAObject>
<UAObject NodeId="ns=1;i=101" BrowseName="dippa_1:Nozzle-yYy"
 ParentNodeId="ns=1;i=100">
  <DisplayName>Nozzle-yYy</DisplayName>
  <References>
    <Reference IsForward="false" ReferenceType="HasComponent">
     ns=1;i=100</Reference>
    <Reference ReferenceType="HasComponent">ns=1;i=113</Reference>
  </References>
</UAObject>
<UAVariable NodeId="ns=1;i=111" BrowseName="dippa_1:ComponentClass"
 ParentNodeId="ns=1;i=100" DataType="String"
 AccessLevel="3" UserAccessLevel="3">
 <DisplayName>ComponentClass</DisplayName>
  <References>
    <Reference IsForward="false" ReferenceType="HasComponent">
     ns=1;i=100</Reference>
  </References>
  <Value>
    <uax:String>Tank</uax:String>
  </Value>
</UAVariable>
<UAVariable NodeId="ns=1;i=112" BrowseName="dippa_1:ID"
 ParentNodeId="ns=1;i=100" DataType="String"
 AccessLevel="3" UserAccessLevel="3">
 <DisplayName>ID</DisplayName>
  <References>
    <Reference IsForward="false" ReferenceType="HasComponent">
     ns=1;i=100</Reference>
  </References>
```

```
    <Value>
46    <uax:String>xXx</uax:String>
    </Value>
48 </UAVariable>
  <UAVariable NodeId="ns=1;i=113" BrowseName="dippa_1:ID"
50  ParentNodeId="ns=1;i=101" DataType="String"
    AccessLevel="3" UserAccessLevel="3">
52  <DisplayName>ID</DisplayName>
    <References>
54    <Reference IsForward="false" ReferenceType="HasComponent">
       ns=1;i=101</Reference>
56  </References>
    <Value>
58    <uax:String>yYy</uax:String>
    </Value>
60 </UAVariable>
  </UANodeSet>
62 </UANodeSet>
```

Ohjelma C.1 *Kuvan 5.10 muunnosesimerkki kokonaisuudessaan OPC UA NodeSet:nä.*

LIITE D. MUUNNOSPROSESSIN TUOTOS

```
<?xml version="1.0" encoding="UTF-8"?>
<UANodeSet xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<UAObject NodeId="ns=1;i=60166390" BrowseName="dippa_1:Equipment-xXx">
  <DisplayName>Equipment-xXx</DisplayName>
  <References>
    <Reference IsForward="false" ReferenceType="Organizes">
     i=85</Reference>
    <Reference ReferenceType="HasComponent">
     ns=1;i=60166391</Reference>
    <Reference ReferenceType="HasComponent">
     ns=1;i=60166394</Reference>
    <Reference ReferenceType="HasComponent">
     ns=1;i=60166393</Reference>
  </References>
</UAObject>
<UAObject BrowseName="dippa_1:Nozzle-yYy" NodeId="ns=1;i=60166391"
 ParentNodeId="ns=1;i=60166390">
  <DisplayName>Nozzle-yYy</DisplayName>
  <References>
    <Reference IsForward="false" ReferenceType="HasComponent">
     ns=1;i=60166390</Reference>
    <Reference ReferenceType="HasComponent">
     ns=1;i=60166392</Reference>
  </References>
</UAObject>
<UAVariable BrowseName="dippa_1:ComponentClass" NodeId="ns=1;i=60166394"
 ParentNodeId="ns=1;i=60166390" DataType="String"
 AccessLevel="3" UserAccessLevel="3">
  <Value>
    <String xmlns="http://opcfoundation.org/UA/2008/02/Types.xsd">
     Tank</String>
  </Value>
  <DisplayName>ComponentClass</DisplayName>
  <References>
    <Reference ReferenceType="HasTypeDefinition">
     i=63</Reference>
    <Reference IsForward="false" ReferenceType="HasComponent">
     ns=1;i=60166390</Reference>
  </References>
</UAVariable>
<UAVariable BrowseName="dippa_1:ID" NodeId="ns=1;i=60166392"
 ParentNodeId="ns=1;i=60166391" DataType="String"
```

```
   AccessLevel="3" UserAccessLevel="3">
46  <Value>
      <String xmlns="http://opcfoundation.org/UA/2008/02/Types.xsd">
48     yYy</String>
    </Value>
50  <DisplayName>ID</DisplayName>
    <References>
52    <Reference ReferenceType="HasTypeDefinition">
       i=63</Reference>
54    <Reference IsForward="false" ReferenceType="HasComponent">
       ns=1;i=60166391</Reference>
56  </References>
  </UAVariable>
58 <UAVariable BrowseName="dippa_1:ID" NodeId="ns=1;i=60166393"
   ParentNodeId="ns=1;i=60166390" DataType="String"
60  AccessLevel="3" UserAccessLevel="3">
    <Value>
62    <String xmlns="http://opcfoundation.org/UA/2008/02/Types.xsd">
       xXx</String>
64  </Value>
    <DisplayName>ID</DisplayName>
66  <References>
      <Reference ReferenceType="HasTypeDefinition">
68     i=63</Reference>
      <Reference IsForward="false" ReferenceType="HasComponent">
70     ns=1;i=60166390</Reference>
    </References>
72 </UAVariable>
  </UANodeSet>
```

Ohjelma D.1 *Ohjelma 5.4 muunnettu OPC UA NodeSet:ksi.*

LIITE E. XSLT-DOKUMENTTI MUUNNOKSEN JÄLKIPROSESSOINTIIN

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:uax ="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
    version="1.0">
    <xsl:strip-space elements="*"/>
    <xsl:output indent="yes"/>
    <xsl:template match="node()|@*">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
        </xsl:copy>
    </xsl:template>
    <xsl:template match="uax:UAVariable">
        <xsl:copy>
            <xsl:apply-templates select="@*"/>
            <xsl:apply-templates select="uax:DisplayName"/>
            <xsl:apply-templates select="uax:References"/>
            <xsl:apply-templates select="uax:Value"/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Ohjelma E.1 *XSLT-dokumentti muunnostuotoksena saadun XML-tiedoston UA-muuttujan lapsielementtien järjestyksen jälkiprosessointiin.*