Sähkötekniikan korkeakoulu

Automaatio- ja systeemitekniikka

FINNISH

KANDIDAATINTYÖN TIIVISTELMÄ

Tekijä: Ian Tuomi

Työn nimi:

Päivämäärä: Kieli: Suomi Sivumäärä:5+20

Tutkinto-ohjelma:

Vastuuopettaja:

Tämän tutkimuksen kohteena on järjestelmän ohjauslogiikan formaalin kuvauksen muuntaminen standardeita noudattavaksi logiikkakaavioksi.

Avainsanat: Automaatiojärjestelmä, Automatisoitu suunnittelu, ohjelmoitava logiikka, PLC, IEC 61131, FBD, graafinpiirto, suunnattu graafi

Esipuhe

Ilman seuraavia ihmisiä en olisi onnistunut tässä työssä.

Ohjaajaani Mika Strömmania hyvistä huomioista ja asiallisesta palautteesta ja skarpista ohjauksesta.

Petri Kokkoa ja Pekka Niemistä ideoinnista ja suunnannäyttämisestä.

Kandiryhmäni jäseniä Lauri Vepsäläistä, Heikki 'Taffis' Tahvanaista, Panu Kauppista, Peter 'Bembu' Kronströmiä ja Lauri Andleria.

Aalto- yliopiston henkilökuntaa hyvästä työstä. Vanhempiani tuesta ja kannustuksesta.

Otaniemi, Itsenäisyyspäivänä 6.12.2012

Ian Tuomi

Sisältö

Käsitteet ja lyhenteet

Lanka	Distributed Control System Hajautettu ohjausjärjestelmä
DCS	Distributed Control System Hajautettu ohjausjärjestelmä
FBD	Function Block Diagram, IEC 61131-3 -standardin määrittelemä logiikkaohjelmointikieli
IEC	International Electrotechnical Commission
PLC	Programmable Logic Circuit Ohjelmoitava logiikkapiiri

1 Johdanto

Teollisen automaatiojärjestelmän suunnittelutyön tulos koostuu suuresta määrästä erilaisia dokumentteja. Laajan järjestelmän tapauksessa kyse voi olla jopa tuhansista yksittäisistä kaavioista, asettelukuvista ja listauksista. Ne yhdessä muodostavat kuvauksen, joka mahdollistaa järjestelmän toteutuksen, käyttöönoton ja ylläpidon. Dokumentaation laatiminen käsin on suurissa projekteissa työlästä ja tehotonta. Nykyaikaisessa automaatio- ja instrumentointisuunnittelussa ongelma korostuu. Ratkaisuista on tullut monimutkaisempia ja laatu- ja joustavuusvaatimukset ovat lisääntyneet. Lisäksi vaaditut toteutusajat ovat jatkuvasti lyhentyneet. [1]

Suunnitteluprosessiin kohdistuneista vaatimuksesta johtuen työtä on pyritty tehostamaan automatisoiduilla apuvälineillä. Alan metodiikka ei ole ilmeisistä standardoimiseduista huolimatta yhtenäistynyt ja yhteistyö alan tutkimuksessa on vähäistä. Vaikka lähestymistavat ovat vaihtelevia, alalla on kuitenkin nähtävissä yhtenäinen siirtymä kohti jaettuja tietomalleja.

Automaatiojärjestelmien suunnittelun näkökulmasta jaettujen tietomallien käyttö tarkoittaa, että suunnittelujärjestelmään tallentuu projektin edetessä kaikki sähköistys ja instrumentointi automaatiojärjestelmineen ja kenttälaitteineen. Erilaiset kaaviot voidaan tuottaa suoraan tietomallin sisältävästä tietokannasta.

Tietomalliin perustuva lähestymistapa vaatii sen laajuuden ja rajoitukset määrittelevän metamallin. Se ohjaa ja määrittää suunnittelun kohteen lisäksi myös yleisesti suunnittelutöiden kulkua.

Tämän tutkimuksen kohteena on hajautetun ohjausjärjestelmän logiikan tietomallin muuntaminen standardeja noudattavaksi kaavioksi. Logiikkakaavio on tapa pelkistää järjestelmän eri osien syöttöjen ja lähtöjen välillä vallitseva logiikka visuaaliseen ja helposti ymmärrettävään muotoon. Se on määritelty yksiselitteisesti ja on tulkittavissa suunnatuksi graafiksi, jolloin algoritmillisen graafinpiirron tuloksia voidaan hyödyntää.

Algoritmillinen graafipiirto on perusteellisesti tutkittu ala, jonka tuloksia voidaan käyttää kaikenlaisten suunnattujen kaavioiden piirtoon. Työssä esitellään Sugiyaman ym. [2] esittämään hierarkiseen lähestymistapaan perustuva, tietovirtakaavioiden esittämiseen soveltuva algoritmi.

Esimerkki tyypillisestä teollisuuden logiikkakaaviosta esitellään ja se asetellaan se

käyttäen esiteltyä algoritmia. Lisäksi pohditaan miten työn menetelmät voitaisiin toteuttaa laajemmalla mittakaavalla.

2 Tehtaan mallit

Tehdasmalli on kuvaus, joka mahdollistaa tehtaan toteutuksen, käyttöönoton ja ylläpidon. Se on tietomalli, joka kuvaa tehtaan toimintaa, sen prosessia, organisaatiota, ihmisiä ja näitten aktiviteetteja. Pohjimmiltaan se koostuu tehdasobjekteista ominaisuuksineen sekä niitten välisistä relaatioista. [1]

Suunnittelijoilla on työtehtävistään riippuen erilaisia tarpeita tehdasmallin suhteen. Työn kohde on monitahoinen eikä mikään yksi malli pysty kattamaan kaikkia suunniteltavan järjestelmän näkymiä. Tehdasta tulee kuvata useilta eri näkökannoilta erilaisilla tarkkuuksilla kunkin näkymän vaatimuksien mukaisesti. Tehdasmalliin kuuluu siis useita malleja, jotka yhdessä muodostavat kokonaismallin.

Jokaisella mallilla on myös metamalli. Se määrittelee millaisia objekteja malli voi sisältää, mitä ominaisuuksia sillä voi tai täytyy olla ja millaisia yhteyksiä objektien välillä on. Metamallin loogiset riippuvuudet määrittelevät suunnittelun työtapoja ja järjestystä. Ne kuvaavat millaiset tietomallit ovat suunnittelujärjestelmän kannalta hyväksyttäviä tehdasmalleja ja määräävät sen laajuuden. Metamallin määrittäminen tekee työn tuloksesta ennakoitavan ja mahdollistaa tehokkaiden apuvälineiden kehittämisen. Määrittelyllä myös suljetaan pois eriäviä tapoja kuvata tehdasta ja mahdollistetaan määrittelyyn perustuvat työkalut ja suunnittelukäytännöt. Tällaisia ovat esimerkiksi uudelleenkäytettävää koodia sisältävät kirjastot ja erilaiset mallimuunnokset.

Suunnittelussa käytettävät metamallit valitaan siten, etteivät ne ole ristiriidassa ja muodostavat yhdessä toisiaan täydentävän kokonaisuuden. Tällöin metamallien konfiguraatio muodostaa itsessään mallin.

2.1 Automaatiojärjestelmän kuvaukset

Teollisuuden hajautettujen ohjausjärjestelmien suunnittelussa on havaittu tarpeellisiksi ainakin kolme kuvausta: toiminnallinen, fyysinen sekä ohjelmistollinen [3].

2.1.1 Toiminnallinen kuvaus

Toiminnallinen kuvaus määrittelee säätöjärjestelmän toiminnan tavalla, joka toteutettuna täyttää sille asetetut *käyttäjävaatimukset*. Se kuvaa järjestelmän toimintaa

ja, muttei välttämättä määrittele sen tarkkaa teknistä toteutusta. [**?**]

Tätä näkymää määrittäessä voivat eri alojen suunnittelijat osallistua työhön ilman tuntemusta instrumentoinnista tai ohjelmistokehityksestä. Kun päätökset on tehty abstraktilla tasolla, voidaan hankinnat räätälöidä sen mukaisesti ja tulos toteuttaa tehokkaasti. Tällaisen määrittelyn avulla on mahdollista käyttää formaaleja verifiointimenetelmiä joitten avulla voidaan osoittaa, että järjestelmän suunnitelma toteuttaa sille asetetut vaatimukset. Näin suunnitelman oikeellisuus voidaan osoittaa ennen investointeja toteutukseen. [**?**]RefWorks:41

2.1.2 Fyysinen näkymä

Fyysiseen näkymään suunnitellaan järjestelmän johdotukset, kaapelit, kenttäväylät, ohjauskeskukset, toimilaitteiden sijainnit sekä ylipäänsä kaikki mitä vaaditaan järjestelmän fyysiseen käyttöönottoon.

2.1.3 Ohjelmistonäkymä

Ohjelmistonäkymä kattaa funktionaalisen suunnittelun toteutuksen ohjelmamuodossa. Asiakkaalla on yleensä omat toiveensa sen suhteen, millaiseen muotoon ohjelma tehdään. Ohjelmiston toteuttamiseen ei pitäisi liittyä enää suunnittelutyötä - toisin sanoen toiminnallisen kuvauksen määrittelyjen täytyy olla riittävän tarkkoja jotta niitten mukaan tehdyssä ohjelmassa ei ole sen toimintaan vaikuttavia tulkinnanvaraisuuksia.

Ohjelmistotototeutuksen työtavat jäävät usein määrittelemättömiksi ja muusta suunnittelutyöstä irrallisiksi, vaikka pyrkimyksiä toteutustapojen nitten integroimiseksi suunnitteluprosessiin entistä vahvemmin on ollut. Toteutustapa jää yksittäisten suunnittelijoiden määriteltäväksi, mistä seuraa että yhtenäistä työtapaa harvoin on. Toteutetun ohjelmakoodin uudelleenkäyttö jää myös tyypillisesti vähäiselle asteelle. [4]

2.2 Logiikan kuvaukset

Logiikan kuvaukset helpottavat tehtaan toiminnan ymmärrystä suunnittelu- ja rakennusvaiheen lisäksi tehtaan ylläpidon ja huollon yhteydessä ja ovat siksi tärkeä

osa tehtaan lopullista dokumentaatiota. Loogista järjestelmää voidaan kuvata monin erilaisin tavoin. Nämä erilaiset kuvaukset voivat näyttää erilaisilta, mutta olla silti täysin yksiselitteisiä suhteessa kuvattuun järjestelmään. Teollisessa automaatiosuunnittelussa sallitut logiikan esitystavat on määritelty erityisesti selkeyttä, turvallisuutta ja ennustettavaa toimintaa silmälläpitäen. Jatkuva prosessiohjaus voidaan esittää selkeästi järjestelmän PI-kaavioissa, mutta diskreetin ohjauksen esittämiseen tarvitaan erilaisia esityksiä, kuten logiikkadiagrammeja. [5]

Logiikkadiagrammit kuvaavat syöttöjen ja lähtöjen välistä logiikkaa. Määrittelyn mukaisia kaavioita ei kuitenkaan ole välttämättä helppoa laatia.

Kun kaavioista tulee monimutkaisia, kuluu niitten asetteluun huomattavan paljon aikaa silloin, kun tavoitteena on mahdollisimman luettava kaavio. Klausken ja Dziobekin mukaan kaavioiden asetteluun kuluu suunnittelijoiden ajasta n. 30%. Lisäksi he arvioivat, että kaavioiden asettelulla saavutettavan luettavuuden myötä voitaisiin toimilohkoihin perustuvaan suunnittelu- ja ohjelmointityöhön käytettävä työaikaa lyhentää parhaimillaan puoleen. [**?**]

Koska työaika on yrityksessä rajallinen voimavara, Kun tietokoneen avulla toteutettava algoritmillinen muunnos kuvauksesta kaaviomuotoon onnistuu ilman suunnittelijalle siitä koituvaa vaivaa tai aikaa, kannattaa suunnittelijan laatia kahdesta kuvaustyypistä helpompi ja antaa algoritmin hoitaa lopullisen kuvauksen laatiminen. Algoritmillisesti laaditut kaaviot voivat olla jopa parempia kuin käsintehdyt *[lähde on, pitää kaivaa]*

2.2.1 Toimintakaaviot

Toimintakaavio on funktionaalisen näkymän toteutus, joka tavallisesti valmistellaan järjestelmän suunnittelun varhaisessa vaiheessa prosessikaaviosta. Se määrittelee ohjausjärjestelmän toiminnot ja sitä pidetään ajantasaisena järjestelmän suunnittelun edetessä. Se toimii suunnittelun apuna ja on lopulta osa lopullisen järjestelmän ohjeistusta.

Suomen automaatioseura

2.2.2 Ohjelmakoodi

Ohjelmakoodi on ohjelmanäkymän toteutus. IEC 61131-3 on laajassa käytössä oleva avoin ohjelmakoodin standardi joka määrittelee automaatiologiikan ohjelmointimenetelmät ja pyrkii olemaan riittävän joustava mihin tahansa tarkoitukseen. [**?**] Se sisältää viisi erilaista ohjelmointikieltä, joista tässä työssä käsitellään erityisesti FBD:tä.

FBD-kieli koostuu toimilohkoista ja niiden välisistä langoista. Toimilohkot kuvaavat toisistaan erillisiä itsenäisiä laskennallisia yksiköitä. Langat kuvaavat niitten relaatioita toisiinsa. Toimilohko voi tilastaan riippuen lähettää tapahtumia jotka vaikuttavat muitten blokkien toimintaan. Sen tekemien laskutoimitusten tulos riippuu sen vasemmalta puolelta tulevista syöttötiedoista ja sen suorittaman laskennan tulokset lähetetään blokin oikealta puolelta. Toisiinsa liitetyt funktioblokit muodostavat verkon, joka määrittelee laajemman toiminnallisuuden. [6]

Funktioblokkien editointiin on olemassa useita työkaluja. Näistä mainitsemisen arvoisia ovat ainakin FBDK, 4DIAC ISaGRAF ja NxtControl. Ne eivät kuitenkaan tarjoa graafien asetteluun kuin alkeellisia työkaluja, jotka pikemmin helpottavat suunnittelijan asettelua kuin tuottavat itsessään ratkaisuja.

Automaatiokaavioiden suoritusjärjestys etenee standardin IEC 61131-3 mukaan vasemmalta oikealle. Lisäksi toimilohkoa ei ajeta ennen kuin kaikki sitä edeltävät toimilohkot on suoritettu. Keskinäisen suoritusjärjestyksen täytyy myös tulla esille toteuttavassa ohjelmistossa. Suoritusjärjestys kulkee usein ylhäältä alas, mutta tätä ei voi olettaa aina todeksi. [**?**]

Suoritusjärjestysvaatimus asettaa seuraavassa kappaleessa tarkasteltavalle graafinpiirrolle rajoituksia. Jos niitä ei oteta huomioon, saattaa graafin algoritmillinen piirto johtaa vääränlaiseen kaavioon. Mahdollisten takaisinkytkentöjen tapauksessa vaaditaan huolellisuutta, jottei syklien poistovaiheessa kaavion järjestys muutu.

2.2.3 Logiikan tietokantakuvaukset

Suunniteltu logiikka tallennetaan jaettuun suunnittelumalliin, johon kaikilla järjestelmän suunnittelijoilla on yhteys. Tapa jolla logiikka tallennetaan täytyy olla yhdenmukainen, jotta suunnittelutyössä voidaan tehdä tehokasta yhteistyötä työpaikalla jaettujen yhtenäisten suunnittelun apuvälineiden avulla. Lisäksi kuvauksen

tulee olla formaali jotta logiikkaa voitaisiin käsitellä algoritmillisesti. Formaalilla tarkoitetaan tietynlaista, tarkkaan määritetyn syntaksin mukaista kuvausta. Tällöin tulee myös mahdolliseksi toteuttaa helposti tietoa käsitteleviä muunnosalgoritmeja. Muunnoksella tarkoitetaan jonkin tiedon kuvauksen muuttamista toiseksi kuvaukseksi.

Standardin IEC 61131-3 mukaisten ohjelmointikielten kuvaamiseksi on ehdotettu PLCopen-standardia. PLCopen -standardin mukainen logiikan XML-esitys voi sisältää toimilohkon toiminnallisuuden kannalta välttämättömien asioiden lisäksi toimilohkon sijainnin, koon ja jopa lankojen reititykset. PLCopen -standardin mukainen esitys voidaan tällöin laatia siten, että se on tuotettavan kaavion asettelun kannalta täysin yksiselitteinen. Tällaista esitystä voidaan silloin pitää dokumentaation lopullisena muotona. [**?**]

Kun logiikkakuvauksen muoto on määritelty tarkasti, voidaan määritelmän mukaisesti laadittu logiikkakuvaus muuntaa määritelmään nojaavien sääntöjen perusteella. Monet automaatiologiikan suunnitteluohjelmistot pystyvät muuntamaan käyttämänsä kuvauksen PLCopen-muotoiseksi tiedoksi, jolloin suunnittelutietoa voidaan siirtää eri järjestelmien kesken. Erilaisia XML-kuvauksia on helppoa laatia ja niitten sisältämien tietojen pohjalta voidaan tuottaa XSL-muunnoksilla. [**?**]

Olemassaolevien suunnitteluohjelmistojen varaan rakennettu suunnittelujärjestelmä voi kohdata haasteita, jos se perustuu ohjelmien standardinmukaisuuteen. Valmistajat jättävät usein standardinmukaiseksi kutsutuista ohjelmistoista määrittelyn mukaisia ominaisuuksia toteuttamatta. Tämä johtuu standardinmukaisuuden määritelmästä: ohjelma on standardinmukainen, jos se toteuttaa standardista jonkin osan ja mainitsee mitä se jättää toteuttamatta. [4]

3 Graafit ja niiden asettelu

3.1 Määritelmiä

Lähdetään liikkeelle graafin määritelmästä [7]. Graafi on yleisen määritelmänsä mukaan joukko solmuja ja niitä yhteen liittäviä lankoja. Lankoja kutsutaan usein myös kaariksi, reunoiksi, väleiksi tai linkeiksi. Solmusta käytetään myös ilmaisua noodi. Matemaattisesti esitettynä graafi on joukko $G = (V, E)$, missä V on rajoitettu solmujoukko ja E on rajoitettu lankajoukko.

Kerrostus on jaottelu $\mathfrak{L} = (L_1, L_2, ..., L_h)$, jonka alajoukot sisältävät yhdessä kaikki solmut V. Solmujoukon erilaisia kerrostuksia vertaillaan jaottelun *leveyden* ja *korkeuden* avulla. Korkeus[1] on kerrosten määrä h ja leveys[2] on suurimman kerrosjoukon L_i koko.

Laajennetaan määritelmää ja sisällytetään graafin määritelmään joukko portteja P ja funktio $\mathsf{n} : P \to V$, joka kartoittaa portit niitä vastaaville solmuille.

Lanka koostuu porttiparista, joita se yhdistää. Graafi on *suunnattu* silloin, kun sen langat ovat järjestettyjä pareja, joista toinen on tulo- ja toinen lähtöpiste. Toisinsanoen, $e = (p_1, p_2)$, missä p_1 on lähtö- ja p_2 tuloportti.

Solmu v_j on solmun v_i *seuraaja* silloin, kun on olemassa lanka $e \in E$, jolle $\mathsf{n}(p_1) = v_i$ ja $\mathsf{n}(p_2) = v_j$. Polku on solmusekvenssi $(v_1, v_2, ..., v_k)$ jolle pätee, että $k > 1$ ja jokainen sekvenssin solmu v_{i+1} on solmun v_i seuraaja.

Graafi on *syklitön* silloin, kun mikään polku ei sisällä samaa solmua enemmän kuin kerran. Lanka (p_1, p_2) on *itseohjautuva* silloin, kun $\mathsf{n}(p_1) = \mathsf{n}(p_2)$. Kerrostus on *kunnollinen*[3] silloin, kun jokaisen kerrostusjoukon L_i jokaisen solmun seuraajat kuuluuvat kerrokseen L_{i+1}.

Virityspuu on verkon aliverkko G_s joka sisältää verkon kaikki solmut siten, että se sisältää mahdollisimman vähän lankoja. Virityspuussa ei ole syklejä, koska mikä tahansa virityspuuksi ehdotetussa verkossa sijaitseva sykli $v_i, ..., v_i$ voidaan rikkoa poistamalla mikä tahansa yksittäinen sen sisältämistä langoista ilman, että sen sisältämän solmujoukon E koko muuttuu. Kun virityspuusta poistetaan lanka, jakaantuu

[1] Tietovirtakaavioissa vasemmalta oikealle
[2] Tietovirtakaavioissa ylhäältä alas
[3] eng. proper

puu kahdeksi.

3.2 Tulosten arviointiperusteet

Asetteluista halutaan mahdollisimman esteettisiä. Estetiikka erittelee ominaisuudet mitä graafiin halutaan mahdollisimman paljon, jota se olisi mahdollismman luettava. Asettelulle voidaan muodostaa

Purchase ym. ovat tutkineet estetiikkaan vaikuttavia tekijöitä. [8]

Tärkeimmät ominaisuudet järjestyksessä Purchase et. al. mukaan:

1. Lankojen risteysmäärä
2. Kaavion koko
3. Lankojen yhteispituus
4. Pisin lankapituus
5. Symmetria

Näistä ominaisuuksista lankojen risteysmäärä muodostui tärkeimmäksi. Myös lankojen pituus oli merkittävässä asemassa luettavuuden kannalta.

Esteettiset ominaisuudet ovat usein ristiriidassa keskenään. Jotta voitaisiin esimerkiksi välttää langan piirto solmun läpi, joudutaan siihen usein tekemään mutkia. Kompromisoimattoman esteettiset kaaviot ovat harvinaisuuksia.

Valmiin kaavion asettelua voidaan arvioida automatisoidusti laskemalla kaavioista esteettisyyteen vaikuttavia tekijöitä, kuten lankojen risteysmääriä.

3.3 Liittyvä työ

Suurin osa graafinpiirrollisista lähestymistavoista on suunniteltu ennalta tuntemattomien verkkojen piirtämiseen. Ne pyrkivät tuottamaan keskimäärin hyvän asettelun, oli graafi millainen hyvänsä. Suunnatun verkon rakenteen huomioonottavat algoritmit tuottavat parempia tuloksia.

Suunnattuja graafeja käytetään laajalti riippuvuussuhteiden mallintamiseen. Esimerkkejä suunnatuista graafeista ovat projektinhallintaverkot, ontologioiden mää-

rittelykaaviot (kuuluu ryhmään), ohjelmien kutsupuut, syy-seurausdiagrammit sekä aiemmin osioissa 2.2.1 ja 2.2.2 käsittelemämme toiminta- ja ohjelmakoodikaaviot.

Ylivoimaisesti suosituin tapa suunnattujen graafien piirtoon on Sugiyaman menetelmä. [2] Alkuperäinen ajatus graafien kerrostamisesta voidaan johtaa Warfieldin [9] ja Carpanon [10] esittämiin ajatuksiin graafinpiirrosta. Kerroksittainen tapa esittää syklittömiä ja suunnattuja verkkoja on intuitiivinen ja helppolukuinen. Sugiyaman algoritmia ei voi kuitenkaan käyttää sellaisenaan tietovirtakaavioiden asetteluun. Aseteltavissa tietovirtakaavioissa esiintyvät toimilohkot sisältävät useita portteja, joiden keskinäinen järjestys on säilytettävä. Lisäksi langat ovat tietovirtakaavioissa yleisen käytännön mukaisesti ortogonaalisia, jolloin algoritmia pitää täydentää erillisellä langoille tarkoitetulla reititysalgortimilla. Sugiyama ym. lähestymistapa jaottelee algoritmin vaiheisiin, joista jokaista voidaan tutkia erillisenä ongelmana.

Sen määrittelemä jaottelu on säilynyt hierarkisen graafinpiirron tutkimuskentässä ajankohtaisena, vaikkei sen esittämiä algoritmeja käytetäkään. Vaiheet ovat järjestyksessä *kerrostus*, *risteysten vähennys* ja *lopullinen asettelu*. Myöhemmin tietovirtakaavioiden piirtoon on kehitetty lisävaiheet *suuntaus* ja *lankojen reititys*.

Gansner ym. esittivät porttiongelmaan ratkaisun, joka perustuu noodisijoitteluesityksen laajentamiseen porttien suhteellista sijaintia kuvaavalla tiedolla.[11] Sander ym. esitti reititysongelmaan ratkaisun, jossa käsitellään yksitellen jokaisen ruudun ylittävien lankasegmenttiryhmää omana graafiongelmanaan, johon sovelletaan suurempaan ongelmaankin käytettäviä graafinpiirron menetelmiä. [**?**]RefWorks a Työvaiheista kerrostus ja risteyksien vähennysvaihe saattavat muuttaa suoritusjärjestystä. Tällöin algoritmin toimintaa pitää yksinkertaisesti rajoittaa järjestysherkkien blokkien osalta. Koska algoritmit ovat heuristisia, ovat siihen tehtävät yksinkertaiset rajoitukset helppoja toteuttaa.

Toimilohkojen syötöt ovat yleensä lohkon vasemmalla ja ulostulot oikealla puolella. Näin on FBD-ohjelmointikielen tapauksessa aina. Toimintakaavioiden tapauksessa syöttöjen ja ulostulojen sijainteja ei ole määritelty yhtä tarkasti, jolloin sivuportit on otettava asettelevassa algoritmissa myös huomioon jos sitä halutaan käyttää. Sivuporttien huomioonottoon on esitetty ratkaisu, jossa lankojen reititystä varten asetetaan valesolmuja. [12]

Sovelluksia graafinasettelualgoritmeille mallipohjaisessa suunnittelussa ovat kehittäneet ainakin Klauske...

3.4 Graafinasettelualgoritmi

Algoritmin eri vaiheiden algoritmit on valittu ottaen huomioon datavirtauskaavioiden rajoitukset ja erityiset tarpeet sekä niitten rakenteelliset ominaisuudet. Laajoja käytännön tutkimuksia datavirtakaavioiden asettelusta on tehnyt Klauske ym.[13].

Se koostuu viidestä eri vaiheesta:

1. Syklien poisto, seuraa DFS-hakua [**?**] tai Eades ym. ahnetta algoritmia [14]
2. Kerrostus, seuraa Gansner ym. lankapituuden minimimoivaa menetelmää. [11]
3. Risteysten vähentäminen, seuraa Sugiyama ym. alkuperäistä menetelmää. [2]
4. Poikittaisasettelu, seuraa Sanderin lineaaristen segmenttien menetelmää. [15]
5. Lankareititys, seuraa Sanderin monilankareititysalgoritmia. [16]

Nämä vaiheet ovat toisistaan erillisiä ja voidaan tilanteen mukaan vaihtaa toisiksi tai jättää suorittamatta. Jos esimerkiksi toimilohkojen paikat halutaan pitää paikallaan mutta laatia lankojen reitityket automaattisesti, voitaisiin päättää ajaa pelkästään algoritmin viides osio.

3.4.1 Syklien poisto

Syklinpoistovaiheessa muotoillaan graafi siten, että se on syklitön. Syklit eli takaisinkytkennät ovat graafinpiirtoalgoritmien vaatimuksista huolimatta välttämätön osa kaavioita, eikä niitä voida kokonaan poistaa. Tämän vuoksi syklit rikotaan kääntämällä yksi tai useampi syklin sisältämistä langoista.

Tarkemmin määriteltynä tavoitteena on löytää mahdollisimman pieni lankajoukko $FS \subset E$ jotka kääntämällä verkossa ei ole enää syklejä jäljellä. Tämä ongelman on osoitettu olevan NP-vaikea. [**?**]

Syklien poistovaiheessa käsiteltävä tietomalli sisältää syklinpoistoa helpottavaa tietoa, nimittäin jokaisessa puussa ensimmäisenä suoritettavan solmun. Koska suorituspuun ensimmäinen kerros on jo ennen asettelua tiedossa, voidaan riittävän pieni lankajoukko FS löytää DFS-haulla.

DFS-haussa edetään ensimmäisistä solmuista lankoja pitkin eteenpäin kunnes on luotua polkua $(e_1, e_2 ... e_h)$ ei voida enää jatkaa. Tämän jälkeen palataan takaisin

kunnes löydetään solmu josta lähtee [**?**]

Käännetyt langat käännetään lankojen reitittämisvaiheessa takaisin oikein päin käyttäen hyväksi lankajoukkoa FS.

Jos ensimmäinen kerros ei ole valmiiksi tiedossa tai sillä ei ole merkitystä, voidaan käyttää Berger-Shor algoritmiin [**?**] perustuvaa ahnetta algoritmia [14].

3.4.2 Kerrostus

Kerrostukselta vaaditaan, että se on *kunnollinen* osion 3.1 mukaisesti, sekä *tiivis*[4]. Tiiviydellä tarkoitetaan, että kerrostuksen leveys ja pituus ovat pieniä ja kerrosten välinen etäisyys on vakio.

Tämän saavuttamiseksi graafiin lisätään valesolmuja pitkien lankojen katkaisemiseksi. Valesolmut ovat verkkoon lisättäviä kuvitteellisia solmuja joita ei piirretä lopulliseen kaavioon, mutta jotka helpottavat verkon algoritmillista käsittelyä. Kerrostukseen halutaan valesolmuja mahdollisimman vähän tiiviyden, lyhyiden lankapituuksien sekä nopean laskennallisen suoritusajan varmistamiseksi.

Tiiviyden äärimmäisenä rajoituksena toimii tarkastelutavasta riippuen näyttöpääte tai paperi, jolta aseteltua verkkoa on tarkoitus tarkastella. Verkon pituuden alaraja on sen sisältämän pisimmän yhtenäisen ketjun pituus.

Erilaisilla kerrostusmenetelmillä graafin pituus voidaan minimoida leveyden kustannuksella tai siitä voidaan tehdä mahdollisimman kapea pituuden kustannuksella. Sekä leveyden että pituuden minimointi samanaikaisesti on rinnastettavissa multiprosessoriajastusongelmaan ja on siten NP-täydellinen ongelma. [7]

Yksinkertainen tapa tuottaa kerrostus on pisimmän reitin tapa. Se asettaa ensin kaikki solmut, joilla ei ole seuraajia kerrokseen L_1. Tämän jälkeen jokainen muu solmu asetetaan kerrokseen L_{p+1}, missä p on lähin etäisyys johonkin kerroksen L_1 solmuista. Tällä tavalla laaditut kerrostukset sisältävät mahdollisimman vähän kerroksia. Tämä lähestymistapa ei tuota hyviä asetteluja, sillä joistakin kerroksista tulee huomattavan suuria.

Kun halutaan mahdollisimman kapeita kerrostuksia, on suotavaa käyttää multiprosessoriajastusongelman ratkaisemiseksi kehitettyä Coffmann-Graham -algoritmia.

[4] eng. compact

[17] Ongelmana on ajastaa eri tehtävät W kappaleelle prosessoreita siten, että kaikki tehtävät suoritetaan ajassa H. Löydämme vastaavuuden verkon kerroksien määrälle ja suoritusajalle sekä suurimmalle sallitulle kerroskoolle ja prosessorien määrälle.

On olemassa myös tapa laatia kerrostus, joka minimoi valesolmujen määrän. Se löydetään kun otetaan käyttöön lineaarisen ohjelmoinnin optimointimenetelmä jolla minimoidaan

$$f = \sum_{(v_1,v_2)\in V} (i - j - 1),$$

, missä $v_1 \in L_i$ ja $v_2 \in L_j$.

Huomionarvoista on se, että f on sama kuin valesolmujen määrä.

nk. *Network Simplex*- algoritmiin.

Sen aikakompleksisuutta ei ole pystytty osoittamaan, mutta se on käytännössä osoittautunut suoritusajaltaan nopeaksi.

Menetelmässä laaditaan ensiksi graafin virityspuu *(ks. osio 3.1)*. Kerrostamisessa valitaan ensin lähdesolmu, jonka kerros on L_0. Jokainen solmun seuraaja asetetaan kerrokseen L_1 ja niin edelleen

Kun mahdollinen virityspuu on laadittu, käytetään hyväksi virityspuun ominaisuutta, että jos jokin sen lanka poistetaan, tuotetaan kaksi toisistaan erillistä puuta, Jokaiselle langalle määritellään *leikkausarvo*. Leikkausarvolla tarkoitetaan kaikkien lankojen summaa päänoodista loppunoodiin, josta vähennetään

3.4.3 Risteyksien vähentäminen

Lankojen risteysten määrä on osoittautunut olemaan suurin yksittäinen tekijä graafin luettavuuden kannalta. [8] Risteysmäärän vähentäminen pieneksi on tällöin mille tahansa luettavuutteen tähtäävälle asettelualgoritmille tärkeä tavoite.

Graafin kokonaisristeysmäärän vähentäminen ei perustu solmujen tarkkoihin sijainteihin, vaan niitten keskinäiseen järjestykseen. Ongelma on siis luonteeltaan kombinatorinen eikä geometrinen, mikä huomattavasti yksinkertaistaa hyvän ratkaisun

löytämistä. Graafi on myös muokattu kerrostamisvaiheessa kelvolliseksi, minkä seurauksena ongelmaa ratkaistessa voidaan keskittyä vuorotellen vain kerrosparin L_i ja L_{i+1} välisiin lankoihin. Näistä helpotuksista huolimatta kyseessä on NP-täydellinen ongelma siinäkin tapauksessa että koko graafissa on kerroksia vain kaksi. [18]

Parhaan mahdollisen kombinaation löytämiseksi joudutaan yksinkertaisen ratkaisun puutteessa käyttämään heuristisia menetelmiä. Eräs suosittu lähestymistapa perustuu solmun seuraajasolmujen sijoittamiseen mahdollisimman lähelle lähdettään. Teorian kannalta tällä nk. mediaanimenetelmällä on useita hyviä ominaisuuksia. Sen avulla tuotettu ratkaisun risteysmäärä on vakiossa suhteessa optimaaliseen. Lisäksi algoritmi voidaan suorittaa polynomisessa ajassa. Käytännön kokeiluissa satunnaisilla graafeilla on kuitenkin todettu, ettei mediaaneihin perustuva menetelmä ole käyttökelpoisin. [19] Battisti ym. suosittelevatkin eri menetelmiä yhdisteleviä hybridialgoritmeja, kuten mediaanimenetelmän ja satunnaisten tranpositioiden yhdistämistä[7].

Työssä käytetty algoritmi arpoo ensimmäisen kerroksen L_1 järjestyksen. Se sitten etenee kerroksittain eteen- ja taaksepäin tehden järjestykseen transpositioita. Risteysten määrät lasketaan eri tuloksista ja ratkaisu, jossa on vähiten risteyksiä valitaan lopulliseksi.

3.4.4 Poikittainen asettelu

Jokainen kulma langassa aiheuttaa ylimääräisen rasitteen ihmisen hahmotuskyvylle. [8] Esteettistä lopputulosta tavoittelevan asettelualgoritmin tulee siis pyrkiä minimoimaan lankojen kulmat. Siinä missä noodikerroksien noodikombinaatiot vaikuttavat lankojen risteysten määrään, vaikuttaa niiden tarkka poikittainen asettelu langoissa olevien kulmien määrään. Solmujen poikittaisen asettelun avulla solmujen välisten lankojen käännösten määrää pyritään vähentämään jotta kaavion luettavuus paranee. Asettelun merkitys korostuu kun lankoja on paljon, kuten automaation logiikkakaavioille on tyypillistä.

Risteysten vähentämisvaiheessa saavutettu kombinatorinen ratkaisu pyritään säilyttämään poikittaisasetteluvaiheessa. Poikittainen asettelu tuottaa pitkittäisten kerrospositioitden $1...h$ lisäksi poikittaiset noodipositiot $1...n$ Samalla poikittaisella positiolla olevien noodien välille voidaan reitittää suoria lankoja, joten solmujen seuraajat pyritään sijoittamaan samalle poikittaispositiolle jos mahdollista.

Sander ehdottaa kaksivaiheista lähestymistapaa, jossa aluksi määritellä hyvä alkusijoittelu joka sitten tasapainoitetaan.

Verkko jaetaan lineaarisiin segmentteihin. Lineaarinen segmentti sisältää joko peräkkäisiä valesolmuja tai yksittäisen solmun. Solmu v

Aluksi pyritään tuottamaan asettelu, joka on ylipäänsä mahdollinen. Tämä tehdään jakamalla poikittais- ja pystypositiot yksinkertaisesti pienimmästä suurimpaan. Tämän jälkeen graafin solmut lajitellaan topologisesti. Suunnatun graafin topologinen lajittelu

Kun on saavutettu yksinkertaisin mahdollinen asettelu, voidaan siirtyä tasapainottamaan asettelua. Solmuasettelu tasapainoitetaan mallintamalla solmuja levossa olevina heilureina. Solmut ovat heilurin päässä oleva paino ja lanka painoa kannatteleva naru. Tasapainotettavaa kerrosta seuraava kerros pysyy kiinteänä paikoillaan.

Liikutettavat asiat ovat eivät ole yksittäisiä noodeja vaan lineaarisia segmenttejä. Ne pysyvät suorina linjoina.

Solmun seuraajan poikkeama voidaan laskea

$$D_p(v) = \frac{\sum_{e \in E} D_p(e)}{indeg(v)}$$

$$D_p(e) = x(s) - x(t)$$

Kun solmujen sijainnit on tasapainotettu, jaotellaan asetteluun käytössä oleva tila ruudukoksi ja asetetaan solmut niitten tasapainotettuja sijainteja lähinnä oleviin ruudukkopisteisiin. Tällainen ruudukointi tekee kaaviosta vähemmän tasapainoisen, mutta käytännössä ruduukointi selkeyttää luettavuutta.

[15]

3.4.5 Lankojen reititys

Kaavion langat reititetään lopulta ortogonaalisesti..[16] Kun jokaiselle solmulle on tiedossa niiden lopullinen sijainti, luodaan lankasegmenttejä siten, että ne liikkuvat solmien koordinaattien välillä. Poikittaiset segmentit jotka asetetaan tyhjään tilaan

rivien j ja $j+1$ välille saavat koordinaatikseen $j+0.5$. Tämä toistetaan pystysuuntaisille segmenteille.

Langat ryhmitellään lähtöporttien mukaan *hyper*langoiksi, joilla voi olla useita lähteitä.

Jokaiselle lankajoukolle luodaan ne yhdistävä pystysegmentti.

Kaikki samassa ruudussa tai sarakkeessa k olevat segmentit kootaan ryhmäksi joukkoon S. Tavoitteena on levittää ne koordinaateille välillä $[k, k+1]$.

Risteysten minimoimiseksi luodaan segmenttiylitysgraafi jokaiselle ruudulle. Jokainen ruutuun kuuluva segmentti vastaa yhtä segmenttiylitysgraafin noodia. Jokaiselle segmenttiparille s1, s2 lasketaan risteysmäärät C1 ja C2. Jos C1 < C2, lisätään reuna segmentin s1 ja s2 välille hintaan C2 - C1. Muussa tapauksessa lisätään lanka s2 ja s1 välille hintaan C1 - C2.

Asyklisen segmenttiylitysgraafin tapauksessa ylitysgraafi voidaan järjestää topologisesti. Tässä tapauksessa risteysmäärä on vähin mahdollinen. Ylitysgraafi kuitenkin yleensä sisältää syklejä. Tässä tapauksessa syklit rikotaan, jo tunnetulla tavalla [14].

Lopulta segmenttien sijoitus lasketaan syklittömästä segmenttiristeysgraafista. Segmentti jonka sijoitus on $r \in \{1, ..., r_{max}\}$ saa koordinaatikseen $k + r/(r_{max}+1)$. Kun tämä vaihe on suoritettu kaikille poikittaisille segmenteille, sama tehdään kaikille poikittaisille segmenteille.

4 Menetelmät ja esimerkki

4.1 Kaavioiden tuottaminen

Kaavioiden tuottaminen voidaan toteuttaa suunnittelijan käyttämällä tietokoneella tai keskitetyllä kaaviopalvelimella. Verkossa sijaitseva kaavioipalvelin soveltuu suurten järjestelmien suunnitteluun erityisen hyvin, kun suunnittelijoita on monta. Kaikki suunnittelijat pystyvät hakemaan keskitetystä suunnittelujärjestelmästä ajantasaisimman version kaaviosta ilman, että kaaviot tarvitsee ladata järjestelmään erikseen. Lisäksi näin vältytään versioinnin tuottamilta haasteilta - uusi versio asettelualgoritmista voidaan tuoda kaikille suunnittelujärjestelmän käyttäjille yhdenaikaisesti.

Eclipse-projektin päälle rakennettu kokoelma mallipohjaisen suunnittelun apuvälineitä. Sitä kehitetään Kielin yliopistossa.

4.2 Esimerkki

Lopullisessa työssä on esimerkki. Opponointiversioon se ei ehtinyt.

5 Yhteenveto ja tulevaisuudennäkymät

Teollisuuden automaatiosuunnittelun yhtenäistämistä on akateemisissa julkaisuissa kartoitettu runsaasti. Nämä pyrkimykset ovat kuitenkin usein jääneet abstraktille, suunnittelijat vieraannuttavalle tasolle.

Automaatiojärjestelmien suunnittelu- ja toteutusprosessi on monimutkainen, kallis ja pitkä eivätkä sen . Suunnittelussa käytettäviä työvälineitä tehostamalla voidaan saavuttaa merkittäviä säästöjä ja nopeuttaa toimihenkilöiden työtä.

Tämä työ on pyrkinyt tarjoamaan konkreettisen ratkaisun konkreettiseen suunnittelutyön ongelmaan, visuaalisen suunnittelun ja ohjelmoinnin kaavioiden asetteluun. Ensisijainen pyrkimys on ollut kuvailla yksinkertaisesti toteutettava ratkaisu joka voidaan helposti integroida suunnitteluprosessiin.

Työssä on osoitettu yhteys suunnattujen graafien ja logiikkadiagrammien välillä sekä suunnattujen graafien asetteluun kehitettyjen algoritmien käyttökelpoisuus teollisuuden automaatiosuunnittelussa. Työ esitteli logiikkakaavioiden asetteluun soveltuvan algoritmin, joka on muokattavissa edelleen erilaisiin teollisuuden tarpeisiin.

On huomattava määrä teknologisia uudistuksia, jotka eivät päädy käyttöön. Vaikka tuote olisikin mahdollinen, käytännöllinen ja kannattava, tarvitaab niitten lisäksi toimijoiden tahtotila olemassaolevien prosessien muuttamiseen ja uuden opetteluun. Työssä esitelty ja sitä vastaavat menetelmät voivat olla vaikeita toteuttaa käytännössä. Tästä huolimatta täytyy suunnittelualan yritysten pyrkiä kohti entistä enemmän integroidumpaa ja virtaviivaistettua suunnitteluprosessia jos halutaan pysyä perässä teollisuuden kilpailussa.

Viitteet

[1] Suomen automaatioseura ry. Automaatiosuunnittelun prosessimalli: Yhteiset käsitteet verkottuneen suunnittelun perustana. Technical report, Suomen automaatioseura ry, 2007.

[2] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2):109–125, 1981.

[3] M. Marcos and E. Estevez. Model-driven design of industrial control systems. In *Computer-Aided Control Systems, 2008. CACSD 2008. IEEE International Conference on*, pages 1253–1258. IEEE, 2008.

[4] J. Kääriäinen. Ohjelmakirjaston hyödyntäminen automaatiojärjestelmässä. 2008.

[5] F. Meier, P. F. D. PFD, and L. Numbering. Control system documentation. *A Guide to the Automation Body of Knowledge*, page 75, 2006.

[6] K. H. John and M. Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-making Aids*. Springer, 2010.

[7] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Layered Drawings of Digraphs*. Graph drawing: algorithms for the visualization of graphs. Prentice Hall PTR, 1998.

[8] H. Purchase, R. Cohen, and M. James. Validating graph drawing aesthetics. In *Graph Drawing*, pages 435–446. Springer, 1996.

[9] J. N. Warfield. Crossing theory and hierarchy mapping. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(7):505–523, 1977.

[10] M. J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *Systems, Man and Cybernetics, IEEE Transactions on*, 10(11):705–715, 1980.

[11] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. *Software Engineering, IEEE Transactions on*, 19(3):214–230, 1993.

[12] G. Sander. Graph layout through the vcg tool. In *Graph Drawing*, pages 194–205. Springer, 1995.

[13] L. Klauske, C. Schulze, M. Spönemann, and R. von Hanxleden. Improved layout for data flow diagrams with port constraints. *Diagrammatic Representation and Inference*, pages 65–79, 2012.

[14] P. Eades, X. Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993.

[15] G. Sander. A fast heuristic for hierarchical manhattan layout. In *Graph Drawing*, pages 447–458. Springer, 1996.

[16] G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. In *Graph Drawing*, pages 381–386. Springer, 2004.

[17] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3):200–213, 1972.

[18] M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[19] M. Jünger and P. Mutzel. *2-layer straightline crossing minimization: Performance of exact and heuristic algorithms*. MPI Informatik, Bibliothek und Dokumentation, 1996.