Ian Tuomi

# Logiikkakaavioiden generointi tehdassuunnittelussa

**Automaatio- ja systeemitekniikan laitos**

Kandidaatintyö

Espoo 6.12.2012

**Vastuuopettaja:**

TkT Pekka Forsman

**Työn ohjaaja:**

DI Mika Strömman

Aalto-yliopisto
Sähkötekniikan korkeakoulu

Aalto-yliopisto
Sähkötekniikan korkeakoulu

kandidaatintyön
tiivistelmä

Tekijä: Ian Tuomi

Työn nimi: Logiikkakaavioiden generointi tehdassuunnittelussa

Päivämäärä: 6.12.2012 Kieli: Suomi Sivumäärä: 6+19

Tutkinto-ohjelma: Automaatio- ja systeemitekniikka

Vastuuopettaja: TkT Pekka Forsman

Ohjaaja: DI Mika Strömman

Työssä esitellään ohjauslogiikan automaatiosuunnitteluun liittyviä malleja, kuvauksia ja metamalleja. Lisäksi tarkastellaan suunnittelun työtapoja ja logiikan kaaviomuotoisia esityksiä.
Esteettisen kaavion tunnusmerkit määritellään. Lisäksi esitellään graafinasettelualgoritmi, jonka avulla voidaan tuottaa näitä tunnusmerkkejä noudattavia kaavioita tehdassuunnittelun tarpeisiin.
Lisäksi pohditaan kuvaillun järjestelmän käytöstä seuraavia hyötyjä ja haasteita, ja esitetään esimerkkejä algoritmin tuloksena syntyneistä kaavioista.

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

Aalto-university
School of Electrical Engineering

abstract of the
bachelor's thesis

Author: Ian Tuomi

Title: Logic Diagram Generation in Factory Design

Date: 6.12.2012 Language: Finnish Number of pages: 6+19

Degree program: Automation and Systems Technology

Supervisor: Pekka Forsman, D.Sc.

Instructor: Mika Strömman, M.Sc.

This work presents models, descriptions and frameworks dealing with the design of automation control system logic. In addition, the practices of automation design work and diagrammatic representations of logic are discussed.
The properties of an aesthetic diagram are defined. A graph layout algorithm capable of generating aesthetic diagrams for the purposes of control system design is described in detail.
In addition, the pros and cons following from the use of such a system are discussed, and examples of layouted diagrams are presented.

Keywords: Automation system, Automated design, Programmable Logic, PLC, IEC 61131-3, FBD, Graph layout, Directed Graph

# Esipuhe

On joitain ihmisiä, joita ilman tämä työ ei olisi toteutunut, ja monia joita ilman se olisi ollut vaikeampaa. Haluan kiittää näistä erityisesti seuraavia ihmisiä:

Työn ohjaajaa Mika Strömmania tarkoista huomioista, rakentavista keskusteluista ja asiantuntevasta ohjauksesta.

Petri Kokkoa ja Pekka Niemistä johdatuksesta aiheen piiriin työpaikalla, suunnannäytöstä ja ideoista.

Kandiryhmäni 6BÄKin mainioita jäseniä Peter 'Bembu' Kronströmiä, Lauri Vepsäläistä, Heikki 'Taffis' Tahvanaista, Panu Kauppista ja Lauri Andleria vertaistuesta ja hyvästä yhteishengestä.

Kämppiksiäni Emmaa, Emmiä, Ennaa, Katria, Niinaa, Nikkeä, Ollia, Sariannaa, Spördeä, Teemua, Tuomasta ja Vesaa arjen merkittävästä parantamisesta sekä juttuja kahviseurasta.

Tyttöystävääni Marjoa kärsivällisyydestä työn aiheuttamien kiireiden aikana.

Vanhempiani tuesta ja kannustuksesta.

Otaniemessä itsenäisyyspäivänä 6.12.2012

Ian Tuomi

# Sisältö

# Lyhenteet

| | |
|---|---|
| DCS | Distributed Control System<br>Hajautettu ohjausjärjestelmä |
| DFS | Depth-first search<br>Syvyyssuuntainen läpikäynti |
| FBD | Function Block Diagram<br>IEC 61131-3 standardin määrittelemä ohjelmointikieli |
| IEC | International Electrotechnical Commission,<br>Kansainvälinen sähköalan standarointiorganisaatio |
| NP | Nondeterministic Polynomial time<br>Epädeterministisellä Turingin koneella polynomiaalisessa ajassa ratkeavien ongelmien joukko |
| PI | Prosessien instrumentointi |
| PLC | Programmable Logic Circuit<br>Ohjelmoitava logiikkapiiri |
| XML | Extensible Markup Language<br>Merkintäkieli, johon voidaan sisällyttää ja jolla voidaan jäsentää tietoa |
| XSL | Extensible Stylesheet Language<br>Kieliperhe, jolla voidaan määritellä ja muuntaa XML-formaatteja |

# 1 Johdanto

Teollista automaatiojärjestelmää suunnitteltaessa tuotetaan suuri määrä erilaisia dokumentteja, jotka yhdessä muodostavat järjestelmän toteutuksen, käyttöönoton ja ylläpidon mahdollistavan kuvauksen. Dokumentaation laatiminen käsin on suurissa projekteissa työlästä ja tehotonta. Nykyaikaisessa automaatio- ja instrumentointisuunnittelussa ongelma korostuu [1]. Ratkaisuista on tullut monimutkaisempia, vaaditut toteutusajat ovat lyhentyneet ja laatu- ja joustavuusvaatimukset ovat lisääntyneet.

Näistä vaatimuksista johtuen teolliset toimijat ovat pyrkineet tehostamaan työtä automatisoiduilla apuvälineillä. Metodiikka ei kuitenkaan ole sen ilmeisistä eduista huolimatta yhtenäistynyt. Automaatiosuunnittelu akateemisena tutkimuskenttänä on lisäksi hajanainen ja yhteistyö on vähäistä. Vaikka lähestymistavat ovat vaihtelevia, suunnittelussa on nähtävissä yhtenäinen siirtymä kohti jaettuja tietomalleja. Käytännössä jaettujen tietomallien käyttö tarkoittaa, että suunnittelujärjestelmään tallentuu projektin edetessä kaikki sähköistys ja instrumentointi automaatiojärjestelmineen ja kenttälaitteineen. Tietomalliin perustuva lähestymistapa vaatii sen laajuuden ja rajoitukset määrittelevän metamallin. Se ohjaa ja määrittää suunnittelun kohteen lisäksi myös yleisesti suunnittelutöiden kulkua.

Erilaiset kaaviot voidaan usein tuottaa suoraan tietomallin sisältävästä tietokannasta. Tämän kandidaatintyön kohteena on ohjausjärjestelmän logiikan 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 voidaan hyödyntää algoritmillisen graafiasettelun tuloksia.

Kaavioiden asetteluun käsin kuluu huomattavan paljon aikaa silloin, kun tavoitteena on mahdollisimman luettava kaavio. Klausken ja Dziobekin mukaan kaavioiden asetteluun kuluu ajasta n. 30%. Lisäksi he arvioivat, että kaavioiden asettelulla saavutettavan luettavuuden myötä voitaisiin toimilohkoihin perustuvaan suunnitteluja ohjelmointityöhön käytettävää työaikaa lyhentää parhaassa tapauksessa puoleen nykyisestä. [2] Kun tietokoneen avulla toteutettava algoritmillinen muunnos kuvauksesta kaaviomuotoon onnistuu ilman suunnittelijalle siitä koituvaa vaivaa tai aikaa, kannattaa suunnittelutyön kohdistua kuvauksen laatimiseen.

Hierarkinen graafiasettelu soveltuu erityisen hyvin logiikkakaavioiden kaltaisten tietovirtakaavioiden asetteluun. Tässä työssä eritellään hierarkisen graafiasettelun idea, sen vaiheet ja niihin liittyvät ongelmat, sekä algoritmillisia ratkaisuja kyseisiin ongelmiin. Työssä esitellään myös algoritmin tuottamia asetteluja.

# 2 Tehtaan mallit

Tehdasmalli on kuvaus, joka mahdollistaa tehtaan toteutuksen, käyttöönoton ja ylläpidon. Se on tietomalli, joka kuvaa tehtaan toimintaa, sen prosessia ja laajuudestaan riippuen myös sen ihmisten toimintaa ja organisaatiota. Pohjimmiltaan se koostuu tehdasobjekteista ominaisuuksineen ja niitten välisistä relaatioista. [1]

Suunnittelijoilla on erilaisten työtehtäviensä myötä erilaisia tarpeita myös tehdasmallin suhteen. Työn kohde on monitahoinen, eikä mikään yksi esitystapa 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 se 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 muttei määrää sen tarkkaa teknistä toteutusta. [4]

Tätä näkymää määritettäessä eri alojen asiantuntijat voivat osallistua työn suunnitteluun ilman, että heillä on tuntemusta instrumentoinnista tai ohjelmistokehityksestä. Kun päätökset on tehty toiminnalisella tasolla, voidaan hankinnat räätälöidä sen mukaisesti ja 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. [1]

### 2.1.2 Fyysinen näkymä

Fyysiseen näkymään suunnitellaan järjestelmän johdotukset, kaapelit, kenttäväylät, ohjauskeskukset, toimilaitteiden sijainnit ja ylipäänsä kaikki, mitä vaaditaan järjestelmän fyysiseen käyttöönottoon ja ylläpitoon. Sen käsittely ei kuulu tämän työn laajuteen.

### 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 niitten integroimiseksi suunnitteluprosessiin entistä vahvemmin on ollut. Toteutustapa jää silloin yksittäisten suunnittelijoiden määriteltäväksi, mistä seuraa, että yhtenäistä työtapaa on harvoin. Toteutetun ohjelmakoodin uudelleenkäyttö jää myös tyypillisesti vähäiseksi. [5]

## 2.2 Logiikan kuvaukset

Logiikan kuvaukset helpottavat tehtaan toiminnan ymmärtämistä suunnittelu- ja rakennusvaiheen lisäksi tehtaan toiminnan ja huollon yhteydessä ja ovat siksi tärkeä osa dokumentaatiota. Teollisessa automaatiosuunnittelussa sallitut logiikan esitystavat on määritelty erityisesti selkeyttä, turvallisuutta ja ennustettavaa toimintaa silmällä pitäen. Joskus loogista järjestelmää voidaan kuvailla kertomalla esimerkinomaisesti sen toiminnasta eri olosuhteissa. Tällainen lähestymistapa ei kuitenkaan yleensä ole riittävä. Lisäksi määritellään yleensä ainakin muuttujia, säätöpiirejä, sekvenssejä sekä niitten välisiä kytkentöjä.[1] Jatkuva prosessiohjaus voidaan esittää selkeästi järjestelmän PI-kaavioissa, mutta diskreetin ohjauksen kuvaamiseen tarvitaan erilaisia esityksiä, kuten logiikkadiagrammeja [6]. Logiikkadiagrammit kuvaavat syöttöjen ja lähtöjen välistä logiikkaa.

### 2.2.1 Toimintakaaviot

Toimintakaavio on funktionaalisen näkymän toteutus, joka tavallisesti valmistellaan järjestelmän suunnittelun varhaisessa vaiheessa prosessikaaviosta. Se määrittelee oh-

jausjärjestelmän toiminnot, ja sitä pidetään ajantasaisena järjestelmän suunnittelun edetessä. Se toimii suunnittelun apuna ja on lopulta osa lopullisen järjestelmän ohjeistusta.

Nykyaikaisessa automaation ohjelmistosuunnittelussa käytetään usein ohjelmointikieliä, jotka ovat muodoltaan lähellä toiminnallisia logiikan kuvauksia ja dokumentoivat itsensä hyvin. Tämän vuoksi toiminnallinen kuvaus usein jätetään tekemättä, jolloin syntyy säästöjä. [1]

### 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 kaikkiin PLC-ohjelmoinnin tarkoituksiin. Se sisältää viisi erilaista ohjelmointikieltä, joista tässä työssä käsitellään erityisesti FBD:tä. [7]

FBD-kieli koostuu toimilohkoista ja niiden välisistä langoista. Toimilohkot kuvaavat toisistaan erillisiä itsenäisiä laskennallisia yksiköitä, ja langat kuvaavat niitten relaatioita toisiinsa. Toimilohko lähettää tilastaan riippuen tapahtumia jotka vaikuttavat muitten toimilohkojen toimintaan. Sen tekemien laskutoimitusten tulos riippuu sen vasemmalta puolelta tulevista syöttötiedoista ja sen suorittaman laskennan tulokset lähetetään toimilohkon oikealta puolelta. Toisiinsa liitetyt toimilohkot muodostavat verkon, joka määrittelee laajemman toiminnallisuuden. [8]

Toimilohkojen muokkaamiseen on olemassa useita työkaluja. Näistä mainitsemisen arvoisia ovat ainakin FBDK, 4DIAC, ISaGRAF ja NxtControl. Ne eivät kuitenkaan tarjoa kaavioiden asetteluun muuta kuin yksinkertaisia työkaluja. [9]

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. [7]

Suoritusjärjestysvaatimus asettaa kappaleessa 3 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

Logiikka tallennetaan jaettuun suunnittelumalliin, johon kaikilla järjestelmän suunnittelijoilla on yhteys. Tavan, jolla logiikka tallennetaan, täytyy olla yhdenmukainen, jotta suunnittelutyössä voidaan tehdä tehokasta yhteistyötä työpaikalla jaettujen apuvälineiden avulla. Lisäksi kuvauksen tulee olla formaali, jotta logiikkaa

voitaisiin käsitellä algoritmillisesti. Formaalilla tarkoitetaan tietynlaista, tarkkaan määritetyn syntaksin mukaista kuvausta. Tällöin tulee mahdolliseksi toteuttaa tietoa käsitteleviä muunnosalgoritmeja. Muunnoksella tarkoitetaan jonkin kuvauksen tulkitsemista ja kuvaamista uudelleen eri tavalla.

Standardin IEC 61131-3 mukaisten ohjelmointikielten kuvaamiseen on ehdotettu PLCopen-standardia. Sen mukainen logiikan XML-muotoinen kuvaus voi sisältää toimilohkon toiminnallisuuden kannalta välttämättömien tietojen lisäksi toimilohkon sijainnin, koon ja jopa lankojen reititykset. Esitys voidaan tällöin laatia siten, että se on tuotettavan kaavion asettelun kannalta yksiselitteinen. Tällaista kuvausta voidaan pitää dokumentaation lopullisena muotona. [10]

Kun logiikkakuvauksen muoto on määritelty tarkasti, voidaan määritelmän mukaisesti laadittu logiikkakuvaus muuntaa määritelmään perustuvien 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 sisältämään tietoon perustuen voidaan tuottaa minkä tahansa muotoisia dokumentteja käyttäen hyväksi XSL-muunnoksia [11]. Eri XML-pohjaisia tiedostoformaatteja käyttävien suunnitteluvälineiden välinen integraatio on tällöin helposti toteutettavissa.

Olemassaolevien suunnitteluohjelmistojen varaan rakennettu suunnittelujärjestelmä voi kuitenkin kohdata haasteita, jos luotetaan liikaa ohjelmien standardinmukaisuuteen. Valmistajat jättävät usein standardinmukaisiksi kutsutuissa ohjelmistoissa määrittelyn mukaisia ominaisuuksia toteuttamatta. Tämä johtuu siitä, että ohjelma on määritelty standardinmukaiseksi, jos se toteuttaa siitä jonkin osan ja mainitsee, mitä se jättää toteuttamatta. [5]

# 3 Graafit ja niiden asettelu

## 3.1 Määritelmiä

Graafi tai verkko 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 $\mathfrak{L} = (L_1, L_2, \ldots, L_h)$, 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 graafin määritelmää sisällyttämällä siihen 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 lähtöportti. Toisin sanoen $e = (p_1, p_2)$, 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 $(v_1, v_2, \ldots, v_k)$, 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 useammin 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 kuuluvat 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 sen sisältämä yksittäinen lanka ilman, että sen sisältämän solmujoukon $E$ koko muuttuu. Kun virityspuusta poistetaan lanka, jakaantuu puu kahdeksi.

Graafin *asettelu* on tieto jokaisen solmun suhteellisista koordinaateista ja koosta, sen porttien sijainneista ja lankojen piirtoreiteistä. Asettelun tarkka esitystapa on sovelluksesta riippuvainen.

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

## 3.2 Tulosten arviointiperusteet

Asettelun esteettisyydellä tarkoitetaan ominaisuuksia, jotka parantavat sen luettavuutta. Purchase ym. [12] ovat tutkineet esteettisyyteen vaikuttavia tekijöitä ja määritelleet asettelun luettavuudelle tärkeimmiksi seuraavat ominaisuudet:

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 tekemään mutkia. Valmiin kaavion asettelua voidaan arvioida laskemalla kaavioista esteettisyyteen vaikuttavia tekijöitä automatisoidusti.

## 3.3 Liittyvä työ

Suunnattuja graafeja käytetään laajalti riippuvuussuhteiden mallintamiseen. Esimerkkejä suunnatuista graafeista ovat projektinhallintaverkot, ontologioiden määrittelykaaviot, ohjelmien kutsupuut, syy-seurausdiagrammit sekä aiemmin osioissa 2.2.1 ja 2.2.2 käsittellyt toimintakaaviot ja FBD-kieliset kaaviot.

Suurin osa asettelualgoritmeista on suunniteltu geneerisille graafeille. Ne pyrkivät tuottamaan keskimäärin hyvän asettelun, oli graafi millainen hyvänsä. Suunnatun verkon rakenteen huomioon ottavat algoritmit tuottavat parempia tuloksia. Tällaisia ovat kerrostetut asettelumenetelmät.

Kerroksittainen tapa esittää syklittömiä ja suunnattuja verkkoja on intuitiivinen ja helppolukuinen. Alkuperäinen ajatus graafien kerrostamisesta voidaan johtaa Warfieldin [13] ja Carpanon [14] tekemään työhön. Ylivoimaisesti suosituin tapa lähestyä suunnattujen graafien piirtoa on Sugiyaman menetelmä, joka jakaa ongelman vaiheisiin [15]. Sugiyaman ym. määrittelemä jaottelu on säilynyt hierarkkisen graafinpiirron tutkimuskentässä ajankohtaisena, vaikkei sen esittämiä algoritmeja juuri käytetäkään. Kun jokaista vaihetta voidaan tutkia erillisenä ongelmana, tutkimustyö helpottuu. Sitä onkin kehitetty valtavasti eteenpäin ja siihen on tehty uusia käyttötarkoituksia palvelevia laajennuksia.

Laajoja käytännön tutkimuksia tietovirtakaavioiden asettelusta ovat tehneet Klauske ym. [16]. Tietovirtakaavioiden toimilohkot sisältävät useita portteja, joiden keskinäinen järjestys on säilytettävä. Lisäksi langat ovat tyypillisesti ortogonaalisia,

jolloin algoritmia pitää täydentää erillisellä langoille tarkoitetulla reititysalgoritmilla. Työvaiheista kerrostus ja risteysten vähennys saattavat muuttaa ohjelman suoritusjärjestystä. Tällöin algoritmin toimintaa pitää yksinkertaisesti rajoittaa järjestysherkkien blokkien osalta. Koska algoritmit ovat heuristisia, ovat niihin 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 niitä halutaan käyttää. Sivuporttien huomioonottoon on esitetty ratkaisu, jossa lankojen reititystä varten asetetaan valesolmuja. [17]

## 3.4 Asettelualgoritmi

Sugiyaman menetelmään perustuva asettelu koostuu viidestä eri vaiheesta, jotka on esitetty myös kuvassa 1.

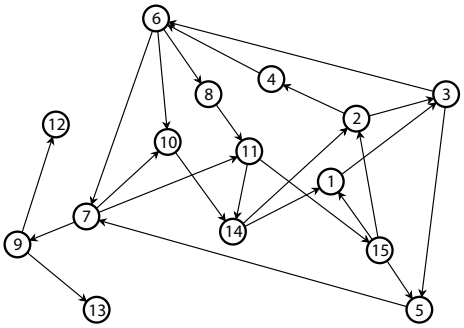
1. Syklien poisto, seuraa DFS-hakua [19] tai Eadesin ym. ahnetta algoritmia [20].
2. Kerrostus, seuraa Gansnerin ym. lankapituuden minimimoivaa menetelmää [21].
3. Risteysten vähentäminen, seuraa Sugiyaman ym. alkuperäistä menetelmää [15].
4. Poikittaisasettelu, seuraa Sanderin lineaaristen segmenttien menetelmää [22].
5. Lankareititys, seuraa Sanderin monilankareititysalgoritmia [23].

Asettelussa käytettävät algoritmit on valittu ottaen huomioon datavirtauskaavioiden rajoitukset ja rakenteelliset ominaisuudet. 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 ajaa pelkkä viides osio. Seuraavaksi esitellään jokaisen vaiheen tavoitteet, niihin liittyvää tutkimustyötä sekä kuvataan algoritmit työhön sovellettavilta osin.

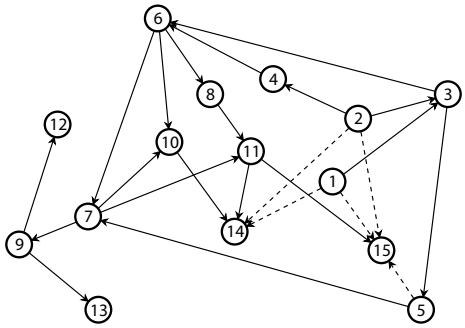
### 3.4.1 Syklien poisto

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 ne rikotaan kääntämällä yksi tai useampi syklin sisältämistä langoista.

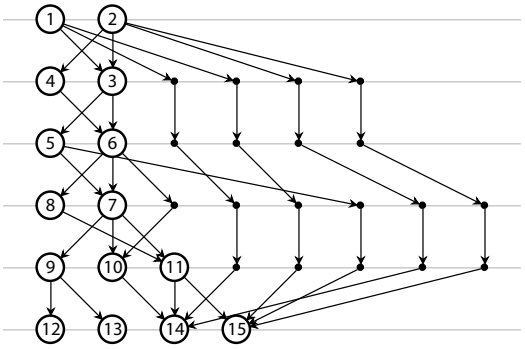
Tarkemmin määriteltynä tavoite on löytää mahdollisimman pieni lankajoukko $FS \subset E$, jonka sisältämät langat kääntämällä verkko on syklitön. Tämä ongelman on



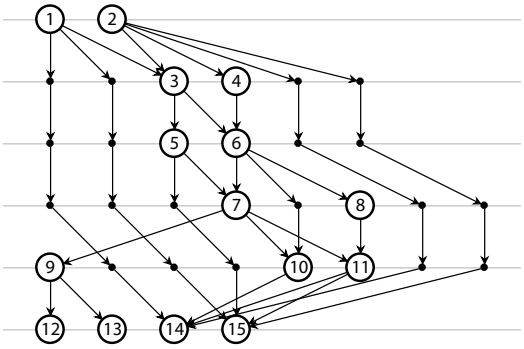
(a) Alkuperäinen graafi G



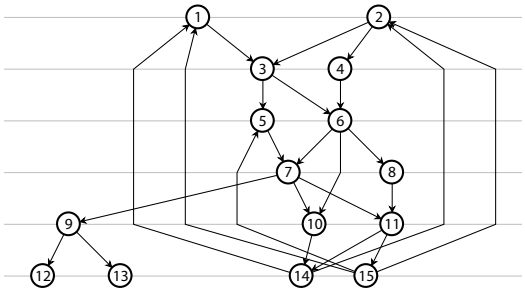
(b) Syklinpoisto



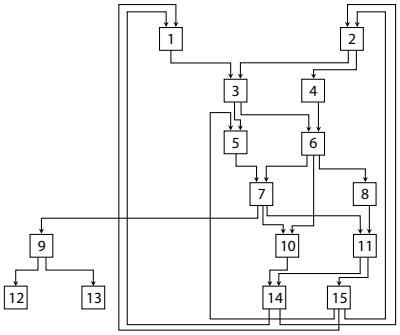
(c) Kerrostus



(d) Risteysten vähentäminen



(e) Poikittainen asettelu, lankojen kääntö



(f) Lankareititys

Kuva 1: Sugiyaman menetelmän eri vaiheet (Healy ja Nikolov [18], lankareititys lisätty).

osoitettu olevan NP-vaikea [24].

Syklien poistovaiheessa käsiteltävä tietomalli sisältää tiedon jokaisessa puussa ensimmäisenä suoritettavista solmuista. Tämä mahdollistaa huomattavan yksinkertaisten algoritmien käytön. Yksi tällaisista menetelmistä mahdollisimman pienen lankajoukon $FS$ etsimiseen on DFS-haku. DFS-haussa edetään ensimmäisenä suoritettavista solmuista mielivaltaista polkua pitkin ja käännetään matkalla kohdatut käänteiset langat. Tätä jatketaan, kunnes luotua polkua $(e_1, e_2, ..., e_h)$ ei voida enää jatkaa. Tämän jälkeen palataan takaisin lankaa $e_h$ pitkin ja jatketaan, kunnes löydetään solmu, josta lähtevää lankaa ei ole vielä seurattu. [19] 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-Shorin algoritmiin [25] perustuvaa ahnetta algoritmia [20].

### 3.4.2 Kerrostus

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

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

Tiiviyden äärimmäisenä rajoituksena toimii lopullisesta tarkastelutavasta riippuen näyttöpääte tai paperi, jolta aseteltua kaaviota on tarkoitus tarkastella. 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. [26]

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 lyhin etäisyys johonkin kerroksen $L_1$ solmuista. Pisimmän reitin tapa tuottaa mahdollisimman vähän kerroksia mutta saattaa tuottaa huomattavan suuria yksittäisiä kerroksia.

Kun kerrostuksesta halutaan mahdollisimman kapea, käytetään multiprosessoriajastusongelman ratkaisemiseen kehitettyä Coffmann-Grahamin algoritmia [27]. 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.

[4] eng. compact

Nämä kerrostusmenetelmät voivat olla hyödyllisiä sellaisissa sovelluksissa, joissa aseteltavan kaavion muodolla on erityisiä kapeus- tai pituusvaatimuksia. Näitä ei sovelluksessamme ole, vaan tavoitteena on yksinkertaisesti mahdollisimman tiivis asettelu. Koska tiiviys korreloi valesolmujen vähäisen määrän kanssa, voidaan lähestyä ongelmaa valesolmujen määrän minimoinnin kautta. On olemassa myös kerrostuksen laatimisen tapa, joka tekee tämän ja voidaan suorittaa polynomisessa ajassa. Se löydetään, kun otetaan käyttöön lineaarisen ohjelmoinnin optimointimenetelmä, jolla minimoidaan

$$f = \sum_{(p_1,p_2)\in E} (i - j - 1), \qquad \mathsf{n}(p_1) \in L_i \quad ja \quad \mathsf{n}(p_2) \in L_j \tag{1}$$

,

missä minimoitava arvo $f$ on samalla valesolmujen määrä.

Tälle lineaarisen ohjelmoinnin optimointiongelmalle on tehokas ratkaisutapa, joka perustuu niinkutsuttuun Network Simplex -algoritmiin [28]. Sen aikakompleksisuutta ei ole pystytty osoittamaan, mutta se on käytännössä osoittautunut suoritusajaltaan erittäin nopeaksi [21]. Menetelmässä muodostetaan ensin verkon virityspuu. Kun virityspuusta poistetaan lanka $e = (p_1, p_2)$, tuloksena syntyy kaksi toisistaan erillistä puuta $G_1$ ja $G_2$, joille $G_1 \cup G_2 = G_s$ ja $G_1 \cap G_2 = \emptyset$ ja joista molemmat sisältävät toisen katkaistun langan yhdistämistä solmuista $\mathsf{n}(p_1) \in G_1$ ja $\mathsf{n}(p_2) \in G_2$. Näitten kahden puun perusteella lasketaan jokaiselle langalle leikkausarvo. Leikkausarvo on määränpääsolmun sisältävästä puusta ja lähdesolmun sisältävään puuhun kulkevien lankojen pituudet, josta vähennetään päinvastaiseen suuntaan kulkevien lankojen pituudet. Negatiivinen leikkausarvo viittaa siihen, että kyseistä lankaa kannattaa todennäköisesti pidentää mahdollisimman paljon. Tuloksena saadusta kerrostuksesta muodostetaan uusi virityspuu. Menetelmää jatketaan edelleen, kunnes kaikki virityspuun leikkausarvot ovat positiivisia.

### 3.4.3 Risteyksien vähentäminen

Risteysten määrä on osoittautunut suurimmaksi yksittäiseksi tekijäksi graafin luettavuuden kannalta [12]. Graafin kokonaisristeysmäärän vähentäminen ei perustu solmujen tarkkoihin sijainteihin vaan niitten keskinäiseen järjestykseen. Kahden kerroksen ylitysongelmassa etsitään joukolle $L_i$ järjestys, joka minimoi risteysten määrän, kun joukko $L_{i+1}$ pysyy paikallaan. 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 [29].

Parhaan mahdollisen kombinaation löytämiseksi joudutaan yksinkertaisen ratkaisun puutteessa käyttämään heuristisia menetelmiä. Eräs suosittu lähestymistapa perustuu solmujen sijoittamiseen sen kohde- ja lähdesolmujen järjestyslukujen mediaaniin.

Tätä toistetaan graafin jokaiselle solmulle, kunnes järjestys ei enää muutu. Teorian kannalta mediaanimenetelmällä on useita hyviä ominaisuuksia. Sen avulla tuotetun ratkaisun risteysmäärän suhde parhaaseen mahdolliseen on vakio. Lisäksi algoritmi voidaan suorittaa polynomisessa ajassa. Käytännön kokeiluissa satunnaisilla graafeilla on kuitenkin todettu, ettei mediaaneihin perustuva menetelmä tuota parhaita tuloksia [30]. Battisti ym. suosittelevatkin eri menetelmiä yhdisteleviä hybridialgoritmeja, kuten mediaanimenetelmän ja satunnaisten tranpositioiden yhdistämistä [26].

### 3.4.4 Poikittainen asettelu

Jokainen kulma langassa aiheuttaa ylimääräisen rasitteen ihmisen hahmotuskyvylle [12]. Esteettistä lopputulosta tavoittelevan asettelualgoritmin tulee siis pyrkiä minimoimaan lankojen kulmien määrä.

Risteysten vähentämisvaiheessa saavutettu kombinatorinen ratkaisu pyritään säilyttämään poikittaisasetteluvaiheessa. Siinä missä kerrosten solmukombinaatiot vaikuttavat lankojen risteysten määrään, vaikuttaa niiden tarkka poikittainen asettelu langoissa olevien kulmien määrään. Solmujen välisten lankojen käännösten määrää pyritään vähentämään, jotta kaavion luettavuus paranee. Asettelun merkitys korostuu, kun lankoja on paljon, kuten automaation logiikkakaavioille on tyypillistä.

Risteysten vähennysvaiheessa laaditun järjestyksen pohjalta muodostetaan triviaali poikittaisasettelu, jota tasapainotetaan. Tasapainotus tapahtuu mallintamalla solmuja levossa olevina heilureina. Solmut ovat heilurin päässä oleva paino ja lanka painoa kannatteleva naru. Tasapainotettavaa kerrosta seuraava kerros pysyy kiinteänä paikoillaan.

Solmujen poikkeamat lasketaan ja vierekkäisten solmujen päällekkäisyyksien tapauksessa siirretään niitten muodostamia osioita kauemmaksi toisistaan, kunnes päällekkäisyyksiä ei enää esiinny.

$$D_p(e) = x(v_1) - x(v_2) \tag{2}$$

$$D_p(v) = \frac{\sum_{e \in E} D_p(e)}{indeg(v)} \tag{3}$$

Segmentin siirtymä on sen sisältämien solmujen siirtymien summa, ja segmenttijoukon siirtymä on sen kaikkien segmenttien siirtymien keskiarvo.

$$D_p(S) = \sum_{v \in L} D_p(v) \tag{4}$$

$$D_p(v) = \frac{\sum_{i \in 1,...,k} D_p(S_i)}{k} \tag{5}$$

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

### 3.4.5 Lankojen reititys

Tietovirtakaavioiden toimilohkoissa on useita portteja, ja niitten sisältämät lankareititykset ovat ortogonaalisia. Nämä rajoitukset tulee ottaa tässä kaaviopiirron viimeisessä vaiheessa huomioon. Gansner ym. esittivät porttiongelmaan ratkaisun, joka perustuu noodisijoitteluesityksen laajentamiseen porttien suhteellista sijaintia kuvaavalla tiedolla. [21]

Kun jokaiselle solmulle on tiedossa sen lopullinen sijainti, luodaan lankasegmenttejä siten, että ne liikkuvat solmujen koordinaattien välillä. Poikittaiset segmentit, jotka asetetaan tyhjään tilaan rivien $j$ ja $j + 1$ välille, saavat koordinaatikseen $j + 0.5$. Tämä toistetaan pystysuuntaisille segmenteille.

Kaikki samassa ruudussa tai sarakkeessa $k$ olevat segmentit kootaan ryhmäksi joukkoon $S$. Tavoitteena on levittää ne koordinaateille välillä $[k, k + 1]$. Sander ym. ovat esittäneet reititysongelman ratkaisun, jossa käsitellään yksitellen jokaisen ruudun ylittävien lankasegmenttien muodostaman ryhmän paikallista asettelua erillisenä graafina. [23] Menetelmässä luodaan jokaiselle ruudulle erillinen segmenttiylitysgraafi, johon sovelletaan tässäkin työssä kuvattuja kaavioasettelun menetelmiä.

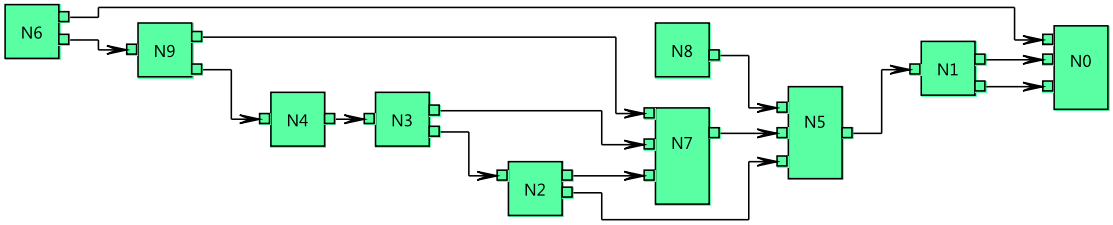
Risteysten minimoimiseksi luodaan segmenttiylitysgraafi jokaiselle ruudulle. Jokainen graafiin kuuluva segmentti vastaa yhtä segmenttiylitysgraafin solmua. Segmenttiylitysgraafin syklit vastaavat asettelussa olevia risteyksiä. Asyklinen segmenttiylitysgraafi ei silloin sisällä syklejä ja se voidaan järjestää topologisesti. Muussa tapauksessa sen sisältämät syklit rikotaan käyttäen syklinpoistoon soveltuvaa algoritmia. [20].

Lopuksi segmenttien sijoitus lasketaan syklittömästä segmenttiristeysgraafista. Segmentti, jonka sijoitus on $r \in \{1, ..., r_{max}\}$, saa koordinaatikseen $k+r/(r_{max}+1)$.

# 4 Esimerkkejä

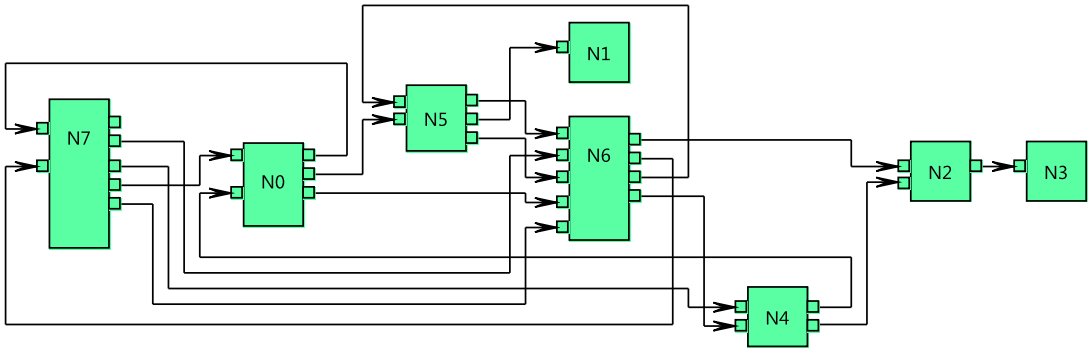
Menetelmän soveltuvuutta logiikkakaavioiden asetteluun tutkittiin luomalla satunnaisia graafeja, jotka aseteltiin KIELER-sovelluksen avulla. KIELER on Eclipse-projektin päälle rakennettu kokoelma mallipohjaisen suunnittelun apuvälineitä, jota kehitetään Kielin yliopistossa. [31] Se toteuttaa useimmat esitetyistä kaavioasettelun algoritmeista, eikä sitä tarvinnut muokata merkittävästi haluttujen tulosten saavuttamiseksi. Tässä esitellään kolme erityyppistä verkkoa, joiden kaltaisia algoritmi saattaisi kohdata.

Kuvassa 2 on yksinkertaiselle, yksi tai kaksi lähtöä sisältävien solmujen syklittömälle verkolle tehty asettelu. Suurin osa aseteltavista kaavioista on monimutkaisuudessaan todennäköisesti tätä luokkaa. Huomataan että algoritmi suoriutuu tehtävästä erinomaisesti.



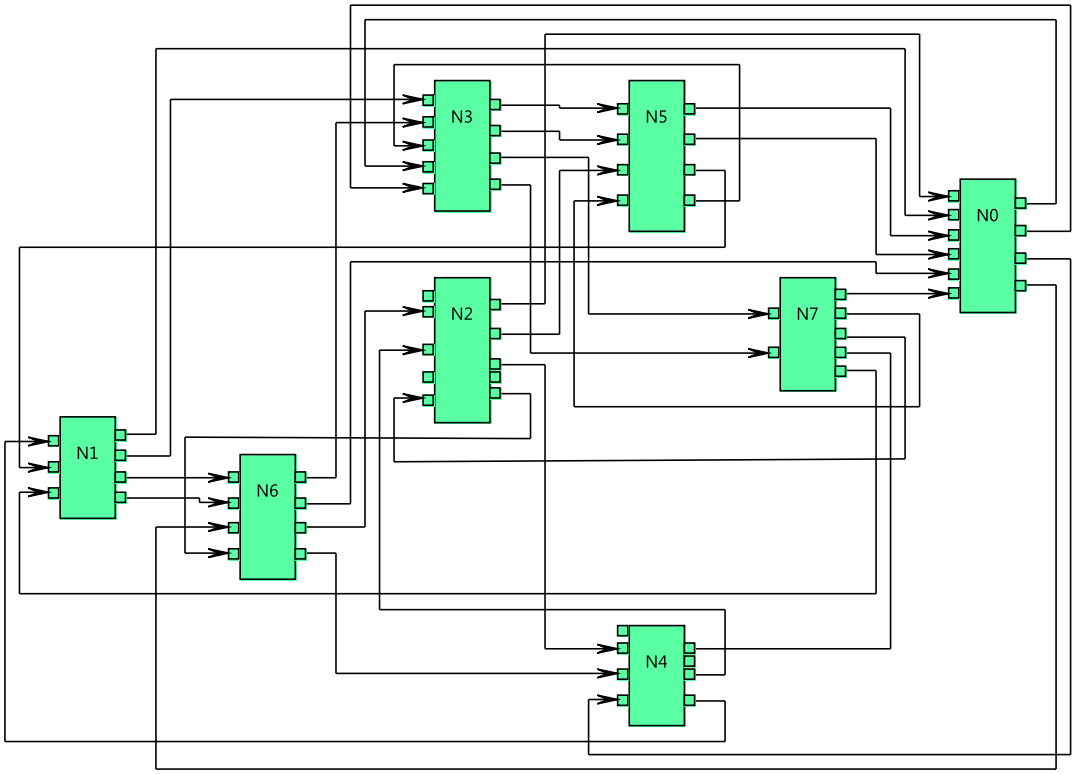

Kuva 2: Yksinkertaisen verkon asettelu. Toimilohkoissa on 1–2 lähtöä ja syklejä ei ole.

Kuvassa kolme nähdään paljon vaihtelua sisältävä verkko. Toimilohkoissa on lähtöjä välillä 0–4 ja takaisinkytkentöjä on paljon. Verkon kuvaama tietovirta ei ole sikäli todenmukainen, että siinä olevat yhteydet ovat täysin satunnaisia eivätkä luonnostaan mitenkään kerrostuneita. Algoritmi onnistuu kuitenkin luomaan jotakin järkeä täysin satunnaiseen kaavioon.




Kuva 3: Vaihtelua sisältävän verkon asettelu. Toimilohkoissa on 0–4 lähtöä.

Viimeinen esimerkki on huomattavan vaikea, eikä tuloksena odotettukaan kovin luettavaa esitystä. Jokaisessa toimilohkossa on neljä lähtöä, minkä seurauksena kuvassa on 32 yksittäistä ja täysin satunnaista lankaa. Valmiin kaavion luettavuus on jo merkittävän huono. Huomataan, että parhaatkaan algoritmit eivät suoriudu näin haastavasta tehtävästä. On kuitenkin kyseenalaista, pystyisikö ihminen näin mutkikkaan kaavion tapauksessa tuottamaan edes vastaavaa tulosta.




Kuva 4: Huomattavan monta lankaa sisältävän verkon asettelu. Jokaisesta toimilohkosta lähtee neljä lankaa

.

# 5 Yhteenveto

Automaatiojärjestelmien suunnittelu- ja ohjelmointityö on muuttunut nopeasti viime vuosikymmeninä, ja on todennäköistä että se muuttuu edelleen. Turhaa työtä pyritään vähentämään ja välttämätöntä työtä tehostamaan uusilla käytännöillä ja menetelmillä. Näitä ovat eri työvaiheiden- ja tehtävien saumaton yhteistyö jaettujen työkalujen ja mallien avulla, mallien uudelleenkäytettävyyden parantaminen sekä entistä yhtenäisempi suunnitteluprosessi.

Työvälineiden tehostamista on akateemisissa julkaisuissa kartoitettu runsaasti. Tutkimukset ovat kuitenkin olleet usein abstrakteja ja vieraannuttavia teollisille toimijoille. Tämä työ on pyrkinyt tarjoamaan ratkaisun, jolla suunnittelutyötä voidaan tehostaa, ja joka voidaan helposti lisätä jo olemassaoleviin järjestelmiin.

Työssä on osoitettu yhteys suunnattujen graafien ja logiikkadiagrammien välillä, sekä todettu niitten asetteluun kehitettyjen algoritmien käyttökelpoisuus teollisuuden automaatiosuunnittelussa. Esitelty algoritmi tuottaa hyviä tuloksia, ja on muunneltavissa edelleen erilaisiin tarpeisiin. Asettelualgoritmin käyttö erottaa varsinaisen logiikan sen kaaviomuotoisesta kuvauksesta. Sen lisäksi, että asetteluun ei kulu suunnittelijoilta aikaa, logiikan tietomalli helpottaa suunnittelua tehostavien muunnosten ja kirjastojen kehittämistä. Muunnosten avulla voidaan tietomallia muokata huomattavan nopeasti ja asettelun avulla tulokset ovat välittömästi asiakkaalle luettavassa muodossa.

Kaavioiden tuottaminen voidaan toteuttaa suunnittelijan käyttämällä tietokoneella tai keskitetyllä kaavioiden asettelupalvelimella. Tästä on erityisen paljon hyötyä sillion, kun suunnittelijoita on monta. Versioinnin tuottamilta haasteilta vältytään, ja sama työkalu voidaan tuoda kaikille suunnittelujärjestelmän käyttäjille yhdenaikaisesti.

# Viitteet


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

[2] L. K. Klauske and C. Dziobek. Improving modeling usability: Automated layout generation for simulink. In *Proceedings of the MathWorks Automotive Conference (MAC'10)*, 2010.

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

[4] R. Ajo. *Laatu automaatiossa: parhaat käytännöt*. Suomen automaatioseura, 2001.

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

[6] F. Meier. Control system documentation. In V. Trevathan, editor, *A Guide to the Automation Body of Knowledge*, pages 75–88. ISA, 2006.

[7] International Electrotechnical Commission. Iec 61131-3 programmable controllers part 3: Programming languages. Technical report, 1993.

[8] 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.

[9] M. Schmeling, R. von Hanxleden, D. I. M. Spönemann, and P. S. Roop. Effective visualization of iec 61499 function blocks. 2010.

[10] E. van der Wal. Introduction into IEC 1131-3 and PLCopen. In *The Application of IEC 61131 to Industrial Control*, pages 2/1–2/8. IET, 1999.

[11] J. Clark. Xsl transformations (xslt). *World Wide Web Consortium (W3C)*, 1999.

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

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

[14] 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.

[15] 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.

[16] 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.

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

[18] R. Tamassia. *Handbook of graph drawing and visualization*. Chapman & Hall/CRC, 2007.

[19] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[20] 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.

[21] 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.

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

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

[24] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, CA, 1979.

[25] B. Berger and P. W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 236–243. Society for Industrial and Applied Mathematics, 1990.

[26] 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.

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

[28] W. H. Cunningham. A network simplex method. *Mathematical Programming*, 11(1):105–116, 1976.

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

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

[31] Reinhard von Hanxleden, Hauke Fuhrmann, and Miro Spönemann. Kieler—the kiel integrated environment for layout eclipse rich client. In *Proceedings of the Design, Automation and Test in Europe University Booth (DATE'11)*, Grenoble, France, March 2011.