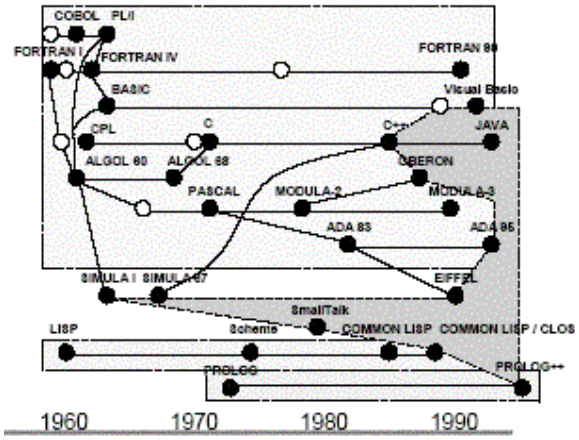
TEKNILLINEN KORKEAKOULU
Automaation tietotekniikka

Ohjelmointiparadigmat automaatiossa

Tuomas Nieminen



1. Tiivistelmä

Seminaariesitelmä liittyy diplomityöhöni ”Automaatio-ohjelmistojen arkkitehtuurit ja malliratkaisut”. Perusajatuksena on tutkia millaisia paradigmoja on käytössä IEC 61131-3 standardin määrittelemissä PLC-kielissä ja mitä muita paradigmoja voisi olla käytössä PLC-ohjelmoinnissa.

Seminaarissa esittelen myös lyhyesti yleisimmät paradigmat, IEC 61131-3 standardin ja mitä paradigmoja se noudattaa. Mukaan tulee myös lyhyt katsaus uudesta IEC 61499 standardista ja mitä uutta se tuo PLC ohjelmointiin.

2. Sisällysluettelo

3. Johdanto

Tämä seminaari liittyy diplomityöhöni ”Automaatio-ohjelmistojen arkkitehtuurit ja malliratkaisut”.

Seminaariesitelmässä käydään käydä läpi tärkeimmät paradigmat ja esitellä niitä, jotta lukija saisi käsityksen mitä paradigmalla oikein tarkoitetaan. Toisessa osassa käydään läpi IEC 61131-3 standardi ja esitellään mihin paradigmoihin sen ohjelmointikielet kuuluvat.

Alunperin idea tälläisen aiheen liittämisestä diplomityöhöni tuli ohjaajaltani Mika Strömmanilta. Perusteena on kokonaisvaltaisen kuvan saaminen automaatio-ohjelmointiin.

Työ on tehty kokonaan tutustumalla alan kirjallisuuteen ja www-sivustoihin ja tekemällä johtopäätöksiä niiden pohjalta. Työssä ei ole kokeellista osiota.

4. Ohjelmointiparadigmat

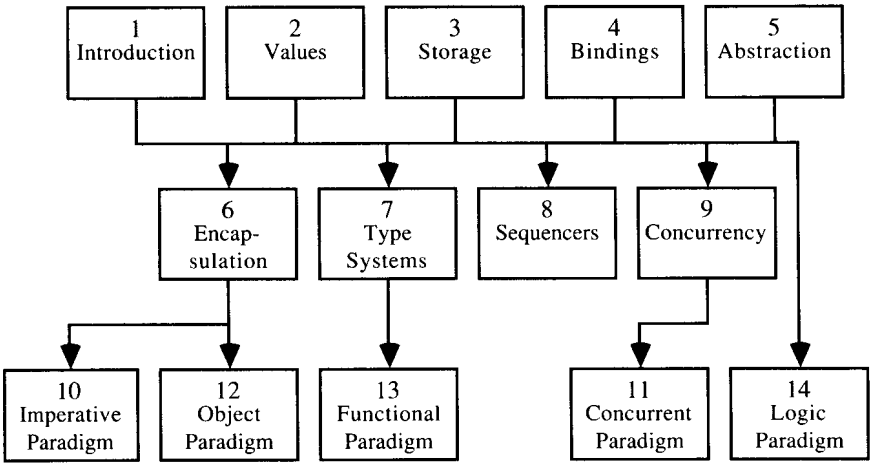
Itse sana paradigma tarkoittaa esikuvaa, mallia tai kaavaa. Ohjelmointiparadigma ei ole virallisesti määritelty termi mutta sitä käytetään kuvaamaan tapaa ja mallia ohjelmoida. [Wegner 1990] [Watt 1990]

Periaatteessa jokainen eri ohjelmointityyli on oma paradigmansa mutta yleensä niitä määritellään viisi: Olio(object)-, funktionaalinen(functional) -, logiikka(logic)-, imperatiivinen(imperative)- ja rinnakkainen(concurrent,paraller) ohjelmointi (kuva 1). Joissain yhteyksissä myös tietokannat(database) esitellään omana paradigmanaan. Paradigmoja on myös kehitelty lisää mutta ne eivät ole merkittävässä asemassa ja ne eivät sovellu automaatio-ohjelmointiin, joten en esittele niitä tässä. [Watt 1990] [Appleby 1997]

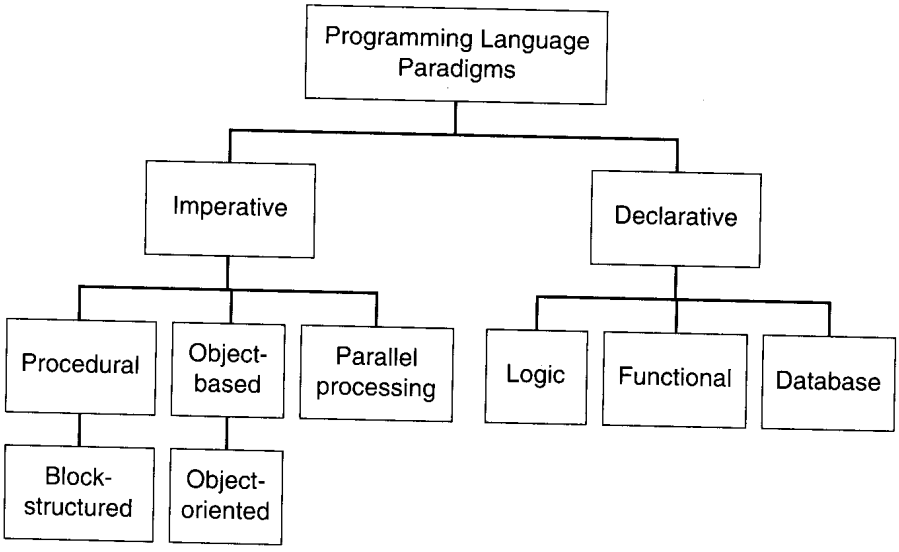
Tätä jakoa on myös kehitelty ja [Appleby 1997] esittää, että jako olisi kuvan 2 mukainen. Tässä imperatiivinen paradigma on siis proseduaalisen-, olio- ja rinnakkaisen ohjelmoinnin yläparadigma. Uutena paradigmana esitellään deklaratiivinen paradigma, joka on loogisen, funktionaalisen ja tietokantaohjelmoinnin yläparadigma. Proseduraaliset(procedual) ja lohkorakenteiset(block-structured) paradigmat ovat kuitenkin vain laajennuksia imperatiiviseen paradigmaan ja eivät tuo mitään fundamentaalisti uutta, minkä takia niitä pitäisi tämän työn laajuudessa esitellä erikseen.

Vaikka todetaan ohjelmointikielen kuuluvan johonkin paradigmaan, niin jako ei ole absoluuttinen, vaan monet kielet tukevat eri paradigmoja eri määrin. Ilmoitettua kieltä siis vain yleisimmin käytetään kyseisellä paradigmalla. Paradigmat voivat olla myös päällekkäisiä, eli olio-ohjelma on myös imperatiivinen. Kuitenkin imperatiivinen ohjelma ei välttämättä ole yksikään alaparadigmoistaan. Myös monilla imperatiivisilla paradigmoilla voi ohjelmoida deklaratiivisesti, toisinpäin tämä ei kuitenkaan yleensä onnistu.

Seuraavaksi esittelen paradigmat tarkemmin ja lopuksi selvennän millaisia paradigmoja käytetään automaatio-ohjelmoinnissa. Tietokantaohjelmointia en kuitenkaan esittele, koska se on niin kaukana seminaarin fokuksesta.



Kuva 1. Ohjelmointiparadigmat [Watt 1990]



Kuva 2. Ohjelmointiparadigmat [Appleby 1997]

4.1 Imperatiivinen ohjelmointi

Imperatiivinen ohjelmointiparadigma, on vanhin ja suosituin käytetyistä paradigmoista. Se perustuu nykyisin jo hyvinkin yksinkertaiseen ideaan, eli muuttujiin. Imperatiivisessa ohjelmoinnissa annetaan suoritusjärjestyksessä rivejä, joilla komennetaan konetta joko muuttamaan tai lukemaan muistipaikoissa olevia muuttujia. Nykyisin imperatiiviset kielet sisältävät myös paljon erilaisia muuttujien käsittelyyn soveltuvia komentoja, kuten silmukoita ja matemaattisia operaatioita.

Alkujaan paradigma kehittyi, kun ohjelmointikielten kehittäjät huomasivat, että muuttujat ja käskyt muodostavat muistihakujen ja koneen käskylistojen päivitysten yksinkertaisen mutta käytännöllisen abstraktion. Koska niillä on läheinen suhde koneen arkkitehtuuriin, niin ainakin teoriassa niiden toteuttaminen voidaan tehdä tehokkaasti. [Watt 1990]

Tärkein syy imperatiivisen ohjelmoinnin hyötyyn on ohjelmointityylin sopivuus oikean maailman mallintamiseen. Oikeassa maailmassa asiat monesti muuttuvat ajan funktiona ja imperatiivisen ohjelmoinnin muuttujat sopivat tämän mallintamiseen erittäin hyvin. Imperatiiviset kielet siis käyttävät muuttujia, joiden tiloja luetaan ja muutetaan. [Watt 1990] [Appleby 1997]

Imperatiiviseen paradigmaan sopivia ohjelmointikieliä ovat muun muassa Algol, PL/1, Pascal, C, Ada, Modula, Oberon. [Pascal]

4.2 Olio-ohjelmointi

Olio-ohjelmointi kehitettiin imperatiivisesta paradigmasta kun 1970 luvulla huomattiin, että globaaleja muuttujia voitiin muuttaa kaikkialta ja tämä hankaloitti isojen ohjelmistojen ylläpitoa kohtuuttomasti. Ratkaisuksi esitettiin tiedon piiloittamista, eli käyttäjä näkisi vain itselleen merkittävät muuttuja ja funktioiden rajapinnat, mikä nykyisin on olio-ohjelmoinnin kantavia periaatteita. [Watt 1990]

Muita tärkeitä periaatteita olioparadigmassa ovat olio (object), luokka (class) ja periytyminen (inheritance). Näiden virallinen määrittely on edelleen tekemättä, koska ne ovat niin monitahoisia käsitteitä mutta olen kerännyt tähän joitain määritelmiä.

”Object is a group of procedures that share a state” [Wegner 1988]

“An object is a software bundle of variables and related methods.” [javaolio]

“A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.” [javaluokka]

“Class is a collection of objects sharing the same attributes” [Appleby 1997]

“Inheritance is the ability to organize object classes into a hierarchy of subclasses and superclasses” [Watt 1990]

Periaatteessa olio on siis luokan ilmentymä(instance), joka sisältää muuttujia ja metodeja, joita voidaan kutsua muualta mutta niiden varsinainen toiminta on piiloitettu käyttäjältä. Yhdestä luokasta voi tehdä monta oliota ja jokainen niistä sisältää omat kopiot muuttujista eli ne ovat erillisiä. Luokka sisältää tiedot metodeista ja muuttujista, joiden pohjalta olio muodostetaan. Periytyminen on keino hallita luokkia ja saada niille samaa toiminnallisuutta.

Olioparadigma siis mahdollistaa isojen ohjelmistojen pilkkomisen pienempiin ja helpommin hallittaviin osasiin. Se myös mahdollistaa turhan informaation piilottamisen käyttäjältä ja samalla sisältäen imperatiivisen paradigman muuttujien käytön.

Olio paradigmaan sopivia ohjelmointikieliä ovat muun muassa Java, C++, Smalltalk, Eiffel, Simula 67. [Pascal]

4.3 Rinnakkainen ohjelmointi

Rinnakkainen ohjelmointi on laajennus muihin paradigmoihin mutta yleensä sitä käytetään imperatiivisten kielien kanssa. Tämän takia se monasti mielletään imperatiivisen paradigman aliparadigmaksi vaikka tämä ei itseasiassa täysin pidä paikkansa, koska ei ole mitään periaatteellista estettä käyttää sitä deklaratiivisten kielten kanssa. [Appleby 1997]

Rinnakkainen ohjelmointi voi tarkoittaa kahta asiaa. Joko käytössä on yksi prosessori, joka vuorotellen ajaa kahta eri prosessia tai sitten käytössä on monta prosessoria, joista jokainen ajaa omaa prosessiaan. [Watt 1990] [Appleby 1997]

Voitaisiin ehkä puhua rinnakkaisesta ohjelmoinnista ja aidosti rinnakkaisesta ohjelmoinnista mutta mitään virallista termistöä asialle ei ole. Englannissa käytössä on kaksi termiä concurrent ja paraller, joista concurrent tarkoittaa yhtä prosessoria ja paraller monta prosessoria. Tästä jaosta näytetään myös lipsuvan ja monasti yhden prosessorin systeemit näyttävät olevan paraller tai toisinpäin. Joten kannattaa olla aika tarkkana kun asiaa koskevaa tekstiä lukee.

Rinnakkaiseen paradigmaan sopivia kieliä ovat CSP, Concurrent Pascal, Ada, Argus, Concurrent C, Java [Pascal]

4.4 Deklaratiivinen ohjelmointi

Deklaratiivinen on toinen käytössä oleva perusparadigma. Tämä ei ole niin selvästi yhtenäinen, kuin imperatiivinen paradigma johtuen siitä, että tämä yläparadigma määritettiin vasta myöhemmin kuin aliparadigmansa. Eli funktionaalinen-, looginen- ja tietokanta ohjelmointi kehitettiin ensin ja sitten huomattiin, että niillä on yhdistäviä tekijöitä.[Appleby 1997] [Watt 1990] [Heinonen 1996]

Deklaratiivisen ohjelmoinnin perusidea on toimia käytössä olevien asioiden esittelynä, eli jos imperatiivisissa kielissä komennetaan niin tässä esitellään.[Ohjelmistotekniikka]

[Glaser 1984] esitellään alunperin funkitionaaliseen ohjelmointiin kehitetty esimerkki, joka minusta kuvaa hyvin koko deklaratiivista ohjelmointia.

Vajan rakennus imperatiivisesti

1. Laske perustus.
2. Rakenna seinät.
3. Tee lattia.
4. Pystytä katto.

Deklaratiivisesti sama olisi

1. Vaja koostuu seinistä, joita perustukset tukevat,
2. lattiasta, jota perustukset tukevat ja
3. katosta, jota seinät tukevat

Tästä selkeästi huomaa, ettei deklaratiivisessa kielessä määritetä erikseen miten vaja tulee rakentaa vaan mistä se koostuu ja miten osat liittyvät toisiinsa. Deklaratiiviset kielet ovat siis korkeamman abstraktiotason omaavia kieliä, kuin imperatiiviset. Kääntäessä tämä ero kuitenkin häviää, koska kone ei ymmärrä kuin muutamaa yksinkertaista käskyä.

Deklaratiiviset kielet eivät ole saavuttaneet imperatiivisten kielten suosiota mutta ovat silti joillakin erityisaloilla hyvin tärkeitä, kuten tekoälytutkimuksessa ja matematiikassa. Syitä suosion puuttumiseen on useita mutta itse nostaisin kaksi ylitse muiden, tottumus ja muuttujien puute. Eli uudet ohjelmoijat yleensä koulutetaan imperatiivisilla kielillä ja ajattelutavan vaihto deklaratiiviseen ei ole kovin helppo. Muuttujat soveltuvat todellisuuden mallintamiseen hyvin ja monien asioiden tekeminen ilman niitä on hankalaa.

4.5 Funktionaalinen ohjelmointi

Funktionaalisessa ohjelmoinnissa asiat pyritään esittämään matemaattisten funktioiden syntaksissa, eli ohjelma koostuu funktioista. Funktiot tarvitsevat kuvauksen toiminnallisuudestaan ja funktion argumenttien ja palautusarvojen tyypit pitää määrittää. Eli syötteenä saadaan tietyt atribuutit, jotka sijoitetaan käytettyyn funktioon ja niistä evaluoidaan palautusarvot. [Heinonen 1996]

Periaatteessa vaikka C:llä voi tehdä funktionaalista ohjelmointia mutta siinä joutuu ottamaan itse huomioon muuttujien käsittelyn, puhtaasti funktionaalinen kieli abstraktoi nämä pois.

Funktionaaliseen paradigmaan soveltuvia kieliä ovat muunmuassa Haskell, Lisp, Scheme ja ML [Pascal]

4.6 Logiikkaohjelmointi

Logiikkaohjelmointi perustuu puhtaasti logiikan soveltamiseen ohjelmointikielenä. Logiikkaohjelmointi on tiedon ja sen suhteiden esittämistä loogisina päättelyketjuina. Tämä on jäänyt pitkälle akateemiseksi huviksi, eikä oikeastaan minkäänlaisia sovelluksia löydy. [Pascal] [Appleby 1997]

Laskennallisesti loogiset ohjelmat ovat tehottomia mutta laajojen tietomäärien loogisten riippuvuuksien hallintaan se sopii. [Pascal]

Loogiseen paradigmaan sopivia kieliä ovat muunmuassa Lisp ja Prolog. [Pascal]

5. IEC 61131-3 standardi

IEC 61131-3 standardi määrittää viisi kieltä ohjelmoitavien logiikoiden(PLC) ohjelmointiin. Nykyisin nämä tai näistä vähän kehitellyt versiot ovat yleisesti käytössä, joten tutkimus keskittyy näitten kielten paradigmojen määrittämiseen.

Tässä ei käydä koko standardia läpi vaan esitellään tutkittavat kielet ja niiden periaatteet, joiden pohjalta lukija saa yleiskuvan PLC-ohjelmoinnista.

Kaikista kielistä pitää todeta ettei niistä mikään ole sen parempi kuin toinen tai varsinaisesti sovellu vain tiettyyn tarkoitukseen. Niiden käyttö on täysin tottumiskysymys. Pitää kuitenkin huomioida etteivät ne ole suoraan siirrettäviä, eli käskylistalla tehtyä ohjelmaa ei voi koneellisesti kääntää toimilohkokaavioksi.

5.1 IL – Instruction List

Käskylista on lista peräkkäisiä käskyjä, joita kone suorittaa järjestyksessä. Jokainen käsky koostuu kahdesta osasta, operaattorista ja sitä seuraavista operandeista. Operaattori on käytettävä funktio ja operandit ovat funktion parametreiksi saamia muuttujia. Jokaisen listan aluksi jokin muuttuja talletetaan rekisteriin ja listan loput käskyt käyttävät tai muuttavat tätä muuttujaa. [Lewis 1998] [Bonfatti 1999]

```
LD plc_tila
ST liikuta_hihnaa
```

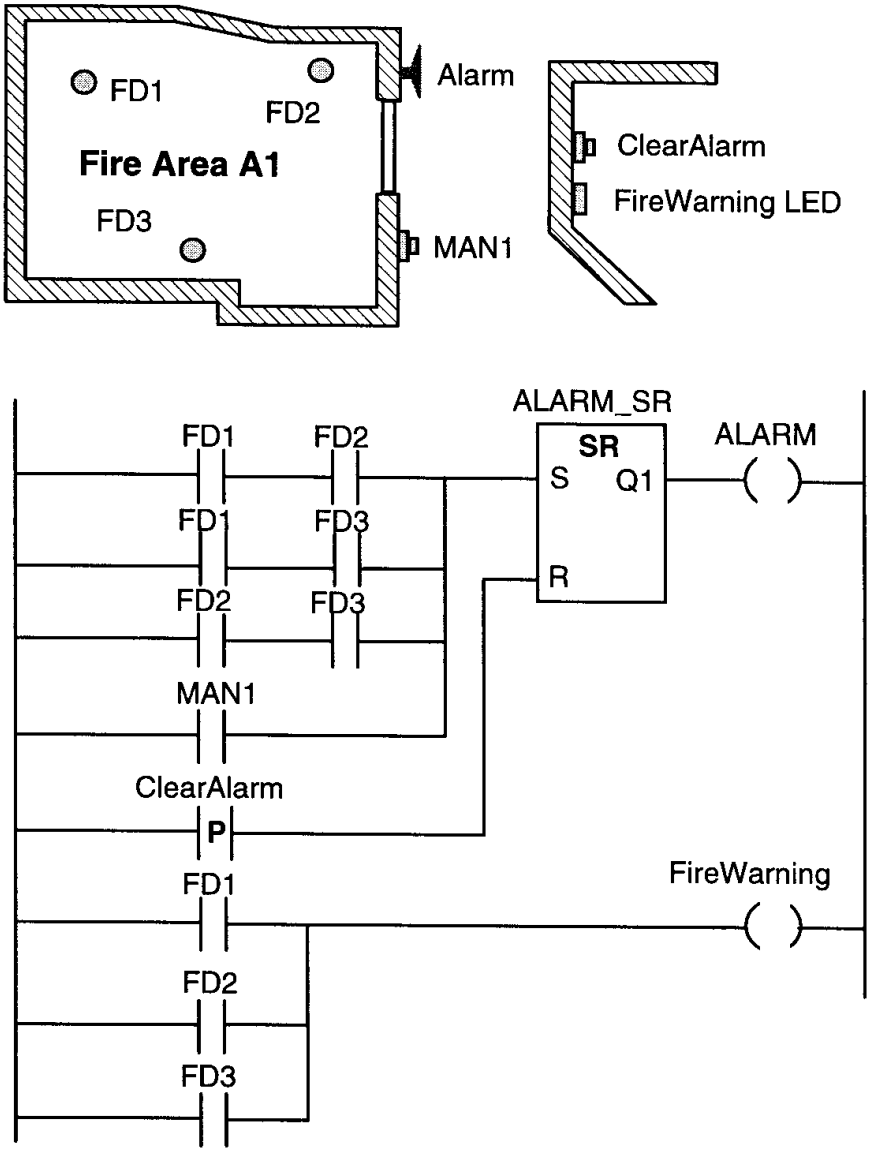
Tämä koodi lataa plc_tila muuttujan muistiin ja tallentaa sen arvon liikuta_hihnaa muuttujaan. Käytössä on myös muuttujien vertailu-, laskenta- ja siirtymäkomentoja.

5.2 LD – Ladder Diagram

Tikapuukaavio on relelogiikkaan perustuva kieli. Tämä sopii erityisesti sähköasentajien tai vastaavan osaamisen hallitseville ihmisille.

Logiikka rakentuu kahden kiskon väliin, joista vasen on virtakisko, joka syöttää ykkös bittiä ja oikea on nollakisko. Releitä(contact) on avautuvia ja sulkeutuvia. Releelle aina nimetään muuttuja. Jos muuttuja on ykkönen niin rele toimii, eli sulkeutuva rele sulkeutuu ja päästää virran läpi ja avautuva rele avautuu ja estää virran kulun. Virtaketjussa voi olla myös funktioita, jotka esitetään toimilohkoina. Ketjun lopuksi on aina käämi (coil), joka edustaa ohjattavaa muuttujaa. Muuttuja saa käämin vasemmalla puolella olevan arvon. [Lewis 1998] [Bonfatti 1999]

Ohjelma suoritetaan ylhäältä alas yksi askelma(rung) kerrallaan. Jokainen askelma käydään läpi vasemmalta oikealle. Askelmalla tarkoitetaan releitä, toimilohkoja ja käämejä jotka ovat yhteydessä toisiinsa. [Lewis 1998] [Bonfatti 1999]

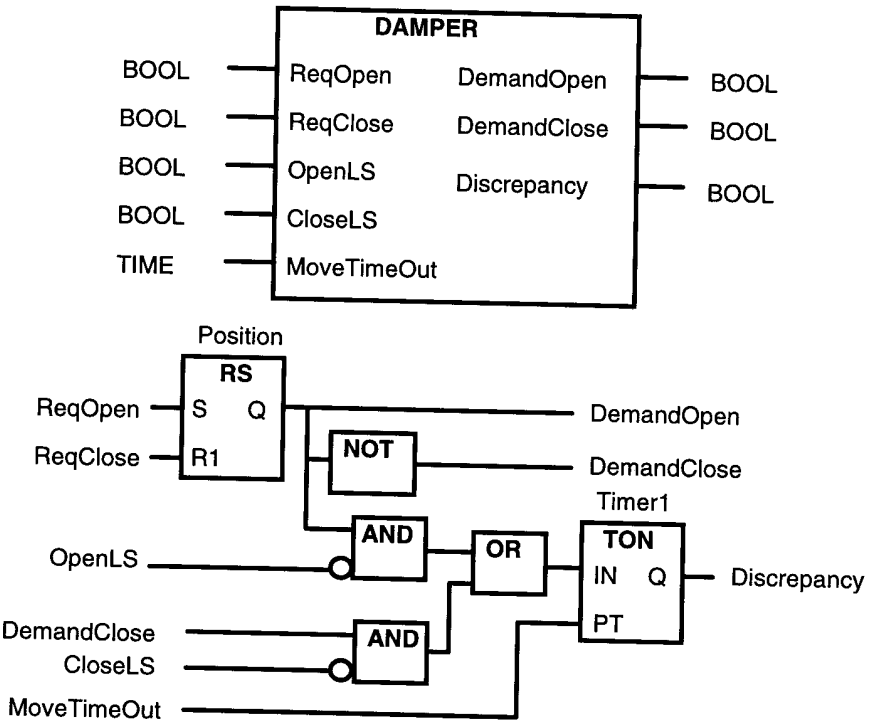


Kuva 3 Tikapuukaavio [Lewis 1998]

5.3 FBD- Function Block Diagram

Toimilohkokaavio koostuu toimilohkoista ja jokaisen ketjun lopussa on aina vähintään yksi muuttuja jota ohjataan. Myös tässä on käytössä virtapiiriajattelu mutta releiden sijasta käytetään pelkästään toimilohkoja. Tässä on käytössä sama ajojärjestys kuin tikapuukaaviossa.

Toimilohkot muistuttavat digitaalitekniikassa käytettävää merkintatapaa ja standardissa on määritelty mitä toimilohkoja on käytössä. Pääosin nämä ovat kiikkuja, liipaisimia, vertailua, matemaattisia operaatioita, muisteja ja viiveitä. Eli toimintalohkon vasemmalta puolelta tulee tarvittavat inputit, joita käsitellään toimilohkon funktion mukaan ja lopputulokset tulevat ulos oikealta puolelta. Koko askeleen sisääntulot ovat muuttujia. [Lewis 1998] [Bonfatti 1999]



Kuva 4. Toimilohkokaavio [Lewis 1998]

5.4 ST – Structured Text

Rakenteellinen teksti muistuttaa normaalien ohjelmointikielten syntakseja mutta on kuitenkin erillinen kieli. Kieli muistuttaa lähinnä Pascalia. Käytössä ovat silmukat, ehtolauseet ja muuttujien sijoitukset.

Rekursiiviset kutsut on kuitenkin estetty stabiilisuuden takia. Lisäksi käytössä ovat samat funktiot kuin muissakin standardin kielissä. Erona muihin kieliin on paljon suurempi vapaus käyttää muuttujia. Tämä kieli luonnollisesti sopii Pascalia, C:tä tai vastaavaa kieltä osaavalle. [Strömman 2002]

```
Fault := FALSE;
FOR I:= 1 TO  20 DO
  FOR J := 0 TO 9 DO
    IF FaultList[I,J] THEN
      FaultNo := I*10 + J;
      Fault := TRUE; EXIT;
    END_IF;
  END_FOR;
  IF Fault THEN EXIT; END_IF;
END_FOR;
```

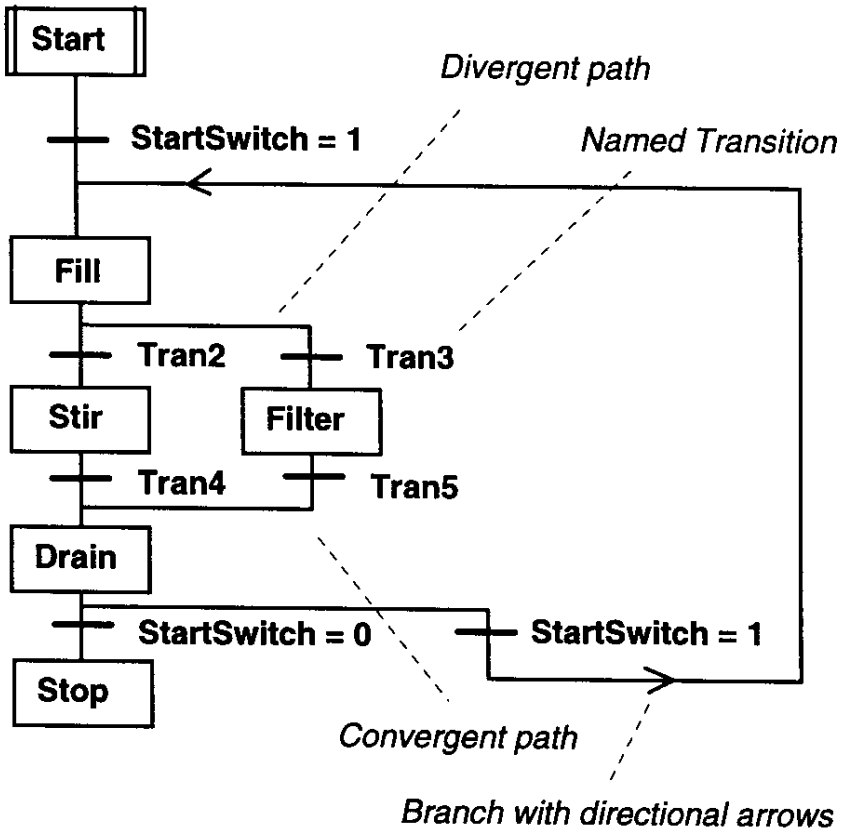
Kuva 5. Rakenteellinen teksti [Lewis 1998]

5.5 SFC – Sequential Function Chart

Sekvenssikaavio ei ole varsinainen itsenäinen kieli vaan sillä kuvataan ohjelman sekvenssejä. Sekvenssit sitten ohjelmoidaan jollain muulla IEC-kielellä.

Kaavio rakentuu askeleista(step), niiden välisistä siirtymistä ja siirtymäehdoista(transition, transition condition) ja haarautumisista. Askeleeseen liitetään toimintoja(action). Sekvenssikaaviossa on suoraan määritetty joitain yksinkertaisia toimintoja kuten set ja reset mutta monimutkaisemmat pitää toteuttaa itse. Pelkällä sekvenssikaaviolla voi siis tehdä vain hyvin yksinkertaisia ohjelmia. [Lewis 1998] [Bonfatti 1999]

Sekvenssikaavio on kehitetty, jotta ohjelman tila siirtymien syitä olisi helpompi seurata. Tämä tietysti vaatii, että ohjelma on luonnostaan sekvenssinen, koska keinotekoisia sekvenssejä on vaikea keksiä ja ainakin menetetään mahdollisuus tehdä todellisuutta mallintavia ohjelmia. [Lewis 1998] [Bonfatti 1999]



Kuva 6. Sekvenssikaavio [Lewis 1998]

6. Johtopäätöksiä

Kaikki muut IEC 61131-3 standardin kielet, paitsi sekvenssikaavio, ovat imperatiivisia, koska niissä jo lähtökohtaisesti pelataan muuttujilla. Mikään niistä ei myöskään sovi olio - tai rinnakkaiseen paradigmaan. Seuraava kysymys onkin, miksi asia on näin?

Syy on itse tarkoituksessa mihin nämä kielet on luotu. Eli niillä pitää pystyä ohjelmoimaan automaatiojärjestelmässä käytössä olevaa logiikkaa. Tällöin on hyvin luontevaa määrittää antureiden ja ohjausten tiedot muuttujiksi, joita sitten voidaan käsitellä. Tämä on todettu toimivaksi, eli näillä kielillä voidaan tehdä täysin toimivia ohjelmistoja.

Onko sitten mitään varsinaista estettä, miksei voitaisi tehdä deklaratiivista PLC-kieltä? Henkilökohtaisesti uskon, ettei se onnistu koska PLC on niin sidottu laitteisiin ja antureihin ja niiden antamaa tietoa on pystyttävä käsittelemään yksityiskohtaisesti. Jonkinlaisen korkeamman tason kielen, jossa käytetään jo valmiiksi tehtyjä komponentteja voisi kuvitella mutta ainakaan lähitulevaisuudessa emme tule näkemään tälläistä, jos koskaan.

Olioparadigmaan sopivia PLC-kieliä sen sijan kyllä uskoisin kehiteltävän, koska myös PLC-ohjelmoinnissa isot ohjelmat ovat hankalia lukea ja ylläpitää. Olio-ohjelmointi PLC-maailmassa ei kuitenkaan suoraan istu nykyisin käytettävään paradigmaan, koska esimerkiksi valmiita jatkuvasti tarvittavia luokkia on paljon vaikeampi löytää. Enemmänkin luokkien pitäisi perustua laitteiston fyysiseen jakoon, siten että tiettyä yleistä tarvetta varten olisi yksi luokka. Tämän metodit saisivat parametreikseen asiaan liittyvät muuttujat ja näitten avulla pystyisi muodostamaan ohjelman joka ohjaisi laitteistoa.

Tätä ei ole tutkittu mutta voisin kuvitella tälläisiä löytyvän esimerkiksi kuljettimien risteyksistä. Siinä parametreiksi saataisiin anturit ja eri risteystyypit voisivat olla periytettyjä toisistaan. Tämä tosin vaatisi jonkinlaista standardia antureitten paikoista, jotta kone tietäisi mitä esimerkiksi lähestymisanturi tarkoittaa. Ongelmaton tälläinen lähestymistapa ei todellakaan olisi mutta tarve on kuitenkin selvä.

Jos yleiset luokat hylätään niin paradigmasta tulee vielä houkuttelevampi. Varsinkin isompi yritys voisi laatia itselleen standardin jota se itse noudattaa ja rakentaa olio kielellä kirjastoja omille tuotteilleen. Tässä ongelmat tulevat vastaan jos järjestelmässä on monien valmistajien tuotteita mutta se on nykyisinkin ongelma.

Tulevaisuudessa uusi standardi IEC 61499 tuo uutena ideana pelkän logiikan lisäksi mukaan tapahtumat, eli jos 61131-3 standardi mahdollistaa syklisen toteutuksen niin 61499 mahdollistaa lohkojen dynaamisen hallinnan tapahtumien kautta. 61499 standardi luo mahdollisuuden tiedon piiloittamiseen ja sinällään muistuttaa olio paradigmaa. [Christensen] myös kirjoittaa, että IEC 61499 standardissa toimilohkot ovat toimilohkotyyppien instansseja(instances). Standardi ei ole vielä valmis vaan se on kehittelyasteella.

Sekvenssikaavio ei ole ohjelmointikieli ja siten mikään ohjelmoinparadigma ei sovellu siihen. Lähinnä se muistuttaa imperatiivista paradigmaa.

7. Viitteet

[Appleby 1997] Appleby D.,VandeKopple J.: Programming languages – paradigm and practice, WCB/McGraw-Hill, 1997

[Wegner 1990] Wegner P.: Concepts and paradigms of object-oriented programming OOPS Messenger, 1(1):7-87, 1990. Expansion of OOPSLA'89 Keynote Talk.

[Wegner 1988] Wegner P.: Object-oriented concept hierarchies, Distributed at CASE '88 workshop, July 1988.

[Watt 1990] Watt D.: Programming Language Concepts and Paradigms, Prentice Hall 1990

[Strömman 2002] Strömman M.: Ohjelmoitavan logiikan ohjelmointi ohjelmistotuotantoprosessina, Teknillinen Korkeakoulu 2002

[Lewis 1998] Lewis R. W.: Programming industrial controlling systems using IEC 1131-1 Revised edition, The Institution of Electrical Engineers, London 1998

[Bonfatti 1999] Bonfatti F., Monari P. D., Sampieri U.: IEC 113-1 Programming Methodology – Software engineering methods for industrial automated systems, CJ International 1999

[Heinonen 1996] Heinonen M.: Funktionaalinen ohjelmointi, Jyväskylän yliopisto 1996

[Glaser 1984] Glaser H., Hankin C., Till D.: Principles of Function Programming, Prentice-Hall 1984

[Pascal] Pascal opas luku 5. Ohjelmointiparadigmat.
http://www.tuug.utu.fi/~f/pascal/luku05.html viitattu 10.11.2003

[Ohjelmistotekniikka] Maarit Harsu, TTY Ohjelmistotekniikka 2003, Ohjelmointiparadigmat.
http://www.cs.tut.fi/~okp/luennot/kalvot/johdanto.pdf viitattu 10.11.2003

[javaolio] Sun yhtiön javaopas oliot.
Hakusanoja: java tutorial definition "what is an object"
http://java.sun.com/docs/books/tutorial/java/concepts/object.html Linkkiin viitattu 7.11.2003.

[javaluokka] Sun yhtiön javaopas luokat.
Hakusanoja: java tutorial definition "what is a class"
http://java.sun.com/docs/books/tutorial/java/concepts/class.html Linkkiin viitattu 7.11.2003

[Christensen] Christensen J.: Basic Concepts of IEC 61499
http://www.holobloc.com/papers/1499_conc.zip Linkkiin viitattu 10.11.2003.