hyväksymispäivä arvosana

arvostelija

Imperatiivisten ohjelmien organisointiparadigmojen historia

Timo Tapanainen

Helsinki 25.2.2007

Seminaarityö: Tietojenkäsittelytieteen historia

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Paradigma-sana tuli yleiseen käyttöön Thomas Kuhnin tieteenfilosofisen kirjoituksen myötä [Kuh62]. Paradigma tarkoittaa jonkin tieteenalan tiettynä ajanjaksona vallalla olevia perusperiaatteita ja ajatusmalleja, joiden varaan paradigmaa kannattava tiedeyhteisö perustaa toimintansa ja joita ei yleensä kyseenalaisteta. Kuhnin mukaan tiede ei edisty yksittäisten uusien teorioiden kautta vaan vallitsevan paradigman vaihdoksina. Paradigman vaihdos alkaa tutkimustuloksista, jotka ovat ristiriidassa vallitsevan paradigman perusperiaatteiden kanssa. Useat ristiriidat ajavat vallitsevan paradigman kriisiin, jolloin sen korvaa uusi paradigma.

Tietojenkäsittelytieteen tunnetuimmat paradigmat ovat ohjelmointiparadigmoja. Ohjelmointiparadigma on laskennan malli, joka ohjaa ohjelman ongelmanratkaisutapaa. Laskennan malleista tunnetuimmat ovat imperatiivinen ohjelmointi, logiikkaohjelmointi ja funktionaalinen ohjelmointi. Imperatiivisessa ohjelmointiparadigmassa ohjelma koostuu tilasta ja sitä muuttavista lauseista (käskyistä), jotka kuvaavat askel askeelta kuinka ratkaisu muodostetaan. Logiikkaohjelmoinnissa ja funktionaalisessa ohjelmoinnissa ei kuvata ratkaisuun johtavia toimenpiteitä vaan niissä kerrotaan ratkaisuun liittyvät faktat ja yhtälöt, joiden perusteella ohjelmaa suorittava systeemi tekee tarvittavat toimenpiteet.

Imperatiivinen ohjelmointiparadigma on paradigmoista eniten ja laajimmin käytetty. Tämä on pitkälti imperatiivisten ohjelmien tehokkuuden asiota, sillä ne ovat tehokkaampia kuin muihin paradigmoihin perustuvat ohjelmat. Muiden paradigmojen tehottomuus johtuu niiden taustalla käytetystä tietokonearkkitehtuurista. Nykyaikaiset tietokoneet perustuvat von Neumanin tietokonarkkitehtuuriin, joka on luonteeltaan myös imperatiivinen. Nykyinen vallitseva tietokonearkkitehtuuri ei siis tue suoraan funktionaalisia ohjelmia tai logiikkaohjelmia, mikä vaikuttaa heikentävästi näillä laadittujen ohjelmien suorituskykyyn.

Imperatiivisen ohjelmoinnin rinnalle tarvittiin muita ohjelmointiparadigmoja, sillä pelkkä laskennan malli ei riittänyt aina vain monimutkaisempien ohjelmien toteuttamiseen. Keskeinen ongelma oli ohjelman toiminnallisuuden ja tiedon organisointi. Tätä tarvetta täyttämään syntyi erilaisia ohjelman rakennetta ohjaavia paradigmoja, jotka kertoivat miten ja minkälaisiin ohjelmayksiköihin ohjelman toiminnallisuus ja tieto on jaettava. Tällaisia ohjelmointiparadigmoja olivat esim. rakenteinen ohjelmointi, proseduraalinen ohjelmointi ja olio-ohjelmointi. Näiden paradigmojen välillä on nähtävissä selkeitä paradigman

vaihdoksia ja yleensä uusi paradigma on sisällyttänyt osia vanhan paradigman perusperiaatteista itseensä.

Tässä kirjallisessa esitelmässä perehdytään imperatiivisten ohjelmien rakennetta ohjaavien ohjelmointiparadigmojen historiaan. Historiakatsauksessa ovat mukana seuraavat paradigmat: rakenteeton ohjelmointi, proseduraalinen ohjelmointi, rakenteinen ohjelmointi, modulaarinen ohjelmointi ja olio-ohjelmointi. Paradigmat esitetään kronologisessa järjestyksessä vanhimmasta uusimpaan.

2 Imperatiivinen ohjelmointi ilman rakennetta

Imperatiivinen ohjelmointi syntyi 40-luvulla ensimmäisten konekielten myötä. Konekieliset ohjelmat kirjoittiin binäärimuodossa ja ne koostuivat suorittimen käskykannan käskyistä. Konekieliä seurasivat symbooliset konekielet, joissa binääristen konekäskyjen tilalla käytettiin symboolisia konekäskyjä. 50-luvulla syntyivät ensimmäiset korkean tason ohjelmointikielet: FORTRAN, ALGOL ja COBOL, jotka tarjosivat ensimmäisiä abstraktioita alla olevaan koneeseen.

Imperatiivisen paradigman läheinen suhde alla olevaan tietokoneeseen vaikutti huomattavasti imperatiivisten ohjelmien rakenteeseen. Tietokoneen muistissa olevalla ohjelmalla ei ole ”rakennetta”, sillä se koostuu vain joukosta peräkkäisiä käskyjä. Tämä ajattelutapa näkyi myös useimmissa imperatiivisissa ohjelmissa, jotka koostuivat yhdestä yhtenäisestä ohjelmalohkosta. Näissä ohjelmissa ei ollut selkeää rakennetta, minkä vuoksi tällaista ohjelmointityyliä kutsutaan rakenteettomaksi ohjelmoinniksi.

Ensimmäisten imperatiivisten ohjelmien kontrollivuon ohjaukseen käytettiin pääasiassa hyppylauseita. Kone- ja assemblykielissä kontrollinohjaus perustuu ehdottomiin ja ehdollisiin hyppykäskyihin, joilla suorittimelle osoitetaan seuraavaksi suoritettava käsky. Korkean tason imperatiiviset ohjelmointikielet tarjosivat tämän kontrollinohjausmallin ohjelmoijalle GOTO-lauseiden muodossa, joilla kontrolli voitiin siirtää GOTO-lauseessa ilmoitettuun kohtaan.

FORTRAN:in ensimmäisissä versioissa kontrollinohjaus perustui yksinomaan GOTO-lauseisiin. COBOL ja PL/I ohjelmointikielet olivat niin ikään riipuvaisia GOTO-lauseista. ALGOL 60 ja sen seuraajat tarjosivat monia GOTO-riippumattomia kontrollirakenteita. ALGOL 60 sisälsi mm. if...then...else, for ja while kontrollirakenteet, mutta GOTO-lausetta voitiin kuitenkin käyttää, sillä sen käyttöä ei estetty [Sco00, s. 267].

3 Ohjelman organisointi proseduureilla

Rakenteetonta ohjelmointia seurasi proseduraalinen ohjelmointi. Proseduraalisen ohjelmoinnin syntyyn ei ole selvää ajankohtaa, mutta sen voidaan katsoa syntyneen 60-luvun alussa ensimmäisten proseduuria tukeneiden ohjelmointikielen myötä. Proseduurit ovat olleet useiden ohjelmointikielten tärkein ohjelman organisointiin käytetty rakenneyksikkö [Bud03, s. 234] ja ne ovat vahvasti esillä proseduraalista ohjelmointi seuranneissa paradigmoissa.

Proseduraalisessa ohjelmoinnissa ohjelman perusyksikkö on proseduuri, joita nimitetään myös funktioiksi, metodeiksi tai aliohjelmiksi. Proseduraalinen ohjelma koostuu proseduurien puumaisesta rakenteesta, jossa ylemmän tason proseduurit kutsuvat yhtä tai useampaa alemman tason proseduuria.

Proseduraaliseen ohjelmointiin kuului olennaisena osana asteittaisen tarkentamisen (stepwise refinement) suunnittelustrategia, joka tunnetaan myös ylhäältä alas (top-down) suunnittelustrategiana. Siinä ohjelman toiminnallisuus jaetaan asteittain proseduureille kunnes jaettava toiminnallisuus on niin yksinkertertainen, että se voidaan toteuttaa suoraan. Asteittaista tarkentamista huomoidaan sekä toiminnallisuuden ja tiedon organisointi, mutta pääpaino on selvästi toiminnallisuuden organisoimisessa.

Proseduraalisissa ohjelmoinnissa käytettiin myös GOTO-lauseita. Niitä käytettiin proseduurin ennenaikaiseen päättämiseen return-lauseiden tapaan. Tämä oli yleistä esim. Pascal- ja ALGOL 60-ohjelmointikielissä, jotka eivät sisältäneet return-lausetta. Ohjelman poikkeuskäsittely saatettiin toteuttaa myös GOTO-lauseilla. Proseduuri saattoi hypätä poikkeustilanteessa proseduurin ulkopuoliseen poikkeuskäsittelijään. Proseduurien ei-paikalliset hypyt vaikeuttivat ohjelmointikielen toteutusta ja ne vaikeuttivat huomattavasti ohjelman ymmärrettävyyttä [Sco00, s. 269].

4 Ohjelmistokriisi ja rakenteinen ohjelmointi

Tietokoneet yleistyivät ja kehittyivät merkittävästi 40- ja 70-luvun välillä. Keskustietokoneiden käyttö yleistyi yrityksissä ja minitietokoneiden myötä myös pienillä yrityksillä oli varaa tietokoneisiin [Rac97]. Tietokoneiden muistin ja laskentatehon kasvaessa niillä voitiin ratkaista aina vain monimutkaisempia ja laajempia ongelmia, mikä teki myös ohjelmista monimutkaisia ja laajoja. Ohjelmistotekniikka – joka ”syntyi” vasta 1968 NATO:n konferenssissa [NaR68] – ei ollut kehittynyt tarpeeksi vastaamaan tuon ajan haasteisiin. Ohjelmat olivat arvioitua kalliimpia, virheellisiä, vaikeasti ylläpidettäviä ja niiden toimitukset myöhästelivät. Tietojenkäsittely oli ohjelmistokriisissä.

Rakenteinen ohjelmointi syntyi 70-luvun alussa ja se oli ensimmäinen yritys ohjelmistokriisin ratkaisemiseen. Rakenteisen ohjelmoinnin tavoitteena oli tuottaa ohjelmia, joiden suoritusvuota oli helppo seurata ohjelmatekstistä. Ohjelmien haluttiin kommunikoivan hyvin myös ihmisille.

Rakenteisessa ohjelmoinnissa luovuttiin GOTO-lauseiden käytöstä, sillä niiden hallitsematon käyttö johti helposti ns. ”spagettikoodiin” – koodiin, jonka kontrollirakenne oli yhtä monimutkainen ja sotkeutunut kuin kasa spagettia. Sekava kontrollirakenteen vuoksi ohjelman suoritusvuota oli vaikea päätellä ohjelmatekstistä, mikä vaikeutti merkittävästi ohjelman ymmärrettävyyttä ja sitä kautta ylläpidettävyyttä.

GOTO-lauseiden tilalla tuli käyttää vain kolmea eri kontrollirakennetta: toistoa, valintaa ja peräkkäisyyttä. Kontrollirakenteilta ja proseduureilta edellytettiin myös että niillä oli vain yksi sisäänmeno- ja ulostulokohta, jolloin kontrollivuota oli helpompi seurata. Yllä mainittuja kontrollirakenteita käyttämällä ohjelman rakenteesta tuli puumainen [Weg73] ja jonka kontrollivuota voitiin seurata lukemalla ohjelmatekstiä ylhäältä alas [Jon76].

Edsger W. Dijkstralla oli merkittävä rooli rakenteisen ohjelmoinnin kehityksessä. Rakenteisen ohjelmoinnin pohja luotiin vuonna 1968 artikkelissa, jossa Dijkstra totesi GOTO-lauseiden olevan haitallisia [Dij68]. Dijkstran mukaan GOTO-lauseet vaikeuttivat ohjelman kontrollivuon seuraamista. Artikkelissa todettiin GOTO-lauseet myös tarpeettomiksi, sillä Böhm ja Jacopinin olivat todistaneet, että minkä tahansa ohjelman kontrollivuo voidaan toteuttaa käyttämällä vain kolmea kontrollirakennetta: toistoa, valintaa ja peräkkäisyyttä. Termi ”rakenteinen ohjelmointi” esiintyi ensimmäisen kerran vuonna 1969 Dijkstran esseekokoelmassa, joka kiersi nimellä: ”Notes on Structured Programming” [Dij72]. Nämä muistiinpanot sisälsivät Dijkstran aikaisempia rakenteiseen ohjelmointiin liittyviä kirjoituksia [Wei78][Weg76].

Rakenteista ohjelmointi seuranneet paradigmat sisällyttivät osia rakenteisesta ohjelmoinnin perusperiaatteista itseensä. Niissä rakenteista ohjelmointia sovelletaan yleensä toteutuksen matalalla tasolla ohjelmayksiköiden sisällä. Tämä on nähtävissä esim. siitä että nykyisten ohjelmien kontrollivuota voidaan seurata lukemalla ohjelmatekstiä ylhäältä alas.

5 Modulaarinen ohjelmointi

Modulaarinen ohjelmointi oli muodissa 60-luvun loppupuolella vaikkakin se oli mahdollista FORTRAN:illa jo useita vuosia aikaisemmin [Ran79]. Ensimmäiset modulaarista ohjelmointia aidosti tukevat kielet olivat Clu, Modula ja Ada. C-

ohjelmointikieli ei suoraan tukenut moduuleja, mutta niitä voitiin emuloida erillisen kääntämisen mekanismeilla [Sco00, s. 123].

Modulaarinen ohjelmointi perustuu moduuleihin: itsenäisesti toteutettaviin ja käännettäviin ohjelmayksiköihin, jotka kapseloivat sisältämänsä tiedon ja proseduurit. Moduulit tekevät eron myös julkisen rajapinnan ja sen toteutuksen välillä [Wik07]. Julkinen rajapinta voidaan antaa erillisessä ohjelmayksikössä tai yhdessä moduulin toteutuksen kanssa. Moduulia voidaan käyttää vain sen julkisen rajapinnan kautta sillä moduuli voi peittää toteutuksen rajoittamalla moduulin proseduurien tai tiedon näkyvyyttä. Näin moduulia käyttävät asiakkaat eivät voi käsitellä moduulin peitettyä tietoa vaan se on tehtävä moduulin julkisen rajapinnan kautta.

Modulaaristen ohjelmointikielten tuki moduulityypeille vaihteli kielestä toiseen. Simula, Euclid ja ML tarjosivat moduulit tyyppeinä. Moduulityypit käyttäytyivät normaalien tyyppien tapaan. Niistä voitiin luoda useita instansseja ja ne olivat tyyppiturvallisia. Modula-2 ja Ada 83 eivät taas tukeneet moduulityyppejä vaan moduuli edusti vain yhtä ilmentymään [Sco00, s. 126]. Esim. pinoa mallintavasta moduulista voitiin luoda vain yksi pino. Ongelmaa voitiin kiertää moduulimanagereilla, jotka hoitivat moduulin instanssien luomisen. Ohjelmoijan täytyi kuitenkin aina välittää käsiteltävä instanssi moduulin metodille, joten moduulin instanssia ei voitu käyttää aidon tyypin tapaan.

Moduulien hyödyt on tunnettu jo modulaarisen ohjelmoinnin alkuajoilta asti. Parnaksen mukaan modulaarinen ohjelmointi lyhentää ohjelman kehitysaikaa, lisää tuotteen joustavuutta ja helpottaa ohjelman ymmärtämistä [Par72]. Ohjelman kehitysaika lyhenee, koska moduulien toteutus voidaan aloittaa samanaikaisesti riippuvuuksista huolimatta. Riippuvuuksien osalta riittää että tarvittavien moduulien julkiset rajapinnat ovat saatavilla. Julkiset rajapinnat vähentävät myös eri moduuleita kehittävien tiimien kommunikointitarvetta. Tuotteen joustavuus seuraa tiedon peittämisestä, sillä muutokset moduulin peitettyyn tietoon eivät vaikuta muihin moduuleihin, mikäli moduulin julkinen rajapinta ei muutu. Parantunut ymmärrettävyys on myös julkisten rajapintojen ansiota, sillä julkista rajapintaa käyttävän asiakkaan ei tarvitse tietää mitään moduulin sisäisestä toteutuksen.

6 Olio-ohjelmointi

Olio-ohjelmoinnista tuli ohjelmoinnin pääsuuntaus 90-luvulla. Olio-ohjelmointi syntyi kuitenkin jo 60-luvulla ensimmäisen olio-ohjelmointikielen Simulan myötä. Simula sisälsi luokka-käsitteen ja sillä voitiin ilmaista luokkien välinen

perintä [Cap03]. Simulaa seurasi ensimmäinen ”puhdas” oliokieli Smalltalk, jossa myös primitiivityypit olivat olioita. Olio-kielten suosio alkoi kuitenkin vasta 80-luvun puolivälissä C++-ohjelmointikielestä, joka laajensi suosittua C-kieltä oliopiirteillä. Muutkin ohjelmointikielet seurasivat C++-kielen esimerkkiä omaksumalla oliopiirteet. Olio-ohjelmoinnin suosiota kasvatti entisestään vuonna 1995 esitelty Java-ohjelmointikieli, joka soveltui hyvin Internet-aikakaudelle turvallisuutensa ja siirrettävyytensä vuoksi.

Olio-ohjelmoinnille ominaista ovat luokat, perintä ja dynaaminen sidonta. Luokka vastaa ominaisuuksiltaan moduulityyppiä, mutta luokat voivat periä myös toisten luokkien ominaisuuksia. Perinnän avulla voidaan luoda uusia luokkia (aliluokka) uudelleenkäyttämällä olemassa olevia luokkia (yliluokka). Dynaaminen sidonnan ja perinnän ansiosta aliluokkia voidaan käyttää siellä missä yliluokkiakin ja tämän lisäksi aliluokat voivat tuoda näihin yhteyksiin uutta toiminnallisuutta.

7 Yhteenveto

Tietokoneohjelmoinnin historia on sisältänyt useita paradigman vaihdoksia. Rakenteinen ohjelmointi korvasi rakenteettoman ohjelmoinnin 70-luvun alussa. Rakenteisessa ohjelmoinnissa luovuttiin GOTO-lauseista ja ne korvattiin kontrollirakenteilla, jotka helpottivat ohjelman suoritusvuon seuraamista. 60-luvun lopulla nousi esiin modulaarinen ohjelmointi, jolla ohjelma voitiin osittaa itsenäisesti toteutettaviin osiin, moduuleihin. Modulaarinen ohjelmointi helpotti laajojen ja monimutkaisten ohjelmien osittamista ja se mahdollisti näiden samanaikaisen kehittämisen. Moduulaarista ohjelmointi seurasi olio-ohjelmointi, joka nousi ohjelmoinnin pääsuuntaukseksi 90-luvun alussa.

8 Lähteet

[Bud03] Budgen, D. 2003 *Software Design*. 2. edition. Addison-Wesley Longman Publishing Co., Inc.

[Cap03] Capretz, L. F. 2003. A brief history of the object-oriented approach. *SIGSOFT Softw. Eng. Notes* 28, 2 (Mar. 2003), 6.

[Dij68] Dijkstra, E. W. 1968. Letters to the editor: go to statement considered harmful. *Commun. ACM* 11, 3 (Mar. 1968), 147-148.

[Dij72] Dijkstra, E. W. 1972, Notes on structured programming, in "Structured Programming", Academic Press, 1972, pp. 1--82.

[Jon76] Jones, D. E. 1976. Structured programming and FORTRAN. In *Proceedings of the 14th Annual Southeast Regional Conference* (Birmingham, Alabama, April 22 - 24, 1976). ACM-SE 14. ACM Press, New York, NY, 1-6.

[Kuh62] Kuhn, Thomas S. *The Structure of Scientific Revolutions*. University of Chicago Press, 1962

[NaR68] Naur P., Randell B. et. al Software Engineering: A Report on A Conference Sponsored by the NATO Science Committee, NATO, 1968

[Par72] Parnas, D. L. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (Dec. 1972), 1053-1058.

[Rac97] Raccoon, L. B. 1997. Fifty years of progress in software engineering. *SIGSOFT Softw. Eng. Notes* 22, 1 (Jan. 1997), 88-104

[Ran79] Randell, B. 1979. Software engineering in 1968. In *Proceedings of the 4th international Conference on Software Engineering* (Munich, Germany, September 17 - 19, 1979). International Conference on Software Engineering. IEEE Press, Piscataway, NJ, 1-10.

[Sco00] Scott, M. L. 2000 *Programming Language Pragmatics*. Morgan Kaufmann Publishers Inc.

[Weg73] Wegner, E. 1973. Tree-structured programs. *Commun. ACM* 16, 11 (Nov. 1973), 704-705.

[Weg76] Wegner, P. Programming Languages—The First 25 Years., *IEEE Transactions on Computers*, Vol.C-25, Iss.12, Dec 1976 Pages: 1207- 1225

[Wei78] Weiner, L. H. 1978. The roots of structured programming. In *Papers of the SIGCSE/CSA Technical Symposium on Computer Science Education* (Detroit, Michigan, February 23 - 24, 1978). ACM Press, New York, NY, 243-254

[Wik07] Module (programming). (2007, February 2). In *Wikipedia, The Free Encyclopedia*. Retrieved 11:40, February 20, 2007, from http://en.wikipedia.org/w/index.php?title=Module_%28programming%29&oldid=105143019