Ohjelmointiparadigmojen historia

Mikko Teräs

Helsinki, 14. toukokuuta 2001
Tietojenkäsittelytieteen historia -seminaari

Helsingin yliopisto
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Korkean tason ohjelmointikieliä alettiin tutkia pian ensimmäisten tietokoneiden ilmestymisen jälkeen. Ensimmäiset kielet olivat kömpelöitä ja ad hoc -menetelmin suunniteltuja. Seuraavien vuosikymmenien aikana tietämys formaaleista menetelmistä ja järjestelmällisemmästä suunnittelusta poiki käyttökelpoisempia kieliä. Tämä korkean tason ohjelmointikielten evoluutio on paitsi muokannut kielistä ilmaisuvoimaisempia myös eriyttänyt kielet keskenään erilaisiin *paradigmoihin*.

Tässä esitelmässä tutustutaan ohjelmointikielten kehitykseen paradigmojen näkökulmasta. Tarkastelemme eri paradigmojen erkanemista toisistaan ja kehitystä niiden sisällä. Tarkastelu painottuu kolmeen valtavirran paradigmaan: imperatiiviseen, funktionaaliseen ja oliokeskeiseen. Muitakin paradigmoja on, mutta niiden asema on edellämainittuihin verrattuna varsin marginaalinen.

Tarkastelu painottuu paradigmojen alkuaikoihin: toisessa luvussa käsitellään 1950-lukua, kolmannessa 60-lukua ja neljännessä 70-lukua. 1970-luvun jälkeisiä tapahtumia tarkastellaan lyhyesti viidennessä luvussa.

1.1 Ohjelmointikielten jakamisesta paradigmoihin

Ohjelmointikielet voidaan jakaa perheisiin sen mukaan, minkälaisista perusosista ohjelmat niissä muodostetaan [Sethi96]. Absoluuttista jakoa ei ole mahdollista tehdä, koska monet kielistä sisältävät piirteitä useista eri paradigmoista. Lisäksi yhden paradigman sisällä voi olla useita eri tyylisuuntia. Tässä esitelmässä keskitytään kolmeen merkittävimpään paradigmaan.

Valtaosa suosituista kielistä kuuluu *imperatiiviseen* paradigmaan. Imperatiiviset kielet ovat toimintapainotteisia: ohjelma kuvataan sarjana komentoja, jotka tietokone suorittaa järjestyksessä. Muuttujat ja arvojen sijoittaminen niihin ovat olennainen osa imperatiivista ohjelmointia. Imperatiivinen paradigma jakautuu useisiin tyyleihin, kuten lohkorakenteiseen ohjelmointiin, modulaariseen ohjelmointiin ja olio-ohjelmointiin. Viimeksimainittua tarkastellaan tässä omana paradigmanaan.

Lähes päinvastaista ideologiaa edustaa *funktionaalinen* paradigma, jossa ohjelmat rakennetaan matemaattisista funktioista ja niiden (tyypillisesti rekursiivisista) sovelluksista. Puhtaassa funktionaalisessa ohjelmassa ei ole muuttujia eikä sijoituslauseita. Tunnetuin funktionaalinen ohjelmointikieli on LISP[1].

[1] LISPissä on kyllä sijoituslause, mutta LISPin mielenkiintoisin osa on sen puhdas funktionaalinen osajoukko. Sitä on mahdollista ja tarkoituksenmuksistakin käyttää oikeaan ohjelmointiin.

Olioparadigmassa ohjelmat rakennetaan keskenään keskustelevista, tietopainotteisista olioista. Vaikka olio-ohjelmointikielten voitaisiin yhtä hyvin katsoa kuuluvan muihin paradigmoihin (kielestä riippuen yleensä joko imperatiiviseen tai funktionaaliseen), oliopohjaisuus on sinällään niin suuri ja erilainen käsite, että siihen painottuvia kieliä kannattaa tarkastella omana perheenään. Oliokielet ovat toisaalta keskenään hyvinkin erilaisia: oliopohjaisuuden lisäksi esimerkiksi CLOSilla, Smalltalkilla ja C++:lla ei ole juuri yhteisiä piirteitä.

2 Korkean tason kielten ensiaskeleet

Varhaisista (Fortrania edeltävistä) ohjelmointikielistä yksikään ei saavuttanut laajaa suosiota. Nämä kielet kuitenkin toimivat innoittajina "oikeille" kielille. Imperatiivisen ja funktionaalisen paradigman tiet erosivat jo ennen Fortranin ilmestymistä.

2.1 Varhaiset korkean tason kielet

Korkean tason ohjelmointikieliä alettiin tutkia 1940-luvulla. Ensimmäisenä korkean tason kielenä pidetään Konrad Zusen Plankalküliä, jonka hän suunnitteli vuonna 1945. Plankalkül oli hämmästyttävän ilmaisuvoimainen: se sisälsi esimerkiksi työkalut uusien tietotyyppien määrittelyyn, joka tuli oikeissa ohjelmointikielissä mahdolliseksi vasta 60-luvulla ja aidosti käyttökelpoiseksi vieläkin myöhemmin. Zusen työt julkaistiin kuitenkin vasta 1972, jolloin niillä oli enää akateemista mielenkiintoa.

Ensimmäinen laajalti tunnettu ohjelmointikieli oli John Mauchlyn BINAC-koneelle suunnittelema Short Code vuodelta 1949. Short Code oli kuitenkin käsin käännettävä kieli. Ensimmäinen kieli, jolle toteutettiin myös kääntäjä, oli vuonna 1952 julkaistu AUTOCODE. Näitä seurasi useita kokeellisia ohjelmointikieliä: ennen Fortrania suunniteltiin yhteensä parikymmentä kieltä, joista yksikään ei tullut laajaan käyttöön. Valtaosa kielistä oli imperatiivisia, mutta USA:n puolustusvoimien ADES-projektin myötä myös funktionaalinen paradigma sai alkunsa [Knuth77].

Ensimmäiset korkean tason ohjelmointikielet olivat lähinnä tutkimusmielessä tehtyjä ja ad hoc -tyyppisiä kokeiluja. Useimmat olivat yksittäisten ihmisten ideoihin perustuvia. Kielten notaatiot olivat keskenään hyvin erilaisia (joskin poikkeuksetta täysin erilaisia kuin nykyiset) ja keskeisenä ajatuksena vaikutti olevan jonkinlaisen täsmällisen, matemaattista muistuttavan notaation käyttöönotto. Abstraktiomenetelmiä ei juurikaan ollut käytössä, vaan kielissä oli käytettävissä vain jo Ada Augusta Lovelacen käyttämät peräkkäisyys, ehdollisuus ja toisto.

2.2 Fortran

Korkean tason ohjelmointikieliä tutkittiin ohjelmoinnin helpottamiseksi. AUTOCODEn kehittäjä Glennie kirjoitti vuonna 1952:

> "The difficulty of programming has become the main difficulty in the use of machines. – – Present notations have many disadvantages: all are incomprehensible to the novice, they are all different (one for each machine) and they are never easy to read." [Knuth77]

Vaikka konekielen ymmärrettiin olevan hankalaa ja konekohtaista, olivat laskentatehot riittämättömät korkean tason kielten laajalle käyttöönotolle. Kääntäjätekniikka oli lapsenkengissään eikä tuotetun koodin nopeus mitenkään vetänyt vertoja käsin kirjoitetulle konekielelle.

Vuonna 1954 John Backusin johtama työryhmä aloitti Fortranin ensimmäisen version suunnittelun IBM 704-tietokoneelle. Sen tarkoituksena oli helpottaa ohjelmointia mutta tuottaa silti ohjelmia, jotka olisivat yhtä nopeita kuin konekielellä kirjoitetut. Fortrania suunniteltiin vain IBM:n tietokoneita varten, mutta kun koneriippuvaisesta notaatiosta päästiin eroon, osoittautui Fortranin siirtäminen muillekin tietokoneille mahdolliseksi. Kielen ensimmäinen spesifikaatio julkaistiin vielä vuoden 1954 aikana [Knuth77].

Fortranin ensimmäinen toteutus julkaistiin 1957. Jo ennen julkaisua se oli saanut huomattavasti julkisuutta ja löikin itsensä läpi nopeasti. Vaikka se oli julkaisuhetkellään aikataulustaan jäljessä (suunniteltu julkaisuajankohta oli ollut loppuvuodesta 1956), keskeneräinen eikä edes täysin toimiva, osoitti jo seuraavana vuonna tehty tutkimus IBM 704-tietokoneiden käyttäjistä puolien käyttävän Fortrania ainakin puoleen työstään [Knuth77].

2.3 Ensimmäiset funktionaaliset kielet: FLPL ja LISP

Ajatus ohjelmien kuvaamisesta matemaattisina funktioina esiteltiin ensimmäisenä ADES-kielessä, joka kehitettiin vuosien 1955-56 aikana. ADESin kontrollirakenteet perustuivat funkioiden rekursiivisuuteen, mutta lauseidensa puolesta kieli oli lähinnä imperatiivista [Knuth77].

Tunnetuin funktionaalinen ohjelmointikieli lienee edelleen käytössä oleva LISP. Siitä kaavailtiin tekoälyohjelmointiin soveltuvaa kieltä ja listapohjaisuuden katsottiin sopivan tähän tarkoitukseen. Idea listoista oli peräisin IPL 2-kielestä, jota oli käytetty

logiikan ongelmien ratkaisemiseen. Muut rakenteet olivat pääosin Fortranista, jonka algebramuotoiset lauseet vetosivat suunnitteluryhmään [McCarthy79].

LISPiä ryhdyttiin suunnittelemaan kesällä 1956. Sitä kehiteltiin aluksi Fortranin laajennuksena nimellä FLPL (Fortran List Processing Language), joka nimensä mukaisesti lisäsi Fortraniin listojenkäsittelyn. Sitä ei kuitenkaan voida pitää funktionaalisena kielenä: erityisesti siitä puuttuivat rekursiiviset funktiot ja ehtolausekkeet.

LISP 1:n toteutus aloitettiin syksyllä 1958. Tässäkin versiossa kontrollilauseet, mm. goto-lause, olivat peräisin Fortranista. Ehtolausekkeet ja rekursio, roskienkeruu sekä funktioiden matemaattinen muoto otettiin mukaan uusina piirteinä. Vielä 1959 ulos ehti LISP 1.5, joka korjaili joitakin LISP 1:n puutteita. Näistä merkittävin oli käyttökelpoinen laskenta: LISP 1:ssä luvut kuvattiin listoina, mikä teki siitä aivan liian hitaan minkäänlaiseen käytännön laskentaan. Tosin vielä 1.5-versiossakin laskenta oli "ultrahidasta" (n. 10-100 kertaa Fortrania hitaampaa) [McCarthy79], koska LISP oli tyypitön kieli eikä luvuille ollut käytettävissä muusta datasta poikkevaa käsittelyä.

3 Evoluutiota 60-luvulla

Kun Fortran oli 50-luvulla osoittanut korkean tason ohjelmoinnin mielekkääksi ideaksi, oli 60-luvulla aika ryhtyä jalostamaan sitä. Imperatiivisia kieliä ryhdyttiin kehittämään siistimpään muotoon ja funktionaalisella puolella LISP alkoi eriytyä enemmän imperatiivisista kielistä. 60-luvulla nähtiin myös ensimmäinen olio-ohjelmointikieli, tässä vaiheessa vielä tiukasti imperatiivisen paradigman osana.

3.1 Spagetista rakenteeseen: ALGOL

Heti 60-luvun alussa imperatiivinen paradigma otti ALGOLin ilmestymisen myötä reilun harppauksen eteenpäin. 1950-luvun lopulla Association of Computing Machinery ja joukko eurooppalaisia instituutioita olivat perustaneet komitean suunnittelemaan tieteelliseen laskentaan soveltuvaa ohjelmointikieltä [Lindsey93]. ALGOLista kaavailtiin universaalia kieltä, joka mm. korvaisi pseudokoodin algoritmien kuvaamisessa tieteellisissä julkaisuissa. ALGOL oli myös ensimmäisiä kieliä, joissa ohjelmien siirrettävyys oli virallisena suunnittelutavoitteena.

ALGOL 60 esitteli koko joukon uusia ominaisuuksia, kuten lohkorakenteen, leksikaaliset näkyvyyssäännöt ja rekursiiviset aliohjelmat[2]. Näillä piirteillä oli niin suuri

[2]Rekursiiviset *funktiot* olivat käytössä jo aiemmin.

merkitys, että lähes kaikkia sen jälkeen kehitettyjä imperatiivisia kieliä on kutsuttu ALGOLin seuraajiksi [WWW-LG96].

ALGOL 60 saavutti standardin aseman tiedeyhteisössä, mutta teollisella puolella valtaa pitivät Fortran ja vuonna 1959 kaupallis-hallinnolliselle alalle kehitetty COBOL, joiden kohdeyleisö oli laajempi kuin ALGOL 60:n. Tätä ongelmaa ratkaisemaan muodostettiin komitea, joka pitkän ja monimutkaisen prosessin tuloksena sai aikaan ALGOL 68:n. Se ei kuitenkaan lisännyt ALGOLin suosittuutta [Lindsey93].

1960-luku oli kulta-aikaa uusille imperatiivisille kielille, joita ilmestyi ALGOLin perässä useita. Näistä mainittakoon matriisipohjainen, matematiikan sovelluksiin käytetty APL, merkkijonokieli SNOBOL, joka on osittain vaikuttanut nykyisten merkkijonokielten kuten Perlin ja Tcl:n rakenteisiin sekä PL/1, joka oli IBM:n yritys luoda täysin yleiskäyttöinen imperatiivinen kieli [WWW-LG96].

3.2 Reformismin aika

Kun korkean tason kielet alkoivat 60-luvun alussa ALGOLin myötä eriytyä kunnolla konekielestä, alkoivat eräät ohjelmoijat kritisoida niiden konekielimäisiä rakenteita. Hankaliksi ja turvattomiksi koettuja rakenteita haluttiin poistaa kielistä tai ainakin niillä toteutetuista ohjelmista. Kritiikin kohteeksi joutuivat osoittimet, globaalit muuttujat ja funktionaalisen paradigman kehittyessä jopa sijoituslause, mutta ennen kaikkea tulilinjalla oli pahamaineinen goto-lause [Knuth74].

Knuth [Knuth74] arvioi D.V. Schorren vuonna 1960 alkaneet kokeilut ensimmäisiksi tietoisiksi yrityksiksi välttää goto:jen käyttöä ohjelmoinnissa. Ensimmäinen aihetta julkaisussaan käsitellyt ohjelmoija taas oli Peter Naur vuonna 1963. Seuraavina vuosina goto:n vastustajia ilmestyi hiljalleen lisää [Knuth74].

Suurimman kohun sai kuitenkin aikaan E.W. Dijkstran klassikoksi noussut, Communications of the ACM:ssä vuonna 1968 julkaistu artikkeli *Go To Statement Considered Harmful* [Dijkstra68]. Siinä Dijkstra arvioi goto-lauseen lisäksi sen käyttäjiä, mikä käynnisti kiivaan, pitkälle 1970-luvulle jatkuneen keskustelun siitä, millaisia piirteitä ohjelmointikielissä pitäisi olla. Dijkstran julkaisun jälkeen kirjoitettiin näytösmielessä kääntäjiä ja käyttöjärjestelmiä mahdollisimman vähäisellä goto:jen käytöllä ja ACM pyhitti goto:jen poistamiselle yhden kokonaisen konferenssin vuonna 1972 [Knuth74].

3.3 LISP 60-luvulla

60-luvulla funktionaalinen paradigma oli yhtä kuin LISP. Kieltä oli tähän asti käytetty pääasiassa tekoälyohjelmointiin. Vuoden 1962 jälkeen, versioiden 1.5 ja 2 välissä, LISPin kehitys hajaantui ja eri ryhmät alkoivat toteuttaa erilaisia versioita siitä.

LISPin ja funktionaalisen ohjelmoinnin edut alkoivat selvitä tutkijoille 60-luvun aikana [McCarthy79]. Kun Fortranista oli päästy eroon, alkoi aikaansaatu kieli vaikuttaa esteettisesti kauniilta. Kävi ilmi, että LISP oli paitsi käyttökelpoinen kieli myös elegantti matemaattinen järjestelmä. Tästä matemaattisesta asusta muodostui sittemmin tavoite, joka johti monien jo toteutettujen imperatiivisten ominaisuuksien poistamiseen kielen ytimestä.

Kehittäjät uskoivat, että matematiikkaa voitaisiin käyttää apuvälineenä oikeellisten ohjelmien rakentamiseen. Tämä olettamus on sittemmin osoittautunut paikkansapitäväksi: 70-luvulla McCarthy ja Cartwright osoittivat, että LISPin rakenteet voidaan tulkita matemaattisen logiikan lauseiksi [McCarthy79]. Tästä syystä on mahdollista esimerkiksi todistaa funktioiden oikeellisuus matemaattisin menetelmin.

3.4 Olio-ohjelmoinnin alkuhetket: Simula

Vaikka oliokielet tulivat yleiseen tietoisuuteen vasta 1980- ja 90-luvuilla, nähtiin niiden alkuhetket jo 60-luvulla, kun Ole-Johan Dahl ja Kristen Nygaard julkaisivat Simula-ohjelmointikielensä. Vaikka Simula ei itse koskaan saavuttanut laajaa suosiota, se toimi perustana muille kielille, esimerkiksi C++:lle [Stroustrup97].

Norjan tieteellisessä laskentakeskuksessa työskentelevä Kristen Nygaard kiinnostui jo 1950-luvun alussa monimutkaisista todellisen maailman järjestelmistä, joiden toiminnan heterogeenisuuden kuvaaminen oli osoittautunut haasteelliseksi [Holmevik95]. Korkean tason ohjelmointikielten tultua yleiseen käyttöön Nygaard halusi kehittää simulaatioihin painottuvan ohjelmointikielen. Hän keksi kuvata tietorakenteilla todellisen maailman malleja. Nygaard ei kuitenkaan osannut toteuttaa ohjelmointikieltä itse, vaan tarvitsi avukseen Ole-Johan Dahlin.

Simula-1, joka julkaistiin vuonna 1966, soveltui ainoastaan simulaatioihin eikä se ollut todella oliopohjainen. C.A.R. Hoaren ehdotuksesta kielen seuraavaan versioon Simula-67:ään otettiin mukaan tietueet, jolloin todellisen maailman rakenteiden oliopohjainen mallintaminen tuli mahdolliseksi. Tämä versio perustui euroopassa suosituksi tulleeseen ALGOL 60:een. Simula-67 oli kuin "ALGOL 60++": se oli pääosin yhteensopiva ALGOLin kanssa, mutta sisälsi lisänä luokat, oliot, periytymisen ja

virtuaalifunktiot. Simulan kolmas versio julkaistiin kymmenen vuotta myöhemmin. Siinä oli joitakin pieniä parannuksia 67-versioon (mm. kapselointi) [WWW-LG96].

4 70-luku: moduulit ja oliot

1970-luvulla ohjelmistot alkoivat käydä niin suuriksi, että niiden osittamiseksi tarvittiin uusia menetelmiä. Modulaarinen ohjelmointi tarjosi osittaisen ratkaisun tähän ongelmaan. Myös olio-ohjelmointi kehittyi ja herätti uutta mielenkiintoa. LISPiä kehiteltiin useassa paikassa rinnakkain ja se hajosi useiksi murteiksi.

4.1 Lohkorakenteesta moduleihin

Modulaarisen ohjelmoinnin hyödyt oli havaittu jo 1960-luvun alussa. Idea datan ja aliohjelmien ryhmittelystä toiminnallisuuden perusteella sekä toteutuksen piilottamisesta oli luonnollinen oivallus, johon monet eri tahot päätyivät toisistaan riippumatta [Sethi96]. Parnas kirjoitti vuonna 1972:

> The major advancement in the area of modular programming has been the development of coding techniques and assemblers which (1) allow one module to be written with little knowledge of the code in another module, and (2) allow modules to be reassembled and replaced without reassembly of the whole system. [Parnas72]

Ilmeisesti korkean tason kielissä tarve toteutuksen piilottamiselle ja koodin staattiselle osittamiselle ei tullut vastaan läheskään yhtä nopeasti kuin konekielessä, koska korkean tason kielet alkoivat tukea modulaarista ohjelmointia vasta 1970-luvun puolella.

70-luvun aikana ilmaantui useita edelleen merkityksellisiä kieliä. Niklaus Wirth aloitti Pascalin suunnittelun 1968 ja sen ensimmäinen toteutus julkaistiin vuonna 1970. Se oli pieni ja siisti, mutta ennen kaikkea sen menestys on seurausta sen useista laadukkaista toteutuksista [Sethi96]. 1980 Wirth julkaisi Pascalin seuraajan Modula-2:n, joka sisälsi mm. suoran tuen modulaariselle ohjelmoinnille.

Toinen, vielä merkittävämpi uusi kieli oli C, joka kehitettiin AT&T Bell Laboratories:issa Unix-käyttöjärjestelmän ohjelmointikieleksi [Ritchie93]. Myös C tuki modulaarista ohjelmointia, vaikka moduulin käsitettä ei siinä voi suoraan ilmaista. Moduulia vastaa C:ssä käännösyksikkö, jonka sisään aliohjelmien toteutuksen voi piilottaa.

Muita modulaarista ohjelmointititapaa tukevia 70-luvun ohjelmointikieliä olivat Clu ja Mesa, joista jälkimmäinen toimi osittain mallina Modula-2:n rakenteille [Sethi96].

4.2 Smalltalk

Olioparadigma sai uutta tuulta purjeisiinsa 70-luvun alussa, kun Alan Kay siirtyi Utahin yliopistosta Xerox Parcin Dynabook-projektiin [WWW-Squeak]. Siinä oli tarkoituksena kehittää yleiskäyttöinen kannettava henkilökohtainen tietokone, jota myös tietokoneisiin perehtymättömät ihmiset voisivat käyttää. Vision mukaan käyttäjät ohjelmoisivat tietokoneitaan itse, mutta käytettävissä olevat ohjelmointikielet vaativat aivan liikaa tietokoneiden tuntemusta. Tarvittiin siis uusi, yleistajuinen ohjelmointikieli.

Kayllä oli aiempaa kokemusta Simulasta, jonka oliomalli sopi Dynabook-projektiin: koska käyttäjien piti pystyä laajentamaan olemassaolevaa järjestelmää dynaamisesti, haluttiin uuden ohjelmointikielen perustuvan periytymiseen ja kapselointiin. Smalltalkista rakennettiin alusta pitäen paitsi ohjelmointikieltä myös ympäristöä sovellusten kehittämiseen ja suoritukseen.

Smalltalkin ensimmäinen kokeellinen versio saatiin toimimaan vuoden 1972 lopulla. Vuonna -74 sille alettiin rakentaa pieniä sovelluksia. Tässä vaiheessa kieli perustui täysin viestinvälitykseen, mutta muut olio-ominaisuudet puuttuivat. Ensimmäinen todella oliopohjainen versio otettiin käyttöön 1976, kun järjestelmää siistittiin ja periytyminen otettiin käyttöön. Samalla siirryttiin tavukoodipohjaiseen arkkitehtuuriin. Kehitys jatkui aina vuoteen 1980 asti, jolloin kielen ensimmäinen julkaisukelpoinen versio Smalltalk-80 saatiin valmiiksi.

Smalltalkin edelliset versiot olivat pysyneet Xerox Parcin seinien sisällä, vaikka satunnaiset julkaisut ja asiantuntijoiden vierailut Parcissa olivat herättättäneet kiinnostusta myös muualla. Smalltalk-80 päätettiin lopulta julkaista. Byte Magazine omisti elokuun 1981 numeronsa kokonaan Smalltalkille ja oliot tulivat lopullisesti laajan yleisön tietoisuuteen.

4.3 LISPin murteet

LISP oli 70-luvulla edelleen käytännössä yksin funktionaalisella puolella. Kielen kehitys oli kuitenkin levinnyt alkuperäisen työryhmän ulkopuolelle ja siitä rakennettiin erilaisia murteita useille tietokoneille [SteGab93]. Tämän päivän näkökulmasta huomattavin näistä murteista oli Scheme, edelleen käytössä oleva siistitty LISP. Para-

digmavertailun kannalta merkittävä uudistus 70-luvulla oli Flavours-murteen ilmestyminen. Se oli oliopohjainen LISP, joka toimii hyvänä esimerkkinä olioparadigman monimuotoisuudesta: oliot eivät ole vain imperatiivisia kieliä varten.

5 Myöhemmät vaiheet

Vuonna 1981 Japani ilmoitti ryhtyvänsä kehittämään viidennen sukupolven tietokonetta, jonka perustana toimisivat tekoälykielet. Neljä vuotta aiemmin USA:n armeija oli alkanut investoida raskaasti imperatiiviseen Adaan ja ilmoittikin vuonna 1983 jatkavasa panostusta. 80-luvusta uskottiin tulevan taistelun Adan ja tekoälykielten välillä. Toisin kuitenkin kävi.

5.1 Imperatiiviset ja oliopohjaiset kielet

Imperatiivisten kielten voittokulku jatkui 70-luvun jälkeenkin. Yhtäältä pienet ja yksinkertaiset kielet, kuten Pascal ja C, olivat suuressa suosiossa. Toisaalta huomattavasti siistittyjä kieliä, kuten Modula-2 ja Oberon, alkoi ilmestyä. Sovellusten kasvaessa moduulirakenteesta tuli olennainen osa lohkorakenteisia imperatiivisia kieliä.

Smalltalk aiheutti 80-luvun alussa valtavaa innostusta ja olioita tutkittiin ratkaisuna ohjelmistotuotannon keskeisiin ongelmiin. 1980 Bjarne Stroustrup esitteli idean Simula-tyyppisten luokkien lisäämisestä C-kieleen. Tästä ehdotuksesta kasvoi hiljalleen C++, joka löi itsensä läpi jo kehitysvaiheessa. Vaikka puristit eivät pidä sitä oikeana oliokielenä, todisti se ohjelmoivalle yleisölle olioiden elinvoimaisuuden [Sethi96].

5.2 Funktionaaliset kielet

Teollisuuden mielenkiinto funktionaalisia kieliä kohtaan alkoi hiipua. Vaikka LISPiin ei tänäänkään voi olla törmäämättä tutkiessaan uusia ohjelmointikielijulkaisuja, ei funktionaalisella ohjelmoinnilla ilmeisesti ollut teollisten tahojen mielestä riittävästi tarjottavaa.

Toisaalta LISPin puhtaasti funktionaalinen osajoukko vaikutti akateemisissa piireissä kiinnostavammalta. Kun tietotekniikka oli osoittautunut jatkuvasti ja vauhdikkaasti kehittyväksi alaksi, havaittiin funktionaaliset kielet matemaattisen taustansa vuoksi riippumattomiksi perinteisten tietokoneiden pullonkauloista [Sethi96].

80- ja 90-luvulla LISP sai vihdoin seuraa muista funktionaalisista kielistä. ML-kielen myötä päästiin hyödyntämään LISPin matemaattista luonnetta ilman hankalaa syntaksia. ML myös toi funktiokieliin tyypityksen, jota LISPissä ei koskaan ollut. Myöhemmin kehiteltiin puhtaasti funktionaalisia kieliä, kuten Haskell ja Miranda. Nämä kielet eivät salli funktioille lainkaan sivuvaikutuksia.

80-luvulla LISP saatiin vihdoin myös standardisoiduksi. Tuloksena oli Common LISP, huomattavan suuri kieli, jonka ytimenä LISP toimi. Sen perusrakenteet olivat edelleen peräisin varhaisista LISPeistä. Kieleen lisättiin myös oliolaajennus Common LISP Object System (CLOS). LISPiä ei tarvinnut muuttaa tätä varten – CLOS oli funktio- ja makrokirjasto. Tästä huolimatta se sisälsi mm. täysimittaisen, äärimmäisen voimakkaan metaprotokollan.

6 Yhteenveto

Ohjelmointikielet ovat 1940-luvun jälkeen jalostuneet säännönmukaisemmiksi, helppokäyttöisemmiksi ja abstraktimmiksi. Tähän on vaikuttanut kolme asiaa: ohjelmointikielten ymmärryksen kehitys, kasvaneet laitetehot ja sovellusten koon kasvaminen. Toisaalta tämä näkemys on jakautunut eri ohjelmointiparadigmojen kesken: imperatiivisessa paradigmassa päästään parhaisiin suorituskykyihin, olioparadigma soveltuu suurten ohjelmistojen luontevaan osittamiseen ja funktionaalisessa paradigmassa voidaan luottaa matemaattiseen formalismiin ohjelmien tuottamisessa.

Olioparadigmalla vaikuttaa 2000-luvun alussa olevan eniten kannatusta. Vaikka funktionaalisella ohjelmoinnilla on puolestapuhujia erityisesti tiedeyhteisössä, LISPin ja muiden funktionaalisten kielten käyttö teollisessa sovelluskehityksessä on varsin vähäistä. Ohjelmistot kasvavat kuitenkin edelleen eikä olio-ohjelmoinnin abstraktiomalli täysin sovellu kaikkiin tehtäviin. Näiden ongelmien ratkaisemiseksi on jo olemassa joitakin ideoita, esimerkkeinä *reflektiivinen ohjelmointi* [Ferber89] ja *aspektipohjainen ohjelmointi* [Kiczales97].

Viitteet

[Dijkstra68] Dijkstra, E.W.: *Go To Statement Considered Harmful.* Communications of the ACM, Vol.11, No.3 Maaliskuu 1968.

[Ferber89] Ferber, J.: *Computational Reflection in Class based Object Oriented Languages.* Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, sivut 317–326, 1989.

[Holmevik95] Holmevik, J.R.: *The History of Simula.*
`http://java.sun.com/people/jag/SimulaHistory.html`
1995. Viitattu 3.5.2001.

[Kiczales97] Kiczales, G. et al: *Aspect-Oriented Programming.* Proceedings of the European Conference on Object-Oriented Programming, Springer-Verlag, kesäkuu 1997.

[Knuth74] Knuth, D.E.: *Structured Programming with go to Statements.* Computer Sciene Department, Stanford University, 1974,

[Knuth77] Knuth, D.E., Pardo, L.T.: *The Early Development of Programming Languages.* Computer Sciene Department, Stanford University, 1977.

[Lindsey93] Lindsey, C.H.: *A history of ALGOL 68.* ACM SIGPLAN Notices, vol 28, nro 3, maaliskuu 1993.

[McCarthy79] McCarthy, J.: *History of Lisp.*
`http://www-formal.stanford.edu/jmc/history/lisp/lisp.html`
päivätty 1996, viitattu 30.4.2001.

[Parnas72] Parnas, D.L.: *On the Criteria To Be Used in Decomposing Systems into Modules.* Communications of the ACM, vol.15, nro 12, joulukuu 1972.

[Ritchie93] Ritchie, D.M.: *The Development of the C Language.* ACM SIGPLAN Notices, vol 28, nro 3, sivut 201–208, maaliskuu 1993.

[Sethi96] Sethi, R.: *Programming languages: concepts & constructs. 2.ed..* Addison-Wesley, 1996.

[SteGab93] Steele, G.L, Gabriel, R.P.: *The Evolution of Lisp.* ACM SIGPLAN Notices, vol 28, nro 3, sivut 231–270, maaliskuu 1993.

[Stroustrup97] Stroustrup, B.: *The C++ Programming Language, 3.ed.*. Addison Wesley, 1997.

[WWW-LG96] University of Michigan, Computer and Information Science: *The Language Guide.*
`http://www.engin.umd.umich.edu/CIS/course.des/cis400/`
Yksittäisiä sivuja päivätty 1996 ja 1999, viitattu 30.4.2001.

[WWW-Squeak] *Basic Aspects of Squeak and the Smalltalk-80 Programming Language.*
`http://www.cosc.canterbury.ac.nz/`
`~wolfgang/cosc205/smalltalk1.html`
Päivätty 1998, viitattu 3.5.2001.