Välikielet

Jari Saukkonen

Helsinki 29.3.2004

Ohjelmointikielten periaatteet - seminaariesitelmä

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Välikielet tarjoavat abstraktiotason ohjelmointikielen kääntäjän ja suoritusympäristön välille. Eri välikielet on toteutettu erilaisista suunnittelulähtökohdista, jotka heijastavat kielen kanssa käytettäviä ohjelmointikieliä ja vaikuttavat suoritusympäristön toteutukseen. Tutkielman tavoitteena on perehtyä välikieliin, ja tutkia mitä vaikutuksia tehdyillä suunnittelupäätöksillä on näitä kieliä tuottaviin kääntäjiin sekä niiden suoritusympäristöihin.

2 Tarkasteltavat kielet

Tarkasteltavaksi on valittu muutama yleinen välikieli, jotka eroavat toisistaan merkittävästi joillain osa-alueilla: Java bytecode on Java-kääntäjien käyttämä välikieli, MSIL on Microsoftin .NET-ympäristössä toimivien kääntäjien tuottama välikieli, ja Parrot on Perl Foundationin tuottama virtuaalikone tulkattavien kielten tehokkaaseen suoritukseen.

2.1 Java

Java bytecode (tässä lyhennettynä Java) on Sun Microsystemsin Java-ympäristön [GJSB00] käyttämä välikieli. Tätä välikieltä suorittava JVM (Java Virtual Machine) [LY99] toimii pinokoneena: jokainen käsky ottaa operandinsa operandipinon päältä ja tallentaa tuloksensa operandipinon päällimmäiseksi.

Välikieli on vahvasti tyypitetty, joten kutakin operaatiota kohti on olemassa erilliset käskyt eri tyyppiluokkien käsittelyä varten. Käytössä on Java-kielen alkeistyyppejä vastaavat tyypit totuusarvolle, kokonaisluvuille, liukuluvuille, kaksitavuinen unicode-merkki merkkijonoja varten, sekä viitetyyppi olioviittauksia varten.

Koska välikielet toimivat konekieleen nähden yhtä tasoa ylempänä, luokkahierarkiat, olioiden instantiointi, tyyppimuunnokset ym. oliokäsitteet ovat oliojärjestelmien välikielissä selkeästi näkyvillä.

Java ei sisällä datanmäärityskäskyjä, vaan luokkavakiot ja -muuttujat on tallennettu luokka- ja metodikohtaisiin rakenteisiin sekä pinoon. Välikielestä käsin niihin viitataan indekseillä, joten itse vakioita ei tarvitse toistaa suoritettavan koodin sisäl-

lä. Javan erikoisuutena usein käytettyjen vakioiden, kuten pienten kokonaislukujen, lataamiseen on olemassa omat lyhyet käskynsä. Tällöin näitä vakioita ei tarvitse tallentaa luokan vakioihin, vaan ne voidaan lisätä pinoon operandittomalla yksitavuisella käskyllä.

Javan käskykanta ei sinänsä käsittele poikkeuksia, sillä käskykannassa on vain käsky poikkeusten heittämistä varten. Poikkeusten käsittely on toteutettu erillisellä poikkeustaululla, johon on kirjattu miltä koodialueilta on odotettavissa poikkeuksia (`try`-lohko), ja mihin poikkeuksen tullessa täytyy hypätä (`catch`-lohko).

2.2 Microsoft IL

MSIL [Mic00] on Microsoftin .NET-ympäristössä käyttämä välikieli. Javan lailla MSIL on toteutus pinoarkkitehtuurista.

Suurimpana käsitteellisenä erona Javaan MSIL:n muuttujat eivät ole yhtä tiukasti tyypitettyjä. Tiedot muuttujien tyypeistä on tallennettu mukaan pinoon. Täsmällinen suoritettava operaatio määräytyy siis sekä käskyn, että operandityyppien perusteella. Kaikki mahdolliset operandiyhdistelmät eivät kuitenkaan ole sallittuja kaikille käskyille, joten kääntäjän täytyy kieltää mahdottomat yhdistelmät. Näiden tarkkailu ei kumminkaan ole vedenpitävästi mahdollista, sillä kääntäjällä ei välttämättä ole riittävää tyyppitietoa tarkistaa sitä käännösaikaisesti. Tällöin vastuu operaatioiden oikeellisuuden varmistamisesta siirtyy suoritusympäristölle.

Kokonais- ja liukulukujen käsittelyssä on olemassa perustavanlaatuisia eroja. Normaalisti liukulukuaritmetiikassa mahdolliset yli- ja alivuodot havaitaan ja käsitellään. Kokonaisluvut eivät kumminkaan pysty normaalisti havaitsemaan tai tallentamaan yli- tai alivuototilanteita. Koska MSIL:ssä sekä kokonais- että liukulukujen aritmetiikka tapahtuu samoilla käskyillä, on käskykannassa erilliset ylivuodon tunnistavat käskyt.

MSIL:ssä poikkeukset käsitellään vastaavalla tavalla kuin Javassa. Poikkeustauluun tallennetaan `try`- ja `catch`-lohkojen sijainnit, ja suoritusympäristö huolehtii tarvittavista hypyistä näiden suhteen.

2.3 Parrot

Parrot [Per04] on Perl Foundationin tuottama virtuaalikone tulkattavien kielten tehokkaaseen suoritukseen. Parrotin juuret ovat Perlissä, ja sen tavoitteena on lopulta päätyä Perl 6:n virtuaalikoneeksi, ja mahdollisesti myös toimia virtuaalikoneena muille tulkattaville kielille. Sen tavoitteena ei ole ainoastaan pystyä suorittamaan koodia, vaan myös tarjoamaan korkeamman tason palveluita sitä käyttäville kielille.

Perl-historia näkyy erityisesti merkkijonojen käsittelyssä, sillä Parrotin käskykanta sisältää yleisimmät merkkijono-operaatiot sekä säännölliset lausekkeet. Lisäksi käskykannasta löytyy käskyjä yleisimpien käyttöjärjestelmän palveluiden, kuten tiedostojen ja kellon käyttöön.

Toisin kuin Java ja MSIL, Parrot perustuu rekisteriarkkitehtuuriin. Käytössä on neljä 32 rekisterin sarjaa, yksi sarja jokaista perustietotyyppiä kohti. Perustyyppejä ovat kokonaisluku, liukuluku, merkkijono ja *PMC*-tyyppi (Parrot Magic Cookie), joka käsittää monimutkaisemmat tietorakenteet. Kuten rekisteriarkkitehtuureissa on tapana, käskyt sisältävät operandeinaan aina tarvittavat lähde- ja kohderekisterit.

PMC-tyypit ovat Parrotin tapa toteuttaa monimutkaisemmat tyypit, ja eri kielten pienet erot operaatioiden välillä. Operaatioita ei ole yleisesti määritelty, vaan esimerkiksi `inc`-käskyn kutsuminen PMC-oliolle aiheuttaa kutsun olion `increment`-metodiin. Näin ollen eri ohjelmointikieltä varten voidaan toteuttaa kielelle ominaisten tyyppien ominaisuudet.

Poikkeusten käsittely Parrotissa lähtee välikielen tasolta. Parrot sisältää käskyn, jolla voi muuttaa aktiivista poikkeuskäsittelijää, jota kutsutaan jos suorituksen aikana heitetään mikä tahansa poikkeus. Tämän avulla suoritusympäristö voi toteuttaa monimutkaisemmat poikkeuskäsittelyjärjestelmät.

3 Välikielten ominaisuuksia

Esitellyt kielet painivat jokseenkin samassa ongelmakentässä, joten niillä on myös suurinpiirtein samat ongelmat ratkaistavanaan. Näissä kielissä niitä on ratkaistu hieman eri tavoilla, joilla on omat hyvät ja huonot puolensa. Lisäksi kielistä löytyy ratkaisuja, joiden tarkoituksena on puhtaasti tehostaa suoritusympäristön toimintaa.

3.1 Pinon käsittely

Pinokoneina toimivilla kielillä käskykannat sisältävät huomattavasti pinon toimintaa tehostavia operaatioita. Erityisesti on kiinnitetty huomiota operandien lataamiseen, sillä sekä MSIL ja Java sisältävät normaalin parametroitujen latauskäskyjen lisäksi erikoiskäskyt muutaman ensimmäisen lokaalimuuttujan, argumentin tai numeeristen vakioiden lataamiseen. Näistä erikoiskäskyistä on merkittävä hyöty nk. peephole-optimointien yhteydessä. Peephole-optimoinneilla tarkoitetaan hyödyttömien koodinpätkien poistamista jo tuotetusta koodista.

Myös syntyvän koodin kokoon on kiinnitetty huomiota, sillä tällaiset erikoiskäskyt säästävät jokaisessa funktiossa karkeasti funktion parametrien käytön määrään suhteessa olevan määrän tilaa lopullisessa bittiesityksessään. Useimmissa funktioissa on vain muutama parametri, ja ne joudutaan funktion suorituksen aikana lataamaan pinoon vähintään kerran, pitkissä funktioissa helposti useampia kertoja.

3.2 Tyypitys

Tyyppi-informaation tallennus välikieleen lisää tarvittavien operaatioiden määrää, mutta yksinkertaistaa suoritusympäristöä. Kun tyypit ovat etukäteen tunnettuja, ei operaatioiden tarvitse ottaa kantaa tyyppitietoon, vaan ne voidaan suoraan suunnitella toimiviksi tietyntyyppisillä operandeilla.

Javassa laskentaa voidaan suorittaa kahdella kokonaislukutarkkuudella sekä kahdella liukulukutarkkuudella. Tämä yksinkertaistaa virtuaalikoneen rakennetta, sillä esimerkiksi normaalia kokonaislukua (`int`) pienemmät alkeistyypit (`byte`, `char`, `short`) ovat esitykseltään identtisiä normaalin kokonaisluvun kanssa. Tällöin ei tarvita erillisiä pienillä tyypeillä toimivia aritmeettisia operaatioita. Sivuvaikutuksena aritmeettisen lausekkeen arvo on aina vähintään `int`-tyypin kokoinen, joten Javassa lause `byte b = (byte)b1 + (byte)b2;` antaisi varoituksen yrityksestä sijoittaa `int`-tyyppinen arvo `byte`-muuttujaan.

Parrotissa alkeistyyppiluokkia lukuunottamatta tyypitys on löyhää ja PMC-tyypin varassa. Itse välikieli ei siis sinänsä ota juuri kantaa suoritettaviin operaatioihin, vaan ne on jätetty käytettävien tyyppien määriteltäväksi.

Vahvasti tyypitetty kieli on mahdollista sitoa tehokkaammin suorittimen käskyihin. Tarkalla tyyppitiedolla yksinkertaiset operaatiot voivat kuvautua lähes suoraan ko-

nekielisiksi käskyiksi. Jos operaatio on piilotettu abstraktin tyypin taakse, on siitä yleensä odotettavissa vähintään funktiokutsu.

3.3 Pino- ja rekisteriarkkitehtuuri

Päätös pino- tai rekisteriarkkitehtuurin käytöstä on tärkeä kielen kannalta. Pinoarkkitehtuurilla toimiva virtuaalikone on helpompi toteuttaa kieltä tulkattaessa, sillä tällöin ei tarvitse juuri huolehtia menneistä tai tulevista rekisterivarauksista. Se ei kuitenkaan ole kovin tehokas tapa suorittaa laskentaa. Esimerkiksi aritmeettista lauseketta laskettaessa pinoarkkitehtuurilla toteutettu järjestelmä joutuu jatkuvasti tallentamaan välituloksia takaisin pinoon. Jotta tällaisen lausekkeen laskeminen olisi tehokasta, välitulokset pitäisi voida kyetä säilyttämään suorittimen rekistereissä. Käytännössä pinokoneen optimointi tällaiseen tarkoitukseen on kuitenkin vaikeaa, etenkin suoritinarkkitehtuureilla joissa rekistereitä olisi käytettävissä runsaasti.

Rekisteriarkkitehtuuri on lähtökohdaltaan lähempänä suorittimia. Kun data on jaettu jo virtuaalikoneessa rekistereihin, on niiden assosiointi fyysisiin rekistereihin huomattavasti helpompaa. Koska virtuaalikoneessa on usein enemmän rekistereitä, on ongelmana tällöin assosiointien hallinta ja rekisterien ja muistin välisten siirtojen minimointi [Sco00]. Rekisterikoneiden kääntäjien ja optimoijien tuottamisesta tehtyä tutkimusta voidaankin hyödyntää luontevasti virtuaalikoneita kehitettäessä.

4 Yhteenveto

Java, MSIL ja Parrot ovat eri lähtökohdista toteutettuja välikieliä. Tärkeimmät erot niiden välillä ovat kielen tyypitys ja pino- tai rekisterityyppinen toteutus. Vahva tyypitys tekee operaatioista tehokkaampia, mutta samalla nostaa välikielen monimutkaisuutta. Heikko tyypitys pitää välikielen rakenteen yksinkertaisena, mutta johtaa helposti vähemmän tehokkaaseen suoritukseen. Rekisterikoneet ovat helpommin optimoitavissa moderneille suorittimille, mutta pinokoneet ovat huomattavasti helpompia toteuttaa tulkattavissa ympäristöissä.

Lähteet

GJSB00 Gosling, J., Joy, B., Steele, G. ja Bracha, G., *The Java Language Specification*. Addison-Wesley, Boston, Mass., toinen painos, 2000.

LY99 Lindholm, T. ja Yellin, F., *The Java Virtual Machine Specification*. Addison-Wesley, Boston, Mass., toinen painos, 1999.

Mic00 Microsoft Corporation, *MSIL Instruction Set Specification*, 2000.

Per04 The Perl Foundation, *Parrot 0.1.0 User Documentation*, 2004. http://www.parrotcode.org/ (Maaliskuu 2004).

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