$\alpha + \omega$, 2001, Nr. 2, 83–88

Komponentinis programavimas „juodoje dėžėje“

Tatjana Jevsikova
tatjanaj@ktl.mii.lt

Komponentinio programavimo principais paremtas programinės įrangos kūrimas tapo neišvengiamas. Straipsnyje aprašomi pagrindiniai objektinio ir komponentinio programavimo kalbų skirtumai, supažindinama su naujomis programų komponentų kūrimo priemonėmis — Komponentinio Paskalio programavimo kalba ir sistema „BlackBox“, kurioje ši kalba realizuota. Aptariamos galimybės šias priemones panaudoti mokymui.

Straipsnio autorė dirba Matematikos ir informatikos institute, studijuoja informatiką Vilniaus pedagoginiame universitete, domisi programavimu, matematika, muzika.

Programinės įrangos produktai tampa vis sudėtingesni. O sudėtingą daiktą sukurti visą nuo pradžių (nuo nulio) sunku. Racionaliau jį surinkti iš atskirų mažesnių dalių (komponentų). Panašus principas sėkmingai taikomas įvairiose technikos srityse, kur iš atskirų komponentų renkamos sudėtingiausios konstrukcijos (statomi namai, gaminamos mašinos, prietaisai, įrenginiai).

Komponentinio progaramavimo idėja yra kurti programų komponentus, iš kurių būtų galima surinkti norimų galimybių programas, bei tuos komponentus komponuoti. Ši idėja tampa vis populiaresnė: publikacijų komponentinio programavimo tema daugėja eksponentiškai [4]. Tai nėra mada ar eilinė populiari idėja, kuri greitai išnyks. Pagal C. Szyperskį [4], komponentų naudojimas yra „gamtos dėsnis“, kuriam paklūsta visos subrendusios technikos disciplinos.

Jau 1968 m. D. McIlroys prognozavo, kad masinė programinės įrangos komponentų gamyba galėtų padėti išvengti vadinamosios programinės įrangos krizės, apie kurią buvo kalbama 1968 m. Garmiš-Partenkirchene vykusioje konferencijoje. Taigi komponentinio programavimo idėja buvo žinoma jau daugiau kaip prieš 30 metų, bet rimčiau apie ją buvo pradėta kalbėti tik devintojo dešimtmečio antroje pusėje. Tokią pauzę galima paaiškinti tuo, kad programinė įranga yra per daug lanksti, jog greitai būtų galima pajusti neišvengiamą komponentų poreikį.

Kuriant programų komponentus bei juos komponuojant atsiveria galimybės surinkti programas pagal jų vartotojų poreikius, kurie nuolat keičiasi ir auga. Naujiems ir specifiniams poreikiams realizuoti reikės ir naujų komponentų. Juos teks programuoti, bet kitus komponentus, kurie dažniausiai ir sudaro pagrindinę dalį programos, galima paimti iš jau esamų aibės. Toks būdas yra geresnis, negu viską programuoti pačiam, nes užima mažiau laiko, reikalauja mažiau pastangų ir taip sudaro sąlygas greičiau tenkinti naujus vartotojų poreikius.

1. Objektinio ir komponentinio programavimo kalbų pagrindinių bruožų palyginimas

Komponentų kūrimo ir plėtros taisykles specifikuoja objektų modeliai. Šiuo metu dominuojantys yra COM ir Java modeliai. „Microsoft“ kompanijos sukurtas ir plačiai naudojamas rašant komponentus „Windows“ terpei komponentinis objektų modelis COM (angl. *Component Object Model* — Komponentinis objektų modelis) nepriklauso nuo programavimo kalbos, t. y. komponentus galima kurti įvairiomis

kalbomis: C++, Visual Basic, Java, Object Pascal, Komponentiniu Paskaliu.

Natūraliai gali kilti klausimas: „Kokias programavimo kalbas ir kokį objektų modelį geriausiai naudoti kuriant programų komponentus?“ Atsakymas būtų toks: naudoti galima beveik kiekvieną programavimo kalbą, gerai tinka objektinio programavimo kalbos, tačiau yra specialiai šiam tikslui skirtų komponentinio programavimo kalbų, suteikiančių galimybę patikimus programų komponentus rašyti daug greičiau, išvengiant kai kurių klaidų. Tokių kalbų pavyzdžiai – Java, Komponentinis Paskalis. Objektų modelio pasirinkimas priklauso nuo kuriamų komponentų paskirties.

Kuo komponentinio programavimo kalbos skiriasi nuo plačiai naudojamų objektinio programavimo kalbų?

1.1. Komponentinio programavimo kalbos — objektinio programavimo kalbų plėtinys

Pagrindiniai objektinio programavimo kalbų bruožai ir sąvokos — objektas (duomenų ir juos apdorojančių procedūrų — (metodų) rinkinys, klasė (objektų, turinčių vienodą struktūrą, visuma), paveldėjimas, inkapsuliacija (informacijos slėpimas), polimorfizmas.

Komponentinio programavimo kalbos turi visus šiuos charakteringus objektinio programavimo bruožus, tačiau be jų dar turi svarbių papildomų.

Vienas iš tokių bruožų yra *saugumas*. Rašant stambesnes programas apie 50% visų klaidų atsiranda dėl neteisingo atminties skirstymo. Kai objektas nebenaudojamas, t. y. nebelieka į jį nuorodų iš išorės, jo užimama atmintis turi būti atlaisvinta. Bet atlaisvinimas turi įvykti ne anksčiau, negu reikia. Neteisingas atminties naudojimas gali sugadinti atskirus komponentus.

Saugios programavimo kalbos garantuoja atminties vientisumą, t. y. vienas komponentas negali pažeisti kito komponento užimamos atminties.

Komponentiniame programavime į objektą, prieinamą iš išorės, gali kreiptis daugybė kitų objektų, apie kuriuos tas objektas neturi jokios informacijos. Todėl turėtų būti galima nustatyti, ar į atmintinę įkeltas objektas dar tebėra kam nors (t. y. kitam objektui) reikalingas. Tam yra naudojami įvairūs mechanizmai, pavyzdžiui, nuorodų į objektą skaičiavimas: kai nuorodų skaičius sumažėja iki nulio, objekto užimama vieta atmintyje išvaloma. Tai yra vadinamasis *automatinis šiukšlių surinkimas*, kuris nematomas ir leidžia programuotojui nesirūpinti atminties atlaisvinimu. Automatinis šiukšlių surinkimas komponentiniame programavime yra ne prabanga, o tiesiog būtinumas [3]. Deja jo neturi netgi populiariausios objektinio programavimo kalbos, pavyzdžiui, C++.

Kitas specifinis komponentinio programavimo kalbų bruožas yra lankstesnis *informacijos slėpimas*. Daugelyje objektinio programavimo kalbų informaciją galima slėpti tik vienoje klasėje. Bendrų „paslapčių“ kelios klasės negali turėti. Tai nepatogu, nes dažnai kelios klasės turi sąveikauti ir sudaryti uždarą klasių grupę. Tokia klasių sąveika turi būti apsaugota nuo galimos išorinės neigiamos įtakos, atsirandančios keičiant ar tobulinant programas. Šios klasės turi turėti bendrą sąsają. Taigi programavimo kalboje turėtų būti galimybė slėpti informaciją keliose klasėse. Tam komponentinio programavimo kalbose yra „modulio“ arba „pakuotės“ konstrukcijos.

Komponentinio programavimo kalbos yra realizuojamos komponentinio programavimo sistemose. Svarbi tokių sistemų savybė — naujų komponentų dinaminis įkėlimas. Šis bruožas garantuoja komponento įkėlimą į atmintį tik tada, kai jo prireikia, pavyzdžiui, kai jį importuoja vienas iš jau įkeltų komponentų.

1.2. Objektas ir komponentas

„Objektas“ ir „komponentas“ kartais vartojami kaip sinonimai, tačiau iš esmės tai yra skirtingos sąvokos.

Komponentas yra išbaigta „detalė“, kuri gali būti įdėta į programą. Jis turi aiškiai apibrėžtą sąsają su išore. Tuo komponentas iš išorės panašus į objektą. Tačiau jo vidinė struktūra gali būti įvairi.

Komponentas dažniausiai turi kelias klases ir objektus. Be to, komponentu gali būti ne tik klasės. Jame net iš viso gali klasių nebūti. Komponentas gali būti sudarytas iš tradicinių procedūrų, gali turėti globaliųjų, statinių kintamųjų, net gali būti sukurtas naudojantis struktūrinio programavimo ar kitais principais.

Panašiai kaip klasės viena su kita gali būti susietos paveldėjimo ryšiais, komponentai siejami importo ryšiais.

2. Komponentinis Paskalis

Komponentinis Paskalis — pakankamai „jauna“ (sukurta 1997 m.) programavimo kalba, kilusi iš Paskalio, Modulos-2 ir Oberono kalbų [1]. Jos kūrėjai — N. Virto (*Niklaus Wirth*) mokyklos atstovai. Pats kalbos pavadinimas „Komponentinis Paskalis“ puikiai nusako kalbos paskirtį ir kilmę.

Pagrindinė Komponentinio Paskalio paskirtis — kurti programų komponentus ir sudėtingą komponentinio programavimo principais paremtą programinę įrangą. Tai gana galinga programavimo kalba, nors dėl santykinio paprastumo ir artimumo mokyklose ir universitetuose dėstomai Paskalio kalbai Komponentinis Paskalis tinka programavimo mokymui.

Pagrindinis Komponentinio Paskalio privalumas — saugumas (tai būdinga kiekvienai komponentinio programavimo kalbai).

Kitas svarbus kalbos bruožas — paveldėjimo valdymas norint jį padaryti ne tokį pavojingą. Klasės idėja Komponentiname Paskalyje realizuojama siejant procedūras (metodus) su įrašo tipu. Kuriant klasę, nurodomi atributai, nusakantys, kokius paveldėjimo veiksmus galima atlikti su tuo įrašu (metodu): ar įrašas bus išplečiamasis (galima plėsti), abstraktusis (naudojamas kaip kitų įrašų bazinis tipas), ar jis bus galutinis (negalima plėsti), arba ribojamas (tokio įrašo tipo kintamajam atmintis gali būti skiriama tik tame modulyje, kuriame įrašas yra aprašytas). Naujai aprašyti metodai turi naujumą parodantį požymį. Visa tai padeda nustatyti įrašo ir jo plėtinio nesuderinamumus. Be to, Komponentiniame Paskalyje nėra sudėtinio, t. y. iš kelių objektų, paveldėjimo, kuris dažnai kelia problemų.

Be šių originalių Komponentinio Paskalio bruožų, norėtųsi paminėti kai kuriuos tradicinių konstrukcijų skirtumus nuo plačiai vartojamos Paskalio kalbos.

Raidės. Vardams galima naudoti visas kompiuteryje naudojamos koduotės abėcėlės raides, tarp jų ir visas lietuviškas raides su diakritiniais ženklais. Taigi yra daug didesnė galimybė parinkti prasminius vardus. Varduose didžiosios ir mažosios raidės yra skirtingos. Tai dar labiau padidina vardų pasirinkimo laisvę. Tačiau mokantiems Paskalio kalbą ir įpratusiems dirbti su ja, iš pradžių bus neįprasta, kad visi baziniai žodžiai rašomi vien didžiosiomis raidėmis.

Masyvai. Panašiai kaip ir C kalboje, masyvai indeksuojami tik sveikaisiais skaičiais, o elementų numeracija visada prasideda nuo nulio. Yra galimybė apibrėžti atviruosius (be iš anksto nurodyto ilgio) vienmačius ir daugiamačius masyvus. Vietoje simbolių eilučių tipo naudojami simbolių masyvai, kurie baigiasi nuliniu simboliu „0X“ (čia taip pat panašu į C kalbą).

Procedūros ir funkcijos. Ir funkcijos, ir procedūros aprašomos naudojantis baziniu žodžiu PROCEDURE. Net tada, kai nėra formaliųjų parametrų, paliekamas tuščias parametrų sąrašas, pavyzdžiui, PROCEDURE *Brėžti* (). Procedūros-funkcijos gražinama reikšmė nepriskiriama funkcijos vardui, bet grąžinama naudojant sakinį RETURN.

Procedūros pabaigoje (po žodžio END) pakartojamas procedūros vardas.

Aibės ir vardinis tipai. Yra tik vienas tipas SET, apibrėžiantis aibę, kurią gali sudaryti sveikieji skaičiai nuo 0 iki 31. Vardinio tipo nėra.

Įrašai. Jau minėjome, kad kuriant klases metodai siejami su įrašo tipu, nurodant specialius atributus. Todėl įrašai yra labai svarbūs programoms rašyti remiantis objektinio programavimo principais.

Įrašo (taip pat ir rodyklės) tipą galima plėsti. Tipas, kurį plečiame, vadinamas baziniu. Išplėstas tipas susideda iš bazinio tipo laukų

bei metodų ir turi dar naujų. Plečiant įrašo tipus Komponentiniame Paskalyje realizuojamas svarbus objektinio programavimo bruožas — paveldėjimas.

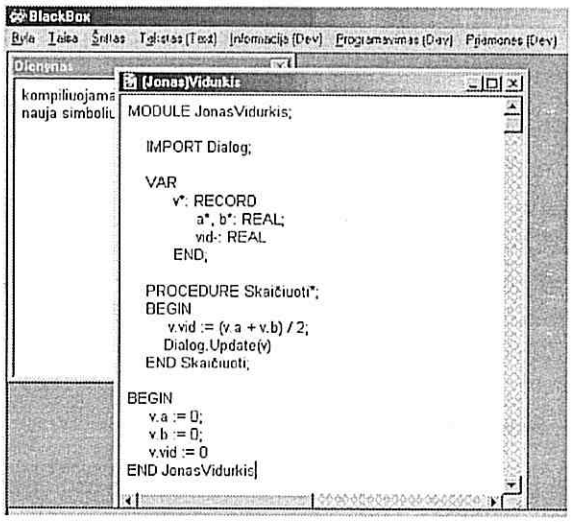
Variantinių įrašų Komponentiniame Paskalyje nėra.

Įvedimas ir išvedimas. Skirtingai nuo Paskalio kalbos, nėra specialių integruotų į kalbą duomenų įvedimo ir išvedimo procedūrų, taip pat ir bylų tipo. Sistemoje „BlackBox", kurioje programuojama Komponentiniu Paskaliu, paprastai šie veiksmai atliekami dialogo languose naudojant įvairius į juos dedamus valdiklius (pavyzdžiui, teksto laukas tinka tiek įvedimui, tiek išvedimui, o pavadinimai (angl. *Caption*) — tik duomenų išvedimui). Galima įvedimui ir išvedimui naudoti pagalbinį sistemos langą (dienyną), tačiau tam reikia pasinaudoti atitinkamais sistemoje esančiais moduliais (juos importuoti).

Programuojant Komponentiniu Paskaliu, iš esmės keičiasi visas programavimo stilius, palyginti su procedūriniu programavimu. Programos atskiros dalys (panašiai kaip procedūros procedūriniame programavime) šiuo atveju yra moduliai. Tik, skirtingai nuo procedūrų, jie kompiliuojami atskirai. Taigi programa gali būti sudaryta iš daugybės tarpusavyje sąveikaujančių dalių. Modulių sąveikos principai aprašomi kitame skyrelyje.

3. Komponentinio programavimo sistema „BlackBox"

Komponentinis Paskalis yra realizuotas sistemoje „BlackBox" (jos pagrindinį langą matome 1 pav.), kuri palengvina programų komponentų, įskaitant tuos, kurie naudoja grafinę sąsają su vartotoju, kūrimą. „BlackBox" tenkina svarbiausius komponentinio programavimo sistemų reikalavimus: turi automatinio šiukšlių surinkimo mechanizmą, dinaminį komponentų įkėlimą, veikia su komponentinio programavimo kalba. Visa „BlackBox" sistema yra parašyta Komponentinio Paskalio kalba, remiantis komponentinio programavimo principais. Todėl besimokančiajam programavimo pati gali tarnauti pavyzdžiu.



1 pav. Sistemos „BlackBox" pagrindinis langas

Į sistemą yra integruotas tekstų redaktorius, grafinis dialogo langų redaktorius, Komponentinio Paskalio kompiliatorius. Yra galimybė jungtis prie duomenų bazių. Kiekviena šių dalių vadimama posisteme (komponentu). Kiekviena posistemė turi sąveikaujančių modulių rinkinį. *Modulis* — tai autonominė, atskirai kompiliuojama programos dalis, kurią sudaro duomenys, procedūros ir klasės. Modulis — tai mažiausias Komponentinio Paskalio komponentas.

Modulyje esantys tipai, konstantos, kintamieji, procedūros gali būti eksportuojami, t. y. matomi išorėje ir prieinami kitiems moduliams. Tokio eksporto pagalba moduliai gali sąveikauti.

Kuriant naują modulį galima pasinaudoti jau esamais, t. y. importuoti reikiamus modulius. Šiuo atveju kuriamas modulis gali būti traktuojamas kaip *klientas*, o importuojamas modulis — kaip *serveris* [5]. Klientas gali nežinoti importuojamo modulio struktūros, jo pradinio teksto. Pakanka žinoti tik serverio eksportuojamus objektus (tipus, konstantas, kintamuosius, procedūras).

Modulio eksportuojamų objektų sąrašas vadinamas to modulio *sąsaja*. Sąsaja — viena svarbiausių komponentinio programavimo sąvokų. Kliento ir serverio sąveika vyksta per

serverio sąsają (2 pav.). Ji apibrėžia taisykles, kurių turi laikytis klientas, norėdamas pasinaudoti importuojamu moduliu.

2 pav. Kliento ir serverio sąveika

Visi neeksportuojami objektai yra paslėpti modulio viduje, t. y. nematomi išorėje. Tai labai svarbu saugiai keičiant bei tobulinant programas.

Koncepcija laikyti serverį „juoda dėže“, kurioje paslėptos modulio detalės, padiktavo sistemos „BlackBox“ pavadinimą (angl. *black box* — juoda dėžė).

4. „BlackBox“ mokykloje

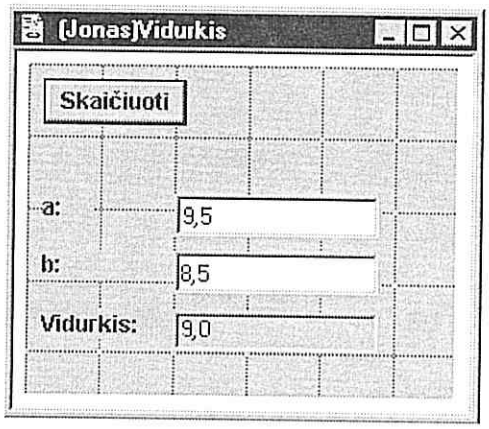
Kadangi sistemos pagrindas yra Komponentinio Paskalio kalba, kuri yra Paskalio kalbos „anūkė“ ir kurios sintaksė yra gana paprasta, tvarkinga, tai ji puikiai tinka mokyti programavimo sklandžiai į jį įjungiant modernius programavimo metodus — objektinį ir komponentinį programavimą.

Ypač svarbi besimokančiųjų programavimo kultūra. „BlackBox“ sistemoje yra keletas papildomų galimybių vaizdžiai ir kultūringai pateikti programų tekstus: naudoti įvairius šriftus (ne vien tik lygiapločius, kaip buvo įprasta programavime iki šiol), įvairius šriftų dydžius, spalvas, į programos komentarus galima įterpti ne tik tekstą, bet ir kitus vaizdinius objektus (rodinius) — paveikslus, mygtukus ir pan. Sistemos „BlackBox“ žinyne patariama komentarus rašyti kursyvu, eksportuojamų objektų vardus — pusjuodžiu šriftu, keičiamas ar naujas programos dalis žymėti kita spalva. Yra patarimų, kaip parinkti vardus programos konstantoms, tipams, kintamiesiems, procedūroms.

Programuojant skatinama naudoti pradinėmis ir galutinėmis sąlygomis. Pradinės sąlygos apibrėžia, ką turi tenkinti pradiniai duomenys (pvz., prieš atliekant sakinių seką), galutinės — ką turi tenkinti rezultatai. Sąlygos, išreikštos loginiu reiškiniu *s*, rašomos į Komponentinio Paskalio integruotą procedūrą ASSERT(s). Vienas kreipinys į šią procedūrą rašomas prieš tikrinamą sakinių seką (su pradinėmis sąlygomis, kitas — po jo (su galutinėmis sąlygomis). Procedūra nutraukia programos vykdymą, kai loginis reiškinys *s* įgyja reikšmę FALSE. Vadinasi, programoje yra klaida (sakinių sekai pateikiami neteisingi pradiniai duomenys arba sakinių seka duoda neteisingą rezultatą). Toks programavimo stilius padeda išvengti klaidų ir teisingai naudotis importuojamų modulių objektais.

Parašius programą svarbu ją tinkamai dokumentuoti. Tai numatyta sistemoje ir sudarytos puikios sąlygos mokytis pateikti dokumentacijas: žinyne yra dokumentavimo taisyklės, kurios naudojamos dokumentuojant visus integruotus į „BlackBox“ modulius. Taisyklėse pabrėžiama pradinių ir galutinių procedūrų sąlygų aprašymo svarba. Programų dokumentacijoms saugoti skirti specialūs katalogai *Docu*.

Beveik kiekvienas besimokantis programuoti nori kuo greičiau išmokti kurti programas, turinčias grafinę sąsają su vartotoju: meniu, dialogo langus (3 pav.).



3 pav. Projektavimo būsenos dialogo langas, skirtas dviejų skaičių vidurkiui skaičiuoti. Vidurkio skaičiavimo modulio tekstą matome 1 paveiksle.

Sistemoje „BlackBox“ žmogus jau turintis pradinių Komponentinio Paskalio žinių gali kurti dialogo langus. Aišku, visa tai neturėtų išstumti tradicinio algoritmavimo mokymo būdo. Todėl komponentinio programavimo ir programų su grafine sąsaja kūrimo galėtų būti mokoma po pagrindinio programavimo kurso. Kitas variantas — tas pačias lavinančias algoritminę mąstyseną programas rašyti taikant modernius objektinio bei komponentinio programavimo principus ir pateikiant kitu pavidalu, pavyzdžiui, įvedant ir išvedant duomenis dialogo languose.

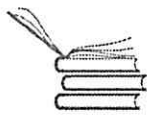
Jau minėjome, kad mokantis komponentinio programavimo pati sistema „BlackBox“ gali tarnauti pavyzdžiu. Taip yra dėl to, kad ji turi gerą žinyną ir yra atvira: prieinamos posistemių išteklių bylos, kurias galima keisti ir tuo pačiu eksperimentuoti, taip pat pridėti savo sukurtų komponentų.

Sistemos „BlackBox“ angliškosios versijos interneto svetainė

http://www.oberon.ch/.

Ji platinama nemokamai. Yra variantai „Windows“ ir „Mac OS“ operacinėms sistemoms. Šiuo metu „BlackBox“ yra verčiama į lietuvių kalbą. Jos lietuviškus priedus galima atsisiųsti iš

ftp://aldona.mii.lt/pub/MII/PMS/progr/BlackBox.



1. Komponentinio Paskalio oficialus aprašas. Informatika. 2000, **35**(1), p. 68–101.
2. Mössenböck H. Object-Oriented Programming in Oberon-2. Springer, 1995.
3. Oberon microsystems. Sistemos „BlackBox“ dokumentacija. 2000 (http://www.oberon.ch/).
4. C. Szyperski. Component Software. ACM Press, 1998.
5. J. S. Warford. The BlackBox Framework. 2001 (ftp://pepvax.pepperdine.edu/pub/compsci/prog-bbox/).