Liet. matem. rink., T. 41, spec. nr., 278–283

Komponentinio programavimo taikymas informatikos mokyme

Gintautas GRIGAS, Tatjana JEVSIKOVA (MII)
el. paštas: grigas@ktl.mii.lt, tatjanaj@ktl.mii.lt

1. Įvadas

Programinės įrangos produktai tampa vis sudėtingesni. Jeigu programa yra monolitinė, ją nėra paprasta ne tik kurti, bet ir tobulinti ateityje, tenkinant nuolat augančius vartotojų poreikius. Ji greitai sensta, kartais tampa pasenusi net tada, kai dar nėra sukurta iki galo. Todėl racionaliau programą 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).

Komponentinis programavimas apima programų komponentų, iš kurių būtų galima surinkti norimų galimybių programas, kūrimą ir komponavimą. Ši idėja tampa vis populiaresnė: publikacijų skaičius komponentinio programavimo tema auga eksponentiškai [4]. Tai nėra mada ar eilinė populiari idėja, kuri greitai išnyktų. Pagal C. Szyperskį [4], komponentų naudojimas yra „gamtos dėsnis“, kuriam paklūsta visos subrendusios technikos disciplinos.

Kuriant programų komponentus bei juos komponuojant atsiveria galimybė surinkti programas pagal jų naudotojų 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 programos dalį, galima paimti iš jau esamų aibės. Toks būdas yra geresnis negu viską programuoti nuo nulio, nes užima mažiau laiko, reikalauja mažiau pastangų ir taip sudaro sąlygas greičiau tenkinti naujus naudotojų poreikius.

Komponentinis programavimas mokykloje yra patrauklus dėl dviejų priežasčių: 1) tai nauja ir perspektyvi programavimo kryptis ir 2) mokiniai galėtų sukurti praktiniam panaudojimui tinkamų produktų, nes atskirą komponentą sukurti lengviau, negu net ir nedidelę išbaigtą monolitinę programą. Tačiau komponentinio programavimo panaudojimas informatikos pamokose nebuvo nagrinėtas. Matyt taip yra dėl to, kad ši programavimo kryptis yra dar palyginti nauja, o pramoninės komponentinio programavimo sistemos per daug sudėtingos, kad jas būtų galima nagrinėti pamokose. Šią problemą galėtų padėti išspręsti nesenai sukurta Komponentinio Paskalio [1] programavimo kalba ir ją naudojanti komponentinio programavimo sistema „BlackBox“ [3]. Tiek kalba, tiek sistema pasižymi paprastumu. Dėl to atsiranda palankesnės sąlygos su šia nauja kryptimi supažindinti programavimu besidominčius mokinius.

2. Komponentinio programavimo sistemos „BlackBox" panaudojimas mokyme

Kurti programų komponentus galima naudojant įvairias programavimo kalbas ir įvairias programavimo terpes. Tinka objektinio programavimo kalbos, tačiau yra specialiai šiam tikslui skirtų komponentinio programavimo kalbų, kurių naudojimas apsaugo nuo tam tikrų klaidų dirbant su atmintimi. Be to, nei mokymui naudojama programavimo sistema, nei programavimo kalbos sintaksė neturi baiminti sudėtingumu. Gerai, kai net ir labai paprastus programų komponentus moksleiviai suprogramuoja patys ir pajunta tų komponentų sąveiką.

Komponentinio programavimo sistema „BlackBox" tinka komponentiniam programavimui mokyti dėl kelių priežasčių.

1. Sistemos pagrindas – moderni komponentinio programavimo kalba Komponentinis Paskalis. Kalbos sintaksė nėra sudėtinga, be to, ji yra artima Paskalio kalbai, naudojamai mokant programavimo mokyklose. Tačiau nepaisant sintaksės paprastumo, Komponentinis Paskalis yra pakankamai galinga kalba, kad būtų galima kurti ir sudėtingus programų komponentus.
2. Mokyme yra svarbi besimokančiųjų programavimo kultūra. „BlackBox" sistemoje yra 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 įterpti ne tik tekstą, bet ir kitus vaizdinius objektus (rodinius) – paveikslus, mygtukus ir pan. Sistemos „BlackBox" žinyne [3] pateikti išsamūs patarimai, kaip kultūringai rašyti programas panaudojant šias papildomas galimybes ir parašytą programą tinkamai dokumentuoti. Programų dokumentacijoms saugoti skirti specialūs katalogai *Docu*. Žinyne esančiuose patarimuose kaip rašyti programas ir jas dokumentuoti skatinama naudoti pradines ir galutines sąlygas, kurios apibrėžia, ką turi tenkinti pradiniai duomenys (pvz., prieš atliekant sakinių seką) ir ką turi tenkinti rezultatai. Komponentiniame Paskalyje yra patogi integruota procedūra pradinių ir galutinių sąlygų tikrinimui.
3. Sistemoje „BlackBox" yra galimybė kurti turinčias grafinę sąsają su vartotoju programas (programų komponentus), kas atspindi daugelio besimokančiųjų norus. Žmogus jau su pradinėmis Komponentinio Paskalio žiniomis gali kurti dialogo langus, naudoti valdiklius, įvairius duomenų įvedimo ir išvedimo būdus.
4. „BlackBox" sistema parašyta Komponentinio Paskalio kalba ir yra atvira. Nagrinėjant sistemos vidinę struktūrą (pavyzdžiui, ryšius tarp integruotų į sistemą modulių, jų sąsajas, meniu komandas, modulių dokumentacijas) ji pati gali būti puikus komponentinio programavimo pavyzdys.
5. Sistemos „BlackBox" mokomasis variantas platinamas nemokamai.

3. Modulio temos

Modulio turinį suskirstysime į temas, o temas – į skyrelius. Pateiksime trumpus skyrelių medžiagos apibūdinimus.

3.1. *Įvadas į objektinį ir komponentinį programavimą*

Objektinio programavimo pagrindinės sąvokos. Pradėti reikėtų nuo objektinio programavimo, nes rašant programų komponentus, dažniausiai naudojamasi objektinio programavimo principais. Mokinius reikėtų supažindinti su klasės ir objekto sąvokomis [2], santykiu tarp jų, pagrindinėmis objektų savybėmis: paveldėjimu (galima išnagrinėti paprasčiausius paveldėjimo atvejus), informacijos slėpimu, polimorfizmu, iliustravimui naudojant Komponentinį Paskalį.

Objektas ir komponentas. Taip, kaip objektiniame programavime operuojama objektais, komponentiniame programavime operuojama komponentais. Sąvokos „objektas" ir „komponentas" dažnai painiojamos, todėl besimokantiesiems iš karto reikėtų akcentuoti skirtumą tarp jų. 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, komponente gali būti ne tik klasės, arba net iš viso gali klasių nebūti. Jis gali būti sudarytas iš tradicinių procedūrų, gali turėti globaliųjų, statinių kintamųjų, gali būti sukurtas netgi nesinaudojant struktūrinio ar objektinio programavimo principais.

Šios dalies tikslas – gerai suvokti pagrindines sąvokas. Reikėtų nevengti prie jų grįžti ir nagrinėjant tolesnes temas.

3.2. *Komponentinio Paskalio ir Paskalio pagrindiniai skirtumai*

Nors šios dvi kalbos yra giminingos, tačiau turi skirtumų. Yra prasmė juos panagrinėti, jei besimokantieji jau buvo dirbę su Paskalio kalba. Lyginant naujos kalbos sintaksę su jau žinoma, nereikia iš naujo mokyti pagrindinių konstrukcijų, o tik akcentuoti skirtumus. Tokiu būdu daugiau dėmesio galima skirti specifiniams kalbos bruožams.

3.3. *Moduliai Komponentiniame Paskalyje*

Modulis – mažiausias komponentas Komponentiniame Paskalyje. Programuojant Komponentiniu Paskaliu iš esmės keičiasi visas programavimo stilius, lyginant 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.

Modulio sąsaja. 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. Ji apibrėžia

taisykles, kurių turi laikytis klientas, norėdamas korektiškai pasinaudoti importuojamu moduliu. Panašiai kaip klasės gali būti susietos viena su kita paveldėjimo ryšiais, komponentai siejami importo ryšiais.

Komandos ir patikslinti vardai. Kad būtų galima kreiptis į modulio eksportuojamus objektus jo išorėje, naudojami patikslinti vardai (kai kreipiamasi į tipus, kintamuosius, konstantas) arba komandos (kai kreipiamasi į eksportuojamas procedūras). Panašiai, kaip objektiniame programavime kreipiantis į objektų duomenis ar metodus, naudojamas taško operatorius.

3.4. *„BlackBox" sistemoje sukurtų darbų saugojimas diske*

Kiekviena programavimo sistema turi savitą programų (projektų) saugojimo būdą. Gerai, kai besimokantieji iš karto, nuo pat pirmųjų savo darbų kūrimo, pripranta juos tvarkingai saugoti diske. Pagrindinis sistemos katalogas vadinasi *BlackBox*. Šiame kataloge kiekvienas mokinys susikuria savo katalogą, o jame – dar penkis pakatalogius: parašytų modulių tekstams saugoti (*Mod*), modulių dokumentacijoms saugoti (*Docu*), modulių ištekliams (dialogo langai, eilutės ir pan.) saugoti (*Rsrc*), modulių simbolių ir kodo byloms saugoti (atitinkamai katalogai *Sym* ir *Code*).

Simbolių ir kodo bylos sukuriamos kompiliuojant Komponentiniu Paskaliu parašytą modulį. Svarbu iš karto atkreipti dėmesį į tai, kam reikalingos šios dvi bylos: simbolių byla – tai sukompiliuoto modulio sąsaja, kuri generuojama kompiliatoriumi ir naudojama kompiliavimo metu tikrinti, ar korektiškai panaudojami importuojamų modulių objektai; kodo byla – kompiliatoriaus generuota mašininio kodo byla, kuri iškviečiama modulio vykdymo metu. Tam, kad galima būtų vykdyti modulį, nebūtina turėti jo pradinį tekstą, pakanka turėti jo kodo bylą. Jeigu katalogai *Sym* ir *Code* nebuvo sukurti, jie bus sukuriami automatiškai.

Katalogo *Docu* turėjimas ugdo gerą įprotį dokumentuoti kiekvieną parašytą modulį. Tegu pradžioje, kol moduliai labai paprasti, tai būna trumpas programos veiksmų aprašymas su modulio iškvietimo komanda. Vėliau jame bus pateikiamos kiekvieno modulio dokumentacijos pagal žinyne pateiktas taisykles.

3.5. *Posistemės*

Sistema „BlackBox" parašyta remiantis komponentinio programavimo principais ir todėl ji pati susideda iš komponentų (posistemių). Kiekviena posistemė turi sąveikaujančių modulių rinkinį ir diske pateikiama pagrindinio katalogo *BlackBox* pakatalogiu. Pavyzdžiui, posistemė *Text* skirta darbui su tekstu, *Form* – dialogo langų projektavimui, *Dev* – programavimui. Kiekvieno besimokančiojo sukurtas katalogas su parašytais darbais taip pat gali būti traktuojamas kaip posistemė ir yra atvaizduojamas sistemos saugykloje kartu su kitų posistemių katalogais.

Reikia atkreipti dėmesį į modulių vardų sudarymo taisykles, kurias sąlygoja posistemės pavadinimas. Modulio vardo prefiksas turi būti posistemės vardas, o likusi dalis – modulio vardas. Įrašant modulio bylą į *Mod* katalogą, jai suteikiamas vardas, sutampantis

su modulio vardu, tik be posistemės prefikso. Šitokios taisyklės naudojamos visoje sistemoje „BlackBox", todėl jų laikytis yra naudinga dar ir dėl to, kad lengviau būtų galima susigaudyti nagrinėjant integruotų modulių sąsajas ir dokumentacijas.

3.6. *Dialogo langai*

Dialogo langų sukūrimo būdai. Yra du būdai kurti dialogo langus, su kuriais reikėtų supažindinti besimokančiuosius. Paprastų modulių atvejais (tokį būdą galima naudoti pradžioje) rašomas ir kompiliuojamas modulis, o kuriant naują dialogo langą nurodomas susiejimas su tuo moduliu. Kitas būdas – dialogo lango maketą ir modulį kurti atskirai, vėliau nurodant valdiklių susiejimą su modulio objektais.

Valdikliai. Pradedant kurti grafinę sąsają, lygiagrečiai reikėtų supažindinti besimokančiuosius su pagrindiniais valdikliais, dedamais į dialogo langus (pvz., teksto laukas, mygtukas, pavadinimas, žymimasis langelis, sąrašas). Kiekvienas iš valdiklių gali būti susietas tik su tam tikro tipo objektu iš modulio. Naudojant dialogo langus ir valdiklius keičiasi duomenų įvedimo ir išvedimo galimybės, lyginant su Paskaliu. Įvedimas ir išvedimas vyksta interaktyviai, jam naudojami valdikliai, atitinkantys įvedamų arba išvedamų duomenų tipus.

Dialogo langų būsenos. Dialogo langai kuriami ir įrašomi į diską maketavimo būsenoje. Tam, kad būtų galima pasinaudoti sukurtu dialogo langu, jis turi būti atvertas bandymų būsenoje. Tą galima padaryti įvairiais būdais, bet rezultatas tas pats – bus įvykdyta ta pati komanda (vieno iš integruotų modulių procedūra). Besimokančiuosius reikėtų supažindinti su įvairiais dialogo langų iškvietimo būdais.

Saugai. Kuriant grafinę sąsają su vartotoju, svarbu, kad ji būtų draugiška, t.y., valdikliai arba meniu punktai, kurie šiuo metu (priklausomai nuo konteksto) neduos jokių rezultatų, turėtų būti neveiklūs. Tam sistemoje „BlackBox" rašomos specialios procedūros, vadinamos „saugais" ir siejamos su atitinkamais dialogo lango valdikliais arba meniu elementais.

3.7. *Programavimo kultūra*

Svarbu, kad besimokantysis išmoktų ne tik programavimo, bet ir programavimo kultūros. Tai aktualu ne tik nagrinėjant kitų žmonių parašytas programas, bet ir tobulinant savo parašytą programą (ypač jei ji buvo parašyta seniai).

Programos teksto išdėstymas. Išsamūs patarimai, kaip rašyti programas, pateikti sistemos žinyne. Programose galima naudoti įvairius šriftus ir kitokias išraiškos priemones. Tačiau visa tai reikia naudoti prasmingai. Pavyzdžiui, patariama eksportuojamus modulio objektus rašyti pusjuodžiu šriftu, tekstinius komentarus – kursyvu, naujas arba šiuo metu keičiamas programos vietas – kita spalva. Parenkant vardus modulio procedūroms, tipams, kintamiesiems, konstantoms, reikia nepamiršti, kad jie būtų mnemoniški ir vardų sistema būtų vieninga ir prasminga. Sistemoje tai yra numatyta – varduose galima vartoti visas kompiuteryje naudojamos kodūotės raides (tarp jų ir lietuviškas).

Pradinių ir galutinių sąlygų tikrinimas. Išvengti klaidų arba jas nustatyti padeda pradinių ir galutinių sąlygų tikrinimas. 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. Tai reiškia, kad programoje yra klaida (sakinių sekai pateikiami neteisingi pradiniai duomenys arba sakinių seka duoda neteisingą rezultatą). Toks programavimo stilius padeda išvengti klaidų ir korektiškai naudotis importuojamų modulių objektais.

Programų dokumentavimas. Programavimo kultūros svarbus elementas – tinkamai programą dokumentuoti. „BlackBox" sistemoje sutarytos geros sąlygos dokumentuoti programas. Su katalogais *Docu* besimokantieji jau yra susipažinę. Dokumentuojant modulius pagal taisykles, kurių laikomasi sistemoje „BlackBox", dokumentacijoje pateikiama modulio sąsaja, modulio ir kiekvieno jo objekto paskirties trumpas aprašymas, nurodant pradines ir galutines sąlygas. Dokumentavimo taisyklės pateiktos „BlackBox" žinyne.

4. Išvados

Pasinaudojus paprasta ir dalinai atvira komponentinio programavimo sistema „BlackBox" ir į ją integruotu Komponentinio Paskalio transliatoriumi, atsiranda reali galimybė į informatikos kursą įtraukti išplėstinį arba tikslinį komponentinio programavimo modulį, kurio programos metmenis pateikiame šiame straipsnyje.

Literatūra

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

Component programming at school

G. Grigas, T. Jevsikova

The issues of component programming teaching at school are discussed. The main features and advantages of system "BlackBox" for teaching of modern programming topics are shortly described. The outline of curriculum based on this system is presented. It includes such topics as introduction into object and component programming, the main differences between Pascal and Component Pascal, modules, projects and management of their data on disk, subsystems of "BlackBox", dialog boxes, programming and documentation style. The module may be recommended for advanced school students interested in modern programming.