PROGRAMAVIMAS IR PROGRAMINĖĮRANGA

Generavimo metodų naudojimas kuriant .NET komponentines programų sistemas

Vaidas Giedrimas

Šiaulių universiteto lektorius
Lecturer of Šiauliai University
Višinskio g. 25, LT-76351 Šiauliai
Tel. (+370 650) 72042
El. paštas: vaigie@mi.su.lt

Komponentinė paradigma leidžia greičiau sukurti programų sistemas, efektyviau pakartotinai naudoti anksčiau sukurtus programinius artefaktus – komponentus. Deja, praktiškai kuriant komponentines programų sistemas daug laiko sugaištama komponentų paieškai ir verifikavimui. Vienas galimų šios problemos sprendimų – komponentinių programų verifikavimui ir sintezei naudoti formaliuosius metodus. Komponentinių (ypač iš pasaulinio tinklo paslaugų sudarytų) programų kūrimas formaliaisiais metodais plačiai diskutuojamas, tačiau kol kas yra tik pradinės stadijos. Straipsnyje supažindinama su sintezės sistema, skirta komponentinėms programoms iš .NET komponentų kurti. Sistema realizuota naudojant du generavimo metodus: struktūrinę sintezę ir indukciją. Pateiktas konkretus programų sistemos generavimo pavyzdys.

Komponentinė paradigma yra viena iš dominuojančių. Ypač daug diskutuojama dėl vieno komponentų tipo – pasaulinio tinklo paslaugų (angl. *web-services*). Komponentinė paradigma leidžia greičiau sukurti programų sistemas efektyviau pakartotinai naudojant gatavus programinius artefaktus – komponentus. Be to, komponentinės programos yra ypač lengvai testuojamos ir modifikuojamos. Deja, praktiškai kuriant komponentines programų sistemas daug laiko sugaištama komponentų paieškai ir verifikavimui (Crnkovic, Larsson, 2003).

Vienas galimų šios problemos sprendimų – komponentinių programų specifikavimui, sintezei ir verifikavimui taikyti formaliuosius metodus. Juos naudojančių sintezės sistemų pranašumas – galimybė kurti „teisingas“, specifikaciją atitinkančias programas be klaidų. Be to, tokios sistemos leidžia kurti programas „iš viršaus žemyn“, specifikuojant jas deklaratyviuoju, o ne imperatyviuoju būdu (Lau, 2003). Formaliųjų metodų nauda kuriant komponentines programų sistemas nagrinėjama Lau (2003), Lupeikienės, Giedrimo (2005), Chen (2006), Liu, Jifeng (2006) darbuose, tačiau pasigendama konkrečių siūlomų programų kūrimo metodikų realizacijų.

Straipsnio tikslas – pateikti per pastaruosius ketverius metus sukurtą automatizuoto programų sistemų surinkimo iš komponentų metodą ir jį realizuojančią programų sintezės sistemą.

.NET komponento modelis

Ši programų sintezės sistema generuoja Microsoft .NET Framework platformai skirtas

komponentines programas, naudodama gatavus .NET komponentus.

.NET yra plačiai naudojama, sparčiausiai besivystanti komponentinė technologija, įgalinanti kurti didelės spartos monolitines ir paskirstytąsias programų sistemas (Lowy, 2005). Pamatine komponentine technologija programų sistemų generavimo eksperimentams atlikti .NET pasirinkta dėl šių priežasčių:

- .NET technologija yra nepriklausoma nuo platformos (dėl MONO ir DotGNU projektų, įgalinančių .NET programas veikti ir kitose nei *Windows* operacinėse sistemose) ir nuo programavimo kalbos (galima naudoti bet kurią, CLI atitinkančią programavimo kalbą);
- galingi .NET refleksijos įrankiai leidžia inspektuoti bet kurį binarinį (*.exe, *.dll ir pan.) komponentą, dinamiškai kviesti jo operacijas, dinamiškai kurti ir keisti komponentus programos vykdymo metu (angl. *Run-time*);
- galimybė .NET komponentų ir jų sudedamųjų dalių (klasių, jų metodų ir kt.) savybes aprašyti vartotojo atributais (angl. *custom attributes*) leidžia kurti programas atsižvelgiant ir į būsimos programų sistemos nefunkcinius reikalavimus. Be to, vartotojo atributai sudaro prielaidas patikrinti, ar komponentai yra suderinami ne tik sintaksiškai, bet ir semantiškai;
- vienalaikio kelių to paties komponento versijų naudojimo galimybė (angl. *side-by-side execution*).

Formalus .NET komponento modelis sukurtas atsižvelgiant į aprašytas (Crnkovic, Larsson, 2003; Lupeikienė, Giedrimas, 2005; Giedrimas, 2005; Chen, 2006; Liu, Jifeng, 2006) komponentų ir komponavimo proceso ypatybes.

Generavimo metodai

Pristatomoji sintezės sistema sukurta naudojant du generavimo metodus: struktūrinę programų sintezę ir indukciją.

Struktūrinės programų sintezės (SSP) metodo esmė – konstruoti programas iš atskirų modulių („juodųjų dėžių“), atsižvelgiant tik į jų įeities (angl. *input*) bei išeities (angl. *output*) taškus ir visiškai neatsižvelgiant į jų realizavimo detales. Kiekvieną tokį modulį atitinka viena intuicionistinio teiginių skaičiavimo (ITS) aksioma (Matskin, Tyugu, 2001). Programos sintezė vyksta tokiais etapais:

- Pateikiama uždavinio specifikacija;
- Ši specifikacija transformuojama į ITS teoremą;
- Naudojantis žinių baze, kuriai priklauso ir anksčiau minėtos aksiomos, ieškoma konstruktyvaus teoremos įrodymo;
- Jei įrodymas rastas, jis apibrėžia programos struktūrą, kuri, keičiant panaudotas aksiomas jas atitinkančiais moduliais, „užpildoma“ ir gaunama visa programa.

Kaip ir kiti dedukciniai metodai, SSP užtikrina, kad sintezės rezultatas visada yra teisingas (Matskin, Tyugu, 2001). SSP yra sėkmingai panaudotas PRIZ, NUT ir kitose sintezės sistemose, generuojančiose programas iš modulių ir objektų. Metodas komponentinėms programoms kurti dar nebuvo taikomas, tačiau teorinės tokių taikymų prielaidos egzistuoja (Giedrimas, Lupeikienė, 2005; Giedrimas, 2005).

Indukcija leidžia kurti hipotezes ir taip praplėsti uždavinio teoriją. Remiantis tais pačiais pavyzdžiais gali būti gaunami skirtingi nauji teiginiai. Teorines prielaidas naudoti induktyviojo loginio programavimo metodą komponentinėms programų sistemoms aprašė Martelli, Mascardi, Zini (1999), Lau (2003). Kaip parodyta (Giedrimas 2006b), induktyvusis metodas komponentinių programų sistemų sintezės sistemoje gali būti sėkmingai naudojamas kartu su struktūrinės sintezės metodu, kad būtų išspręstos neišsamių specifikacijų, neapibrėžtųjų komponentų ir sintezės rezultatų neatitikimo nefunkciniams reikalavimams problemos (Giedrimas, Lupeikienė, 2005). Tačiau, siekiant išvengti neteisingo rezultato, visos sintezės metu padarytos induktyviosios išvados privalo būti papildomai verifikuojamos.

Uždavinio specifikavimo kalba

Būsimojo programinio produkto specifikacijos sudarymas – pirmasis automatizuoto programavimo proceso etapas. Kiekvienoje struktūrinę sintezę naudojančioje sistemoje uždavinio specifikacijos buvo užrašomos skirtinga kalba. PRIZ sistemoje naudota UTOPIST kalba, NUT ir rNUT sitemose – NUT kalba. Šiuolaikinėse sistemose (pvz., COCOVILA projekte) pradedamos naudoti trečiosios kartos – vizualiosios – specifikavimo kalbos (Grigorenko, Saabas, Tyugu, 2005).

Šiame straipsnyje aprašomoje sintezės sistemoje naudojama XML pagrindu sukurta specifikavimo kalba, panaši į WSDL ir WSFL kalbas. Vizualioji kalba kol kas nenaudojama, tačiau, anot Erwig (2000), yra visos prielaidos ją sukurti. Detaliau komponentinių sistemų specifikavimo problemas aptaria Giedrimas ir Lupeikienė (2004).

Sintezės sistema

Sintezės sistemos įvestis yra *uždavinio specifikacija*, *uždavinio teorija*, *pakartotinai naudojami komponentai* ir pakartotinai naudojamos *programų architektūros*. Sistemos išvestis yra paruošta vykdyti *programa*. Programų sintezės sistemą sudaro šie elementai:

- *Transliatorius* kaip argumentus naudoja funkcinius reikalavimus iš uždavinio specifikacijos, išskaido juos į įvesties bei išvesties kintamuosius ir suformuoja uždavinio medį – *uždavinį vaizduojantį grafą*.
- *Planavimo modulis* – pagrindinė sistemos dalis. Jos argumentai yra trasliatoriaus suformuotas *uždavinio medis* ir *uždavinio teorija*. Modulio rezultatas yra konstruktyvusis uždavinio sprendinio teoremos *įrodymas*. Planavimo modulio vaidmenį sistemoje atlieka specializuotas automatinis *intuicionistinio teiginių skaičiavimo* teoremų įrodymo įrankis.
- *Pirmasis optimizavimo modulis* yra tipinis SSP metodą naudojančioms sistemoms. Jis atlieka tarpininko tarp planavimo modulio ir kodo generatoriaus vaidmenį. Šio modulio tikslas – pašalinti nereikalingus įrodymo žingsnius. Šie pertekliniai žingsniai gali būti generuoti planavimo moduliui vykdant prielaidomis grįstą paiešką pirmyn (angl. *assumption driven forward search*).
- *Programos kodo generavimo modulis*. Jei įrodymas rastas, programos generavimo modulis, naudodamas gautą konstruktyvųjį įrodymą, generuoja programą. Pažymėtina, kad čia sąvoka „programa“ vartojama platesne prasme. Komponentinės programos abstrakcijos lygmuo yra aukštesnis nei, pavyzdžiui, objektinės programos, todėl šiuo atveju sintezės rezultatas („programa“) yra architektūra, aprašyta pasirinkta architektūros aprašymo kalba (pvz., WSFL). Kita vertus, kad sintezės sistema generuotų žemesnio abstrakcijos lygmens programą, užrašytą kuria nors .NET programavimo kalba (pvz., C#), generatorius turi apimti dvi dalis: architektūros generatorių ir programos generatorių. Pirmasis, naudodamas teoremos įrodymą, generuoja programos architektūrą, ją išsaugo *architektūrų saugykloje* ir perduoda programos generatoriui. Pastarasis *komponentų saugykloje* suranda reikiamus komponentus ir, įstatydamas juos į architektūroje apibrėžtas vietas, generuoja programą.
- *Antrasis optimizavimo modulis* naudojamas siekiant atsižvelgti į nefunkcinius programų sistemos reikalavimus. Šio modulio argumentai yra architektūrų saugykloje išsaugota *programos architektūra* ir nefunkciniai reikalavimai, gauti iš *transliatoriaus*, rezultatas – programa, geriau atitinkanti nefunkcinius reikalavimus. Modulis, atsižvelgdamas į argumentus, iš *komponentų saugyklos* parenka tokius komponentus, kurie geriau tenkina minėtus reikalavimus nei jau panaudoti.

Išsamiau sintezės sistemos architektūra aprašoma darbuose (Giedrimas, 2006a; Giedrimas, 2006b) .

Sintezės proceso pavyzdys

Programų sintezės sistemos veikimo principą iliustruoja trumpas pavyzdys. Tarkime, *komponentų saugykloje* yra keturi .NET komponentai vaizdo medžiagai apdoroti. Sintezės sistema, naudodama .NET refleksijos mechanizmą, šiuos binarinius komponentus automatiškai inspektuoja, nustato, kokias sąsajas (ir su kokiomis operacijomis) jie realizuoja ir kokių reikalauja. Paskui sintezės sistema, remdamasi šiais duomenimis, sudaro komponentus aprašančias aksiomas ir taip suformuoja *uždavinio teoriją.* Aksiomos užrašomos *intuicionistinio teiginių skaičiavimo* logikos kalba (žr. lentelę).

Tarkime, sintezės sistemos naudotojas nori generuoti komponentinę programą, kuri nurodytą **.avi* formato vaizdo rinkmeną transliuotų internetu. Tam jis aukštesnio lygmens specifikavimo kalba specifikuoja uždavinį, kuris užrašant transformuojamas į tokią intuicionistinio teiginių skaičiavimo teoremą:

[rinkmena], AVI→ [transliacija]

Planavimo modulis, naudodamasis SSP taikomomis išvedimo taisyklėmis (Matskin, Tyugu, 2001), randa teoremos įrodymą:

$$\frac{AVI, AVI \rightarrow WAV}{AVI \rightarrow WAV};$$

$$\frac{AVI \rightarrow WAV, (AVI \rightarrow WAV) \rightarrow ([rinkmena] \rightarrow WAV)}{[rinkmena] \rightarrow WAV};$$

$$\frac{[rinkmena] \rightarrow WAV, WAV \rightarrow [transliacija]}{[rinkmena] \rightarrow [transliacija]}.$$

Kadangi teoremos įrodymas sėkmingas, aksiomų pakako, *aksiomų generavimo modulis* nenaudojamas. Taip pat šiuo atveju nenaudojamas ir *pirmasis optimizavimo modulis*. Jis būtų efektyvus tik tada, kai ieškant įrodymo būtų panaudota ir visiškai nereikalinga aksioma, pavyzdžiui, AVI → MPEG.

Programos kodo generavimo modulis, remdamasis gautu teoremos įrodymu, generuoja komponentinę programą, kurios tekstas čia nepateikiamas dėl vietos stokos.

Išvados

Eksperimentai su šiame straipsnyje aprašytu sintezės sistemos prototipu parodė, kad komponentinių programų sistemų kūrimo pro-

Lentelė. ***Komponentai programų sintezės eksperimentui***

Komponentas	Paskirtis	Aksiomos
Vaizdo medžiagos konvertavimo (VMK) komponentas	Komponento įeitis: vieno pasirinkto vaizdo formato rinkmena, išeitis – kito pasirinkto vaizdo formato rinkmena. Jei yra įdiegti papildomi kodekai, įrankis geba dirbti su šių formatų rinkmenomis: *.avi, *.mpeg, *.wav	(AVI → MPEG) →([rinkmena] → MPEG) (AVI → WAV) →([rinkmena] → WAV) (WAV → AVI) →([rinkmena] → AVI) (WAV → MPEG) →([rinkmena] → MPEG) (MPEG → WAV) →([rinkmena] → WAV) (MPEG → AVI) →([rinkmena] → AVI)
Kodekas K1	Komponentas realizuoja duomenų konvertavimo iš *.avi į *.wav formatą algoritmą	AVI →WAV
Kodekas K2	Komponentas realizuoja duomenų konvertavimo iš *.avi į *.mpeg formatą algoritmą	AVI →MPEG
Vaizdo medžiagos transliavimo internetu komponentas (VMT)	Komponento įeitis: *.wav formato vaizdo rinkmena, išeitis – transliacija internetu	WAV → [transliacija]

cesą galima iš dalies automatizuoti naudojant formaliuosius generavimo metodus.

Ateityje sistemą ketinama plėsti: pasiūlyti vizualiąją kalbą, kuria būtų galima pateikti uždavinių specifikacijas, sukurti specifikacijų tikslinimo modulį, realizuojantį induktyvųjį metodą.

LITERATŪRA

CHEN, X., JIFENG, H., LIU, Z., ZHAN, N. (2006). A Model of Component-Based Programming. UNU-IIST Report No. 350. Prieiga per internetą: http://www.iist.unu.edu/newrh/III/1/docs/techreports/report350.pdf , [žiūrėta 2007-01-12].

CRNKOVIC I., LARSSON M. (2003). *Building Realiable Component-Based Software Systems*. Eds. Crnkovic, I., Larsson, M. London: Artech House.

ERWIG, M. (2000). A Visual Language for XML. In *Proceedings of the 2000 IEEE international Symposium on Visual Languages (Vl'00)*. VL. IEEE Computer Society, Washington, DC, 47.

GIEDRIMAS, V. (2005). Component-based Software Generation: The Structural Synthesis Approach. In *Proceedings of 7th GPCE YRW 2005*. Tallinn.

GIEDRIMAS, V., LUPEIKIENĖ, A. (2004). Komponento specifikacijos formalizavimas *Liet. matem. rink.,* 44, spec. nr., p. 276–280.

GIEDRIMAS, V., LUPEIKIENĖ, A. (2005). Komponentinių programų struktūrinės sintezės teorinės problemos. *Liet. matem. rink.*, 45, spec. nr., p. 139–143. ISSN 0132-2818

GIEDRIMAS, V. (2006a). Architectures of Component-Based Structural Synthesis Systems. In *Databases and Information Systems: Seventh International Baltic Conference on Databases and Information Systems. Communications.* Vilnius: Technika, p. 331–315.

GIEDRIMAS, V. (2006b). Induktyvinis metodas komponentinių programų sintezėje. *Liet. matem. rink.,* 46, spec. nr., p. 103–106.

GRIGORENKO, P., SAABAS, A., TYUGU, E. (2005). COCOVILA – Compiler-Compiler for Visual Languages. In *Proc. of the 5th Workshop on Language Descriptions, Tools and Applications*, vol. 141, no. 4 of Electron. Notes in Theor. Comput. Sci., Elsevier, p. 137–142.

LIU, Z., JIFENG, H. (2006). *Mathematical Frameworks for Component Software: Models for Analysis and Synthesis* (Series on Component-Based Software Development). World Scientific Publishing Co., Inc.

LAU, K.-K. (2003) Component-Based Software Development and Logic Programming, *LNCS 2916*, p. 103–108.

LOWY, J. (2005). P*rogramming .NET Components*. 2nd Edition. O'Reilly

LUPEIKIENĖ, A., GIEDRIMAS, V. (2005). Component model and its formalisation for structural synthesis. In *Scientific Proc. of Riga Technical University. Computer Science,* vol. 22, p. 169–180.

MARTELLI, M., MASCARDI, V., ZINI, F. (1999). A Logic Programming Framework for Component-Based Software Prototyping. In *Proceeding of 2nd International Workshop on Component-based Software Development in Computational Logic (COCL'99)*, Brogi, A. and Hill, P. (eds), Paris.

MATSKIN, M., TYUGU, E. (2001). Strategies of Structural Synthesis of Programs and Its Extensions. *Computing and Informatics*, vol. 20, p. 1–25.

SZYPERSKI, C. (2002). *Component Software: Beyond Object-Oriented Programming*. 2nd. Addison-Wesley Longman Publishing Co., Inc.

THE APPLICATION OF GENERATIVE METHODS FOR .NET COMPONENT-BASED SOFTWARE SYNTHESIS

Vaidas Giedrimas

S u m m a r y

The component paradigm increases the performance of software development process by reusing pre-fabricated software components. However, the search and verification of components still take relatively long time. One of the possible solutions to increase software development productivity and quality is the application of formal methods. Many approaches have been presented to scientific community, however the biggest part of them are still in the beginning stage – the formal description of component model without practical implementations. The synthesis system of .NET component-based software is presented in this arcticle. This system is based on two generative methods: the Structural Synthesis of Programs (SSP) and the induction. One small example of component-based software synthesis is also described.