Liet. mat. rink. LMD darbai, **50**, 2009, 397–401

Kompozicinės aspektinių šablonų savybės

Žilvinas VAIRA, Albertas ČAPLINSKAS
Matematikos ir informatikos institutas
Akademijos g. 4, LT-08663 Vilnius
el. paštas: zilvinas@ik.ku.lt; alcapl@ktl.mii.lt

Santrauka. Straipsnyje konkretaus objektinio karkaso pavyzdžiu nagrinėjama aspektinio programavimo įtaka tradicinių objektinių projektavimo šablonų kompozicijoms. Žiūrima, ar jose nesidubliuoja komponuojamuose šablonuose atskirti turiniai ir, jeigu jie dubliuojasi, aiškinamasi to priežastys bei siūloma, kaip reikėtų keisti tų šablonų aspektizavimą, kad toks dubliavimasis išnyktų.

Raktiniai žodžiai: aspektinis programavimas, projektavimo šablonai, šablonų kompozicijos, turinių atskyrimas.

Įvadas

Šiuolaikinėje programų sistemų inžinerijoje objektinė paradigma užima labai svarbią vietą. Nepaisant to, yra kuriamos naujos paradigmos ir gali būti, kad jos ateityje pakeis objektinę. Viena iš objektinėje paradigmoje neišsprendžiamų problemų, skatinančių naujų programavimo krypčių atsiradimą, yra susipynusių turinių atskyrimas. Aspektinė paradigma siūlo būdą, leidžiantį panaikinti ar bent sumažinti turinių susipynimą sistemoje.

Ši paradigma gimė daugiau nei prieš 10 metų. Jai pradžią davė aspektinis programavimas [9,11], sukurtas *Xerox Palto Alto Research Center* tyrimų grupėje vadovaujamoje *Gregor Kiczales*. Pirmoji aspektinio programavimo kalba buvo *AspectJ* [10]. Nepaisant pasiektos pažangos, aspektinė paradigma vis dar nėra galutinai susiformavusi. Vis dar nėra pakankamai ištobulinti nei aspektinių kalbų realizavimo metodai, nei būdai, kaip tas kalbas reikėtų taikyti, kuriant programų sistemas. Nėra dar pakankamai ištobulinti ir aspektinėms sistemoms projektuoti skirti tipiniai projektavimo sprendimai, kurie, trumpumo dėlei, vadinami aspektiniais projektavimo šablonais. Objektiniai projektavimo šablonai [5] buvo sukurti beveik tuo pačiu metu kaip ir aspektinis programavimas. Taigi, jie naudojami ir tobulinami jau daug metų. Aspektiniai šablonai kol kas yra dar tik pradedami kurti. Vienas iš būdų kurti aspektinius šablonus yra objektinių šablonų aspektizavimas. Pirmieji bandymai [7] buvo daromi dar 2002 metais, rezultatai patobulinti 2004 metais. Darbe buvo pasiūlyta, kaip objektinius šablonus realizuoti *AspectJ* kalba. Straipsnis [7] iki šiol lieka savotišku modeliniu pavyzdžiu, savo darbuose juo remiasi daugelis autorių. Tačiau aspektinės realizacijos jame buvo gautos tiesiogiai perrašant objektinių šablonų realizacijas *Java* kalba. Toks sprendimas yra lokalus, šablonai perrašomi po vieną ir lieka neaišku, kokiu mąstu atsiskirs turiniai, imant šablonų kompozicijas. Šis straipsnis *SimJ* [3] objektinio

karkaso pavyzdžiu nagrinėja darbe [7] pasiūlytų aspektinių šablonų realizacijų kompozicines savybes. Siekta aspektizuoti karkase žurnalizavimą (angl. *logger*), atskiriant jį nuo kitų turinių, su kuriais jis yra supintas. Tai buvo daroma dviem būdais: panaudojant darbe [7] pasiūlytus aspektinius šablonus ir aspektizuojant karkasą tiesiogiai, t.y. nenaudojant šablonų. Bandymas patvirtino pradinę prielaidą, kad lokalus šablonų aspektizavimas nebūtinai užtikrina juos panaudojant sukurtos sistemos aspektizavimą.

Likusią straipsnio dalį sudaro 3 skyriai ir išvados. Pirmajame skyriuje apžvelgiami susiję kitų autorių darbai, antrajame skyriuje trumpai aptariamas eksperimentui naudotas objektinis karkasas ir pačio eksperimento metodika, trečiajame skyriuje aprašomi atlikto eksperimento rezultatai.

1. Susijusių darbų apžvalga

Siekiant, kad būtų aiškiau, trumpai aptarsime, kas tai yra aspektinis programavimas ir kuo jis skiriasi nuo objektinio.

Turinių atskyrimas yra vienas iš esminių programų sistemų inžinerijos principų, kuriuo siekiama suskaidyti sistemą į atskiras, kuo mažiau tarpusavyje persipinančias, dalis. Dauguma programavimo kalbų mums leidžia pasiekti tam tikrą turinių atskyrimo lygį, kuriant sistemą kaip hierarchinę kompoziciją, susidedančią iš mažesnių funkcinių komponentų (objektų, funkcijų, modulių ir pan.) [2]. Visgi tradicinių programavimo kalbų notacijų nepakanka atskirti daugiau komponentų turintiems turiniams, tokiems kaip: žurnalizavimas, sinchronizacija, saugumo kontrolė ir kt. Tokių turinių realizacijų fragmentai dažniausiai būna išsibarstę po visą sistemą. Aspektinis programavimas leidžia išspręsti šią problemą. Konkrečios aspektinės kalbos paveldėjo ir objektams būdingas savybes, tačiau esminė jų struktūra ir elgsena smarkiai skiriasi nuo objektų. Aspektai yra aprašomi kaip tam tikros taisyklės, nurodančios sistemos turinių susikirtimo taškus, pagal kurias aspektinis kompiliatorius išbarsto reikiamus kodo fragmentus kituose turiniuose. Tokie skirtumai leidžia manyti, kad aspektinis programavimas gali įtakoti objektinių projektavimo šablonų struktūrą ar sąlygoti naujų šablonų atsiradimą.

Daugelyje tyrimų bandoma parodyti, kad aspektinis programavimas pagreitina paties programavimo procesą ir palengvina sukurtų programų priežiūrą [12]. Rezultatai yra vertinami pagal įvairius kriterijus, įskaitant kodo eilučių skaičių, kodo pasikartojimus, programų rišlumą, projekto vykdymo laiką ir pan. [4,12]. Aspektinės projektavimo šablonų realizacijos beveik visais atvejais yra pripažįstamos geresnėmis nei jų objektiniai analogai [7,6]. Tačiau daugelis šių darbų analizuoja šablonų realizacijas, palikdami nuošalyje tokius svarbius klausimus kaip: ar objektiniai šablonai apskritai keltini į aspektinę paradigmą (galbūt čia reikia visai kitų projektavimo šablonų), o jei taip, tai ar aspektinė paradigma kaip nors keičia perkeliamų šablonų pobūdį arba jų panaudojimo kontekstą? Vienas iš detalesnių tyrimų, kuriame bandoma analizuoti aspektinių šablonų kompozicijas didelėse sistemose, pateiktas darbe [1]. Straipsnyje analizuojamos šablonų porų kompozicijos. Autoriai suskirsto objektinių šablonų kompozicijas į 4 kategorijas: kreipimusi paremtos kompozicijos, klasių persidengimu paremtos kompozicijos, metodų persidengimu paremtos kompozicijos ir sutampančios kompozicijos. Pirmoji kategorija nagrinėja kompozicijas, kuriose šablonai susieti tik kreipiniais iš vieno šablono į kitame šablone esančių klasių metodus. Antrajai kategorijai priklauso kompozicijos, kuriose šablonai persidengia naudodami tas pačias klases

skirtingiems vaidmenims realizuoti. Trečia kategorija apima kompozicijas, kuriose šablonai persidengia, naudodami tuos pačius metodus skirtingiems vaidmenims realizuoti. Ir ketvirta kategorija apima kompozicijas, kuriose šablonai arba iš dalies persidengia, naudodami keletą tų pačių klasių ir metodų, arba vienas šablonas yra panaudotas kito šablono viduje. Toliau šablonai buvo aspektizuojami darbe [7] pasiūlytu būdu *AspectJ* kalba, nežymiai keičiant kai kurių šablonų realizaciją. Tyrimo rezultatai vertinami remiantis tokiais matavimais, kaip: susipynimo laipsnis, programos rišlumas, programos dydis ir turinių atskyrimo laipsnis [13]. Matuojamos ir tarpusavyje lyginamos buvo objektinių ir aspektizuotų šablonų kompozicijos. Tyrimas parodė, kad aspektizavimo rezultatai priklauso nuo to, kiek sėkmingai pavyko atskirti turinius aspektizuotuose šablonuose, nuo turinių susipynimo kompozicijose ir nuo aspektizuojamos sistemos reikalavimų. Tačiau straipsnyje nepasakyta, kaip konkrečiai buvo pakeista šablonų realizacija ir kokių turinių nepavyko atskirti. Todėl iš pateiktų rezultatų spręsti apie nagrinėtų šablonų konkrečias kompozicines savybes negalima.

2. Eksperimento metodika

Kompozicinėms aspektinių projektavimo šablonų savybėms tirti buvo panaudotas objektinis karkasas *SimJ* [3].

Kompozicinės šablonų savybės darbe yra suprantamos dvejopai: pirma – tai savybės leidžiančios šablonus apjungti tarpusavyje, antra – tai tokios šablonų savybės, kaip veiksmingumas ir nekintamumas, kurios turi būti užtikrinamos apjungiant šablonus. Transformuojant objektinius šablonus į aspektinius, gali pakisti šablonų struktūra ir šablonų jungimo komponentai, tai gali įtakoti ir jų veiksmingumą kompozicijose.

Karkase yra panaudoti šie objektiniai šablonai: *Singleton*, *Adapter*, *Fasade*, *Factory Method*, *Flyweight*, *Iterator*, *State* ir *Template* [5]. Apskritai, naudojami kompozicijose jie persidengia ir ne visais atvejais juos galima atpažinti ir išskirti. Todėl ne visų karkase panaudotų šablonų kompozicijos buvo tiriamos.

Susidurta ir su kitais sunkumais. Darbe [7] pateiktų pavyzdžių nepakako aspektizuoti šablonus konkrečiuose kontekstuose. Pavyzdžiui, *Singleton* realizacijos pavyzdys negali būti tiesiogiai pritaikomas, kai konstruktoriuje yra naudojami parametrai. Todėl jį pavyko aspektizuoti tik kai kuriais atvejais. Dažniausiai šablonų realizacijos yra priklausomos nuo jų panaudojimo konteksto. Be to, vieno šablono aspektizavimo būdas daro įtaką kitų, su juo tame kontekste susijusių, šablonų aspektizavimui.

SimJ karkasas yra skirtas kurti lokalių ir išskirstytų diskrečiųjų įvykių simuliatorius, panaudojant aukšto lygmens architektūras [8]. Simuliatoriai gali būti kuriami skirtingose dalykinėse srityse. Tyrime buvo naudojamas eilių parduotuvėse simuliatorius, sukurtas panaudojant šį karkasą. Kadangi tyrime yra svarbiausia nustatyti aspektizavimo įtaką objektiniams projektavimo šablonams, karkasas ir simuliatorius yra naudojami tik kaip eksperimento įrankiai. Svarbu yra tai, kad šablonai karkase yra apjungti į kompozicijas, o ne aprašyti, kaip atskiri pavyzdžiai. Ateityje planuojama tirti šablonus ir kitokio tipo realizacijose.

Karkase turinių atskyrimas buvo atliktas dviem būdais: panaudojant aspektizuotus šablonus ir tiesiogiai aspektizuojant atitinkamus modulius. Buvo siekta aspektizuoti tik patį karkasą sudarančias abstrakčiąsias klases, nekeičiant parduotuvių simuliatoriaus

realizacijos. Tačiau tai padaryti pavyko ne visais atvejais, nes buvo šablonų, kurie tik iš dalies realizuojami pačiame karkase. Jų realizacija yra užbaigiama karkaso užpilde.

Atliekant bandymą, siekta nuo kitų turinių atskirti žurnalizaciją, kurią realizuoja specialiai tam skirtas modulis, kuris susipina su kitomis karkaso dalimis, t.y. yra išbarstytas po kitus modulius.

3. Eksperimento rezultatai

Palyginus abi realizacijas, atliktas aspektizuojant karkasą prieš tai minėtais dviem būdais, buvo gauti žemiau pateikiami rezultatai.

Pirmuoju atveju teko šiek tiek modifikuoti šablonų realizacijas tam, kad būtų galima pritaikyti jas kiekvienai kompozicijai. Vienam *Singleton* šablonui aspektizavimas nebuvo pritaikytas, kadangi aspektinis šablono variantas tam netiko. Realizuojant *Adapter* šablonus, teko šiek tiek modifikuoti aspektinį šablono variantą.

Antruoju atveju, gavosi labiau atskirtas žurnalizacijos turinys ir lankstesnis kodas bei struktūra. Nenaudojant šablonų, suteikiama didesnė programavimo laisvė, todėl buvo gautas lankstesnis ir aiškesnis kodas. Tačiau, skirtingai nei pirmuoju atveju, pranešimų perkėlimas į atskirą klasę įtakoja karkaso užpildą.

Apibendrinti eksperimentų rezultatai yra pateikiami 1 lentelėje. Abiem atvejais pavyko atskirti žurnalizacijos turinį. Eksperimentų metu buvo aspektizuojami *Adapter*, *Singleton* ir *Flyweight* šablonai. Tiesiogiai su žurnalizacijos turiniu buvo susiję 4 *Adapter* ir 1 *Singleton* šablonas. Kadangi šie šablonai jungiasi kompozicijoje su kitais, aspektizavimas paveikė ir pastaruosius šablonus.

Iš gautų rezultatų matome, kad [7] pasiūlyti aspektiniai šablonai nėra tokie universalūs kaip jų objektiniai analogai. Nekeičiant šablonų struktūros nepavyksta visiškai išvengti turinių dubliavimosi. Tai patvirtina prielaidą, kad [7] naudotas šablonų transformavimas yra lokalus ir orientuotas į konkretų pavyzdį. Dėl to pakinta šablonų kompozicinės savybės. Siekiant išvengti tokių nesuderinamumų, transformuojant šablonus reikėtų išlaikyti pagrindinius šablonų jungimosi komponentus. Taip pat, atsižvelgiant į rezultatus, gautus nenaudojant šablonų, reikėtų patikrinti galimas alternatyvias konstrukcijas. Tačiau tam reikalingas platesnis praktinis tyrimas, ką ir planuojama atlikti tolimesniuose darbuose.

1 lentelė. Gauti rezultatai taikant skirtingus metodus

Rezultatai	Metodas	
	Taikant šablonus	Netaikant šablonų
Žurnalizacijos turinio atskyrimas	Atskirtas iš dalies	Pilnai atskirtas
Gauti komponentai	1 aspektas	2 aspektai, 2 klasės
Susiję šablonai	4 Adapter, 1 Singleton	
Paveikti šablonai	4 Adapter, 4 Singleton, 1 Flyweight	
Karkaso užpildo įtakojimas	Karkaso užpildas išlieka nepakitęs	Keičiasi karkaso užpildas

4. Išvados

Remiantis gautais rezultatais galima teigti, kad norint atskirti turinius sistemoje, nepakanka esamų aspektizuotizuotų objektinių šablonų. Jų realizacijos nėra pakankamai universalios ir nenusako panaudojimo konteksto. Siekiant išnagrinėti visas aspektinių šablonų kompozicines savybes reikia bendresnių šablonų realizacijų pavyzdžių ir detalesnio jų panaudojimo tyrimo. Tokius tyrimus planuojama atlikti artimiausioje ateityje.

Literatūra

1. N. Cacho, E. Figueiredo, C. Santanna, A. Garcia, T. Batista, C. Lucena. Aspect-oriented composition of design patterns: a quantitative assessment. *Monografias em Ciencia da Computacao, Rio De Janeiro, Brasil*, 5(34):34–67, 2005.
2. K. Czarnecki, U. W. Eisencker. *Generative Programming*: *Methods, Tools, and Applications*. Addison-Wesley, 2000.
3. A. D'Ambrogio, D. Gianni, G. Iazeolla. SimJ: a framework to distributed simulators. In: *Proc. of the 2006 Summer Computer Simulation Conference* (*SCSC06*), Calgary, Canada. Series: Simulation. Society for modeling and simulation, 149–156, 2006.
4. T. Eidson, J. Dongarra, V. Eijkhout. Applying aspect-orient programming concepts to a component-based programming model. In: *Proceedings of the 17th International Parallel and Distributed Processing Symposium* (*IPDPS*), Nice, France. IEEE Computer Society, 205b, 2003.
5. E. Gamma, R. Helm, R. Johnson, J. Vlissides, G. Booch. *Design Patterns*: *Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1998.
6. O. Hachani, D. Bardou. Using aspect-oriented programming for design patterns implementation. In: *Proc. OOIS 2002 Workshop on Reuse in Object-Oriented Information Systems Design*, Montpellier, France. Springer-Verlag, 2002.
7. J. Hannemann, G. Kiczales. Design pattern implementation in Java and AspectJ. In: *Proc. of the 17th Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA '02*). ACM Press, 161–173, 2002.
8. F. Khul, R. Weatherly, J. Dahmann. *Creating Computer Simulation Systems: An Introduction to High Level Architecture*. Prentice Hall, 1999.
9. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loingtier, J. Irwin. Aspect oriented programming. In: *Proc. of the European Conference on Object-Oriented Programming* (*ECOOP 1997*), Jyväskylä, Finland. Springer-Verlag, 220–242, 1997.
10. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G. Griswold. An overview of AspectJ. In: *Proc. of the European Conference on Object-Oriented Programming* (*ECOOP 2001*) Budapest, Hungary. Springer-Verlag, 327–353, 2001.
11. C.V. Lopes. Aspect-oriented programming: A historical perspective (what's in a name?). In: *Proc. Aspect-Oriented Software Development*, Bonn, Germany. Addison-Wesley, 97–122, 2005.
12. O. Papapetrou, G.A. Papadopoulos. Aspect Oriented Programming for a component based real life application: A case study. In: *Proc. ACM Symposium on Applied Computing*, Nicosia, Cyprus. ACM, 1554–1558, 2004.
13. C. Santanna, A. Garcia, C. Chavez, C. Lucena, A. von Staa. On the reuse and maintenance of aspect-oriented software: An assessment framework. In: *Proc. of Brazilian Symposium on Software Engineering* (*SBES'03*), Manaus, Brazil. ACM, 19–34, 2003.

SUMMARY

Ž. Vaira, A. Čaplinskas. Compositional aspect-oriented design pattern properties

In this publication we analyze traditional object-oriented design pattern aspectual compositions implemented in object-oriented framework. The main task is to look whether the design pattern compositions present concern duplications, and if they do then what causes it and how to avoid it.

Keywords: aspect-oriented programming, design patterns, design pattern composition.