Elektrotehniški vestnik 73(1): 71-76, 2006
Electrotechnical Review: Ljubljana, Slovenija

Identifikacija ustreznih realizacij načrtovalskih vzorcev

Simon Beloglavec, Marjan Heričko

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17, 2000 Maribor, Slovenija
E-pošta: {simon.beloglavec, marjan.hericko}@uni-mb.si

Povzetek. Načrtovalski vzorci so za izkušene načrtovalce nepogrešljivo orodje pri razvoju objektno orientirane programske opreme. Izkušnje, zajete v vzorcih, se preslikujejo v načrt programske opreme skozi realizacijo načrtovalskega vzorca. Realizacija nadgrajuje načrt s spremembami v strukturi objektov in v njihovi medsebojni komunikaciji. Vsi načrtovalski vzorci dopuščajo množico različnih realizacij. Le nekatere od njih so ustrezne za posamezno problemsko področje. Stopnja ustreznosti se določa s pomočjo programsko-načrtovalskih metrik. Metrične ocene razkrivajo kvalitativne lastnosti nadgrajenega načrta. Pridobivanje in primerjanje metričnih ocen omogoča identifikacijo najprimernejše realizacije. Ocenjevanje in primerjanje velikega števila realizacij za posamezni vzorec v določen del načrta je zamudno opravilo, ki ni izvedljivo pri projektih s kratkim izvedbenim rokom. Tako je izbira učinkovite realizacije odvisna od izkušenj načrtovalca programske opreme. Sprotna podpora pri izbiri realizacije omogoča manj izkušenim načrtovalcem gradnjo kakovostnejših načrtov. Podpora postane smiselna, če zagotavlja primerjavo in izbiro realizacije v sprejemljivem časovnem okviru. Osnova za iskanje ustrezne realizacije je specifikacija dopustnih realizacij načrtovalskega vzorca. Članek predstavlja rešitvi za ključna izziva: opisovanje dopustnih realizacij za posamezni načrtovalski vzorec in razvoj podpornega orodja za identifikacijo ustreznih realizacij. Predstavljen opis dopustnih realizacij temelji na formalni notaciji TLA+. Predstavljena in ovrednotena je arhitektura razvitega porazdeljenega podpornega orodja za identifikacijo realizacij.

Ključne besede: objektno orientirano načrtovanje, načrtovalski vzorci, metrike, realizacija načrtovalskih vzorcev, Java

Identification of Suitable Design-Pattern Realizations

Extended abstract. Using design patterns in the process of developing object-oriented software has become mandatory for experienced engineers. The proven knowledge that is contained in design patterns is mapped into a software design through pattern realization. Pattern realization updates the design with changes in the object's structure and inter-object communication. All design patterns allow a variety of acceptable realizations. In a specific solution domain, some realizations prove to be more suitable than others. The level of suitability can be determined with the help of software design metrics. The metric scores reveal the quality attributes of the updated design. Through the gathering and comparison of the metric scores the appropriate realizations can be determined. The number of possible realizations for the design pattern can be significantly high and the execution of such tasks is not always feasible for projects with tight schedules. The selection of effective realizations depends on the experiences of a software engineer. Continuous support during the selection process can help less experienced engineers build better software designs. This support should provide a comparison for selecting pattern realizations in an acceptable amount of time. The presented approach makes use of concurrency and networking mechanisms in the Java software-platform, which enables evaluation of design pattern realizations in reasonable time. Acceptable pattern realization should be specified for each design pattern. For this purpose we present the formal specification for design patterns with the TLA+ specification language. This specification clearly specifies allowable changes of internal pattern elements. With this additional pattern information, possible pattern realization can be generated and evaluated by a support tool. The concurrent evaluation that is performed on multiple network nodes ensures acceptable response times. We present a prototype for the support tool developed during the research.

Software engineers use design-pattern catalogues containing related design patterns. Pattern descriptions given in the catalogues are general and do not specify possible pattern realizations. Section 1 presents the background information on attempts for specifying and

Prejet 1. junij, 2005
Odobren 30. januar, 2006

selecting appropriate pattern realization. A new approach to the realization specifications, with the help of specification language TLA+, is presented in Section 2. We demonstrate our approach to specifying realization on the example of commonly used MVC pattern presented in Figure 1. Construction of pattern realization specification is shown on some elementary design patterns from GoF catalogue. Section 3 presents challenges encountered during developing the support tool that automates the selection process. Figure 2 gives a comparison of the various considered architectures. The analysis of the measurements performed for all cases has led to the architecture depicted in Figure 3. The use of the support tool is shown in Figures 4 and 5.

Key words: object-oriented design, design patterns, metrics, design pattern realizations, TLA+, Java.

1 Uvod

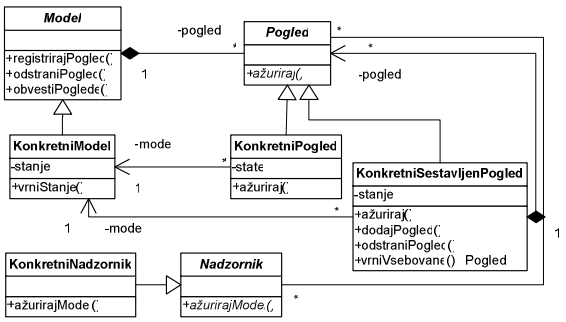
Načrtovalski vzorci so nepogrešljivo orodje pri zagotavljanju kakovostnih načrtov programske opreme. Uporaba katalogov dobrih praks je znana že s področja arhitekture, iz del avtorja Alexandra[1], ki so motivacija za oblikovanje temeljnega kataloga načrtovalskih vzorcev za programsko opremo [2]. Po uveljavitvi tega kataloga, ki vsebuje ne-formalne definicije vzorcev, je nastala množica neodvisnih poskusov formaliziranja opisov vzorcev ([3], [4]). V obstoječih delih se avtorji usmerjajo na formalizacijo samega vzorca, njihova realizacija v določenih kontekstih ostaja nenaslovljena. Natančna definicija mogočih realizacij za posamezne programske jezike omogoča avtomatsko kreiranje realizacij. Mogočih je veliko realizacij za posamezni načrtovalski vzorec in to število ni primerno za neposredno uporabo brez računalniško podprtega izbiranja realizacij. Za dinamično obvladovanje množice realizacij med procesom načrtovanja potrebujemo podporno orodje, ki zmore zagotoviti sprotno pomoč pri izbiri najustreznejše realizacije za posamezni kontekst. Realizacija z najustreznejšimi lastnostmi se identificira s pomočjo metričnega ocenjevanja.

Načrtovanje arhitekture podpornega ocenjevalnega orodja zajema razvoj strežniške infrastrukture, ki zmore ponuditi ocenjevanje predvidljivih realizacij v sprejemljivem času. Sprejemljivo je orodje, ki je sposobno identificirati najustreznejšo realizacijo glede na podana merila v enakem ali manjšem času, kot načrtovalec brez podpore predvidi mogoče realizacije brez ocenjevanja ustreznosti. Načrtovalec miselno analizira predvidene realizacije glede na svoje izkušnje. Orodje natančno zajame pri predvidevanju vse mogoče kombinacije in jih metrično ovrednoti. Mogoče strežniške infrastrukture, ki so sposobne ustrezno oceniti veliko množico kandidatnih realizacij, so natančno analizirane v raziskavi [5].

Prispevek predstavlja v drugem razdelku predlog za formalizacijo realizacij načrtovalskih vzorcev. Skupina opisov je specifična za posamezen programski jezik. V okviru raziskave smo dopolnili temeljni katalog načrtovalskih vzorcev GoF s specifikacijami realizacij za programski jezik Java. Specifikacije omogočajo natančno predvidevanje mogočih realizacij, ki jih je mogoče s pomočjo podpornega orodja oblikovati in metrično ovrednotiti. Metrično ovrednotenje izpostavi kvalitativne lastnosti posamezne realizacije in poenostavi iskanje ustrezne realizacije. Zaradi pomembnosti obvladovanja množice kombinacij je poudarek, razen na formalizaciji možnih realizacij, tudi na razvoju ustreznega podpornega orodja. Izzivi pri načrtovanju takega orodja so predstavljeni v tretjem razdelku. Četrti razdelek povzema izsledke raziskave.

2 Specifikacija realizacij načrtovalskih vzorcev

Obstoječi katalogi načrtovalskih vzorcev ne dajejo natančnega opisa mogočih realizacij vzorcev v posameznih programskih jezikih. Opisi vzorcev v katalogih zajemajo le nabor smernic, ki naredijo načrtovalski vzorec uporaben v različnih domenah in kontekstih. V fazi podrobnega načrtovanja programske opreme je treba vzorec iz kataloga realizirati v načrtu. Glede na podano strukturo posameznega vzorca in značilnosti ciljne domene je mogoče predvideti dopustne realizacije. Sam postopek ugotavljanja le-teh je zaradi velikega števila možnosti zamuden. Za podrobnejšo razlago raznolikosti realizacij bo uporabljen primer pogosto uporabljenega načrtovalskega vzorca MVC (*Model-View-Controller*) [2] oziroma Model-Pogled-Nadzornik. Vzorec predstavlja ustaljeno prakso za komunikacijo med podatkovnimi objekti in objekti za vizualizacijo podatkov. Ideja vzorca je prav tako uporabljena v tri-nivojskih aplikacijah. Realizacija ideje vzorca ne pomeni enolične preslikave v posameznem kontekstu. Vzorec MVC ni atomarni vzorec, temveč ga sestavljajo trije vzorci: kompozicija, strategija in opazovalec (predstavljeni v katalogu GoF [2]). Kompozicija skrbi za učinkovito sestavljanje pogledov in komunikacijo med njimi. Vzorec strategija narekuje način vključevanja različnih načinov spreminjanja modela iz posameznih pogledov. Opazovalec narekuje način obveščanja pogledov s strani modela, ko se spremenijo podatki.



Slika 1. Razredni diagram realizacije načrtovalskega vzorca MVC
Figure 1. Class diagram for the MVC design pattern

Slika 1 prikazuje eno izmed mogočih realizacij vzorca MVC. Izbira najbolj splošne realizacije lahko povzroči vnašanje množice objektov v načrt, ki omogočajo fleksibilnost, vendar za aktualno rešitev nimajo pravega pomena. Prikazani primer predvideva uporabo sestavljenih pogledov, zagotavljanje splošnosti pri definiranju novih modelov in nadzornikov. V določenih primerih vsa omenjena splošnost vnaša le dodatno kompleksnost v model ter slabi njegovo preglednost in kakovost.

Zgodnja analiza predvidljivih realizacij prepreči neustrezne izbire. Predvidevanje realizacij mora upoštevati veljavna pravila ciljnega programskega jezika, idejo vzorca in možnosti, ki jih daje ciljni načrt. Če je na voljo formalni zapis mogočih realizacij, je omogočena avtomatizacija postopka. Avtomatsko kreiranje pravil za realizacije ni mogoče, saj je načrtovalski vzorec praviloma zapisan v splošni obliki. Splošnost daje možnost raznolikih aplikacij, hkrati pa je vzrok za neustrezne strukture v danem kontekstu. Vsak vzorec ima navedeno referenčno strukturo realizacije, ki vsebuje opcijske in ponovljive dele. Opcijske dele vzorca je mogoče izpustiti brez spreminjanja ideje vzorca. Ponovljivi del ima določeno minimalno kardinalnost, maksimalna ni nujno omejena.

Realizacije vzorca lahko obravnavamo kot transformacije od najfleksibilnejše oblike do minimalne oblike (najmanj splošne). Opis v nadaljevanju obravnava načrtovalski vzorec kot dinamično, spreminjajočo se strukturo in ne kot statičen načrt. Formalna notacija TLA+ [6] predstavlja primerno obliko za opisovanje. Slovnica notacije temelji na začasni logiki prvega reda in teoriji množic.

V nadaljevanju bomo podali primer zapisa mogočih realizacij za načrtovalski vzorec Opazovalec, ki je del vzorca MVC. Podani primer se zaradi preglednosti omejuje na raven razredov in vmesnikov. Podrobna specifikacija zajema še dopustne spremembe atributov in metod. Pri opisovanju v slovnici TLA+ definiramo v prvem koraku konstante in spremenljivke opisovanega sistema. Sestavni deli vzorca bodo predstavljali spremenljivke, ki se lahko spreminjajo v skladu s specifikacijo. Obravnavani vzorec sestavljajo elementi: predpis in delna implementacija skupnih funkcionalnosti opazovanega dela (*opDel*), konkretna implementacija predpisov opazovanega dela (*konOpDel*), predpis opazovalca (*opazovalec*) in konkretna implementacija tega predpisa (*konOpazovalec*). Konstante označujejo vrednosti, ki jih poleg vgrajenih tipov lahko zavzamejo spremenljivke.

CONSTANTS
abstraktniRazred, razred, vmesnik

VARIABLES
opDel, konOpDel, opazovalec, konOpazovalec

V nadaljevanju je treba opredeliti dovoljene vrednosti za posamezne spremenljivke. V tej definiciji dejansko določimo tipe vrednosti skladno s smernicami vzorca in pravili ciljnega programskega jezika.

$$TypeInvOpDel == opDel \in [tip: \{razred, abstraktniRazred\}, stRealizacij: \text{INTEGER}]$$

$$TypeInvKonOpDel == opDel \in [tip : \{razred, abstraktniRazred\}, stRealizacij: \text{INTEGER}]$$

$$TypeInvOpazovalec == opazovalec \in [tip: \{razred, abstraktniRazred, vmesnik\}, stRealizacij: \text{INTEGER}]$$

$$TypeKonOpazovalec == konOpazovalec \in [tip: \{razred, abstraktniRazred\}, steviloRealizacij: \text{INTEGER}]$$

$$TypeInv == TypeInvOpDel \land TypeInvKonOpDel \land TypeInvOpazovalec \land TypeInvKonOpazovalec$$

Zadnji izraz določa vse dopustne tipe, ki lahko nastopijo v realizacijah vzorca. Predstavljena specifikacija realizacij je tudi polna in primerljiva specifikacija celotnega načrtovalskega vzorca glede na obstoječe specifikacije ([3], [4]).

$$InitOpazovaniDel == opDel \in [tip: \{abstraktniRazred\}, stRealizacij : \{1\}]$$

$$InitKonOpDel == opDel \in [tip : \{ abstraktniRazred \}, stRealizacij : \{1\}]$$

$$InitOpazovalec == opazovalec \in [tip : \{vmesnik\}, stRealizacij : \{1\}]$$

$$InitKonOpazovalec == konOpazovalec \in [tip : \{ abstraktniRazred \}, stRealizacij : \{1\}]$$

$$Init = InitOpDel \land InitKonOpDel \land InitOpazovalec \land InitKonOpazovalec$$

Začetno stanje predstavlja najsplošnejšo obliko vzorca, ki zajema vse opcijske dele in najsplošnejše oblike. Dopustno ponovljivi elementi vzorca so v tem stanju navedeni največ enkrat.

$$specializirajOpDel(i) == \land opDel.stRealizacij = 1 \land \neg opDel[i].specializirajOpDel \land opDel' = [opDel \text{ EXCEPT } !.tip = razred \land \text{UNCHANGED} << konOpDel, opazovalec, konOpazovalec >>$$

$$specializirajKonOpDel(i) == \land konOpDel.stRealizacij > 0 \land \neg konOpDel[i].specializirajKonOpDel(i)$$

$\wedge$ *konOpDel*' = [*konOpDel* EXCEPT !.*tip* = *razred*
$\wedge$ UNCHANGED<<*opDel,opazovalec, konOpazovalec*>>

specializirajOpazovalec(*i*) ==
$\vee\wedge$ *opazovalec.stRealizacij* = 1
$\wedge$ *opazovalec.tip* = *vmesnik*
$\wedge$ *opazovalec*' = [*opazovalec* EXCEPT !.*tip* = *abstraktniRazred*]
$\wedge$ UNCHANGED<<*opDel,konOpDel*>>
$\vee\wedge$ *opazovalec.stRealizacij* = 1
$\wedge$ *opazovalec.tip* = *abstraktniRazred*
$\wedge$ *opazovalec*' = [*opazovalec* EXCEPT !.*tip* = *razred*]
$\wedge$ UNCHANGED<<*opDel, konOpDel, konOpazovalec*>>

specializirajKonOpazovalec(*i*) ==
$\wedge$ *konOpazovalec.stRealizacij* > 0
$\wedge \neg$ *konOpazovalec[i].specializirajKonOpazovalec*(*i*)
$\wedge$ *konOpazovalec*' = [*konOpazovalec* EXCEPT !.*tip* = *razred*
$\wedge$ UNCHANGED<<*opDel,konOpDel,opazovalec*>>

dodajKonOpDel() ==
$\wedge$ *konOpDel*'=[*konOpDel* EXCEPT !.*stRealizacij*'=*stRealizacij*+1]
$\wedge$ *dodajKonOpazovalec*()
dodajKonOpazovalec() == $\wedge$ *konOpazovalec*' = [*konOpazovalec*
EXCEPT !.*stRealizacij*' = *stRealizacij*+1]

Dopustno spreminjanje realizacij načrtovalskega vzorca opisujeta operaciji specializacije in dodajanja. Specializacija zmanjšuje splošnost posameznega tipa, operacija dodajanje pa omogoča večanje kardinalnosti redundantnih elementov. Posamezna definicija operacije natančno določa pogoje, pod katerimi se lahko izvede. Na primer: specializacija opazovanega dela se lahko izvede, če že obstaja realizacija in sama specializacija še ni bila izvedena. Po specializaciji se spremeni tip, drugi elementi vzorca pa ostanejo nespremenjeni. Element opazovalec ima definirani dve fazi specializacije. Načrtovalski vzorec Opazovalec v osnovni obliki nima opcijskih delov, ki jih je mogoče izpustiti, zato operacija za izpustitev delov vzorca ni navedena.

specializiraj(*i*) == *specializirajKonOpDel(i)* $\vee$
specializirajOpazovalec $\vee$ *specializirajKonOpazovalec*

dodaj() == *dodajKonOpDel*(*i*) $\vee$ *dodajKonOpazovalec*(*i*)

preoblikuj(*i*) == *specializiraj*(*i*) $\vee$ *dodaj*()

Spec ==
Init$\wedge$[][*preoblikuj*]<<*opDel,konOpDel,opazovalec,konOpazovalec*>>

Posamezne definicije operacije so v sklepu združene v skupno definicijo, ki je enotno določilo za obnašanje sistema za kreiranje dopustnih realizacij vzorca.

Za notacijo TLA+ [7] obstaja množica razpoznavalnikov, ki omogočajo preverjanje pravilnosti notacije. TLA+ predstavlja zrelo sintakso, ki se je razvila iz sintakse TLA, predstavljene leta 1994. Podporno orodje, razvito v okviru raziskave, vsebuje definicije dopustnih realizacij za temeljni katalog načrtovalskih vzorcev GoF [2]. Orodje prav tako omogoča poznejše dopolnjevanje množice določil s specifikacijami drugih katalogov.

Specifikacija dopustnih realizacij omogoča predvidevanje mogočih realizacij in njihovega vključevanja v določeni del načrta. Pri vključevanju ima načrtovalec možnost odločiti, ali se realizacija posameznega dela vzorca zlije z elementom obstoječega načrta (element načrta mora biti istega tipa kot element vzorca), ali predstavlja nov element načrta. Vsaka mogoča kombinacija z načrtom se metrično ovrednoti.

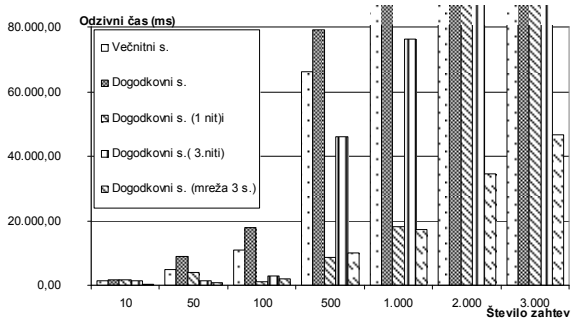
Metrično ovrednotenje uporablja nabore metrik, ki so značilni za ciljno programsko paradigmo ali za programski jezik. Raziskava se je omejila na objektno-orientiran pristop in programski jezik Java. Smernica pri oblikovanju nabora načrtovalskih metrik so bili uveljavljeni nabori za objektno-orientirane sisteme ([7], [8]).

Določanje intervalov sprejemljivosti za metrične ocene omogoča zmanjševanje števila kandidatov. Z izpostavljanjem posamezne lastnosti pa se iz množice izpostavi najugodnejša realizacija. Na primer načrtovalec komponente bo dovoljeval sklopljenost med razredi iste komponente, medtem ko bo v drugih primerih le-ta nezaželena. Področje metričnega ovrednotenja programske kode je dobro raziskano področje, zato mu v prispevku ni dan poseben poudarek.

3 Orodje za identifikacijo ustrezne realizacije

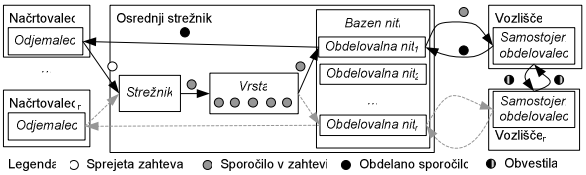
Cilj razvoja podpornega orodja je bil pokazati možnost razvoja orodja, ki se lahko vključi v proces razvoja programske opreme in daje sprejemljivo sprotno podporo pri izbiranju realizacije posameznega vzorca. Ocenjevalno orodje, ki zagotavlja sprotno metrično ocenjevanje in identifikacijo ustreznih realizacij, mora upoštevati dejstvo, da je število predvidljivih realizacij za posamezni vzorec veliko in ni primerno za obdelavo na samostojnem računalniku. Primerna osnova za postavitev arhitekture je omrežni način obdelave (*grid computing*) [8]. Za ciljno platformo je bila izbrana Java, predvsem zaradi prenosljivosti med strojnimi platformami in prisotnosti optimizacijskih mehanizmov za strežniške obdelave. Namen podpornega orodja je izkoriščati množico razpoložljivih računalnikov (vozlišč) v omrežju. Možnost za doseganje zadovoljive odzivnosti je na podlagi pridobljenih ocen pričakovana le v združevanju procesorskih moči posameznih vozlišč. Podporno orodje upravlja omrežno komunikacijo in zagotavlja hkratno ocenjevanje s pomočjo dveh razpoložljivih osnovnih oblik strežniške infrastrukture: večnitnega in dogodkovno-orientiranega strežnika. Na podlagi že izvedenih raziskav [5] je bila sestavljena primerna kombinacija obeh infrastruktur. Pri postavljanju osnovne strežniške infrastrukture so bile upoštevane značilnosti izbrane javanske platforme. Na učinkovitost obravnave velikega števila odjemalcev vplivajo poleg moči procesorja, velikosti pomnilnika in pasovne širine mrežne povezave še dejavniki, povezani z mehanizmi temeljnih storitev v izvajalni platformi. To so storitve za upravljanje večnitnih aplikacij in omrežnih protokolov.

Namen orodja je zagotoviti hkratno ocenjevanje čim večjega števila posameznih realizacij v čim krajšem času. Večina vozlišč je eno-procesorskih, zato je hkratnost na takih vozliščih navidezna. Hkratnost sprejemanja zahtev za ocenjevanje in njegova izvedba sta ključnega pomena. Zagotovljena je lahko v obliki večnitnega strežnika, ki dodeli vsakemu odjemalcu posebno nit, v okviru katere se sprejemajo njegove zahteve za ocenjevanje. Nit je dodeljena za celoten čas seje z odjemalcem in se sprosti po ocenitvi. Alternativa strogo ekskluzivnemu dodeljevanju niti je možnost prestrezanja omrežnih dogodkov, ki nastopijo med komunikacijo z odjemalcem. Primera takih dogodkov sta registracija novega odjemalca in prispetje posameznih delov zahteve. Po prejemu zahteve je na vrsti ocenjevanje. V dogodkovnem strežniku brez optimizacij to pomeni, da v aktualni niti poleg upravljanja dogodkov poteka še ocenjevanje. Če je ocenjevanje dolgotrajno, se kopičijo obvestila o še neobdelanih dogodkih. Mogoče nadgradnje vključujejo vpeljavo večnitnega pristopa, ki v tem primeru skrbi za izvedbo ocenjevanja. Za učinkovito hkratno obravnavo opravil obstajajo načrtovalski vzorci. Ti so bili smernice za postavitev končne arhitekture [9]. Dodatne "delavske niti" prevzemajo ocenjevanje in s tem sprostijo glavno nit, katere izključna naloga je sprejemati mrežne dogodke. Z namenom, da se zagotovi dejansko hkratno izvajanje, je izvajanje ocenjevanja preseljeno na vozlišča. Izvedena je bila primerjava izvedbe ocenjevanja na prej omenjenih arhitekturah. Simulirana množica odjemalcev je pošiljala zahteve po ocenjevanju podpornemu orodju. Predvidevamo, da bo načrtovalec pri svojem delu posredoval ocenjevalnemu orodju hkrati tudi po več zahtev, v katerih bo iskal ustrezno rešitev za različne dele načrta.



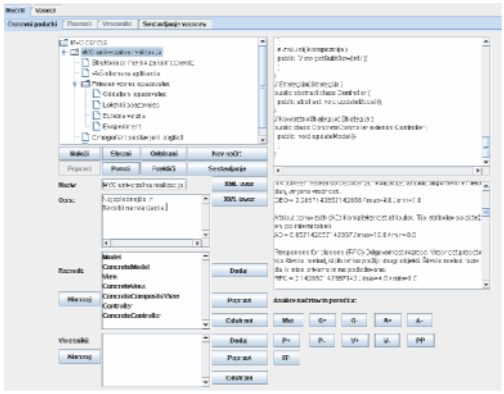
Slika 2. Primerjava različnih kandidatnih strežniških arhitektur za ocenjevalno orodje
Figure 2. Comparision of average evaluation times on various candidate server-architectures for the support tool

Slika 2 prikazuje povprečne odzivne čase, ki so bili potrebni za izvedbo istega metričnega ocenjevanja na različnih strežniških arhitekturah. Prikazane vrednosti nakazujejo pomanjkljivosti posameznih zasnov. Izvedba ocenjevanja se na večnitnem strežniku pri večjem številu zahtev močno poslabša zaradi zahtevnega upravljanja množice niti, pri katerem je kritična operacija zamenjava kontekstov posameznih niti. Učinkovitost dogodkovnega strežnika brez optimizacij se zaradi izvajanja v eni niti eksponentno poslabša že pri majhnem povečanju števila zahtev. Stolpci, ki izstopajo iz vidnega območja grafa, ponazarjajo situacije, ko je povprečna vrednost presegala celoten čas merjenja. Zaradi nenehnega dotoka novih zahtev se preobremeni računalnik v smislu rezervacije virov, kar vodi do popolne blokade izvajanja, saj je operacijski sistem prezaseden z upravljanjem navideznega pomnilnika.



Slika 3. Izbrana arhitektura za podporno orodje
Figure 3. Chosen architecture for the support tool

Zamudno ocenjevanje onemogoča izkoriščanje prednosti dogodkovnega strežnika. Dodatna delavska nit znatno izboljša položaj, vendar se pri večjem številu zahtev ne izkaže. Večanje števila niti na eno-procesorskem računalniku ne vodi do zaželenih rezultatov. Vzpostavitev omrežnega ocenjevanja zagotovi z že majhnim številom vozlišč zadovoljive rezultate. Zgrajeno podporno orodje uporablja arhitekturo, predstavljeno na sliki 3.



Slika 4. Nadzorno okno podpornega orodja, kjer se hrani zgodovina realizacij načrtovalskih vzorcev
Figure 4. Main control window of the support tool containing history of the applied realizations

Osrednji del orodja, ki teče na ločenem strežniku, prevzame zahtevo po ocenjevanju in jo naprej posreduje obdelovalni niti, ki je ne oceni, temveč jo posreduje razpoložljivim vozliščem. Namen je čim učinkoviteje izkoristiti vozlišča, kadar niso obremenjena z drugimi aplikacijami. Upoštevali smo, da postane neko vozlišče

v določenem trenutku zasedeno. Zaradi tega osrednji del orodja razpošlje isto zahtevo več vozliščem. Če posamezno vozlišče konča ocenjevanje, obvesti preostala vozlišča, da je postopek ocenjevanja končan in lahko prekinejo ocenjevanje iste zahteve. Načrtovalec komunicira s podpornim orodjem prek aplikacije, ki se povezuje z osrednjim strežnikom. Slika 4 prikazuje glavno nadzorno okno, prek katerega načrtovalec gradi svoj načrt. Orodje omogoča hranjenje zgodovine dodajanja vzorcev in omogoča načrtovalcu, da se vrača v posamezne faze in eksperimentalno nadaljuje iz njih analizo nove veje realizacij. Rezultate načrtovalec pregleduje s pomočjo poročil, ki jih je mogoče izvoziti v druga orodja.

Slika 5. Ocenjevalni modul za identifikacijo ustreznih realizacij vzorcev s pomočjo podajanja ocenjevalnih meril
Figure 5. Evaluation module for identification of effective pattern realizations

Slika 5 prikazuje ocenjevalni modul podpornega orodja, v katerem načrtovalec podaja želena merila v obliki metričnih intervalov za načrtovalske in specializirane metrike. Po postavitvi želenih intervalov se informacije posredujejo osrednjemu strežniku, ki oblikuje zahtevo, in jo s pomočjo razpoložljivih vozlišč oceni ter nato vrne ustrezen nabor realizacij. Načrtovalec se odloča med pridobljenim naborom in sprejema posamezne realizacije, ki postanejo po potrditvi del modela načrtovanja, ki je prikazan v osrednjem oknu orodja (slika 4).

4 Sklep

Uporaba načrtovalskih vzorcev brez predvidevanja mogočih realizacij in njihovega ocenjevanja je odvisna od izkušenj načrtovalca. Veliko število sestavnih elementov in medsebojnih povezav povzroči, da postane analiza kompleksnejših načrtov in načrtovalskih vzorcev brez računalniške podpore neobvladljiva. Katalogi vzorcev z dopolnilnimi opisi mogočih realizacij v posameznih programskih jezikih omogočajo avtomatizacijo gradnje realizacij, iz katerih načrtovalec s pomočjo metričnih ocen izbere najustreznejše. V članku smo predstavili eno izmed oblik opisovanja dopustnih realizacij, na podlagi katere se lahko predvidi množica realizacij za posamezni vzorec v dani del načrta. Prikazani primer ocenitve in izbire ustrezne realizacije predstavlja možnost za doseganje boljše obvladljivosti pri uporabi načrtovalskih vzorcev. Ocenjevanje in identifikacija ustreznih realizacij zaradi obsega in zahtevnosti nista smiselna brez ustrezne računalniške podpore. Zato je bilo v sklopu raziskave razvito ustrezno podporno orodje, ki omogoči primerno sprotno uporabo opisanega postopka. Uporaba orodja v praksi je nakazala možnosti zagotavljanja kakovostnejših načrtov pri uporabi načrtovalskih vzorcev pri manj izkušenih načrtovalcih.

5 Literatura

[1] C. Alexander, S. Ishikawa, M. Silverstein, *A Pattern Language*, Oxford University Press, 1977.

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns – elements of reusable object-oriented software*, Addison-Wesley, Reading, MA, 1995.

[3] A. H. Eden, *Precise Specification of Design Patterns and Tool Support in Their Application*, Doctoral Thesis, The Department of Computer Science, Tel Aviv University, Israel, 2000.

[4] T. Taibi, D. C. Ngo, Formal Specification of Design Pattern Combination Using BPSL, *Information and Software Technology,* vol. 45, 2003, pp. 157-170.

[5] S. Beloglavec, M. Heričko, M. B. Jurič, I. Rozman, Analysis of the Limitations of Multiple Client Handling in a Java Server Environment, *ACM SIGPLAN Notices*, Vol. 40(4), pp. 20-28, 2005.

[6] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Microsoft Research - Addison Wesley, 2002.

[7] S. Chidamber, C. Kemerer, A Metric Suite for Object-Oriented Design, *IEEE Transactions on Software Engineering*, vol. 20(6), 1994, pp.476-493.

[8] J. Joseph, C. Fellenstein, *Grid Computing*, Chapter 3: The Grid Computing Anatomy, Upper Sadle River, NJ, USA, Prentice Hall, 2003.

[9] M. Welsh, S. D, Gribble, A. E. Brewer, D. A. Culler, Design Framework for Highly Concurrent Systems, *UC Berkley CS Technical Report*, No. UCB/CSD-00-1108, 2000.

Simon Beloglavec je asistent na Fakulteti za elektrotehniko računalništvo in informatiko Univerze v Mariboru. Diplomiral (1998) in doktoriral (2005) je s področja računalništva in informatike. Raziskovalno se ukvarja z objektno orientiranim načrtovanjem komponent, načrtovalskimi vzorci, in programiranjem porazdeljenih omrežnih sistemov v Javi.

Marjan Heričko je izredni profesor na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Magistriral (1993) in doktoriral (1998) je s področja računalništva in informatike. Raziskovalno se ukvarja z vsemi vidiki razvoja informacijskih sistemov s poudarkom na metrikah, procesnih modelih, modeliranju in ponovni uporabi.