OBNAVLJANJE NAČRTOVANJA S POMOČJO VZORCEV NAČRTOVANJA

Tomaž Domajnko, Ivan Rozman, Marjan Heričko
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova ulica 17, 2000 Maribor, Slovenija
e-mail: {tomaz.domajnko, ivan.rozman, marjan.hericko}@uni-mb.si

Izvleček

Vzorci načrtovanja so deležni velikega zanimanja v svetu razvijalcev objektnih sistemov, saj tvorijo pomemben delež visokonivojskih informacij, ki lahko v veliki meri poenostavijo razumevanje in obvladovanje kompleksnih sistemov. Hkrati pa se razvijalci zavedajo, da za večino objektnih sistemov ne obstaja dokumentacija, ki bi vzorce načrtovanja eksplicitno izražala. Zato v prispevku predstavljamo pristop, ki omogoča ekstrahiranje informacij o uporabljenih vzorcih. Pristop temelji na natančni in nedvoumno predstavitvi vzorcev načrtovanja. Na podlagi predstavitve vzorcev načrtovanja v prispevku definiramo postopek identifikacije vzorcev načrtovanja v obstoječih sistemih. Pristop zagotavlja identifikacijo vseh vzorcev načrtovanja, ki so bili uporabljeni ob načrtovanju in implementaciji objektnega sistema. To pa je osnova za ekstrakcijo odločitev, ki smo jih izvedli v aktivnostih arhitekturnega načrtovanja, načrtovanja objektov in implementacije. V prispevku podajamo rezultate analize nekaterih objektnih ogrodij in knjižnic razredov, ki potrjujejo, da je možna zanesljiva identifikacija vzorcev načrtovanja in s tem ekstrakcija pomembnih, visokonivojskih informacij iz obstoječe programske kode.

Abstract

The design patterns present a very important source of high-level information about the understanding and the control of a complex object system. At the same time the development community realizes that the documentation of the current object systems does not include the information about the use of design patterns to the appropriate extent. Therefore in the article we will present the approach to the generation of the information. The approach is based on the rigorous and unambiguous presentation of the design patterns, which will also be presented in the article. The presentation enables a simple and effective implementation of the identification process for any instance of the design pattern, implemented in the object system according to the specification. We present the results of the analysis of a set of object frameworks and object systems confirming the hypothesis that the reliable identification of design pattern instances is both possible and feasible, providing high-level information about the design and implementation decisions.

■■■

Uvod

Vzorci načrtovanja so bili v svet programskega inženirstva vpeljani v delu [Gamma 1995] kot koncepti, ki povečajo razumevanje objektnih sistemov. Vsak vzorec načrtovanja vsebuje izkušnje in znanje strokovnjakov; predstavlja idejo rešitve problema pri načrtovanju sistemov, ki jo lahko enostavno in učinkovito uporabimo v različnih problemskih situacijah. Gledano bolj splošno, lahko vsak vzorec načrtovanja identificiramo kot arhitekturni element na višjem nivoju abstrakcije kot je nivo razredov [Domajnko 1998a]. Eden od osnovnih namenov vzorcev načrtovanja je, da so vzorci načrtovanja načrtovalski elementi, ki poenostavljajo aktivnost načrtovanja objektnih sistemov in dvigajo nivo komunikacije v razvojni skupini [Domajnko 1999a]. K temu dodaja tudi največkrat uporabljena predstavitev vzorcev, ki sestoji iz opisa problema, definicije ideje rešitve, lastnosti in omejitve rešitve, določitve terminologije in povezave na različne že uspešno izvedene rešitve s pomočjo njegove uporabe [Domajnko 1997].

Vendar pa lahko poenostavljeno razumevanje sistemov s pomočjo vzorcev načrtovanja dosežemo le, če dokumentacija objektnega sistema eksplicitno navaja uporabljene vzorce načrtovanja. Eden od razlogov, ki jih avtorji ugotavljajo, je ta, da s pomočjo vzorcev načrtovanja postopke vzdrževanja ali nadgraditve sistemov izvajamo na višjem nivoju abstrakcije, kar kažejo tudi rezultati v [Domajnko 1999b]. Dejansko pa

danes velika večina objektnih sistemov takšne dokumentacije ne vsebuje. Zato bi bilo zelo koristno, če bi lahko v obstoječi programski kodi identificirali primerke vzorcev načrtovanja.

Zato bomo v prispevku prikazali predstavitev vzorcev načrtovanja, ki omogoča razvoj podpornega okolja, ki lahko identificira vsak uporabljeni vzorec načrtovanja, implementiran skladno z definicijo vzorca načrtovanja. S tem pa tudi možnost, da dopolnimo dokumentacijo objektnih sistemov.

Predstavitev vzorcev načrtovanja

Na podlagi definicije vzorca načrtovanja iz [Gamma 1995]: »*Vzorec je ideja rešitve, ki se je izkazala uporabna v določenem kontekstu in bi bila najverjetneje koristna tudi v drugih kontekstih. Vzorec predstavlja večkratne odločitve strokovnjaka, ki sicer vodijo do različnih rešitev, a vsebujejo določen nivo kakovosti*« si poglejmo, kako so vzorci načrtovanja največkrat predstavljeni v literaturi [Gamma 1995, Buschmann 1996, Domajnko 1998b, Domajnko 1998c]. Tabela 1 prikazuje opisno predstavitev.

Tabela 1 Opisna predstavitev vzorcev

Sekcija	Pomen sekcije
Ime	Identifikator vzorca
Namen	Definicija problema, ki ga vzorec rešuje in rezultati uporabe vzorca
Uporabnost	Opis situacije, v kateri lahko vzorec uporabimo
Struktura	Praviloma razredni diagram, ki predstavlja statično sliko vzorca
Udeleženci	Seznam razredov, ki so vključeni v uporabo vzorca
Sodelovanja	Tekstoven opis dinamične komponente vzorca
Učinek	Podaja stanje ali konfiguracijo sistema po uporabi vzorca
Implementacija	Napotki za implementacijo vzorca
Pravila uporabe	Utemeljitev korakov uporabe vzorca
Sorodni vzorci	Opis statičnih in dinamičnih povezav med vzorci
Primeri uporabe	Opis uporabe vzorcev v obstoječih sistemih

Jasno je, da opisna predstavitev vzorcev ni primerna za računalniško podporo. To je vodilo k raziskavi razvoja jezika za opis vzorcev načrtovanja, ki bi dopolnil opis vzorcev načrtovanja in omogočil razvoj računalniške podpore. Jezik za specifikacijo vzorcev načrtovanja (poimenovali smo ga PatL) je bil zgrajen na podlagi opazovanj elementov, ki tvorijo vodilno misel kateregakoli vzorca načrtovanja. Jezik temelji na predstavitvi elementov modela abstraktne sintakse (model AST), s katero abstrahiramo implementacijske podrobnosti posameznih objektnih programskih jezikov.

Na podlagi definiranega modela AST in množice tipov udeležencev $\Re \equiv \{T_1, T_2 \ldots T_n\}$, množico relacij $\Re \equiv \{R_1, R_2 \ldots R_n\}$ in funkcijo M, ki preslika konstrukte modela AST v T in izraze AST v $\Re$, definiramo M(p) kot **model programa p v modelu AST**. Množico osnovnih konstruktov jezika PatL (element e) sestavljajo razredi (razred c) in metode (metoda m)[1]. Osnovne konstrukte združujemo v množice elementov enakega tipa - *enolične množice*. Med osnovnimi konstrukti so definirane *osnovne relacije*. Osnovne relacije smo definirali na podlagi opazovanja katalogov [Gamma 1995] in [Grand 1998]. Edina omejitev osnovnih relacij je, da morajo imeti kanonično implementacijo v več kot le enem objektnem jeziku. Tabela 2 podaja seznam osnovnih relacij med elementi, gradniki jezika PatL.

Tabela 2 Seznam osnovnih relacij

Osnovna relacija	Pomen relacije
razred (c)	delna, unarna relacija, ki izraža da je element *c* razred.
dedovanje (c_1, c_2)	relacija, ki izraža dedovanje med nadrazredom c_1 in podrazredom c_2.
metoda (m)	delna, unarna relacija, ki izraža da je element *m* metoda.
abstrakt (e)	relacija, ki določa, da je element *e* abstrakten.
definiran V (e, c)	relacija, ki izraža da je element *e* definiran v razredu *c*.
return Tip (m, c)	relacija, ki izraža da metoda *m* vrača objekte primerke razreda *c*.
konstruktor (m, c)	relacija, določa, da je metoda *m* konstruktor razreda *c*.
enakoime (m_1, m_2)	relacija, ki izraža, da sta imeni m_1 in m_2 enaki.
število argumentov (m)=n	relacija, ki definirana da ima metoda *m n* argumentov.
argument (n, m)=c	funkcijska relacija, ki za metodo *m* preslikuje zaporedno številko argumenta v tip argumenta.
proženje (m_1, m_2)	relacija, ki izraža, da v telesu metode m_1 (brez upoštevanja kontrolnih struktur) prihaja do proženja metode m_2.
enojnareferenca (c_1, c_2)	relacija, ki izraža, da obstaja objektna referenca v razredu c_1 do razreda razreda c_2 brez ločevanja med kompozicijsko, agregacijsko ali asociacijsko referenco.
multireferenca (c_1, c_2)	relacija, ki izraža, da razred c_1 v svoji definiciji vsebuje večštevno referenco do razreda c_2 brez ločevanja med kompozicijsko, agregacijsko ali asociacijsko referenco.
prirejanje (m, c_1, c_2)	relacija, ki izraža, da metoda *m* priredi referenco razreda c_1 do razreda c_2.
prirejanje po (m_1, m_2, c_1, c_2)	relacija, ki izraža, da metoda m_1 po zaključku izvajanja metode m_2 priredi referenco razreda c_1 do razreda c_2.

1 *Množico osnovnih konstruktov označimo s simbolom E, množico razredov s simbolom C, množico metod pa s simbolom* M.

Tabela 3 Seznam izpeljanih relacij

Izpeljana relacija	Definicija relacije
enak podpis(m_1, m_2)	relacija, ki izraža da imata metoda m_1 in m_2 enak podpis. $enakpodpis(m_1,m_2) \Leftrightarrow enakoime(m_1,m_2) \wedge stargumentov(m_1) = \text{stargumentov}(m_2) \wedge \prod_{i=0}^{i = stargumentov(m_1)} [\text{argument}(i,m_1) == \text{argument}(i,m_2)]$
posredovanje(m_1, m_2)	relacija, ki izraža da metoda m_1 proži metodo m_2, pri tem pa uporabi svoje argumente brez sprememb kot parametre metode m_2. $posredovanje(m_1,m_2) \Leftrightarrow proženje(m_1,m_2) \wedge enakpodpis(m_1,m_2)$
kreiranje(m_1, m_2)	relacija, ki izraža klic konstruktorske metode m_2 s strani metode m_1. $kreiranje(m_1,m_2) \Leftrightarrow proženje(m_1,m_2) \wedge konstruktor(m_2)$
produkcija(m_1, m_2, c)	relacija, ki izraža da metoda m_1 kliče metodo m_2, ta pa je konstruktor nekega razreda. Hkrati metoda m_1 tudi vrne kreiran objekt. $produkcija(m_1,m_2,c) \Leftrightarrow kreiranje(m_1,m_2) \wedge returntip(m_1,c)$

Na podlagi osnovnih relacij lahko definiramo množico izpeljanih relacij, ki imajo večjo izrazno moč, uporabljene osnovne relacije pa zagotavljajo njihovo implementacijo. Tabela 3 podaja seznam izpeljanih relacij in njihovih definicij.

Tabela 4 podaja seznam posplošitev osnovnih in izpeljanih relacij, ki dajejo matematično podlagi predstavitve vzorcev načrtovanja s pomočjo jezika PatL.

V modelu programa o imenujemo urejeno zaporedje udeležencev $\omega_1 \ldots \omega_N$ v p **kontekst**. S pomočjo kontekstov lahko opišemo strukturo poljubnega programa, zapisanega s pomočjo modela AST. Tudi opisna predstavitev vzorcev načrtovanja predstavlja vodilno misel vzorca načrtovanja (osnovno idejo) s pomočjo *udeležencev* (razredi, metode) in njihovega *sodelovanja*.

Če kontekst $\underline{\omega}$ *implementira* vzorec načrtovanja, bomo v kontekstu $\underline{\omega}$ zasledili elemente, ki implementirajo udeležence in sodelovanja, ki zagotavljajo implementacijo rešitve vzorca načrtovanja. Tako lahko definiramo, da je kontekst $\underline{\omega}$ posplošena oblika *primerka vzorca* π v programu p natanko takrat, če ima vsak $\omega_1 \ldots \omega_N$ iz $\underline{\omega}$ ustrezen tip (določen v π) in da $\omega_1 \ldots \omega_N$ sodelujejo, kot je določeno v specifikaciji vzorca π. Tako lahko vodilno misel (osnovno idejo) vzorca načrtovanja zapišemo v obliki *sheme*, kot jo definira [Computer 1990]. Shema predstavlja popolno in dokončno predstavitev vodilne misli posameznega vzorca in določa *udeležence* in *sodelovanja*. Skladno s to definicijo shema sestoji iz deklaracije *spremenljivk* (predstavljajo udeležence) in množice *predikatov*, ki izražajo relacije med spremenljivkami (izražajo sodelovanje).

V formulah uporabljamo relacije med strogo tipiziranimi spremenljivkami naslednjih tipov:

- Elementarne spremenljivke, ki predstavljajo osnovne konstrukte (razrede in metode).
- Spremenljivke višjih dimenzij, ki predstavljajo množice.
- Spremenljivke hierarhij, ki predstavljajo množico razredov v neki relaciji.

Slika 1 prikazuje dve alternativi prikaza predstavitvene sheme vzorcev načrtovanja (s pomočjo grafične sheme ali formule). V vsaki predstavitvi sheme najprej navedemo ime sheme (predstavlja

Tabela 4. Posplošitve osnovnih in izpeljanih relacij

Vrsta relacije	Definicija relacije
unarna relacija	relacija, definirana nad enim samim elementom v obliki parcialne funkcije. $r(X) \Leftrightarrow \forall x \in X : r(x)$
tranzitivna relacija	tranzitivna relacija je tranzitivno zaprtje osnovne relacije. $r^+(x,y) \Leftrightarrow r(x,y) \vee \exists z : \left[r^+(x,z) \wedge r(z,y)\right]$
totalna relacija	relacija med osnovnimi elementi ali množicami, ki prevzema obliko totalne funkcije. $r_T(X,Y) \Leftrightarrow \forall x \in X\ \exists y \in Y : r(x,y)$
popolna relacija	relacija, ki prevzema obliko izomorfne preslikave. $r_I(X,Y) \Leftrightarrow \forall x \in X\ \exists y \in Y : r(x,y) \wedge \forall y \in Y\ \exists x \in X : r(x,y)$
relacija med množicami	relacija med množicami se prevede na relacijo med elementi množic. $r(X,Y) \Leftrightarrow \forall x \in X\ \exists y \in Y : r(x,y)$ $r(x,Y) \Leftrightarrow r(\{x\},Y);\quad r(X,y) \Leftrightarrow r(X,\{y\})$
enakost relacij	enakost relacij določa, da sta relaciji r_1 in r_2 v množici S izomorfni. $enakost_{r_1,r_2}(S) \Leftrightarrow \forall p,q \in S : \left[r_1(p,q) \leftrightarrow r_2(p,q)\right]$ $enakost_{r_1,r_2}(X,Y) \Leftrightarrow \forall x \in X\ \forall y \in Y : \left[r_1(x,y) \leftrightarrow r_2(x,y)\right]$
hierarhija	relacija, ki določa množico razredov v hierarhiji dedovanja. $\text{hierarhija(h)} \Leftrightarrow \left[\begin{array}{l}\forall c_N \in h\ \exists c_0 \in h : \\ \quad \text{abstrakt}(c_0) \wedge \\ \quad \text{dedovanje}^+(c_N, c_0)\end{array}\right]$ **Relacijo *r*** nad hierarhijo $h_i \in H$ če r ni popolna relacija, definiramo kot: $r(h_i,S) \Leftrightarrow r(koren(h_i),S)$ $r(S,h_i) \Leftrightarrow r(S,listi(H_i))$ $r(h_1,h_2) \Leftrightarrow r(listi(h_1),koren(h_2))$ **Relacijo r** nad hierarhijo $h_i \in H$, kjer je je *r* popolna relacija definiramo kot: $r_P(h_i,S) \Leftrightarrow r_P(listi(h_i) \cup koren(h_i),S)$ $r_P(S,h_i) \Leftrightarrow r_P(S,listi(h_i) \cup koren(h_i))$ $r_P(h_1,h_2) \Leftrightarrow r_P(listi(h_1) \cup koren(h_1),listi(h_2) \cup koren(h_2))$
družina	je enolična množica metod z enakim podpisom, ki so definirane v množici razredov *C*. $družina(M,C) \equiv M_C^d \subseteq \left\{\begin{array}{l}M_i \mid \exists M_j : M_i \subseteq M \wedge M_j \subseteq M \\ enakvmesnik(M_i,M_j) \wedge \\ definiranv(C,M_i) \wedge definiranv(C,M_j)\end{array}\right\}$
rod	je enolična množica družin. Množica množic metod $\{F_i\}$ je *rod* v množici razredov C, tedaj in le tedaj, če je vsaka metoda $F \in \{F_i\}$ družina v podmnožici. $C\,M_C^r \Leftrightarrow \left\{M_{X_i}^d : X_i \subseteq C\right\}$

Shema

$\exists\ (x_1, x_2, \ldots x_n)$ spremenljivke

$\prod_i \Re_j(y_{i1}, y_{i2}, \ldots y_{i_n})$ predikati, povezani z operatorjem konjunkcije

$\text{shema} \equiv (x_1, x_2 \ldots x_n) \mapsto \prod_i R_i(x_1, x_2 \ldots x_m)$

Slika 1 Dve alternativni predstavitvi sheme vzorcev načrtovanja

enolično ime vzorca načrtovanja), nato navedemo seznam spremenljivk (določimo udeležence in pri tem upoštevamo strogo tipiziranost) in na koncu še množico predikatov (definiramo sodelovanje udeležencev).

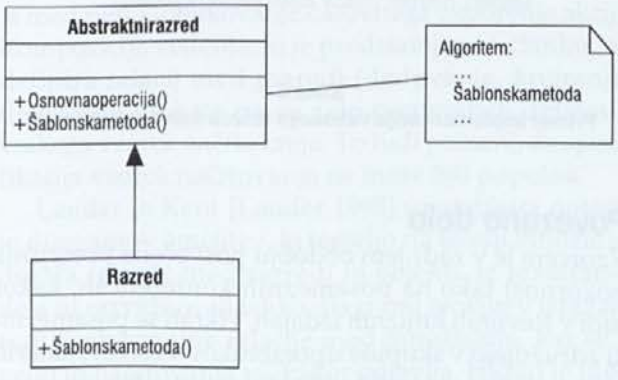
Nad shemami, ki specificirajo posamezne vzorce načrtovanja, definiramo operacije. Operacije predstavljajo uporabo in posplošitev operacij iz [IEEE 1990/5].

Tabela 5 Seznam operacij nad shemami

Operacija	Pomen operacije nad shemo
Preimenovanje komponent	zagotavlja oblikovanje nove sheme na podlagi obstoječe s pomočjo sistematičnega preimenovanja komponent. *Novashema/Osnovna [(Novakomponenta/Starakomponenta)*]*
Dekoracija sheme	zagotavlja ločevanje med spremenljivkami in predikati pred in po izvedbi določene akcije. *komponenta* - označuje komponento pred izvedbo akcije. *komponenta'* - označuje dekorirano komponento.
Vključitev sheme	zagotavlja možnost ponovne uporabe posameznih shem v kontekstu deklaracij drugih shem. Vključitev sheme u v shemo y definiramo kot[1]: $\psi \equiv (\phi_1, \phi_2 ... \phi_N) \mapsto \prod_i R_i(\phi_A ... \phi_B)$ $\upsilon \equiv (\alpha_1, \alpha_2 ... \alpha_M) \mapsto \prod_k R_k(\alpha_U ... \alpha_V)$ $\psi \oplus \upsilon \Leftrightarrow (\alpha_1, \alpha_2 .. \alpha_M, \phi_1, \phi_2 ... \phi_N) \mapsto \prod_k R_k(\alpha_U ... \alpha_V) \wedge \prod_i R_i(\phi_A ... \phi_B)$
Konjunkcija sheme	konjunkcijo dveh shem formiramo z združitvijo njunih signatur, identificiranjem skupnih spremenljivk (katerih tipi se morajo ujemati) in združitvijo predikatov. Shema ≡ ShemaA ∧ ShemaB
Disjunkcija sheme	disjunkcijo shem formiramo z združitvijo njunih signatur: identificiranjem skupnih spremenljivk (tipi se morajo ujemati) in razdruženjem njunih predikatov. Shema ≡ ShemaA ∨ ShemaB
Negacija sheme	negacijo sheme formiramo tako, da ne spreminjamo identificiranih spremenljivk, negiramo pa vse predike. Shema ≡ ¬ ShemaA
Skrivanje sheme	predstavlja mehanizem jezika shem, ki poenostavlja specifikacije. ShemaA \ (komponenta)*
Kompozicija shem	označuje relacijsko kompozicijo dveh shem. Obe shemi morata zadostiti pogojem spremenljivk v stanju po operaciji. Shema ≡ ShemaA • ShemaB

Tabela 5 prikazuje seznam operacij nad shemami.

S pomočjo zgrajene osnove jezika PatL si sedaj poglejmo primera definicij poznanih vzorcev načrtovanja. Vedenjski vzorec načrtovanja ***Šablonskametoda*** definira ogrodje algoritma v operaciji z odlaganjem nekaterih korakov v podrazrede. Vzorec dovoljuje podrazredom ponovno definirati nekatere korake algoritma, brez da bi spreminjali strukturo algoritma.



Slika 2 Razredni diagram vzorca Šablonskametoda

Slika 2 prikazuje razredni diagram vzorca. Največkrat se vzorec uporabi pri izgradnji fleksibilnih algoritmov, kjer je s pomočjo drugačne implementacije primitivnih metod moč spremeniti potek algoritma.

Slika 3 prikazuje formalno predstavitev vzorca s pomočjo jezika PatL. Osnovna razlika so dodane informacije:

- proženje metode, ki je bilo prej izraženo s pomočjo naravnega jezika, temelji sedaj na relaciji, ki ima kanonično predstavitev v objektnih jezikih,
- relacija dedovanja med razredom Abstraktnirazred in razredom Razred ni eksplicitno izražena, ker ni nujno potrebna (vpeljava koncepta vmesnika),
- proženje šablonske metode s strani osnovne operacije ni nujno neposredno, ampak je lahko posredno preko kateregakoli tranzitivnega zaprtja relacije,

2 *Vključitev sheme vsebuje še pogoj, da pri vključevanju v shemah ne obstajajo enako poimenovane komponente. V primeru, da pogoj ni izpolnjen, je potrebno komponente preimenovati.*

- definicija vzorca dopušča tudi predifiniranje osnovnih operacij, saj le te po definiciji tvorijo družino znotraj hierarhije razredov,
- razred Razred je element hierarhije razredov, njegov položaj pa je popolnoma nepomemben.
- definicija vzorca dopušča tudi definicijo več kot le ene šablonske metode in več kot le eno implementacijo le te znotraj hierarhije razredov.

Vzorec: Šablonskametoda

Šablonskametoda $\in$ M

Osnovnaoperacija $\in \{M_H^d\}$,

Razred $\in$ H

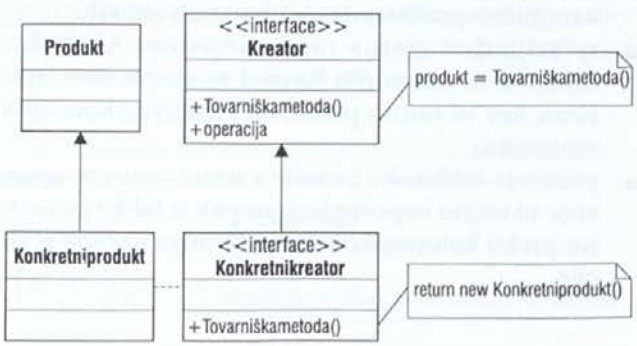
družina(Osnovnaoperacija, Razred) $\wedge$
proženje$_T$(Šablonskametoda, Osnovnaoperacija) $\wedge$
definiranv$_T$(Šablonskametoda, Razred)

Slika 3 Predstavitev vzorca Šablonskametoda v jeziku PatL

Na podlagi podanih ugotovitev lahko zaključimo, da nove informacije bolj natančno specificirajo osnovno vodilo vzorca načrtovanja (oziroma, dopolnjujejo opisno definicijo vzorca načrtovanja).

Kreatorski vzorec *Tovarniškametoda* definira vmesnik za kreiranje objekta, vendar dopušča podrazredom, da določijo, iz katerega razreda bodo izhajali. Vzorec dovoljuje, da razred odloži instanciranje na podrazrede. Slika 4 prikazuje razredni diagram vzorca, kjer lahko opazimo, da s pomočjo naravnega jezika definiramo dinamično komponento vzorca.

Slika 5 prikazuje formalno definicijo vzorca, kjer smo poleg strukture razredov (sedaj povezanih s hierarhijo in ne v neposredno dedovanje) določili tudi dinamično komponento ideje vzorca. S tem smo dodali informacije, ki lahko v veliki meri vplivajo na uporabnost samega vzorca pri rutinski uporabi.



Slika 4 Razredni diagram vzorca Tovarniškametoda

Vzorec: Tovarniškametoda

Tovarniškametoda $\{M_{H_1}^d\}$,

Kreatorji $\in H_1$,

Produkti $\in H_2$

družina(Tovarniškametoda, *Kreatorji*) $\wedge$
produkcija$_t$(Tovarniškametoda, *Produkti*) $\wedge$
returnTip$_t$(Tovarniškametoda, *Produkti*) $\wedge$
enakost $_{returnTip\ produkcija}$(Tovarniškametoda, *Produkti*)

Slika 5 Razredni diagram vzorca Tovarniškametoda

V nadaljevanju si poglejmo še enostavnost implementacije validacije vzorca. Slika 6 prikazuje implementacijo validacije vzorca Šablonskametoda, ki smo ga prej formalno definirali. Kot je moč videti iz primera, je implementacija validacije popolnoma skladna z definicijo vzorca. To kaže tudi na dejstvo, da je moč generiranje pravil za validacijo avtomatizirati s pomočjo enostavnega orodja, ki formalno specifikacijo pretvori v sintakso implementacijskega okolja (v našem primeru ekspertne lupine CLIPS in razširka JESS), samo okolje po z vgrajenimi algoritmi poskrbi za izvajanje validacije. V primeru na sliki sicer v bazo dejstev samo dodamo dejstvo, da smo našli primerek vzorca Šablonskametoda, podporno okolje pa mora dodati informacije, ki so potrebne za popolno validacijo vzorca (vsaj udeležence in njihove vloge).

```
(defrule validirajŠablonskametoda
        (razred (naziv ?razred))
        (metoda (naziv ?Šablonskametoda))
        (metoda (naziv ?Osnovnaoperacija))
        (definiranV (razred (naziv ?razred) (metoda (naziv
?Šablonskametoda)))
        (družina (metoda (naziv ?Šablonskametoda) (razred
(naziv ?razred))))
        (proženje (metoda (naziv ?Osnovnaoperacija)) (meto-
da (naziv ?Šablonskametoda)))
 =>
        (assert (vzorec (naziv Šablonskametoda)))
)
```

Slika 6
Primer implementacije validacije vzorca Šablonskametoda

Povezano delo

Vzorcem je v zadnjem obdobju posvečena precejšnja pozornost tako na posameznih konferencah, kakor tudi v številnih knjižnih izdajah. Hkrati se posamezniki združujejo v skupine uporabnikov vzorcev, katerih primarni cilj je identifikacija in zapis novih vzorcev in nadzor nad obstoječimi vzorci.

Vzorci programskega inženirstva, še posebej *vzorci načrtovanja*, so največkrat objavljeni v obliki *katalogov*, ki predstavljajo delo, v katerem so posamezni elementi neodvisni in nepovezani. Serija knjig PloPD (Pattern Languages of Program Design) predstavljajo zbirko vzorcev različnih problemskih domen in formatov, povzetih po PloP (Pattern Languages of Programming) konferencah [PloP 1995, PloP 1996, PloP 1997, PloP 1998]. Prispevke različnih avtorjev lahko razvrstimo v različne domene, kot so *vzorci načrtovanja, specializacija obstoječih vzorcev, domensko specifični vzorci, organizacijski* in *upravljalski vzorci* ter *jezikovno odvisni vzorci*.

Trenutno stanje je tako, da obstaja v svetu cela vrsta predlaganih predstavitev vzorcev v opisni obliki, kjer avtorji vzorce opisujejo z vnaprej določeno množico lastnosti, ki onemogočajo formaliziranje uporabe vzorcev. Enega prvih poskusov formalizacije predstavitev vzorcev predstavlja model LayOM [Bosch 1996], ki s pomočjo razširitve klasičnega objektnega modela s konceptoma *stanje* (state), *nivo* (layer) zagotavlja predstavitev vzorcev načrtovanja. Izraznost tako nastalega specifikacijskega jezika je prikazana z določanjem omejitev reakcij objektov in protokola sodelovanja objektov. Vendar pa tak pristop omejuje pravila na posamezne objekte, kar zagotavlja podporo omejenemu številu primerov, kjer ni potreben nadzor na višjem nivoju. Pristop zato ni primeren kot osnova za abstrahiranje načrtovalskih odločitev.

V [Ducasse 1995] avtorji definirajo nivo abstraktnega pošiljanja sporočil med objekti, kjer je nivo v obliki konektorjev sposoben prestreganja sporočil enega objekta drugemu in s tem zagotavljati neposredno specifikacijo in implementacijo *"izvedljivih konektorjev"*. Avtorji predstavijo tudi pristop k specifikaciji vzorcev načrtovanja, vendar sami podajajo omejitve pristopa, ki je primerna samo za specifikacijo dinamičnih lastnosti vzorcev, medtem ko strukurnih lastnosti ni moč določiti.

Helm, Holland in Gangopadhyay v [Helm 90] definirajo pojem *pogodba*. Pogodba predstavlja razširitev predikatne logike prvega reda z možnostjo predstavitve funkcijskih klicev, prirejanj in relacij urejanja med njimi (oblikovanje časovnega zaporedja akcij). Kompozicija vedenja, ki je predstavljena v članku, ne definira relacij med razredi (dedovanje, kreiranje, delegacija itd.), te pa so zelo pomemben dejavnik vsakega vzorca načrtovanja. To tudi pomeni, da specifikacija vzorca načrtovanja ne more biti popolna.

Lauder in Kent [Lauder 1998] uporabljata notacijo diagramov omejitev, ki temeljo na teoriji množic in določa relacije med razredi in objekti. Iz rezultatov dela ni razvidno, ali lahko diagrami omejitev izražajo tudi funkcionalne relacije med množicami, kar se v vzorcih načrtovanja vsekakor pojavlja. Hkrati je nivo abstrakcije diagramov omejitev prenizek (oziroma, diagrami so preveč podrobni) za specifikacijo zelo kompleksnih vzorcev načrtovanja, oziroma za iskanje načrtovalskih odločitev.

Budinsky, Finnie in Yu v [Budinsky 1996] predstavljajo orodje, ki podpira aplikacijo vzorcev načrtovanja in generiranje izvorne kode. Razvijalec določa postopek generacije izvorne kode s posebnim skriptnim jezikom COGENT, vendar orodje ne podpira poti nazaj – iz izvorne kode ni moč ugotoviti, kateri vzorci so bili uporabljeni. Precej podobno Quintessoft Inc v [Quintessoft 1997] opisuje orodje CASE, ki je sposobno generirati izvorno kodo v jeziku C++. Enako kot v prejšnjem primeru orodje generira izvorno kodo vzorca načrtovanja, ki je ločena od ostalega sistema, povratnih informacij pa orodj ni možno oblikovati iz obstoječe programske kode.

Kljub temu, da so danes vzorci načrtovanja konstrukti, ki so že nekaj let poznani razvijalcem programskih sistemov, pa so raziskave na področju identifikacije vzorcev načrtovanja v obstoječih sistemih precej redke. Kljub številnim raziskavam na področju povratnega inženirstva pa je namen raziskave drugačen. V raziskavi namreč ne gre za podrobno razumevanje implementacije objektnega sistema, ampak le za ekstrakcijo visokonivojskih informacij iz obstoječe kode.

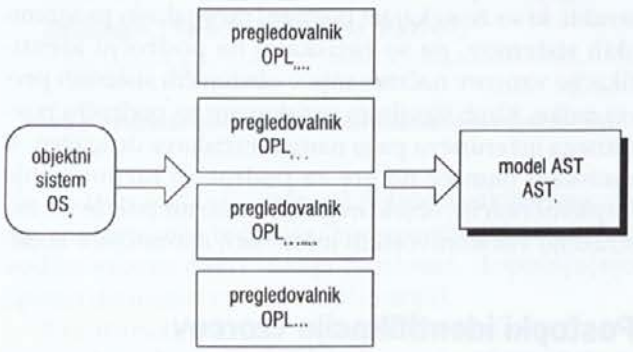
Postopki identifikacije vzorcev načrtovanja

Model predstavitve vzorcev načrtovanja zagotavlja enolično in nedvoumno predstavitev vzorcev. S pomočjo modela abstraktne sintakse (AST) lahko zagotovimo neodvisnost od programskega jezika (vse dokler lahko s pomočjo pregledovalnika izvorne kode generiramo ustrezno predstavitev objektnega sistema s pomočjo modela AST). S pomočjo jezika PatL lahko vzorce načrtovanja rigorozno definiramo. Na podlagi ugotovitev lahko definiramo naslednje aktivnosti:

- **Aplikacija vzorca** je proces, kjer je vzorec π apliciramo v kontekst $\underline{\omega}$ in konkretni elementi $\omega_1 \ldots \omega_N$ prevzamejo vloge sodelujočih iz specifikacije vzorca p. To pomeni, da se posamezni elementi spremenijo na tak način, da zadostijo zahtevam specifikacije vzorca π. Največkrat aplikacije vzorca ne izvajamo v neinicializiranem okolju temveč v določenem kontekstu. To pomeni, da prilagajamo in dopolnjujemo konstrukte, ki že obstajajo v kontekstu $\underline{\omega}$. V danem kontekstu $\underline{\omega}$ bomo večkrat identificirali, da posamezni sodelujoči specifikacije vzorca nimajo dejanskega elementa. Takrat pravimo, da je $\underline{\omega}$ *nepopolni kontekst* in implicira, da je potrebno manjkajoče vloge specifikacije vzorca kreirati. V takem primeru aplikacija vzorca rezultira v novih konstruktih, ki zadoščajo pogojem, podanih v specifikaciji vzorca načrtovanja.

- **Validacija vzorca** je proces, ki za podan kontekst ω v programu p in podan vzorec π odgovori na vprašanje ali je ω primerek vzorca π.
- **Identifikacija vzorca** je proces, ki za podan kontekst ω v programu p poda seznam validacij vzorcev π_i v kontekstu ω.
- **Identifikacija relacij med vzorci** je aktivnost, s pomočjo katere ugotavljamo binarne relacije med vzorci. Relacije praviloma klasificirajo vzorce.

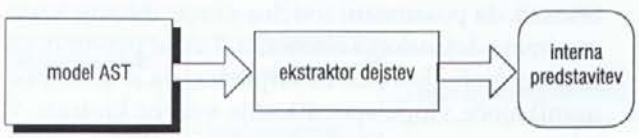
Slika 7 prikazuje prvi korak pri identifikaciji vzorcev načrtovanja, ki so bili uporabljeni pri načrtovanju (in implementaciji) objektnega sistema. S pomočjo razmeroma enostavnega pregledovalnika izvorne kode generiramo modela programa z uporabo modela AST.



Slika 7 Proces kreiranja modela objektnega sistema

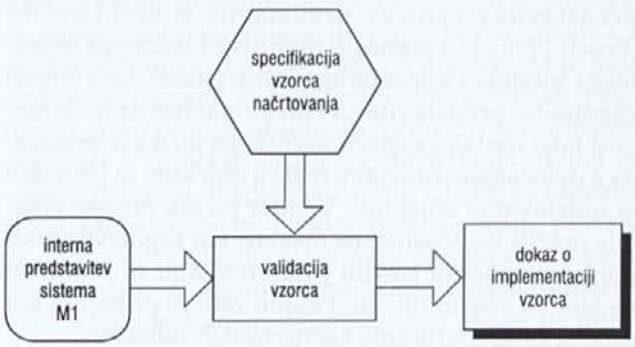
Definirani model AST lahko uporabimo za predstavitev poljubnega objektnega sistema, implementiranega v objektnem programskem jeziku, za katerega smo razvili pregledovalnik, ki ekstrahira primerne informacije. Objektni sistem, predstavljen s pomočjo modela AST v naslednjem koraku s pomočjo ekstraktorja dejstev pretvorimo v interno predstavitev, ki je skladna z jezikom PatL (interna predstavitev je odvisna od izbranega okolja, v naši raziskavi smo uporabili ekspertno lupino CLIPS, ki temelji na ogrodjih in njeno Java razširitev JESS). Zato ekstraktor dejstev na podlagi modela programa in definiranih relacij oblikuje bazo znanja v ekspertni lupini.

V tem trenutku lahko izvedemo validacijo vzorca načrtovanja. S pomočjo formalne specifikacije vzorcev



Slika 8 Proces oblikovanja interne predstavitve

in uporabe interne predstavitve lahko proces validacije izvedemo na preprost način s pomočjo enostavnih pravil, katere implementiramo v ekspertni lupini (definicija vsakega izmed vzorcev predstavlja iskalni kriterij, ki mu ekspertna lupina s pomočjo vgrajenih algoritmov poskuša zadostiti. Slika 9 prikazuje osnovo procesa. Proces validacije vzorca je deterministične in voden s principi boolove logike – ali vzorec je implementiran v objektnem sistemu ali pa ni. To je tudi edina omejitev pristopa, saj tak pristop ne zagotavlja identifikacije vzorcev, ki so skoraj v popolnosti implementirani v programski kodi. Če obstaja razlika med specifikacijo in dejansko implementacijo vzorca, proces identifikacije takšne implementacije vzorcev ne razpozna.



Slika 9 Proces validacije vzorca načrtovanja

Proces identifikacije vzorcev v objektnem sistemu predstavlja le uporabo validacije posameznega vzorca v objektnem sistemu. Prikazuje proces, kjer iz interne predstavitve poljubnega objektnega sistema pridobimo informacije o vzorcih, ki so implementirani v sistemu.

Relacije med vzorci načrtovanja

Kot smo zapisali, je klasifikacija vzorcev eden od predpogojev, da bomo tudi v času, ko bo število vzorcev načrtovanja narastlo, lahko s pridom in učinkovito izkoriščali vgrajeno strokovno znanje. S pomočjo formalne definicije vzorcev postane proces identifikacije relacij med vzorci enostaven in deterministične.

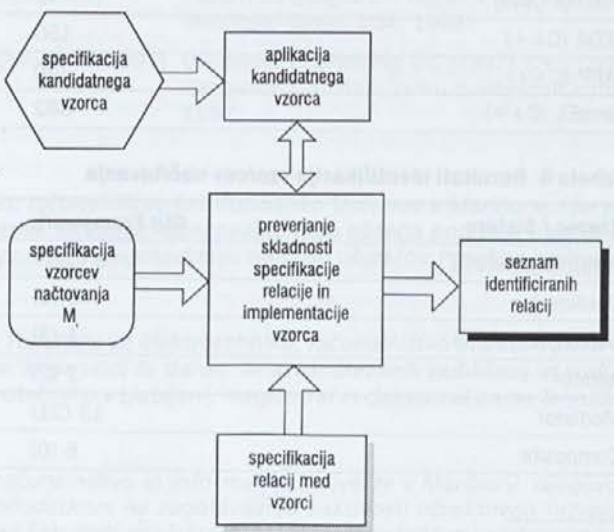
Enako kot smo definirali posamezne vzorce načrtovanja, lahko definiramo tudi množico relacij med vzorci in z enakim postopkom, kot smo izvajali validacijo vzorca načrtovanja v objektnem sistemu, lahko izvedemo tudi primerjavo vzorcev načrtovanja med seboj. Tabela 6 prikazuje definicijo nekaterih najpomembnejših relacij med vzorci načrtovanja.

Tabela 6 Opisna predstavitev vzorcev

Relacija med vzorci	Pomen relacije
specializira(π,σ)	izraža, da vzorec π specializira vzorec σ. $\pi \equiv (\pi_1, \pi_2 ... \pi_N) \mapsto \prod_j R_j(\pi_{A_j} ... \pi_{B_j})$ $\sigma \equiv (\sigma_1, \sigma_2 ... \sigma_N) \mapsto \prod_i R_i(\phi_{A_i} ... \phi_{B_i})$ $specializira(\pi, \sigma) \Leftrightarrow$ $\exists \zeta : \zeta / \sigma[\sigma_1 / \pi_1, \sigma_2 / \pi_2 ... \sigma_N / \pi_N] \wedge \{R_\zeta\} \subseteq \{R_\pi\}$
uporablja (π, σ)	izraža, da vzorec π uporablja vzorec σ. $\pi \equiv (\pi_1, \pi_2 ... \phi_N) \mapsto \prod_j R_{\pi j}(\pi_{A_j} ... \pi_{B_j})$ $\sigma \equiv (\sigma_1, \sigma_2 ... \sigma_M) \mapsto \prod_i R_{\sigma i}(\phi_{U_i} ... \phi_{V_i})$ $uporablja(\pi, \sigma) \Leftrightarrow \forall R_{\sigma i}(\sigma_{U_i} ... \sigma_{V_i})$ $\exists \zeta_{\sigma i} : \zeta_{\sigma i} / \sigma[(\sigma_{O_i} / \pi_{P_i})*] \wedge$ $R_{\zeta i}(\pi_{U_i}, \pi_{V_i}) \equiv R_{\pi k}(\pi_{U_i}, \pi_{U_i})$
specializiranv (π, σ)	izraža, da je specifikacija vzorca π specializirana v specifikaciji vzorca σ. $specializiranv(\pi, \sigma) \Leftrightarrow specializacija(\sigma, \pi)$
nivo razlikovanja $(\pi, \sigma) \mapsto \mathbf{N}\{\cup\}$	je parcialna funkcija, ki za par vzorcev <π, σ> vrne razdaljo razlikovanja. $\pi \equiv (\pi_1, \pi_2 ... \phi_N) \mapsto \prod_j R_{\pi j}(\pi_{A_j} ... \pi_{B_j})$ $\sigma \equiv (\sigma_1, \sigma_2 ... \sigma_M) \mapsto \prod_i R_{\sigma i}(\phi_{U_i} ... \phi_{V_i})$ $nivorazlikovanja(\pi, \sigma) \equiv$ $\max\left(\left\|\left\{R_{\zeta / \sigma[(\pi_M / \sigma_N)*]}\right\} - \{R_\pi\}\right\|, \left\|\{R_\pi\} - \left\{R_{\zeta / \sigma[(\pi_M / \sigma_N)*]}\right\}\right\|\right)$

Slika 10 prikazuje proces ugotavljanja relacij med vzorci. Proces izvedemo s pomočjo aktivnosti aplikacije (uporabe vzorca načrtovanja) v prazen kontekst in aktivnosti identifikacije vzorcev v nastalem kontekstu.

Kot lahko vidimo, gre v resnici za aplikacijo kandidatnega vzorca v prazen kontekst. S tem dobimo minimalno implementacijo vzorca in to implementacijo lahko nato preverimo s pomočjo enostavnih pravil – ali zadošča kriterijem kakšne izmed definiranih relacij. Proces aplikacije in preverjanja skladnosti med specifikacijo in implementacijo smo opisali v prejšnjih poglavjih.



Slika 10. Proces identifikacije relacij med vzorci načrtovanja

Rezultati analize objektnih ogrodij

Kot osnovo za preverjanje pravilnosti modela in postopka identifikacije vzorcev smo uporabili več različnih

razrednih knjižnic in objektnih sistemov različne obsega. Zaradi omejenega prostora bomo prikazali rezultate le nekaj vzorcev načrtovanja. Tabela 7 prikazuje seznam objektnih sistemov in programskih jezikov, v katerih so bili implementirani. Za prikaz velikosti sistemov je v tabelo dodanih še nekaj metrik (število razredov, skupno število podatkovnih atributov, skupno število metod, skupno število vseh relacij in število vseh dedovanj). Tabela 8 prikazuje rezultate identifikacije vzorcev načrtovanja, izvedenega s pomočjo podpornega okolja.

Kot lahko razberemo iz rezultatov, je dejansko število uporabljenih vzorcev precej majhno. Razlogov za to je več, glavni pa ta, da je veliko implementacij vzorcev nepopolnih. Na to kaže tudi dejstvo, da smo, ko smo dovolili majhno odstopanje pri implementaciji vzorca, našli večje število primerkov vzorcev (pri tem se moramo zavedati, da nismo ločevali med vrsto neskladja med definicijo in implementacijo vzorca, kar je lahko v nekaterih primerih privedlo do napačne interpretacije). Ti rezultati so v tabeli navedeni v oklepajih. Drugi razlog pa je ta, da so bili trije sistemi (LEDA, zAPP in jamaEL) zasnovani in implementirani v času, ko vzorci še niso bili splošno poznani. Zato so načrtovalci in implementatorji uporabljali svoje rešitve, ki pa vedno niso enake rešitvam, ki jih ponujajo vzorci načrtovanja. Edini sistem, ki je bil v veliki meri zasnovan s pomočjo vzorcev, je samo podporno okolje PatTool, ki izkazuje tudi največje število identificiranih primerkov vzorcev načrtovanja.

Vendarle pa rezultati analize te množice objektnih sistemov kažejo, da je s pomočjo formalne definicije vzorcev načrtovanja možno implementirati podporno okolje, ki je sposobno ekstrahirati visokonivojske podatke iz same programske kode.

Zaključek

Kot kažejo empirične raziskave tako v industrijskem kot tudi akademskem okolju, so informacije o uporabljenih vzorcih načrtovanja lahko pomembne za proces vzdrževanja in nadgradnje objektnih sistemov, saj omogočajo višjenivojski pogled na objektni sistem, z vgrajenim znanjem pa pomagajo graditi robustnejše in bolj prilagodljive sisteme.

Osnovna naloga raziskave, razvoj jezika za formalni opis vzorcev načrtovanja in njegova uporaba v procesu ekstrakcije visokonivojski podatkov iz obstoječe programske kode, je bila v celoti izpolnjena. Hkrati pa smo ugotovili še nekatere druge prednosti formalne definicije vzorcev, med katerimi je vsekakor najbolj pomembna možnost formalne definicije in deterministične identifikacije relacij med vzorci načrtovanja. Le-ta namreč zagotavlja trden temelj klasifikaciji vzorcev načrtovanja. Le s trdno in enolično določeno klasifikacijo se je namreč moč upreti vse večjemu številu zapisanih vzorcev in se izogniti redundanci in prekrivanju znanja

Kot je bilo prikazano v prispevku, je deterministična identifikacija vzorcev načrtovanja v obstoječi

Tabela 7 Seznam uporabljenih objektnih sistemov

Sistem / Merila	Razredi	Atributi	Metode	Relacije	Dedovanja
GUI Framework (Java)	45	187	3652	118	23
PatTool (Java)	72	67	2456	74	8
LEDA (C++)	150	501	4084	242	67
zAPP (C++)	240	1176	3590	298	145
jamaEL (C++)	382	2523	3845	723	212

Tabela 8 Rezultati identifikacije vzorcev načrtovanja

Vzorec / Sistem	GUI Framework	PatTool	LEDA	zAPP	jamaEL
Tovarniškametoda	9 (12)[1]	34 (34)	12 (16)	0 (14)	0 (0)
Obiskovalec	0 (0)	1 (2)	0 (0)	0 (0)	2 (2)
Šablonskametoda	1 (3)	4 (4)	0 (6)	2 (2)	6 (6)
Iterator	2 (2)	19 (19)	5 (9)	4 (8)	2 (3)
Mediator	13 (21)	2 (2)	0 (4)	0 (10)	0 (0)
Composite	6 (6)	0 (0)	4 (5)	2 (2)	4 (4)

3 *Prva števila pomeni število validiranih primerkov vzorca v programski kodi, medtem ko druga številka pomeni število validiranih vzorcev v programski kodi, kjer je nivo razlikovanja manjši kot 3.*

programski kodi možna. S pomočjo formalne predstavitve vzorcev načrtovanja je implementacija okolja, ki izkorišča prednosti uporabe vzorcev v razvoju, enostavna in učinkovita.

V sklopu nadaljnjega dela je vsekakor potrebno pomisliti na dejstvo, da je danes na voljo vedno več dokumentacije o razvoju sistema, ki bi jo lahko uporabili tudi za ekstrakcijo informacij o uporabljenih vzorcih načrtovanja. Na tem področju je zelo obetaven jezik XMI kot komunikacijski standard med CASE in razvojnimi orodji, ki s svojo izrazno močjo v veliki meri poenostavlja ekstrakcijo podatkov iz posameznega programskega jezika v obliko, potrebno za podporno okolje. Poleg tega bo naše nadaljnje delo usmerjeno k razvoju okolja, ki bi podpiralo grafično uporabo vzorcev načrtovanja. To bi po naši presoji prineslo še večjo učinkovitost uporabe vzorcev v proces razvoja in s tem pripomoglo h kakovostnejšemu razvoju programske opreme in še posebej kompleksnih objektnih sistemov.

Literatura

[Bosch 1996] Bosch J. "Language Support for Design Patterns. Proceedings TOOLS Europe '96, 1996

[Budinsky 1996] Budinsky F. J., M. A. Finnie, J. M. Vlissides, and P. S. Yu (1996). "Automatic code generation from design patterns". Object Technology Vol. 35, No. 2, 1996

[Computer 1990] Formal Methods, Computer, Vol. 23, No. 9, 1990

[Domajnko 1997] DOMAJNKO Tomaž, KOREN Silvo, CELCER Borut, DRVARIČ Ivan, "Uporaba objektnega pristopa na primeru izgradnje informacijske podpore poslovne funkcije v zavarvalništvu, zbornik srečanja Objektna tehnologija v Sloveniji OTS'97, 1997

[Domajnko 1998a] DOMAJNKO, Tomaž, ROZMAN, Ivan, HERIČKO, Marjan, GYÖRKÖS, József. "Vzorci in ponovna uporaba izkušenj", Uporabna informatika (Ljubljana), 1998

[Domajnko 1998b] DOMAJNKO, Tomaž, JURIČ, Branko-Matjaž, HERIČKO, Marjan, ROZMAN, Ivan, "Vzorci – sedanjost ali prihodnost?", Dnevi slovenske informatike, Portorož, 6. – 9. maj 1998. Zbornik posvetovanja, 1998

[Domajnko 1998c] Domajnko Tomaž, Rozman Ivan, Heričko Marjan, Jurič Branko-Matjaž, Beloglavec Simon, "Java in vzorci", Fakulteta za elektrotehniko, računalništvo in informatiko Maribor, Inštitut za informatiko, Center za objektno tehnologijo, 1998

[Domajnko 1999a] DOMAJNKO, Tomaž, JURIČ, Branko-Matjaž, HERIČKO, Marjan, ROZMAN, Ivan, "Formal based design patterns notation and support environment", 10th International Conference Information and intelligent systems IIS '99, Varaždin, 1999

[Domajnko 1999b] DOMAJNKO, Tomaž, JURIČ, Branko-Matjaž, BELOGLAVEC, Simon, ROZMAN, Ivan, "Design patterns management framework", Applied informatics : proceedings of the Seventeenth IASTED International Conference, Innsbruck, Austria, 1999

[Ducasse 1995] Ducasse Stephane, Mireille Blay-Fornarino, Anne-Marie Pinna, "A REflective Model for First Class Dependencies", Tools 1996, NY, 1995

[Gamma 1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns - Elements of Reusable Object- Oriented Software, Addison-Wesley, Reading, MA, 1995.

[Grand 1998] M. Grand, Patterns in Java – volume 1, Wiley, 1998

[Helm 90] Richard Helm, Ian M. Holland, and Dipayan Gangopadhyay, "Contracts: Specifying Behavioral Compositions in Object- Oriented Systems", OOPSLA '90 Conference Proceedings, Ottawa, Canada, 1990

[IEEE 1990/5] Formal Methods, IEEE Software, Vol. 7, No. 5, 1990

[PloP 1996] Pattern Languages of Programs, Allerton Park, Monticello-Illinois, USA, 1996

[PloP 1997] Pattern Languages of Programs, Allerton Park, Monticello-Illinois, USA, 1997

[PloP 1998] Pattern Languages of Programs, Allerton Park, Monticello-Illinois, USA, 1998

[PloP 1999] Pattern Languages of Programs, Allerton Park, Monticello-Illinois, USA, 1999

[*Quinressoft 1997*] *Quintessoft Engineering, Inc. (1997). C++ Code Navigator 1.1.* http://www.quintessoft.com, *1997*

◆

Tomaž Domajnko je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, kjer je vpisan v doktorski študij na področju računalništva in informatike. Njegovo raziskovalno delo obsega področje objektne tehnologije s poudarkom na izkoriščanju potenciala ponovne uporabe in zagotavljanju trajnosti objektov. Posebno področje njegovih raziskav so vzorci v programskem inženirstvu.

◆

Dr. Ivan Rozman je redni profesor Univerze v Mariboru, dekan Fakultete za elektrotehniko, računalništvo in informatiko v Mariboru in ustanovitelj Laboratorija za informacijske sisteme, ki ga vodi še danes. Je avtor številnih publikacij in vodja več raziskovalnih projektov. Diplomiral je na Fakulteti za elektrotehniko v Ljubljani, magistriral in doktoriral pa na Tehniški fakulteti v Mariboru.

◆

Dr. Marjan Heričko je docent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Njegovo raziskovalno delo obsega vse vidike objektne tehnologije s poudarkom na zagotavljanju kakovosti objektnega razvoja programskih sistemov. Diplomiral, magistriral in doktoriral je na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru.