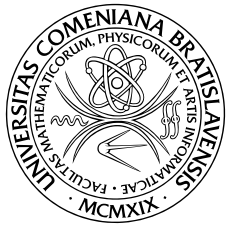
Fakulta matematiky, fyziky a informatiky
Univerzity Komenského v Bratislave



RNDr. Radovan Brečka

Autoreferát dizertačnej práce

Configuration-Oriented Programming

(Konfiguračne Orientované Programovanie)

na získanie vedecko-akademickej hodnosti philosophiæ doctor
v odbore doktorandského štúdia: 9.2.1. informatika

Bratislava 2014

Dizertačná práca bola vypracovaná v externej forme doktorandského štúdia na Katedre informatiky Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislave.

Predkladateľ: RNDr. Radovan Brečka
Katedra informatiky
Fakulta matematiky, fyziky a informatiky
Univerzity Komenského
Mlynská dolina
842 48 Bratislava

Školiteľ: Prof. RNDr. Branislav Rovan, PhD.
Katedra informatiky FMFI UK
Bratislava

Oponenti: .....

.....

.....

.....

.....

Obhajoba dizertačnej práce sa koná dňa ..... o ..... h.
pred komisiou pre obhajobu dizertačnej práce v odbore doktorandského štúdia
vymenovanou predsedom odborovej komisie dňa .....

v študijnom odbore 9.2.1. informatika

na .....

Predseda odborovej komisie:
Prof. RNDr. Branislav Rovan, PhD.
Fakulta matematiky, fyziky a informatiky
Univerzity Komenského
Mlynská dolina
842 48 Bratislava

1 Úvod

V predkladanej práci definujeme nový spôsob tvorby komplexných informačných systémov: Konfiguračne orientované programovanie (COP).

V súčasnej dobe je najrozšírenejším spôsobom programovania Objektovo orientované programovanie (OOP) založené na paradigme imperatívneho programovania.

Objektové programovanie prinieslo do programovania entitu Objektu s pozitívnymi vlastnosťami, ktoré umožnili efektívnejšie budovanie komplexných informačných systémov hlavne pomocou:

- dekompozície informačného systému do objektov,
- využitia vlastností objektov ako: znovu použiteľnosti, dedičnosti a zapuzdrenia objektov,
- využitia programovacích techník nad objektmi ako napr. polymorfizmu.

Žiaľ tieto pozitívne vlastnosti OOP platia iba na jednotlivých objektoch a už neplatia na štruktúrach objektov – jednotlivých častiach informačných systémov (časti informačného systému implementovaného v OOP už nemusia byť znovu použiteľné a mechanizmy ako napr. dedičnosť a polymorfizmus na častiach informačných systémov nie sú principiálne definované).

Vzhľadom na to, že OOP nerieši principiálne architektúru informačných systémov začali v poslednej dobe vznikať nové princípy a techniky, ktoré sa snažia eliminovať architektonické problémy vznikajúce pri budovaní a údržbe komplexných informačných systémov implementovaných pomocou OOP.

Z praktických skúsenosti z implementácie a údržby informačných systémov sme intuitívne vnímali stále opakujúce sa problémy, respektíve neefektívne činnosti. To nás motivovalo k ich podrobnejšej identifikácii a hľadaniu spôsobu eliminácie daných problémov.

Táto snaha nakoniec vyústila k identifikovaniu nového spôsobu tvorby informačných systémov – Konfiguračne orientvaného programovania a implementácii prototypu, ktorý tento spôsob umožňuje a ktorý popisujeme v predkladanej práci.

Identifikácia tohto nového spôsobu nás motivovala k podrobnejšiemu štúdiu danej oblasti a napísaniu tejto práce, kde sme sa podrobnou analýzou implementovaného prototypu dopátrali k identifikovaniu hlavnej príčiny: GOTO problému OOP a podrobnému popisu dôsledkov, ktoré daný problém zapríčiňuje. Tento popis tvorí prvú časť predkladanej práce. Zároveň sme v práci začali podrobnejšie skúmať aktuálne metódy tvorby informačných systémov z pohľadu GOTO problému OOP.

Druhá časť práce je venovaná samotnému popisu nového spôsobu tvorby komplexných informačných systémov. Najprv sa snažíme pomocou ilustračných obrázkov priblížiť princíp COP. Následne popisujeme definíciu operačnej sémantiky COP.

V poslednej časti práce popisujeme praktické skúsenosti z implementácie komplexných informačných systémov pomocou COP a popisujeme ako COP prirodzene eliminuje GOTO problém OOP.

2 Ciele práce

Ciele práce sú nasledovné:

- Identifikovať príčiny architektonických problémov pri tvorbe komplexných informačných systémov pomocou OOP.
- Popísať, ako účinne aktuálne moderné spôsoby tvorby komplexných informačných systémov eliminujú príčiny architektonických problémov OOP.
- Definovať operačnú sémantiku nového spôsobu tvorby komplexných informačných systémov: Konfiguračne orientovaného programovania.
- Popísať ako nový spôsob tvorby eliminuje identifikované problémy.
- Popísať praktické skúsenosti a postrehy získane pri tvorbe komplexných informačných systémov pomocou COP.

3 Výsledky a ich význam

3.1 Identifikovanie GOTO problému OOP

Aj keď sa to na prvý pohľad zdá byť neuveriteľné, tak na architektonickej úrovni sa opakuje analogický GOTO problém, ako pri kóde imperatívneho programovania. Tento problém spočíva vo vzájomnom priamom volaní objektov informačného systému. Pri grafickom zobrazení objektov a ich vzájomných volaní dostávame už pri malých informačných systémoch neprehľadný graf (pozri obrázok 1).

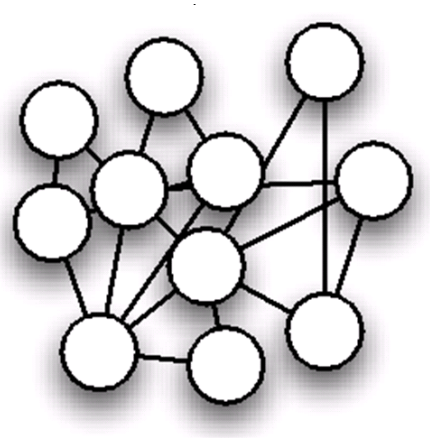
Takáto architektúra pripomína „špagetový efekt“ kódu imperatívneho programovania a spôsobuje problémy pri tvorbe a údržbe takto implementovaných komplexných informačných systémov.

Tento problém sme analogicky ku GOTO problému imperatívneho programovania nazvali GOTO problém OOP.

3.1.1 Hlavné prejavy GOTO problému OOP

Medzi najvážnejšie prejavy GOTO problému OOP patria:

1. netransparentná architektúra IS,
2. nízka miera znovu použiteľnosti častí IS,



Obr. 1: Príklad OOP architektúry

3. absencia jasných princípov kompozície častí IS.

Netransparentná architektúra IS
Vzájomné volania medzi objektmi spôsobujú na úrovni architektúry IS rovnaký problém ako príkaz GOTO v imperatívnom programovaní na úrovni kódu IS. Ak chce programátor zistiť aká je architektúra informačného systému, tak musí analyzovať všetky vzájomné volania medzi objektmi a architektúru informačného systému „vydedukovať“ z kódu IS.

Netransparentná architektúra výrazne komplikuje orientáciu v kóde IS a navyše pri údržbe znemožňuje efektívne meniť a adaptovať IS novým podmienkam a požiadavkám používateľa.

Dôsledkom tohto problému je nevyhnutnosť tvorby dokumentácie, ktorá externou formou zachytáva architektúru implementovaného IS.

Nízka miera znovu použiteľnosti častí IS
OOP prinášalo nádej na efektívnejšiu tvorbu a implementáciu komplexných IS a to hlavne deklaráciou nových vlastnosti objektovo orientovaného kódu akými boli zapúzdrenie, znovu použiteľnosť, dedičnosť a pod. Problémom je, že tieto vlastnosti platia len na úrovni jedného objektu a neplatia na štruktúrach objektov – častiach informačného systému. Rovnako ako v imperatívnom kóde riadenom príkazom GOTO nie je možné ľahko vybrať a skopírovať ľubovoľnú časť kódu, nie je ani v OOP možné ľahko skopírovať a znovu použiť časť IS, lebo nie je zrejmé či takáto časť IS nemá externé väzby s ostatnými časťami daného IS.

Absencia jasných princípov kompozície častí IS
Pri budovaní komplexných informačných systémov je okrem znovu použiteľnosti dôležité mať zabezpečené mechanizmy efektívnej kompozície IS – tj. kompozície IS z častí už implementovaných podobných IS.

Obdobne ako pri znovu použiteľnosti IS, je pri tvorbe IS pomocou OOP kompozícia možná len na úrovni kompozície objektov, ale už nie je možná na úrovni štruktúr objektov.

3.2 Definovanie operačnej sémantiky COP

3.2.1 Špecifikácia a inštancia

V práci sme definovali dva základné pojmy COP: *Modul* a *Konfigurácia*.

Najprv sme definovali základné stavebné prvky definície: *Špecifikáciu modulu*, ktorá definuje základne vlastnosti Modulu a *Inštanciu modulu*, ktorá definuje základné vlastnosti Konfigurácie. Vzhľadom na to, že medzi týmito základnými pojmami je cyklická väzba sú tieto pojmy definované v jednej definícii.

Následne sme definovali hierarchiu modulov pomoocu P-konzistencie pomocou ktorej definujeme Modul a Konfiguráciu.

Definícia 1 (Špecifikácia modulu a Inštancia modulu):
Špecifikácia modulu je pätica $M = (m,\ p,\ D,\ S,\ O)$, *pričom:*

i) $m \in \Sigma^+$ – *je názov modulu,*

ii) $p \in \Sigma^*$ – *je názov rodičovského modulu,*

iii) $D = \{(k_1,\ DI_1),\ (k_2,\ DI_2), \ldots,\ (k_{n_D},\ DI_{n_D})\}$ – *je Dátová deklarácia modulu, pričom:*

- $n_D \geq 0$,
- $k_i \in \Sigma^+$, $1 \leq i \leq n_D$, k_i *sú navzájom rôzne identifikátory,*
- $DI_i \in \mathbb{C}$, $1 \leq i \leq n_D$ – *sú privátne inštancie modulov,*

iv) $S = (S_A,\ S_I)$ – *je Deklarácia konfiguračnej šablóny modulu, pričom:*

- $S_A = \{k_1,\ k_2, \ldots,\ k_{n_{S_A}}\}$ – *je deklarácia povolených identifikátorov atribútov modulu, pričom:*
  - $n_{S_A} \geq 0$,
  - $k_i \in \Sigma^+$, $1 \leq i \leq n_{S_A}$, k_i *sú navzájom rôzne,*
- $S_I = \{(k_1,\ m_1),\ (k_2,\ m_2), \ldots,\ (k_{n_{S_I}},\ m_{n_{S_I}})\}$ – *je deklarácia povolených identifikátorov a názvov Špecifikácii modulov inštancovaných podmodulov, pričom:*
  - $n_{S_I} \geq 0$,
  - $k_i \in \Sigma^*$, $1 \leq i \leq n_{S_I}$, – *je deklarácia povolených identifikátorov, (ak* $k_i = \lambda$ *potom sú povolené inštancie podmodulov s ľubovoľným identifikátorom),*
  - $m_i \in \Sigma^+,\ \ 1 \leq i \leq n_{S_I}$ – *je deklarácia povolených názvov Špecifikácii modulov* m_i, $M_i = (m_i,\ p_i,\ D_i,\ S_i,\ O_i)$ *inštancovaných podmodulov,*

v) $O = \{(k_1,\ o_1),\ (k_2,\ o_2), \ldots,\ (k_{n_O},\ o_{n_O})\}$ – *je množina metód modulu, pričom:*

- $n_O \geq 0$,
- $k_i \in \Sigma^+$, $1 \leq i \leq n_O$, k_i *sú navzájom rôzne – sú identifikátory metód,*
- $o_i : (\mathbb{V} \times \mathbb{D} \times \mathbb{C}) \rightarrow \mathbb{V}$, $1 \leq i \leq n_O$, *– sú funkcie reprezentujúce kód metód, pričom:*
  - $\mathbb{D}$ *je množina všetkých možných Dátových deklarácii modulu* D_j *Špecifikácie modulu* $M_j = (m_j,\ p_j,\ D_j,\ S_j,\ O_j) \in \mathbb{M}$.

Množinu všetkých Špecifikácii modulov nad abecedou Σ *označujeme ako* $\mathbb{M}$.

Inštancia modulu *Špecifikácie modulu* $M = (m,\ p,\ D,\ S = (S_A,\ S_I),\ O)$ *je trojica* $C = (M,\ A,\ I)$, *pričom:*

i) $M \in \mathbb{M}$,

ii) $A = \{(k_1,\ a_1),\ (k_2,\ a_2), \ldots,\ (k_{n_A},\ a_{n_A})\}$ *– je množina atribútových hodnôt, pričom:*

- $n_A \geq 0$,
- $k_i \in S_A$, $1 \leq i \leq n_A$, k_i *sú navzájom rôzne – sú identifikátory atribútov podľa šablóny* S_A *definovanej v Špecifikácii modulu* M,
- $a_i \in \Sigma^*$ *– sú hodnoty atribútov,*

iii) $I = \{(k_1,\ SI_1),\ (k_2,\ SI_2), \ldots,\ (k_{n_I},\ SI_{n_I})\}$, *pričom:*

- $n_I \geq 0$,
- $k_i \in \Sigma^+$, $1 \leq i \leq n_I$, k_i *sú navzájom rôzne,*
- $SI_i = (M_i,\ A_i,\ I_i) \in \mathbb{C}$, $1 \leq i \leq n_I$.

Množinu všetkých Inštancií modulov nad abecedou Σ *označujeme* $\mathbb{C}$.

Špecifikáciu modulu M, $C = (M,\ A,\ I)$ *nazývame* ***Inštancovaný modul***.

Inštanciu SI_i, $(k_i,\ SI_i) \in I$, $C = (M,\ A,\ I)$ *nazývame* ***Podmodul***.

3.2.2 Rodičovská konzistentnosť

Vzhľadom na to, že nad modulmi COP je definovaný vzťah dedičnosti potrebujeme, aby moduly mali medzi sebou správnu hierarchiu dedenia. Preto si zavedieme ďalšie pomocné pojmy, ktorými túto vlastnosť vyjadríme formálne.

Definícia 2 (Rodičovsky konzistentná množina Špecifikácií modulov):
Ľubovoľná neprázdna podmnožina $P \subset \mathbb{M}$ *je Rodičovsky konzistentná množina Špecifikácií modulov ak platí::*

i) $\forall M_1,\ M_2 \in P,\ M_1 = (m_1,\ p_1,\ D_1,\ S_1,\ O_1),\ M_2 = (m_2,\ p_2,\ D_2,\ S_2,\ O_2)$ *platí:*
$m_1 \neq m_2$ *implikuje* $M_1 \neq M_2$ *(Špecifikácie modulu majú unikátne mená),*

ii) $\forall M \in P,\ M = (m,\ p,\ D,\ S,\ O)$ *platí:*

(a) $p = \lambda$ *(Špecifikácia modulu nemá rodičovský modul)*
alebo

(b) $p \in \Sigma^+$, *pričom existuje postupnosť Špecifikácií modulu* $M_1,\ M_2,\ \ldots,\ M_k,\ k > 0,\ M_i = (m_i,\ p_i,\ D_i,\ S_i,\ O_i) \in P,\ 1 \leq i \leq k$: *pričom* $p = m_1\ \wedge\ p_j = m_{j+1}$: $1 \leq j < k\ \wedge\ p_k = \lambda$

Postupnosť $M_1,\ M_2,\ \ldots,\ M_k$ *označujeme ako* ***pred(M)****, a hovoríme, že Špecifikácia modulu* M *je následníkom Špecifikácie modulu* M_i *a Špecifikácia modulu* M_i *je prechodcom Špecifikácie modulu* M. *V prípade, že* $p = \lambda$, *je* $pred(M) = \{\}$.

Rovnako ako je v OOP definované dedenie metód objektov je aj v COP definované dedenie metód. Zároveň je obdobným spôsobom definované dedenie Dátových deklarácií modulu (dedenie privátnych Inštancií modulov). V nasledujúcom texte sú tieto vlastnosti vyjadrené formálne.

Definícia 3 (Dedená množina dátových deklarácii):
Dedená množina dátových deklarácii ***pred-D(M)*** *Špecifikácie modulu* $M = (m,\ p,\ D,\ S,\ O)$ *je množina:*

i) $pred\text{-}D(M) = D \cup \{(k_x,\ DI_y) \in D_i,\ M_i = (m_i,\ p_i,\ D_i,\ S_i,\ O_i) \in pred(M)$, *pričom:*

(a) $\nexists (k_x,\ DI_x) \in D$
a zároveň

(b) $i = min\{j \mid (k_x,\ DI_z) \in D_j,\ M_j = (m_j,\ p_j,\ D_j,\ S_j,\ O_j) \in pred(M)\}$

– je to množina Dátových deklarácií modulu doplnená o množiny Dátových deklarácií jeho predchodcov, pričom prioritu majú deklarácie podľa postupnosti $pred(M)$.

Definícia 4 (Dedená množina metód):
Dedená množina metód ***pred-O(M)*** *Špecifkácie modulu* $M = (m,\ p,\ D,\ S,\ O)$ *je množina:*

i) $pred\text{-}O(M) = O \cup \{(k_x,\ o_y) \in O_i,\ M_i = (m_i,\ p_i,\ D_i,\ S_i,\ O_i) \in pred(M)$, *where:*

(a) $\nexists (k_x,\ o_x) \in O$,
and

(b) $i = min\{j \mid (k_x,\ o_z) \in O_j,\ M_j = (m_j,\ p_j,\ D_j,\ S_j,\ O_j) \in pred(M)\}$.

– je množina dedených metód analogicky ako sú dedené metódy v OOP.

Okrem správneho dedenia modulov je potrebné, aby aj konfigurácia bola zložená z už definovaných modulov – tj. aby konfigurácia neodkazovala na modul, ktorý nie je definovaný. Túto vlastnosť vyjadríme definovaním pojmu *P-konzistentná inštancia modulu*, ktorá vyjadruje túto vlastnosť na konfigurácií a definovaním pojmu *P-konzistentná množina modulov*, ktorá vyjadruje túto vlastnosť na moduloch.

Definícia 5 (*P*-konzistentná inštancia modulu)):
Inštancia modulu $C \in \mathbb{C}$, $C = (M,\ A,\ I)$, $M = (m,\ p,\ D,\ S = (S_A,\ S_I),\ O)$ *je P-konzistentná inštancia modulu pre Rodičovsky konzistentnú množinu Špecifikácií modulov P, ak platí:*

i) $M \in P$ – C *je inštanciou Špecifikácie modulu z* P,

ii) $\forall(k_i,\ SI_i) \in I : SI_i$ *je P-konzistentná inštancia modulu,*

iii) $\forall(k_i,\ SI_i) \in I,\ SI_i = (M_i,\ A_i,\ I_i),\ M_i = (m_i,\ p_i,\ D_i,\ S_i,\ O_i) \in P$ *platí:*

- $(k_i,\ m_i) \in S_I$ – *inštancia* SI_i *modulu* M_i *je priamo povoleného typu* m_i,
- *alebo*
  $\exists(k_i,\ m_j) \in S_I\ \wedge\ \exists M_j = (m_j,\ p_j,\ D_j,\ S_j,\ O_j) \in pred(M_i)$, – *inštancia* SI_i *modulu* M_i *je následníkom modulu* M_j, *ktorý je povoleného typu* m_j,
- *alebo*
  $(k_i,\ \lambda) \in S_I$ – *sú povolené ľubovoľné typy modulov s identifikátorom* k_i,
- *alebo*
  $\exists(\lambda,\ m_i) \in S_I$ – *inštancia* SI_i *modulu* M_i *je priamo povoleného typu* m_i *so všeobecným identifikátorom* λ,
- *alebo*
  $\exists(\lambda,\ m_j) \in S_I \wedge \exists M_j = (m_j,\ p_j,\ D_j,\ S_j,\ O_j) \in pred(M_i)$, – *inštancia* SI_i *modulu* M_i *je následníkom modulu* M_j, *ktorý je povoleného typu* m_j *so všeobecným identifikátorom* λ,
- *alebo*
  $(\lambda,\ \lambda) \in S_I$ – *sú povolené ľubovoľné typy modulov s ľubovoľným identifikátorom.*

P-konzistentnú inštanciu modulu nazývame ***Konfigurácia****.*

Ďalej potrebujeme zabezpečiť, aby Dátová deklarácia modulu obsahovala len inštancie modulu, ktoré sú Konfigurácie. Túto vlastnosť definujeme v nasledujúcej definícii.

Definícia 6 (*P*-konzistentná množina Špecifikácií modulov):
Ľubovoľná neprázdna podmnožina $P \subset \mathbb{M}$ *je P-konzistentná množina Špecifikácií modulov, ak platí:*

i) P *je Rodičovsky konzistentná množina Špecifikácií modulov,*

ii) $\forall M \in P,\ M = (m,\ p,\ D,\ S,\ O),\ \forall (k_i,\ DI_i) \in D : DI_i$ *je P-konzistentná inštancia modulu pre Rodičovsky konzistentnú množinu Špecifikácií modulov* P.

Prvok P-konzistentnej množiny Špecifikácií modulov nazývame ***Modul***.

3.2.3 Definícia aplikácie a spustenie aplikácie

Ak máme definované pojmy *Modul* a *Konfigurácia* možeme definovať aplikáciu.

Definícia 7 (Aplikácia):
Aplikácia je dvojica $App = (Mod,\ Config)$, *pričom:*

i) Mod je množina Modulov (P-konzistentná množina Špecifikácií modulov),

ii) Config je Konfigurácia (P-konzistentná inštancia modulu).

Aplikácia je zložená z množiny modulov aplikácie *Mod* a aplikačnej konfigurácie *Config*, pričom nad modulmi platia hore uvedené pravidlá – tj. moduly majú striktnú stromovú hierarchiu, Dátová deklarácia modulu a aplikačná Konfigurácia sú zložené z inštancií definovaných modulov v súlade s konfiguračnou šablónou S_A a S_I.

Poslednou časťou definície COP je definovanie ako sa aplikácia vykonáva / spúšťa. Na vykonanie aplikácie potrebujeme najprv definovať čo znamená vykonanie metódy v COP.

Definícia 8 (Spustenie metódy):
Spustenie metódy o s identifikátorom k_o, *vstupom* $In \in \mathbb{V}$ *a Konfiguráciou* $C = (M,\ A,\ I),\ M = (m,\ p,\ D,\ S,\ O),\ M \in Mod$ *je definované ako:*

i) $exec\text{-}App(C,\ k_o,\ In) = c(In,\ pred\text{-}D(M),\ C) \Leftrightarrow \exists (k_o,\ o) \in pred\text{-}O(M)$

ii) $exec\text{-}App(C,\ k_o,\ In) = \lambda \Leftrightarrow \nexists (k_o,\ o) \in pred\text{-}O(M)$

Čiže vykonanie metódy je na konfigurácii (nie na module, kde je metóda definovaná) a výsledok metódy závisí od vstupu In, dedenej dátovej deklarácie inštancovaného modulu $pred\text{-}D$ a samotnej konfigurácie C.

Posledným krokom je definovanie spustenia aplikácie, čo je v podstate spustenie metódy o s identifikátorom k_o na koreni aplikačnej konfigurácie $Config$.

Definícia 9 (Spustenie aplikácie):
Spustenie aplikácie App = (Mod, Config) je definované ako:

i) $exec\text{-}App(Config,\ k_o,\ In),\ k_o \in \Sigma^+,\ In \in \mathbb{V}$.

3.3 Popis praktických skúsenosti s COP

V ďalšej časti práce sa venujeme popisu praktických skúseností pri implementácii konfiguračného nástroja blox® platform a popisu praktických skúsenosti získaných pri tvorbe komplexných informačných systémov pomocou Konfiguračne orientovaného programovania a nástroja blox® platform (zoznam projektov uvádzame v prílohe A).

Zároveň v práci popisujeme ako Konfiguračne orientované programovanie prirodzene eliminuje GOTO problém OOP.

4 Záver

V tejto práci sme popísali základné myšlienky konfiguračne orientovaného programovania (COP), ktoré otvára nové perspektívy v spôsobe tvorby komplexných informačných systémov. Zároveň sme popísali hlavné výhody nového spôsobu programovania, ktoré sa nám podarilo odhaliť pri tvorbe komplexných intranetových a extranetových informačných systémov pomocou COP a prototypu blox® platform.

Ná základe dôvodov uvedených v texte tejto práce, ktorými sú hlavne:

- programovanie pomocou konfiguračných vzoriek,
- konfiguračná abstrakcia,
- polymorfizmus na konfigurácii,
- eliminácia *GOTO problému OOP*,

sme presvedčení, že COP je prirodzeným nástupcom OOP pri tvorbe komplexných informačných systémov a prináša spôsob, ako komplexné informačné systémy rádovo efektívnejšie implementovať a udržiavať.

Literatúra

[Arm06] Deborah J. Armstrong. The quarks of object-oriented development. *Communications of the ACM*, 49(2):123–128, February 2006.

[Boo98] Grady Booch. *Object-Oriented Analysis and Design.* Addison Wesley, Santa Clara, California, 2nd edition, 1998.

[Bre10] Radovan Brečka. Configuration-oriented programming. *Proceedings of 6th Central and Eastern European Software Engineering Conference*, October 2010.

[Bre11] Radovan Brečka. *Konfiguračne orientované programovanie.* Rigorózna práca (RNDr. thesis), Department of Computer Science, FMFI, Comenius University, Bratislava, Bratislava, 2011.

[DDH72] Ole-Johan Dahl, Edsger W. Dijkstra, and Charles A. R. Hoare. *Structured Programming*. Academic Press, London, 1972.

[Dij68] Edsger W. Dijkstra. Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, March 1968.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, New York, 1995.

[JPW04] Ivar Jacobson and Ng Pan-Wei. *Aspect-Oriented Software Development with Use Cases*. Addison Wesley Professional, New York, 2004.

[KMSD92] Jeff Kramer, Jeff Magee, Morris Sloman, and Naranker Dulay. Configuring object-based distributed programs in rex. *IEE Software Engineering Journal*, 7(2):139–149, March 1992.

[Lar02] Craig Larman. *Applying UML and patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, Inc., New York, 2002.

[LZ75] Barbara Liskov and Stephen Zilles. Specification techniques for data abstractions. *ACM SIGPLAN Notices - International Conference on Reliable Software Homepage archive*, 10(6):72–87, June 1975.

[Mit03] John C. Mitchell. *Concepts in programming languages*. Cambridge University Press, Cambridge, 2003.

[MW06] Thomas Mattern and Dan Woods. *Enterprise SOA: Designing IT for Business Innovation*. O'Reilly Media, Sebastopol, 2006.

[Rie00] Dirk Riehle. *Framework Design: A Role Modeling Approach*. Ph.D. Thesis, No. 13509. ETH Zürich, Zürich, 2000.

[Seb02] Robert W. Sebesta. *Concepts of Programming Languages*. Addison-Wesley, Boston, 5th edition, 2002.

[ST04] Alan Shalloway and James R. Trott. *Design Patterns Explained, A New Perspective on Object–Oriented Design*. Addison-Wesley Professional, Boston, 2004.

[Str07] Bjarne Stroustrup. Evolving a language in and for the real world: C++ 1991-2006. *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, June 2007.

[Str09] Bjarne Stroustrup. *Programming: Principles and Practice using C++*. Addison-Wesley, 2009.