RAZMIŠLJANJE O ZASNOVI JEZIKA CSP

Bojan Peček

Članek je osebno razmišljanje avtorja o zasnovi produkta in se ograjuje od ocenjevanja kakovosti in učinkovitosti samega izdelka. Obravnava in komentira osebne poglede na zasnovo jezika CSP (Cross System Product) firme IBM, ki naj bi bil jezik IV. generacije. Stvar res ni več sveža, vendar našo revijo razumem kot mesto, kjer bomo izmenjavali izkušnje v zvezi z novimi informacijskimi tehnikami in orodji. Želim, da članek spodbudi še koga, ki bo repliciral s svojim, morda drugačnim videnjem.

1. UVOD

Vsak dan se pojavljajo nove metodologije od sistemske analize do vodenja projektov, katerih namen je dvig produktivnosti pri izdelavi programske opreme, kot npr. orodja CASE. Svoj delež naj bi prispevali tudi jeziki IV. generacije, ki naj bi nadomestili klasične kot so COBOL, PL/1 itd. Povezava metodologij, tehnologij in novih jezikov naj bi temeljito spremenila produktivnost programerskega kadra.

2. OSNOVNE ZNAČILNOSTI JEZIKA CSP

Opazna razlika, ki jo programer 'začuti' v prvem kontaktu z jezikom, je gotovo parcialnost programa. Dosedanji način obravnavanja programa, ki se vleče še iz obdobja programiranja v zbirnikih, je bil v celovitosti programske kode. Vse definicije v zvezi s programom so se nahajale v eni datoteki, oziroma enem šopu kartic. Tak skup je vseboval tako definicijo delovnega pomnilnika, izgled vhodnih in izhodnih struktur vključno z obliko zaslona, kot samo izvajalno kodo. INCLUDE oziroma COPY stavki sicer lahko razbijejo program na več datotek, zagotavljanje celovitosti programa in pregled medsebojne prepletenosti sta v tem primeru prepuščena organiziranosti okolja.

CSP prekinja s tradicionalnim pristopom, kar se opazi že pri uporabi notacije. Izraz 'program', ki se uporablja za tradicionalno programsko enoto, je nadomestil pojem 'aplikacija'. Aplikacija v CSP-ju je pravzaprav samo skupni pojem, ki združuje vse koščke programa: definicije delovnega prostora, opise zaslonskih slik (mape) in izpisnih list, module, ki vsebujejo ukaze programske logike. Moduli programske logike vsebujejo samo ukaze. Opis spremenljivk, ki jih uporabljajo, je v segmentu za opis definicij delovnega prostora. Celovitega programa, kot je v COBOL-u opisan v štirih divizijah, ne vidimo nikoli.

Segmenti se hranijo v knjižnjici - MSL (Module Source Library). Pregled seznama posameznih definicij je še najbolj podoben pregledu imenika (direktorija). Aplikacija vsebuje module programske logike, koda v njih se sklicuje na podatkovne elemente, ki so vsebovani v opisih podatkovnih struktur.

Razen svojih definicij lahko programer enakovredno uporablja tudi definicije kolegov v njihovih knjižnjicah. Program poišče posamezno definicijo po vrstnem redu priključitve knjižnic, vendar definicij na drugih knjižnjicah ne more spreminjati. S tem je zagotovljena enkratnost opisov.

3. ORGANIZACIJA CSP-ja

3.1. Definiranje spremenljivk

Način opisa delovnega prostora je zelo podoben opisom v COBOL-u oz. PL/1. Pomnilnik se lahko opisuje v treh oblikah:

- tabele, kjer so podatki statični in inicializirani
- zapisi struktur na vhodno izhodnih medijih
- prosto uporabljen delovni pomnilnik.

Vsaka definicija delovnega pomnilnika je zapis v knjižnjici, ki jo lahko vidijo vsi, kar pomeni, da za podatkovne elemente zadostuje ena definicija. Slednja velja tako za vključitev podatka v strukturo datoteke, kot tudi za zaslonsko sliko. Kadar posameznik s skupno definicijo, ki se nahaja v neki skupni knjižnjici, ni zadovoljen, lahko definicijo z enakim imenom opredeli v svoji knjižnici.

Velja omejitev, da je vsaka definicija lahko opisana samo enkrat z nižjim nivojem. Cobolske klavzule "REDEFINE" CSP ne pozna. Še bolj 'udarnih posebnosti' kot npr. v COBOL-u "REFERENCE IS POINTER" ali v PL/1 "DEFINED" oz. "BASED" CSP ne pozna.

Strukturiranje zapisov na zunanjih medijih (npr. v tabeli baze ali klasični datoteki) ni dovoljeno. Sprejema samo elementarne podatke.

Žal pri definiciji prostora ni mogoče vpisati inicializacije spremenljivk, kar je brez dvoma v nasprotju s tem, kar nam poiskušajo vtepsti v glavo 'guruji', ki nas poučujejo o načinih programiranja. Privzete vrednosti je mogoče opisati samo v tabeli (ki je ne smemo zamenjati s tabelo iz baze!).

3.2 Programska logika

3.2.1 Sintaksa ukazov

Povezanosti s COBOL-om CSP ne more skriti. Celo pri generiranju aplikacije se najprej vsi moduli in opisi sestavijo v cobolsko kodo, ki se nato prevaja in veriži. Vpliv je viden

tudi v obliki in imenih ukazov. Glavne najdemo tudi v COBOL-u, n.pr. MOVE, COMPUTE, IF/ELSE, PERFORM, DO WHILE.

Poleg glavnih ukazov pozna CSP tudi klic zunanjega programa CALL. Zanimivo je, da vsebuje tudi ukaz, podoben cobolskemu, SEARCH ALL za preiskovanje urejenih tabel, čeprav se izredno redko uporablja.

S tem je nabor ukazov pravzaprav izčrpan. Obstaja še niz ključnih besed, oziroma registrov, preko katerih lahko programer testira status baze, zaključno tipko pri vnosu podatkov na mapo, itd. Vsekakor 'udarnih' ukazov, kot so SORT, STRING, UNSTRING ali MOVE z dinamičnimi mejami, kot ga v COBOLu predpisuje ANSI-85, ni!

Snovalci CSP-ja so se pri znani programerski dilemi "To GOTO or not to GOTO - this is the question now" odločili za "not to GOTO". Kar pomeni, da je CSP jezik brez GOTO stavka, labele sploh ne obstajajo. Prekinitev sekvence je mogoče izvesti samo na tako imenovanem postprocesiranju modula na najvišjem nivoju (nivo 001). V tem procesu lahko programer zahteva nadaljevanje izvajanja na začetku nekega drugega modula, ki pa mora biti ravno tako na prvem nivoju.

Drugi moduli, ki so nižje v strukturi, lahko samo na dva načina predčasno prekinejo izvajanje svojih procedur:

- pred iztekom zapustijo izvajanje in se vrnejo v predhodni modul, od koder so bili klicani
- ali pa zahtevajo takojšno prekinitev izvajanja in skok v postprocesiranje modula na prvem nivoju.

Zanimiva rešitev, saj vsi drugi programski jeziki ohranjajo GOTO stavek, kljub temu, da prisegajo na svojo modularno zasnovo.

3.2.2 Modularnost

Sedemdeseta leta označuje iskanje, eksperimentiranje in uvajanje računalniških jezikov (RATFOR, PASCAL itd), ki naj bi omogočali strukturiranje in moduliranje programov.

V CSP-ju je posamezen zapis programske logike (modul) v knjižnjici še najbolj podoben paragrafu v COBOL-u. Vsak modul lahko kliče ostale (z ukazom PERFORM). Tako dobimo hierarhijo modulov, kjer je na vrhu aplikacija, ta pa kliče procese na prvem nivoju, slednji pa njim podrejene itd. V CICS-u struktura pregleda spominja na eksplozijo v kosovničnem procesu.

Najbolj zanimiva in marsikateremu programerju najbolj sporna se zdi omejitev namena posameznega modula. Vsak modul v CSP-ju ima lahko neomejeno število logičnih ukazov, vendar lahko opravi samo eno periferno operacijo. Obstajajo tudi moduli brez posebne funcije, ki najbolj pogosto samo kličejo druge module.

Modulova funkcija se lahko izvaja kadarkoli - ukazi logike so lahko tako pred, kot tudi po izvajanju modulove funkcije. Običajno so modulove funkcije vhodno izhodni ukazi. V povezavi z DB/2 so to npr. običajni SQL ukazi. Ker lahko vsak modul izvaja samo eno zunanjo funkcijo (opcijo), mora programer, če želi spremeniti vsebino nekega zapisa v datoteki (bazi podatkov), napisati tri module:

- modul glavnega procesa, ki ne bo usmerjen v nobeno funkcijo
- (opcija EXECUTE), klical pa bo ostala dva:
- modul z opcijo čitanja zapisa z zaklepanjem, ter
- modul z opcijo spreminjanja vsebine.

Logika, oziroma koda polnjenja spremenjene vsebine je lahko v kateremkoli od treh modulov. Potrditev se izvaja kot poseben ukaz programske logike in ne potrebuje posebnega modula. Isto logiko je seveda mogoče 'stlačiti' tudi v dva modula, kjer bo glavni modul s funkcijo zapisa vsebine v bazo podatkov poklical še en modul, ki bo zapis včital in zaklenil. Torej navkljub 'zaprtosti', oziroma urejenosti kode so še vedno mogoče bogate variacije v stilih programiranja. Zato dodatni dogovori (interni standardi) v zvezi s stilom programiranja v centru ne bodo odveč!

4. OPAŽANJA PRI DELU S CSP-jem

4.1 Vtisi programerjev

Verjetno ni bralca, ki se ne bi strinjal, da je potrebno med razvijanjem programa bolj razmišljati o vzdrževanju, kot o hitrem razvoju. Verjetno je bil ozki nabor ukazov zasnovan z namenom, da bi dosegli enostavno in kar najbolj uniformno programiranje. Žal se 'stari programerski mački' zgražajo: "Za vsako najmanjšo malenkost imaš neskončno veliko kode". Prisegajo na dodatne module v PL/1, COBOL-u, itd, ki naj 'omilijo štoravost' programiranja.

Ena od boljših stvari, ki jo prinaša CSP, je avtomatski zamik kode. Kodo v telesu IF in DO WHILE stavka, kot tudi nadaljevanje ukaza v naslednji vrstici, editor avtomatsko zamakne za dva znaka v desno. Proti temu ukrepu se programer na srečo ne more boriti. Resda je lahko sporno, ali sta dva znaka, ki ju CSP zamakne, dovolj, saj je marsikdo navajen na večji ali manjši zamik. Toda ideja je vsekakor dobra, saj uvaja vsaj nekaj reda v kodo. CSP zahteva tudi, da je v vsaki vrstici največ en ukaz.

4.2 Spremljevalno okolje

Programer dela v CSP-ju vseskozi z internim editorjem. V verziji za operacijski sistem MVS je seveda orientiran na CICS in s tem na protokol 3270. Predvsem moti, da program dovoli naenkrat editirati samo eno definicijo. Zato ni mogoče hkrati opisovati delovne spremeljnike in kode. Sedaj, ko smo vsi navajeni na večokenske urejevalce besedil, vsekakor neprijetna omejitev.

Če je osnovni editor še nekako sprejemljiv, to gotovo ne velja za zaslonski urejevalec slik, oziroma editor, ki omogoča definiranje in vzdrževanje ekranskih mask. Protokol obravnava celotno masko kot enovito polje 1920 znakov, zato se pojavljajo neopravičljive težave pri vrivanju novih polj. Problem doseže vrhunec, če hoče programer neko polje v maski prestaviti na drugo lokacijo.

V današnji stopnji programerskega udobja je skoraj nerazumljiva odtujenost diagnostike. Prevajalnik sicer jasno izpiše diagnozo o prenosu spremeljivke tipa karakter v spremenljivko definirano kot binarno. Pri tem vedno točno pove ime modula. Vendar nikoli ne pove, v kateri vrstici modula.

In če vsebuje modul 500 vrstic, potem je programer prepuščen lastni inovativnosti in intuitivnosti. Izpisnega 'listinga' namreč ni!

Podatkovni slovar je realiziran preko knjižnic. Organizacija knjižnjice je VSAM s strukturiranim ključem. Zato je vsaka definicija, na katerega se sklicujejo drugi moduli ali zapisi direktno, hitro dosegljiva. Stvar pa se zaplete pri izdelavi seznama modulov, ki se sklicujejo na posamezen pojem. Običajno ima programer priključene tudi druge knjižnice. Program zaporedno preiskuje vse knjižnice, kar traja izredno dolgo. Zaustavitev zgrešenega poizvedovanja pa je seveda vezana na administriranje CICS-a.

Preprosta organizacija podatkovnega slovarja povzroča nevšečnosti tudi pri vsakdanjem delu. Če nekdo spremeni definicijo npr. podatkovnega elementa v skupni projektni knjižnjici, podatkovni slovar ne javi nobenega opozorila. Šele čez čas, ko prevajamo neki drug program, prevajalnik odkrije neskladje v podakovni strukturi. Za nič krivega in nič dolžnega programerja je to kot udarec po glavi.

Verjetno lahko povežemo enostavno organizacijo podatkovnega slovarja tudi s priporočili, ki jih najdemo v sistemski literaturi. Eno teh priporočil predlaga, da se vsako ime (bodisi programske spremenljivke, ime mape, procesa itd.) začne s prvimi štirimi črkami aplikacije, v kateri se definicija uporablja. S tem se navidezno množica definicij uredi, nastanejo pa problemi, če se definicija uporablja tudi v drugih aplikacijah (programih).

Če hočemo striktno upoštevati priporočilo, potem v naslednjem programu ne smemo uporabiti že napisanega modula za npr. obstoj ključa v neki datoteki. Potrebno je napisati enak modul z drugim imenom, kar seveda privede do podvajanje kode.

Glede na vse opisane težave, ki lahko nastanejo z nenatančnim delom, potrebuje celoten sistem preveč togega administriranja, strogo vezanega na CICS. Morda velja omeniti, da nekateri avtorji pričakujejo od jezikov IV. generacije takšno enostavnost, da bi jih lahko uporabljali tudi neprofesionalni programerji. Zanimivo je, da v uvodu k verziji 2.1 proizvajalec še deklarira CSP kot jezik IV. generacije, slednje pa v podobnem uvodu k verziji 3.1 izgine.

5. ZAKLJUČEK

CSP je vsekakor jezik z najbolj zanimivo zasnovo, kar sem jih videl v sferi poslovne informatike. V nadaljnem razvoju lahko pričakujemo povezavo z orodji CASE in repozitorijem, čemur je bil glede na zasnovo verjetno tudi namenjen.

Vsekakor ne gre prezreti, da zasnova jezika nekako materializira mnoga navodila, ki smo jih pri programiranju privzeli - modularnost, kohezija modulov, zamik kode, itd. Osebno sem navdušen tudi nad enostavno zasnovo in predvsem majhnim številom ukazov. Menim, da slednje otežuje pisanje 'sofisticirane kode', ko se programer ponaša s svojim znanjem in zamislimi.

Verjamem tudi v 'vzgojnost' CSP-ja. Ko sem vzporedno pisal bolj zapletene programe v PL/1, sem kar prevzel pravila, ki jih nekako neagresivno ponuja CSP.

■■■

Kdo potrebuje informacijska znanja?

Od bralke Tanje Kovač smo prejeli pismo, ki ga v celoti objavljamo

Spoštovani!

V reviji Uporabna informatika št. 2, letnik II 94, me je članek gospe Marte Božič "Infomacijska znanja za informacijsko dobo" spodbudil, da vam pošiljam svoj komentar.

Na področju informatike delam 14 let, torej od začetka 80. let, ko smo pri nas pravzaprav začeli vstopati v informacijsko dobo. Delala sem na različnih področjih (trgovina, proizvodnja, uprava, malo gospodarstvo), zato se lahko pohvalim z bogatimi izkušnjami spremljanja razvoja informatike in sicer iz raznih vidikov: od konkretnih izvedbenih del, pojava potreb po izobraževanju v delovnih organizacijah do pojava problema ustreznih kadrov, ki bi prenašali potrebna informacijska znanja v poslovni sistem. Informatiko sem privzela za svojo ciljno poklicno usmeritev, ker imam to področje rada, ker zadovoljuje moje razvojne potrebe in ker vidim ogromne možnosti razvojnega, ustvarjalnega dela. Toda, ali tako vidijo tudi poslovneži - managerji?

Omenjeni članek toplo pozdravljam in podpiram reševanje krize, ki jo predstavlja pomanjkanje strokovnjakov - informatikov. Želim pa dodati svoje izkustveno spoznanje: ne manjka le informatikov, manjka nam informacijske kulture, osveščenosti, da je informatika stroka, da se rezultati informacijskega znanja odražajo na poslovnem uspehu, da je naložba v informacijska znanja hkrati naložba v donosen posel - in da tega znanja ne more nadomestiti programer za PC-jem oziroma tajnica s poznavanjem Wordstara.

Morda sem groba v ocenah realnega stanja v naši regiji (žalski občini), vendar ugotavljam, da strokovnjaka - informatika v naših podjetjih ne rabijo in ga tudi niso pripravljeni plačevati (rezultatov njegovega dela se ne da že jutri otipati ali videti). Cenejši je občasni ali stalni programer, ki vsakokrat problem " na hitro" reši na svojem PC-ju (daje hitre in vidne rezultate, je učinkovit).

Kako sem prišla do takšnega spoznanja? Preprosto - iskala sem zaposlitev. Kjerkoli sem razložila, kaj ponujam kot delavec - informatik, sem hitro ugotovila, kako zelo me "ne potrebujejo". Kljub vsemu sem optimist, ker sem prepričana, da bodo zahteve poslovnega sistema po učinkovitem in celovitem informacijskem sistemu in informacijsko - organizacijskem znanju prerasle sedanji nivo razmišljanja naših managerjev. Zato bi morali ob študijskih programih informatike hkrati razmišljati tudi o kvalitetnih programih izobraževanja za uporabnike (niso to tečaji "Osnove uporabe PC-ja" ali " Osnove Wordstara" itd), takšnih, da bodo le-ti spoznali, kaj daje ta stroka, kako vpliva na poslovni sistem ali povedano v njihovem jeziku: kaj lahko pričakujejo kot rezultat dela informatika v svoji hiši, kolikšna je vrednost dobička (donosnost) od naložbe (njegove plače). Pokazati jim je potrebno kalkulacijo za naložbo v informacijsko - organizacijsko znanje.

Lep pozdrav

Tanja Kovač