ZVÝŠÍ PROTOTYPOVÉ PROGRAMY PRODUKTIVITU PROGRAMOVÁNÍ ?

Ing.Vojtěch Hruška - AVIA Praha

Výchozí předpoklad

Snaha, dosahovat stejných nebo raději lepších výsledků se stále nižším množstvím vynaložené práce, patří k základním lidským vlastnostem. Dokladem toho, že ani nám, programátorům tato snaha nechybí, je například pravidelné konání seminářů "Programování".

Ano, jsou i tací, kteří v podobné snaze spatřují především lenost. Ti nás zahrnují poučeními typu " jen trvalým zvýšeným úsilím", "napřít všechny síly" případně " a hlavně nepolevovat...". I těmito poučeními se lze řídit. K předčasné invaliditě či kremaci pak již zbývá jenom krok.

Ne,nejsem proti poctivé práci. Vážím si píle a pracovitosti. Naplňuje mne však hlubokým smutkem pohled na člověka, který po celodenní vysilující dřině odchází z pracoviště, výsledků po sobě nezanechav. Zvláště, jde-li o programátora.

Co jsme více či méně vyzkoušeli

V prehistorii programátorské profese "drátovali" pionýři našeho oboru programy nepevno do desky. Od té doby uplynulo několik desetiletí. Programátor dostal v průběhu této doby nepřeberné množství nástrojů, jež mu usnadňují práci. Jsou to například :

- MAT
- normované programování
- systémová analýza
- VS 9
- JSP a JSD
- CULPRIT
- VARS
- CP/M
- dBASE II
- EMPO
- WORDSTAR
- TURBO PASCAL
- MS DOS
- OVERSEER
- C
- COBOL
- OS 4
- DOS 1.7
- MARS
- DOS-3/ES
- BASIC
- Metodické pokyny k budování ASŘ
- LUISA
- ILYDIR
- POWER (DOS 1.7, CP/M, DOS-4)
- TLIOCS
- BATCHNET
- DUORS
- MOVE-IN-UP
- IQF, IUP

- NORTHON COMMANDER - IDMS DC
- LOTUS - INTERBAS atd.

Pozorný čtenář si jistě na tomto místě povzdechne: Co je to za nesmyslnou směť systémů, programů, metodik ... ? Ano, zcela úmyslně jsem uvedl tento malý vzorek prostředků. Jejich hlavním posláním je usnadňovat a zefektivňovat práci programátora. Výběr i pořadí komponent se může blížit tomu, jak se s nimi programátor v běžné praxi setkává.

Kromě operačních systémů, kompilátorů atp. obsahuje vzorek i některé rozsáhlé typové projekty. Nechci hodnotit jejich kvalitu a význam. Nasazením těchto typových projektů se měla výrazně snížit potřeba projekčních a programátorských kapacit v ČSSR. Zda se to podařilo, nechť posoudí čtenář sám.

Další rozsáhlou skupinou prostředků, jež mají především šetřit práci programátora, tvoří univerzální programy. Snad nebudu daleko od pravdy, prohlásím-li, že se už dnes sotva najde programátor, který by si sám programoval třídící algoritmus, běžné konverze datových souborů atp. Určitě by to bylo nesmyslné mrhání časem a energií. Běžně užíváme obecných parametrických či generačních programů.

Při stavbě takových programů však stále narážíme na zásadní problém. Univerzální program je efektivní, jednoduchý a snadno ovladatelný jen tehdy, plní-li poměrně úzce vymezenou funkci. Při každém pokusu o vytvoření mohutného obecného programu, schopného vykonávat širokou paletu funkcí nakonec zjistíme, že výsledkem je nestabilní a těžkopádné monstrum, k jehož ovládání musí uživatel nastudovat nový komplikovaný "programovací jazyk" či "ovládací systém". Protože podobné programy bývají obvykle zcela závislé na konkrétním typu a verzi operačního systému, zůstává, až na některé výjimky, skutečná míra jejich využití daleko za původními záměry autorů.

A jsme opět u onoho mnohokrát diskutovaného problému. I přes širokou paletu základního, typového a podpůrného programového vybavení, jehož význam nechci nikterak snižovat, se zatím bez běžných aplikačních programů neobejdeme. Při jejich tvorbě však neustále dochází k mrhání řešitelskými kapacitami. Další a další řešitelé promýšlejí tentýž problém, dopouštějí se podobných chyb, vytvářejí unikátní programová řešení, která se ve svých funkcích nakonec téměř shodují. A to mnohdy dokonce v jednom podniku, výpočetním středisku či v téže kanceláři. Přitom téměř vždy zahajují řešení nad čistým papírem, resp. před prázdnou obrazovkou.

Jako bych v tuto chvíli slyšel rozhořčené hlasy obětavých správců

národních či dokonce celostátních knihoven programů : Proč tedy lépe nevyužijete našich rozsáhlých programových fondů ?!?

Snad každý větší řešitelský tým již tuto cestu zkusil. Výsledky však vesměs nejsou povzbudivé. Proč? Není tak velkým problémem (jsou-li podmínky distribuce rozumné), převzít cizí aplikační programové vybavení. To pravé utrpení nastane až ve chvíli, kdy je potřeba do fungujícího komplexu provést první větší změnu. Většina pokusů o převzetí cizího APV pak končí obdobným způsobem. Z vlastní zkušenosti znám případ, kdy APV, zakoupené za sedmicifernou částku, fungovalo u uživatele jediný rok. Poté bylo vlastními kapacitami zcela přeprogramováno. Údržba a další rozvoj přejaté aplikace jsou většinou tak nesnadné a nákladné, že není jiné cesty.

Ve výjimečných případech nedochází k úplnému zavržení nakoupených programů. Nejsou-li zdrojové texty zcela změnyvzdorné, upravují a rozšiřují je řešitelé dle potřeb. Postupně tak vzniká vlastní APV, postavené na bázi původního přejatého p r o t o t y p u.

Jaké vlastnosti by měl mít dobrý prototyp ?

Prehistorickým příkladem velmi dobrého prototypového algoritmu je normované schema. Ne, rozhodně se nechci vracet k dávno překonané metodě. Pakterm však zůstává, že dodnes je tato metoda ojedinělá mírou sdělnosti, kterou zdrojovému textu zaručuje. Dokonce si dovolím tvrdit, že jen v případě normovaných programů se mohu spolehnout, že dva programátoři z různých výpočetních středisek, pracující na různých strojích v odlišných vyšších jazycích, jsou schopni se navzájem rychle zorientovat ve svých programech.

A to je také první požadavek. Dobrý prototyp musí mít takovou vypovídací schopnost o svých vlastnostech již na úrovni zdrojového textu, že nepotřebuje žádnou další rozsáhlou dokumentaci.

Pod pojmem p r o t o t y p o v ý p r o g r a m si představuji vzorové, zobecněné řešení opakovaně se vyskytujícího typu úlohy. Řešení je jako celek odladěno na vzorových datech a spolu s nimi distribuováno. Všechny části prototypu jsou současně k dispozici uživateli ve zdrojovém tvaru. Prototypový program je řešen modulárně. Ty části prototypu (moduly), do nichž bude uživatel zasahovat, musí být realizovány ve vyšším jazyce. Místa očekávaných úprav by měla být ve zdrojovém textu předem vyznačena. Je-li to možné, měl by být prototyp nezávislý na datech, s nimiž pracuje. Z požadavku na odladění prototypu plyne, že nelze očekávat implementační nezávislost.

Programátor, řešící problém tohoto typu, používá prototypu jako výchozího zdroje své aplikace. Nezahajuje tedy řešení nad čistým papírem. Vyhne se tak hrubým analytickým chybám, nezabývá se programováním funkcí, které jsou pro danou aplikaci standardní. Je-li prototyp ve středisku užíván opakovaně, může významně přispět ke sjednocení formy programů, primárních datových vstupů, tiskových výstupů i interakce. Především však podstatně zvýší produktivitu práce.

Zbývá tedy vyřešit poslední otázku. Lze vůbec nalézt takové typy úloh, jež umožní prototypové řešení ? Pěkným příkladem konkrétního prototypu je Tool Box, dodávaný firmou Borland International k Turbo Pascalu. Nabízí uživateli elegantní řešení malé databáze. Na mnoha našich pracovištích jsou jistě užívány další pěkné prototypy z domácí produkce. Dva vlastní pokusy o prototypová řešení se pokusím popsat v následujících kapitolách.

Prototyp aktualizačního programu

Program realizuje údržbu souboru s přímým přístupem v dávkovém i interakčním režimu práce. Variantní části prototypu (moduly) jsou psány v Cobolu, pevně definované funkce v Assembleru. Prototyp je odladěn v prostředí operačního systému DOS-4/EC, pro interakce užívá podpory transakčního monitoru DUORS (resp.DUORS/TM).

V aktualizovaném souboru lze provádět tyto typy změn :

DEL - fyzické zrušení věty kmenového souboru,

INS - vložení nové věty,

UPD - změna libovolného množství datových prvků ve větě, s výjimkou prvků, užívaných jako přístupový klíč věty,

UPK - změna všech datových prvků včetně klíčových /program fyzicky ruší původní větu a vkládá upravenou větu pod novým klíčem/,

DUP - odvození nové věty změnou klíče, původní věta zůstává zachována.

Z hlediska filosofie návrhu je program koncipován jako modulární stavebnice. Ze stavebnice jsou vytvořeny dvě verze aktualizačního programu, a to pro dávkovou a interakční údržbu. Většina použitých modulů je pro obě verze společná. Součástí stavebnice je i další dávkový program, který s využitím archivní kopie a žurnálu změn zajišťuje rekonstrukci kmenového souboru po havárii.

V dalších kapitolách stručně popíši funkce prototypu. Funkce, uvedené velkými písmeny, jsou realizovány pomocí samostatných

modulů. Moduly, označené (*), je nutno pro konkrétní aplikaci modifikovat.

Dávková aktualizace

ŘÍDÍCÍ MODUL (*)

Na první úrovni dekompozice se modul skládá ze vstupní a výstupní sekce sortu. Ve vstupní sekci :

- Zpracuje fakultativní řídící parametr. Pomocí parametru lze nastavit režim "ladění". V tomto režimu program poskytuje ladící výpisy (změnová věta, odpovídající kmenová věta, změny po konverzích a kontrolách, aktualizovaná kmenová věta). Vlastní I/O operace, zasahující do kmenového souboru jsou naopak zablokovány. Ladění programu tak může probíhat opakovaně, kmenový soubor zůstává neporušený.
- Zajistí částečnou nezávislost programu na datech. Té je dosaženo vytvořením vnitřních popisů datových struktur kmenové a změnových vět, získaných analýzou příslušných cobolských copy-struktur ze zdrojové knihovny. K tomuto účelu je užit modul

GENEROVÁNÍ VNITŘNÍ MAPY VĚTY

Modul analyzuje copy-strukturu kmenové věty a ve vymezené části virtuální paměti generuje vnitřní mapu. Jedná se o tabulku, v níž je každý datový prvek věty popsán těmito atributy :

- jméno prvku
- umístění ve větě (displacement)
- formát
- délka
- identifikace logické kontroly
- počet desetinných míst

Identifikace logické kontroly je převzata z komentáře prvku. Algoritmus analýzy předpokládá mírnou formalizaci copy-struktury, aby došlo k jednoznačné identifikaci elementárních datových prvků jejich jménem. Jména datových prvků jsou pak dále využita ke komunikaci (na obrazovce či tiskovém protokolu o údržbě) s uživatelem aktualizačního programu.

- Dále pokračuje řídící modul setříděním mapy podle jmen prvků (aby mohlo být při hledání prvku použito rychlého půlení intervalu).
- V cyklu čte soubor změn a setřídí jej dle přístupového klíče a kódu transakce. Setřídění změn v pořadí DEL, INS, UPD umožní

současné provedení těchto transakcí nad jedinou kmenovou větou.

Výstupní sekce sortu je koncipována jako normované schema s jedním stupněm autokontroly. Obsahuje klasické bloky B až G schematu :

B - odebírání vět změn ze sortu,

C - nevyužit,

D - autokontrola, v případě změny klíče ukončí zpracování předchozí skupiny blokem G

E - zpracování změnové věty v následujících krocích :

- Pokus o načtení kmenové věty, řízený standardním statutem:
  - 0 - věta dosud nebyla čtena,
  - 1 - věta již načtena,
  - 2 - věta daného klíče neexistuje.
- Formální kontroly náležitostí změnové věty jako celku. V případě chyby opis celé věty s příslušným komentářem a návrat do bloku B.
- Rozvětvení zpracování dle kódu transakce.
  - Rušení. Existuje-li kmenová věta (viz status), je zrušena a rušení je zaprotokolováno v tisku i žurnálu změn.
  - Vkládání. Existuje-li kmenová věta (předpoklad DUPKEV=NO), hlášení chyby a návrat do B. V opačném případě pokračuje zpracování konverzí změnové věty do standardního tvaru v modulu

    KONVERZE ZMĚNOVÝCH VĚT

    Modul nageneruje (s využitím předchozího modulu) vnitřní mapu změnové věty. Datové prvky, jejichž jména se shodují v mapě kmenové i změnové věty přenese do standardní struktury (tabulky změn). Ta obsahuje jména a obsahy datových prvků ze změnové věty. Do tabulky změn jsou pochopitelně ukládány pouze "vyplněné" datové prvky. Tabulka změn je při zpracování jednotlivých změnových vět na daný klíč postupně rozšiřována.
  - Změny. Neexistuje-li změnovaná věta, standardní hlášení. V ostatních případech pokračuje zpracování stejnou konverzí změn do tabulek, jako v případě vkládání.

Pozn.: Moduly pro tisk chybových zpráv a vytváření žurnálu změn, použité v bloku E budou popsány dále.

F - Ukončuje zpracování.

G - Blok autokontroly ukončuje zpracování skupiny změnových vět na daný přístupový klíč kmenového souboru. Veškeré zpracování zde

probíhá již nad standardizovanou strukturou - tabulkou změn. Ta obsahuje při vstupu do bloku všechny datové prvky, jež mají být v kmenové větě změněny (při INS vloženy). Další zpracování tabulky změn pak probíhá v následujících krocích :

DOPLNĚNÍ ATRIBUTŮ PRVKŮM V TABULCE ZMĚN

Modul doplní z vnitřní mapy kmenové věty atributy prvků v tabulce změn.

FORMÁLNÍ KONTROLY

Dle atributů formátu a délky provede modul formální kontroly datových prvků v tabulce změn. Nevyhovující prvky jsou v tabulce označeny kódem chyby. Bezchybné prvky jsou zkonvertovány do formátu (dekadický zhuštěný, binární), v němž figurují v kmenové větě.

LOGICKÉ KONTROLY (x)

Modul je souhrnem drobných podprogramů, jejichž úkolem je prověřit logickou správnost datových prvků v tabulce změn (číselníky, vzájemné vazby prvků, automatické generování změn v závislosti na přítomnosti či obsahu prvku a pod.). Podprogramy jsou sekvenčně číslovány. Číslo podprogramu logické kontroly poznamená programátor do komentáře deklarace prvku v copy-struktuře kmenové věty. Modul pak provede nad tabulkou změn příslušné logické kontroly. Výsledek zaznamená u jednotlivých prvků stejně, jako modul formálních kontrol. Je-li to zapotřebí, rozšíří tabulku změn o prvky automaticky generované.

PŘENOS ZMĚN Z TABULKY DO KMENOVÉ VĚTY

Bezchybné prvky z tabulky změn přenese modul do kmenové věty. V režimu "vkládání" se jedná o větu předem inicializovanou na základní hodnoty prvků.

ŽURNÁLOVÁNÍ ZMĚN

Modul zajišťuje žurnálování změn, provedených v kmenovém souboru. Vytváří větu proměnné délky. Pevná stanardní část věty obsahuje přístupový klíč, kód transakce, datum, čas a případné údaje o původci změny (jméno terminálu, LOGON jméno a heslo - má význam v interakci). Variabilní část věty pak obsahuje bezchybné prvky z tabulky změn.

TISK PROTOKOLU O ÚDRŽBĚ (x)

Do tohoto modulu je vkopírován standardní číselník chyb, jež lze dle potřeby rozšiřovat. Modul prochází tabulku změn a opíše v tiskovém protokolu jak vadné prvky s příslušným komentářem chyby, tak prvky promítnuté do kmenové věty.

Modul je vybaven dalšími vstupními body pro tisk informace o rušených větách a pro opis celých nezpracovatelných změnových vět.

Na závěr bloku G je v závislosti na kvalitě změn a kódu transakce proveden WRITE či REWRITE kmenové věty. Vnitřní struktury jsou inicializovány pro zpracování další skupiny vět.

Na první pohled se může zdát, že program je velmi rozsáhlý a zásahy do něho nebudou jednoduché. Co tedy musí programátor učinit, přeje-li si prototypu využít ?

V první řadě nadeklarovat struktury vět a uložit je na knihovnu. Editorem pak vyhledat v řídícím modulu řádky, označené " " (je jich 29). Zde upravit jména souborů,vět, délku přístupového klíče atp. V modulu logických kontrol nahradit vzorové podprogramy skutečnými požadovanými kontrolami (rozsah úprav je dán množstvím a složitostí plánovaných logických kontrol). Doplnit standardní číselník chyb hlášeními, jež mohou vzniknout jako následek vlastních logických kontrol. Upravit záhlaví tiskového protokolu o údržbě (opět v modulu vyznačeno " ").

Zbývá přeložit upravené moduly, spojit a spustit program v režimu "ladění". Pokud je doba odezvy snesitelná, stroj nepadá každou hodinu a neobtěžují nevítané návštěvy, lze toto vše včetně odladění zvládnout za jeden den.

Interakční verze

Tato verze programu je zitem DUORSU. Od dávkové verze se liší jen v maličkostech :

- Změnovou větou je zde obrazovka, na níž uživatel vyplnil nebo přepsal datové prvky. Modul pro vytváření tabulky změn je zde proto nahrazen modulem ANALÝZA OBRAZOVKY.
- Netiskne se protokol o údržbě. Standardní číselník chyb je vkopírován přímo do řídícího modulu. Chybové stavy jsou vyřizovány okamžitě, vrácením dat s příslušným komentářem chyby na obrazovku.

V prototypu řídícího modulu interakční aktualizace je nutno modifikovat 14 řádků zdrojového textu. Je-li již připraven modul logických kontrol, copy-struktura věty a standardní číselník chyb, mělo by být odvození vlastního aktualizačního programu otázkou několika minut. K tomuto času je samozřejmě nutno připočíst asi hodinu práce na návrh každé obrazovky, s níž bude program pracovat.

* * * * *

Výsledným produktem je v obou případech program v rozsahu cca 60 KB. Asi nebude mít význam využívat prototypu k údržbě školního souboru typu "telefonní seznam". V praxi podnikových ASŘ se však častěji objevují soubory, jejichž struktury obsahují desítky i stovky datových prvků. A každý z těchto souborů vyžaduje aktualizaci. Netroufám si ani odhadnout, kolik programátorů se již trápilo při stavbě toho "svého" aktualizačního programu, kolikrát byla vítězně odhalena tatáž logická chyba v algoritmu, kolik práce a času bylo vynaloženo.

Prototyp interakční uživatelské aplikace pro mikropočítač

Program je realizován v dBASE II, pod operačním systémem SCP (CP/M) na stroji R 1715. Na rozdíl od předchozího prototypu je daleko volnější, neboť nelze rozumně vytvořit šablonu, vyhovující libovolné aplikaci.

Základní rysy prototypu lze charakterizovat těmito vlastnostmi:

- Aplikace, realizovaná pomocí desítek programových modulů a dalších systémových komponent, se k uživateli chová jako celistvý komplex, ovládaný výhradně prostřednictvím menu, resp. vyžádaných odpovědí. Spouští se automaticky se studeným startem operačního systému.
- Aplikace je vybavena instalačním modulem. Jeho prostřednictvím je možno nastavit počáteční stavy a parametry úlohy.
- Úloha je chráněna proti neoprávněným manipulacím heslem.
- Na ochranu dat dohlíží automatický operátor. Při startu úlohy zjistí, zda minulé ukončení práce proběhlo legálně. Ne-li, prověří postupně všechny datové soubory. Je-li některý ze souborů narušen, pokusí se sám o jeho rekonstrukci. O všech prováděných akcích průběžně informuje uživatele. Jsou-li data po předchozí havárii nenávratně zničena, informuje uživatele o datu poslední archivace a o počtu změn, které byly od té doby provedeny v datové základně. Poté zajistí obnovu dat z poslední archivní kopie.
  Při startu úlohy dále operátor zkoumá, kdy byla provedena poslední archivace a kolik změn je od té doby promítnuto v datech. Je-li počet změn vyšší než X (instalovatelný parametr), pokusí se donutit uživatele k archivaci. Tutéž činnost provádí operátor i v okamžiku konce komunikace.
- Forma komunikace úlohy s uživatelem je zcela jednotná. Probíhá zásadně pomocí menu. Důležitá rozhodnutí musí uživatel opakovaně

potvrdit.

- První volba v každém menu je návrat v komunikaci. Tato volba je implicitní. Na druhém místě je vždy HELP. Poskytování nápovědy zajišťuje v rámci celého programu speciální modul. Pracuje s odděleným souborem textů, v němž každé úrovni menu odpovídá určitá část textu. Na první úrovni menu nabízí tento modul vytištění celého souboru. Kompletní vytištěný HELP tvoří ucelenou uživatelskou příručku úlohy, strukturovanou do kapitol, paragrafů, odstavců atd. tak, jak je vnitřně strukturována celá aplikace.
- V hlavním menu jsou pak k dispozici vlastní aplikační funkce. Jsou to vzorové funkce pro pořizování datové základny, její aktualizaci a presentaci dat jak na obrazovce, tak ve formě tisků.
- Prototyp dále obsahuje komplex standardních rekonstrukčních funkcí. Jedná se o archivaci, obnovu, čištění, případně reindexaci datové základny. Funkce jsou poskytovány prostřednictvím menu uživateli a současně je užívá i automatický operátor.
- Pro uživatele, jež se hlouběji zajímá o možnosti dBASE II je k dispozici jak interakční výuka, tak přímý přístup k funkcím dBASE II (ASSIST). Obě funkce komunikují s uživatelem česky.
- Prototyp zajišťuje rovněž některé základní statistiky o své činnosti. Autor aplikace má možnost se po čase přesvědčit, nakolik je programové vybavení využito, které funkce uživatel nepoužívá, resp. které funkce by bylo vhodné optimalizovat.
- V prototypu je vzhledem k omezením dBASE II zavedeno standardní značení paměťových proměnných. Při opouštění určité úrovně pak nečiní problémy uvolňovat proměnné tak, aby nedocházelo ke kolizím.

Jak už bylo řečeno v úvodu, jedná se o poměrně volný prototyp. Jeho přizpůsobení konkrétní aplikaci si vyžádá více času, než u předchozí stavebnice aktualizačního programu. Vlastní výkonné funkce se musí v podstatě programovat vždy znovu. Přesto byl i tento prototyp přijat čtenáři s povděkem. Je přece jen o mnoho snazší, zejména programují-li poprvé v dBASE II, vyjít již z hotového odladěného programu, než začínat od čisté obrazovky.

Z á v ě r

Neznám přesné statistiky. Dovolím si však odhadnout, že před pěti lety u nás připadalo v průměru 5 - 10 profesionálních programátorů na jeden počítač. Odmyslím-li domácí osmibitové hračky, ale

započtu-li samozřejmě i profesionální PC stroje, může být do pěti či deseti let poměr přesně obrácený.

Dosud jsme programovali ve velké většině faktografické informační systémy v dávkovém režimu. Nyní je ovšem požadován dialog, dokumentografické informační systémy, znalostní báze, grafika, řízení procesů v reálném čase. V nedalské budoucnosti budeme vymýšlet algoritmy pro dokonalé zpracování obrazu, expertní systémy, analýzu a syntézu mluvené řeči, umělou inteligenci.

Nevěřím, že se v krátké době podaří řádově zvýšit počty kvalifikovaných programátorů. Rovněž se mi zdá nepravděpodobné, že široká uživatelská veřejnost v historicky krátké době zvládne prostředky typu SUPERCALC, IDMS DML, LOTUS, dBASE IV atd. Asi skutečně nezbude nic jiného, že že se naučíme pracovat méně, přitom však vyprodukovat daleko více. Snad nám v tom alespoň pomůže trochu i programování na bázi prototypů.