Objektové programování v Cobolu pomocí OMS

Vlastimil Čevela

Úvod

Obsahem předkládaného příspěvku jsou některé poznatky z více než roční práce s Objektově Modulární Stavebnicí (OMS), která rozšiřuje možnosti běžných kompilátorů jazyka Cobol o nástroje, umožňující objektové programování.

Příspěvek přímo navazuje na popis Objektově orientovaného generátoru programů Cobol ve sborníku Programování '93.

Objektově modulární stavebnice Cobol s komfortním vývojovým prostředím VPC je realizována na platformě PC/MS DOS. Při využití vhodných komunikačních prostředků pro přenosy zdrojových textů ke kompilátoru a protokolů o překladu zpět je však možno ji využívat i pro jakouliv jinou platformu, kterou lze takto s PC propojit.

Vzhledem k tomu, že vlastní generátor je napsán v Cobolu, počítá se dle zájmu s jeho přenosem i na další HW a SW platformy. V současné době je ověřována možnost práce přímo v prostředí SINIX a Siemens BS200 a uvažuje se též s verzí MF/.GNT.

1. Objektově modulární stavebnice - (Object Modular Supplement)

Hlavní části OMS Cobol

- jazykové prvky, rozšiřující syntaxi jazyka Cobol
- objektově orientovaný generátor programů
- dokumentační pravidla pro definování rozhraní modulů
- knihovna typových modulů - závisí na dialektu Cobol
- podmínky a nástroje integrace do vývojového prostředí

Principem Objektově modulární stavebnice je skládání výsledného programu z tzv. objektových modulů, které je možno vícenásobně používat, modifikovat a integrovat do libovolných celků.

Skládání výsledného programu ze stavebnice modulů, realizované generátorem programů, je řízeno pomocí parametrů (kdekoliv v textu) a řídících kódů (2-znakové kombinace od 1. sloupce, následované mezerou). V rámci jednoho programu může být až 999 úrovní odkazů na dílčí moduly.

Dokumentační pravidla pro definování rozhraní modulů pomocí jednoznakových dekompozičních kódů formalizují zadávání identifikačních přípon (N) a ostatních parametrů pro generátor (G), předávání dat při práci programu (I,O,W) a předávání řízení při práci programu (A,P,R,C,S) - viz příklad dále.

Pro kompilátory Cobol dialektu Microsoft - Micro Focus je k dispozici knihovna, obsahující několik desítek typových modulů. Bližší informace k oblastem využití takových typů, které je možno vytvořit i pro jiné dialekty, jsou uvedeny dále.

Úspěšná realizace systému OMS vyžaduje zajištění návazností na vývojové prostředí, obsahující editor a vlastní kompilátor Cobol. V současné době je spolehlivě vyřešena integrace do vlastního vývojového prostředí VPC 2.0 na platformě MS DOS a ověřena spolupráce s kompilátory MS Cobol a MX Cobol.

Vzhledem k tomu, že na různých platformách je vývojové prostředí rozdílné, je třeba podmínky a případné doplňující nástroje integrace definovat pro každé prostředí individuálně.

2. Jazykové prvky OMS, rozšiřující syntaxi jazyka Cobol

Definice proměnných

Definice proměnných prvků umožňují parametrizaci zdrojového textu modulu. Jednoduché formální parametry %0 až %40 se využívají obdobně, jako %0 až %9 v souborech ".bat" a přípona "-%" slouží k definování lokálních jmén, která nejsou mimo modul přístupná. Pomocí řídících kódů pak je možno vymezit modifikovatelné, volitelné a komentářové řádky a pásmo volných deklarací.

Záhlaví blokové struktury

Řídící kódy, definující záhlaví oddílů a sekcí umožňují blokovou strukturu zdrojového programu, tj. libovolné střídání deklarací dat a příkazů procedur pro jejich zpracování.

Příkazy pro generování

Základními příkazy pro generování je parametrizovatelný odkaz na dílčí modul a odkaz na modul standardních záhlaví. Dále pak jsou to příkazy k vytvoření pracovního modulu a definice dat, ukončení opakovacího modulu a příkazy ke zrušení, náhradě a výběru vymezených řádků.

3. Oblasti využití technologie OMS pro tvorbu typů

Na základě dosavadních zkušeností lze uvést některé třídy algoritmů, pro které lze pomocí OMS vytvářet užitečné typové moduly.

Zprostředkování volání programů a podprogramů

Při využívání sytémových, standardních či aplikačních podprogramů i při spolupráci s databankovým nebo jiným programovým prostředím je často potřeba naplňovat různě složitá pole parametrů a přesně dodržovat formáty popisu dat i volání. Typové moduly se výhodně uplatňují jako mezivrstva, která odstíní detaily a celkově zjednoduší zápis programu.

Opakování textu s 1 proměnnou

Příkladem je řádková definice datové struktury dle slovníku dat:

```
d.& 2 UDAJ1 UDAJ2 UDAJ3 #&
```

Realizace algoritmu, který vrací řízení

Typickým případem je dále uvedený příklad typového modulu "precti.m".

Realizace algoritmu, který vrací hodnotu

Pomocí parametrizovatelného odkazu na typový modul lze do programu zařadit předem naprogramovanou a prakticky libovolně složitou výpočtovou funkci.

Provedení určité akce

Typickým příkladem akce je např. zobrazení okna se statickým textem, ale může to být i prohlížení anebo editace či aktualizace souboru v okně apod.

V konkretních případech mohou být výše uvedené algoritmy dle potřeby libovolně kombinovány. Např. provedení akce "nabidkové okno" může současně vracet řízení do odstavců dle provedeného výběru.

4. Dělba práce a zmenšování složitosti programů

Jedním ze silných hnacích motorů technického pokroku je dělba práce. V programátorském řemesle, které je založeno především na duševní tvůrčí činnosti je však uplatnění dělby práce velkým problémem.

Způsob myšlení každého člověka je různý, fakta si často spojuje do zcela jiných asociací a v neposledni řadě se uplatňuje též rozdílná úroveň znalostí a schopnosti abstrakce.

Při prezentaci objektového programování se autoři většinou zaměřují na výhody opakovaného využívání jednou vytvořených objektů. Rovněž v dokumentaci konkretních programovacích nástrojů se klade důraz především na práci s knihovnami typových objektů.

Ve výše uvedených souvislostech pak nezaslouženě zůstává stranou fakt, že objektový přístup může výrazně přispět k řešení problému dělby práce.

I jednoduchá metoda OMS Cobol pro definování rozhraní je například velice vhodná nejen na popis typových modulů, ale také na stručné a přitom zcela jednoznačné vymezení dílčích částí v rámci většího celku.

Použitím dekompozičních kódů OMS, které se doslova dají spočítat na prstech lze totiž nejen uvést přehled všech použitých parametrů pro generování výsledných modulů, ale i stanovit naprosto přesná pravidla pro předávání dat i řízení mezi modulem a programem, který jej využívá.

Dosavadní zkušenosti ukázaly, že definování rozhraní dle pravidel OMS umožňuje rychle a stručně zapsat vše, co je podstatné k tomu, aby se složitý program rozdělil mezi paralelně pracující řešitele.

Je vhodné zdůraznit, že takto lze rozdělit práci na tvorbě jakéhokoliv zdrojového textu. A každou část je možno řešit i odladit a ověřit jako úplně zapouzdřenou černou skříňku při prakticky zcela volitelném počtu řídicích i datových vazeb.

Z ověřené praxe lze dále uvést, že rozdělení složitého problému na několik jednodušších je velice výhodné i při individuálním programování. Programátor totiž může řeit úlohu po částech a výsledkem je kvalitnější a udržovatelnější produkt.

Poněvadž skládání prvků OMS je realizováno na úrovni zdrojových textů Cobol, bez problémů lze realizovat nejen obousměrné předávání dat mezi moduly, ale též obousměrné předávání řízení.

Použitá metoda je prakticky triviální - příkaz "perform" v případné kombinaci s nastavováním nebo vyhodnocováním podmínkových jmen, ale výsledek je velice užitečný. Princip si ukážeme na dále uvedeném příkladu.

Příklad ". precti.m":

Předpokládejme, že máme v zadání programu kromě jiného též potřebu sekvenčně přečíst soubor a zpracovat jeho jednotlivé věty.

Pro tento dílčí úkol můžeme vytvořit typový modul, který bude realizovat mechanizmus testu na existenci souboru, otevření souboru, cyklus čtení po větách a uzavření souboru.

Začlenění typového modulu do programu zajistíme odkazem s definováním potřebné identifikační přípony:

". precti.m I"

V rámci hlavního modulu programu pak na příslušném místě naplníme datovou položku modulu "JM-I" jménem souboru a příkazem "perform PRECTI-I" vyžádáme od modulu provedení akce přečtení souboru.

Ke zpracování každé jednotlivé věty pak stačí, abychom v hlavním programu definovali odstavec "VETA-I.", který si modul ve vhodnou chvíli příkazem "perform VETA-I" vyvolá, a ve kterém pak máme možnost libovolně pracovat s datovou položkou "VT-I" s právě přečtenými daty.

Test na podmínkové jméno "PRVNI-I" umožňuje individuální zpracování první věty a případný příkaz "set KONEC-I to true" pak předčasné ukončení čtení.

Definice rozhraní pro uvedený typový modul "precti.m" bude vypadat následovně:

formát odkazu na modul

precti.m %1

vlastní rozhraní

N identifikační přípona	%1
I údaj pro zadání jména souboru	JM-%1
O přečtená věta	VT-%1
A sekvenční přečtení souboru	PRECTI-%1
R předání přečtené věty	VETA-%1
C signalizace o stavu čtení	
/ přečtená věta je první větou	PRVNI-%1
/ další věty	DALSI-%1
S přepínač předčasného ukončení čtení	KONEC-%1

5. Cobol, OMS a neprocedurální jazyky

V posledních letech se intenzívně prosazují programovací prostředky, ve kterých procedurální stránka ustupuje do pozadí. Nemíním s tímto trendem polemizovat, pouze se domnívám, že i za této situace má procedurální programování své významné místo v sadě metod pro tvorbu programů.

Je samozřejmé, že pro určité třídy úloh je daleko efektivnější využít jednoduchý neprocedurální příkaz, anebo interaktivně vytvořit právě potřebnou variantu aplikace.

Kromě toho však musí existovat programátoři, kteří takové snadno přizpůsobitelné programové systémy navrhují, a ti se bez procedurálního programování neobejdou. V neposlední řadě pak ale existují další třídy úloh, pro které je přesné, na míru šité procedurální řešení výhodné.

Jako důkaz správnosti výše uvedených úvah lze uvést, že sice existují zástupy aplikačních programů, napsaných v jazycích typu DBase, Foxbase nebo Magic, ale tato skutečnost vůbec neznamená výraznější pokles zájmu o programování v jazycích Pascal nebo C.

Předmětem našeho zájmu je dnes již klasický jazyk Cobol. Jeho typickou vlastností je, že byl navržen se snahou o maximální přiblížení k syntaxi běžného anglického jazyka. A právě tato charakteristika z něj činí možná jeden z nejdokonalejších programátorských nástrojů pro popis procedury provádění algoritmů.

Současné programování už dávno není "tajemnou magií", ale pragmatickou výrobou "součástek pro informační technologie". Na rozdíl od technického vybavení počítačů a jiných hromadně, stavebnicovým způsobem, vyráběných výrobků však podobná "obecně modulární" výroba programů má k všeobecnému zvládnutí zatím ještě hodně daleko.

Objektově Modulární Stavebnice Cobol je drobným příspěvkem který se pokouší výše uvedené přístupy kombinovat. Umožňuje totiž nejen úplné zapouzdření objektů, ale jazykové prvky jsou záměrně navrženy tak, aby nepotlačovaly vynikající procedurální vlastnosti jazyka Cobol, ale naopak je dále co nejpřirozenějším způsobem rozvíjely - především pomocí příkazů "perform".

Dle mého osobního názoru je totiž procedurální myšlení většině programátorů bližší. Mám dojem, že motivace tvořit mechanizmus "fungování" je výrazně silnější, než motivace pro pouhé "skládání". S tím možná souvisí i obtížnější prosazování technologií, založených výhradně na "černých skříňkách".

Aby mi bylo dobře rozuměno - princip neprocedurálního programování neodmítám. Pouze si myslím, že určitý prostor pro uplatnění tvůrčí invence programátora při návrhu řízení zpracování, tj. procedurální části programu, je velice důležitou podmínkou pro jeho motivaci. A lidský faktor patří určitě mezi největší rezervy na cestě ke zvyšování kvality a produktivity při tvorbě programového díla.

Autor:

Ing. Vlastimil Čevela
Benešova 279
664 42 Modřice
tel. 05 - 303 367