JACKSONOVO STRUKTUROVANÉ PROGRAMOVÁNÍ A MODULARIZACE PŘI TVORBĚ PROGRAMU V DATABANKOVÉM PROSTŘEDÍ

Michal Kretschmer, prom. mat., MÚZO, Praha

1. Úvod

Následující úvahy jsou shrnutím a zobecněním zkušeností autora tohoto příspěvku s tvorbou velmi rozsáhlého programu provádějícího aktualizaci databanky. Základním vstupním sekvenčním souborem byl soubor jednotlivých požadavků na změny databanky, přičemž existoval větší počet různých typů těchto změn /transakcí/. Při realizaci tohoto programu byl učiněn pokus využívat jak Jacksonovy metody strukturovaného programování /v dalším JSP/, tak i budovat program modulárním způsobem v duchu zásad Wirthsovy koncepce souběžného návrhu. Navíc pak bylo nutné zohlednit specifiky databankového prostředí. V dalším předpokládám, že čtenář je seznámen se základními ideami obou těchto přístupů, minimálně v rozsahu, ve kterém byly již dříve presentovány na semináři Programování [illegible] /2/.

2. Databankové prostředí z pohledu JSP

JSP definuje programovou strukturu z datových struktur sekvenčních souborů. Soubory s přímým přístupem se pak prohlížejí pouze do jednotlivých selekcí následujících za operací přístupu k těmto souborům /např. po operaci čti dle klíče bude selekce rozlišující případ "věta nalezena" a případ "věta nenalezena"/. Bližší analýza JSP ukazuje, že zde není podstatná organizace souboru, ale způsob jeho zpracování, takže např. při odvozování programové struktury není rozdíl mezi sekvenčním souborem a sekvenčně zpracovávaným index-sekvenčním souborem.

Databanka, tak jak ji známe z definice CODASYL, může sice představovat z pohledu operačního systému fyzicky jediný soubor, avšak z pohledu zpracování obsahuje různé typy vět s možností přímého přístupu a skupiny spolu propojených vět, tzv. sety. Z pohledu JSP je sekvenční čtení setu od aktuální /current/ věty do konce setu ekvivalentní čtení sekvenčního souboru. Totéž platí i o čtení všech vět /event. vět určitého typu/ v rámci jedné oblasti /area/ databanky. Naproti tomu čtení nebo ukládání věty v CALC módu nebo čtení či ukládání věty podle databázového klíče je přímým přístupem k datům. Z tohoto pohledu nevyžaduje tedy práce v databankovém prostředí nic nového při tvorbě programové struktury.

Významný rozdíl mezi zpracováním sekvenčního souboru a zpracováním setu spočívá však v tom, že zatímco pro sekvenční soubor je první instrukce čtení prováděna na začátku programu bezprostředně po instrukci otevření souboru, pro set jednak odpovídající instrukce otevření neexistuje /napojení databanky na operační systém nebo otvírání oblastí má jiný charakter, který není přímo spojen s konkrétním setem/, jednak čtení první věty setu je odlišné od čtení dalších vět setu. K první větě setu se totiž dostáváme jinou přístupovou cestou než k dalším větám setu, např. čtením vlastníka /owner/ setu v CALC modu, nebo přístupem z jiného setu. Toto postupné vyhledávání vět v databance bývá též nazýváno navigací /3/. Z toho vyplývá, že datová struktura setu /v nejjednodušším a nejčastějším případě jako iterace vět setu/ se v programové struktuře zohlední teprve až po vyhledání první zpracovávané věty setu. Programová struktura zde musí vždy být též odrazem volby přístupové cesty k větám.

3. JSP a modularizace

Důvody pro rozdělení programu do modulů mohou být dvojí:

a/ zvýšení spolehlivosti programu, tj. sledujeme takové cíle jako jsou systematické zjednodušování programu, jeho zpřehlednění a tím i zlepšení jeho čitelnosti; usnadnění jeho prov ení a otestování; zvýšení jeho přenositelnosti na jiné operační sy ; zvýšení jeho adaptibilního potenciálu; oddělení opakovatelně použitelných relativně nezávislých obecnějších funkcí /budování knihovny společných programových prvků/; oddělení částí závislých na operačních systémech; zkrácení nutných časů pro kompilace; zlepšení předpokladů pro efektivní organizaci týmové práce;

b/ zprovoznění programu s příliš velkými paměťovými nároky, tj. chceme optimalizovat využití přidělené paměťové kapacity pomocí techniky překrývání segmentů.

Pod pojmem modul rozumíme takovou souvislou část programové struktury s přiřazenými operacemi, která je samostatně kompilovatelná a která je

a) z hiearchického hlediska relativně samostatnou elementární programovou operací /tj. nemusí nutně souhlasit s některou řídící strukturou/;

b) z programového hlediska je relativně uzavřená jednotka - má počátek a konec; může vlastnit lokální data; styk s globálními daty se děje pouze prostřednictvím přesně definovaného interfacu /nejlépe prostřednictvím parametrů/, tj. modul má charakter programu, podprogramu nebo korutiny, jehož struktura a logika je určena vstupními a výstupními sekvenčními daty s požadovanou funkčností; může vytvářet provozuschopný cílový modul, který může být buďto samostatně přímo prováděn nebo

volán v určitém kontextu z jiného modulu;

c) z logického hlediska je to takové seskupení programových instrukcí, které navzájem spolu logicky souvisí tím, že řeší /alespoň do jisté hloubky/ určitou programovou funkci.

JSP se primárně zabývá způsobem tvorby programové struktury, pravidla pro kódování jednotlivých struktur a technika programové inverze jsou pro tuto metodu jen něčím doplňujícím. Výsledkem návrhu programu pomocí metodiky JSP může být velmi rozsáhlá programová struktura s přiřazenými operacemi. Z důvodu přehlednosti mohou být některé komponenty dále rozpracovány zvlášť na samostatných listech. Přitom však sledovaným cílem je přehlednost grafického znázornění struktury programu, nikoliv návrh rozdělení programu do modulů. Tím se vůbec JSP nezabývá, neboť hlavním záměrem této metodiky je zachytit programovou logiku neboli vlastně dynamické chování programu statickou formou. Kódování a také modularizace spadají až do fáze realizace logiky programu /určené jeho strukturou a přiřazenými operacemi/ v příslušném programovém jazyce. Provádění modularizace může též spadat i do fáze optimalizace programu, pokud totiž některé opakující se části programové struktury budou zakódovány jen jednou a na příslušných místech bude tento společný kód pouze volán.

Elementární operací je v JSP taková operace, která na dané úrovni abstrakce již nemá podřazenou strukturu. Z tohoto pohledu je volání modulu elementární operací, jejímž sémantickým obsahem je "proveď určitou funkci". Modul pak představuje realizaci této elementární operace. Modul pracuje s daty, jimiž jsou jednak jeho parametry, jednak data, která tento modul čte a vytváří. Nic nebrání tomu, aby programová struktura modulu byla vyjádřena strukturním diagramem dle metodiky JSP. V programové struktuře modulu, která je podprogramem /nikoliv tedy hlavním řídícím modulem programu nebo korutinou/, se promítnou ovšem jen ta data, která jsou zpracovávána na jedno jeho vyvolání. V programové struktuře modulu, která je korutinou, se promítne mezisoubor jako iterace vět, a to bez ohledu na ten případ, že korutina je invertována vůči tomuto mezisouboru a že na jedno vyvolání zpracovává vždy jednu větu mezisouboru.

Při provádění modularizace musí být dodržena zásada, že jestliže modul obsahuje začátek některé části programové struktury, musí obsahovat v sobě kód pro provedení celé této struktury. To znamená, že jestliže zdrojový text modulu obsahuje úvodní IF selekce nebo iterace, musí obsahovat i její koncové návěští; jinak by byla narušena integrita modulu, neboť by byl požadován skok mimo tento modul. Podobně invertovaná korutina musí obsahovat všechna Qi návěští, jestliže nemá dojít ke komplikovaným konstrukcím, které nejsou v souladu se základními záměry strukturova-

ného programování.

Problém vzniká při quitu, který může směřovat k nadřazené komponentě mimo rozsah modulu. Tento případ je nutné řešit tím, že k parametrům modulu je přidán quit-indikátor, který na výstupu z modulu umožňuje rozlišit, zda má či nemá být příslušný quit proveden. Vlastní test indikátoru a tedy i quit je proveden v nadřazeném modulu bezprostředně za vyvoláním modulu nižší úrovně. Tento způsob kódování quitu nevede k neřízeným skokovým příkazům. Quit-indikátor má čistě místní použití: je plněn jednak na konci posit, jednak na konci admit části volaného modulu a testován pouze bezprostředně za vyvoláním modulu.

Menší odchylkou od standardního způsobu kódování je případ, kdy invertovaná korutina je používána opakovaně pro sekvenční zpracování dat /např. setu/ tak, že po skončení zpracování jedněch těchto sekvenčních dat budou od začátku zpracovávána jiná sekvenční data. V tomto případě musí být při dosažení konce dat programem obnovena stavová proměnná QS na počáteční hodnotu 1.

4. Databankové prostředí z pohledu modularizace a kódování programu

Ze znaků databankového prostředí, které je předpokládáno v dalších úvahách, jsou význačné následující:

Instrukce pro práci s databankou jsou překládány /ať již přímo nebo processorem/ jako CALL určitého firemního modulu s parametry, kterými jsou funkční kód, jméno věty databanky, jméno setu databanky, jméno oblasti databanky apod.

Existuje instrukce prvotního připojení se k firemnímu programovému vybavení pro obsluhu databanky. Úspěšné provedení této instrukce je nezbytným předpokladem pro to, aby mohly být úspěšně provedeny další instrukce pro práci s databankou.

Jednotlivé instrukce pro práci s databankou předávají po jejich provedení informace v komunikační oblasti, jejíž adresa je jedním z parametrů předávaných při prvotním připojení se k programovému vybavení pro obsluhu databanky. Tato komunikační oblast obsahuje mimo jiné informaci o chybových a vyjímečných stavech, databázový klíč vyhledané nebo uložené věty a jméno přečtené věty. Při chybě není tedy řízení automaticky předáno na nějakou rutinu pro její zpracování, nýbrž je nutné po každé instrukci pro práci s databankou testovat její úspěšnost, popř. přípustné vyjímečné stavy /obvykle jen konec setu nebo oblasti, věta nenalezena, duplicitní klíč/. V případě nepřijatelných chybových stavů je třeba začlenit do programu výpis zprávy o této chybě a ukončit zpracování.

Bezprostředně po připojení se k programovému vybavení pro obsluhu databanky jsou též předány adresy jednotlivých vět databanky, na kterých jsou tyto věty k dispozici po jejich přečtení nebo na kterých je nutno vytvořit větu určenou k zápisu. Každému typu věty je tedy přiřazena jediná adresa paměti, kterou nelze změnit až do odpojení se od programového vybavení pro obsluhu databanky.

Pro úspěšné provedení některých instrukcí pracujících s databankou je nutné splnění některých předpokladů. Tím je jednak správné naplnění některých polí v programu /např. klíče věty čtené v CALC modu/, jednak zajištění správnosti nastavení příslušné aktuální věty /currency/. Obrácené provedení databankových instrukcí má vliv na nastavení příslušných aktuálních vět v rámci typu vět, setů, oblastí a programu.

Z této dílčí charakteristiky databankového prostředí lze navrhnout některá doporučení týkající se rozčlenění programu do modulů, předávání parametrů, specifikace modul interfacu a kódování ošetření vyjímečných a chybových stavů.

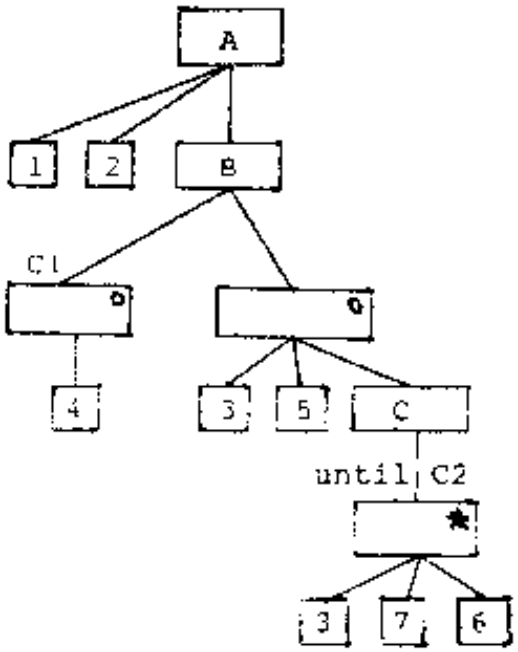
Instrukce prvotního připojení k firemnímu programovému vybavení pro obsluhu databanky je uvedena v hlavním /top/ modulu programu. V tomto modulu jsou též vymezeny oblasti pro jednotlivé typy vět subschematu, komunikační oblast a tabulky konstant obsahujících jména vět, setů a oblastí a funkční kódy.

Tyto oblasti a tabulky konstant jsou v programu vymezeny jen jednou. Pokud podřízené moduly některé z nich potřebují pro svou činnost, jsou jim předávány jako parametry. Každý podřízený modul obdrží jako parametr jen oblasti pro ty typy vět, se kterými pracuje buďto on sám nebo některý hlearchicky jemu podřízený modul. Podobně obdrží pouze ty tabulky konstant, které potřebuje. Komunikační oblast je předávána každému modulu, který pracuje s databankou. Struktury těchto předávaných parametrů jsou v programu definovány způsobem odpovídajícím používanému programovacímu jazyku /v LINKAGE SECTION v Cobolu, v DSECT v Assembleru/. Pokud firemní programové vybavení nezajišťuje možnost takovéhoto předávání parametrů generuje vždy celé subschema, tabulky konstant generuje do WORKING -STORAGE SECTION/, lze vypracovat vlastní program, který provede před kompilací potřebnou úpravu zdrojového textu modulu.

Po každé instrukci pracující s databankou jsou nejprve otestovány /pokud existují/ přípustné vyjímečné stavy a poté je vyvolán modul, který zjišťuje, zda došlo k nějakému vyjímečnému nebo nepřípustnému chybovému stavu. Tento modul soustřeďuje tedy všechny potřebné standardní testy a jestliže zjistí jiný stav než úspěšné provedení instrukce, vypíše zprá-

vu obsahující kód chyby, typ věty, oblast a set vztahující se k této chybě a odpojí se od programového vybavení pro obsluhu databanky s automatickou obnovou /rollback/ databanky k poslednímu kontrolnímu bodu /checkpoint/. Níže na obr. 1 je uveden příklad programové struktury vyjádřené strukturním diagramem s přiřazenými operacemi a jejího zakódování obsahujícího též ošetření očekávaných vyjímečných stavů v selekci a iteraci.

Programová struktura:



Seznam operací:

1. naplň klíč věty R1
2. čti větu R1 dle klíče
3. testuj možnou chybu při práci s databankou
4. zpracuj případ věta R1 nenalezena
5. čti první větu R2 setu S1
6. čti další větu R2 setu S1
7. zpracuj přečtenou větu S2

Seznam podmínek:

C1 věta R1 nenalezena
C2 konec setu S1

zdrojový kód:

```
A-SEQ.
    MOVE K TO KLIC OF R1.
    FETCH R1 RECORD.
B-SIC.
    IF NOT DMS-RECD-NFD
            GO TO B-OR1.
*   VETA R1 NENALEZENA
    .
    .
    .
    GO TO B-END.
B-OR1.
    CALL 'DMSSTA' USING SUBSCHEMA-DMCA
                        IDBMSCOM-TABLE.
*   VETA R1 USPESNE PRECTENA
    FETCH FIRST R2 RECORD OF S1 SET.
```

```
C-ITR.
    IF DMS-END-SET
            GO TO C-END.
    CALL 'DMSSTA' USING SUBSCHEMA-DMCA
                        IDBMSCOM-TABLE.
*   VETA R2 USPESNE PRECTENA
    .
    .
    .
    FETCH NEXT R2 RECORD OF S1 SET.
    GO TO C-ITR.
C-END.
B-END.
A-END.
```

Vysvětlivky:

DMS-RECD-NFD a DMS-END-SET jsou podmínková jména pro stavy věta nenalezena, resp. konec setu

SUBSCHEMA-DMCA je jméno komunikační oblasti

IDBMSCOM-TABLE je jméno tabulky funkčních kódů

DMSSTA je jméno modulu pro test chybových a vyjímečných stavů při práci s databankou

obr. 1

Pro použití modulu je nutné znát předpoklady, za kterých může úspěšně vykonat svoji činnost. Mezi ně náleží i předpoklady týkající se práce s databankou, zejména pak nastavení aktuálních vět, otevření oblastí, existence či neexistence věty o určitém klíči, naplnění pole databázovým klíčem věty, což jsou často předpoklady, které nejsou zachyceny v předávaných parametrech. Tyto předpoklady je proto třeba uvést v modul interfacu. Podobně je významné znát možný vliv provedení modulu na databanku a s tím spojené nastavení aktuálních vět v rámci typů věty, setů, oblastí a programu. Rovněž tyto skutečnosti je třeba uvést v modul interfacu v oddíle vnější efekty.

Výše uvedený způsob modulární výstavby programu pracujícího s databankou není jediný možný; je však v souladu s požadavky uvedenými v práci Myerse /4/. Za zmínku stojí uvažovat možnost, že by všechny instrukce pracující s databankou byly soustředěny v jediném modulu, takže ostatní moduly by obsahovaly volání tohoto modulu styku s databankou s příslušnými parametry. Diskuse tohoto pojetí však překračuje rámec tohoto příspěvku.

5. JSP a koncepce souhrného návrhu

Myersova koncepce souhrného návrhu popsaná v /4/ vychází z ideje modulu jako základní jednotky programové struktury, rozlišuje mezi funkcí a logikou modulu, uvažuje hierarchii a nezávislost modulů, zavádí pojmy soudržnost modulu /module strength/ a spřaženost modulů /module coupling/ s cílem navrhnout program tak, aby jeho moduly měly maximální soudržnost a minimální spřaženost, přičemž určité době uvážené kompromisy jsou nezbytné. Tyto ideje, jak vyplývá již z předchozího textu, jsou slučitelné s JSP s vyjímkou pojetí modulu jako základní jednotky programové struktury. Pro JSP je prvním cílem odvodit programovou strukturu a tím i logiku programu z datových struktur a zadání programu; poté teprve následuje jeho realizace, pro kterou je u řady programů účelné jejich rozdělení do modulů o takových vlastnostech, jak uvádí Myers.

Myers uvádí též postup, jak dospět k optimálnímu návrhu modulárního programu. Používá k tomu STS, transakční a funkční dekomposici.

Technika dekomposice STS je aplikovatelná tam, kde problém může být znázorněn jako pevná sekvence dílčích problémů, které jsou spolu svázány tokem dat. Přitom dochází k postupné transformaci vstupních dat až k bodu nejvyšší abstrakce, kdy hlavní vstupní proud existuje naposledy jako logická entita. Podobně zpětným postupem od výstupu lze najít druhý bod nejvyšší abstrakce, kde poprvé existuje výstupní tok. Modul se tedy rozpadá na tři bezprostředně podřízené moduly: vstup, transformace, výstup. Jestliže tento problém budeme řešit pomocí JSP, zjistíme v řadě případů, že mezi strukturami vstupních a výstupních dat existuje rozpor nebo že je účelné je od sebe oddělit. K tomu slouží právě pojetí mezisouborů a programové inverze. Transformační modul bude tedy jako korutinu volat invertovanou korutinu s funkcí "čti vstup a dodej jej v interním transformovaném tvaru jako vstup pro ústřední transformaci" a invertovanou korutinu s funkcí "převezmi v interním transformovaném tvaru výstup ústřední transformace, převeď jej do cílového tvaru a zapiš". Bodům nejvyšší abstrakce odpovídají tedy oba mezisoubory. Jestliže v řešeném problému chybí vstupní nebo výstupní transformace, odpadne i příslušná korutina. Vidíme tedy zřejmou analogii a slučitelnost STS dekomposice a programové inverse, i když svými východisky se liší: STS dekomposice vychází z toku dat, JSP ze struktur dat.

Transakční dekomposice se užívá tam, kde modul se rozpadá do dílčích funkcí v závislosti na vstupních datech. Z hlediska JSP se tedy jedná o selekci, přičemž volání podřízených modulů můžeme chápat jako elementár-

ní operace na úrovni tohoto dekomponovaného modulu.

Funkční dekomposice je rovněž slučitelná s JSP; jedná se o realizaci funkce jako elementární operace samostatným modulem. Vytváření modulů s informační soudržností, které zahrnují několik funkcí pracujících s týmiž daty je vhodné a neodporuje JSP, neboť tyto funkce jsou z hlediska jejich provádění nezávislé a souvisí spolu pouze společnými daty, jejichž strukturu v řadě případů skrývají před nadřazenými moduly.

Zásadní rozdíl mezi JSP a koncepcí souhrného návrhu spočívá v základním přístupu: JSP vychází ze struktury dat a směřuje k zachycení logiky programu statickou programovou strukturou s přiřazenými operacemi; Myers vychází z funkcí programu a směřuje k dekomposici do modulů, v dalším kroku pak specifikuje modul interface a teprve potom propracovává logiku jednotlivých modulů. Obě metody jsou slučitelné; jde jen o to, čím dříve začít. Nic nebrání tomu, aby nejprve byla provedena dekomposice programu podle zásad koncepce souhrného návrhu a potom modul po modulu byla metoda JSP aplikována na data, se kterými jednotlivý modul pracuje, a zapsána logika těchto modulů strukturními diagramy s přiřazenými operacemi.

Náročnější, ale také správnější je obrácený postup, tj. nejprve alespoň do určité úrovně podrobnosti vypracovat podle JSP programovou strukturu s přiřazenými operacemi určující logiku celého programu a pak přistoupit k realizační fázi programu, která u rozsáhlejšího programu znamená jeho dekomposici na jednotlivé moduly. Vyčleníme funkce programu, které je účelné pro jejich netriviálnost a opakovatelnost realizovat samostatnými moduly, příp. sdružit více funkcí do jednoho modulu s informační soudržností /z toho hlediska je sice škoda, že Cobol 1974 nepřipouští více vstupních bodů v jednom modulu, avšak lze se s tím smířit/. Vzhledem k tomu, že Cobol 1974 neumožňuje pracovat s jedním souborem ve více modulech, je nutné soustředit všechnu práci se sekvenčním vstupním či výstupním souborem do jediného modulu, který je invertovanou korutinou pro čtení resp. zápis tohoto souboru. Pro soubor, který není sekvenčně zpracováván, musíme soustředit všechny operace s ním do jednoho modulu, přičemž jedním z jeho parametrů bude funkční kód umožňující rozlišit mezi jednotlivými požadovanými operacemi. Další moduly odvodíme ze selekcí, pro které podřízené komponenty představují relativně samostatnou a účelnou část zadání programu. Konečně další moduly vzniknou v důsledku použití techniky programové inverze, což odpovídá STS dekomposici. Dalším krokem bude vypracování modul interfacu, který zejména pro programy pracující v databankovém prostředí musí obsahovat dobře propracované předpoklady pro vyvolání modulu a vnější efekty. V této fázi zároveň

prověříme, zda takto definované moduly vyhovují požadavkům pokud se týče jejich spřaženosti a soudržnosti. Poté propracujeme detailně podle JSP strukturu jednotlivých modulů a kódujeme je.

Literatura

/1/ Vondráček B., Kretschmer M., Orbal P., Suchomel J., Technologie strukturovaného programování, sborník Programování '80

/2/ Čimbura V., Tvrdík J., Problémy návrhu modulárních programů, sborník Programování '80

/3/ Kováč J., Tvorba programov v databankovom prostredí IDMS, sborník Programování '83

/4/ Myers G. J., Composite /Structured Design, Van Nostrand Reinhold Comp., 1978