ZVÝŠENÍ PRODUKTIVITY PRACÍ PŘI ŘEŠENÍ SYSTÉMU AUTOMATIZOVANÉHO ZPRACOVÁNÍ DAT POMOCÍ METODY DATOVÝCH TOKŮ

Ing. J. Klečka, Ing. A. Winkler

Čas a úsilí pro vývoj konvenčních aplikací zpracování dat stále vzrůstá a stává se úzkým hrdlem. Tak jak se stále zlepšuje poměr "cena - výkon" technických prostředků a jak se objevuje stále více nových aplikací, roste i tlak na růst produktivity práce tvůrců těchto aplikací.

Jednou z cest ke zvýšení produktivity vývojových prací je použití metody d a t o v ý c h t o k ů. Jedná se o techniku, která umožňuje, aby programy byly vyvíjeny jako kombinace nezávislých funkcí, které jsou těsně propojeny toky dat.

Konvenční programování se koncentruje spíše na popis toku řídících povelů, než na tok dat samých. Konvenční programování, založené na klasické koncepci von Neumannova počítače, specifikuje přesné sekvence činností a rozhodování, které musí proběhnout, aby mohla být data zpracována. Koncepce na základě datových toků se soustřeďuje na toky dat programem a na transformace, které budou na těchto datech prováděny.

V řídícím procesu pozoruje člověk vzájemně působící funkce jako sítě datových toků, ukazující, jak lidé mezi sebou vzájemně komunikují a spolupracují. Typické je, jak lidé funkce používají a jak mezi sebou sdílí důležité informace, při čemž tyto informace mezi sebou ve svých funkcích sdílí asynchronně. Diagramy datových toků zobrazují tyto vzájemné vazby tak, jak ve skutečnosti existují.

Pohled na aplikaci, kterou vývoj systému implementuje, je podstatný pro produktivitu. Ve většině dnes vyvíjených systémů musí vývojový pracovník převést pohled datového toku /toku dat mezi řídícími místy/ manuálních aplikací do procedurálního pohledu. Tento převod je nutný, protože dnešní konstrukce von Neumannova počítače je orientována procedurálně.

Analýza a konstrukce programů se doposud provádí většinou pomocí vývojových diagramů, které modelují postup zpracování dat počítačem. Jelikož však vývojový diagram nemodeluje přímo uživatelovu aplikaci, nemůže si uživatel ověřit, že navržený vývojový diagram reprezentuje všechny požadavky jeho aplikace.

Zakreslením aplikace formou diagramu datových toků během analýzy a konstrukce se vyhneme nutnosti překládat tuto aplikaci do procedurální formy, přičemž ji popíšeme formou, které koncový uživatel může rozumět, ověřovat a efektivně měnit. Jestliže budeme schopni toto udělat, vyhneme se prodlení, vzniklému překladem do procedurální formy. Překladu do strojově orientovaného procedurálního pohledu se můžeme vyhnout použitím již existujících funkcí nebo použitím neprocedurálních generátorů. V jakémkoliv případě je procedurální pohled omezen jen na tvorbu jednotlivých funkcí a to jak již dříve připravovaných, tak nově potřebných a tudíž nově vytvářených.

I když perspektivy použití metody datových toků jsou charakteristické zejména pro distribuované zpracování na vysoce výkonných počítačových architekturách, využívajících asynchronního zpracování na multiprocesorech, neznamená to, že popsané metody nelze použít již dnes použitím multizpracování a multitaskingu.

Diagram datového toku je vhodnou cestou k popsání funkcí, které jsou propojeny toky dat. Propojení mezi funkcemi může být jak lineární, tak síťové.

Nasazení metody datových toků předpokládá, že program bude kombinací f u n k c í, propojených datovými toky. Jednotlivé funkce jsou pak implementovány jako nezávisle kompilované subrutiny nebo korutiny. Každá funkce může být vyvíjena jako téměř úplně nezávislá na ostatních funkcích. Jestliže je jednou funkce vyvinuta, je nejdůležitější, aby dělala to, co se po ní požaduje a že se nemusí nikdy předělávat. Jednoduše řečeno, funkce může být znovu použita beze změn. Jediným omezením je požadavek, aby vstupovala a vystupovala požadovaná data a zpracování proběhlo na cílovém zařízení.

Schopnost funkce zpracovat data tak, jako kdyby šlo o sekvenční soubory, činí každou funkci nazávislou na zdrojích a určení svých dat. Funkce tak může být použite v jakémkoliv programu, v němž je požadována její transformace dat vstupních na výstupní. Datový tok umožňuje, aby všechny funkce v programu byly napsány stejnou, znovu použitelnou formou. Použitelnost funkcí v jiných programech se pochopitelně mění od maximální opakovatelnosti až po jednorázové použití a závisí především na tom, jak jednotlivé funkce vzájemně mezi sebou souvisí. Čím menší vzájemné vztahy mezi funkcemi existují, tím mají vyšší opakovatelnost v jiných programech. Nejméně závislé a nejvíce využitelné funkce jsou většinou malé programové moduly, odděleně kompilované, provádějící jednoduché transformace dat.

Každé funkce může mít jednu nebo více následujících vlastností:

1. Sdílení dat s jinými funkcemi.
2. Předávání řízení jiné pojmenované funkci /např. výrokem CALL/.
3. Kompilace do stejného fyzického modulu.
4. Sdílení stejných lokálních proměnných a řady definic návěští.
5. Skoky dovnitř a ven z kódu jiných funkcí.
6. Fyzické rozprostření funkce mezi ostatní funkce /tzn. kód funkce není kontinuální/.

Funkce uvnitř velkého monolitického programu jsou často vztaženy na všechny výše uvedené body. Výsledkem jsou prakticky neopakovatelné funkce. Strukturované programování /1/, používající výroku INCLUDE pro začlenění segmentů zdrojového jazyka, eliminuje body 5 a 6, protože se jedná o zdrojové segmenty s jedním vstupem a jedním výstupem. To sice činí funkce vhodnějšími pro nové použití, ale ne natolik, aby nové použití bylo praktické.

Strukturovaný návrh /1/ eliminuje body 3-6 tím, že dělí programy do hierarchií samostatně kompilovaných modulů, které se navzájem volají a data si předávají jako parametry výroku CALL. Jestliže je funkce samostatně kompilována, je nové použití prakticky možné, protože není požadována žádná další práce pro porozumění, výběr a nové využití funkce. Předávání dat prostřednictvím CALL však vyžaduje volání cílové funkce jejím jménem a předání řízení i s potřebnými daty. Je proto obtížné použít znovu tutéž funkci v jiném programu, kde cílová funkce může být jiná. Je rovněž obtížné posílat do jedné funkce vícenásobné datové proudy, vznikající v různých zdrojích - zejména když cílová funkce nemůže být zpracována, dokud nejsou k dispozici všechna vstupní data /1/. Je obtížné implementovat obecné sítě funkcí, představujících hierarchie volaných subrutin. Je rovněž zřejmé, že předávají-li si jednotlivé funkce navzájem řízení, nemohou být zpracovány asynchronně.

Mnohem snazší je použití funkce, která nepředává řízení žádné další funkci. To platí i pro hierarchie volaných modulů, kde opětné použití je možné u těch modulů, které nevolají žádné další moduly - platí pro ně bod 1. Datový tok poskytuje daleko větší nezávislost mezi funkcemi uvnitř programu. Pro jakoukoliv funkci není nezbytné, aby volala nebo sama zahrnovala jakoukoliv další funkci. Je proto možné znovu použít jakoukoliv takovou funkci v jiném programu, aniž by musela tato funkce být měněna.

V automatizovaném zpracování dat je neustále nutno bojovat proti různým omezením, snižujícím výkonnost výpočetních systémů. Jedním z největších omezení je složitost navrhovaných aplikací. Produktivita programování prudce klesá v závislosti na složitosti programů, které vytváříme. Ze zkušeností vyplývá, že čím větší a tím složitější programy vytváříme, tím nižší produktivitu práce dosahujeme. Lze dokázat, že se jedná o exponenciální závislost. Implementace velkých programů může dokonce přesáhnout i možnosti vývojových týmů.

Složitost implementace velkých programů lze redukovat a dnes se již touto cestou ubíráme, rozdělením programů na jednotlivé moduly. Musíme se ovšem současně snažit o omezení vzájemných vazeb mezi těmito moduly. Jelikož metoda datových toků umožňuje větší nezávislost mezi moduly - funkcemi, než to dovolují hierarchie volaných rutin, je možnost růstu produktivity vyšší, než je tomu u strukturovaného návrhu, přičemž strukturovaný návrh sám umožňuje vyšší produktivitu než strukturované programování se svými vysoce propojenými segmenty. Čím jsou jednotlivé funkce nezávislejší, tím snadněji se zkracuje čas, potřebný pro implementaci, protože lze rozdělit práci mezi více vývojových pracovníků. Vyplývá to především ze skutečnosti, že nezávislé funkce vyžadují méně komunikací mezi vývojovými pracovníky, kteří funkce vyvíjí.

Abychom dosáhli postačující úrovně složitosti, musíme zajistit i dobrou podporu pro tvorbu nezávislých programových částí, složených z více programových modulů. Aplikační vývojoví pracovníci mohou budovat vysoce složité systémy efektivně jen tehdy, když budou mít k dispozici programové celky, které zevně mohou vypadat jako jednoduché, přičemž se však mohou skládat z většího množství podcelků, o jejichž funkci nemusí mít vývojový pracovník ani zdání. Touto cestou se dnes ubírají prakticky všechna výrobní odvětví. Programované inteligence jednotlivých aplikací obvykle zahrnuje následující úrovně:

- jobový proud;
- program /jobový krok/ anebo on-line transakce;
- subrutina/makroinstrukce;
- výrok jazyka vyšší úrovně;
- strojová instrukce;
- instrukce mikrokódu.

Pro uživatele v následující vyšší úrovni je irelevantní, zda je použito funkce, složené z více podcelků nebo ne. Redukce složitosti se projeví jen tehdy, když použití podcelků nižší úrovně nezávisí na nutnosti porozumět jim.

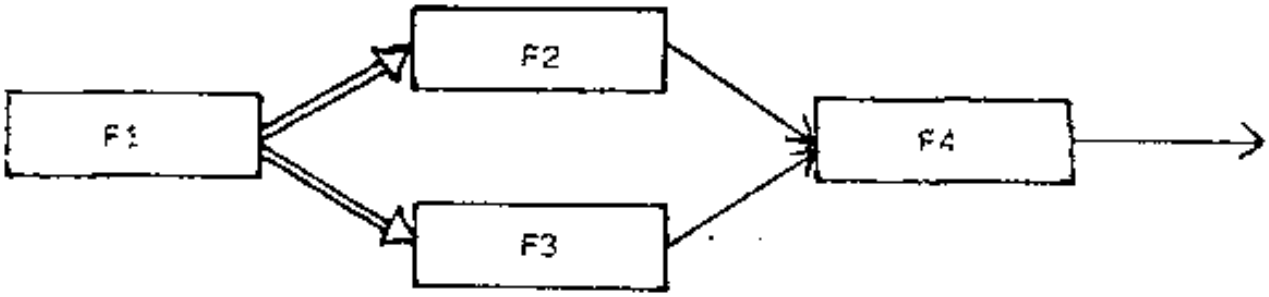
Metoda datových toků podporuje vznik sítí funkcí, které mohou být použity jako podčásti v jiných sítích. V literatuře /2/ se doporučuje pro lepší porozumění popsat aplikace jako hierarchie diagramů datových toků. Použití takovýchto hierarchií může být významně zejména pro porozumění velkým a rozsáhlým aplikacím. Ze stejného důvodu může být schopnost vyvinout velký program významně zvýšena schopností použít tuto hierarchii jako hierarchii podcelků.

Kromě již zmíněných fází vývoje aplikací poskytuje metoda datových toků i další fáze. Sem patří především ladění, usnadňované především možností vytvářet z již existujících funkcí prototypy programu, snadno proveditelné změny činnosti programu, jeho optimalizace a údržba. Je pochopitelné, že každá z výše uvedených fází bude tím snadnější, čím budou jednotlivé funkce samostatnější a nezávislejší na jiných funkcích do sítě zapojovaných a zejména pak na činnosti programu samotného. Tím, že jsou jednotlivé fáze snadnější, mohou být provedeny úplněji a přesněji - výsledkem je zvýšení kvality výsledného produktu.

Tvorba jednotlivých funkcí předpokládá, že funkce budou programovány jako reentrantní moduly. Tato vlastnost umožní, aby se funkce v případě opakovaného volání v programu v něm vyskytovala pouze jednou, přičemž jednotlivé datové toky budou funkci předávány prostřednictvím datové fronty, zpracovávané formou "první dovnitř, první ven". V literatuře /3/ se definují datové toky jako entity dat, předávané postupně do nebo ven z funkce /programového modulu/, přičemž se pochopitelně předpokládají určité vlastnosti:

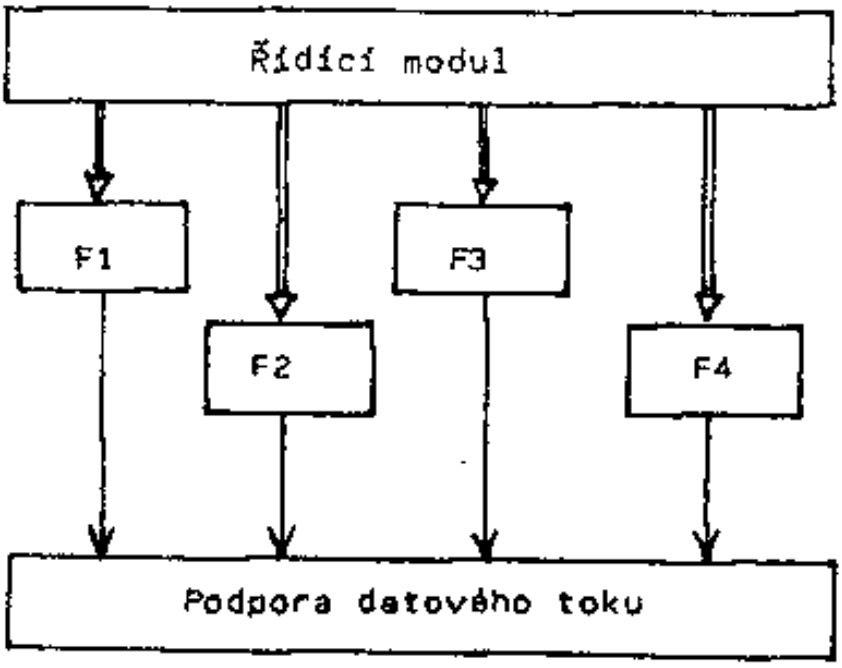
- na vstupu do funkce může být více než jeden datový tok,
- programový modul zpracovává vždy jen jednu datovou entitu,
- datová fronta může být během zpracování neúplná nebo prázdná - potom je činnost funkce pozastavena. Z praktických důvodů jsou rozsahy jednotlivých front omezené - vejde se do nich omezený počet datových entit,
- činnost programové funkce se zahajuje a ukončuje zvláštními datovými entitami.

Při vícenásobném výskytu funkce v programu jsou tedy jednou reentrantní funkcí zpracovávány postupně jednotlivé datové toky tak dlouho, až je zpracována poslední "koncová" entita. Programování funkcí se zjednodušuje tehdy, když si funkce volá programovou podporu pro práci s datovými entitami, která předá funkci data v požadované formě. Je to podstatně jednodušší než kdyby se volání dat muselo v každé funkci řešit samostatně. Lze to ukázat např. na síti volaných funkcí (viz obr. 1).



obr. 1

V tomto případě funkce F1 a F4 pracují každá se dvěma datovými toky, přičemž funkce F4 by musela rozhodovat, zda obě fronty do ní vstupující jsou již ukončeny. Síť volaných funkcí lze ale realizovat pomocí metody datových toků efektivněji tak, jak je uvedeno na obrázku č. 2.



obr. č. 2

kde lze všechny funkce volat z řídicího modulu, přičemž celou situaci lze vyřešit voláním již zmíněné programové podpory datových toků, která je schopna "číst" a "psát" data. Podrobný popis jednoho hotového systému /Advanced Modular Processing System - AMPS/ je popsán v článku J. P. Morrisona /3/.

Na základě předchozích částí určitě někteří z čtenářů dospějí k názoru, že již metodu datových toků vlastně používají i bez znalosti naznačeného aparátu. K tomuto způsobu práce došli intuitivně při snaze o zvýšení racionalizace vlastní práce.

Prvním předpokladem pro tento způsob práce je zvládnutí modulární výstavby systémů, a to nejen v oblasti programového zabezpečení, ale již při systémové analýze problému. Právě během analýzy problému je možno najít určité funkce, které je možno opakovaně použít jak při výstavbě konkrétního systému, tak i při budování dalších obdobných systémů.

Pro názornost uvedeme příklad distribuovaného zpracování dat. Obecně lze tyto systémy jako systémy otevřené s dynamickým chováním dekomponovat na vstup - zpracování - výstup. Je zřejmé, že takové členění je příliš hrubé pro specifikaci obecných funkcí nad toky dat procházejících tímto systémem. Jestliže však půjdeme na nižší rozlišovací úroveň, můžeme například vstupní oblast rozdělit na následující části:

- formátování vstupních zařízení (např. formáty displejových obrazovek pro vstupní formuláře)
- formální kontroly vstupních dat (kontrola na numeričnost rozsah atd.)
- kontroly logické správnosti vstupních dat

⋮

atd.

První dvě části jsou na datech prakticky nezávislé. Třetí část je již na datech závislá, protože se zabývá jejich sémantickým obsahem. Části "formátování vstupních zařízení" a "formální kontroly vstupních dat" je tedy možno vypracovat jako obecné funkce. Při projektování dalšího systému je lze již použít jako hotové stavební kameny.

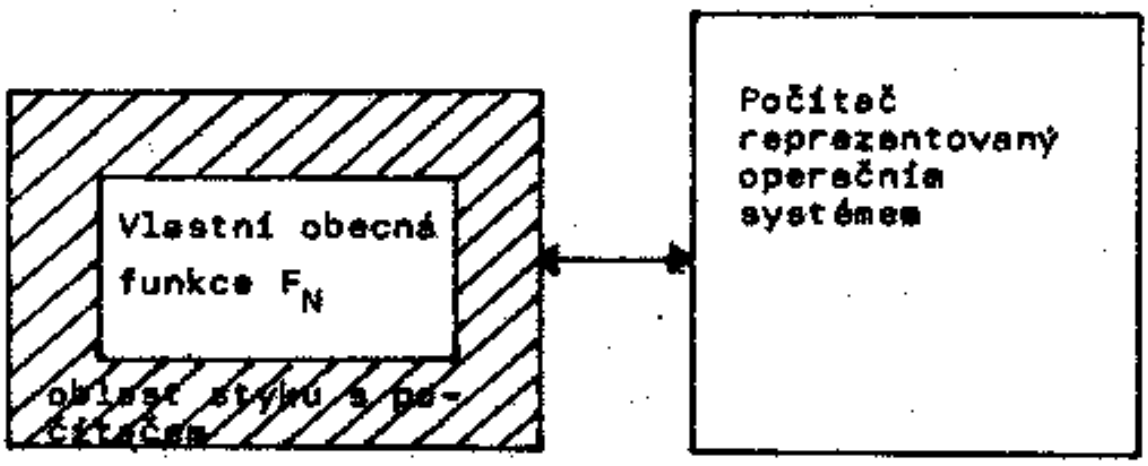
Jak zabezpečíme obecnost u takto specifikovaných funkcí? Existuje velká škála možných řešení. Nejběžnější jsou tato:

- parametrizace
- generování
- interpretátory

Není možno jednoznačně určit, které řešení je nejvhodnější, protože to závisí na technickém, programovém i personálním vybavení.

Je však zapotřebí si uvědomit, že nedokonalá dekompozice může někdy způsobit to, že je budován například složitý generátor programů přičemž při správně provedené dekompozici by byl problém řešitelný několika jednoduchými parametrickými programy. Na druhé straně při příliš velké podrobnosti se může dekompozice stát tak rozsáhlá, že obsluha spojená s vyvoláváním jednotlivých funkcí zhorší parametry systému jako celku.

Konkrétní aplikace metody datových toků jsou sice závislé na technických prostředcích, na kterých je systém provozován, ale jedná se zde o formu vyvolávání jednotlivých funkcí a nikoliv o jejich obsah. Pokud chceme, aby jednotlivé obecné funkce byly přenositelné na různé typy počítačů je vhodné oddělit při jejich vytvoření část funkční od části mezistyku s počítačem viz obrázek č. 3.



obr. č. 3

Při dodržení těchto zásad lze systémy budované touto metodou převádět velmi efektivně na různé typy počítačů.

Metoda datových toků je velmi efektivně podporována tzv. transakční dekompozicí. Její realizaci si můžeme představit tak, že např. v paměti je uložen rezidentní modul a tento modul vyvolává a zpětně uvolňuje jednotlivé funkce pracující nad datovými toky.

V našich podmínkách zpravidla nedisponujeme paralelním způsobem zpracování, kde jsou velmi složité vzájemné synchronizace činností. I přes tuto skutečnost je zřejmé, že metoda datových toků zavedením určitých posloupností funkcí nad daty umožňuje vysokou adeptabilitu systému na změny a stavebnicové budování. Změnu nebo rozšíření lze snadno realizovat vložením nebo zřetězením nové posloupnosti funkcí.

Z hlediska uživatele je tato uspořádanost celého problému na sledy jednoduchých funkcí mnohem snáze pochopitelná, než velké monolitické programy s mnoha kumulovanými funkcemi.

V naší organizaci byla metoda datových toků realizována v oblasti projekce dispečerského řídícího systému. V tomto systému právě použití této metody umožňuje operativní přeskupování funkcí nad daty a rozšiřování systému v závislosti na neustále se měnících vnějších podmínkách. Dalším známým programovým systémem, který plně využívá metodu datových toků, je programový balík "GIN-SMEP" viz pozn. */. Systém "GIN SMEP" byl dekomponován do čtyř základních skupin parametrických programů:

- programy pro formuláře (návrhy formátování dat pro vstupní i výstupní operace)
- programy pro věty (popis vět, aktualizace vytváření dávek atd.)
- programy pro soubory (operace se soubory, třídění, řetězení, konverze atd.)
- programy pro manipulace (logické a fyzické operace při manipulaci s větami).

Systém bude nasazen od úrovně technologická až po úroveň ministerstva. Zde právě šíře záběru daného systému podporuje aplikaci výše popisované metody datových toků.

Na závěr je zapotřebí říci, že příspěvek si neklade za cíl prezentovat metodu datových toků jako nový a naprosto univerzální prostředek, ale podat sdělení o jednom z možných přístupů k řešení projekce a realizace rozsáhlejších programových systémů.

Použitá literatura:

1. W. P. Stevens "How data flow can improve application development productivity", IBM Systems Journal 21, No. 2 /1982/

2. J. A. Zachman "Bussiness Systems Planning and Bussiness Information Control Study - A comparison", IBM Systems Journal 21, No. 1 /1982/

3. J. P. Morrison "Data Stream Linkage Mechanism", IBM Systems Journal 17, No. 4 /1978/

1. Programový systém pro sběr a zpracování dat v reálném čase "GIN - SMEP" průběžná výzkumná zpráva Koubek a kol. /1982/

+/ Tento systém je v současnosti vyvíjen jako základní programová podpora pro počítače SMEP nasazené v řídicím systému, který je cílem řešení státního úkolu "Víceúrovňový systém řízení důlní výroby v podmínkách VHO koncernového typu". Řešitelem tohoto systému je VÚEPE - OSTRAVA.