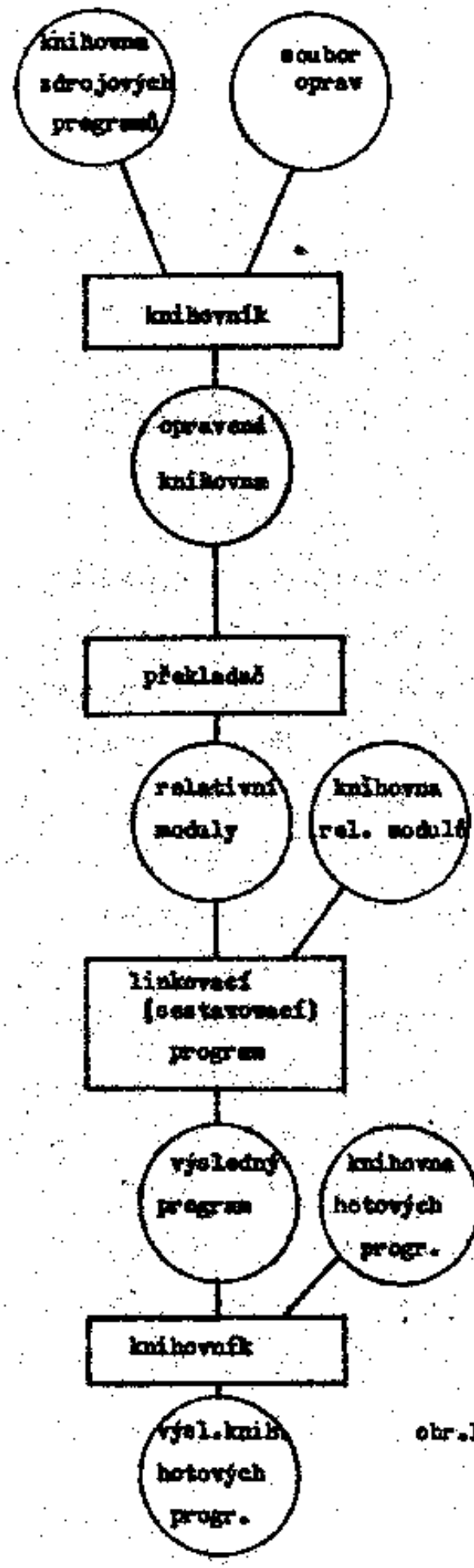
Jaroslav Josífko, prom. mat.
n.p. Ostroj Opava

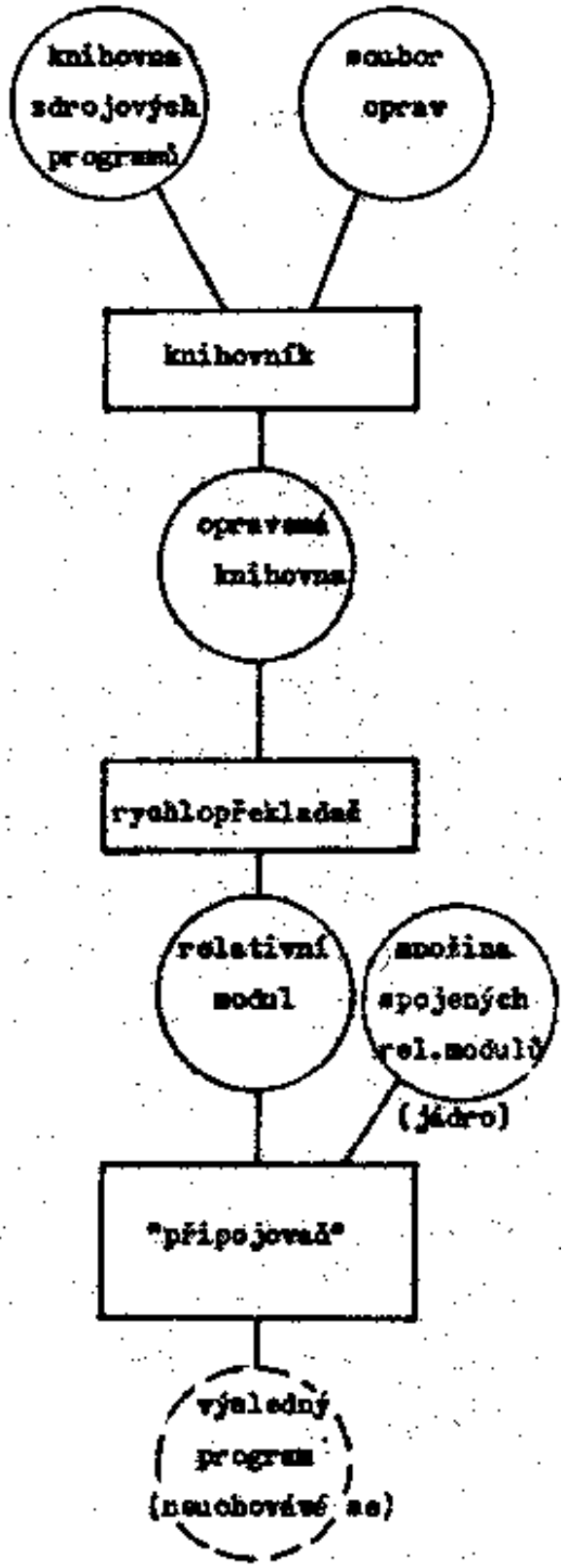
SYSTÉM ZDROJOVÝCH MODULŮ VE SPOJENÍ S RYCHLOPŘEKLADAČEM

1. Úvod

Při zpracování celého komplexu programů pro určitou agendu hromadného zpracování dat nebo dokonce pro integrovaný systém těchto agend je již samozřejmostí, že musíme aktualisovat nejméně dva druhy knihoven programů. Máme knihovny s uloženými programy ve zdrojové formě a další pak již s hotovými, t.j. přeloženými a slinkovanými (sestavenými) programy. Další knihovnou je knihovna relativních modulů, kterou spravidla již běžný programátor neaktualisuje, ale využívá ji při linkování (sestavování) hotových programů. Změna v alespoň jednom programu nás nutí nejenom programy přeložit a sestavit, ale také opravit všechny dotyčné knihovny. Zjednodušený sled činnosti můžeme znázornit na obr. 1. Neutěšená situace nastává tehdy, máme-li k dispozici pomalý kompilátor a knihovny na magnetických páskách. Kupříkladu při programování v jazyce Cobol na Tesle 200 může promítnutí i malé změny v jednom


knihovna zdrojových programů
soubor oprav
knihovník
opravená knihovna
překladač
relativní moduly
knihovna rel. modulů
linkovací (sestavovací) program
výsledný program
knihovna hotových progr.
knihovník
výsl.knih. hotových progr.

obr.1


knihovna zdrojových programů
soubor oprav
knihovník
opravená knihovna
rychlopřekladač
relativní modul
množina spojených rel.modulů (jádro)
"připojovač"
výsledný program (neuchovává se)

obr. 2

cobolského programu trvat 0,5 až 1 hodinu, s aktualisací obou knihoven i déle.

Existují sice způsoby, jak tento čas zkrátit, ale nikoliv na úrovni původního vyššího programovacího jazyka. Pomocí různých REPů, ALTERů atd. můžeme opravit přímo strojové instrukce. Tato změna však neopraví současně zdrojovou ani výslednou knihovnu a podle mého názoru by neměl být tento způsob na úrovni programovacích metod 3. generace počítačů používán.

Položme si však otázku - je schema z obr. 1 i pro budoucnost to nejlepší? Není možno postupovat i jiným způsobem? Při podrobnější analýze zjišťujeme, že nejpádnějším důvodem proč dodržujeme toto schema je obrovská výhoda využívání již odzkoušených (většinou firemních) relativních modulů, takže zbývá při linkování pouze připojovat své moduly. Stručně řečeno - využít co možno nejvíce již hotového, odladěného a soustředit se na svůj problém, který je vždy jiný. Je však náš problém vždy opravdu jiný? Nekopírujeme ze svých starých programů celé "osvědčené" úseky, abychom je nechali opětovně překládat? Proč systémoví programátoři většiny výpočetních středisek vytvářejí univerzální moduly ulehčující tisk, práci s údaji na děrné pásce atp.? Statistickým zkoumáním hotových výsledných programů objevujeme skoro tutéž množinu použitých relativních modulů. Zefektivnit práci podle schematu na obr.1 tedy znamená obohatit knihovnu relativních modulů o opakované algoritmy a nejpoužívanější procedury, aby se ve fázi překlad zpracovávalo jen to, co je výjimečné a charakteristické pro danou úlohu.

2. Pojem rychlopřekladač

Rozviňme naše úvahy o zefektivnění schematu z obr. 1 dále. Představme si, že jsme našli takovou nadmnožinu relativních modulů, že pro většinu (dejme tomu 95 %) našich úloh v ní naleznem všechny potřebné moduly. Spojme nyní tyto moduly a vytvořme jakýsi polotovar, který nazveme jádrem. Problém linkování se značně zjednoduší, neboť se redukuje na "připojování" jednoho přeloženého modulu. Pokud bude překlad daného modulu pro nás dostatečně rychlý, můžeme upustit od archivace výsledného programu.

Při spouštění hotového programu musí operační systém provést jeho vyhledání v knihovně, natažení do paměti a určité počáteční úpravy. Dobu trvání této operace nazvěme manipulačním časem spouštění. Překlad bude dostatečně rychlý tehdy, bude-li součet časů při opětovném překladu srovnatelný řádově s manipulačním časem spouštění. Překladeč, který tuto podmínku splňuje nazveme rychlopřekladač. Činnosti z obr. 1 pak můžeme nahradit schematem na obr. 2.

3. Zdrojový modul

Použití "jádra" (spojení nejužívanějších relativních modulů) vyvolá změnu v charakteru vstupujícího zdrojového programu. Například celé pasáže programu, týkající se ošetření proměnného formátu údajů z děrné pásky jsou zde nahrazeny odvolávkou na hotový modul obsažený ve společném jádře. Při stejném konečném efektu budou zdrojové programy mnohem kratší. Většinu programů můžeme rozložit na menší logické

celky z hlediska zpracování souborů (viz obr.3).

Množinu zdrojových příkazů, která tvoří logický celek začínající otvíráním a končící uzavíráním souborů budeme nazývat zdrojovým modulem. Každý program, který řeší běžnou úlohu z oblasti hromadného zpracování dat, se dá vyjádřit většinou jedním, případně několika zdrojovými moduly.

4. Universální program

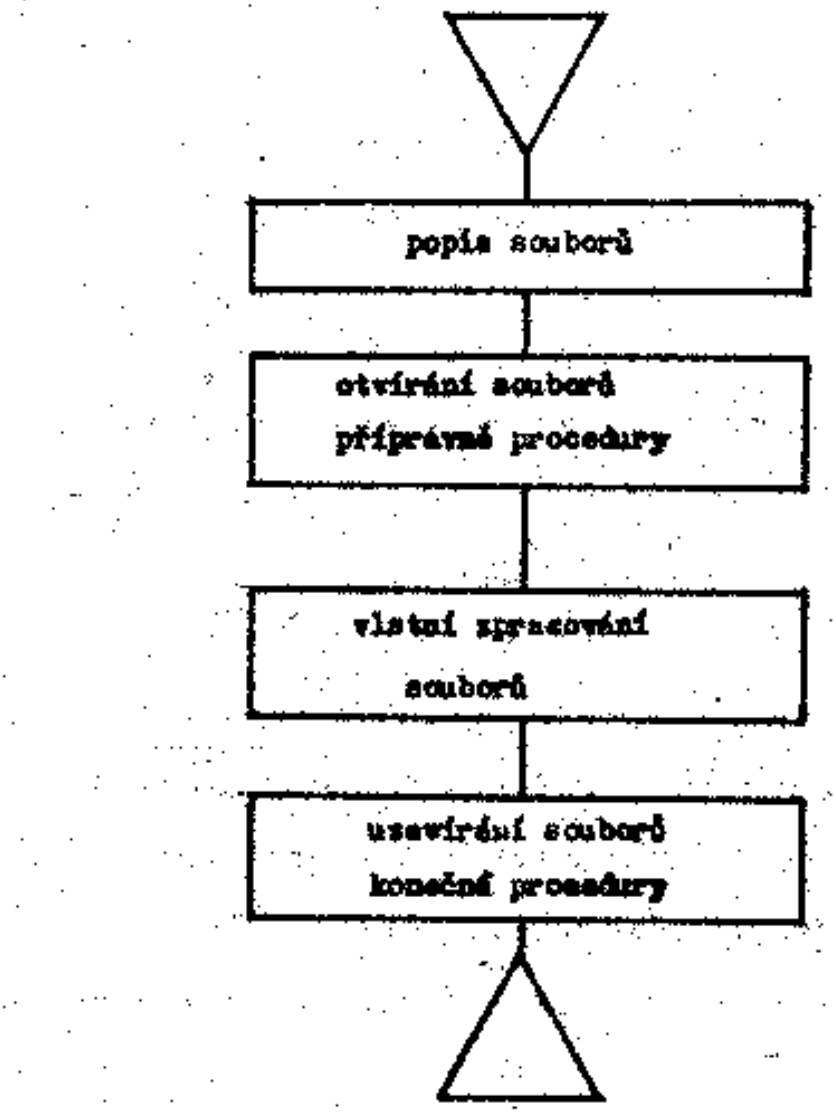
Pokud jsme zdrojové programy redukovali na zdrojové moduly, můžeme v obr.2 knihovnu zdrojových programů nahradit knihovnou zdrojových modulů. Pro identifikaci v knihovně by každý modul měl mít své jméno resp. své číslo a vzestupně očíslované příkazy. Pokud knihovna zdrojových modulů bude obsahovat moduly napsané v témže zdrojovém jazyce a jejich počet bude dostatečný pro řešení ucelého problému hromadného zpracovní dat, pak budeme mluvit o systému zdrojových modulů.

Vytvořme nyní universální program, který umí plnit současně všechny funkce z obr. 2. Zpracování pak bude probíhat podle obr.4 a bude vyžadovat pouze údržbu knihovny se zdrojovými moduly.

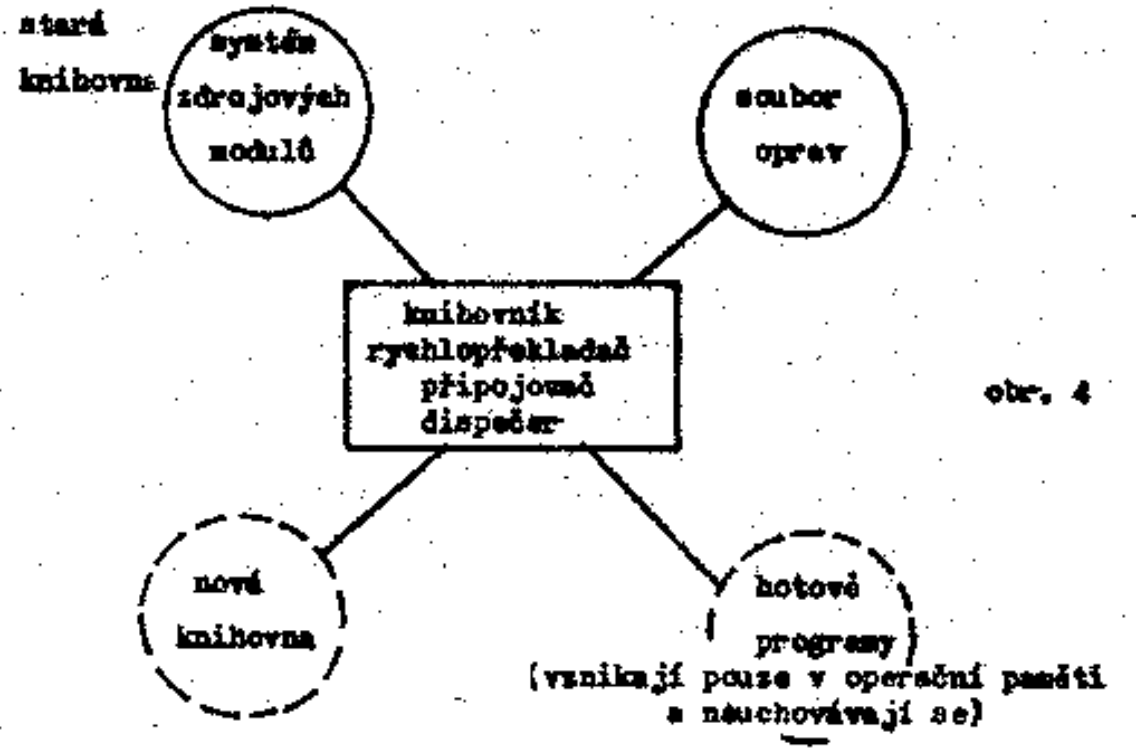
5. Dynamická změna modulu

Při využívání systému zdrojových modulů a universálního programu, obsahujícího rychlopřekladač pojem překladu ztrácí pro programátora význam. Z jeho hlediska se z překladu stala jakási vnitřní manipulační záležitost, která časově odpovídá době spouštění již hotových programů podle klasického způsobu (podle obr. 1).

Účelová logická část programu pro jeden zdrojový modul



obr. 3



obr. 4

V jednom chodu lze opravit a spustit postupně několik modulů. Soubor oprav se může promítnout buď trvale a vzniká nová knihovna, nebo platí pouze pro právě probíhající chod a nová knihovna nevzniká. Pokud není proveden trvalý zásah do knihovny zdrojových modulů, mluvíme o dynamické změně (opravě).

Dynamická změna se liší od zmíněných REPů a ALTERů především tím, že příkazy souboru oprav jsou psány v téměž zdrojovém jazyce jako opravované moduly. Soubor oprav pro dynamickou i trvalou opravu musí být formálně totožný.

6. Realisace

Předchozí teoretické úvahy byly prakticky odzkoušeny na počítači TESLA 200. Vzhledem k tomu, že celý univerzální program jsem tvořil sám za velmi omezujících podmínek a mimo své pracovní úkoly, rozhodl jsem se pro realisaci použít vlastní programovací jazyk, značně jednodušší než např. jazyk Cobol (a pochopitelně ne tak universální jako Cobol). Tím se mi zjednodušila i konstrukce rychlopřekladače, který je v tomto případě nejpracnější částí programu. Hrubé schema struktury universálního programu je na obr. 5.

Knihovna zdrojových modulů je umístěna na magnetické pásce, systém zdrojových modulů je řešen jako jeden soubor rozdělený do podsouborů. Soubory oprav jsou na 80-ti sloupcových děrných štítcích.

Programu bylo použito například pro zpracování agendy třídenních mezinárodních závodů v orientačním běhu. Výsledkem bylo zpracování presentace tisícovky závodníků, startovních a výsledkových listin za první, druhý, první a druhý, třetí

Příloha č. 1

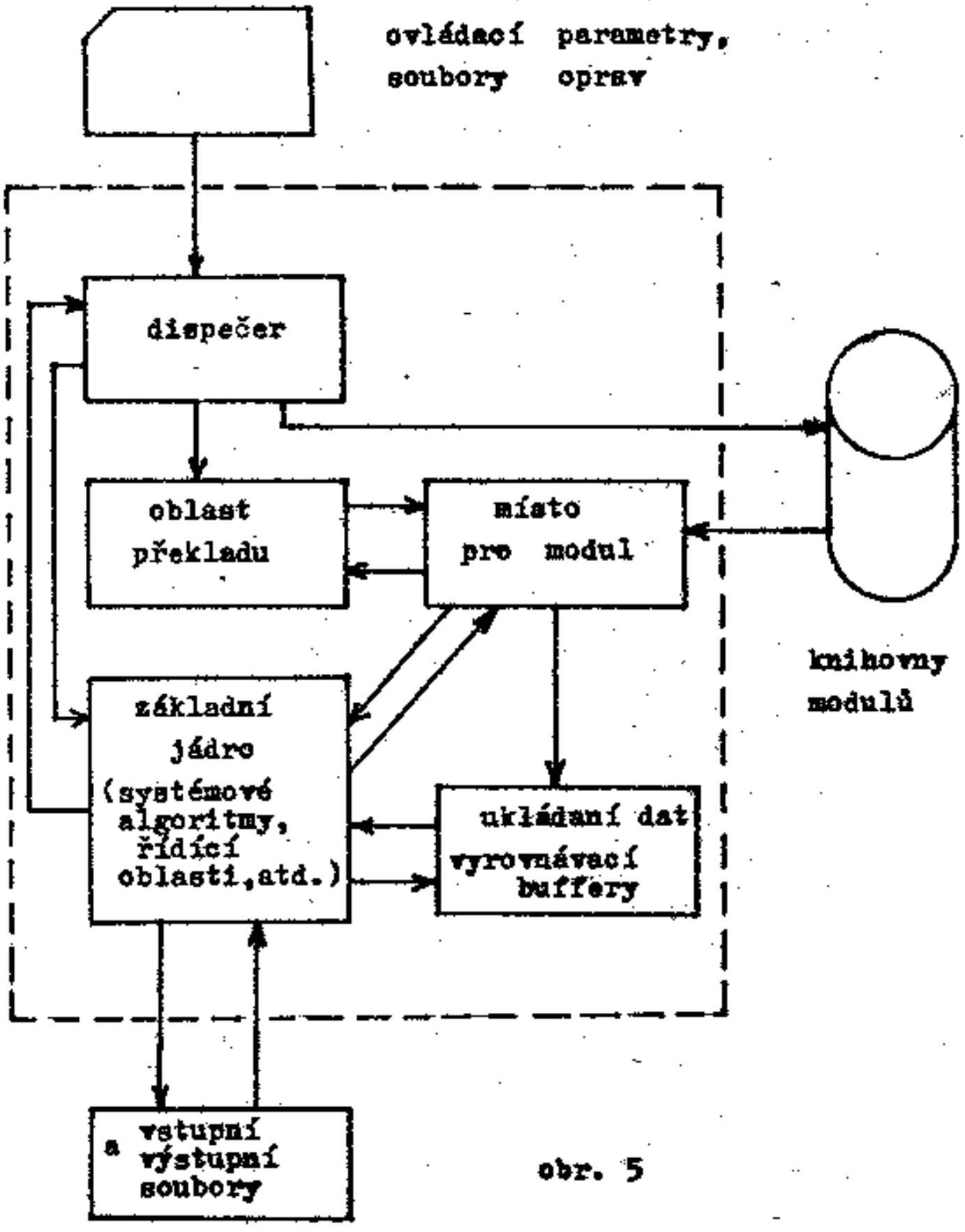
Ukázka části výpisu knihovny zdrojových modulů v jazyce OIZD2

```
                                                   14
VYPIS AKTUALISOVANEHO SOUBORU KNG-OS       Z PASKY 8552

0010      DOPLNOVANI PROVOZU A KONTROLA DUPLICIT                    480 0010
0020 *        FDB     DP                                            480 0020
0030 *        FDD     MT5 M512, 17 F 'DPP1'                         480 0030
0040 *        M       BO 17                                         480 0040
0050 *        ZAPISD                                                480 0050
0060 *        KONEC                                                 480 0060
0070 **                                                             480 0070

0010       DOPLNOVANI PROVOZU A KONTROLA,DUPLICIT                   490 0010
0020 *          FDN   MT5 M512,17 F  'DPP1'                         490 0020
0030 *          FDA   MT6 M512,17 F  'DPP2'                         490 0030
0040 *          FDB   MT2 M942,94 F  'NDPLAN10'                     490 0040
0050 *          FDC   MT4 M96,94 F   'NDPLAN10'                     490 0050
0060 *          ZOTRI4  17/KFY Z1=0,15/     C320                    490 0060
0070 *          PROGR   N1 N2 N3                                    490 0070
0080 *          KLIC1C  A0,3  B12,3                                 490 0080
0090 *          KLIC2C  A3,12 B0,12                                 490 0090
0100 *     N2   JE-LC   B0,15 = C300,15 p N7                        490 0100
0110 *          M       B0,94                                       490 0110
0120 *          ZAPIS                                               490 0120
0130 *         PPREC  C300,15  B0,15                                490 0130
0140 *          KONEC                                               490 0140
0150 *     N3   ULOZ  '501' C210                                    490 0150
0160       VYLUCOVANI DUPLICIT                                      490 0160
0170 *          JE-LC  B0,15 = C300,15 p N7                         490 0170
0180 *          NAPL                                                490 0180
```

Zjednodušené schema universálního programu



obr. 5

den a celkové výsledky za všechny dny. Startovní časy v dalších dnech závisely na výsledcích předchozích dnů podle složitých algoritmů. Danou programovou technikou byl odladěn během 14-ti dnů systém třiceti zdrojových modulů, které odpovídaly rozsahem asi dvaceti běžným programům. Na práci se podíleli pouze dva programátoři.

Výhody daného způsobu se projevují především v ladění celého systému programů. Velmi rychle se odzkoušují vzájemné vazby jednotlivých členů mezi sebou.

Jinou agendou, kde byla nová programovací technika vyzkoušena byla agenda plánování náhradních dílů. Programové zpracování této agendy by muselo být z důvodu nedostatku programátorské kapacity při použití běžných programovacích metod uživateli odmítnuto. Díky nové metodě mohl uživatel dopřesňovat a rozšiřovat zadání úlohy ve fázi náběhu své agendy, což bývá prógramátoru při použití běžných metod dosti nepříjemné. Uživateli, který mnohdy formuluje své požadavky pro počítačové zpracování poprvé je tímto způsobem dána možnost naučit se formulovat přesně své požadavky a obdržet nakonec takové výsledky, které jsou pro něho nejpotřebnější.

Závěr

Používání systému zdrojových modulů ve spojení s rychlo překladačem je nová netradiční programovací technika, které značně urychluje odlaďování jednotlivých programů a zejména vazeb mezi nimi. Pracuje pouze se zdrojovou knihovnou, knihovna hotových programů a její údržba zcela odpadá. Parametrisace programu je zbytečná, neboť zdrojové moduly můžeme dynamicky

měnit při každém spouštění zaváděním souboru oprav. Dynamická změna nemění původní obsah knihovny a přitom je universálnější proti různým vkládáním pomocí PARM, OPTION atp. Dynamickou změnou můžeme vkládat nové zdrojové příkazy, příkazy měnit i rušit. Můžeme měnit obsah textových řetězců, popisy souboru, třídící klíče, vyvolávání procedur atd., pochopitelně v souladu s možnostmi použitého zdrojového jazyka. Soubor oprav pro dynamickou změnu můžeme fyzicky použít v nezměněné podobě pro provedení trvalého zásahu do knihovny, mění se pouze ovládací (řídící) příkazy.

Metoda byla úspěšně odzkoušena na počítači Tesla 200 pro vlastní jednoduchý zdrojový jazyk GIZD2, pro který nebylo tak obtížné sestrojit potřebný rychlopřekladač.

Využití nové metody na jiných počítačích je podmíněno existencí vhodného universálního programu s rychlopřekladačem pro zdrojový jazyk.

Literatura

(1) Germain C.B., Programing the IBM 360, 1967

(2) Donovan J.J., Systems progreming, 1972

(3) Firemní software k počítači Tesla 200

(4) Josífko J., Manuál k programu GIZD2, 1975,
interní zpráva n.p. Ostroj.