Eduard Ježowicz, prom.mat., CSc.
Ústav pro využití výpočetní techniky v řízení, Praha

POJETÍ TYPOVÝCH ŘEŠENÍ A PŘÍSTUP K JEJICH TVORBĚ V OBLASTI APLIKAČNÍHO PROGRAMOVÉHO VYBAVENÍ

Úvod

Pojem typizace v tvorbě uživatelského programového vybavení se objevuje v posledních létech velmi frekventovaně v souvislosti s vytvářením automatizovaných systémů řízení. Tento proces je charakteristický mnohonásobným opakováním projektových prací za účelem vytvoření velkého množství obvykle rozsáhlých programových systémů konkrétních uživatelů. Od typizace v této oblasti se očekává zefektivnění těchto prací cestou opakovaných aplikací tzv. typových prvků.

Pojetí typizace prodělává přirozený vývoj charakteristický diferencovanými přístupy, někdy doprovázen přílišným optimismem vyplývajícím z dílčích úspěchů na jedné straně, na druhé straně se setkáváme se značně kritickými postoji k možnostem typizace v dané oblasti.

Tento příspěvek si klade za cíl pokusit se přesněji vymezit pojetí typizace v tvorbě aplikačního programového vybavení, definovat některé základní pojmy a uvést určitý prostředek k řešení problematiky.

Pojetí typizace

Obecně pod pojmem typizace v tvorbě uživatelského programového vybavení rozumíme vytváření programových řešení znovupoužitelných při projektování konkrétních programových systémů a skutečné využívání těchto řešení. Kriteriem využitelnosti programového řešení je efektivnost projektování konkrétních programových systémů, kterou lze charakterizovat poměrem nákladů na vytvoření a provozování konkrétních systémů s využitím "typových" programových řešení k nákladům vynaloženým bez jejich využití za účelem dosažení srovnatelných cílů.

Přijmeme-li toto obecné vymezení pojetí typizace, pak typizace neznamená jen unifikaci, standardizaci či normalizaci v tvorbě programového vybavení, ale zahrnuje především problematiku tvorby obecných, účelově přizpůsobitelných řešení pro určité, co nejčetnější třídy aplikací.

Z dosavadních zkušeností vyplývá, že řešení problému typizace častou převzímání programových řešení vytvořených účelově pro konkrétní systém a jejich uplatňování v jiných systémech nepřináší požadované efekty. Jsou známy akce jejichž cílem je evidence vytvořených programových řešení pro informování potenciálních uživatelů, při čemž celkový efekt těchto snah nelze považovat za uspokojivý. Na druhé straně existují příklady takových řešení, která byla nebo dosud jsou znovupoužitelně úspěšně využívána u řady uživatelů. Společným rysem takových řešení je převážně vždy skutečnost, že tato řešení byla cílevědomě koncipována pro širší okruh uživatelů, pro konkrétní uživatele byla dána explicitně možnost určitého přizpůsobení daného řešení pro konkrétní podmínky.

Uvedené skutečnosti naznačují další postup v řešení otázek typizace. Jde kromě řešení otázek obsahové náplně jednotlivých řešení především o využívání a rozvíjení

vhodných metod a prostředků pro tvorbu programových řešení zabezpečujících požadované vlastnosti typovosti těchto řešení na jedné straně a rozvinutí tvorby typových řešení se současným organizováním přizpůsobených služeb pro konkrétní uživatele na straně druhé /vytváření knihoven typových prvků, metodicko-pedagogická činnost atd./.

Základní pojmy

Obecné pojetí typizace vyplývá z předpokladu, že v určitých třídách automatizovaných systémů řízení se vyskytují ve značné míře řešení ne-li totožných, pak alespoň v jistém smyslu obdobných problémů.

Pokusme se dospět k exaktnější formulaci této problematiky. Ze systémového hlediska určitý subsystém nějakého systému representuje řešení konkrétního problému definovaného v systému. Definici takového problému nazveme konkrétní úlohou, která může být obecně dána:

- definičním oborem U vstupních informací
- definičním oborem V výstupních informací
- zobrazením F /implicitně nebo explicitně definovaným/ množiny vstupních informací do množiny výstupních informací,

$$(F(U,V) \text{ nebo } U \xrightarrow{F} V),$$

Obecně je tedy konkrétní úloha definována jako trojice (U,F,V). Prvky U,F,V jsou definovány soustavou $\mathbb{P}$ výroků vyjádřených určitým /obvykle na programovacím jazykem/. Předpokládejme, že je dána třída konkrétních úloh

$$\mathbb{P} = \left\{ P_i = (U_i, F_i, V_i) \;/\; i \in I \right\}$$

taková, že pro ni existuje parametrická soustava výroků $P(\mathcal{L})$, $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2, \ldots \mathcal{L}_n) \in \Sigma$, že platí:

Pro každé $i \in I$ existuje $\mathcal{L}^i \in \Sigma$ takové, že

$$P_i = P(\mathcal{L}^i) \;,$$

t.j. pro každou úlohu P_i soustavy $\mathfrak{P}$ existuje taková hodnota tohoto parametru $\alpha = \alpha^i$ v množině $\mathfrak{A}$ hodnot parametru α, že konkrétní úloha definovaná soustavou výroků P_i je totožná s úlohou definovanou soustavou výroků $P(\alpha^i), U_i = U(\alpha^i), V_i = V(\alpha^i), Y_i = Y(\alpha^i)$. Parametrickou soustavu výroků $P(\alpha), \alpha \in \mathfrak{A}$ nazveme typovou úlohou soustavy $\mathfrak{P}$ konkrétních úloh.

Typová úloha je tedy zobecněním skupiny konkrétních úloh pro níž existuje jednotná parametrická definice reprezentující všechny úlohy této skupiny.

Programový produkt /část programu, program, programový systém/ vyjadřující algoritmus řešení konkrétní úlohy P_i nazveme konkrétním řešením, označené R_i. Formálně jde o určitou transformaci definice konkrétní úlohy v níž zobrazení definované v úloze je uvedeno do explicitního tvaru prostřednictvím programového jazyka.

Jestliže pro typovou úlohu $P(\alpha), \alpha \in \mathfrak{A}$ existuje programový produkt $R(\rho), \rho \in \Gamma$ takový, že pro každou konkrétní úlohu P_i reprezentovanou typovou úlohou $P(\alpha)$ s hodnotou parametru $\alpha = \alpha^i$ existuje $\rho^i \in \Gamma$ takové, že $R(\rho^i)$ je řešením úlohy P_i, pak programový produkt $R(\rho), \rho \in \Gamma$ nazveme typovým /projektovým/ řešením odpovídajícím typové úloze $P(\alpha), \alpha \in \mathfrak{A}$.

Proces tvorby a využití typového prvku reprezentovaného typovou úlohou a odpovídajícím typovým řešením lze znázornit tímto schématem /schéma 1/:

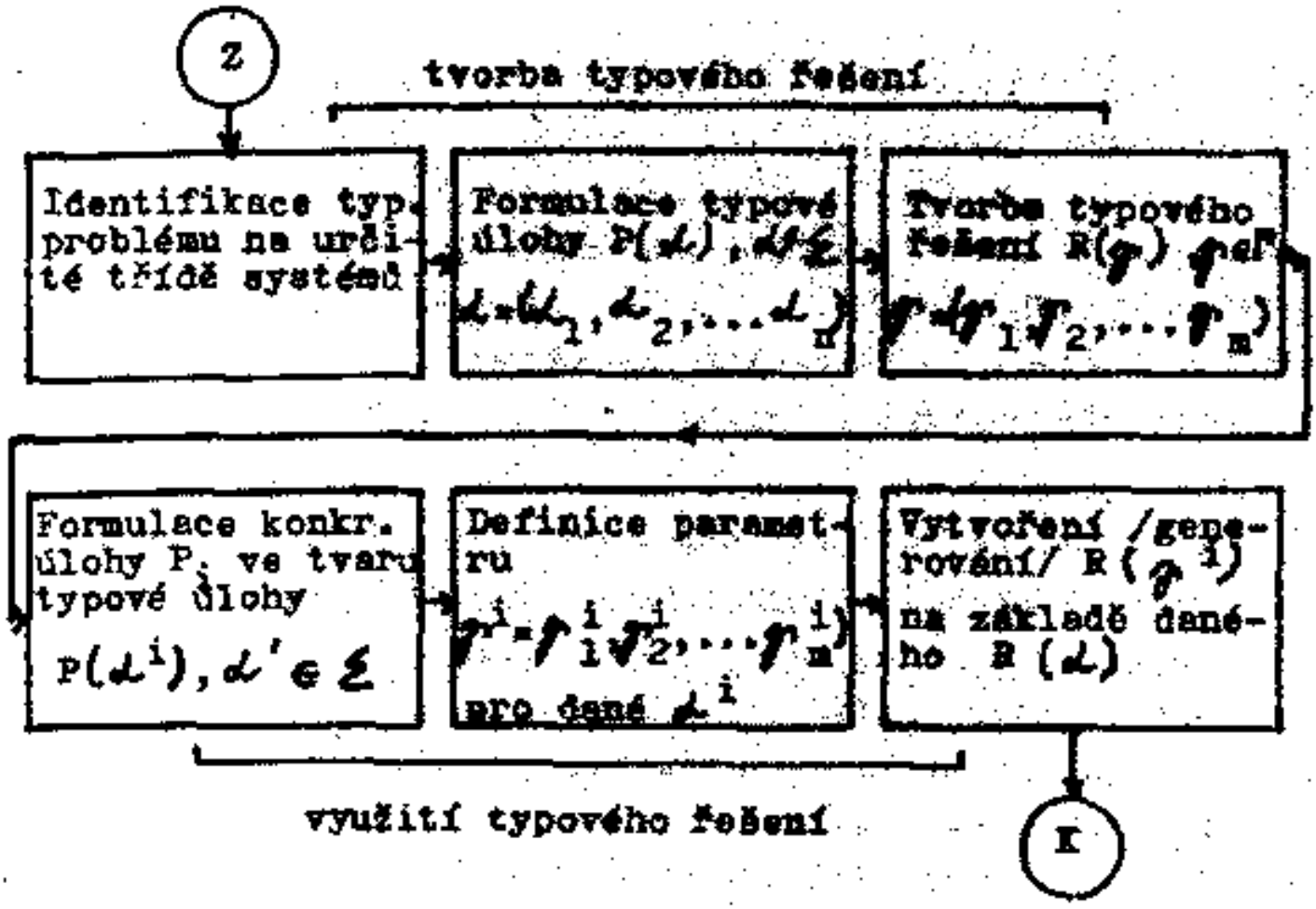


Schéma 1.

Perspektivním cílem typizace je dosažení maximálního stupně automatizace celého procesu tvorby a využití typových prvků. Pro omezený rozsah tohoto příspěvku nebudeme se zde zabývat podrobnější charakteristikou této problematiky. Poznamenejme jen, že v této souvislosti vyvstávají otázky tvorby formálních jazyků pro popis úloh, automatizace rozpoznání konkrétní úlohy jakožto člena třídy úloh reprezentovaných typovou úlohou a převod formulace konkrétní úlohy do tvaru varianty typové úlohy, dále pak tvorbu generátorů /reprezentujících typová řešení/ schopných na základě konkrétních variant typových úloh generovat konkrétní řešení a některé další otázky.

I když dosažení uvedeného cíle typizace je otázkou perspektivní, výzkumné řešení problémů podmiňujících dosažení tohoto cíle je aktuální otázkou. Reálným požadavkem těchto dnů se jeví řešení alespoň automatizace poslední etapy využití typového řešení, t.j. automatické vytvoření /vygenerování/ konkrétního řešení na základě konkretizovaných parametrů typového řešení.

Požadavky na vlastnosti typových řešení

Pod obecnými vlastnostmi typových řešení rozumíme vlastnosti společné pro všechna typová řešení bez ohledu na jejich problémový obsah. Požadavky některých vlastností jsou odvozovány od jejich základní uživatelské funkce, kterou je možnost kompozice typových řešení v rámci určité třídy systémů v modifikaci definované konkrétní úlohou. Vzhledem k tomu, že hovoříme o typových řešeních aplikačního programového vybavení je přirozené, že splnění všech požadavků kladených na programové vybavení je evidentní /požadavky z hlediska údržby, použití vhodných projektových postupů, metod programování a pod./.

Věnujme pozornost těm požadavkům na vlastnosti typových řešení jejichž splněním je zaručeno, že programový produkt skutečně typové řešení reprezentuje. Mezi základní požadavky tohoto typu zařazujeme:

1. potenciální mnohonásobná využitelnost typového řešení požadující, aby obsahová náplň typové úlohy pro níž je typové řešení vytvářeno byla podrobně analysována z hlediska výskytu v dostatečně četné třídě systémů,
2. účelová flexibilita typového řešení. Tento požadavek stanoví, aby parametrizace /v obecném smyslu/ typové úlohy a z toho vyplývající parametrizace typového řešení byla provedena tak, aby na jedné straně zaručovala možnost modifikace zapojení a funkce v co nejčetnější třídě konkrétních systémů, na druhé straně aby rozsah

této parametrizace byl s hlediska uživatele účelově minimalizován - účelově zvolen v nutné míře splňující podmínku efektivnosti využití typového řešení,

3. standardizace - zahrnutí do typového řešení v maximální míře standardů, standardních postupů jak z hlediska obsahového tak z hlediska formálních prostředků /progr. jazyky, op. systémy, metody řešení a pod./,
4. automatizace procesu tvorby konkrétních řešení, t.j. automatizace procesu konkretizace parametrů typového řešení podle požadavků typové úlohy,
5. hierarchická strukturalizace a modularita umožňující tvorbu typových řešení na základě využití existujících typových řešení,
6. srozumitelnost.

2 podrobnější specifikace těchto požadavků by měla vzniknout kriteria pro hodnocení programových řešení z hlediska jejich"typovosti".

Jeden konkrétní přístup k tvorbě typových řešení

Celkový efekt procesu typizace v oblasti aplikačního programového vybavení je podmíněn jak obsahovou stránkou typových řešení tak i formální podmiňující zásadně jejich využitelnost.

Nebudeme se zde zabývat problematikou řešení věcné náplně, která je svázána vždy s příslušným problémovým obsahem. Výsledkem tohoto řešení by měla být zformována typová úloha v pojetí vymezeném v předchozím výkladu, t.j. parametrizovaný popis def. oborů vstupních a výstupních informací a zobrazení vstupů do výstupu. Další etapou je pak formulace typového řešení.

Současné programovací prostředky jsou v podstatě konstruovány pro tvorbu konkrétních řešení. Obsahují sice určité prostředky pro parametrizaci programových řešení, ale

z podrobnější analýzy možných požadavků kladených typovou úlohou na typová řešení zjišťujeme, že tyto prostředky jsou značně omezené, nebo realizace těchto požadavků je velmi náročná.

Jedním, velmi efektivním přístupem k tvorbě typových řešení je přístup založený na filosofii makrojazyka - přístup založený na automatizaci úprav určitých účelově parametrizovaných programových textů /makrodefinic/ do tvaru konkrétních programových textů reprezentujících konkrétní řešení, což v principu odpovídá vymezenému pojetí typových řešení.

Univerzální řešení této koncepce řešení nezávislé na konkrétních programovacích jazycích, tzv. systém programových schémat, jehož základní idea a stav současného řešení je předmětem dalšího výkladu.

Typovou úlohu lze chápat jako obecnou strukturu, v níž na různých rozlišovacích úrovních jsou předpokládány určité modifikace /viz schéma 2./.

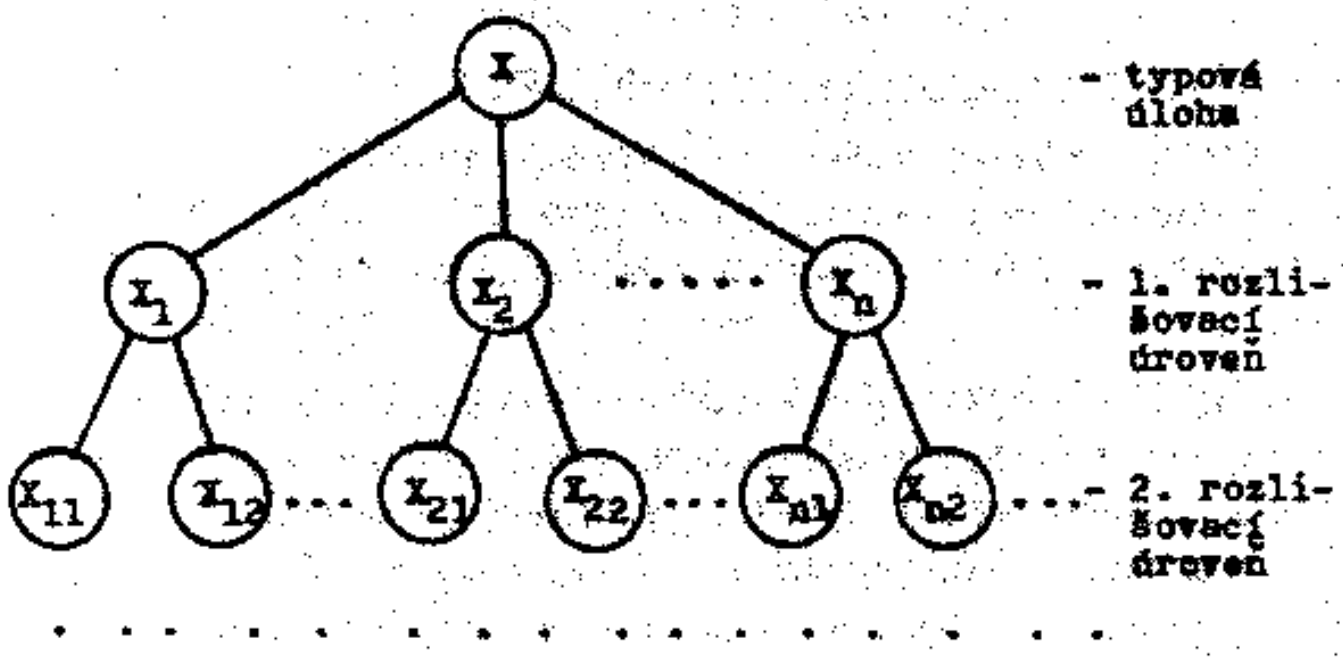
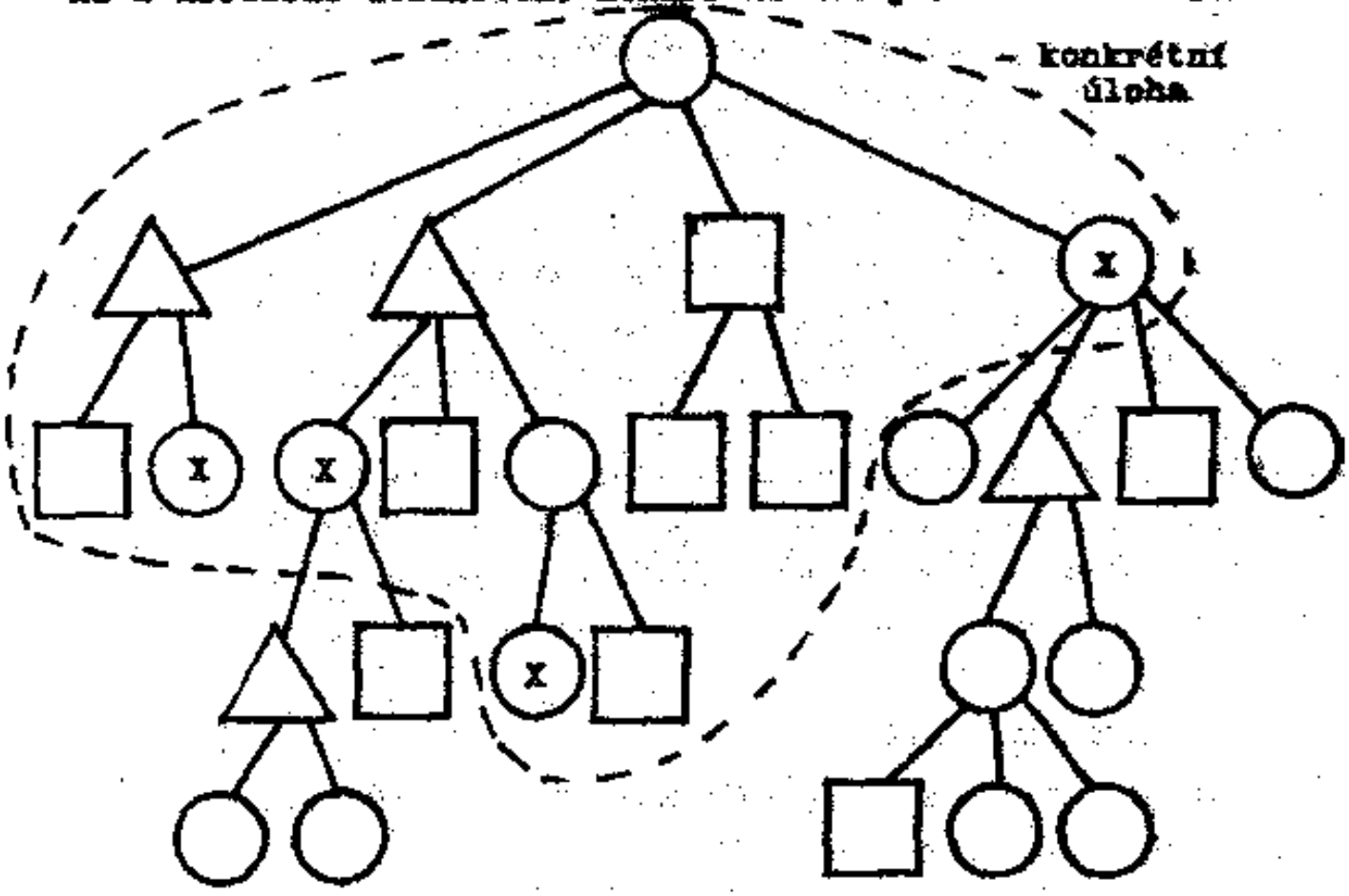


Schéma 2.

Prvek $X_{i1,i2}$ ... reprezentuje jednu z následujících tří variant:

1. konkrétní část typové úlohy z hlediska příslušné rozlišovací úrovně,
2. abstraktní část reprezentující třídu přípustných variant daných požadavkytřídy konkrétních úloh dané typové úlohy, při čemž tomuto prvku je explicitně přiřazena určitá standardní varianta pokud takovou variantu lze definovat,
3. konstantní část z hlediska všech následujících rozlišovacích úrovní.

Prvky typu 1. a 2. na další rozlišovací úrovni obsahují alespoň jeden prvek typu 2. Prvky typu 3. obsahují zase jen prvky typu 3. Konkrétní úloha na základě dané typové úlohy vznikne tak, že pro každou větev na určité rozlišovací úrovni nahradíme abstraktní nebo konkrétní prvek /pokud existuje/ určitou konstantní variantou konsistentní z hlediska definované konkrétní úlohy /viz schéma 3./.



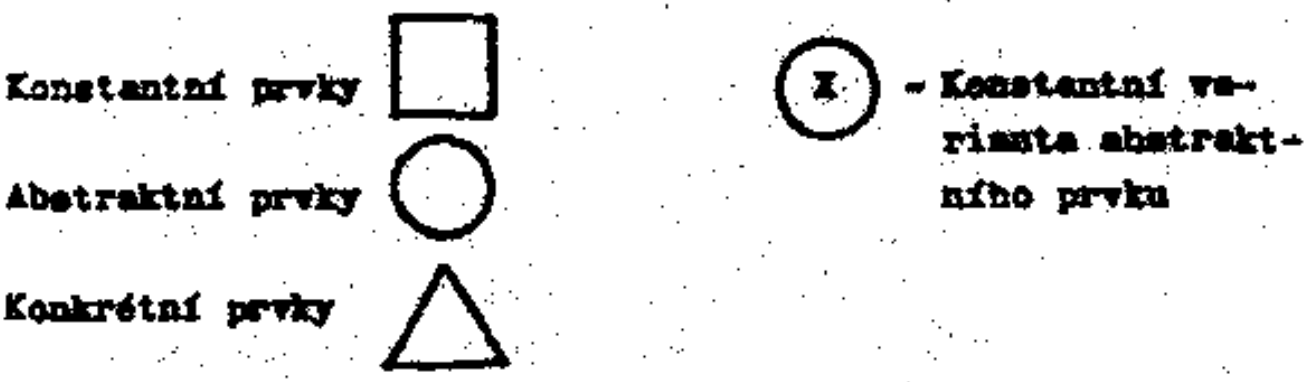


Schéma 3.

Pak prvky podřízené v dané hierarchii těmto prvkům ztrácejí význam a konkrétní úloha je representována stromem /subgrafem původního stromu/ v němž každá větev je ukončena konstantním prvkem nebo konstantní variantou.

Stejným způsobem lze zobrazit odpovídající typová řešení /při čemž bohužel obecně nelze zaručit jedno-jednoznačnou korespondenci mezi prvky definované ve struktuře typového řešení/.

Koncepce programových schémat je tedy založena na tvorbě hierarchicky strukturovaných programových textů, v nichž úseky representují abstraktní prvky jsou identifikovatelné a automaticky zaměnitelné jinými úseky.

V základní variantě tohoto řešení určitá vzájemná podřízenost volby konkrétních variant jednotlivých prvků je umožněna vhodným identifikačním systémem těchto prvků, což samo o sobě dává zajímavé možnosti při zachování přehlednosti základní logiky typových řešení. V další verzi se uvažuje o způsobu rozšíření základní varianty o soustavu příkazů umožňujících realisaci tvorby těchto typových řešení a zvýraznění možnosti podmíněného řízení procesu generování, což s druhé strany vnese do řešení určitou složitost.

Systém programových schémat

V ÚVTR je v současné době připravována do uživatelské formy základní verze systému.

Systém je řešen na základě operačního systému DOS. jeho základní struktura je následující /viz schéma 4./:

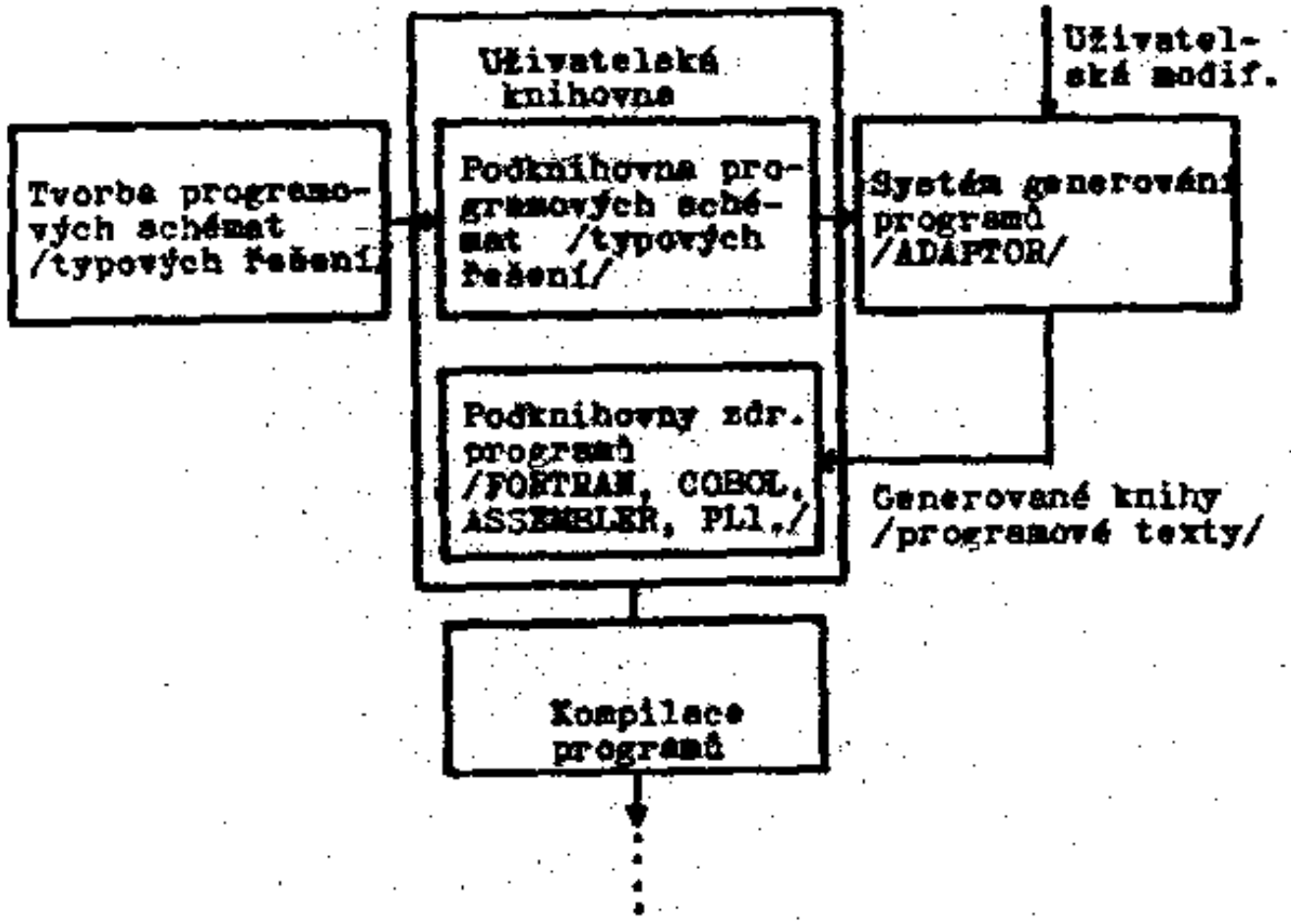


Schéma 4.

Typové řešení v systému programových schémat má formu tzv. programového schématu, jehož struktura je následující:

```
# <jméno PS>
   $MODIFIERS:        seznam modifikátorů
   $VARIANTS:         seznam variant
   $SPECIFICATIONS:   seznam specifikací
   $TEXT:             operační část
```

Operační část programového schématu uvedena za $TEXT: představuje jádro programového schématu. Obsahuje popis věcného obsahu typového řešení vytvořený na základě následujících syntaktických pravidel:

⟨operační část⟩:: =⟨prvek operační části⟩ |
⟨operační část⟩ ⟨prvek operační části⟩

⟨prvek operační části⟩:: = ⟨úsek programu⟩ | ⟨modifikátor⟩ |
⟨volání PS⟩ | /

⟨úsek programu⟩:: (⟨libovolný text vytvořený podle pravidel libovolného programovacího jazyka⟩)'

⟨modifikátor⟩:: = &⟨jméno modifikátoru⟩ * |
&⟨jméno modifikátoru⟩ : ⟨operační část⟩ * |
• ⟨modifikátor⟩

⟨volání PS⟩:: = # ⟨jméno PS⟩ * |
⟨jméno PS⟩ : ⟨seznam modifikací⟩ *

⟨seznam modifikací⟩:: = ⟨prázdný⟩ | ⟨prvek modifikace⟩ |
⟨seznam modifikací⟩ ⟨prvek modifikace⟩

⟨prvek modifikace⟩:: = ⟨modifikace⟩ | ⟨sdružená modifikace⟩

⟨sdružená modifikace⟩:: = &⟨jméno varianty⟩ *

⟨modifikace⟩:: = ⟨identifikace modifikátoru⟩ : ⟨operační část⟩ *

⟨identifikace modifikátoru⟩ :: = &⟨jméno modifikátoru⟩ |
⟨identifikace modifikátoru⟩@⟨jméno modifikátoru⟩

Všechna jména jsou definována jako posloupnosti znaků neobsahující znaky : , @ , = , *. Použití znaku = je ekvivalentní s použitím znaku :. Mezery jsou ve jménech ignorovány. Délka jména není omezená, jméno jednoznačně určuje prvních 8 znaků /bez mezer/.

Část programového schématu uvedená za MODIFIKÁTORY má dokumentační význam, je určena pro uživatelský popis významu jednotlivých modifikátorů. Její obsah je libovolný s tím, že má být uzavřená v soustavě závorek, v níž znaky &, # representují otevírací závorky, k nimž je znak * - uzavírací závorkou a dále dvojici otevíracích a uzavíracích

závorek reprezentují (,)' .

Část uvedená za &VARIANTS: reprezentuje skupiny modifikací sdružené v tzv. variantě. Formálně:

<seznam variant>:: =<varianta>|<seznam variant><varianta>
<varianta> :: =&<jméno varianty> : <seznam modifikací>*

Část za &SPECIFICATION: obsahuje popisy programových schémat /tzv. lokálních programových schémat/, jejichž volání se může vyskytovat v operační části programového schématu obsahující jejich popisy nebo v operačních částech uvedených ve variantách nebo v modifikacích při volání daného programového schématu.

Je-li určité programové schéma voláno, pak namísto volání je v průběhu generování dosazen obsah definovaný operační částí volaného programového schématu a v případě modifikátoru buďto operační část přiřazená modifikátoru v seznamu modifikací nebo standardní obsah uvedený za daným modifikátorem. Volaná programová schémata musí být popsána buďto v části specifikací volajícího programového schématu nebo uvedená samostatně jako kniha v knihovně programových schémat. Je-li v operační části přiřazen modifikátoru v seznamu modifikací modifikátor ve tvaru

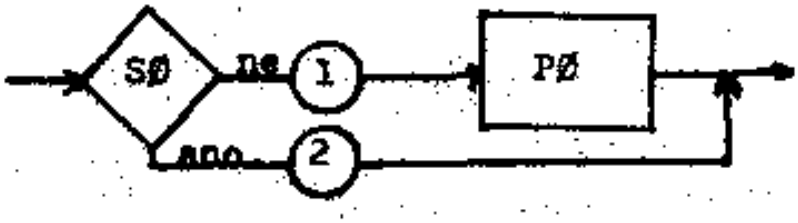
. &<jméno modifikátoru> .....

pak je tento modifikátor považován za modifikátor volajícího programového schématu.

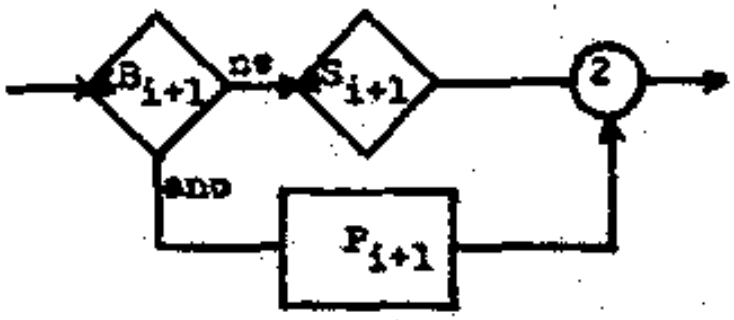
Znak / v operační části reprezentuje konec generování celku, který je v knihovně zdrojových programů katalogizován jako samostatná kniha.

Pro ilustraci uvádíme příklad mnohonásobného větvení programu, jehož obecnou strukturu vyjadřují následující schémata:

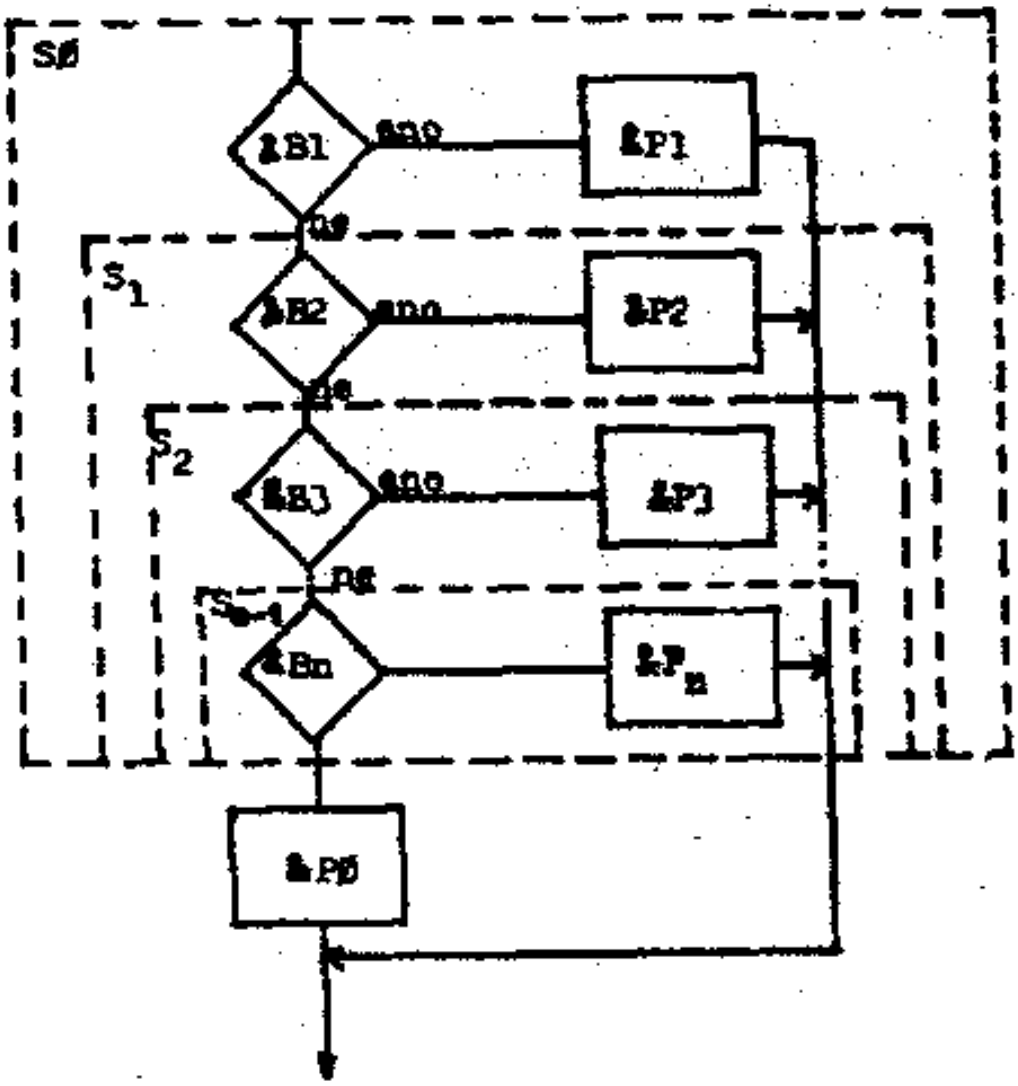
1. rozlišovací úroveň



při čemž blok S_i, i=0,1 ... n, má následující strukturu:



t.j. celkové schéma vypadá následovně:



V odpovídajícím programovém schématu chceme modifikovat:

- počet větvení
- obsahy podmínek $\&B_i$
- obsahy bloků $\&P_i$,

při čemž maximální počet větvení je 10.

Tuto velmi elementární typovou úlohu lze řešit např. tímto programovým schématem:

```
*VETV:
   &MODIFIERS: B1, B2, ... B10
               PØ, P1, ... P10
               SØ, S1, ... S10 *.
   &TEXT:
   &SØ: (IF) '&B1 *(THEN PERFORM)'&P1 *(ELSE)'
   &S1: (IF) '&B2 *(THEN PERFORM)'&P2 *(ELSE)'
   &S2: (IF) '&B3 *(THEN PERFORM)'&P3 *(ELSE)'
   &S3: (IF) '&B4 *(THEN PERFORM)'&P4 *(ELSE)'
   &S4: (IF) '&B5 *(THEN PERFORM)'&P5 *(ELSE)'
   &S5: (IF) '&B6 *(THEN PERFORM)'&P6 *(ELSE)'
   &S6: (IF) '&B7 *(THEN PERFORM)'&P7 *(ELSE)'
   &S7: (IF) '&B8 *(THEN PERFORM)'&P8 *(ELSE)'
   &S8: (IF) '&B9 *(THEN PERFORM)'&P9 *(ELSE)'
   &S9: (IF) '&B10*(THEN PERFORM)'&P10*(ELSE)'
        * * * * * * * * * *
        (PERFORM)' &PØ *(.)'**
```

Volání tohoto programového schématu ve tvaru:

```
   VETV: &S2=*, &B1=(U<V)', &P1=(A)', &B2=(U=W)', &P2=(B)', &PØ=(C)'**
```

reprezentuje řešení:

```
        IF  U<V  THEN PERFORM A ELSE
        IF  U =W  THEN PERFORM B ELSE
            PERFORM C .
```

Možnost definování rozsáhlých programových schémat umožňuje generovat celé programové systémy, při čemž společné modifikace v jednotlivých programech jsou prováděny jen jednou.

Příklad: Mějme existují programová schémata:

```
      *A:
        &MODIFIERS: *A1* &A2* &A3 * * ....
        &TEXT: ...                    * *
```

```
#B:
  &MODIFIERS: &B1*&B2**&B3*, B4**....
  &TEXT: .....                    **
 #C:
  &MODIFIERS: &C1*&C2*&C3**.....
  &TEXT: .......                  **
```

reprezentující programy.

Nechť v typovém řešení programového systému jsou využita programová schémata A, B, C, při čemž je požadováno aby modifikátory A1, B1, C1 v odpovídajících schématech plnily totožnou funkci (modifikátor) U, modifikátor A2, B2, C2 totožnou funkci V, ostatním modifikátorům (A3, B3, B4, C3) je přiřazen konkrétní obsah. Takto vzniká programové schéma schématicky znázorněné takto:

```
#W:
  &MODIFIERS:  &U*, &V*, ....
  &TEXT: ...  #A: &A1=.&U*, &A2=.&V**, &A3= ....**
         ...  #B: &B1=.&U*, &B2=.&V**, &B3= ...* ,
                  &B4=.....* *
         ...  #C: &C1=.&U*, &C2=.&V**, &C3= ....**
         ..... *
```

Volánim W: &U = ...
&V = ...

jsou modifikována požadovaným způsobem programová schémata A, B, C.

Tolik k charakteristice metody programových schémat. V rámci ověřování této metody bylo kromě vyložené experimentálních programových schémat vytvořeno programové schéma GNPS reprezentující typové řešení úlohy sekvenčního zpracování u souborů řízených společným klíčem (analogie generátorů normovaného programování). Toto programové schéma je důsledně řešeno podle zásad strukturovaného programování, je vypracováno na bázi jazyka COBOL, obsahuje standardizované řešení všech oddílů (oddíl identifikací, oddíl zařízení,

oddíl dat a oddíl procedur). Uživateli dovoluje generování konkrétních programů uvedeného typu nebo jednoduchou tvorbu specializovaných programových schémat v rámci tvorby typových řešení částí ASŘ. Přes to, že GNPS obsahuje poměrně značný počet modifikátorů zabezpečujících dostatečný uživatelský komfort, při dodržení zásady maximální standardizace uživatel řeší pouze upřesnění popisu vstupních a výstupních souborů, definování uživatelské pracovní oblasti a obsah následujících procedur:

$HEAD\emptyset$, HEAD1, ..., $HEAD_n$	- úvodní zpracování na všech úrovních změny řídícího klíče n - počet elem. položek řídícího klíče
$FOOT\emptyset$, FOOT1, ..., $FOOT_n$	- závěrečné zpracování na všech úrovních změny klíče
PROCF1,PROCF2,...,$PROCF_m$	- zpracování vět jednotlivých vstupních souborů m - počet vstupních souborů

GNPS sám o sobě prakticky využitelný programový produkt ilustruje především možnosti praktického uplatnění metody programových schémat pro relativně jednoduché, principiálně jednotné efektivní řešení generátorů programových soustav na bázi vyšších programovacích jazyků.