P E R S I S T E N T N Í S O F T W A R E

Ing. E. KOPČIL

V současném stadiu vývoje programátorské profese obvykle očekáváme od kvalitního softwarového produktu zhruba tyto vlastnosti:

- S P R Á V N O S T (dodáváme-li programu korektní vstupy, dostaneme správné výstupy)
- S P O L E H L I V O S T (nekorektní vstupy ani defekt hardware nezpůsobí chybu, ale pouze chybové hlášení)
- S R O Z U M I T E L N O S T (funkce programu musí s maximální názorností plynout z jeho zdrojového textu)
- U D R Ž O V A T E L N O S T (funkci programu musí být možno modifikovat - souvisí se srozumitelností)

Souhlasit s tímto tvrzením je nesrovnatelně snadnější než je prakticky realizovat, a proto v reálném životě musíme velmi často neohroženě přihlížet k rutinnímu provozování nejednoho projektu, který uvedené zásahy nesplňuje ani zdaleka a nejsme schopni zabránit nejenom dalšímu uvádění, ale dokonce ani předejít novému a novému vyvíjení takového "HORROR-WARE".

Při budování rozsáhlého systému je běžné zjednodušit (za cenu zanedbání některých méně významných vazeb) řešenou problematiku tak, aby ji bylo možno rozčlenit na menší celky, umožňující postupnou realizaci systému při respektování možností a kapacit řešitele.

S postupující realizací systému však zákonitě roste nutnost respektovat i ty vazby, které mohly být v předchozích fázích vývoje systému ignorovány. Každému dostatečně rozsáhlému systému

(fungujícímu v reálných, tedy vyvíjejících se podmínkách) proto hrozí nebezpečí přechodu dalšího vývoje v nekonečnou údržbu jeho zprovozněných částí.

Jedná se o problém, limitující nejenom možnosti rozvoje systémů, ale vlastně i existenci programátorské profese v její nynější poměrně tvůrčí formě.

TENTO STAV NELZE ZVRÁTIT ANI ÚSPĚŠNÁ REALIZACE PROKLAMACÍ TYPU:
"PROGRAMOVÁNÍ - DRUHÁ GRAMOTNOST"
(nebo převedeno do našich podmínek : DO ČECH - TO PROGRAMETOR)

Nemíníme-li se dostat do slepé uličky, musíme bezpodmínečně zajistit radikální pokles pracnosti údržby a úprav zprovozněných částí systémů bez ohledu na to, zda se jedná o vnitřní potřebu v rámci postupného rozvoje systému, nebo o reakci na změnu v okolí systému.

K dosažení tohoto cíle je třeba zvýšit kompatibilitu softwarového aparátu se zvýšeným důrazem na jeho PERSISTENCI (to jest stabilitu funkce programů v podmínkách proměnného okolí):

- zaměřit se na maximální využívání částí zdrojových textů, uložených mimo program (COPY - knihovny),
- stanovit, jaké změny jsou přípustné při úpravách datových struktur, aniž by bylo nutno zasahovat do algoritmů,
- stanovit, které úseky algoritmů programů je třeba přebírat z COPY - knihovny, aby byla optimalizována necitlivost vlastních zdrojových textů ke změnám datových struktur,
- softwarovým sledováním změn, prováděných ve zdrojových textech a v COPY - knihovnách umožnit automatizaci provádění těchto změn (modifikace COPY - knihoven, zdrojových textů, jejich rekompilace,...).

V zájmu účinného zajištění FLEXIBILITY (to jest přizpůsobivosti funkce programů změnám v okolí) je dále potřeba:

-- vyvinout generátory programů orientované na analytiky (případně na uživatele), umožňující "programování bez programátora", ale hlavně eliminující vznik neudržovatelných programů:
   -- vkládání případně pořizování dat s možností indikace chyb a s možností provádění oprav a změn,
   -- výstup dat (ať už ve formě tiskových sestav, nebo jinak) včetně výběrů, případně třídění,
   -- "mixování"dat z různých zdrojů.

- minimalizovat pracnost při vytváření dokumentace (a tedy i nebezpečí její neaktuálnosti) tím, že pasáže dokumentace závislé na změnách ve zdrojovém textu, podobně jako pasáže závislé na datových strukturách, které budou rovněž AUTOMATIZOVANĚ "vytahovány" z příslušné COPY - knihovny.

Je vhodné zdůraznit, že persistence software se nemůže pozitivně projevit tam, kde se jedná o práci malého rozsahu, ani tam, kde promítnutí potřebných změn vede k revolučnímu zásahu do funkce nebo struktury celého systému.

V případech postupného rozvoje a vývoje rozsáhlých systémů však k zásadním zvratům nedochází (aspoň by nemělo), takže jsme většinou nuceni řešit právě opačnou otázku - jak zajistit neměnnost funkce řady programů NAVZDORY provedeným změnám. V těchto případech jedině persistence vytvořeného softwarw umožní úspěšné dokončení úkolu. Cílem tohoto referátu je proto navrhnout metodiku tvorby persistentního software na bázi jazyka COBOL.

1. "DATA - LESS" PROGRAMY

Teoretické předpoklady v poměrně dobré shodě s dlouhodobými zkušenostmi nám říkají, že nejčastěji se při rozvoji programového vybavení vyskytují změny,vyvolané :

- potřebou doplnit nové informace do stávajících datových struktur

(např. při napojování na subsystém mezd zjistíme, že v subsystému dopravy scházejí potřebná osobní čísla závozníků) .....odhadem 60 % změn

- aktualizací a novelizací číselníků vyhlašovaných mimo systém, nebo i vlastních
  (např. nadřízený orgán ohlásí změny ve struktuře statistických údajů, automatizovaně vykazovaných subsystémem dopravy).....odhadem 30 % změn
- změnami metodiky zpracování dat, mající dopady na stávající algoritmus výpočtu
  (např. bude rozhodnuto, že odměňování řidičů má nadále záviset na vyfakturované hodnotě, místo dosavadních ujetých kilometrů).....odhadem 10 % změn

Četnost posledního typu změn je podhodnocena pouze zdánlivě. Při systématické práci s číselníky totiž lze významnou část změn algoritmů realizovat prostřednictvím změn v číselníku.

V takovéto úvaze má své kořeny i jeden z nedávných hitů v oblasti programování - DATABÁNKA. Bohužel ani v zahraničí dosud není využívání databanky zcela běžné a v našich podmínkách se pro dohlednou budoucnost jako příliš reálné rozhodně nejeví.

Při udržování programů je největším problémem vysoká pracnost a zejména malá spolehlivost "hlavoručního" vyhledávání míst ve zdrojových textech, do kterých je nutnost zasahovat v případech změn struktury, nebo počátečního nastavení hodnot datových proměnných.

Praktickým dopadem je, že po každé změně je nutno zopakovat minimálně otestování, často však i nové a pracné ladění už odladěných programů.

Abychom tomu mohli předejít, musíme odstranit nežádoucí citlivost programů, (jak ke změnám datových struktur, tak i k převážné části změn číselníků) tím, že nebudeme ve zdrojových textech programů používat žádná data, ani jejich deklarace:

- v deklaracích ani v algoritmech nepoužívat textové řetězce ani numerické konstanty (s vyjímkou čísel úrovní v deklaracích a snad s vyjímkou inkrementační jedničky v algoritmech).

- veškeré deklarace datových struktur (skupinových a zejména elementárních položek) přebírat z COPY - knihovny.

Soustředěním veškerých informací o datech zpracovávaných v systému do COPY - knihoven (nemusí být jediná) vznikne METABÁZE SYSTÉMU, která (s multistromem vazeb mezi texty knihoven) zajistí jak dokonalý přehled o datových strukturách a jejich použití v systému, tak předpoklady pro automatizaci prací při provádění nutných změn.

Pro programy splňující uvedené zásady zavedeme pracovní označení

" D A T A - L E S S "

Využitím této programovací techniky dojde k transformaci problému údržby programů na problém tvorby a aktualizace logicky konsistentní METABÁZE, který je bezesporu rovněž velmi náročný, ale rozhodně daleko lépe řešitelný.

Pro řadového programátora bude (za předpokladu existence správné, spolehlivé, aktuální a přehledné METABÁZE) akceptování těchto zásad poměrně dobře přijatelné, protože :

- podíl literalů v běžném programu je zanedbatelný, takže používat na jejich místě konstanty z METABÁZE nepřinese žádné potíže,

- práci s psaním deklarací (obzvláště opakovaných) si každý rád odřekne,

- lze celkem snadno a AUTOMATIZOVANĚ zjistit, zda program je nebo není "DATA-LESS".

Prvními kroky na této cestě musí být :

- navržení vhodné a kontrolovatelné metodiky programování,

- implementace programového aparátu METABÁZE, pro spolehlivé a bezpečné užívání COPY - knihoven.

2. ELEMENTÁRNÍ ČÍSELNÍKY

Základním předpokladem realizace jakéhokoliv řízení je možnost vzájemného porovnávání jednotlivých řízených procesů a předmětů.

Mají-li jednotlivé řízené procesy i předměty vesměs unikátní charakter, vzniká potřeba zobecňování s cílem vytvořit vhodné skupiny procesů nebo předmětů, v rámci kterých by bylo možné provádět potřebná porovnávání.

Pro klasifikaci jednotlivých skupin procesů a předmětů se obvykle používají číselníky (například "charakter dodávky" - viz dále).

Popisy vlastností jednotlivých procesů nebo předmětů v rámci jisté skupiny (např. ceníky stavebních prací a materiálů) lze rovněž chápat jako číselníky.

Z hlediska zpracování dat lze na číselník pohlížet jako na datový soubor, jehož každá věta obsahuje dvě části :

- K L Í Č O V É S L O V O - identifikuje proces, předmět nebo skupinu procesů nebo předmětů
- D A T O V Á Č Á S T - popisuje, jaké má proces nebo předmět vlastnosti, nebo jaké vlastnosti musí mít, aby patřil do popisované skupiny

Pro úplnost poznamenejme, že je zvykem třídit věty číselníkových souborů podle hodnoty klíčového slova (zpravidla sestupně).

Číselníky byly pro potřeby řízení používány už dávno před nástupem výpočetní techniky (jedním z nejstarších a nejhlouběji propracovaných komplexů číselníků je účetní osnova).

Své postavení si číselníky v automatizovaném procesu řízení nejen udržely, ale dokonce i upevnily, protože mohou sloužit rovněž k řízení samotného procesu řízení : vhodnou změnou v číselníku můžeme zdůraznit právě ty procesy, které jsou pro řízení v dané situaci rozhodující.

Jako příklad práce s číselníkem můžeme uvést vývoj resortního číselníku ministerstva stavebnictví "charakter dodávky" :

př. 1

PRO 7. PĚTILETKU PLATIL ČÍSELNÍK :

```
1....S-hodnota na invest. výstavbě - nové stavby
2 ...S-hodnota na invest. výstavbě - rekonstrukce
3 ...S-hodnota na invest. výstavbě - přímé dodávky
4 ...S-hodnota - výstavba v zahraničí
5 ...S-hodnota - opravy
7 ...S-hodnota - ostatní výstavba
```

PRO 8. PĚTILETKU BYL VYHLÁŠEN ČÍSELNÍK :

```
1 ...S-hodnota na invest. výstavbě - nové stavby
2 ...S-hodnota na invest. výstavbě - rekonstrukce a modernizace
3 ...S-hodnota na invest. výstavbě - přímé dodávky
4 ...S-hodnota na invest. výstavbě - stroje a zařízení nezahrnuté do rozprac.
5 ...S-hodnota - vybrané integrační akce
6 ...S-hodnota - výstavba v zahraničí
7 ...S-hodnota - opravy spojené s rekonstrukcí a moderniz.
8 ...S-hodnota - opravy ostatní
9 ...S-hodnota - ostatní výstavba
```

└ KLÍČOVÉ SLOVO └ DATOVÁ ČÁST

Uvidíme zde např. zvýšený důraz kladený na opravy spojené s rekonstrukcí a modernizací, který se promítl rozdělením skupiny (5) do dvou nových skupin (7,8) u číselníku pro 8. pě-

tiletku.

V prostředí automatizovaného systému řízení přebírají číselníky ještě další cennou funkci : číselníky tvoří poměrně nenásilnéa přirozené rozhraní (interface) mezi člověkem a počítačem.

Navzdory nezastupitelné roli, kterou číselníky hrají v procesu automatizovaného řízení, je většina stávajících číselníku koncipována tak, jak bylo zvykem před nástupem výpočetní techniky, takže potřebám současného automatizovaného zpracování většinou příliš nevyhovují.

Počítač, pro který není problémem vyřešení sebesložitější úlohy, je-li její zadání exaktně formulováno (např. vyřešení rozsáhlé soustavy rovnic), totiž zcela spolehlivě ztroskotá, je-li postaven před neformálizovatelný úkol (např. nedokáže rozhodnout, je-li "OSTWALDOV" název nového města, nebo jedná-li se o chybný pravopis).

Je-li při zpracování nutno přihlížet k vlastnostem jednotlivých skupin definovaných číselníkem (např. rozpoznávat , které skupiny dle "charakteru dodávky" mají povahu "investiční výstavby", protože při jejich zpracování je nutno zvlášť evidovat tzv. inženýrskou přirážku), měla by datová část číselníku obsahovat potřebné informace o těchto vlastnostech natolik formalizovaně, aby byly pro počítač akceptovatelné.

Pro vyřešení výše uvedené problematiky rozlišování "investiční výstavby" lze číselníku "charakter dodávky" (viz. př. 1) použít pouze za předpokladu, že do programu uložíme další informaci o tom, ke kterým skupinám "charakteru dodávky" se váže vlastnost "investiční výstavba".

Protože lze oprávněně předpokládat, že u resortního číselníku "charakter dodávky" (př. 1) bude i v budoucnosti docházet ke změnám, nebylo by patrně prozíravé, kódovat informaci o "investiční výstavbě" přímo do zdrojových textů programů. Přijatelnějším řešením by zřejmě bylo zavedení číselníku, například "povaha dodávky";

Př.2

PRO 7. PĚTILETKU :

1 ... investiční výstavba: charakter dodávky = 1, 2, 3, 4
2 ... opravy : charakter dodávky = 5
3 ... ostatní výstavba : charakter dodávky = 0, 6, 7, 8, 9

KDEŽTO PRO 8. PĚTILETKU :

1 ... investiční výstavba: charakter dodávky = 1, 2, 3, 4, 5, 6
2 ... opravy : charakter dodávky = 7, 8
3 ... ostatní výstavba : charakter dodávky = 0, 9

Vlastně zde provádíme upřesnění původního číselníku, protože samotný číselník "charakter dodávky" naprosto nic neříká o tom, mají-li "přímé dodávky" a "práce v zahraničí" povahu investiční výstavby.

Automatizované využívání tohoto číselníku lze realizovat formouvyhledávání klíčového slova číselníku "charakter dodávky" v seznamu hodnot na konci datové části každé věty číselníku "povaha dodávky" (dokáže zajistit např. Jiříčkův "Překladač tabulek a číselníků" - viz sborník "Programování´76").

Je však zřejmé, že řešení by mohlo být daleko jednodušší, kdyby informace o povaze dodávky byly doplněny přímo do číselníku "charakter dodávky" tak, aby je bylo možno přebírat přímo odtamtud.

Jinak řečeno : mají-li se změny číselníků stát účinným nástrojem pro řízení řídícího procesu a ne prostředkem pro destabilizaci funkce stávajícího programového vybavení, je nezbytné pracovat s dostatečně elementárními pojmy, protože :

- pouze skutečně elementární pojmy jsou vhodně formalizovatelné
- elementárnější pojmy mívají stabilnější význam

Dodržení této zásady tvorby software uživatel nijak nepocítí, protože je možné podržet se stávajících číselníků a pouze zajistit jejich doplnění a odkazy na elementární číselníky, vhodné pro automatizované zpracování :

Př.3

PRO 7. PĚTILETKU BY PLATIL ČÍSELNÍK :

1 ... 1121 S-hodn. na inv. výst. - nové stavby
2 ... 1111 S-hodn. na inv. výst. - rekonstrukce
3 ... 1122 S-hodn. na inv. výst. - přímé dodávky
4 ... 1221 S-hodn. - výstavba v zahraničí
5 ... 2121 S-hodn. - opravy
7 ... 3121 S-hodn. - ostatní výstavba

PRO 8. PĚTILETKU BY BYL VYHLÁŠEN ČÍSELNÍK :

1 ... 1121 S-hodn. na inv. výst.- nové stavby
2 ... 1111 S-hodn. na inv.v.-rekonstrukce a modernizace
3 ... 1122 S-hod. na inv. výst. - přímé dodávky
4 ... 1121 S-hodn. na inv.v. - stroje a zaříz. mimo rozpočet
5 ... 1221 S-hodn.- vybrané integrační akce
6 ... 1221 S-hodn. - výstavba v zahraničí
7 ... 2111 S-hodn. - opravy spojené s rekonst. a modernizací
8 ... 2121 S-hodn. - opravy ostatní
9 ... 3121 S-hodn. - ostatní výstavba

KDE ČÍSLA NA ZAČÁTKU DATOVÉ ČÁSTI MAJÍ VÝZNAM :
1=dodávky dle smluv, 2=přímé dodávky
1=rekonstrukce a modernizace, 2=jiné
1=tuzemsko, 2=zahraničí
1=investice, 2=opravy, 3=ostatní

Všechny programy pro 7. pětiletku (pracující s elementárními pojmy) by mohly s takovýmto číselníkem správně fungovat bez jakéhokoliv zásahu také v 8. pětiletce.

Pokud by funkce programu měla být doplněna o sledování dalších (nových) charakteristik, bylo by nutno přikročit k za-

vedení dalších elementárních číselníků :

Př. 4

PRO 8. PĚTILETKU BY POTOM MOHL BÝT VYHLÁŠEN ČÍSELNÍK :

```
1 ... 112111 S-hodn.na inv.výst. - nové stavby
2 ... 111111 S-hodn.nainv.výst.  - rekonstrukce a modernizace
3 ... 112211 S-hodn.na inv.výst. - přímé dodávky
4 ... 112121 S-hodn.na inv.výst. - stroje a zařízení mimo roz.
5 ... 122112 S-hodn. - vybrané integrační akce
6 ... 122111 S-hodn.-- výstavba v zahraničí
7 ... 111111 S-hodn. - opravy spojené s rekons-a modernizací
8 ... 212111 S-hodn. - opravy ostatní
9 ... 312111 S-hodn. - ostatní výstavba
```

KDE NOVĚ DOPLNĚNÁ ČÍSLA MAJÍ VÝZNAM :
1=běžná akce, 2= V I A
1=dle rozpočtu, 2=mimo rozpočet

V této situaci by bylo potřeba zpracovat nové (nebo modifikovat stávající) programy pouze tam, kde se bude pracovat s nově zavedenými charakteristikami. U programů nepracujících s novými charakteristikami postačí prosté provedení rekompilace (bez nebezpečí zavlečení chyb v procesu "hlavoruční" modifikace zdrojového textu).

Popsaná "elementarizace" číselníků však bude účinná jen tehdy, budou-li použité elementární číselníky relativně stabilní.

3. EXTERNÍ DEFINICE STRUKTUR

Programovací jazyk COBOL poskytuje podporu pro aktualizaci zdrojových textů prostřednictvím příkazu COPY. Tento příkaz však není ani při využívání konstrukce REPLACING pro naše účely plně dostatečný, neboť :

- nedovoluje používání příkazu COPY ve vkopírovávaném textu a při užití REPLACING předpokládá konkrétní znalost nahra-

zovaného i nahrazujícího slova (nevhodné pro vytváření nebo modifikaci hierarchických datových struktur),

- neřeší problém překročení povolené délky řádku při nahrazování kratšího slova slovem delším (kompilátor nemusí zjistit chybu!),
- neumožňuje zjistit, které programy musíme po změně určitého textu v COPY-knihovně překompilovat a které nemusíme (značně problematické při větším počtu programů).

Byl proto vypracován speciální program podporující využívání METABÁZE (viz samostatná příručka "COPY - preprocesor, uživatelský manuál"), který :

- odstraňuje popsané nedostatky standardního zpracování příkazu COPY aparátem jazyka COBOL,
- průběžně aktualizuje multistrom vazeb, umožňující jak spolehlivou orientaci, tak i automatizaci v případě změn,
- otevírá možnost dalšího vývoje systému v souladu se zkušenostmi, postupně získávanými při využívání METABÁZE .

Doplnění nových objektů do stávajících datových struktur lze u jazyka COBOL (a podobně i u ostatních programovacích jazycích umožňujících explicitní deklaraci datových struktur) vyřešit, úpravou úseku zdrojového textu, popisujícího příslušnou datovou strukturu.

Nejspolehlivějším řešením je zásadně nepoužívat vlastní deklarace datových struktur, ale pouze odkazů do METABÁZE.

Deklarace v METABÁZI je nutno vytvářet tak, aby každá datová struktura byla definována pouze jedenkrát - složitější struktury stavebnicově skládat ze struktur už definovaných. Získáme tím záruku, že každá úprava se zaručeně promítne na všechna potřebná místa.

V případě změn v číselníku bývá většinou problém složitější. Je totiž zakořeněným zlozvykem , že přístupový klíč není používán pouze ke zpřístupnění údajů z číselníku, ale slouží také jako zdroj informací (např. rozpoznávání náhradních dílů

pouze čísla materiálu).

Takovýmto řešením pracovníkovi obeznámenému s konstrukcí klíče umožníme rychlejší orientaci a ušetříme v číselníku něco málo místa. Programátora však nutíme, aby psal programy zafixované na okamžitý tvar číselníku, tedy programy, které při změně konstrukce přístupového klíče budou působit potíže.

Možnost řešení je zde v zásadě dvojí :

- číselníky které nahrazujeme (nebo můžeme ovlivnit) řešíme tak, aby všechny informace byly obsaženy v těle věty číselníku ve formě samostatných položek a přístupový klíč aby sloužil výhradně pro dosažení těla věty,
- pro ostatní číselníky uložíme do METABÁZE navíc :
  — deklaraci pomocného pole, obsahujícího proměnné pro informace, které nejsou u číselníku přímo referencovatelné,
  — podprogram, obsazující toto pomocné pole hodnotami odvozenými z příslušné věty číselníku,
  a při každém přístupu do číselníku tento aparát použijeme.

Dodržíme-li tyto postupy, vystačíme při většině změn pouze se zásahem DO METABÁZE a nenarušíme-li její konsistenci, máme jistotu, že provedením rekompilace získáme správně fungující programy.

Dokonalé odolnosti programu proti změnám vstupních a výstupních souborů dosáhneme, použijeme-li v algoritmu větu vstupního/výstupního souboru (fyzickou) přímo, ale pracujeme s větou (logickou), kterou přečte/zapíše přístupový podprogram, uložený V METABÁZI.

Pro praktické používání je výhodné zvolit zásady pro tvorbu jmen, usnadňující orientaci v obsahu a struktuře METABÁZE.

4. ZÁSADY VYTVÁŘENÍ METABÁZE

Při vytváření a aktualizaci metabáze je nutno dodržovat tyto zásady :

- jména používaná v metabázi je nutno volit uváženě tak, aby později nevyvstala nutnost přejmenovávání používaných objektů metabáze (nedodržení této zásady může způsobit nepoužitelnost metabáze).
- každý zdrojový text vložit pouze do jednoho volaného nebo řídícího textu metabáze (nedodržení této zásady může ohrozit logickou konsistenci aktualizované metabáze).
- je nepřípustné měnit postavení proměnné v deklarované struktuře (skupina / element) nebo její typ (číslo / text) - nedodržení této zásady může mít za následek nesprávnou funkci, nebo dokonce havarování výsledných programů během výpočtu.
- je nutno dodržet zvolenou hierarchii položek (kvalifikaci), aby bylo možno využívat konstrukce c o r r e s p o n d i n g.
- při deklaraci datových struktur zahajovat popis každé struktury urovní "01" (je předpokladem pro vytváření hierarchických deklarací aparátem C O P Y - preprocesoru).
- potřebujeme-li redefinovat elementární položku nebo skupinu, neredefinujeme ji skupinou, ale zásadně elementární položkou. Nejlepší by bylo nechat délku redefinující položky vypočítávat aparátem C O P Y - preprocesoru. V takovém případě bychom místo opakovače ve vzoru redefinující položky zadávali znak " * " :

```
01  SIGMA
 02 BETA          PIC 99(4) .
 02 ALFA          PIC  9(8) .
01  C-SIGMA REDEFINES SIGMA PIC 9(*) .
```

nebo ještě raději :

```
01  SIGMA
 02   COPY BETA .
 02   COPY ALFA .
01  C-SIGMA REDEFINES SIGMA PIC 9(*) .
```

"neurčitý opakovač, který COPY-preprocesor nahradí příslušným číslem (zde 12) výpoč. z kontextu

5. MULTISTROM VAZEB

Rozsah reálné metabáze se bude pohybovat mezi několika stovkami až několika tisíci nejrůznějších textů.

Při využívání takového rozsahu informací si lze udržet potřebnou orientaci pouze za předpokladu vytvoření automatické evidence o metabázi.

Pro ukládání těchto údajů slouží speciální průběžně aktualizovaný soubor - " MULTISTROM VAZEB ", umožňující :

- kontrolovatelnost stavu metabáze před každým jejím použitím (viz 1. příznak),
- automatické promítnutí potřebných změn z aktualizovaných volaných textů do všech příslušných textů výsledných (= REKONSISTENACE),
- informace o stavu a struktuře metabáze potřebných pro její tvorbu, údržbu a využívání.

Následující schéma struktury multistromu vazeb používá symboly :

- ..... významná mezera před údajem (za údajem jsou nevýznamné)

CIS ..... trojmístné číslo určující pořadí průb. rekonsistenace

JMENO ... jméno souboru, dle potřeby včetně přípony (JJJJJJ.PPP)

ŘETĚZ ... libovolný text (délky udané v popisu přísluš.rubriky)

RRMMDD? . poslední dvojčíslí roku + měsíc + den + příznak :

1.příznak : "-" ... všechno je v pořádku
"?" ... nedefinovaný stav (po aktualizaci)
"1" ... řídící text má starší datum než volaný
"2" ... schází původ.řídící text (chyba "?")
"4" ... schází volaný text (chyba "#")
"8" ... smyčka (řídící text je volán zpětně)
numerické příznaky mohou být superponovány

2.příznak : "-" ... běžný text
"+" ... výsledný text
"?" ... chyba : hlavní text nebyl "data-less"

Výrazem "REKONSISTENACE" rozumíme spuštění aparátu, zajišťujícího vygenerování těch výsledných textů, které vznikají ze zaktualizovaných vstupních textů - tedy obnovení logické konsistence metabáze.

6. TYPY VĚT V MULTISTROMU

PŘÍSTUPOVÝ KLÍČ		
SKUPINOVÝ KLÍČ	POM	INFORMACE

IDENTIFIKACE MULTISTROMU:

	-	----RETEZ	⟵ * jméno multistromu(max.6 zn)
	-	---RRMMDD-	⟵ * datum aktualizace
	-	-RETEZ	⟵ * jméno aktualizátora(max.9zn)
	&	---RRMMDDP	⟵ * datum rekonsis. + 1.příznak
	&	-RETEZ	⟵ * jméno rekonsistenátora (max.9 zn)

INFORMACE PRO REKONSISTENACI:

CIS	-	JMENO
	-	. . .
	-	JMENO

seznam textů "CIS"- té úrovné rekonsistenace multistromu (úroven 000 mohou mít jen listy)

CHYBY "#" :

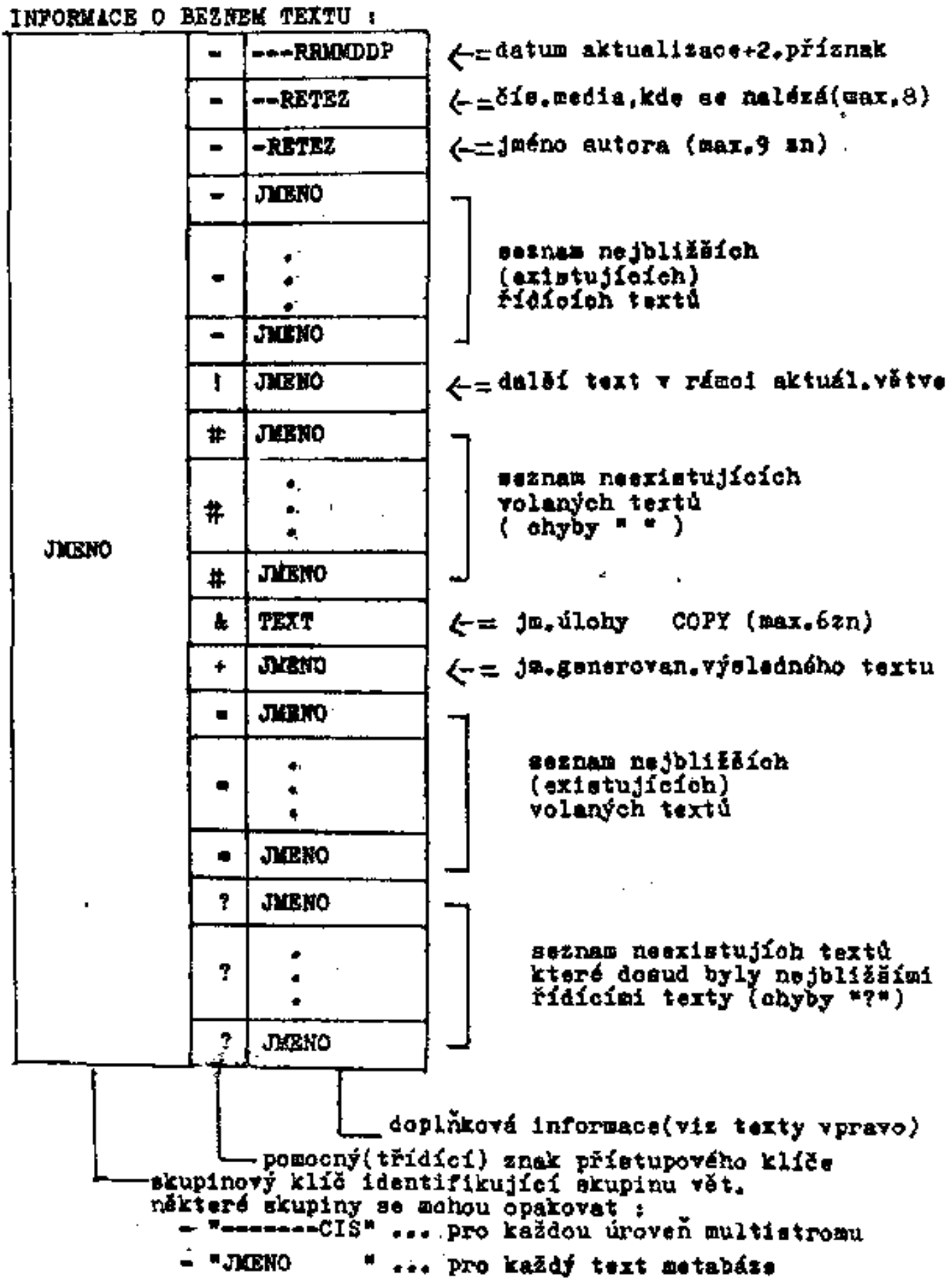
#########	-	JMENO
	-	. . .
	-	JMENO

seznam textů, obsahujících odkaz "COPY" na neexistující volaný text

CHYBY "?" :

?????????	-	JMENO
	-	. . .
	-	JMENO

seznam textů, ke kterým neexistuje původní nejbližší řídící text


INFORMACE O BEZNEM TEXTU :
JMENO
- ---RRMMDDP
<=datum aktualizace+2.příznak
- --RETEZ
<=čís.media,kde se nalézá(max.8)
- -RETEZ
<=jméno autora (max.9 zn)
- JMENO
- JMENO
seznam nejbližších (existujících) řídících textů
! JMENO
<=další text v rámci aktuál.větve
JMENO
JMENO
seznam neexistujících volaných textů (chyby " # ")
& TEXT
<= jm.úlohy COPY (max.6zn)
+ JMENO
<= jm.generovan.výsledného textu
= JMENO
= JMENO
seznam nejbližších (existujících) volaných textů
? JMENO
? JMENO
seznam neexistujících textů které dosud byly nejbližšími řídícími texty (chyby "?")
doplňková informace(viz texty vpravo)
pomocný(třídící) znak přístupového klíče
skupinový klíč identifikující skupinu vět.
některé skupiny se mohou opakovat :
- "-------CIS" ... pro každou úroveň multistromu
- "JMENO " ... pro každý text metabáze