Pokus o zjednodušený model CASE, sestavený s využitím systémového přístupu

Vlastimil Čevela

1. Úvod

Před časem jsem měl kontakt s firmou, dovážející velice renomovaný CASE. Nabízela za dočasně výhodně zlevněnou cenu 150 tisíc Kčs nástroj na popis systému. K tomu pak bylo možno v obdobně lidové ceně zakoupit i generátor programů (v Cobolu). Po dobu několika měsíců se mi však zatím nepodařilo získat odpověď na otázku, jak vlastně vypadá vygenerované programové vybavení a jaká je náročnost jeho provozování.

Nemám rád příliš komplikované věci a už vůbec ne takové, které za velké peníze představují zajíce v pytli. A tak jsem oprášil myšlenky pana Dijkstry a spol. o proslulých třech strukturách a neexistenci složitých úloh, pokud jsou vhodně dekomponovány a systémový přístup pana Langeforse ze Švédska.

Se značnou dávkou troufalosti jsem k tomu přihodil některé vlastní praktické zkušenosti s metodami popisu systému, počítačovou dokumentací a generováním programů – a pokusil se vytvořit alespoň částečně smysluplný model něčeho jako CASE.

Jsem si dobře vědom složitosti a náročnosti této problematiky, protože automatizace analyticko-programátorské práce je dlouhá léta předmětem mého více i méně úspěšného snažení. Domnívám se však, že k užitečné manipulaci s předměty není vždy nutný věžový jeřáb a nejnovější kamion zn. Mercedes – k tomu, aby se člověk zbytečně nenadřel totiž někdy stačí i jednoduchá páka a trakař.

Takže předkládám příspěvek, obsahující zjednodušenou metodiku popisu informačního systému, která umožňuje průběžně aktualizovat něco jako databázi s popisem systému (repository) a poskytovat přehled prvků a vazeb, případně adresář dat.

V dalším pak je předložen návrh základních funkcí částečně objektově orientovaného generátoru programových textů, spolu s popisem jazyka pro jeho ovládání.

Předem považuji za nutné zdůraznit, že se nejedná o žádné precizované návody, ale o nástin určité metodiky, která umožňuje velice volnou aplikaci, s plným využitím analyticko-programátorské invence.

Konkrétní zkušenosti s dílčím řešením uvedených námětů vycházejí z víceletých pokusů na počítačích Tesla 200, EC 1026, PC/XT–AT a používání metodiky projektování ASŘ,

která byla autorem vytvořena v r. 1988 na jeho tehdejším pracovišti ve VS Ingstav Brno pro bývalý resort stavebnictví ČSR.

Programovacím jazykem byl na všech vyjmenovaných počítačích osvědčený klasický COBOL. V posledních případech pod MS DOS se jedná o MS COBOL 4.0 a 4.5, jazykově shodný s MF COBOL/2.

2. Zavedení základních pojmů pro CASE

2.1 CASE = Computer Aided Software Engineering

V zásadě jde o programový a metodický systém, který podporuje softwarové (SW) inženýrství.

2.2 Činnosti v rámci SW inženýrství

Obecně je snad možno je vymezit jako:

- analýza a syntéza problému
- definice datových a výkonných prvků systému
- vyvoření programového vybavení

Fyzickým výsledkem softwarového inženýrství by měl být PROGRAMOVÝ SYSTÉM, který řeší požadavky zákazníka a DOKUMENTACE, která tento systém popisuje.

2.3 Dokumentace, prvotní a odvozené dokumentační prvky

Z hlediska tvůrců (a navazujících udržovatelů!!!) je možno dokumentaci definovat jako úplný a dostatečně podrobný popis celého řešení, zajišťující možnost kdykoliv do všech podrobností prostudovat jeho obsah a všechny uvažované funkce.

Jde tedy o souhrn písemných nebo jinak srozumitelně zaznamenaných svědectví, potvrzujících určitý stav a jeho změny.

Velice výhodné je tvořit tzv. AKTIVNÍ DOKUMENTAČNÍ PRVKY, které se kromě poskytování informací řešiteli, mohou po příslušném překladu přímo podílet na provádění nebo řízení počítačového zpracování, na automatické kontrole návaznosti i postupu vývojových prací a pod.

PRVOTNÍ DOKUMENTAČNÍ PRVKY jsou ty, které musí vzniknout tvůrčí činností řešitele. ODVOZENÁ DOKUMENTACE pak vzniká automaticky, jako výsledek činnosti CASE.

2.4 Systémový přístup

Základním principem je, že na určité úrovni dekompozice je celek definován jako množina dílčích částí (prvků), jsou specifikovány vnější vazby tohoto celku a stanoveny jeho základní vlastnosti. Pro každou dílčí část na úrovni větší podrobnosti pak lze postupovat obdobně.

Z hlediska účelu je možno jeden fyzicky existující systém zkoumat z různých pohledů a pro každý z nich může být skladba prvků, vazeb i úrovní jiná.

Otázka, zda systém řešit shora–dolů či zdola–nahoru je zbytečná – oba přístupy je třeba vhodně kombinovat.

2.5 Data a jejich popis, slovník a adresář dat

Elementární datové položky, datové skupiny, struktury, soubory, parametry, formuláře, obrazovky atd.

Účelem datových popisů je definování datových struktur a jejich vzájemných vztahů v rámci vyšších celků.

SLOVNÍK DAT obsahuje definice datových prvků, ADRESÁŘ DAT poskytuje informace o jejich výskytu v rámci datových struktur a vyšších celků.

2.6 Výkonné prvky a jejich popis, přehled prvků a vazeb

Programy, skupiny úloh, agendy a pod., ale i technické vybavení, nebo manuální pracoviště.

Základním nástrojem pro definování funkcí výkonných prvků je tzv. FUNKČNÍ POPIS, založený na systémovém přístupu.

Přehled prvků a vazeb popisuje strukturu systému a toky dat na zvolené úrovni podrobnosti.

2.7 Uspořádání prvků v rámci celku

Formu uspořádání prvků v rámci celku lze prakticky vždy převést do maximálně 3 základních struktur, kterými jsou POSTUPNÝ, ALTERNATIVNÍ a OPAKOVANÝ výskyt.

Výše uvedené pravidlo lze použít nejen pro sestavení algoritmu programu, ale i pro skládání výkonných prvků do vyšších celků a datových popisů.

Shodným principem se řídí i definování programovacích, příp. jiných jazyků, popisujících systém.

3. Prvotní dokumentace – vstupy pro CASE

V návaznosti na základní pojmy zavedené výše, budeme CASE chápat jako počítačový systém zpracování dat, který slouží řešitelům zákaznického programového vybavení.

Praktická realizace počítačového ukládání prvotní dokumentace, která představuje vstupy do tohoto systému, může mít různé interaktivní formy. Od textového editoru, přes nabídkový dialog s formuláři, až po kreslení a generování deklarací obrazovek či výstupních sestav.

Společnou podstatou však zůstává, že se stále jedná o informace, které nemohou vzniknout jinak, než tvůrčí činností řešitele.

V oblasti datové do této kategorie patří především slovník dat, obsahující definice elementárních datových položek, popisy datových struktur a návrhy uživatelského informačního rozhraní (obrazovky, sestavy). Různé formy datových popisů jsou běžně známy, takže zde dále nebudou rozebírány.

Jako základní a univerzální nástroj prvotní dokumentace pro definování funkcí výkonných prvků je navrhován FUNKČNÍ POPIS.

V souladu se systémovým přístupem umožňuje formálně velice jednoduchým způsobem popisovat celek s vnějšími vazbami i členěním do dílčích částí (procesů) a v navazujícím volném textovém popisu specifikovat jeho základní vlastnosti i případné další doplňující informace.

Podstatu pravidel jazyka funkčních popisů tvoří členění do dále specifikovaných bloků 1 až 5 s předepsaným řazením jednotlivých údajů ve formalizovaných řádcích bloků 1 až 3 a využíváním tzv. dekompozičních kódů I, O, W v bloku 2.

funkční popis

```
1
   {textový soubor popisovaného celku}          k  JMENO-C     [pozn]
      [komentářový řádek] ...
2
   I {textový název datového vstupu}            k  JMENO-D     [pozn]
      [komentářový řádek] ...
   ...
   W {textový název aktualizovaných dat}        k  JMENO-D     [pozn]
      [komentářový řádek] ...
   ...
   O {textový název datového výstupu}           k  JMENO-D     [pozn]
      [komentářový řádek] ...
   ...
3
   {textový název dílčího procesu}              k  JMENO-P     [pozn]
4
   Volným textem popis základních vlastností celku
5
   Volným textem specifikace podmínek ovládání a dalších
```

Formalizované bloky:

1 = identifikace popisovaného celku

2 = vstup/výstupní charakteristika (vnější datové vazby)

3 = dílčí procesy v popisovaném celku

Neformalizované bloky:

4 = základní funkce (cíle a vlastnosti) popisovaného celku

5 = doplňující podmínky činnosti popisovaného celku

k = znak pro označení kategorie celku a prvků

JMENO–C,D,P = formalizovaná jména celku a prvků

Poznámky k používání funkčních popisů

Funkční popisy umožňují definovat formalizovaně i volně všechny základní vlastnosti výkonných prvků na všech úrovních, tj. od systémů či subsystémů jako velkých celků, až po programy a podprogramy, anebo i jednotlivé dílčí algoritmy.

Při vhodně navržených kategoriích se proto mohou stát základní formou prvotní dokumentace, ze kterých lze jednoznačně algoritmicky odvodit strukturu systému.

4. Odvozená dokumentace – výstupy z CASE

V souladu s výše uvedenými požadavky na dokumentaci by měl systém CASE průběžně poskytovat veškeré informace o řešeném programovém systémy ve všech fázích jeho životního cyklu.

Základní podmínkou ke splnění tohoto cíle je tedy především komplexnost. Prvotní dokumentace proto musí být integrována do jednoho celku, tvořícího prvotní bázi dat s informacemi o systému (repository).

V našem modelu CASE se omezíme na nejjednodušší, ale velice užitečné odvozené výstupy – adresář dat a přehled prvků a vazeb.

ADRESÁŘ DAT je souhrnem výskytů jednotlivých datových položek v rámci datových struktur systému nebo jeho částí. V návaznosti na přehled vazeb datových souborů pak poskytuje hlavní podklad pro vysledování toků dat.

PŘEHLED PRVKU a VAZEB lze odvodit z formalizovaných bloků ve funkčních popisech. Příslušná transformace zahrnuje výběr jednotlivých řádků v blocích 1 až 3, dosazení formalizovaných jmen výskytu spolu s čísly bloků a setřídění dle jmen a kategorií.

přehled prvků a vazeb

k	jméno prvku	b	i/o	textový název	výskyt	pozn
k	JMENO–D1	2	I	text ...	JMENO–C1	xxx
k	JMENO–D1	2	W	text ...	JMENO–C2	
...						
k	JMENO–P1	1		text ...	JMENO–P1	
k	JMENO–P1	3		text ...	JMENO–C2	xxx
... atd						

Je zřejmé, že problém je daleko složitější a jeho komplexní řešení výrazně náročnější. Funkční popisy však představují jednoduchý prostředek na formalizované zachycení popisu skutečně jakéhokoliv výkonného prvku, takže v rámci přehledu prvků a vazeb lze vysledovat opravdu jakékoliv vztahy.

V úvodu bylo řečeno, že se jedná především o náměty k volné aplikaci a rozvíjení invence. Nebudeme tedy zacházet do dalších podrobností.

Závěrem je však třeba dodat, že i když je forma funkčních popisů v podstatě triviální, jejich sestavení není vůbec snadnou záležitostí. Většinou totiž vyžadují velice důkladnou znalost řešené problematiky a ujasnění si všech souvislostí i v řadě běžně přehlížených detailů.

Pro předkládaný model CASE platí stejná pravidla, jako pro každý běžný systém počítačového zpracování dat. Kvalita informací, které systém poskytuje, bude v každém okamžiku záviset především na kvalitě a úplnosti prvotních vstupů.

5. Návrh objektově orientovaného generátoru programových textů

V dalších odstavcích je formulována představa o základních funkcích, které by měl řešit částečně objektově orientovaný generátor zdrojových programových textů. Předpokládá se práce především v jazyku COBOL, ale většina principů je použitelná i obecně.

Celé řízení tvorby definic typových modulů i jejich aplikačního využívání je navrhováno tak, že může být využíváno i pro tvorbu dávkových souborů .BAT, vyplňování formulářů a pod. Rámcovým cílem je získat nástroj, který by umožnil navázat na výše definovaný popis systému a generovat určité části programového vybavení.

Pro ovládání generátoru platí základní výchozí podmínka, aby se co nejvíce vycházelo z běžné syntaxe standardního programátorského jazyka a potřebné řídící a doplňkové kódy se omezily na skutečně nezbytné minimum.

5.1 Skládání výsledných programových textů

Generování programových textů je založeno na skládání výsledných textů zdrojových programů, řízené textem konkrétního aplikačního modulu z předem připravené stavebnice typových modulů.

Typové moduly mohou být definovány a ukládány na více úrovních, takže mohou sloužit k využívání v celém rozsáhlém systému, anebo pouze v jednom programu.

Výsledné zdrojové programové texty jsou pak překládány normálním kompilátorem příslušného jazyka.

5.2 Bloková struktura programu

Každý aplikační i typový modul může obsahovat nejen procedurální programový text, ale i definice datových položek a odkazy na jiné moduly.

Každý modul může obsahovat systémové odkazy a komentáře, definice, které mohou být vícekrát použity a pásmo vlastního programu. Ten lze libovolně rozdělit do bloků, podle kterých pak bude skládán výsledný text.

příklad blokové struktury

definice základního typového modulu:

```
1.
< TYP–Z
ws 1
     working–storage section.
pd 2
     procedure division.
>
```

vstupní text:	výsledný text:
1.	
a ——— modul a —	working–storage section.
ws	1 A pic xx.
1 A pic xx.	1 B pic xx.
1 B pic xx.	1 C pic 99.
pd	1 D pic 99.
move A to B.	procedure division.
	move A to B.
b ——— modul b —	add C to D.
ws	
1 C pic 99.	
1 D pic 99.	
pd	
add C to D.	

Uživatelská jména mohou být volitelně buď s globální anebo pouze lokální platností, tj. jednoznačně přiřazená k danému použití příslušného modulu.

5.3 Parametrizace typových modulů

Texty typových modulů mohou obsahovat formální parametry (proměnné), které při jejich pozdějším využívání umožňují doplnit, změnit nebo vypustit předem definované úseky.

Parametricky je také řízena tvorba jednoznačných lokálních uživatelských jmen.

Parametrizace typových modulů, ale i definic přímo v aplikačním modulu, dává možnost rozsáhlé aplikační modifikovatelnosti.

```
┌──────────────────────────────────────< příklad volných deklarací >──┐
| vstupni text:                                                       |
├──── 1.──── 8.───────────────────────────────────────────────────────┤
|     [       1    TAB1.                                              |
|              aaa    123    'text-1 a b c''                          |
|              bbb    345    'text-2 e    f'                          |
|                                                                     |
|     [x      1    TAB2.                                              |
|     *   typ        kod   řízení                                     |
|         A043B1     C0    C5  AC        *> varianta A                |
|         A044B2     31    C8  B2        *> varianta B                |
|                                                                     |
|     [8      1    TAB3.                                              |
|     ....*....1....*....2....*....3....*....4....* atd               |
|     libovolně obsazený řádek                   až do konce          |
|     ]                                                               |
├─────────────────────────────────────────────────────────────────────┤
| výstupni text:                                                      |
├─────────────────────────────────────────────────────────────────────┤
|             1    TAB1.                                              |
|              2    pic x(18) value "aaa123text-1 a b c".             |
|              2    pic x(18) value "bbb345text-2 e    f".            |
|                                                                     |
|             1    TAB2.                                              |
|              2    pic x(6) value x"A043B1C0C5AC".                   |
|              2    pic x(6) value x"A044B231C8B2".                   |
|                                                                     |
|             1    TAB3.                                              |
|              2    pic x(40) value "....*....1. atd..".              |
|              2    pic x(40) value "....*....5. atd..".              |
|              2    pic x(40) value "libovolně    atd..".             |
|              2    pic x(40) value "až do konce atd..".              |
└─────────────────────────────────────────────────────────────────────┘
```

5.4 Volné deklarace v jazyku COBOL

Generátor dává programátorovi možnost výrazně jednoduššího a přehlednějšího zápisu konstant, příp. tabulek formou tzv. VOLNÉ DEKLARACE.

V úsecích programu, vymezených speciálními znaky je totiž možno veškeré znakové, číselné, textové i hexadecimální řetězce psát s libovolným počtem oddělovacích mezer zcela volně. Generátor řetězce spojí, spočte délky a vytvoří formálně správné deklarace „pic x(délka) value '...'. ", případně „pic x(délka) value x'...'. ".

Obdobně pak lze deklarovat i plný 80-ti nebo 120-ti sloupcový řádek pro obrazovky nebo sestavy.

5.5 Ovládání generátoru

Celé řízení tvorby definic typových modulů i jejich aplikačního využívání je realizováno prostřednictvím speciálních řídících znaků na prvních dvou pozicích běžného 80-ti znakového textového řádku.

Formální parametry (proměnné) v typových modulech a v definicích jsou označovány speciálním znakem a číslicí, obdobně jako v souborech .BAT pod operačním systémem MS DOS na PC.

Poznámky k zápisu formátu vstupního textu pro generátor

Jsou navrhovány částečně upravené zvyklosti, běžně používané pro jazyk Cobol:

a) postupný výskyt = posloupnost slov a řádků, odkazy na podrobnější formát jsou v uvozovkách

b) alternativní výskyt začíná kombinací '/ mezera', nepovinný výskyt je v hranatých závorkách

c) opakovaný výskyt = 3 tečky za nebo pod, svorkové závorky { } vymezují opakovanou skupinu

Ostatní znakové kombinace na začátcích řádků v popisu formátu reprezentují řídící znaky pro 1. a 2. sloupec:

; přiřazení knihoven modulů

: odkaz na základní typový modul

* komentářový řádek

/* začátek komentářového bloku

*/ konec komentářového bloku

<< začátek definice slovníku dat

< začátek definice programového modulu

< formát-1 >

```
vstupní text pro generátor

    pásmo
        / systémové odkazy a komentáře

            / řádek odkazu na knihovny modulů
               ;  LIB-1  [ LIB-2  [ LIB-3 ] ]
            / řádek odkazu na základní typový modul
               :  JMENO-T
            / komentářový řádek
               *  jakýkoliv text
            / komentářový blok s absolutní přednostÍ
               /*
                řádek  ...
               */
        / definice

            / definice slovníku dat
               <<  JMENO-S
                   JMENO-D  "definice datové položky"
                   ...
               ...
            / definice programového modulu
               <  JMENO-P
                  [ "řádek záhlaví bloku" ]
                     pásmo "program"
                        dle potřeby obsahuje "f-parametry"
                     ...
                  ...
               ...
            >
        / program

            [ "řádek záhlaví bloku" ]
               skupina běžných řádků
                  / volná deklarace konstant
                    (znaky [ ] zde nejsou jako závorky)
                     [z   1  JMENO-V.
                        "řádek volných deklarací"
                        ...
                     ...
                     ]
                  / odkaz na programový modul
                     .  JMENO-P  [ "parametry" ]
                        [ "parametry" ]
                        ...
                  / odkaz na datovou položku
                     ,  U  < JMENO-D[ KOD ] > ...
                     ...
                  / ostatní
                     programový řádek
                     ...
               ...
            ...
    ...
```

> konec poslední definice v řadě

[z začátek volných deklarací

] konec poslední volné deklarace v řadě

. odkaz na programový modul

, odkaz na datovou položku

Literatura

[1] Sborníky ze seminářů Programování v letech 1975 až 1991 DT ČSVTS Ostrava

[2] Čevela V.: Počítačová tvorba dokumentace projektu zpracování dat s využitím prostředků jazyka Cobol TESLA 200 DT ČSVTS Pardubice, 1979

[3] Čevela V.: Metodika projektování ASŘ (doporučené zásady) ÚRS Praha pro resort stavebnictví ČSR, 1988

[4] Popis funkce generátoru programových textů 3.x (interní dok.) Čevela – Programátorské služby Cobol, 1992

[5] Langefors B.: Teoretická analýza informačných systémov, Alfa Bratislava, 1981

Autor: Ing. Vlastimil Čevela
Benešova 279
664 42 Modřice
tel. 05 – 303 367, fax 05 – 322 504