ČTYŘI ÚROVNĚ ABSTRAKCE PŘI KONCIPOVÁNÍ MODELŮ A PROCESŮ

Ing. Andrej Boldiš, FMF Praha

1. Úvod

"Inženýrství je modelování a počítání", tak nazval svou vysokoškolskou učebnici americký profesor Gajda /1/. Přeloženo do jazyka programátorů, inženýrství je navrhování datových modelů a výpočetních procesů neboli prostě zpracování dat.

Model (resp. datový model) je zjednodušený popis jistého výseku reálného světa, jistých entit, jevů nebo zařízení. Účelem např. popisných modelů je pomoci při analýze a pochopení modelovaného výseku reálného světa. Účelem preskriptivních modelů je zase předpovídat (predikovat) nebo duplikovat chování reálného systému. Úkolem inženýra při modelování je objevit a definovat na modelovaném objektu systém, t. j. definovat zjednodušující pohled ze stanoviska jisté teoretické nebo aplikované disciplíny. Kromě o zmíněných deskriptivních a preskriptivních modelech lze mluvit též o modelech statických a dynamických, deterministických a pravděpodobnostních, analogových a symbolických. Podrobně a soustavně o problematice modelování pojednává např. kniha Prof. J. Vlčka /2/.

V běžných počítačových aplikacích zvláštní důležitosti nabyly modely, které bychom mohli označit jako deterministické, diskrétní a výčtové (enumerative). Příspěvek je věnován právě těmto modelům.

Pod procesy budeme rozumět nejenom počítačové procesy resp. zpracování dat a jejich konkrétní formy jako jsou úlohy (tasks, jobs), programy, procedury, funkce a operace, ale i abstraktnější nepočítačové procesy jako jsou algoritmy, kalkuly a inferenční (deduktivní, odvozovací) procesy.

Abstrakce je pojem, kterého bychom se neměli bát a obcházet ho, protože určitou schopností abstrahovat se vyznačuje každý. Abstrahovat znamená v podstatě pouze nepřihlédnout k něčemu, pomíjet nebo odmyslet něco.

Problematika abstrakcí v programování byla v minulých létech centrálním tématem v teorii programování a programovacích jazyků, viz např. /3/, /4/. Klíčovým problémem při návrhu velkých softwareových systémů je totiž redukce komplexnosti, t. j. snížení počtu detailů, se kterými je nutné počítat a vypořádat se v každém okamžiku. Této redukce lze docílit právě prostřednictvím abstrakcí. Dobrá abstrakce je taková abstrakce, která zdůrazňuje významné detaily a nao-

pak potlačuje bezvýznamné detaily. Abstrakci v programování lze ještě výstižně charakterizovat jako maximální lokalizaci objektů a procedur. Tato lokalizace objektů a procedur se uplatnila v procedurálních abstrakcích a zejména v konceptu funkce. Dalším významným výsledkem v teorii programování je koncepce datových abstrakcí, jehož uplatnění co do rozsahu zatím nelze považovat za ukončené.

Kromě uvedených abstrakcí se v počítačových aplikacích uplatňují další vyšší abstrakce, v nichž např. myslíme, definujeme modely a procesy a zadáváme je k programové realizaci ve formě projektů informačních a řídících systémů.

2. Úrovně abstrakcí

V běžných soudobých počítačových aplikacích je vhodné rozpoznávat čtyři významné úrovně abstrakcí. V následující tabulce jsou uvedeny jisté základní pojmy a kategorie, uspořádané do sloupců podle těchto úrovní abstrakce. V řádkách tabulky se nalézají zase pojmy "přibližně" sémanticky ekvivalentní. Označení úrovní abstrakcí je v záhlaví tabulky. V tabulce je 5 sloupců, což nesmí mást, intensionální a extensionální abstrakce jsou totiž sémanticky ekvivalentní a žádná z nich nepředstavuje vyšší abstrakci té druhé.

Problématika naznačených úrovní abstrakce je tak rozsáhlá a obtížná, že referát si nečiní nárok být ani úplným ani detailním a lecos z referátu je zajisté diskutabilní a napadnutelné. Ideje, prezentované v referátě, navazují na několik pozoruhodných výsledků z počítačové vědy.

Mezi prvními, kteří považovali za vhodné rozpoznávat čtyři úrovně abstrakce v datovém modelování, byl Chen ve svém článku /5/ o entitně-relačním modelu. Rozdíl mezi tímto referátem a Chenovým pojednáním je hlavně v používané terminologii nikoliv v obsahu. Chen dále neuvažuje o intensionální úrovni, což neochuzuje jeho koncepci, uvažuje pouze o datových modelech a nikoliv o procesech.

Kowalski v článku /6/, který se zase zabývá speciálně procesy, navrhuje, aby se v algoritmech rozlišovaly dvě složky: logika a řízení. Jinými slovy, ohlédneme-li od řídící složky procedur, dostáváme kalkul, vyjadřující konstruktivní podstatu algoritmu.

Hlavní tézí logického programování /7/ je, že predikátovou logiku je možné chápat jako programovací jazyk velmi vysoké úrovně. Konkrétněji, výroky z atomárních predikátů se mají chápat jako elementární fakta neboli data; jisté výroky z predikátů (neatomárních) složených pomocí propozicionálních spojek,

mohou mít zase procedurální interpretaci.

Na uvedené tři zdroje navazuje tento referát. Pojmy a termíny používané v referátu speciálně v tabulce odpovídají jednak tradiční matematické logice a jednak aktuálnímu slovníku procedurálního programování, viz např. /8/ až /11/.

V existenciálních pojmech myslíme a mluvíme, resp. definujeme na modelovaném objektu systém, náš mentální model výseku reálného světa.

V pojmech konstruktivně-konceptuální úrovně hledáme už principiální možnosti přejít na soustavu (modelujících) manipulovatelných objektů (např. materiálních znaků, datových hodnot a objektů) a na soustavu triviálních konstruktivních operací, umožňujících konstruovat nové objekty při využití objektů už zkonstruovaných. Při hledání těchto možností neklademe si ještě žádná omezení na prostor a čas, resp. na množství paměťových prostředků a na dobu potřebnou pro výpočet. V pojmech této konstruktivní konceptuální úrovně je formulována řada teoretických datových modelů, třem z nich je věnován jeden z následujících odstavců.

Zatímco v pojmech konceptuálních úrovní formulujeme pouhé koncepty datových modelů a procesů, v pojmech programovacích jazyků se popisují beze zbytků tyto datové modely a algoritmy tak, že tyto lze svěřit k provedení strojům. Vyšším programovacím jazykům a hlavně tomu, čím se tyto liší od nižších jazyků, je věnován rovněž jeden z následujících odstavců.

Obecně lze říci, že rozhraní mezi úrovněmi abstrakcí v některých případech je více a v jiných méně ostré a zřetelné. Zdá se, že nejméně zformovány jsou úrovně konceptuálně-konstruktivní a vyšší programovací a že právě tyto úrovně představují onu sématickou mezeru mezi existenciálním způsobem myšlení a praktickým programováním, o které je občasná zmínka v literatuře.

Pro porovnání pojmů v jednotlivých řádkách tabulky bude vhodné si uvědomit jistou závažnou skutečnost. Abstrakce je vlastně pomíjení jistých konkrétních detailů, proto abstraktnější pojem může být sémanticky méně bohatý než jeho konkrétnější protějšek. Lze tušit rovněž, že i každý abstraktnější formalizovaný jazyk bude nejspíš méně sémanticky bohatý než jazyk nižší. Toto zúžování významu a obsahu pojmů lze ukázat a vidět i na několika málo příkladech.

K tomu, aby neuspořádaný výčet skalárních objektů měl význam množiny elementárních prvků, musí splňovat jistou jednoduchou podmínku korektnosti, a sice jednoznačnost prvků ve výčtu. V opačném případě výčet nemůže mít význam množiny. Ve skutečnosti neuspořádaný výčet je pojem sémanticky bohatší, než množina, relace a zobrazení.

Existují procedury a funkce, jejichž význam a účel nepřesahuje implementační úroveň. Takovou operací je např. přetřiďování souborů (SORT). Protože ale souboru odpovídá konceptuálně neuspořádaný výčet, t. j. výčet, u kterého nezáleží na pořadí prvků výčtu, jakákoliv změna pořadí prvků výčtu nemění hodnotu objektu typu soubor. Přesto třídění má v programování veliký význam pro efektivní implementaci algoritmů, souvisí však pouze s řídicí složkou algoritmů.

O vztahu kalkulu a procedury již byla zmínka. Jako příklad si uvedeme známý kalkul pro generování slov nad určitou abecedou: Je-li Z libovolná konečná neprázdná množina, pak Z^+ bude označovat množinu všech konečných neprázdných slov sestrojených z prvků množiny Z takto:

a) je-li $z \in Z$, pak $z \in Z^+$,

b) je-li $w \in Z^+$, $z \in Z$, pak $wz \in Z^+$,

c) do Z^+ patří pouze slova vzniklá konečně mnoha aplikacemi pravidel a), b).

Kalkul je skutečně pouze jisté konstruktivní schema algoritmu. Chybí v něm ta složka, kterou je určeno pořadí aplikace pravidel a kterou by byla zajištěna podmínka korektnosti výčtu, tak aby tento výčet mohl být interpretován jako množina slov nad abecedou. Procedury naproti tomu implementují algoritmy komplexně, přesněji, zajišťují i korektnost a úplnost algoritmu, proto pojem procedury je sémanticky bohatší než kalkul.

Při pohledu zdola nahoru dochází tedy k sémantickému zužování. Jak je to ale obráceně, při pohledu zhora dolů? Zdá se, že problém by se měl principiálně stavět takto: mají-li abstraktnější koncepty svoje opodstatnění, pak by se měly promítat i do bezprostředně nižších úrovní; každý abstraktnější koncept by měl mít svůj konkrétnější sémantický ekvivalent. Zatím tomu tak není a mezi nejvyššími abstrakcemi a praktickým programováním zejí místy značné sémantické mezery.

3. Datové, procedurální, funkční a řídicí abstrakce

Vyšší programovací jazyky lze stručně a výstižně označit a charakterizovat jako soustavu standardních datových, procedurálních, funkčních a řídicích abstrakcí. Uvedené čtyři druhy abstrakcí představují současně čtyři cesty, kterými se ubíral a ubírá proces zdokonalování programovacích jazyků.

Problematika typů neboli datových typů je v poslední době oprávněně předmětem velikého zájmu viz např. /3/, /4/. V novějším pojetí /8/ "typ je charakterizován jistou množinou hodnot a jistou množinou operací". V tomto smyslu kategorie typu je konkretizace jistých teoretických principů známých jako datové abstrakce nebo též jako abstraktní datové typy. Podle těchto principů podstatu určitého typu hodnot vystihují právě operace (funkce) použitelné na da-

ný typ hodnot.

Abstraktní datový typ znamená ovšem i něco mnohem konkrétnějšího. Datové objekty určitého typu musí být úplně charakterizovány soustavou operací tvořící daný typ, to znamená, že tyto operace smí být jedinými přímými prostředky pro vytváření a manipulování s datovými objekty. V takovémto pojetí abstraktní datový typ byl vlastně aplikován už v prvních vyšších jazycích u numerických datových objektů. V dalším vývoji vyšších jazyků představuje významný mezník koncepce skalárního výčtového typu. Od této doby se ví, že reprezentace (volba kodů) dat nepatří do vyššího jazyka.

V novějších vyšších jazycích, např. v Adě, je v popředí zájmu tzv. "podporování" datových abstrakcí, t. j. umožňování uživatelům jazyka definovat si vlastní, privátní, (nestandardní) datové abstrakce. Podstatným prvkem tohoto podporování je tzv. zatajování, skrývání informace (information hiding), což lze též označit jako lokalizace těch datových a procedurálních detailů, od kterých se právě abstrahuje.

Takové příkazy vyšších jazyků jako jsou if-then-else a různé druhy cyklů jsou označovány velmi správně jako (standardní) řídící abstrakce. Jiné příkazy a funkce vyššího jazyka představují zase standardní procedurální a funkční abstrakce. Zvláště důležité při zdokonalování programovacích jazyků jsou funkční abstrakce. Ve funkčních abstrakcích je totiž silně potlačována řídící složka procedur, což se projevuje v možnosti skládání elementárních funkcí do výrazů (složených funkcí). Touto možností funkce přispívají podstatnou měrou k řádovému snižování procedurálnosti programů. Algolské procedury a funkce představují obdobně jako privátní datové abstrakce možnost tvorby privátních (nestandardních) procedurálních a funkčních abstrakcí.

Proces zvyšování úrovně abstraktnosti programovacích jazyků není ještě zdaleka ukončen. Zdá se, že značné rezervy a možnosti zůstávají zatím neprobádané a nevyužité. Jedná se zejména o datové objekty a typy vyšších úrovní jako jsou např. výčty resp. soubory a seznamy. Možnost tvorby privátních datových, procedurálních a funkčních abstrakcí nemůže plně nahradit logicky konzistentní soustavu příkazů a funkcí pro práci s výčty, sémanticky minimálně tak bohatou jako je predikátová logika.

4. Relační, entitně-relační a síťový datový model

Zde není dost místa podrobně popisovat uvedení teoretické datové modely a rozdíly mezi nimi. Síťový datový model je ale dobře známý ve světě praktiků,

relační (R) model je zase velmi populární v akademicky orientovaných kruzích. Paradoxní je, že entitně-relační (ER) model, který je "sémanticky nejbohatší" a "nejpřirozenější" je nejméně známý a populární.

Základní pojmy ER modelu jsou : entita, vztah mezi entitami a atribut entity resp. vztahu. Terminologie ER modelu se liší od tradiční matematické terminologie spíše formálně a je v jistém smyslu i výhodnější. Entitě odpovídá (elementární) prvek (množiny). Vztahu entit odpovídá n-tice (elementárních) prvků ve vztahu. N-tici "entita (resp. vztah) a její atributy" odpovídá soustava uspořádaných dvojic zobrazení, a sice prvním prvkem dvojice (vzorem) je vždycky entita resp. vztah a druhým prvkem (obrazem) je jeden z atributů. V ER modelu se důsledně rozlišují relace a funkce (zobrazení). N-tice entit (prvků) ve vztahu je správně považována za prvek vyššího typu, jenž může mít své atributy, a relace může být definičním oborem zobrazení.

R model se obešel bez pojmu zobrazení (funkce) a zná pouze n-tici prvků ve vztahu a relaci. Takže řada atributů v ER modelu jsou podle R modelu prvky ve vztahu ! Takováto koncepce způsobuje značnou sémantickou vágnost a jistou "nepřirozenost" modelu. Podle novějších interpretací R modelu konvertuje tento k ER modelu. Zásluhy R modelu ale tkví spíše v koncepci tzv. relační algebry, jazyka pro práci s relacemi (soubory). Tento jazyk je sice sémanticky chudý a nepoužitelný pro profesionální programování, přesto ho lze hodnotit nejenom jako ukázku jazyka pro práci s objekty vyššího typu, ale i pro docenění principů datových abstrakcí (i když se o těchto v R modelu explicitně nemluví).

Terminologie síťových modelů by se formálně od ER modelu ani nelišila. Horší ale je, že vztah mezi entitami je chápán a implementován značně primitivně. Vztah v síťovém modelu není chápán jako objekt vyššího typu, jemuž by na konstruktivních úrovních měl odpovídat materiální objekt a který by mohl mít též své atributy. Tento vztah je chápán jako něco zcela pomyslného a implementován jako adresové odkazy mezi větami (records), modelující entity a jejich atributy. Vztah v síťovém modelu je tedy redukován na přístup (access) k objektům. Takováto koncepce vztahu neumožňuje hlavně přiřazovat vztahům atributy a tato potřeba se potom složitě obchází. S touto koncepcí těsně souvisí obdobně neudržitelná koncepce tzv. "navigace" v databázi.

Podstatné rozdíly mezi uvedenými datovými modely lze tedy shrnout na jejich vztah k základním matematickým konceptům: množina, relace a zobrazení (funkce). Tyto poslední jsou nejlépe promítnuty v ER modelu. Základní koncepty ER modelu mají evidentní sémantické ekvivalence v naivní (intuitivní) teorii množin, teorii grafů a predikátové logice, proto ER model je sémanticky nejbohatší a nejpřirozenější.

5. Závěr

My všichni jsme účastníky překotného vývoje řady aplikovaných disciplín spjatých s uplatňováním počítačů v praxi. Vedle nesporných úspěchů a vynikajících praktických výsledků, nelze si nevšimnout teoretického chaosu. V současné době nelze bezpečně říci, co by mělo tvořit nadstavbu programování a jak teoreticky připravovat projektanty a programátory. Kdo zase na vlastní pěst podniká exkurzi do světa teorií programování a počítačové vědy, nejspíše podlehne depresi a nejkratší cestou se vrací k praxi a intuici. Neskromným cílem referátu bylo pomoci čtenáři mírně se zorientovat v složité problematice nadstavby praktického programování.

6. Literatura

/1/ Gajda, W. J., Biles, W. E.: Engineering: Modeling and Computation. Houghton Mifflin Comp., Boston

/2/ Vlček, J.: Metody systémového inženýrství. SNTL, Praha, 1984.

/3/ Liskov, B. et al.: Abstraction Mechanisms in CLU. Comm. ACM, Vol. 20, No. 8, August 1977.

/4/ Shaw, M.: The Impact of Abstraction Concerns on Modern Programming Languages. Proc. of the IEEE, Vol. 68, No. 9, September 1980.

/5/ Chen, P.P.: The Entity - Relationship Model - Toward a Unified View of Data. ACM TODS, Vol. 1, No. 1, March 1976.

/6/ Kowalski, R.: Algorithm = Logic + Control. Comm. of the ACM, Vol. 22, No. 7, July 1979.

/7/ Clocksin, W. F., Mellish, C. S.: Programming in Prolog. Springer - Verlag, Berlin, 1981.

/8/ Reference Manual for the Ada Programming Language. US DOD, July 1980.

/9/ Klini, S. K.: Matematičeskaja logika (překl. z angl.), Mir, Moskva, 1973.

/10/ Halmos, P. R.: Naive Set Theory. D. van Nostrand Comp. Inc., New York, 1966.

/11/ Boldiš, A.: Minimální slovník programování. Sborník semináře Programování 84, Ostrava.

Tabulka úrovní abstrakce

konceptuální úrovně			implementační (realizační, datové a programovací) úrovně	
existenciální úrovně		konstruktivní (intultivní) úrovně		
intensionální	extensionální	konstruktivní koncep-tuální úroveň	vyšší programovací úroveň	úroveň fyzikální a strojové
Jazyk predikátové logiky	Jazyk teorie množin	Jazyky kalkulů	Vyšší programovací jazyky	Nižší programovací jazyky
Predikát, P(x)	Univerzální množina	Univerzální výčet	datový typ	reprezentace (kodování) dat
Výrok typu P(a)	prvek, $a \in M$	skalární objekt	skalární hodnota a datový objekt	paměťové místo obsahu-jící jednotku dat
Výrok typu P(a,b)	prvky ve vztahu	uspořádaná n-tice skalárních objektů	hodnota a datový objekt typu uspoř. n-tice nebo dvojice	datový objekt typu record; obyčejně fyzikální posloupnost paměťových míst
	prvky zobrazení	uspořádaná dvojice		
	prvky více zobraz. (jeden vzor, více obrazů)	soustava uspořáda-ných dvojic	datový objekt typu n-tice (nejčastější případ v praxi)	
	(prvek pologrupy nebo monoidu)	uspořádaný výčet (např. skalárních objektů, n-tic, atd)	hodnota a datový objekt typu seznam (list)	
Výrok typu $\forall x P(x)$	množina(prvků) $M=\{x : P(x)\}$	neuspořádaný výčet skalárních objektů	hodnota a datový objekt typu soubor (file)	posloupnosti objektů typu record na různých mediích (operační pa-měť, disk, páska,...), vhodně organizovány za účelem výhodného pří-stupu k větám, (SAM, ISAM, databázové sys-témy)
Výrok typu $\forall x \forall y P(x,y)$	relace $R=\{(x,y) : P(x,y)\}$	neuspořádaný výčet uspořádaných n-tic		
	(pologrupa, monoid)	neuspořádaný výčet uspořádaných výčtů		
	zobrazení množiny do množiny	neuspořádaný výčet uspořádaných dvojic		

Tabulka úrovní abstrakce (pokračování)

Výrok +..,	množina (nebo relace) a více zobrazení	neuspořádaný výčet soustavy usp. dvojic	nejčastější případ souboru v praxi	dtto výše
$\forall x\ (P(x) \& Q(x))$ $\forall x\ (P(x) \vee Q(x))$ $\forall x\ (P(x) \& \neg Q(x))$	$P \cap Q = \{x : P(x) \& Q(x)\}$ $P \cup Q = \{x : P(x) \vee Q(x)\}$ $P \setminus Q = \{x : P(x) \& \neg Q(x)\}$			
Výroky typu : $\forall x(P(x) \supset Q(x))$ $\forall x \forall y(P(x) \& Q(y) \supset R(x,y))$	(algebraické systémy)	kalkuly, produkční pravidla	souborové příkazy a funkce (standardní proc. a funkční abstrakce, např. relační algebra	procedury využívající operace WRITE (PUT) a READ (GET), a pod.