Použití systémů CASE

Pavel Drbal

Do naší republiky vtrhly satelitní antény, tržní hospodářství a píp děvčata (peep girls). Nyní také systémy CASE. Vždy jsem měl slabost pro složitá technická zařízení a chápu, že mít doma video, na kterém si ráno naprogramuji, které pořady a z kterého satelitu mají být nahrány, je velmi fascinující. Co však nemohu pochopit, je to, kdy má ten dotyčný čas se na nahrané pořady dívat. Navíc za podmínek tržního (nebo skoro tržního) hospodářství. Myslím si, že je jen jedno logické vysvětlení, ti majitelé satelitních antén jsou programátoři, za které pracuje jejich CASE. Reklamní materiály systémů CASE uvádějí, že snižují pracnost o 90%. Takže stačí být tak tři čtvrtě hodinky v práci a zbytek dne se mohu koukat na video.

Nejprve trochu statistiky. Systémů, které si říkají CASE, je ke dvěma stovkám (v roce 1990 jich bylo 167). Většina se zaměřuje na jeden či dva programovací jazyky, systém, který spolupracuje se sedmi jazyky, je jen jeden. Nejpoužívanějším jazykem je Cobol, pak následuje C a Ada ("nejpoužívanější" se týká počtu systémů CASE spolupracujících s programovacím jazykem, ne počtu vytvořených produktů). Každý systém CASE, který jsem potkal, spolupracoval s několika databázovými systémy. Nejpoužívanější hardware je PC a WAX, kde pod PC se rozumí AT nebo PS2, pod WAX se rozumí workstation. Vzhledem ke značnému používání grafiky je nutná grafická karta s vyšší rozlišovací schopností, myš a často ploter.

Hned na počátku chci uvést, že systémy CASE jsou tak rozsáhlé, že stěží existuje v naší republice dostatečně stabilní firma, která může věnovat dostatek kapacity na jeho vytvoření. Vzhledem ke kurzu koruny se ceny pohybují kolem půl milionu na jednu instalaci. Možnost úpravy ceny na naši úroveň je sice v rámci různých programů na podporu málo rozvinutých zemí možná, avšak těžko predikovatelná.

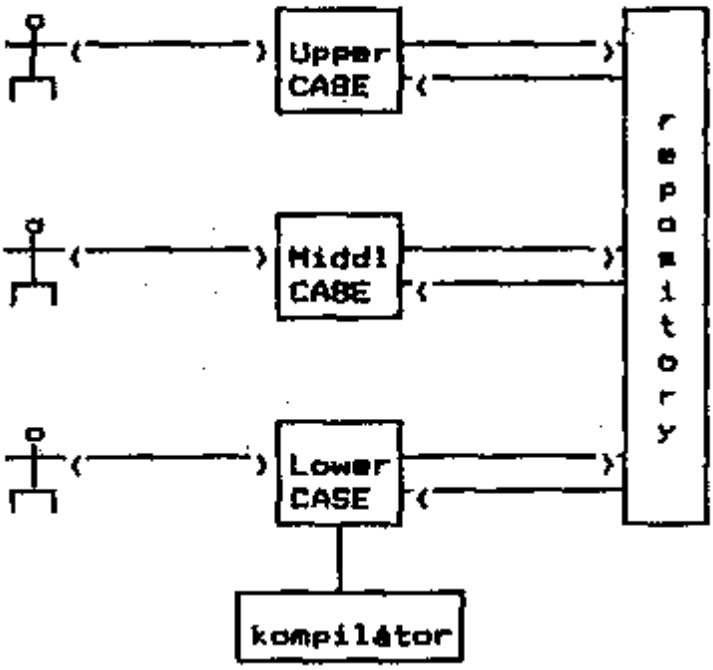
Systémy, které si říkají CASE, jsou různých úrovní a používají různé přístupy. Liší se rozsahem, jak velikostí samotného systému, tak i množstvím nabízených funkcí. Jedno z používaných dělení je na Upper CASE, Middle CASE a Lower CASE.

Lower CASE je nějaký efektivní způsob zápisu programu, například grafický.

Middle CASE zahrnuje prostředky pro návrh programového systému jako celku, patří tam funkční návrh systému (Data Flow Diagramy), datové modely (E/R model) a podobně.

Upper CASE je jakýsi kouzelný prostředek, pomocí kterého získám od budoucích uživatelů požadavky na systém a přetvořím je v zadání systému.

Graficky to lze znázornit asi takto:



Repository je společná databáze (systémová encyklopedie), která umožňuje, že si jednotlivé subsystémy mohou předávat data.

Některé firmy organizují své CASE systémy tímto způsobem, dokonce i tak, že každý subsystém má svou vlastní lokální repository. Po ukončení práce jedním subsystémem jsou relevantní data poslána dalšímu subsystému (způsob export-import).

Takové schema nejvíce odpovídá představě programátora. Matematici sní o tom, že uživatel - odborník popíše požadavky na požadovaný systém a na matematické teorii založený generativní prostředek vygeneruje program, který už není třeba ani ladit. Ovšem je také důležité, jestli tyto představy programátorů a matematiků odpovídají optimálnímu způsobu návrhu programů. Většina systémů CASE má trochu jiný přístup, který je rozebrán dále.

V zásadě lze říci, že na malé úkoly se používají malé prostředky a na velké úkoly velké prostředky. Z tohoto hlediska lze systémy CASE rozdělit na Malé CASE a na Velké CASE.

Malé CASE

Malé CASE (někdy se jim poněkud vznešeněji říká také Lower CASE) jsou určeny pro řešení relativně malých úloh (například administrativa malého podniku nebo jednotlivé administrativní subsystémy). Jsou dvou druhů, buď datově nebo programově orientovány.

Programově orientované systémy jsou zaměřeny na zvýšení efektivnosti čistého programování. Snaží se umožnit rychlou orientaci v programu (většinou grafickými prostředky), těsně svázat běh programu s jeho zdrojovým zápisem (ladění ve zdrojové reprezentaci) a poskytovat podporu (většinou ve formě procedur) pro často používané obraty. Příkladem takových systémů jsou například systémy turbo, RTK, X-TOOLS. Sem lze také zařadit některé speciálně orientované knihovny programovacích jazyků, například VitaminC.

Datově orientované systémy většinou tvoří nadstavbu nad některou databází. Jejich cílem je zajistit aktualizaci položek databáze a jejich využití (zviditelnění) a přitom co nejméně programovat. Řeší se to tak, že se popisným způsoben zadají obrazovky (panely). Při popisu panelu se zadávají stálé texty (literály), vstupní pole (kam se zobrazují informace z databáze) a výstupní pole (kam se zadávají nové hodnoty). Samozřejmě jsou možná pole, která jsou současně vstupní i výstupní (tím se realizuje aktualizace). K popisu databáze patří popis uložených záznamů a jejich položek. Porovnáním těchto dvou popisů, pole v panelu a uloženého záznamu, lze automaticky vytvořit transformační rutinu, a charakteristika pole na panelu (jestli je vstupní nebo výstupní) určuje směr transformace. Podobným způsobem lze popsat i tiskové sestavy. Při takovém způsobu práce se vlastní programování omezuje na řešení nestandardních požadavků a není příliš náročné.

4GL

Zkratkou 4GL se označují neprocedurální jazyky pro zpracování hromadných dat (Business Oriented 4. Generation Language). Princip nahražení programování popisem panelů je u nich doveden do důsledků. Jsou určeny pro tvorbu interakčních programů pro práci s daty. Jsou to do důsledků domyšlené datově orientované systémy, kde je programování skoro úplně potlačeno. Zahrnují 150 až 250 funkcí, které se při zpracování dat používají. Která funkce se provede a kdy se provede se určuje neprocedurálně, dá se říci tabulkově. Struktura programu je určena sítí menu (panely s menu). Jednotlivé funkce se přidělují buď celému panelu nebo jednotlivým polím na panelu zobrazeným. Zobrazovat a aktualizovat se dají data z databáze nebo i proměnné systému. Programování se omezuje na malé rutiny, které

lze "přivěsit" k datům tak, že se provádějí buď před jejich zobrazením nebo před uložením.

Jazyky typu 4GL umožňují také prototyping (simulaci ještě nevytvořeného systému), a to tím, že lze vytvořit síť obrazovek bez napojení na databázi. Takový prototyp je možno předvést uživateli a nechat si odsouhlasit způsob komunikace.

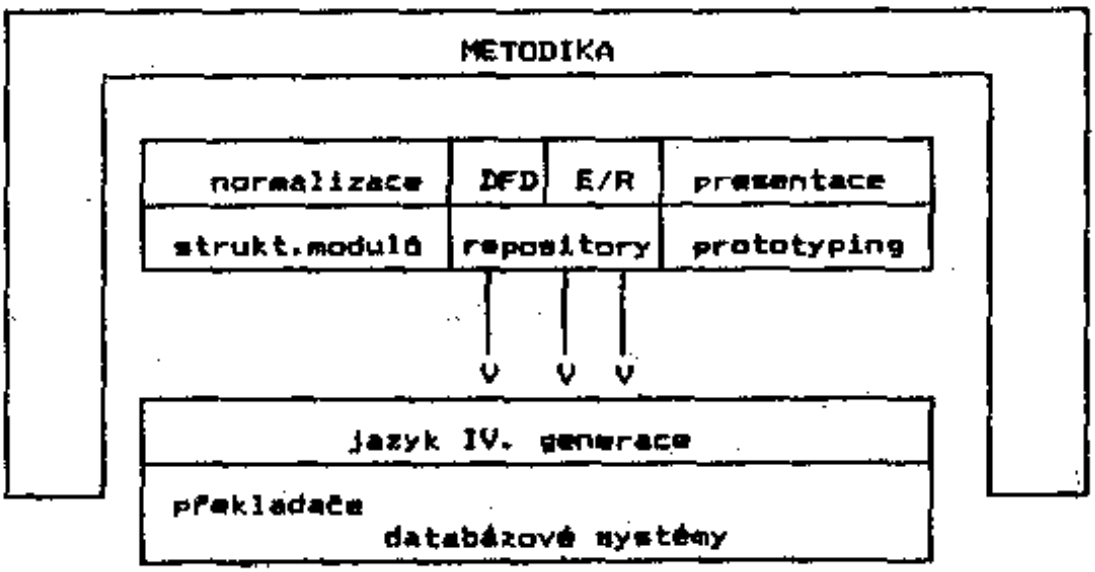
Jazyky 4GL lze samostatně vytvářet programy nebo lze do jazyka 4GL vygenerovat program z prostředků vyšší úrovně.

Velké CASE

Hlavní myšlenka systémů CASE vychází ze skutečnosti, že programování (obzvláště procedurální programování) je jen závěrečnou etapou tvorby programu. Zkušenost říká, že chybu v příkazech programovacího jazyka lze odstranit lehce, ale chybu v počátečních etapách návrhu programového systému lze odstranit jen s velkými náklady a mnohdy ani napravit nejde.

Hlavní pozornost se tedy musí soustředit na logický návrh vytvářeného systému. To se realizuje tak, že se vytvoří **model** navrhovaného systému a ten se zkoumá z různých stran tak dlouho, až vyhovuje všem požadovaným kritériím.

Srdcem systému CASE je **repository** (také se říká systémová encyklopedie), ve které je uložen model vytvářeného systému. Repository je vlastně speciální databanka. Na repository se úzce pojí prostředky pro analýzu zadání úlohy a formování modelu (schéma je uvedeno na následujícím obrázku). Každý takový prostředek má vlastní pohled na model (většinou dost jednostranný) a vlastní metodiku. Právě střídáním různých pohledů se model postupně upravuje až na vyhovující stav. Ilustrujme si to zhruba.



Začneme třeba funkční analýzou. Prostředkem k ní je DFD (Data Flow Diagram). Tam popíšeme okolí vytvářeného systému, jeho jednotlivé funkce a datové toky mezi nimi. Tím jsme získali jakýsi popis zpracovávaných dat. Tato předpokládaná data zpracujeme pomocí datového modelu (prostředek E/R). Určíme podrobně vztahy mezi typy dat (entitami) a pečlivě je popíšeme (atributy). Tento datový model převedeme do normální formy (tj. zbavíme data redundancí - prostředek normalizace). Tyto úpravy nám umožní upřesnit funkční model (DFD). Pak můžeme simulovat styk uživatele se systémem (prostředek prototyping). Při prototypování používáme popisu dat z vytvořeného datového modelu. Tu se může ukázat, že okolí požaduje data, která jsme nedefinovali, anebo námi definovaná data nejsou použita. Změněné požadavky se zase promítnou do datového a funkčního modelu. Model funkcí můžeme také použít k členění vytvářeného programu na moduly atd. Vidíme, že jednotlivé pohledy jsou svázány prostřednictvím repository a že změna v jednom pohledu se adekvátním způsobem promítne do jiných pohledů.

Informace o vytvářeném programovém systému uložené v repository lze použít pro tvorbu programového systému. Tyto generace jsou tří typů:

- generuje se schéma databáze (z datového modelu),
- generuje se program v jazyce typu 4GL,
- generuje se program v klasickém programovacím jazyce (např. Cobolu).

Ne všechny typy generace jsou vždy možné, je však pravidlem, že CASE může generovat schéma několika běžných databází.

Vytvoření programového systému (tj. výše zmíněná generace) není příliš nákladná, takže vytvořený systém lze prověřovat a jestliže je neuspokojivý, vrátit se k úpravě modelu v repository. Takto lze provádět prototyping na vyšší úrovni.

Celý tento postup je řízen metodikou. Ta nemá zvláštní programovou podporu, jen textové editory (hypertextový procesor). Metodika obsahuje návod jak postupovat (know how). Obsahuje například teorii datového modelování, návod na normalizaci datového modelu, principy funkční analýzy, pravidla pro určnení modulové struktury z transakční struktury ap.

Vše, co jsme dosud dělali intuitivně (a mnohdy nedůsledně) je v metodice formulováno, explicitně vyjádřeno, rozděleno do etap a jsou určeny cíle, které musí být jednotlivými etapami naplněny. Velké projekty nejde zvládnout intuicí a chaoticky používanou inteligencí.

Závěr

Metodika poskytuje možnost vyhnout se slepým uličkám ve tvorbě programových systémů s minimálními náklady (je lépe 10 produktů pečlivě kontrolovat než vytvořit jeden nepoužitelný).

CASE poskytuje dobrý prostředek pro boj proti důsledkům fluktuace (výsledky své práce si neodnese pracovník ve své hlavě, zůstanou v repository).

Zůstávají zachyceny všechny etapy vývoje systému a lze se k nim vrátit. Repository a protokoly z kontrolních etap poskytují úplnou dokumentaci vytvářeného systému - a to s minimální spotřebou papíru.

Nejedná se o levnou záležitost. Poměr cen

4GL : repository s prostředky : metodika

je například 1:10:45, cena úplného prostředku se pohybuje mezi jedním a třemi miliony Kčs (za současného kurzu), a to za jednu instalaci!

Trendy

Uvádí se, že náklady na vývoj nového programového vybavení jsou 20% a náklady na údržbu starého jsou 80%. V současné době se rozvíjí "reinženýring", tvorba modelu z existujícího programového vybavení. To umožňuje zefektivnit údržbu již hotového software.

Informace použité v tomto článku byly získány z materiálů a rozhovorů s pracovníky těchto firem a koncernů: Software Consult, Index Technology Corporation, Systemhaus GEI, Siemens, Pandata, Technosoft, Software, IBM, Computer Equipment, Kancelářské stroje, AiD, McDonnell Douglas.

Autor: RNDr. Pavel D r b a l, CSc.
VÚMS a.s.
Lužná 2
161 31 Praha 6
tel. 02-362022