Vývoj rozsáhlých systémů s pomocí nástrojů CASE

Jiří Felbáb

Tento příspěvek se zabývá vývojem informačních systémů za pomoci prostředků pro podporu softwarového inženýrství - CASE.

Nejprve je stručně popsáno prostředí a postupy, v rámci kterých se naše zkušenosti pohybují. Těžiště příspěvku by mělo být v popisu zkušeností, získaných během téměř dvouletého používání prostředku Westmount Open I-CASE Yourdon for Informix, s generátory kódu INFORMIX-Forms i s generátory kódu implementovanými vlastními silami. Na závěr příspěvku je pak uvedeno pár všeobecných poznámek, týkající se popisované problematiky.

Obecný popis možností Westmount I-CASE Yourdon lze najít ve sborníku konference CSUUG z roku 1992 nebo v časopise Silicon World 3/93. Popis Yourdonovy metody i demonstrační příklad byly součástí předloňského sborníku semináře Programování '92. Pokus o vystižení principů Yourdonovy metody obsahuje také článek v časopisu printf 1/94, který je zde na konferenci k dispozici.

Zdá se, že využívání integrovaných CASE prostředků, založených na využívaní standardních metod osvědčených dlouholetým používáním na papíře, v etapě analýzy a návrhu, je dostatečně prověřeno a popsáno. Příspěvek bude proto zaměřen spíše na úroveň implementační a na některé konkrétní problémy, spojené s generováním kódu. Je to jednak oblast, která nabízí řadu různých variantních postupů, není tak jednoznačně zmapovaná, je komplikovanější v tom, že na ni přímo navazují různá implementační prostředí, a je to současně i oblast, do které potenciální uživatel vkládá nejvíce nadějí, protože očekává, že z něho sejme břemeno kódování.

1. Východiska

Firma Softwarové Aplikace & Systémy je jak distributorem firmy Westmount pro Českou i Slovenskou republiku, tak i uživatelem programového vybavení Westmount Open I-CASE Yourdon for Informix. Má to své důvody.

Firma SA&S se zabývá vývojem aplikací zejména z kategorie ekonomických aplikací a nemocničních systémů, provozovaných v prostředí operačního systému UNIX a relačního databázového systému Informix. Sami jsme hledali prostředek pro podporu vývoje aplikací této třídy. S ohledem na cílové prostředí jsme zvolili Westmount I-CASE. Jako distributoři databázového systému Informix jsme hledali vhodný CASE prostředek, který bychom mohli doporučovat svým zákazníkům. Firma Informix nemá vlastní CASE prostředek, ale prostředky ve své třídě nejlepší zařazuje do svého programu OpenCASE, kam patří i Westmount I-CASE. Westmount I-CASE používá i sama firma INFORMIX.

Současná pozice distributora a uživatele má několik aspektů. Těžko lze poskytovat podporu zákazníkovi bez toho, že ten, kdo podporu poskytuje, nástroj sám používá, protože znalost prostředku lze získat jen v každodenní praxi. Skutečnost, že nástroj sami používáme, je i jistým doporučením v tom, že se nesnažíme prodat zákazníkovi výrobek, kterému sami nedůvěřujeme. Určitou výhodou je i možnost získat pohled na popisovaný prostředek z obou stran, protože distributor obvykle tíhne k tomu své zboží pouze vychvalovat, zatímco uživatel zase naopak většinou vidí hlavně nedostatky daného nástroje.

Situace řešitele připomíná občas pohyb ve spirále a čekání na zázrak. Při programování informačních systémů v asembleru se vysvobozením zdál být Basic, při návrhu klasických informačních systémů využívajících souborovou strukturu to byl databázový systém s generátory formulářů a reportů, pak rozšíření těchto nástrojů o vnoření do jazyka C, poté nový jazyk čtvrté generace 4GL a nyní prostředky CASE. Očekávání zázračného řešení je obvykle následováno prohlédnutím případně zavržením zázračného nástroje. O těchto aspektech bylo ostatně dost řečeno ve sborníku semináře Programování '92.

Samo používání CASE prostředků není ještě zárukou funkční aplikace. Zvládnutí techniky a nástrojů, které používají, určitý čas trvá, přestože jsme řadu používaných postupů byli zvyklí využívat i bez podpory počítačového prostředí.

Vytvoření aplikace v prostředí CASE představuje vytvoření modelu navrhovaného systému, na který pak navazuje jeho implementace. Model navrhovaného systému se obvykle skládá ze tří pohledů - z modelu datového, modelu funkčního a z modelu časově závislého chování systému. Toto členění může být podrobnější (komunikační model, konstrukční model apod.), podle povahy aplikace může být kladen větší důraz na jednotlivé pohledy, podle použité metody mohou být použity různé termíny k označení jednotlivých modelů (např. objektový, dynamický a funkční model v objektově orientované metodě OMT), nicméně tyto tři základní pohledy jsou výchozí a bez ohledu na různé konkrétní techniky a nástroje jednotlivých postupů s nimi pracují všechny metody, které stojí v pozadí CASE prostředků. Vyvážení a konzistence těcho dílčích pohledů je základem úspěšné implementace výsledné aplikace. Ne všechny prostředky označované jako CASE ovšem obsahují všechny tři pohledy - některé z nich se omezují např. jen na na datový model.

Druhý pohled na prostředky CASE představuje pohled životního cyklu aplikace, který se skládá, opět podle použité metody, z řady etap, minimálně z analýzy, návrhu, implementace a provozování programového vybavení. Podle toho, zda prostředek podporuje všechny etapy životního cyklu, se pak hovoří o integrovaném CASE (integrated CASE), vyšším CASE (upper CASE) pro analýzu a návrh a nižším CASE (lower CASE) pro implementaci.

Vlastní techniky a nástroje řešení bývají podporovány ještě nástroji pro řízení projektu a prostředky pro automatizovanou tvorbu dokumentace, případně prostředky pro zpětný převod schématu databáze či zdrojového kódu do grafické podoby.

CASE prostředky firmy Westmount podporují několik standardních metod vývoje aplikací - Yourdonovu metodu, objektově orientovanou metodu OMT (Rumbaugh), SSADM a Ward-Mellor, z nichž první tři jsou určeny pro vývoj informačních systémů a poslední pro technické systémy a systémy reálného času. Prostředky jsou určeny pro vývojové a cílové prostředí relačních databází (Informix, Ingres, Oracle, Sybase) nebo jazyka C či C++. Systém, který používáme, je označen Westmount I-CASE Yourdon for Informix a jak vyplývá z názvu, je určen pro vývoj informačních systémů s pomocí Yourdonovy metody v prostředí databázového systému Informix.

V následujícím textu budou postupně popsány některé problémy, na které jsme narazili při řešení aplikací dosud řešených pomocí CASE. CASE postupně používáme při vývoji všech projektů, k dílčí nebo celkové podpoře řešení. Některé projekty jsou vyvíjené v prostředí CASE od samého začátku, u jiných jsme využili CASE až v průběhu vývoje a u některých zavedených aplikací jsme využili některé možnosti CASE, tam kde se nám to zdálo výhodné (datový model, dílčí dokumentace), zejména proto, že předpokládáme další rozvoj příslušné aplikace.

O požadavcích na zvládnutí metody a jednotlivých technik již byla zmínka. Z hlediska konfigurování a správy systému předpokládá jeho používání poměrně hlubokou znalost prostředí používaného typu operačního systému Unix a příslušného databázového systému, a to včetně síťové vrstvy jak databáze, tak operačního systému včetně architektury klient-server, na které je systém vybudován. Prakticky jsme vyzkoušeli a provozujeme distribuovanou instalaci cílové databáze a archivu (repository, encyklopedie), síťové tiskárny i grafických X - serverů, a to nejen v rámci lokální sítě LAN, ale i rozlehlejších metropolitních sítí (MAN). Jako pracoviště je případně možné využívat i PC pracující pod Windows s emulací X-terminálu. Systém vyžaduje znalost architektury i praktické zkušenosti s využíváním grafického prostředí, jak standardu X Window System (OSF, MIT), ale případně i OpenWindows (Sun), protože umí podporovat oba přístupy včetně obou manažerů OpenLook a Motif a kombinovat je. Zde přímo navazují praktické problémy, spojené například s využíváním české sady znaků ať už v samotném prostředí oken, nebo na úrovni vlastního CASE prostředku. Další samostatnou oblast, jejíž podporu systém využívá, představují systémy podporující tvorbu dokumentace. Westmount podporuje více cílových prostředí (FrameMaker, Interleaf). Máme zkušenosti s DTP FrameMaker, který ovšem povahou své lokální instalace nezapadá do jinak všeobecné architektury klient-server. Na úrovni implementace je potřebná znalost jazyka použitého pro implementaci (C a 4GL) a jazyka tcl, používaného pro psaní šablon a generátorů.

2. Datový model

Problémy související s tvorbou datového modelu jsou jednak problémy obecné provahy, které nejsou bezprostředně spojené s použitými prostředky CASE, a jednak problémy specifické pro použitý nástroj. Základním problémem na této obecné úrovni je správné vytvoření datového modelu například pomocí ER diagramu jako je tomu v případě Westmount I-CASE či libovolnou jinou technikou (viz např. Extended Relational Analysis, modelovací technika navržená firmou Relational Systems Corporation a používaná firmou Informix - viz INFORMIX Guide to SQL, Tutorial, Version 4.10, July 1991, str. 8-4 a další) Problémy řešené na obecné úrovni jsou problémy typu: jedná se o atribut či entitu?, existuje mezi atributy entity takový atribut, který lze vybrat jako primární klíč, tedy bude unikátně identifikovat každou instanci typu entity, nebude se měnit v čase a navíc bude přijatelný i z hlediska implementace?, jakým způsobem se vyrovnat s historií - pomocí časových razítek nebo rozdělením dat do více tabulek?, jaký zvolit kompromis mezi čistotou modelu z pohledu teorie a jeho praktickou implementací? apod. Zde asi nejvíce záleží na povaze aplikace a zkušenostech řešitele. V této fázi, která typicky spadá do fáze analýzy a globálního návrhu, nabízí nástroj I-CASE podporu v podobě uživatelsky příjemných grafických editorů, které umožňují jednoduše vytvořit a zejména snadno modifikovat výchozí diagram. Editor současně kontroluje i základní syntaxi vytvářeného modelu a nedovolí nakreslit syntakticky chybný diagram (např. přímo spojit dvě entity bez symbolu vazby, či definovat primární klíč pro vazební tabulku apod.). Další úroveň kontroly nabízejí explicitní kontroly, volané z nabídek, které hlouběji prověřují správnost navrženého modelu (např. entita nemá mezi svými atributy primární klíč). Pokud existuje datový slovník (Jacksonovy diagramy nebo Backus-Naurova forma, je možné zkontrolovat konzistenci navrženého ERD proti tomuto slovníku.

Problémy konkrétní jsou specifické pro použití daného nástroje. Z úrovně problémů spadajících do první kategorie se mohou zdát banální, ale často dokáží řešiteli zkomplikovat život. Z hlediska strukturování projektu je třeba definovat, které datové objekty budou mít globální povahu a budou definovány na úrovni systému a které budou lokální v subsystémech. Řečeno v termínech implementační fáze: budou všechny subsystémy projektu sdílet jednu společnou databázi nebo bude každý subsystém mít svou vlastní databázi? Odpověď těžko bude jednoznačná - nějaká globální data, společná všem subsystémům existují vždycky. Z hlediska přístupu programů k datům bývá jednodušší mít jednu databázi, ovšem z hlediska spolupráce řešitelů jednotlivých subsystémů na jednom projektu je nepříjemné, když změny datového modelu, ke kterým během vývoje nevyhnutelně dochází, vedou při novém generování schématu databáze ke změnám, promítajícím se i do těch subsystémů, které se už s datovým pohledem vyrovnaly a pohled na data aplikace se v nich ustálil. Z hlediska administrace, zálohování a případných havárií dat bude zase výhodnější mít více menších databází.

Samostatnou kapitolu představuje volba identifikátorů jednotlivých datových objektů. První pohled na volbu jména představují doporučení, která stanoví teorie - jméno entity budiž podstatné jméno v jednotném tvaru, vyjadřující roli entity v systému, atd. Druhý pohled představuje použité implemetační prostředí (např. délka identifikátoru jména sloupce je maximálně 18 znaků, musí začínat písmenem atd...) a dále potřeba snadno se orientovat ve výpisu datového slovníku či ve výpisu datových objektů, které jsou v uloženy encyklopedii (repository, archiv). Obsahuje-li datový model 30 entit a každá má v průměru 10 atributů, je to 5 stránek výpisu či 20 obrazovek textu. Snažili jsme se vytvářet identifikátory tak, aby v sobě obsahovaly jak identifikaci objektu, ke kterému patří, tak vlastnost, kterou popisují. Zvolená strategie se pak promítne do výpisu datového slovníku - buď budou seskupeny pohromadě všechny vlastnosti jednoho objektu nebo stejné vlastnosti různých objektů.

Z pohledu nejjednoduššího ER diagramu představuje entita budoucí tabulku a atributy entity sloupce této tabulky. Každý atribut entity má v diagramu své jméno, které se použije na úrovni modelu. Vyjdeme-li z toho, že datový model má sloužit pro komunikaci řešitele s budoucím uživatelem, pak by jména atributů měla být co nejvíce mnemotechnická, aby se dalo snadno odhadnout, co daný atribut vyjadřuje. Samostatný problém představuje otázka češtiny, protože v tomto kontextu by mělo být případně možné používat češtinu. Uvnitř encyklopedie projektové databáze jsou z uživatelského pohledu jednotlivé objekty jednoznačně identifikovány svým jménem. Použijí-li se pro dva atributy dvou různých entit stejná jména, bude se v encyklopedii jednat o jednu datovou položku, se všemi vlastnostmi společnými: s jedním datovým typem, omezením a interním a externím jménem. Jakákoli změna provedená u jednoho objektu se automaticky promítne i do všech ostatních objektů stejného jména. Týká se to samozřejmě objektů srovnatelných, spadajících do stejné kategorie, například atributů ER diagramů a listových uzlů diagramu datových struktur, nikoliv už například datových toků. Tento přístup se podobá koncepci "jmenných prostorů" (name space) v klasických programovacích jazycích, umožňující používat stejné identifikátory pro objekty spadající do různých jmenných prostorů (např. jméno proměnné, návěští a složky struktury v jazyku C).

Kromě názvu atributu jako objektu v rámci ER diagramu je možné standardně spojit s atributem ještě další dvě jména. První představuje interní jméno budoucího sloupce tabulky, který použije generátor v příkazech SQL CREATE TABLE. Druhé představuje externí, popisné jméno sloupce, které se využívá v chybových hlášeních nebo jako textový řetězec popisující ve formuláři korespondující obrazovkové pole. V prvním nepřipadá čeština v úvahu, ve druhém ano. Pokud neuvedete interní jméno atributu, použije systém jako jméno sloupce název atributu a v takových případech by nebylo možné použít češtinu ani ve jménech atributů.

Aby bylo možné vygenerovat z datového modelu schéma databáze, je třeba přidělit každému atributu datový typ. Kromě implementační úrovně se všude pracuje s logickými datovými typy, které se až při přenosu do implementace nahradí fyzickými datovými typy. Výhoda tohoto přístupu je zřejmá. Jestliže se rozšíří například IČO z osmi míst na deset, pak stačí na jednom místě změnit logický datový typ nazvaný například Ico_firmy z CHAR(8) na CHAR(10) a nechat znovu vygenerovat schéma databáze a změna se automaticky promítne do všech tabulek, které obsahují sloupce definované s tímto datovým typem. I u poměrně rozsáhlých projektů s řádově stovkami datových položek nepřekračuje množství logických datových typů únosnou mez, zhruba 30% z objemu datových položek. Záleží samozřejmě na podrobnosti členění (logický datový typ může být jen "cena" nebo "cena nákupní", "cena prodejní","cena výrobní" apod. Mapování logických datových typů na fyzické definují tři soubory, logické typy, standardní typy a fyzické typy. Každý projekt hospodaří se svými vlastními, lokálními logickými datovými typy. Pouhým zkopírováním uvedených tří souborů z adresáře jednoho projektu do adresáře druhého se však logické datové typy jednoduše importují. Tyto lokální soubory je možné přímo editovat a modifikovat tak škálu datových typů, aniž je třeba tyto změny dělat přes nabídky, což by u většího počtu položek bylo zdlouhavé.

Ke generování scénářů SQL se použije vytvořený ER diagram, šablona scénáře příslušné operace DDL (CREATE, DROP) a generátor kódu SQL. Cílové prostředí ve kterém u nás vývoj aplikací probíhá, představuje operační systém UNIX a databázový systém Informix s databázovým strojem INFORMIX-OnLine verze 5.0 a vyšší, dovolující definovat entitní integritní omezení a integritní omezení atributů (CHECK), refereční integritu a primární a cizí klíče (klauzule REFERENCES, PRIMARY KEY, FOREIGN KEY) a uložené procedury (STORED PROCEDURES), tedy rutiny, vykonávané nikoliv na popředí na úrovni aplikačního programu, ale na pozadí databázovým strojem - serverem. Standardní generátor a dvě dodávané šablony slouží k vytvoření a zrušení schématu databáze. Definice schématu databáze obsahuje příkazy pro vytvoření tabulek (CREATE TABLE) a indexů (CREATE INDEX), definici integritních omezení (constraints), pokud byla v datovém modelu uvedena (CHECK), definici referenční integrity realizovanou příkazem ALTER TABLE a zahrnující strategii použitou pro zpracování referenční integrity (např. kaskádní DELETE) a dále definici uložených procedur pro základní databázové operace, které mají povahu manipulace s daty - INSERT, UPDATE a DELETE (CREATE PROCEDURE). Tento systém sám o sobě je velmi propracovaný a umožňuje prakticky okamžitě generovat rozsáhlé scénáře SQL, řádově stovky kilobytů zdrojového textu (pro středně velkou aplikaci se definice schématu databáze pohybuje v rozsahu 200 - 300 Kb zdrojového textu.

Vzhledem k tomu, že jsme generovaný kód SQL chtěli rozšířit ještě o další klauzule, upravili jsme standardně dodávaný generátor SQL ještě o možnost jejich generování. Snadné přizpůsobení systému požadavkům koncového uživatele, jeho otevřenost a flexibilita představují jedny z jeho nejpříjemnějších vlastností. Funkcionalitu je možné upravovat buď na úrovni šablony nebo ve zdrojovém kódu jazyka tcl generátoru SQL. Generátor kódu se skládá ze dvou základních složek. První modul vytvoří interní paměťový tvar datového modelu. Tato část má tvar vykonavatelného programu přeloženého pro konkrétní hardwarovou platformu a její funkcionalitu není možné ani nutné měnit. Druhou část představuje interpretace interního paměťového modelu a generování příslušného scénáře SQL (nebo programu v jazyku 4GL či specifikačního souboru formuláře). Tato druhá část je napsaná v jazyku tcl (Tool Command Language, UCB) a protože jazyk tcl pracuje jako interpret, je dodávána ve zdrojovém tvaru jako scénář tcl a funkcionalitu je možné měnit.

Generátor SQL jsme rozšířili o některé další funkce, které ještě podrobněji specifikují vytvoření databázových tabulek. Jednalo se konkrétně o některé další klauzule příkazu CREATE TABLE, například LOCK MODE, EXTENT SIZE, NEXT SIZE, IN DBSPACE a podobně a o mechanismus volání uložených procedur vzhledem k rozdílným možnostem databázových strojů verze 5.0 a vyšších a aplikačních popředí INFORMIX-4GL verze 4.1.

Rozhraní encyklopedie (PIP - Programmer's Interface to Project Database) realizuje pohodlný a transparentní přístup ke všem datům uloženým v projektové databázi. Její struktura je poměrně složitá, protože data jsou uložena jak v databázových tabulkách, tak v obyčejných souborech (textové informace, protože BLOB datový typ TEXT se nepoužívá). Přímý přístup do archivu není nutný, nicméně je možný. Použili jsme tento postup ve starších verzích systému (2.1), kdy jsme generovali scénáře databáze vlastním generátorem SQL. Generování se provádělo z datového modelu vytvořeného pomocí ER diagramů v systému Westmount a uloženého v encyklopedii. Počínaje verzí systému 3.0 se tento postup stal zbytečným, protože přizpůsobení systému požadavkům uživatele je daleko jednodušší provádět na úrovni generátorů a šablon Westmount v jazyku tcl.

3. Model procesů

Zatímco generování kódu SQL nevyžaduje od uživatele případně žádné úpravy funkcionality generátoru nebo textu šablon, je u generování kódu 4GL situace jiná. Do generovaného kódu se promítnou programátorské zvyky a postupy, které má každý softwarehouse vlastní (od uspořádání obrazovky až po řešení víceuživatelského přístupu, práci s transakcemi, používání jednotlivých nástrojů pro zamykání dat - příkazy LOCK, UNLOCK, zamykání pomocí kurzorů CURSOR FOR UPDATE, řízení úrovně izolace SET ISOLATION TO ... DIRTY READ, COMMITTED READ, CURSOR STABILITY a REPEATABLE READ apod). Další vlastnost, která odlišuje modelování procesů od modelování dat, je skutečnost, že datový model prochází spojitě všemi čtyřmi fázemi životního cyklu, kdežto u modelu procesů je situace složitější. Na úrovni analýzy a globálního návrhu se procesy modelují pomocí diagramu systémové architektury (SAD) a diagramu datových toků (DFD). Listové programové moduly jsou poté popsány pseudokódem v minispecifikacích a na tento pseudokód navazuje vytvoření modelu vnitřní struktury jednotlivých programových modulů pomocí strukturních diagramů (Structure Charts, SCD) ve fázi podrobného návrhu a implementace. Vlastní kód se potom generuje ze strukturních diagramů, představujících model interní stuktury programu, někdy také označovaný jako konstrukční model. Protože je generování kódu programu složitější problém než generování schématu databáze a scénářů SQL, omezují se některé dodávané prostředky CASE jen na problém datového modelu. Jazyk INFORMIX-4GL (a pravděpodobně všechny jazyky čtvrté generace) má navíc specifickou syntaxi procedurálního a neprocedurálního jazyka. Syntaxi jazyka SQL a jeho neprocedurálních příkazů (CREATE, SELECT, INSERT, DELETE a další) definuje norma ANSI a jsou interpretovány databázovým strojem, neprovádí se kontrola jejich syntaxe při překladu, ale až při jejich vykonávání. Neprocedurální a procedurální příkazy vlastního jazyka INFORMIX-4GL (MENU, INPUT, FOR, WHILE, CASE apod.) kontroluje překladač či interpret a vykonává je aplikační program (front-end). Jazyku 4GL bývá vytýkáno, že není jednoklonný - viz např. existence funkcí DATE a DATE() nebo používání datového typu SERIAL. Zřejmě i z tohoto důvodu neobsahuje systém zpětný převod (reverse engineering) ze zdrojového kódu jazyka 4GL do strukturních diagramů, ale jen generování ERD ze schématu existující databáze.

Zdá se, že základním problémem generování kódu je otázka, z jakého zdroje kód generovat. Existují v zásadě dva základní přístupy, z nichž každý má své výhody a nevýhody. První přístup používá například prostředek INFORMIX-FORMS, který představuje programový balík označovaný jako nižší CASE (lower CASE), pomáhající řešiteli ve fázi implementace aplikace, při generování kódu. Zdrojový kód aplikace se generuje z podrobného popisu formuláře, který uživatel vytvoří v dialogu pomocí nabídek. Popíše typ formuláře (header, detail, zoom apod.), pole, které má formulář obsahovat a jejich charakteristiky - šířka pole, reverzní zobrazení a další atributy. Tyto informace použije systém jednak k vygenerování formuláře, rozšířeného proti standardu Informix o jednu sekci navíc, jednak k vygenerování zdrojového kódu jazyka 4GL, tvořeného několika moduly. Tento přístup má výhodu především v tom, že je uživatelsky velmi příjemný a snadno zvládnutelný a výsledný efekt se dostaví velmi záhy. Nevýhodou je, že kód se generuje podle interních pravidel generátoru, která není možné na uživatelské úrovni jednoduše konfigurovat a je otázkou času, kdy uživatel narazí na toto omezení. Bez zásahu do složitého kódu generátoru je možné určovat vlastnosti generovaného kódu pouze těmi prostředky, které nabízí popis formuláře. Uvedený přístup připomíná generátor formulářů I-SQL PERFORM, ale s daleko bohatšími možnostmi.

Druhý možný přístup používá Westmount I-CASE. Generátor nevytváří zdrojový kód na základě interních pravidel a algoritmů, ale nechává popis vnitřní struktury a chování programového modulu na uživateli. Existuje grafický model - strukturní diagram (Structure Chart, SCD), který pomocí grafických symbolů popisuje elementární procedurální i neprocedurální operace, které modul vykonává (sekvence, selekce, iterace, předávání parametrů a návratových hodnot apod.).

Potřeby aplikace jsou podle povahy aplikace někde uprostřed. Globální strukturu aplikace a některé speciální funkce bude vždy nutné popsat v SCD. Část funkcí na listové úrovni je možné vygenerovat, aniž se vytváří model vnitřní struktury funkce, která formulář obsluhuje.

Generování kódu jazyka 4GL ze strukturního diagramu využívá stejně jako generování kódu SQL tři zdroje - grafický model podporovaný jazykem pdl, který generátor převede do vnitřního tvaru, scénář generátoru v jazyku tcl a šablony standardních operací, napsané v jazyku tcl a 4GL. Dodávaný systém obsahuje čtyři šablony pro základní databázové operace (insert, delete, find a update) a demonstrační příklad, na kterém je možné používání šablon prostudovat a který je zajímavý např. z hlediska přístupu k operacím měnícím obsah databáze nebo z pohledu zajímavých programátorských postupů, umožňujících nahradit například kurzory pro zpracování výsledku dotazu dynamicky alokovaným vázaným seznamem, realizovaným pomocí funkce v jazyku C volané z prostředí 4GL. Pro praktickou práci je třeba dodat jak uživatelem definované šablony, tak rozšířit i informace ukládané při vyváření modelu procesů i datového modelu.

Procento generovaného kódu bude přímo úměrné povaze aplikace a délce používání prostředku, která se odráží v počtu různých šablon, ze kterých se kód generuje. Obvykle se uvádí rozsah generovaného kódu v rozsahu 60 - 80%, podle povahy aplikace.

4. Model časově závislého chování

Chování systému v čase je možné modelovat pomocí diagramu posloupnosti formulářů (Form Sequence Diagram), někdy nazývaného také diagram návaznosti obrazovek nebo pomocí stavového diagramu (State Transition Diagram), převzatého nově od verze 3.0 ze systému určeného pro modelování aplikací reálného času. Diagram návaznosti obrazovek zobrazuje navigaci nabídkovým systémem aplikace od vrcholové nabídky po formuláře na listové úrovni. Uzly reprezentují obrazovky jednotlivých nabídek či formulářů a hrany přípustné přechody. Ohodnocení hran definuje událost, která vede ke změně stavu systému a akci, která se provede. Nabídky, které používá systém Westmount v prostředí INFORMIX-4GL jsou horizontální nabídky vytvořené standardním příkazem MENU jazyka 4GL. Protože zavedený systém tvorby nabídek tvoří u nás sloupcové nabídky, které se vytvářejí dynamicky na základě konfiguračních tabulek nebo konfiguračních souborů, využíváme FSD jako grafickou reprezentaci nabídek, ze kterého scénář tcl generuje tyto konfigurační soubory.

Na úrovni formulářů umí systém generovat standardní tvar specifikačního souboru používaný v Informixu zhruba na úrovni prostředku I-SQL Perform. Funkcionalitu generátoru formulářů je možné upravit, protože generování se provádí scénářem tcl. Dále je do systému zaintergrován program pro kreslení formulářů převzatý ze systému INFORMIX-FORMS. Vzhledem k rutině v psaní specifikačních souborů formulářů získané během používání Informixu tyto možnosti nevyužíváme, ukládáme ovšem do popisu formulářů ve FSD některé další informace (například typ formuláře), které jsou nezbytné při generování kódu ze standardních šablon jazyka 4GL.

5. Závěr

V souvislosti s prostředky CASE se nejčastěji nabízejí dvě otázky:

Nepředstavují nástroje CASE jen módní výstřelek, který zavádí do vývoje aplikací zbytečnou byrokracii, potlačující tvůrčí přístup řešitelů? Zdá se, že jestliže metody a postupy, které CASE nástroje využívají, existovaly a používaly se bez ohledu na to, zda byly nebo nebyly podporovány počítačem, pak samotný fakt jejich počítačové implementace může jejich používání jen podpořit, protože se stávají uživatelsky příjemnější a snáze použitelné.

Šetří CASE práci, náklady nebo řešitelskou kapacitu? Při diskusích o výhodnosti prostředků CASE se obvykle uvádí graf, který demonstruje rovnoměrnější rozložení nákladů v průběhu životního cyklu aplikace, s nárůstem nákladů ve fázi analýzy a návrhu a s nižšími náklady ve fázi implementace a údržby. Protože jsme se zatím nedostali do fáze údržby programového díla, je práce spíše více. Používání nástroje se promítlo do povahy práce ve fázi implementace, kde ubývá kódování a mění se povaha práce. Zvládnutí metody a jednotlivých technik i tvorba šablon a úprava generátorů je časově náročná činnost. Na druhé straně ovšem generování SQL scénářů včetně uložených procedur v rozsahu řádově stovek kilobytů zdrojového textu a zdrojového textu jazyka 4GL představuje jak úsporu ve fázi kódování, tak hlavně zajištění bezchybnosti generovaného kódu s perspektivou toho, že práce investovaná do tvorby šablon a úpravy generátorů bude v budoucnosti u dalších projektů už jen přinášet úspory.

Možnosti nástrojů CASE se vyvíjejí velmi dynamicky jak v používaných metodách, tak nástrojích. Zdá se, že budoucí směr vývoje se bude ubírat směrem k další integraci a objektově orientovaným postupům.

Po získaných zkušenostech se zdá, že tyto prostředky splňují to, co obsahují ve svém názvu. Nejsou zázračným nástrojem, ale podporují řešitele v jeho práci a dodávají jí inženýrský rozměr, umožňující efektivněji vést řešení projektu ve všech jeho etapách.

Autor:

Ing. Jiří Felbáb
Softwarové Aplikace & Systémy
Tiskařská 10/257
108 28 Praha 10 - Malešice
tel: 70 51 34, fax: 70 51 28
E-mail: jfel@sasprg.cz