PODPORA VÝVOJA KONFIGURÁCIÍ PROGRAMOVÉHO SYSTÉMU

A.Pandák, I.Bojtošová, M. Sýkora - Výskumný ústav lekárskej bioniky, Bratislava

Údržba rozsiahleho programového systému, ktorý sa ďalej vyvíja v priebehu prevádzky a u ktorého existuje viac implementácií, je náročnou a komplikovanou činnosťou. Bez systematického přístupu ku nej nie je možné zabezpečiť kontrolu nad systémom v období životného cyklu po implementácii systému a jeho vývoj ako celku (a pritom funkčného v rozných lokalitách) smerom ku vyššej kvalite a optimalizácii. Bez počítačovej podpory sa jedná o prácu časove náročnú a málo efektívnu. To je aj dovod toho, že sa v poslednom období vynára nová disciplína v rámci softvérového inžinierstva - údržba konfigurácií vo viacnásobnej (opakovanej) prevádzke vyvíjaných rozsiahlych softvérových projektov. V literatúre sa zvykne označovať ako "software configuration management" (SCM)(1),(2) prípadne"family concept support" (3).

Naše doterajšie skúsenosti s prevádzkou jedného systému v rozných prostrediach (opakovanými implementáciami) potvrdzujú, že je nutné venovať cielené úsilie riešeniu tejto problematiky - jej zanedbanie sa vráti v enormnej spotrebe kapacít nutných pre zvládnutie prevádzky. Z tohto dovodu sme venovali zvláštnu pozornosť problematike SCM, navrhli a vytvorili vlastné prostriedky pre jej riešenie.

Rozsiahly softvérový systém predstavuje interakciu množstva komponentov, ktorých integrácia vytvára výslednú konfiguráciu systému. Modifikáciou niektorej (niektorých) komponentov, resp. pridaním novej môže vzniknúť nová konfigurácia. Cieľom SCM je podpora vývoja softvérového systému predstavovaného množinou konfigurácií (vyjadrujúcich vlastne vývoj systému v diskrétnych bodoch jeho životného cyklu) tak, aby bolo možné efektívne a bezkonfliktne paralelne prevádzkovať jednotlivé konfigurácie v rozných prostrediach (lokalitách, na rozných počítačoch apod.).

Pri plnení tohto cieľa sa od SCM očakáva pomoc pri riešení nasledovných problémov :

- identifikácia konfigurácií (definícia ich komponentov a väzieb ako aj väzieb medzi konfiguráciami)
- podpora zmien komponentov (aj s dozorom dopadu zmien na stav systému, tj. udržanie jeho integrity ako celku),

- zobrazenia aktuálneho stavu systému v danom momente
- zachytenie histórie vývoja systému.

Systematický prístup ku tejto činnosti má zatiaľ krátku históriu, v poslednej dobe sa jedná najmä o činnosť v spojení s programovým prostredím v rámci UNIXu (1) resp. s programami v jazyku ADA. Často sa v tejto súvislosti hovorí o tzv. jazykoch pre prepojenie modulov (Module Interconnection Languages) (4),(5), ktoré sa snažia formálne zachytiť štruktúru konfigurácií systému na základe popisu vzájomných väzieb zložiek systému.

Ukázalo sa, že aby bolo možné takéto prostriedky využiť pri vyvíjajúcom sa systéme (vznik nových verzií jeho subsystémov, konfigurácií apod.), je vhodné o takomto systéme uvažovať ako o rodine príbuzných systémov. Koncepcie rozvíjané v rámci prác na prostriedkoch SCM zdôrazňujú potrebu sústrediť pozornosť na "programovanie vo veľkom" (programming in-large) - tj. na úroveň interakcí medzi modulmi, na rozdiel od "programovania v malom" (in-small) - na úrovni algoritmov, dátových štruktúr apod.

Vychádzajúc z uvedených prístupov a skúseností z prevádzky, sme našu koncepciu SCM založili na predstave formálnej definície konfigurácií systému, ako základného zdroja pre podporu jeho ďalšieho vývoja v podmienkach paralelnej prevádzky väčšieho množstva implementácií.

Prevádzkovaný programový systém chápeme ako strom (acyklický graf), ktorého listy predstavujú moduly a nelistové uzly sú subsystémy. Moduly sú reálne prezentované programom (skupinou programov), subsystém predstavuje príslušný riešený problémový okruh (skupinu problémových okruhov). V rámci stromu systému je potom možné mať hierarchie jednotlivých subsystémov (predchodca, nasledovník).

Každý uzol môže predstavovať rodinu subsystémov - čo je v skutočnosti niekoľko verzií subsystému, v závislosti od toho, ktoré verzie modulov sme pri realizácii použili.

Strom konfigurácií systému predstavuje implicitne vyjadrené všetky možné konfigurácie systému, je však zrejmé, že iba obmedzený počet z nich má praktický význam (nie je zmysluplná konfigurácia verzií modulov v rôznych jazykoch - českom, slovenskom, ruskom apod., resp. verzie pre rôzne počítače - PC verzia s využitím grafiky a verziami pre negrafické terminály apod.).

Aby bolo možné praktický využívať uvedený popis konfigurácií systému, je potrebné mať k dispozícii možnosť zachytiť :

- vertikálne vzťahy medzi prvkami v konfigurácii,
- horizontálne vťaby jednotlivých verzií v rámci rodiny,
- popis existujúcich konfigurácií.

Na tento účel sme zaviedli jednoduchý jazyk CML (Configuration Mangement Language). Jadrom nášho prístupu je zachytiť vzťahy medzi prvkami systému pomocou zoznamov zdrojov poskytovaných prvkami a zoznamov zdrojov požadovaných prvkami. Zdrojom možu byť funkcie, premenné, časti databázy atď. Každý prvok môže požadovať určité zdroje a naopak poskytovať zdroje iným prvkom. Primárnym cieľom je potom zabezpečiť, aby všetky prvky výsledného systému mali přístup ku zdrojom, ktoré požadujú. V tomto zmysle chápeme vertikálne vzťahy medzi prvkami konfigurácie ako rozhranie, ktorým prebieha výmena zdrojov medzi prvkami. Horizontálne vzťahy v rámci prvkov tej istej rodiny charakterizujú rozdiely medzi množinami poskytovaných, resp. požadovaných zdrojov.

V ďalšom budeme za prvky syatému považovať subsystém a problémový okruh (obidva sú neterminálnymi prvkami stromu systému) a problém (terminálne prvky stromu systému, nahradíme nimi použitý pojem modul).

CML umožňuje popísať rodinu problémov a s využitím popisu rodiny problémov rodiny problémových okruhov a subsystémov - tj. jednotlivé konkrétne konfigurácie.

Popis rodiny problému sa skladá z piatich častí :
- zoznamu minimálne poskytovaných zdrojov
- popisu vlastností poskytovaných zdrojov
- zoznamu nutne vyžadovaných zdrojov
- popisu vlastností vyžadovaných zdrojov
- popisu jednotlivých verzií (implementácií).

Prvé štyri časti popisu charakterizujú rodinu problému - každá verzia problému musí vyhovieť tejto charakteristike.

Popis jednotlivých verzií (implementácií), môže už byť špecifický, ovplyvnený príslušnou implementáciou. Obsahuje :
- popis zdrojov poskytovaných a požadovaných naviac (alebo v implementačne viazanej forme) od rodinou minimálne poskytovaných a nutne vyžadovaných zdrojov
- popis realizácie (programy príslušných verzií)
- charakteristiky prípadných variánt v rámci jednej verzie.

Ako príklad uvádzame popis problému ordinácie farmakoterapie.

problémORDPAR
poskytujeZT=zobrazenie terapie pacienta
MT=modifikácia aktuálnej terapie
VT=výpis aktuálnej terapie
LIE=zobrazenie množiny prístupných liekov
vlastnostiZT=zobrazí všetky lieky pacienta v členení -
názov,forma, sila,dávka,podanie,poznámka
MT=...
.
.
.
LIE=umožní výber lieku z kmeňového súboru
liekov podľa prvých troch písmen lieku,
zadaním názvu lieku s využitím Soundexových kódov
alebo listovaním podľa farmakodynamických skupín
požadujeSUBLIE=číselník liekov
COM=identifikačné údaje pacienta
vlastnostiSUBLIE=HVLP MEDIKA, kmeňový súbor
COM=common premenné časť 1
implementácie
verziaNaP L.Dérera
realizácia (zoznam programov realizujúcich verziu)
poskytujeST=sumárny prehľad terapie pre prepúšťaciu správu
vlastnostiST=zoznam všetkých liekov ordinovaných pacientovi
v priebehu hospitalizácie s uvedením celkovej
dávky za toto obdobie
variantaklinika farmakoterapie
realizácia (zoznam upravených programov)
poskytuje u antibiotík počet dní podávania
požaduje -
verziaÚÚŽŽ Praha
realizácia (zoznam programov)
poskytujeČV=česká verzia ordinácie farmakoterapie
ST=sumárny prehľad terapie pre prepúšťaciu správu
vlastnostiČV=všetky výstupné texty sú v češtine
ST=zoznam všetkých liekov ordinovaných pacientovi v
priebehu hospitalizácie s uvedením celkovej dávky za
toto obdobie
požadujeSUBCLIE=číselník liekov
vlastnostiSUBLIE=čeká verzia číselníka liekov

Z uvedeného príkladu je zrejmé, že charakteristika rodiny je veľmi blízka jej špecifikácii (čo robí, čo poskytuje používateľovi). Nad prostriedkami CML sme vybudovali programovú nadstavbu, ktorá spracovaním stromu systému umožňuje používateľovi špecifikovať požiadavky na svoju vlastnú konfiguráciu - výberom z možností poskytovaných v rámci celého stromu.

Na definíciu rodiny problémov nadväzuje popis rodiny implementovaných systémov, tj. jednotlivých konfigurácií. Obsahuje zoznam konfigurácií, pričom každá konfigurácia je zoznamom problémových okruhov ktoré obsahuje a každý problémový okruh konfigurácie je definovaný zoznamom verzií problémov ktoré používa (ktoré jeho riešenie zabezpečujú). Popis konfigurácií zapisujeme nasledovne (PO = problémový okruh).

systém

konfigurácia SM-4=PO A, PO B, PO C,...,PO I

PO A=problém A1, problém A2.verzia 3,
problém A3.verzia 2.variante 2,...

B=problém B1,....

.
.
.

konfigurácia SM 50/50=

.
.
.

Jednotlivé problémy v rámci popisu sú uvedené v tvare (ID = identifikátor problému):

ID	použitá prvá verzia, prvá varianta
ID.číslo verzie	použitá druhá alebo vyššia verzia a jej prvá varianta
ID.číslo verzie.číslo varianty	použitá druhá a vyššia varianta

Ako vidieť zo sposobu zápisu identifikátora problému, okrem verzie rozlišujeme v rámci verzií tzv. varianty - umožňujú rozlíšiť jednoduché rozdiely v riešení (modifikácii) jednotlivých problémov v rámci tej istej varianty.

Popísané prostriedky CML sú súčasťou programového riešenia ktoré sme nazvali Generátor Aplikačných Programov Informačného

Systému (GAPIS). GAPIS predstavuje špecifický typ programového prostredia, umožňujúceho rýchle a efektívne vytvárať informačné systémy určené pre spracovanie údajov v prostredí zdravotníckych zariadení (nemocničné oddelenie, laboratória atď.). Ako sme už naznačili, spracovaním stromu systému je možné navrhnúť požadovanú špecifikáciu takéhoto informačného systému. Táto je vstupom do procesora špecifikácie GAPISu, ktorý ju spracuje a výsledok spracovania (programy, zabezpečujúce požadované riešenie), je zapísaný do DBI (databáze implementácií), ako nová konfigurácia vo forme štruktúr verzií a variánt programov riešiacich problémy a problémové okruhyktoré ju tvoria. Pri tejto činnosti využíva procesor špecifikácie strom systému zapísaný pomocou jazyka CML. Cieľom procesora je transformácia špecifikácie (popisujúcej požiadavky používateľa na pokrytie jednotlivých pracovísk - prostredia IS) do cieľovej štruktúry DBI.

SCM hrá kľúčovú úlohu pri udržiavaní prehľadu o ďalšom vývoji vytvorených systémov. Procesor špecifikácie postupne dopĺňa konfigurácie nových implementácií do súboru DBI. SCM registruje a zachytáva stav týchto konfigurácií a zabezpečuje ich ďalší vývoj - na základe požiadaviek jednotlivých používateľov na úpravu ich implementácií. Vykonané úpravy sa premietajú do stromu systému (nové verzie a varianty), do DBI (modifikácie používaných konfigurácií) a do DB dokumentácie. SCM takto zabezpečuje v systéme úlohu editora uvedených databáz a prostredníctvom svojich možností pre formálny zápis konfigurácií hrá významnú úlohu pri prezentácii a ďalšej údržbe programovej štruktúry vytváraných IS.

Literatúra

(1) MacKay,D.: A Unix Based System for Software Configuration Management, The Computer Journal, No. 6, 1986

(2) Narayanaswamy,K.,Scacchi,W.: Maintaining Configurations of Evolving Software Systems, IEEE Trans.Software Eng, March, 1987

(3) Ramamoorthy, C.V.,Vijay,G.,Prakash,A.: Programming in the Large, IEEE Trans.Software Eng.,July 1986

(4) Prieto-Diaz,R.,Neighbors,J.: Module Interconnection Languages: A Survey.Univ.Carolina,Irvine,ICS Tech.Rep.189,1982

(5) DeRemer,F., Kron,H.K.: Programming-in-the large versus Programming-in-the small, IEEE Trans. Software Eng.,June, 1976