Technologie plné rozšiřitelnosti v objektově orientovaných softwarových systémech

Ing. Vladimír Kuchta, AKOS, Horní předměstí 424, 741 01 Nový Jičín, Česká republika

Abstrakt

Článek pojednává o praktických otázkách znovupoužitelnosti už vytvořených softwarových dílů s ohledem na údržbu těchto dílů.

Popisuje rysy, kterými se softwarový systém musí vyznačovat, aby jej bylo možné nazvat systémem plně rozšiřitelným. Potom vyhodnocuje, do jaké míry se takovými rysy vyznačují jak softwarové systémy uskutečňující znovupoužívání softwarových dílů prostřednictvím dědění, tak softwarové systémy uskutečňující znovupoužívání softwarových dílů prostřednictvím skládání.

Článek vysvětluje, co je překážkou, která způsobuje, že technologie plné rozšiřitelnosti zatím nemůže být, zejména u otevřených softwarových systémů, v praxi běžně využívána.

Potom se zabývá ve světě probíhajícím výzkumem, který je zaměřen na odstranění překážky způsobující prozatímní praktickou nemožnost využívat technologii plné rozšiřitelnosti. Jsou popsány výsledky jedné zahraniční výzkumné práce, která přinesla matematicky dokázané závěry.

Článek uvádí závěry této výzkumné práce jako seznam přikázání pro tvůrce software.

1. PROČ SE TÍM VŮBEC ZABÝVAT

Pro zvýšení produktivity při vytváření, změnách a úpravách softwarových systémů se z implementačního hlediska jako nejdůležitější jeví znovupoužívání již jednou vytvořených softwarových dílů (programových částí). Nejlepší podmínky pro znovupoužívání budou v plně rozšiřitelných softwarových systémech.

V rozšiřitelných nedědicích softwarových systémech čili v softwarových systémech, které se softwarovým systémům plně rozšiřitelným přibližují, jsou pro znovupoužívání nejlepší podmínky, jakých lze dosáhnout v současné době.

2. CO TO JE

Tato kapitola vysvětluje, co se myslí některými slovními spojeními (nebo slovy) používanými v tomto článku.

2.1 Objektový typ, činnost, název činnosti, náplň činnosti příkaz k provedení činnosti, softwarový díl, znovupoužívání, softwarová sada, softwarový systém

Objektovým typem se myslí datový typ, který svým instancím, tj. objektům, určuje jak množinu datových položek, tak množinu hlaviček metod, tak množinu těl metod, tak i přiřazení těl metod hlavičkám metod.

Činností se nazývá taková metoda z objektového typu, která je schopna provedení s pozdní vazbou nebo je konstruktorem. Názvem činnosti se rozumí hlavička metody. Náplní činnosti je tělo metody, včetně všech lokálních deklarací. Náplň činnosti se považuje za nedefinovanou, pokud tělo metody buď neexistuje, nebo obsahuje jen příkazy ke sdělení volajícímu okolí, že náplň činnosti není definována.

Jako příkaz k provedení činnosti se označuje buď volání metody, která je procedurou, nebo příkaz, jehož součástí je výraz obsahující volání metody, která je funkcí.

Implementačním objektovým typem se rozumí objektový typ, který se vyznačuje jak konkrétní množinou datových položek, tak konkrétní množinou názvů činností, tak i konkrétní množinou náplní činností, přičemž všechny činnosti z tohoto objektového typu mají definovanou náplň.

Softwarovým dílem (programovou částí) je implementační objektový typ.

Znovupoužíváním se rozumí používání softwarového dílu prostřednictvím obecného identifikátoru (např. globálního identifikačního čísla) na jiných místech v softwarových systémech, než bylo místo prvního použití. Za znovupoužívání se nepovažuje pouhé přenášení částí zdrojových textů vytvářejících softwarový díl.

Softwarovou sadou se nazývá souhrn několika softwarových dílů. Překlad softwarových dílů se provádí prostřednictvím překladu softwarových sad.

Softwarovým systémem se rozumí takový objektově orientovaný softwarový systém, v němž má každý objekt svůj objektový typ a v němž každý objektový typ může být s jiným objektovým typem ve vztahu dědění, tj. ve vztahu předek - potomek, nebo (prostřednictvím instancí) ve vztahu skládání, tj. ve vztahu celek - část (nadřízený - podřízený).

2.2 Dědicí software, kvazipředek, kvazipotomek, kvazidědicí software

Softwarové systémy uskutečňující znovupoužívání softwarových dílů prostřednictvím dědění jsou systémy, v nichž první softwarový díl vystupuje na místě svého znovupoužití jako předek sloužící k odvození potomka, tj. druhého softwarového dílu. Tímto způsobem jsou vystavěny běžně používané knihovny objektových typů.

Softwarové systémy uskutečňující znovupoužívání softwarových dílů prostřednictvím dědění budou dále označovány slovním spojením dědicí software.

Softwarové systémy uskutečňující znovupoužívání softwarových dílů prostřednictvím skládání jsou systémy, v nichž první softwarový díl vystupuje na místě svého znovupoužití ve formě zmocněnce, tedy ve formě své vlastní instance, která je přiřazena datové položce v nadřízeném druhém softwarovém dílu a která slouží svými činnostmi k alespoň částečnému vytvoření alespoň některých prvků množiny náplní činností nadřízeného druhého softwarového dílu.

První softwarový díl tak prostřednictvím zmocněnce hraje roli kvazipředka a jemu nadřízený druhý softwarový díl prostřednictvím své instance roli kvazipotomka.

Softwarové systémy uskutečňující znovupoužívání softwarových dílů prostřednictvím skládání budou dále označovány slovním spojením kvazidědicí software

2.3 Nezmocněné provedení činnosti, zmocněné provedení činnosti provedení činnosti s kvazibrzkou vazbou, provedení činnosti s kvazipozdní vazbou, nedědicí náplň činnosti

U kvazidědiciho software se v náplních činností kromě parametru Self (This apod., dále jen Self) pracuje s parametrem KvaziSelf (KvaziThis apod., dále jen KvaziSelf).

Instancí, která se nazývá nezmocněncem, je u kvazidědicího software činnost provedena, je-li v náplni provedené činnosti tato instance totožná s instancí představovanou parametrem KvaziSelf. Takové provedení činnosti se dále označuje slovním spojením nezmocněné provedení činnosti.

Instancí, která se nazývá zmocněncem, je u kvazidědicího software činnost provedena, není-li v náplni provedené činnosti tato instance totožná s instancí představovanou parametrem KvaziSelf. Takové provedení činnosti bude dále označováno slovním spojením zmocněné provedení činnosti.

Obdobou provedení činnosti s brzkou vazbou u dědiciho software je u kvazidědicího software zmocněné provedení činnosti. Taková obdoba provedení činnosti s brzkou vazbou se dále označuje jako provedení činnosti s kvazibrzkou vazbou.

Obdobou provedení činnosti s pozdní vazbou u dědiciho software je u kvazidědiciho software nezmocněné provedení činnosti. Taková obdoba provedení činnosti s pozdní vazbou bude dále označována slovním spojením provedení činnosti s kvazipozdní vazbou.

Nedědicí náplní činnosti u kvazidědicího software se rozumí taková náplň činnosti, ve které příkazy k provedení činností (jakýmikoli instancemi) buď žádné nejsou, nebo tam jsou jen příkazy k nezmocněnému provedení činností, z nichž každá má výhradně nedědicí náplň.

2.4 Příkaz k provedení činnosti kvazipředkem, kvazizdědění datové položky, kvazizdědění názvu činnosti, kvazizdědění náplně činnosti, kvazidědičnost

Obdobou příkazu, který se nachází v náplni činnosti potomka, k provedení činnosti předkem u dědicího software je u kvazidědicího software příkaz, který se nachází v náplni činnosti kvazipotomka, ke zmocněnému provedení činnosti. Taková obdoba příkazu k provedení činnosti předkem se dále označuje jako příkaz k provedení činnosti kvazipředkem.

Obdobou zdědění datové položky od předka u dědicího software je u kvazidědicího software možnost používat datovou položku kvazipředka zmocněným prováděním činností. Taková obdoba zdědění datové položky bude dále označována slovním spojením kvazizdědění datové položky.

Obdobou zdědění názvu činnosti od předka u dědicího software je u kvazidědicího software zařazení téhož názvu činnosti do deklarace kvazipotomka. Taková obdoba zdědění názvu činnosti je dále označována jako kvazizdědění názvu činnosti.

Obdobou zdědění náplně činnosti od předka u dědicího software je u kvazidědicího software ten příkaz ke zmocněnému provedení činnosti, který tvoří celou náplň stejnojmenné činnosti kvazipotomka, a to buď sám, nebo spolu s příkazy k odlišně (tj. na instance jiných kvazipředků) zmocněnému provedení výhradně stejnojmenné činnosti. Taková obdoba zdědění náplně činnosti se dále označuje slovním spojením kvazizdědění náplně činnosti.

Taková obdoba dědičnosti u dědicího software, při jejímž využití je u kvazidědicího software nahrazeno jak zdědění datové položky jejím kvazizděděním, tak zdědění názvu činnosti jeho kvazizděděním, tak i zdědění náplně činnosti jejím (náplně) kvazizděděním, bude dále označována slovem kvazidědičnost. Obecně je kvazidědičnost vícenásobná.

2.5 Zobecněně dědicí software, zobecněný předek, zobecněný potomek, parametr ZSelf, zobecněná dědičnost, provedení činnosti se zobecněnou brzkou vazbou, provedení činnosti se zobecněnou pozdní vazbou, příkaz k provedení činnosti zobecněným předkem, zobecněné zdědění datové položky, zobecněné zdědění názvu činnosti, zobecněné zdědění náplně činnosti

Zobecněně dědicím software se rozumí software, který je buď dědicím software, nebo kvazidědicím software.

Zobecněným předkem se nazývá buď předek (v případě, kdy zobecněně dědicí software je dědicím software), nebo kvazipředek (v případě, kdy zobecněně dědicí software je kvazidědicím software). Zobecněným potomkem je buď potomek (v případě, kdy zobecněně dědicí software je dědicím software), nebo kvazipotomek (v případě, kdy zobecněně dědicí software je kvazidědicím software).

Jako parametr ZSelf se označuje buď parametr Self (v případě, kdy zobecněně dědicí software je dědicím software), nebo parametr KvaziSelf (v případě, kdy zobecněné dědicí software je kvazidědicím software).

Zobecněnou dědičností se nazývá buď dědičnost (v případě, kdy, zobecněně dědicí software je dědicím software), nebo kvazidědičnost (v případě, kdy zobecněně dědicí software je kvazidědicím software).
Ostatní pojmy z nadpisu této podkapitoly 2.5 jsou definovány obdobným způsobem jako např. zobecněný předek a zobecněná dědičnost.

2.6 Zobecněně dědicí polymorfismus, nedědicí polymorfismus

Zobecněně dědicím polymorfismem se rozumí polymorfismus uskutečňovaný prostřednictvím využívání zobecněné dědičnosti.

Nedědicím polymorfismem je polymorfismus, který není uskutečňován prostřednictvím využívání zobecněné dědičnosti.

2.7 Odvozenina, odvozenec, odvoditel, odvozovátor, upravenina, úprava, údržbový zásah

Jako odvozenina prvního softwarového dílu se označuje druhý softwarový díl, který je bezprostředním zobecněným potomkem prvního softwarového dílu. Odvozencem druhého softwarového dílu se rozumí první softwarový díl, který je bezprostředním zobecněným předkem druhého softwarového dílu. Odvozenec a odvozenina tvoří bezprostřední dvojici zobecněného předka (odvozenec) a zobecněného potomka (odvozenina).

Odvoditelem je ta část odvozeniny, která nepochází z odvozence, tj. část, jež nebyla do odvozeniny přenesena využitím zobecněné dědičnosti. Ukázku odvoditele přináší podkapitola 3.2.

Odvozovátorem se rozumí operátor, který vytváří ze svých dvou operandů (tj. z odvoditele a odvozence) odvozeninu. Odvození lze pak popsat rovnicí:

Odvoditel Odvozovátor Odvozenec = Odvozenina.

Odvozenec je tedy operandem, nikoli výsledkem. Výsledkem je odvozenina.

Upraveninou prvního softwarového dílu se nazývá druhý softwarový díl, který představuje novou verzi prvního softwarového dílu a který je určen k dosazení na místo prvního softwarového dílu. Upravenina může, ale nemusí, být odvozeninou.

Jako úprava neboli údržbový zásah se označuje takový zásah (změna nebo oprava softwarového dílu i vytvoření zcela nového softwarového dílu), jehož výsledkem je upravenina dosazená na místo softwarového dílu, kterého je upravenina novou verzí.

2.8 Návrhář, realizátor, odvozovatel

Návrhářem softwarového dílu se nazývá osoba, která vytváří deklaraci softwarového dílu (tj. množinu datových položek a množinu názvů činností). Realizátorem softwarového dílu se rozumí osoba, která vytváří množinu náplní činností softwarového dílu. Odvozovatelem je osoba (ať už návrhář, nebo realizátor), která vytváří softwarový díl jako odvozeninu z odvozence.

2.9 Zařaditelnost softwarového dílu

Dva softwarové díly se považují za navzájem zaměnitelné z hlediska názvů činností, mají-li totožné množiny názvů činností.

Druhý softwarový díl je jednosměrně zaměnitelný z hlediska názvů činností za první softwarový díl, je-li množina názvů činností druhého softwarového dílu nadmnožinou množiny názvů činností prvního softwarového dílu. Taková jednosměrná zaměnitelnost z hlediska názvů činností se dále označuje jako zařaditelnost softwarového dílu (na místo jiného softwarového dílu).

Je-li druhý softwarový díl zařaditelný na místo prvního softwarového dílu, znamená to, že instance druhého softwarového dílu jsou schopny reagovat na všechny takové příkazy k provedení činnosti, na něž jsou schopny reagovat instance prvního softwarového dílu.
Splnění zařaditelnosti lze kontrolovat překladačem, poskytuje-li programovací systém nástroje, které takovou kontrolu umožňují.

2.10 Náhradyschopnost softwarového dílu

Chováním softwarového dílu se rozumí zvnějšku pozorovatelné chování instancí tohoto softwarového dílu.

Dva softwarové díly se považují za navzájem zaměnitelné z hlediska náplní činností, mají-li instance prvního softwarového dílu stejné chování jako instance druhého softwarového dílu.

Druhý softwarový díl je jednosměrně zaměnitelný z hlediska náplní činností za první softwarový díl, je-li chování instancí druhého softwarového dílu stejné nebo lepší než chování instancí prvního softwarového dílu. Taková jednosměrná zaměnitelnost z hlediska náplní činností bude dále označována slovním spojením náhradyschopnost softwarového dílu (za jiný softwarový díl).

Je-li druhý softwarový díl náhradyschopný za první softwarový díl, znamená to, že druhý softwarový díl zachovává nebo prohlubuje ty rysy chování (vlastností), které lze prokázat prvnímu softwarovému dílu.

Splnění náhradyschopnosti nelze kontrolovat překladačem, protože pro takovou kontrolu není možné stanovit univerzální postup (jde o obdobu problému zastavení Turingova stroje).

2.11 Dosaditelnost softwarového dílu

Dva softwarové díly se považují za navzájem zaměnitelné jak z hlediska názvů činností, tak z hlediska náplní činností, jsou-li navzájem zaměnitelné z hlediska názvů činností a zároveň navzájem zaměnitelné z hlediska náplní činností.

Druhý softwarový díl je jednosměrně zaměnitelný jak z hlediska názvů činností, tak z hlediska náplní činností za první softwarový díl, je-li druhý softwarový díl zařaditelný na místo prvního softwarového dílu a je-li zároveň druhý softwarový díl náhradyschopný za první softwarový díl. Taková jednosměrná zaměnitelnost jak z hlediska názvů činností, tak z hlediska náplní činností se dále označuje jako dosaditelnost softwarového dílu (na místo jiného softwarového dílu).

2.12 Zdokonalenina

Druhý softwarový díl se nazývá zdokonaleninou prvního softwarového dílu, je-li druhý softwarový díl dosaditelný na místo prvního softwarového dílu.

2.13 Uzavřený softwarový systém, otevřený softwarový systém

Uzavřeným softwarovým systémem se rozumí softwarový systém, v němž všechny softwarové díly, včetně všech odvozenin, jsou pod přímou, zpravidla jedním dodavatelem vykonávanou, správou svých tvůrců. To v principu umožňuje, byť u rozsáhlých systémů s velkými těžkostmi, zachytit problémy způsobené u odvozenin úpravami odvozenců a odstranit tyto problémy úpravami odvozenin. Rovněž je v principu možné po každém souhrnu úprav znovu odladit (a pak provést upgrade u odběratelů) všechny aplikace, v nichž jsou použity upraveniny nebo jejich odvozeniny.

Otevřeným softwarovým systémem se nazývá softwarový systém, v němž tvůrci a dodavatelé softwarových dílů nemají žádný přehled ani o odvozeninách vytvořených z dodaných odvozenců u odběratelů ani o aplikacích, v nichž jsou takové odvozeniny a odvozenci u odběratelů použity. Proto po nahrazení softwarového dílu upraveninou lze upgrade u odběratelů provádět jen na úrovni softwarových sad, ne na úrovni aplikací.

2.14 Houževnatost softwarového systému, ohebnost, monotónnost odvozovátoru

Jako houževnatý se softwarový systém označuje, splňuje-li právě jednu z těchto podmínek:

1) Je používán výhradně nedědicí polymorfismus,
2) Je používán zobecněně dědicí polymorfismus a zároveň je splněna podmínka houževnatosti pro zobecněně dědicí software

Podmínka houževnatosti pro zobecněně dědicí software je splněna, když vždy, při dosazení libovolné upraveniny libovolného odvozence na místo tohoto odvozence,

platí, pro libovolnou tímto dosazením ovlivněnou dvojici odvozenec - odvozenina, zároveň oba tyto výroky [1 - str. 4 a 25] [2 - str.4 a 18]:

1) Je-li U zdokonaleninou C a zároveň (L r C) zdokonaleninou C,
   pak (L r U) je zdokonaleninou C,
2) Je-li U zdokonaleninou C,
   pak (L r U) je zdokonaleninou (L r C),

kde

U - upravenina odvozence, C - odvozenec, L - odvoditel,
r - odvozovátor, (L r C) - původní (neovlivněná) odvozenina,
(L r U) - nová (ovlivněná dosazením upraveniny na místo odvozence) odvozenina.

Hraje-li odvozenina roli odvozence pro jinou odvozeninu, musí být oba výroky zároveň splněny i pro takovou další dvojici odvozenec - odvozenina

Ohebnost dvojice odvozenec - odvozenina je vlastnost vyjádřená výše uvedeným výrokem 1).

Monotónnost odvozovátoru nad odvozenci je vlastnost vyjádřená vztažením výše uvedeného výroku 2) na všechny tyto odvozence.

2.15 Plně rozšiřitelný softwarový systém, rozšiřitelný nedědicí softwarový systém

Plně rozšiřitelným softwarovým systémem se nazývá softwarový systém, jež se vyznačuje těmito rysy [3]:

A) Rysy, které musí být zajištěny použitým programovacím systémem -
   A1) Oddělený překlad softwarových sad,
   A2) Pro překlad softwarové sady, jež obsahuje odvozeninu, není nutné mít k dispozici zdrojový text odvozence,
   A3) Překlad softwarové sady nesmí mít vliv na správnost přeloženého kódu z jiných softwarových sad (jak se může stát u programovacích systémů, v nichž je typová kontrola založena na analýze celé aplikace),
B) Rysy, které musí být zajištěny použitou technologií tvorby software -
   B1) Při dosazování softwarového dílu, pocházejícího z již přeložené softwarové sady, na kterékoli místo jeho použití v kterékoli aplikaci se u žádné softwarové sady nevyžaduje její opětovný překlad,
   B2) Dosazovat softwarový díl, pocházející z již přeložené softwarové sady, na kterékoli místo jeho použití lze za běhu aplikace,
   B3) Houževnatost,
   B4) Zobecněně dědicí polymorfismus.

Rozšiřitelným nedědicím softwarovým systémem se rozumí softwarový systém, jež se vyznačuje kombinací rysů A1 a A2 a A3 a B1 a B2 a B3.

Z rysů náležejících do skupiny B se u dědičného software dá zajistit pouze rys B4, zatímco u kvazidědičného software se dá zajistit buď kombinace rysů B1 a B2 a B3, nebo kombinace rysů B1 a B2 a B4 Kombinace rysů B1 a B2 a B3 se u

kvazidědicího software dá zajistit jen tehdy, když všechny činnosti mají nedědicí náplň

Výhodou kvazidědicího software ve srovnání s dědicím software je schopnost buď zajistit navíc rysy B1 a B2, nebo zajistit rysy B1 a B2 a B3 při nezajištěném rysu B4. Společnou nevýhodou je prozatímní neschopnost zajistit rys B3 při zajištěném rysu B4. Nevýhodou dědicího software ve srovnání s kvazidědicím software je neschopnost zajistit rysy B1 a B2.

2.16 Technologie nedědicí rozšiřitelnosti, technologie plné rozšiřitelnosti

Technologií nedědicí rozšiřitelnosti se rozumí taková technologie tvorby softwarových systémů, která umožňuje tvorbu rozšiřitelných nedědicích softwarových systémů. Tato technologie je takovou nadmnožinou technologie tvorby kvazidědicího software, jež obsahuje navíc požadavek, že náplň všech činností musí být výhradně nedědicí.

Technologií plné rozšiřitelnosti se nazývá taková technologie tvorby softwarových systémů, která umožňuje tvorbu plně rozšiřitelných softwarových systémů. Tato technologie je takovou nadmnožinou technologie tvorby kvazidědicího software, jež musí obsahovat navíc postupy zajišťující houževnatost softwarového systému.

3. JAKÉ JSOU S TÍM POTÍŽE

Ustoupí-li tvůrci software od zobecněné dědicího polymorfismu (který je v současné době nejběžnější), budou potíže s tím, že nedědicí polymorfismus není zatím obvyklý.

Trvají-li tvůrci software na zobecněné dědicím polymorfismu, jsou potíže se zajištěním houževnatosti softwarového systému, protože odpovídající technologií tvorby softwarového systému je v takovém případě technologie plné rozšiřitelnosti a ta dosud není úplná. Chybějí v ní právě postupy, jak zajistit houževnatost. Tyto postupy se zatím nepodařilo stanovit v potřebném rozsahu.

3.1 Problém křehkého odvozence

Jak se projevuje softwarový systém (jež je zobecněné dědicím software), který není houževnatý, vyplyne z jednoduchého příkladu [1].
Mějme odvozence Bag a jeho odvozeninu CountingBag:

```
Bag=
 CLASS
   B:BAG OF CHAR
   Init:
    B:=[]
   Add(VAL X:CHAR):
    B:=B+[X]
   AddAll(VAL BS:BAG OF CHAR):
    BEGIN
```

```
CountingBag=
 CLASS zobecněně dědicí od Bag
  N:INT
  Init:
   N:=0; ZSuper.Init
  Add(VAL X:CHAR):
   N:=N+1; ZSuper.Add(X)
  Cardinality(RES R:INT):
   R:=N
```

```
      VAR Y:CHAR                          END CLASS CountingBag.
      WHILE BS<>[] DO
        Y<-BS;
        ZSelf.Add(Y);
        BS:=BS-[Y]
      END WHILE BS<>[]
    END AddAll
  Cardinality(RES R:INT):
    R:=NUMBER(B)
END CLASS Bag
```

kde ZSelf je parametr definovaný v podkapitole 2.5 a ZSuper je parametr představující bezprostředního zobecněného předka (tj. odvozence)

Všimněme si, že odvozovatel je nucen při ladění ověřit, zda CountingBag je náhradyschopný za Bag, a že tedy po odladění lze odvozeninu CountingBag považovat za zdokonaleninu odvozence Bag.

Mějme dále upraveninu Bag' softwarového dílu Bag:

```
Bag'=
 CLASS
   B:BAG OF CHAR
   Init.
     B:=[]
   Add(VAL X:CHAR):
     B:=B+[X]
   AddAll(VAL BS:BAG OF CHAR):
     B:=B+BS
   Cardinality(RES R:INT)
     R:=NUMBER(B)
 END CLASS Bag'.
```

Je zřejmé, že tato upravenina je zdokonaleninou softwarového dílu Bag.

Po dosazení upraveniny Bag' na místo softwarového dílu Bag však dojde k nepříjemnému překvapení - činnost Cardinality odvozeniny CountingBag najednou vrací obecně chybný počet znaků, přestože na odvozenině CountingBag nebyla provedena žádná změna.

Tento problém se v literatuře označuje jako problém křehkého odvozence (doslova: problém křehké bázové třídy), ačkoli by správněji měl být nazýván problémem křehké odvozeniny, nebo ještě výstižněji problémem křehkého odvozencova orientovaného podgrafu využívání zobecněné dědičnosti

3.2 Ukázka odvoditele

Příklad uvedený v předchozí podkapitole může ještě posloužit k bližšímu vysvětlení pojmu odvoditel definovaného v podkapitole 2.7

Odvození odvozeniny CountingBag z odvozence Bag lze totiž vyjádřit rovnicí:

```
        OdvlCountingBag r Bag = CountingBag,
```
kde
  r - odvozovátor,
  OdvlCountingBag - odvoditel.
    Odvoditel OdvlCountingBag má v tomto příkladu tvar:
```
OdvlCountingBag=
  MODIFIER
    N INT
    Init:
      N:=0; ZSuper.Init
    Add(VAL X:CHAR):
      N:=N+1; ZSuper.Add(X)
    Cardinality(RES R:INT):
      R =N
  END MODIFIER OdvlCountingBag.
```

4. JAK SE POTÍŽE ODSTRAŇUJÍ

Předmětem této kapitoly není odstraňování potíží spojených s používáním nedědicího polymorfismu. Houževnatost softwarového systému však lze při používání nedědicího polymorfismu zajistit. Odpovídající a už dnes použitelnou technologií je technologie nedědicí rozšiřitelnosti.

Tato kapitola pojednává o tom, co bylo nalezeno při hledání způsobu, jak odstranit potíže s houževnatostí v případě, kdy se používá zobecněně dědicího polymorfismu.

4.1 Výzkum

Nalezení technologických postupů, které je třeba doplnit do technologie plné rozšiřitelnosti a jejichž uplatněním při tvorbě softwarového systému by se zajistilo, že vytvořený softwarový systém bude houževnatý, je v současné době cílem výzkumu.

Tento výzkum probíhal a probíhá, na různých pracovištích, v různých obdobích a s různými výsledky, v souhrnu už několik let. Hrubou představu o tématech výzkumu mohou poskytnout seznamy použité literatury u položek [1] a [2] ze seznamu literatury tohoto článku.

V průběhu výzkumu se postupně měnily hypotézy o příčině, která vede k tomu, že softwarový systém (když je zobecněně dědicím software) není houževnatý.

Zbývající část této kapitoly 4 vychází z literatury [1] a [2].

4.2 Překonané představy o příčině potíží s houževnatostí

Na první pohled se zdá, že příčina tkví v nedokumentovaných rysech odvozenců a v nesprávném odhadnutí těchto rysů odvozovateli odvozenin. Skutečnost je však složitější.

Podle jiného názoru příčina spočívá v nutnosti opětovného překladu odvozenin po nahrazení odvozence jeho upraveninou. Avšak i v případě použití technologie zajišťující, že opětovný překlad není nutný, problém s houževnatosti zůstává.

Podle další představy je problém způsoben tím, že odvozenina není náhradyschopná za odvozence. Ale ani dokonalá náhradyschopnost odvozeniny za odvozence sama o sobě houževnatost nezabezpečí.

4.3 Současná představa o příčině potíží s houževnatostí

Za příčinu problému lze označit porušení zapouzdřenosti zobecněnou dědičností.

Problém není závislý na použitém programovacím jazyku.

Bylo odhaleno, že softwarový systém (který je zobecněně dědicím software) není houževnatý, když nesplňuje třeba jen jednu z těchto podmínek.

1) Ohebnost všech dvojic odvozenec - odvozenina,
2) Monotónnost odvozovátoru nad všemi odvozenci.

Za příčinu problému křehkého odvozence (křehké bázové třídy) se však považuje už nesplnění podmínky ohebnosti.

Znění obou podmínek se nalézá ve výše uvedené podkapitole 2.14.

4.4 Jaká oblast byla už prozkoumána

Již ukončená etapa výzkumu se zaměřila pouze na podmínku ohebnosti a navíc byla tato zkoumaná oblast dál zmenšena přijetím těchto omezení (nejsou zde uvedena ta omezení, jež vyplývají z definic obsažených v kapitole 2):

1. Zobecněná dědičnost je pouze jednoduchá, ne vícenásobná,
2. Parametry činnosti ani datové položky softwarových dílů nejsou instancemi softwarových dílů, tj. jsou jednoduchých datových typů,
3. Jak odvozenina z odvozence, tak upravenina odvozence mají stejný počet činností jako odvozenec,
4. V náplních činností odvozence se nevyskytují takové příkazy k provedení činností, jež by zaváděly nepřímou rekurzi,
5. Odvoditel nezavádí do odvozeniny nové datové položky.

4.5 Závěry platné pro již prozkoumanou oblast

Po vyjádření problému křehkého odvozence pomocí ohebnosti a po provedení matematických důkazů se ukázalo, že takové postupy tvorby software, které dodržují níže uvedené zásady, jsou dostačující pro vyřešení problému křehkého odvozence, platí-li zároveň omezení popsaná v podkapitole 4.4. Zásady jsou představeny ve formě přikázání

4.5.1 Přikázání pro návrháře odvozenců a jejich upravenin

1) Odvozenině dovolit přístup k datovým položkám odvozence (nebo jeho upraveniny) pouze prostřednictvím činností, u nichž je zaručeno, že budou provedeny výhradně v případě, že příkaz k jejich provedení se nachází v náplni činnosti zobecněného potomka odvozence (nebo zobecněného potomka jeho upraveniny)
2) Vstoupit do oblasti, která je svěřena na starost realizátorům, a v náplni každé činnosti odvozence (nebo jeho upraveniny) vyjmenovat všechny takové činnosti odvozence (nebo jeho upraveniny), jež realizátor v téže náplni použije formou příkazů k provedení činnosti.

4.5.2 Přikázání pro realizátory odvozenců

1) V souladu s prvním přikázáním pro návrháře omezit přístup k datovým položkám, pokud k takovému omezení neposkytuje programovací systém návrhářům potřebné nástroje.
2) Do náplně každé činnosti odvozence umístit příkazy k provedení všech takových činností odvozencem, jež v téže náplni vyjmenoval návrhář odvozence.

4.5.3 Přikázání pro realizátory upravenin odvozenců

1) V náplni každé činnosti upraveniny umístit příkazy k provedení výhradně takových činností upraveninou, jejichž názvy tvoří podmnožinu množiny názvů činností použitých (formou příkazů k provedení činnosti) ve stejnojmenné náplni u odvozence příslušejícího upravenině.
2) Při ověřování, zda upravenina je zdokonaleninou odvozence z hlediska chování způsobeného prováděním každé své činnosti, si v náplni každé činnosti upraveniny představovat namísto příkazů k provedení činnosti upraveninou příkazy k provedení činnosti odvozencem příslušejícím upravenině.

4.5 4 Přikázání pro odvozovatele

1) V náplni každé činnosti odvozeniny umístit příkaz k provedení stejnojmenné činnosti odvozencem příslušejícím odvozenině nebo příkazy k provedení výhradně takových činností odvozeninou nebo jejím odvozencem, jejichž názvy tvoří podmnožinu množiny názvů činností použitých (formou příkazů k provedení činnosti) ve stejnojmenné náplni u odvozence příslušejícího odvozenině.
2) Při ověřování, zda odvozenina je zdokonaleninou odvozence z hlediska chování způsobeného prováděním každé své činnosti, si v náplni každé činnosti odvozeniny představovat namísto příkazů k provedení činnosti odvozeninou příkazy k provedení činností odvozencem příslušejícím odvozenině.
3) V náplních činností odvozeniny měnit hodnoty datových položek jejího odvozence výhradně prostřednictvím příkazů k provedení činností tímto odvozencem.

4.6 Zaměření dalšího výzkumu

Další etapy výzkumu se zaměří na tyto otázky:

1) Stanovení takových zásad tvorby software, jejichž dodržením se dosáhne monotónnosti odvozovátoru nad odvozenci.
2) Promítnutí zásad, jež byly popsány v podkapitole 4.5, do objektově orientovaných programovacích systémů.
3) Zobecnění zásad, které byly popsány v podkapitole 4.5, do takové míry, aby mohla být zrušena omezení vyjmenovaná v podkapitole 4.4,
4) Oslabení tvrdosti zásad popsaných v podkapitole 4.5.

5. CO SE MŮŽE STÁT

Technologie nedědicí rozšiřitelnosti je zatím posledním bodem, kterého dosáhla znovupoužitelnost na své cestě od reklamního sloganu ke skutečnosti.

Budou-li tvůrci software (jak na straně dodavatelů, tak na straně odběratelů) ochotni ustoupit od zobecněně dědicího polymorfismu, může být technologie nedědicí rozšiřitelnosti, která je úplná, a proto ihned použitelná, prohlášena za technologii plné rozšiřitelnosti.

V opačném případě je třeba sledovat pokračující výzkum a čekat na jeho výsledky. Pokud tyto výsledky budou kladné, umožní to doplnit chybějící postupy do technologie plné rozšiřitelnosti a učinit ji tak technologií úplnou a připravenou k použití.

Běžně užívaným způsobem tvorby software se technologie plné rozšiřitelnosti zřejmě stane až poté, co se změní na technologii úplnou.

LITERATURA

[1] Leonid MICHAJLOV, Emil SEKERINSKI: The Fragile Base Class Problem and its Solution, TUCS Technical Report No 117, Turku Centre for Computer Science, Turku, červen 1997, http://www.abo.fi/~lmikhajl/

[2] Leonid MICHAJLOV, Emil SEKERINSKI A Study of the Fragile Base Class Problem, Turku Centre for Computer Science, Turku, McMaster University, Hamilton, listopad 1997, http://www abo.fi/~lmikhajl/

[3] Dick POUNTAIN, Clemens SZYPERSKI. Extensible Software Systems, Byte 5/94, str 57 až 62, McGraw-Hill, New York, květen 1994

[4] Jiří POLÁK, Vojtěch MERUNKA: OOP IV - třídní společnost, Softwarové noviny 6/93, str 114 až 116, Softwarové noviny, Praha, červen 1993

[5] Barbara H. LISKOV, Jeannette M. WING: A Behavioral Notion of Subtyping, MIT Laboratory for Computer Science, Cambridge, Carnegie Mellon University, Pittsburgh, květen 1994, http://www.cs.cmu.edu/afs/cs.cmu.edu/project/venari/papers/subtype-toplas/paper.ps

[6] Vladimír KUCHTA Vytváření počítačových programů pomocí rozšiřitelnosti - technologický standard a nástroje, přihláška do soutěže o standardní grant

u Grantové agentury Akademie věd ČR,Nový Jičín, září 1994
[7] Sara WILLIAMS, Charles KINDEL: The Component Object Model: Technical Overview, Dr. Dobbs Journal, prosinec 1994
http://www.microsoft.com/oledev/olecom/com_modl.htm
[8] IBM's System Object Model (SOM): Making Reuse a Reality, IBM Corporation, Object Technology Products Group,Austin, červenec 1994
http://www.developer.ibm.com/library/ref/SOM_Reuse_Reality_Art.html
[9] Petr KOUBSKÝ: Cesty moderního programování, JZD AK, Slušovice, leden 1990
[10] Hans-Jochen BARTSCH: Matematické vzorce (Taschenbuch mathematischer Formeln), SNTL, Praha, 1983
[11] Jiří DEMEL: Grafy, SNTL, Praha, 1988