Abstraktní vzhledy dat

Vladimír Sklenář, Václav Snášel

1. Úvod

Vytváření nových aplikací spojováním složek uživatelského rozhraní a složek reprezentujících vnitřní model aplikace je relativně nová idea, jejíž rozpracování je v současnosti námětem řady výzkumných prací v oblasti metodologie návrhu a implementace interaktivních softwarových produktů. Velký význam při jejich řešení má jasné oddělení prvků uživatelského rozhraní a složek aplikace. To totiž umožní jejich opakované využití v široké škále aplikací. Má-li být objekt reprezentující některou vnitřní složku aplikace vícenásobně využitelný, nemůže obsahovat nic, co by záviselo na konkrétním uživatelském prostředí a změna vnějšího prostředí by jej tedy neměla nijak ovlivnit. Složky uživatelského prostředí, jejichž úkolem je zajistit prezentaci objektů aplikace uživateli, musí znát stav aplikace(t.j. musí mít zajištěn přístup k datovým položkám aplikačních objektů) a musí být také schopny přenášet příkazy zadané uživatelem např. pomocí myši nebo klávesnice. Kromě oddělení aplikace a uživatelského prostředí je tedy potřebné umožnit vytváření dvojic objekt/vzhled. Jedno z řešení tohoto problému je založeno na pojmu abstraktní vzhledy dat (Abstract Data Views - ADV).

2. Abstraktní datové typy a abstraktní vzhledy dat

Abstraktní datový typ (ADT) je objekt, který je definován svým stavem (t.j množinou datových položek) a svým funkčním rozhraním. Přistupovat ke stavu ADT je možné pouze prostřednictvím funkcí z jeho funkčního rozhraní. U ADT předpokládáme schopnost vkládání, to znamená, že nový ADT může být vytvořen složením jiných ADT do jednoho celku. Přitom se požaduje, aby takto vytvořený ADT znal své složky a byl schopen s nimi manipulovat. O vložených ADT naopak předpokládáme, že neví nic o ADT do kterého jsou vloženy.

Abstraktní vzhledy dat (ADV) můžeme chápat jako vizuální realizaci abstraktního datového typu. Ve své podstatě je ADV objekt určený pro realizování uživatelských aspektů aplikace. Zná tedy svou grafickou reprezentaci na obrazovce počítače a dokáže reagovat na události generované uživatelem (pomocí klávesnice, myši, atd.). ADV graficky reprezentuje některý z objektů aplikace, které samy o sobě nemají žádnou "grafickou reprezentaci". ADV tedy odděluje objekty aplikace od jejich grafického uživatelského rozhraní vytvořením dvojice objekt/vzhled. Toto spojení je realizováno tak, že ADV obsahuje datovou položku owner obsahující odkaz na objekt aplikace, který má reprezentovat. Využití ADV tedy předpokládá, že ADT představují "vnitřní svět" aplikace a neobsahují žádné operace spojené s uživatelským rozhraním. Ty jsou zajišťovány odpovídajícím ADV.

Tak jako u ADT se požaduje, aby ADV mohly vzniknout spojením jiných již existujících ADV. Vložené ADV se pak zobrazuje uvnitř oblasti příslušející ADV do nějž byl vložen. Jedná se tedy o zobecnění pojmu podokno v tom smyslu, že zobrazování může mít volnou formu a není nutně spojeno s obdelníkovým tvarem okna.

Jedna ze zajímavých charakteristik objektově orientovaného programování přirovnává každý objekt k malému počítači a program k síti sestavené z těchto malých počítačů za účelem splnění daného

úkolu. Tuto metaforu lze rozšířit tak, že ADV přirovnáme k malým terminálům individuálně připojeným k těmto malým počítačům.

3. Formální model ADV

Formální model dvojice ADT/ADV může být vyjádřen následujícími operacemi :

Zobrazení :

(ADV_stav x ADT_stav) -> Okno_stav
ADT_chování :
(ADT_stav x ADT_operace) -> ADT_stav
ADV_změna_stavu:
(ADV_stav x ADT_stav x Sys_udalost) -> ADV_stav
ADV_přenos_události :
(ADV_stav x ADT_stav x Sys_událost) -> ADT_operace
ADV_změna_okolí :
(ADV_stav x ADT_stav x Sys_událost)
-> Sys_událostx ADV_elementy
invarianty : (ADV_stav x ADT_stav) -> boolean

kde
ADV_stav a ADT_stav - vyjadřují stav ADV, resp. ADT (t.j. možné hodnoty , kterých mohou nabývat jejich instance)

Okno_stav vyjadřuje možné stavy a zobrazení aktuálního okna

ADT_operace definuje množinu přípustných operací, které mohou být provedeny s daným ADT.

ADV_elementy je kolekce všech instancí aktuálně definovaných ADV v aplikaci.

Sys_událost definuje vstupní události, které může uživatel vyvolat při interakci s uživatelským prostředím.

Definice operace Zobrazení tedy vyjadřuje, že vzhled aktivního okna závisí jak na stavu ADV, tak na stavu ADT. Definice druhé operace, ADT_chování, je uvedena především pro zdůraznění skutečnosti, že operace definované na abstraktním datovém typu mohou změnit pouze jeho stav a nic jiného. Jinak řečeno nemohou změnit např. stav aktuálního okna. Definice operace ADV_změna_stavu říká, že nový stav ADV může být ovlivněn jeho předchozím stavem, stavem abstraktního datového typu a vstupní operací. ADV_přenos_události říká, že vstupní událost, která vyžaduje interakci s aktuálním stavem abstraktního datového typu se transformuje na operaci obsaženou ve funkčním rozhraní ADT. ADV tedy může měnit stav ADT pouze prostřednictvím jeho přípustných operací. Ovlivní-li vstupní operace pouze stav ADV, pak se žádná operace na ADT neprovede. ADV_změna_okolí vyjadřuje skutečnost, že vstupní událost v jednom oknu může vyvolat požadavek na změnu dalších oken. A konečně operace invarianty vyjadřuje, že ne všechny možné kombinace stavů ADT a ADV jsou přípustné. Korektní reprezentace ADT je zajištěna pouze v případě, že všechny podmínky jsou splněny.

4. Programátorský přístup k ADV

ADV je vlastně speciální případ ADT, to znamená, že musí mít svůj vlastní stav reprezentovaný datovými položkami a množinu operací, které je schopné vykonávat. V případě ADV se tyto operace kryjí s reakcemi na vstupní události. ADV by pak mohlo vyjádřeno např. následující pseudokonstrukcí

```
ADV jméno [ For Type jméno_adt ]
  jmeno_prom1 : Typ1, jméno_prom2 : Typ2, ...

  ADV jmeno1 [ For Type jméno_adt ]

    .
    .
    .

  End jmeno1

  EVENT událost1 (param)
  EVENT událost2 (param)

    .
    .
    .
  EVENT Display ()

End jméno
```

V tomto zápisu klíčove slovo For Type vyjadřuje vazbu mezi ADV a ADT. S jeho uvedením souvisí také zavedení proměnné owner, pomocí níž lze v metodách přistupovat k operacím spojeným s ADT při dodržení principu ukrývání informace. Na příkladu je také ukázána možnost vkládání jednotlivých ADV do sebe.

Podrobněji můžeme použití ADV ilustrovat na příkladu šachové hry. Deklarace základních abstraktních datových typů a jim odpovídajícím abstraktním datovým vzhledům by ve zjednodušené formě mohly mít následující tvar:

```
Type Sachovnice
   obsazena_pole : seznam
   bily_na_tahu : boolean

   Type Figura

      druh : {kral, dama, ... }
      barva : {cerny, bily}
      pozice : Pozice

      Function TAH (nova_pozice:Pozice)

        if (korektniTah (pozice,obsazena_pole,
```

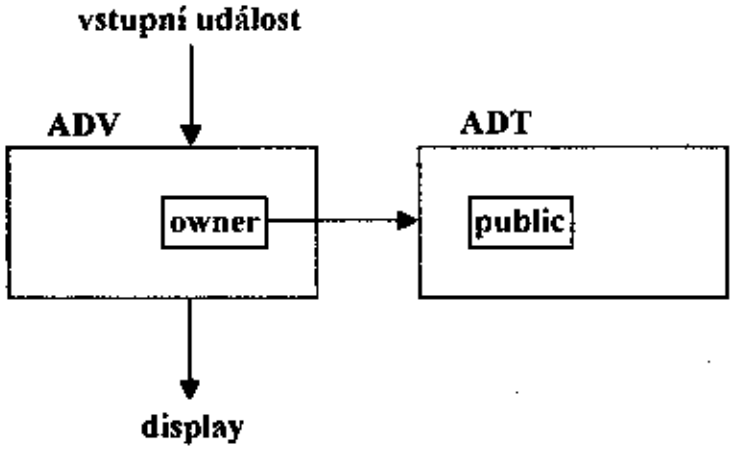
```
                bily_na_tahu, nova_pozice) )
            then pozice = nova_pozice
              aktualizace seznamu obsazena_pole
              bily_na_tahu = not bily_na_tahu
              vrátit informaci o tom, zda byla odebrána
                figura
            else
              vratit chyba


        Function POZICE()
          vrátí hodnotu proměnné pozice

    End Figura

    Function KDO_NA (p:Pozice)
        je-li pozice p obsazena vrátí příslušnou figuru,
          jinak vrátí hodnotu neobsazeno

End Sachovnice

ADV Vzhled_Sachovnice For Type Sachovnice

    delka : integer

    ADV Vzhled_Figury For Type Figura

        umístění : Bod
        chycena : boolean

        EVENT MYS_STISK ()
            chycena = true

        EVENT MYS_PRESUN (p:Bod)
            if (chycena) then umístění = p

        EVENT MYS_UVOLNENI (p:Bod)
            chycena = false
            if (owner.TAH (ZjistiPozici(p)) == chyba)
              then umístění = ZjistiBod(owner.POZICE())

        EVENT DISPLAY ()
            vykresli figurku na obrazovce

    End VzhledFigury

    EVENT DISPLAY ()
        vykresli se šachovnice

End Vzhled_Sachovnice
```

Programovací jazyk, který by přímo podporoval práci s abstraktními vzhledy dat by tedy měl obsahovat některé nové syntaktické konstrukce (For Type, owner, ...). Hlavní myšlenky lze však pomerně bez problémů implementovat i ve stávajících programovacích jazycích, např. C++, Smalltalk.

5. Architektura aplikace využívající ADV

Typický systém využívající ADV se skládá z :

- kolekce abstraktních datových typů, které vyjadřují vnitřní stav aplikace
- kolekce abstraktních datových vzhledů zajišťujících vnější zobrazování a ošetření vstupních událostí



obr.1 model architektury ADV

S jednou instancí ADV může být spojena právě jedna instance ADT. Naopak jedna instance ADT může být zobrazována několika různými způsoby. Například číslo může být na obrazovce reprezentováno přímo zápisem hodnoty, polohou ukazatele na stupnici, velikostí sloupce v grafu atd. Jinak řečeno, jedna instance ADT může být vlastněna několika různými instancemi ADV. Tím je zajištěno,že daná instance abstraktního datového typu může být korektně zobrazována několika různými způsoby. Z pohledu ADT není podstatné, které ADV s ním v daném okamžiku manipuluje.

Řízení chodu celého systému je umístěno v uživatelské části aplikace, protože je určeno především akcemi uživatele. Komunikace mezi ADT a ADV je založena na sychronním volání procedur. ADT nikdy negeneruje události, které musí být asynchronně ošetřovány odpovídajícími ADV.

6. Implementace

Koncept abstraktních vzhledů dat je implementačně nezávislý. Při implementaci pomocí jazyka C++ mohou být třídy reprezentující abstraktní datové typy odvozeny od abstraktní třídy Interactive, která obsahuje seznam odkazů na instance ADV objektů. Třída ADV obsahuje minimálně odkaz na

objekt třídy Interactive (proměnná owner) a charakteristiky okna, do nějž se zobrazuje. Při svém vytváření je ADV objekt automaticky vložen do seznamu u příslušného objektu třídy Interactive. Pomocí proměnné owner může každý ADV objekt manipulovat s připojeným objektem třídy Interactive (ADT) a realizovat tak požadavky přicházející od uživatele. Je-li objekt třídy Interactive změněn vyšle zprávu všem souvisejícím ADV s požadavkem, aby na tutu změnu odpovídajícím způsobem reagovaly. Tím je zajištěno promítnutí změn do všech vzhledů, které objekt reprezentují na obrazovce. Analogický postup podporuje také jazyk Smalltalk-80 (metoda model-view-controller).

Literatura :

D.Cowan, R.Ierusalimschy, C.Lucena, T.Stepien. Abstract Data Views. Structured Programming, 14(1), January 1993

Autor :

RNDr. Vladimír Sklenář, RNDr. Václav Snášel CSc.
Katedra matematické informatiky,
Přírodovědecká fakulta UP Olomouc
Tomkova 38,
Olomouc-Hejčín

tel. (068) 412210