Principy tvorby objektů v komponentním frameworku BCF

Michal Brožek

Katedra měřicí a řídicí techniky, VŠB – Technická univerzita Ostrava, FEI, tř. 17. Listopadu, 708 33, Ostrava – Poruba, Česká republika.

Abstrakt

Komponentní framework je pro většinu programátorů novým konceptem. Jednotné používání návrhových modelů v BCF (BlackBox Component Framework) značně přispívá k pochopení frameworku. V příspěvku budou čtenáři seznámeni s konkrétními postupy používání a vytváření nových objektů na základě návrhových vzorů v komponentním frameworku BCF. Tyto postupy vycházejí z principů komponentního programování v BCF. Jednoduché názorné ukázky konkrétního použití návrhových vzorů budou psány v implementačním jazyce Component Pascal. Protože BCF je zaměřen na zpracování složených dokumentů, i když nejenom na ně, budou pro demonstraci použity příklady zaměřené na zpracování dokumentů.

Příspěvek vznikl na základě práce na projektu GAČR „Vývoj programových a technických prostředků pro přenositelné a bezpečné aplikace řízení procesů“ číslo projektu 102/97/0542. A projektu FRVŠ „Techniky komponentního programování v oblasti návrhu a implementace řídicích systémů“ číslo projektu 0655/98.

1. Úvod

Protože je Component Pascal velmi podobný Pascalu a ještě více Module, mohou programující v těchto jazycích velmi rychle proniknout do problematiky programování v BCF. Flexibilita Component Pascalu umožňuje psát jak jednoduché programy, tak složité aplikace orientované na zpracování dokumentů v tomtéž jazyce.

Vyspělý typový a modulární systém BCF umožňuje vyjádřit návrhové aspekty programu. Tím je umožněna i kontrola kompilátorem. Touto cestou je tedy možno považovat jazyk nejenom za prostředek implementace, ale i za prostředek specifikace. Na rozdíl od C++ nebo Smalltalku lze přímo v Component Pascalu vyjádřit řadu důležitých aspektů softwarové architektury.

BCF je zaměřen na aplikace založené na dokumentech. Jak již jméno samotné napovídá, je BCF založen na principech abstrakce typu black-box. Framework se skládá z jádra, které je tvořeno vrstvenými moduly, a otevřené množiny sub-systémů.

Každý sub-systém je sám o sobě množinou vrstvených modulů. BCF se zaměřuje na vizuální komponenty – což je flexibilní koncept, jak se ukázalo v případech VB, OLE, ActiveX prvků nebo JavaBeans. Základním kamenem vizuálních komponent je to, jak vypadají a interakce se svým obsahem a obsaženými komponentami. Hlavní abstrakcí v BCF je proto obraz (*„view")*. V BCF může být obraz plně soběstačný, tj. obsahuje vlastní zabudovaný model a ovladač (editor). Pro více komplexní obraz, jako jsou třeba kontejnerové obrazy, je použito řádné rozdělení na modely, obrazy a ovladače k dosažení zvýšené možnosti konfigurace a zvládnutí komplexnosti.

2. Objekty v BCF

Pro pochopení dále uvedených příkladů je nutné objasnit způsob, jak jsou v BCF objekty definovány a jaké metody je možné vytvářet. Component Pascal používá jednotnou konstrukci pro označení jak rozhraní objektů, tak i jejich implementací – *záznamy*. Toto sjednocení dovoluje fixovat některé implementační aspekty rozhraní, zatímco jiné mohou zůstat otevřené. Tato flexibilita je v komplexním frameworku často žádoucí. Je důležité sdělit architektonická rozhodnutí co nejpřesněji. Z tohoto důvodu může být záznam v Component Pascalu ohodnocen atributy tak, aby rozhraní explicitně zahrnovalo základní architektonická rozhodnutí. Je výhodou, že kompilátor umožňuje ověřování těchto rozhodnutí. Tyto atributy jsou EXTENSIBLE, ABSTRACT a LIMITED. Tyto atributy umožňují rozlišovat čtyři možné kombinace rozšíření a alokace:

Tab. 1: Atributy architektury záznamu

Atribut	Rozšiřitelnost	Alokace	Přiřazení záznamu
Bez ("FINAL")	NE	ANO	ANO
EXTENSIBLE	ANO	ANO	NE
ABSTRACT	ANO	NE	NE
LIMITED	NE*	NE*	NE

*kromě definujícího modulu

Typ záznam je struktura, která je složena z neměnného počtu elementů, nazývaných položky, které mohou být různých typů. Deklarace typu záznam specifikuje jméno a typ každé položky. Každá položka může být exportována, tedy zpřístupněna. Component Pascal umožňuje kromě standardního exportu navíc tzv. „read-only" export (obsah proměnné, nebo položky záznamu může být pouze přečten, ale ne změněn). V rámci definujícího modulu jsou všechny položky veřejné a je možné je číst i změnit. Atributy jsou tedy uplatněny pouze za hranicemi definujícího modulu.

Tab. 2: Atributy viditelnosti

Atribut	Význam
bez označení	Soukromá položka
mínus –	Export – pouze pro čtení
Hvězdička *	Export – čtení i změna

Pozn.: Atributy jsou používány pro proměnné, konstanty, typy, procedury i metody.

Typy záznamů označené jako ABSTRACT nebo EXTENSIBLE jsou rozšiřitelné, neboli mohou být použity jako základ pro rozšiřující záznam. Uveďme následující příklad:
T0 = EXTENSIBLE RECORD x: INTEGER END
T1 = RECORD (T0) y: REAL END

V tomto příkladu je T1 rozšířením T0 a T0 je základním typem pro T1. Rozšiřující typ T1 se skládá z položek svého základního typu a z položek, které sám deklaruje. Všechny identifikátory, které jsou nově zavedeny, musí být jiné než ty, které byly deklarovány v původním, základním záznamu (záznamech). Existují další pravidla, která zaručují zachování neměnnosti architektonických rozhodnutí, a která jsou kontrolována kompilátorem. Základním typem abstraktního záznamu musí být abstraktní záznam. Záznam, který rozšiřuje skrytý (neexportovaný) záznam, nemůže být exportován. Jakýkoliv záznam je automaticky rozšířením předdeklarovaného záznamu ANYREC. ANYREC neobsahuje žádné položky.

Příklady definic záznamů:

```
Tree = POINTER TO Node
Node = EXTENSIBLE RECORD
    Key : INTEGER;
    Left, right: Tree
END
```

```
CenterTree = POINTER TO CenterNode
CenterNode = RECORD (Node)
        Width: INTEGER;
        subnode: Tree
END
```

```
Object=POINTER TO ABSTRACT RECORD
END;
```

2.1 Metody

Globálně deklarované procedury mohou být ve stejném modulu asociovány se záznamem. Říkáme, že metody jsou svázány s typem záznam. Vazba je vyjádřena typem „příjemce“ v hlavičce deklarace metody. Příjemce může být parametr (VAR nebo IN) nebo odkaz. Příkladem hlavičky deklarace metody svázané se záznamem T0 může být následující:

PROCEDURE (VAR t: T0) Insert (a: INTEGER), NEW;

V tomto případě je příjemcem proměnná t typu T0. To znamená, že metoda Insert je svázaná se záznamem T0. Jestliže je metoda M svázána s typem T0, pak je

explicitně svázána s jakýmkoliv typem T1, který je rozšířením T0. Jestliže je metoda M` (se stejným názvem jako M) deklarována a svázána s T1, pak dojde k přepsání vazby a nová metoda M` je redefinicí původní M pro záznam T1. Formální parametry M a M` musí být shodné. Výjimkou je případ, kdy M je funkce vracející ukazatel. V tomto případě musí být návratový typ (výsledek) funkce M` rozšířením návratového typu M.

Tab. 3: Atributy metod

Atribut	Význam
NEW	Poprvé definovaná metoda
ABSTRACT	Abstraktní metoda
EMPTY	Prázdná – neimplementovaná metoda
EXTENSIBLE	Rozšiřitelná metoda

Atribut NEW musí být použit u všech nově deklarovaných metod a nesmí být použit u předefinování metod. Tento atribut pomáhá detekovat nekonzistence mezi záznamem a jeho rozšířením, pokud jeden z nich je změněn, aniž by byl aktualizovaný druhý.

Abstraktní (ABSTRACT) a prázdné (EMPTY) metody jsou tvořeny pouze hlavičkou procedury. Abstraktní metody nejsou nikdy volány. Záznam, který obsahuje abstraktní metody, musí být rovněž abstraktní. Volání prázdné (EMPTY) metody nemá žádný efekt.

U metod označených symbolem "–" je exportována pouze implementace. Takovéto metody mohou být sice předefinovány, ale jejich volání je možné jen z modulu, kde je provedena implementace.

Příklady metod:

```
PROCEDURE (obj: Object) Draw (w: Window), NEW, ABSTRACT (* nová abstraktní
metoda *)

PROCEDURE (obj: Object) Notify (e: Event), NEW, EMPTY (* nová prázdná metoda
*)

PROCEDURE (t: Tree) Insert (node: Tree), NEW, EXTENSIBLE;
        VAR p, father: Tree;
BEGIN
        (* zde je nějaký implementující kód*)
END Insert

PROCEDURE (t: CenterTree) Insert (node: Tree);  (* redefinice *)
BEGIN
        (* nová implementace *)
```

```
        t.Insert^ (node)  (* volá metodu Insert záznamu Tree – super call*)
END Insert
```

3. Návrhové vzory

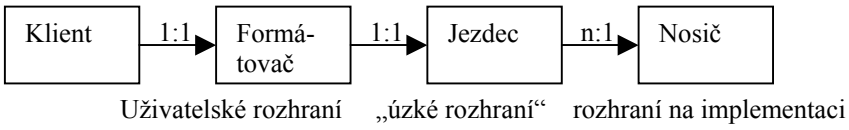
Architektura BCF je založena na velkém množství nových vzorů a postupů. Některé z nich jsou specifické pouze pro BlackBox. V této kapitole se budeme zabývat nejčastěji používanými návrhovými vzory v BCF: carrier-rider-mapper separace (CRM separace), directory objekty a HMVC.

3.1 Nosič-Jezdec-Formátovač návrhový vzor

Nosič-Jezdec-Formátovač (CRM) návrhový vzor je v BCF takřka všudypřítomný. Tím, že je sjednocujícím prvkem v celém BCF, přispívá k porozumitelnosti frameworku. Klíčová idea je oddělení objektů, které nesou data (Nosič), přístupové cesty k datům v těchto objektech (Jezdec) a filtrům, které formátují data (Formátovače).

Nosič udržuje data, která jsou přístupná logicky podle pozice. Abstraktní rozhraní Nosiče otevírá cestu pro rozšiřitelnost: mnoho konkrétních implementací může implementovat právě toto abstraktní rozhraní. Jezdec zapouzdřuje přístupovou cestu k datům na dané pozici. Nosič vytvoří svého Jezdce a tím má Nosič privilegovaný přístup k implementaci Jezdce. Proto Jezdec může efektivně udržovat specifický přístup k Nosiči. Rozdělení na Nosiče a Jezdce je obdobné jako návrhový vzor Iterátor [1][2] (viz. dále). Klienti používají kombinaci přímého rozhraní Nosiče a poskytovaného rozhraní Jezdce pro přístup k Nosiči.

Společně rozhraní Nosiče a Jezdce tvoří „úzké hrdlo“, které ruší vazbu klienta od potenciálně mnoha implementací Nosiče. Formátovače jsou používány pro poskytnutí rozhraní, které je mnohem vhodnější pro určité klienty, jimž nevyhovuje „surové“ rozhraní, které poskytuje Nosič a Jezdec.



Obr. 1 Nosič-Jezdec-Formátovač

3.2 Directory objekty

BlackBox abstrakce v BCF je přivedena do takového extrému, že ani jména abstraktních rozhraní nejsou veřejná. Použití příkazu NEW (součást jazyka) je předepsáno tak, že vyžaduje jméno třídy. Namísto toho jsou nové objekty vytvářeny použitím factory („továrních“) metod nebo objektů[2]. Factory objekty, které jsou v BCF nazývány directory objekty, jsou použity, když je potřeba vytvořit nový objekt.

Každý modul, který zavádí novou abstrakci, zároveň poskytuje konfigurovatelný directory objekt, a pokud ten není k dispozici, je nahrazen standardním directory objektem. Použití je zřejmé z následujícího jednoduchého demonstračního příkladu:

```
Directory = POINTER TO ABSTRACT RECORD END;
VAR dir- : Directory;

PROCEDURE (d : Directory) New* (): Docu, NEW;
VAR doc : Docu;
BEGIN
        NEW(doc);
        (* zde jsou možné počáteční inicializace *)
        RETURN doc (* vrací nově vytvořený objekt *)
END New;
…
VAR Dok1 : Docu;
BEGIN
        Dok1 := dir.New();
```

3.3 Hierarchický Model-Obraz-Ovladač (HMVC)

Původní model MVC je nepoužitelný pro kombinované dokumenty, a proto BCF definuje model, který takovéto dokumenty podporuje [3]. Model je datová struktura, která uchovává data. Příkladem modelu v BlackBoxu je textový model. Text se skládá nejenom z pole znaků, ale obsahuje také font, velikost měřenou obvykle v bodech, a různorodé atributy, jako je podtržené, tlusté, italické atd. Všechny tyto informace musí být uloženy spolu s každým znakem v modelu. Obraz je vizuální reprezentace modelu v okně. V případě textového modelu jsou v okně znázorněny znaky. V závislosti na atributech uložených v textovém modelu budou znaky zobrazovány na obrazovce v daném fontu, budou normální nebo podtržené atd. Už teď je zřejmé, z významu modelu a obrazu, že obraz nebude existovat bez modelu. Protože obraz je vizuální zobrazení modelu na výstupním zařízení (obrazovce), musí mít „něco“, co zobrazí. Z pohledu programování musí být vytvořen nejprve model a teprve následovně je možné jej zobrazit pomocí obrazu. Ovladač je objekt, který řídí vzájemné působení mezi uživatelem a oknem (tedy obrazem), a manipuluje tak s modelem [4]. V příkladu textového modelu uživatel může chtít vložit slovo do textu na určitou pozici ve větě. Uživatel umístí kurzor pomocí myši do místa vkládání, klikne myší a pak začne psát slovo. Ovladač mění tvar kurzoru při pohybu nad textem a při kliknutí myši umístí kurzor na danou pozici. Ovladač je objekt, který řídí vzájemnou interakci mezi uživatelem a obrazem tak, aby měnil model. Primárním návrhovým konceptem MVC je oddělení obrazu od modelu. V HMVC může model obsahovat vnořené obrazy, pro které samozřejmě platí vše řečené.

3.4 Iterátory

Model, obraz a ovladač jsou všechno objekty. Kromě nich je v BlackBoxu ještě jeden druh objektu, který je nutný pro I/O operace v oknech – iterátory. Většina datových struktur je skladovými kontejnery pro sběr hodnot. Iterátor pro datovou strukturu je

objekt, který pracuje na kolekci dat. Iterátory v BlackBoxu jsou nezbytné pro I/O operace oken a jsou navrženy pro práci s hodnotami znaků, uložených v textovém modelu. V BlackBoxu jsou dva druhy iterátoru pro text – jeden pro vkládání textu do modelu, nazývaný formátovač, a jeden pro čtení hodnot z modelu, nazývaný skener. Formátovače jsou používány pro výstup z oken a skenery pro vstup do oken. Každý z těchto iterátorů má vztah na textový model a nemá žádný vztah na obraz. Následující jednoduchý demonstrační příklad ukazuje vztah mezi modelem a obrazem v použití spolu s iterátorem.

```
PROCEDURE (doc : Docu) OpenDocu* , NEW;
VAR
  md : TextModels.Model;  (* model *)
  vw : TextViews.View;  (* obraz *)
  fm : TextMappers.Formatter;  (* iterátor  formátovač *)
BEGIN
        md := TextModels.dir.New();  (* directory objekt pro model *)
        fm.ConnectTo(md);  (* připojení iterátoru na model *)
        fm.WriteString("Ahoj:");
        fm.WriteTab;
        fm.WriteInt(ab.a);
        fm.WriteTab;
        fm.WriteInt(ab.b);
        fm.WriteLn;
        vw := TextViews.dir.New(md);  (* directory objekt pro obraz – napojení na
model*)
        Views.OpenView(vw);
END Print;
```

4. Závěr

Nosič-Jezdec-Formátovač návrhový vzor je v BCF takřka všudypřítomný. Tím, že je sjednocujícím prvkem v celém BCF, přispívá k porozumitelnosti frameworku. Klíčová idea je oddělení objektů, které nesou data (Nosič), přístupové cesty k datům v těchto objektech (Jezdec) a filtrům, které formátují data (Formátovače).

Component Pascal používá jednotnou konstrukci pro označení jak rozhraní objektů, tak i jejich implementací: záznamy. Toto sjednocení dovoluje fixovat některé implementační aspekty rozhraní, zatímco jiné mohou zůstat otevřené. Tato flexibilita je v komplexním frameworku často žádoucí.

Silným konceptem jsou abstraktní rozhraní a directory objekty. Kombinací obou lze vytvářet abstraktní rozhraní komponent s konkrétní i když ne viditelnou implementací, přístupnou právě pomocí těchto directory objektů. Tento koncept rovněž vychází z principů CMR separace.

Literatura:

[1] Gamma, E., Helm, R., Johnson, R.,Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wasley, 1995.
[2] Warford, J., Programming in BlackBox –The PBox Project. ftp://ftp.pepperdine.edu/pub/compsci/prog-bbox
[3] Szyperski, C., Component Software – Beyond Object-Oriented Programming. Reading, MA: Addison-Wesley, 1998.
[4] BlackBox Component Builder Documentation. Switzerland, Oberon microsystems, Inc., 1997.