Návrh programu v Black Box Component Builderu s využitím architektury Model – View – Controller

Gustav Hrudka

Katedra měřicí a řídicí techniky, VŠB – Technická univerzita v Ostravě, tř. 17. listopadu, 708 33 Ostrava-Poruba, Česká republika

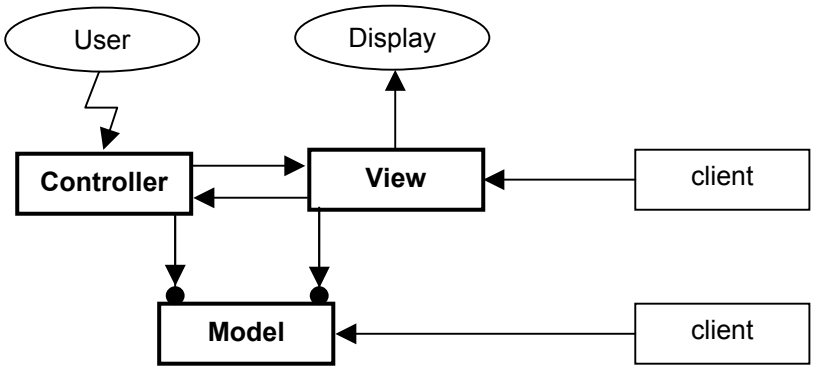
Abstrakt

BlackBox Component Builder představuje produkt švýcarské firmy Oberon microsystems, Inc. Technopark Zürich, CH-8005 Zürich Switzerland. Svým pojetím navazuje na předchozí verze, dříve pod značením Oberon. BlackBox Component Builder (BBCB) představuje modulárně zaměřený a objektově orientovaný programovací prostředek pracující na základě tzv. frameworku. Svou architekturou podporuje vytváření a následnou kompozici komponent.

Následující příspěvek se soustřeďuje na základní principy strukturalizace a separace objektových struktur do tříd "model, view, controller" a jejich využití při tvorbě programového projektu v tomto prostředí.

1. Strukturalizace Model-View-Controller využitá v architektuře framework

Prostředek pracující na principu tzv. *framework* lze definovat jako množinu kooperujících tříd, z nichž některé mohou být abstraktní, a které poskytují *znovupoužitelný* a *modifikovatelný návrh* řešící specifickou třídu softwarových problémových úloh. Jeho hlavní předností je již existující počáteční implementace některých základních funkcí a možnost jejich rozšíření prostou úpravou nebo doplněním odpovídajících komponent frameworku. Jedním z nejčastěji používaných vzorů pro formulaci frameworku je tzv. *MVC framework* (Model-View-Controller Framework). Jeho použitelnost má výhody především v jasné specifikaci rolí, které mají vykonávat dílčí úseky kódu, a v jejich intuitivním rozdělení. *"Model"* reprezentuje a zároveň zapouzdřuje vlastní informaci. *"View"* prezentuje vnější pohled na informaci (např. její grafickou podobu) a *"controller"* zahrnuje vše týkající se uživatelské manipulace s touto informací. Na základě této techniky je možné stejnou informaci prezentovat mnoha odlišnými způsoby. Ze vztahu mezi "view" a "controller" vyplývá, že nejvhodnějším přístupem bude jejich poměr 1:1, tedy každé "view" bude obsluhováno právě jedním "controllerem". Malý počet tříd (model, view, controller) budí zdání spíše návrhového vzoru namísto frameworku. Důležité je však zdůraznit netriviální vazby mezi "view" a "controller", jejichž interakce výrazně závisí na povaze vlastního "view".



Obr. 1. MVC framework

Výše uvedený obrázek znázorňuje, jak jsou uživatelem vyvolané události zpracovávány částí "controller" a jakým způsobem "view" přistupuje k jejich přímému zobrazení prostřednictvím displeje. Důležitá je rovněž skutečnost, že libovolná třída využívající těchto služeb – tzv. *klient* má možnost přímého přístupu ke všem těmto částím (model, view, controller).

2. Uplatněné návrhové techniky

Framework konkretizuje a integruje celou řadu vzorů k dosažení vhodného prokládání a interakce jednotlivých částí. Tak poskytuje vyšší úroveň architektury, ale i infrastruktury, která integruje abstrakce často používané v návrhových vzorech. Stupeň prosazení jednotlivých prostředků je závislý na typu technologie, kterou framework využívá. Jeho návrh může tedy probíhat technikou zdola nahoru řízenou návrhovými vzory nebo může postupovat i shora dolů k specifikovanému cíli. První případ se jeví vhodným tam, kde ještě není v plné míře pochopena doména frameworku, ale je dobře zvládnutá cílová část. *Doménu frameworku* tvoří množina pravidel, úloh a jejich sémantických modelů již zde zahrnutých. *Cílovou doménou* je množina interakcí a entit nalezených v analýze. Důležité je tyto domény od sebe vzájemně odlišit. Doména frameworku je technicky svou povahou introvertní, zatímco doména cílová (řešení) je aplikačně orientována a je povahově extrovertní.

Použití této architektury je namístě především v případě řešení problému blízkého již existující rodiny řešení poskytovaných frameworkem. Způsob separace a vyhrazení interakcí mezi "model-view-controller" je stěžejním a velmi diskutovaným problémem. Jedním z nabízených řešení je potom využití některé z technik objektově orientované analýzy a návrhu. Vhodné je i uvážit, zda tato technika nepovede k vyloučení některých významných prvků cílové podoby a snížení flexibility nebo zda přílišná flexibilita nepovede k nevhodným technikám.

3. Příklad využití architektury MVC Framework

Koncepce *BlackBox Component Builderu* (dále jen BBCB) vychází ze Smalltalku a je navržena pro objektově orientované programování s využitím komponent. Tato skutečnost představuje jisté omezení a zvýšení uniformity programování.

Dále budeme uvažovat již konkrétní případ tvorby grafického rozhraní pro prezentování dat v podobě grafů. Tento příklad byl zvolen pro jednoduchou a intuitivní separaci jednotlivých prvků do tříd "model, view controller". Datovou část, která bude zapouzdřena v "modelu" představují veškerá prezentovaná data spolu se svými vlastnostmi. Do "view" bude integrováno vše co bude souviset s jejich grafickou podobou. Na základě této úvahy již lze provést následující definici typů:

```
TYPE GraphModel* = POINTER TO ABSTRACT RECORD (Models.Model)
TYPE GraphView* = POINTER TO ABSTRACT RECORD (Views.View)
```

Na tomto základě se vytvoří definice dvou typů – tříd zatím abstraktně definovaných jako potomci nadtříd poskytovaných vlastním frameworkem. Jejich definice prostřednictvím ukazatele vyplývá z objektově orientovaného chápání a následné možnosti využití virtuality a polymorfismu. Např. budeme-li dále v programu pracovat s přiřazením ":=" nemusí být na pravé straně instance stejné třídy (typu) jako na straně levé, ale může zde být i instance její podtřídy (tedy potomek).

Dalším návrhovým krokem může být návrh metod, jejichž výhradním prostřednictvím lze s danými prvky komunikovat. V komponentně orientovaném prostředí BBCB se mohou tyto metody stát součástí rozhraní, prostřednictvím kterého může vnější prostředí s komponentou manipulovat.
Pro uváděný případ uvažujme pro jednoduchost pouze následující metody:
PROCEDURE (m: GraphModel) SetData* (data: Data), NEW, ABSTRACT;
PROCEDURE (v: GraphView) SetTitle* (title: String), NEW, ABSTRACT;
PROCEDURE (v: GraphView) SetParams* (p: Params), NEW, ABSTRACT;
Řekněme, že první metoda "SetData" nastaví hodnoty všech prvků "modelu", metoda "SetTitle" nastaví nadpis grafu (zahrnuto jako vlastnost view). Metodou "SetParams" mohou být nastaveny další vlastnosti grafu (tedy view) jako např. barva čar, os, zobrazení rastru atd. Atribut "NEW" používá BBCB pro všechny metody, které nejsou součástí definice nadtřídy. Jedním z jeho významů může být i rozšíření rozhraní.

V tomto stadiu již máme připraveny objekty pro zapouzdření dat ("GraphModel"), jejich grafickou prezentaci ("GraphView") a samozřejmě i metody pro legitimní změnu jejich obsahu.

Nyní je však zapotřebí naplnit abstrakce konkrétní definicí. To platí nejen pro námi abstraktně nadefinované třídy a metody, ale samozřejmě i pro třídy a metody abstraktně definované u nadtřídy, kterou využíváme. Abstrakce definované již ve frameworku představují jakousi část rozhraní, kterou musí uživatel naplnit svou konkrétní představou.

Nejdříve proveďme definici konkrétní podoby našich tříd:

```
Model = POINTER TO RECORD (GraphModel)
        values: Value;   (* např. naměřené hodnoty *)
        smper: INTEGER;(* perioda vzorkování *)
        starttime: TIME; (* čas prvního vzorku *)
        unit: String; (* použitá jednotka *)
END;
View = POINTER TO RECORD (GraphView)
        model: Model;
        (* zde bude připojen model, který view prezentuje *)
        title: String; (* nadpis *)
        grid: BOOLEAN;          (* rastr *)
        autoscale: BOOLEAN; (* automatická změna měřítka *)
        min, max: INTEGER; (* manuálně nastavené extrémy *)
END;
```

Pozn. Definice nestandardních typů jako např. Value, TIME, String zde není uvedena a ponechává se na uvážení pro konkrétní případ realizace.
Dále, jak již bylo řečeno, je zapotřebí provést konkrétní definici všech abstraktních metod nadtříd a připojit se tak na vlastní framework.

3.1 Základní metody pro využití služeb MVC Frameworku

Ve volání metod je užit princip inverzního programování ve smyslu tzv. "up-call". Většina interakcí je zde uskutečněna tak, že systém-framework volá při

specifikovaných událostech metody, které jsou naplněny uživatelským kódem. Dále jsou uvedeny pro předchozí demonstrovaný příklad ty nejdůležitější.

Výraz uvedený před identifikátorem metody stanovuje, zda metoda operuje na "modelu", "view" nebo "controlleru".
- PROCEDURE (v: View) Restore (f: Views.Frame; l,t,r,b: INTEGER);

Tělo této metody by mělo zahrnovat vše co se týká vykreslení "view". Framework tuto metodu volá při různých událostech. Jde především o požadavky aktualizace generované uživatelem nebo i vlastním frameworkem při minimalizaci a následné maximalizaci okna apod. Parametr "f", který je v konečné podobě objektem dědícím z třídy "Ports.Frame", umožňuje vykreslování základních grafických prvků: kružnic, úseček, oblouků, Bezzierových křivek apod.

Zatím nediskutovanou částí je třída "Controller", která specifikuje komunikaci s uživatelem prostřednictvím událostí a z hlediska programu ji lze chápat i jako jakýsi background proces. Zde musí být implementováno vše, co se má dít při uživatelských akcích typu: stisk klávesy, pohyb myši, "double click" …atd. Počet a druhy rozeznatelných akcí vyplývají z platformy, na které je BBCB provozován. Pro předchozí případ grafické prezentace dat by v této části bylo definováno např. co se má stát při stisku šipek (např. pohyb kurzoru grafem s odečítáním hodnot), "doubleclicku" (např. výpis informací o daném grafu) apod.

Stěžejní metodou pro obsluhu výše popsaných událostí je:
- PROCEDURE (v: View) HandleCtrlMsg (f: Views.Frame; VAR msg: Controllers.Message; VAR focus: Views.View);

Zde je nejdůležitější částí parametr "msg", který framework naplní událostí, která se stala. Tak lze zacházet s jednotlivými dílčími případy jako s variabilním záznamem jehož struktura složek závisí na typu události. Jde např. o následující události:
PollFocusMsg, ScrollMsg, EditMsg, ReplaceViewMsg atd.

Samostatnou část prostředků frameworku tvoří možnost ukládání (externalizace) a zpětného načítání (internalizace) dat při splnění jejich persistence. K tomuto účelu existují dvě metody, z nichž každá může pracovat s "view" nebo "modelem":
- PROCEDURE (m: Model) Externalize (VAR wr: Stores.Writer)
- PROCEDURE (v: View) Externalize (VAR wr: Stores.Writer)

Prostřednictvím těchto metod je možné uložit (externalizovat) stav našich dat. Pro demonstrovaný případ grafů by první z metod ukládala hodnoty měření, vzorkovací periodu, čas prvního vzorku a další námi definované vlastnosti "modelu". Druhá z nich by mohla uložit parametry "view", tedy: nadpis, barvy čar atd. Parametr "wr" je zde oním objektem pro zápis jednotlivých dat a poskytuje služby k uložení číselných hodnot (REAL, INTEGER), řetězců (ARRAY OF CHAR), až po vlastní grafickou podobu "view".

K nim je duální dvojice metod:
- PROCEDURE (m: Model) Internalize (VAR wr: Stores.Reader)
- PROCEDURE (v: View) Internalize (VAR wr: Stores.Reader)

Tyto metody pracují inverzním způsobem oproti předchozím, tedy načítají (internalizují) dříve uložené hodnoty do našich instancí. Pro správnou činnost projektu je samozřejmě nutné zabezpečit, aby odpovídaly pořadí a formě, jako byla data předchozími metodami uložena (externalizována).

Výčet těchto metod by mohla uzavřít procedura, jejíž identifikátor je spíše konvencí zvolen "Deposit", a prostřednictvím které je možné námi vytvořený program začlenit do systému. Její tělo pro demonstrovaný příklad by mohlo být následující:

```
- PROCEDURE Deposit*;
  VAR
    v: View;
      m: Model;
  BEGIN
      NEW(m); (* Alokace modelu-může následovat i počáteční
              nastavení jeho položek. *)
      NEW(v); (* Alokování view-rovněž může následovat počáteční
              nastavení např. v.title := "untitled" *)
      v.model := m; (* Připojení modelu k view *)
      Views.Deposit(v); (* Volání standardní operace k zařazení
              view do systému *)
END Deposit;
```

Samozřejmě tento výčet metod není kompletní a pro správnou funkci musí být zajištěna implementace celé řady dalších metod např.

- PROCEDURE (m: Model) InitFrom (source: Models.Model)
- PROCEDURE (m: Model) CopyFrom (source: Stores.Reader)
- PROCEDURE (v: View) ThisModel (): Models.Model
- atd.

4. Význam a využití architektury MVC Framework

Jednou z nejvýznamnějších vlastností poskytovaných na základě separace na třídy "model, view, controller" je možnost různých prezentací téže informace (dat). Pro demonstrovaný příklad grafů by to znamenalo, že pro daná data uložená v "modelu" může existovat větší počet "view", tedy jejich grafických podob. Tak lze pro táž data vykreslit zároveň např. koláčový, sloupcový popř. i jiný graf a při změně hodnot-stavu "modelu" lze vhodným aktualizačním mechanismem zajistit adekvátní odezvu všech grafických pohledů – "view" k danému modelu připojených.

Konstrukce BBCB doplňuje tyto vlastnosti možností tvorby tzv. složených dokumentů (compound documents) a jinými prvky typickými pro dnešní trend softwarové konstrukce. Na základě komponentního přístupu je možné dotvářet a doplňovat vlastnosti a služby poskytované BBCB. Komponentní povaha BBCB klade největší nároky na návrh rozhraní prostřednictvím něhož jsou jednotlivé komponenty propojeny. Zvláštní pozornost je věnována ošetření a omezení šíření chyb. Za nejvýznamnější přednost lze považovat schopnost snadné údržby a rozvoje vlastního prostředí.

Literatura

1. Szyperski: Component Software, Beyond Object-Oriented Programming. Addison-Wesley, 1997, ISBN 0-201-17888-5
2. Pfister: Component Software, A Case Study Using BlackBox Components. http://www.oberon.ch/docu/case_study/index.html.