Přednosti a nedostatky OOP na příkladě reálné průmyslové aplikace

Michal Brožek

VŠB-TU Ostrava,FEI kat. MaR, tř.17 listopadu, 708 33 Ostrava-4, Česká Republika

Abstract
In the introduction of our account are some explanations of basic object oriented programming ideas and some more communicate notes. There are mentioned examples of designed objects and actions of their design. Object oriented programming has been abstractly elaborated in technical literature, which describes a lot of advantages. Target of this article is to compare theoretic closures with concrete data tests in design realisations. Especially we are to bring to bear in chance analyses of object oriented programming under TopSpeed MODULA-2 programming language environment.

1. PROGRAMOVACÍ JAZYK TOPSPEED MODULA-2 A OOP

Při návrhu a implementaci vizualizačního programu pro měření a regulaci koksárenské baterie bylo nutno vycházet ze základního požadavku převést a zobrazit technologický proces na obrazovku monitoru, tak jak je to běžné v technologických schématech. Navíc je nutné, aby takovýto program nejen daná data vizualizoval, ale zároveň tato data archivoval a zpracovával (statistika, trendy...). Pro implementaci byl zvolen programovací jazyk MODULA-2, který je vhodný pro realizaci RT úloh. Za jeho největší výhody lze označit:

- modularita,
- podpora paralelních procesů,
- podpora objektově orientovaného programování.

Prakticky všechny tři vlastnosti tohoto programovacího jazyka a jeho prostředí byly použity při realizaci programu pro vizualizaci technologie. Modularita proto, že každý oddělený problém lze řešit v odděleném programovém modulu. Použití paralelních procesů usnadňuje sběr dat, jejich zpracování a archivování, a v neposlední řadě byla využita i poslední vlastnost, objektově orientované programování. Objektově orientované programování dává programátorovi trojici velice silných programovacích nástrojů:

1. zapouzdření,
2. dědění,
3. polymorfismus.

Na rozdíl od klasického strukturovaného programování, kde je tok programem určen posloupností procedur a funkcí, je hlavním rysem objektově orientovaného programování tok dat. Data jsou „zapouzdřována" do tříd (encapsulating). Zapouzdřování umožňuje ochranu dat před nedovoleným zacházením. Přístup je dovolen pouze metodám, které na těchto datech operují. Všechny objekty pak musí náležet k nějaké třídě. Třída definuje

implementaci určitého objektu, a proto ji zde považujeme za předpis (=typ) pro vytváření objektů daného typu. Třída deklarovaná v TopSpeed MODULE-2 je *typem* a proměnné typu třída jsou nazývány objekty nebo také instancemi třídy (členy třídy).

Myšlenka dědění v OOP realizuje požadavek nového užití nebo sdílení kódu a to mnohem bezpečněji (předepsaným způsobem). Pokud chceme vytvořit třídu, o které víme, že se jen málo liší od již existující třídy, nebo má s touto společné rysy, pak použijeme dědičnost. Dědičnost umožňuje přenesení již existujících dat a metod nějaké třídy do třídy nově vytvářené a jejich doplnění o nová data a metody nebo jen modifikovat již existující metody. V TopSpeed MODULE-2 může být vytvořena celá hierarchie odvozených tříd. Tato hierarchie má hlavní třídu nahoře (base class) a další specifičtější třídy kaskádovitě sestupující dolů. Každá další odvozená třída dědí přístup ke všemu, co bylo již definováno v třídách předcházejících.

Polymorfizmus je schopnost použít stejný příkaz (metodu) na různých objektech. V objektově orientovaných technologii se říká, že objektu se zašle zpráva, která vyvolá metodu stejného jména (pokud existuje).

2. TŘÍDY A VIRTUÁLNÍ METODY

V tomto referátu se zaměříme na využití uvedené výhody a to Objektově Orientovaného Programování (dále jen OOP) v prostředí TopSpeed MODULA-2. V podstatě byly vytvořeny třídy dvojího druhu:
1. třídy pro grafické zobrazení technologických objektů,
2. třídy pro grafické zobrazení digitálních a analogových měřených údajů.

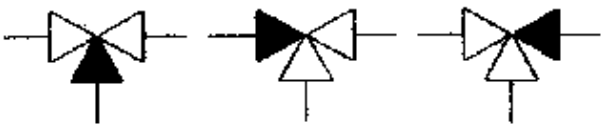
2.1 Třídy pro grafické zobrazení technologických objektů

Obecně lze navrhnout třídu pro realizaci určité skupiny technologických objektů, ale zároveň lze již předem říci, že implementovat takovouto třídu by bylo obtížné. Jednodušší se tedy zdá navrhovat jednotlivé třídy zvlášť pro jednotlivé technologické objekty a v případech, kde je to vhodné, použít dědičnost. To znamená, že například pro jednocestný ventil navrhneme a implementujeme třídu *ventil*, pro trojcestný ventil třídu *trojcestný_ventil*, atd.

Nyní jsou uvedeny příklady grafického znázornění jednocestného ventilu a trojcestného ventilu. Aktuální stav těchto technologických objektů může být znázorněn dvěmi barvami (černobíle viz. Obr.1 a Obr.2), nebo více barvami. V našem případě jsme použili zobrazení více barvami pro rozlišení zda je daný objekt v provozu nebo v poruše (tento stav není zachycen na uvedených obrázcích).

ventil otevřen ventil zavřen

Obr. 1 - ventil



ventil v poloze 1 ventil v poloze 2 ventil v poloze 3
Obr. 2 - trojcestný ventil

Oba uvedené technologické objekty budou v rozhraní dané třídy potřebovat různý počet vstupních údajů. Jednocestný ventil může být otevřený nebo zavřený, trojcestný ventil může být nastaven do několika různých stavů. Vstupní údaje lze rozdělit podle povahy do dvou skupin:

1. vstupní údaje potřebné pro počáteční inicializaci daného objektu,
2. vstupní údaje potřebné pro vlastní zobrazování stavu technologického objektu.

V prvním případě jde o počáteční inicializace ve smyslu umístění vizualizovaného objektu na obrazovku, jeho natočení a další doplňující údaje. Tyto údaje zřejmě mohou být pro větší skupinu tříd stejné. V druhém případě jde o data potřebná pro zobrazování stavů objektu, jako jsou stavy porucha, chod, otevření atd. Jak si ukážeme na příkladu výše zmíněných technologických objektů, lze zde uplatnit dědičnost. Ale to jen v tom případě, že vhodně navrhneme a implementujeme základní třídu *ventil* a tuto budeme dědit v odvozené třídě *trojcestný_ventil*. Nyní uvedeme příklad definice obou tříd. Aby bylo možné použít dědičnosti, zavedeme typ stav. Tento typ použijeme jak pro jednocestný ventil (s omezující podmínkou), tak pro jeho variantu trojcestný ventil.

```
TYPE STAV=(_1,_2,_3);
CLASS VENTIL;
   dataXY  : POSITION;
   popis   : STRING;
   rotace  : INTEGER;
   porucha : POINTER TO BOOLEAN;
   stav    : POINTER TO STAV;
PROCEDURE Init( XY : POSITION; otoceni : INTEGER; txt : STRING );
PROCEDURE Err( ok : ADDRESS );
PROCEDURE Stav( otevreno : ADDRESS );
VIRTUAL PROCEDURE Zobraz;
END VENTIL;

CLASS TROJCESTNY_VENTIL ( ventil );
VIRTUAL PROCEDURE Zobraz;
END TROJCESTNY_VENTIL;
```

Uvedené dva objekty se liší pouze způsobem grafického zobrazení. Klíčové slovo VIRTUAL před procedurou znamená, že daná metoda nebude statická (vazby těchto metod se určují již během kompilace - tzv. včasná vazba), ale virtuální („správný" kód procedury - její obsah, se vyvolá až při vlastním provádění programu - pozdní vazba). V TopSpeed MODULA-2 může být třída použita jako formální parametr libovolné procedury nebo funkce. Zde pak můžeme jako aktuální parametr použít libovolný objekt dané třídy, nebo objekt

dědicí třídy. Typickým příkladem na použití virtuálních metod je právě použití při zobrazování libovolné informace objektu s tím, že dědic používá jiného principu zobrazení.

```
PROCEDURE Refresh(VAR objekt : VENTIL );
BEGIN
   objekt.Zobraz;
END Refresh;
...
VAR 1V : VENTIL;
   3V : TROJCESTNY_VENTIL;
...
Refresh( 1V );
Refresh( 3V );
...
```

Pokud by metoda Zobraz byla statická, pak by v obou případech byly objekty zobrazeny jako jednoduchý ventil. Virtuální metody tedy umožňují rozeznání „správného zobrazení" až při běhu programu. V TopSpeed MODULA2 je umožněna hierarchická dědičnost, a vše, co bylo uvedeno na příkladu jednoduché dědičnosti, platí i pro hierarchickou dědičnost.

Již dříve jsme uvedli, že členská data dané třídy mohou být dvojího druhu: data pro počáteční inicializaci objektu a data pro vlastní práci objektu. Existují další data, která na první pohled jsou z hlediska programátora zbytečná. Totiž data o umístění technologického objektu v provozu, údaje o jeho parametrech, údaje o opravách (kalibrace), připojení na měřící uzel, vedení signálového vedení a mnoho dalších, pro údržbu důležitých údajů. Tyto údaje nemusí, ale mohou být součástí daného objektu. Je otázkou co je z hlediska vizualizace nebo programu výhodnější. Pokud chceme, aby tyto údaje byly zobrazovány např. při poklepání myší (třeba pravým tlačítkem=vlastnosti) na daný objekt, pak musí být tato data umístěna přímo v objektu, nebo tento objekt musí mít k těmto datům přímý přístup (ukazatel na tabulku dat apod.).

Z hlediska automatizace vkládání údajů do objektů je vhodné mít pro inicializaci objektu k dispozici tabulku, která obsahuje všechny potřebné údaje. Tato tabulka může být umístěna přímo v programu (nevhodné, pokud se předpokládají častější změny), nebo v souboru (vhodné řešení). Protože je v některých případech nutné tyto údaje editovat (i za chodu programu), je vhodné mít pro tento případ prostředky, pomocí kterých tuto editaci můžeme provádět. Takovýto editační formulář může být vytvořen jako obecná třída, protože shromažďuje údaje pro všechny technologické objekty společné (viz. výše), a z této obecné třídy (base class) pak mohou být odvozeny další specifičtější třídy. Tato myšlenka je realizována na následujícím příkladě:

```
CLASS FORMULAR;
(* deklarace ukazatele na tabulku dat *)
(* nebo na soubor s těmito údaji     *)
PROCEDURE ZobrazFormular ( VAR pos : CARDINAL );
PROCEDURE EditujFormular;
PROCEDURE UlozFormular;
(* zde mohou být definovány metody  *)
```

```
(* pro zobrazení jednotlivých údajů *)
END FORMULAR;
CLASS VENTIL ( FORMULAR );
...END VENTIL
CLASS TROJCESTNY_VENTIL ( VENTIL );
...END TROJCESTNY_VENTIL;
```

2.2 Třídy pro zobrazení digitálních a analogových měřených údajů

Doposud jsme se zabývali grafickým znázorněním určitého technologického objektu zastoupeného v programu objektem dané třídy. Druhou skupinou objektů jsou objekty používané pro vizualizaci měření. Na Obr. 3 jsou uvedeny dva objekty, objekt pro zobrazení analogového měřeného údaje a objekt pro vizualizaci binárního vstupu.

Obr. 3 - analogový a binární vstup

Následující příklad ukazuje jednu z možností, jak tyto třídy definovat:

```
CLASS Panel;
   dataXY      : POSITION;
   Popis,Jednt : STRING;
   Fyz     POINTER TO REAL;
   Len    : CARDINAL;       (* počet pozic zobrazovaného údaje*)
   Prec   : CARDINAL;       (* počet desetinných míst *)
PROCEDURE Init ( XY : POSITION; Popisek, Jednotka : ARRAY OF CHAR;
            Delka,Presnost : CARDINAL);
PROCEDURE Propoj ( FyzHodnota : ADDRESS );
PROCEDURE Zobraz;  (* zobrazí popisky *)
PROCEDURE Refresh; (* zobrazuje fyzikální hodnotu *)
END Panel;

CLASS Kontrolka;
   X,Y    : CARDINAL;
   PBStav : POINTER TO BOOLEAN;
   PBPor  : POINTER TO BOOLEAN;
   Jm     : STRING;
PROCEDURE Init( Xpos, Ypos : CARDINAL; Jmeno : ARRAY OF CHAR);
PROCEDURE PropojStav( BoolHodnota : ADDRESS );
PROCEDURE PropojPoruchu ( BoolHodnota : ADDRESS );
PROCEDURE Zobraz;  (* zobrazí poisky *)
PROCEDURE Refresh; (* zobrazuje binární hodnotu *)
END Kontrolka;
```

Jak vyplývá z definic těchto tříd, jsou základní údaje obsaženy vždy přímo v objektu. Pokud je nutné používat více údajů (údaje o čidle, kalibraci atd.), musí obě uvedené třídy dědit data a metody již dříve zmiňované třídy

formulář. Třída kontrolka je pouze specifickým případem vizualizovaných technologických objektů (jejich výstupní binární údaje jsou použity jako vstupní pro objekty uvedené v kapitole 2.1) a je požívána v případech, kdy není potřeba zobrazit celý technologický objekt se všemi jeho stavy, ale jen jeden jeho stav (např. porucha, nebo chod, nebo otevřeno apod.).

3. NEVÝHODY A VÝHODY OOP V PROSTŘEDÍ TOPSPEED MODULA-2

Nevýhody, na které jsme při implementaci navrhovaných objektu narazili, se daly s většími či menšími nesnázemi řešit. Jako hlavní nedostatek se jeví absence privátních metod či dat třídy. Na rozdíl od jiných programovacích jazyků, podporujících OOP (např. C++), je nutno data, která mají být společná pro určitou skupinu metod dané třídy, deklarovat v definici třídy. Tímto jsou veškerá používaná data zviditelněna (i když ne zpřístupněna, viz. zapouzdření). Jiný problém nastává, pokud chceme, aby byla vytvořena soukromá metoda třídy. Takovouto metodu v TopSpeed MODULA-2 nelze realizovat, protože jak metody, tak i všechna data musí být, jak už bylo řečeno, deklarovány v definici třídy. Tím se stávají všechny metody dané třídy přístupné. Tento nedostatek lze z jistými omezeními obejít použitím modulů. To znamená, že definici třídy umístíme do definičního modulu a danou třídu realizujeme v implementačním modulu a skryté metody zde realizujeme jako vnitřní procedury či funkce daného modulu.

Výhodou se naopak jeví možnost definování virtuálních metod, jak bylo uvedeno v kapitole 2,1. Při použití virtuálních metod je nutné si pamatovat, že metodu jako virtuální musíme definovat už v základní třídě a ponechat ji virtuální pro každou další odvozenou třídu.

Další výhodou je také možnost jak hierarchického dědění (viz. příklad definice třídy trojcestného ventilu na základě jednoduchého ventilu, který již dědí data a metody třídy formulář), tak i možnost vícenásobného dědění (tzn. vytvořit dědice ze dvou tříd najednou).

```
CLASS FORMULAR;
...END FORMULAR;

CLASS VENTIL;
...END VENTIL;

CLASS JEDNCESTNY_VENTIL ( FORMULAR , VENTIL );
...END JEDNOCESTNY_VENTIL;
CLASS TROJCESTNY_VENTIL ( FORMULAR , VENTIL );
...END TROJCESTNY_VENTIL;
```

4. ZÁVĚR

V článku byly uvedeny příklady návrhu několika vybraných tříd. Z grafických zástupců technologických objektů to byly třída ventil a trojcestný ventil. Uvádíme zde další objekty, pro něž jsou vytvářeny nebo byly vytvořeny třídy, jsou to: klapy, dopravníky, dmychadla, zásobníky. Z grafických tříd pro zobrazení analogových údajů nebyla uvedena třída pro pohyblivý analogový údaj (objekt mění svoji pozici a délku).

Tyto třídy byly použity při vývoji konkrétních řídicích aplikací. Zde se potvrdila zároveň účinnost programovacího jazyka TopSpeed MODULA-2, i pokud jde o jeho další důležité rysy: modularitu a paralelní procesy.