Práce s objektově orientovaným generátorem programů Cobol

Vlastimil Čevela

Předkládaný materiál vychází z prvních praktických zkušeností při využívání objektově orientovaného Generátoru programů Cobol GPC 1.1. dle [5], při práci s jazykem MS Cobol 4.5 na počítačích PC AT 286 a 386SX. S pomocí příkladů ukazuje možnosti uplatnění objektových metod na úrovni návrhu zdrojových textů programu.

1 Úvod

Objektově orientovaná techologie tvorby programů představuje jeden z nejvýkonnějších nástrojů ke snížení náročnosti vývoje a údržby aplikačního programového vybavení. Jeho obecná filozofie spočívá v tom, že data i algoritmy pro jejich zpracování jsou společně reprezentovány ve struktuře, nazývané objekt. Hlavní výhodou pak je mnohonásobná použitelnost jednou vytvořeného programového kódu.

Generátor programů Cobol GPC 1.1 pracuje na úrovni zdrojových textů a je schopen realizovat většinu principů objektově orientovaného programování. Základem jeho činnosti je skládání výsledného programu z tzv. objektových modulů (dále modulů), které je možno vícenásobně používat, modifikovat a integrovat do libovolných celků. Dalším krokem je přeložení výsledného programu běžným kompilátorem.

Jak je z výše uvedeného zřejmé, v případě GPC 1.1 se nejedná o plnou šíři uplatnění objektově orientované technologie, jako je umožnění variability až ve fázi realizace programu, návaznost na speciální objektové jazyky, příp. na objektově realizovanou HW podporu apod.

Předmětem výkladu není popis metodiky objektově orientovaného přístupu k vývoji programů, ten byl fundovaně rozebrán v [1], resp. v [2] a [3].

Cílem dalších odstavců proto je spíše snaha ukázat, jak mohou být obecné přístupy objektově orientovaného programování realizovány i poměrně velice jednoduchými prostředky – a že i v takovém případě mohou přinášet zajímavé výsledky.

2 Principy práce generátoru

Výstupní programový text, který představuje zdrojový program dle běžných pravidel jazyka Cobol, je postupně skládán ze stavebnice objektových „modulů", které reprezentují objekty. V aktuální verzi generátoru GPC 1.1 jsou jednotlivé moduly fyzicky realizovány ve formě textových souborů.

Každý modul může obsahovat vlastní popisy dat a procedury příkazů i odkazy na jiné moduly.

Modul, který obsahuje odkaz na jiný modul považujeme za „nadřízený", modul na který se odkaz vztahuje pak za „podřízený".

V případě potřeby může modul i sám přímo definovat dočasné „pracovní moduly", na které se potom vícekrát odkazuje, resp. na které se mohou odkazovat jemu podřízené moduly.

Pojem „odkaz" znamená, že na místo v nadřízeném modulu, kde se odkaz vyskytuje, bude zařazen příslušně parametricky modifikovaný text podřízeného modulu. Toto pravidlo platí shodně i pro odkazy na pracovní moduly.

Nadřízený modul nejvyšší úrovně, ze kterého se pro příslušný program postupně na základě odkazů odvíjí skládání výsledného zdrojového textu, se nazývá „hlavní modul programu".

Moduly, určené k opakovanému využívání, tj. ty, které jako objekty mají charakter „třídy", ze které jsou pomocí modifikačních parametrů vytvářeny jednotlivé „instance" jejich konkretních aplikací, budeme označovat písmenem „T" jako „typové moduly".

T Příklady typových modulů: (2.1)

soubtx.m	práce s textovým souborem
rol–i.m	rolování v indexovém souboru
akt–i.m	aktualizace indexového soboru
eda.m	aktualizační editor v paměti
_sex.m	kontrola na existenci souboru
_x91.m	zprostředkování exec–volání MS DOS
_uatr.m	zapnutí uživatelského atributu

Jak je vidět z příkladů, modulem může být několik řádek (_x91.m) anebo i více stránek (akt–i.m) zdrojového textu.

Důvodem k rozdělení návrhu programu na objektové moduly nemusí být jen opakovaná využitelnost obecnějších úseků zdrojového textu. Objektový přístup totiž umožňuje

velice snadno „zapouzdřit" určité dílčí části rozsáhlejšího nebo složitějšího problému tak, že pro ně lze definovat jednoznačné „rozhraní" a řešit je zcela samostatně jako „černé skříňky".

Hlavní moduly programu a ty, které jsou jednorázovým řešením určité aplikace nebo její části budeme označovat písmenem „A" jako „aplikační moduly".

A Příklady aplikačních modulů (2.2)

pac.m ..... pacient u lékaře
pac1.m ..... pacient – dítě
pac2.m ..... dospělý pacient

V praxi budou moduly kombinovány, tj. například hlavní modul programu (pac.m) bude rozdělen na části (pac1.m, pac2.m), z nichž každá může různě pracovat s různými datovými soubory (akt–i.m, soubtx.m, rol–i.m, eda.m).

Generátor GPC 1.1 pracuje pouze na úrovni zdrojových textů, takže objektem s opakovaným výskytem v příkladu (2.2) nebude každý jednotlivý pacient, ale spíše aktualizace kartotéky (akt–i.m) s individuálně modifikovanou skladbou dat, klíčů a obrazovek pro každou z kategorií pacientů.

3 Postup při návrhu modulů

V programových textech modulů jsou popisy datových položek včetně případných dalších rubrik, jednotlivých oddílů a sekcí i odstavců procedury příkazů psány v souladu s pravidly jazyka Cobol. Dle potřeby pak obsahují několik typů formálních parametrů, které budou při generování nahrazeny konkrétními hodnotami.

Pro ovládání jednotlivých funkcí generátoru slouží 2–znakové „řídící kódy", uváděné v úvodu řádků, mimo pásmo programu.

V rámci textu kterýchkoliv modulu lze běžně používat „volné deklarace" (blíže lit. [4]), které umožňují v úsecích, vymezených speciálními znaky, psát deklarace počátečních hodnot veškerých znakových, číselných, textových i hexadecimálních řetězců psát zcela volně, tj. nejen bez úrovní a popisů „pic", ale i rozdělené na dílčí části s libovolným počtem oddělovacích mezer.

Přehled parametrů a řídících kódů (3.1)

jednoduché f–parametry	...	–% (č.) %0 (jm. sb.) %1 až %40
řádkové f–parametry	...	=a /a až a/ /– až –/ (volba)
jednoduché k–parametry	...	. jméno P1 až Pn
		.. Pm ald ...

		.. P40
řádkové k–parametry	...	. –a (zrušit)
		+a (vybrat z volitelných)
		>a (nahradit 1 řádkem)
		<a až a> nahradit více řádky
volné deklarace	...	[[– [# [$ [8 [2] *[*] [<]>
bloky	...	xx mezerax mezera–
odkaz na stand. záhlaví	...	:
komentáře	...	*mezera /* až */
definice prac. modulů	...	< až >

Poznámka: Jednoduché f–parametry jsou umístěny kdekoliv v textu, řídící kódy musí být povinně od 1. sloupce.

4 Standardní záhlaví, pracovní moduly a jednoduché parametry

Dále uvedeme příklad na vytvoření pracovního modulu k vypisování zprávy na obrazovku, s možností modifikace textu, umístění i atributů.

A Příklad na vypisování zpráv PG1.m

```
: szcobol
<       _pis.–m
              display
 .                  „%1"
                    at %2
                    with background–color %3
                         foreground–color %4 %5
>
ws
        1     K pic x.
pd
 .            pis.–m 'první zpráva' 1020 1 7
              accept K
 .            _pis.–m 'další zpráva' 1020 3 6 highlight
              stop run.
```

Výsledný vygenerovaný text PG1.cbl

```
working–storage section.
1       K pic x.
procedure division.
        display
```

```
        „první zpráva"
        at 1020
        with background–color 1
        foreground–color 7
accept K
display
        „další zpráva"
        at 1020
        with background–color 3
        foreground–color 6 highlight
stop run.
```

Komenář k příkladu PG1:

– Řídící kód „:" odkazuje na zkratky standardních záhlaví, které jsou spolu s příponou „.cbl" obsaženy v definičním souboru „szcobol", který vypadá následovně:

```
+.cbl
fc      file–contro.l
fs      file section.
ws      working–storage section.
ls      linkage section.
ss      screen section.
pz      procedure division.
pu      procedure division
de      declaratives.
ed      end declaratives.
pd
```

Výše definované 2–písmenové řídící kódy umožňují v textu modulu libovolné střídání deklarací a příkazů – výsledný text bude setříděn v pořadí shodném s výše uvedeným pořadím kódů.

– Kódy „< ... >" vymezují definici pracovního modulu „_pis.–m", který je dále opakovaně využíván pomocí odkazu, definovaného řídícím kódem „. " od 1. sloupce.

– V rámci definice pracovního modulu jsou uvedeny jednoduché formální parametry %1 až %5, které jsou při generování odkazu nahrazeny konkretními hodnotami. Nejsou–li poslední v řadě uvedeny (%5), bajty jejich formální definice se ignorují.

– Přípona „.–m" platí pro dočasné, tj. pracovní moduly. Generátor po skončení své práce totiž automaticky zajišťuje provedení příkazu „del *.–m". Doporu-

čená konvence přípon je „.m“ pro aplikační i typové moduly a „.–“ pro definice dat.

Generátor připouští v každém modulu maximálně 40 jednoduchých parametrů, označovaných kombinacemi „%1“ až „%40“. Pokud se jejich konkretizace při odkazu na modul z nadřízeného modulu nevejde přímo na odkazový řádek s řídícím kódem „. “, je možno pokračovat na dalších řádcích s kódy „..“.

Konkretními jednoduchými parametry mohou být znakové nebo textové (v apostrofech, max. 64 znaků vč. mezer) řetězce a mohou představovat nejen jména a jejich součásti (předpony, přípony), ale též literály, úseky programového textu Cobol, nebo odkazy na jména pracovních modulů.

5 Práce s řádkovými parametry

Předpokládejme typový modul, do kterého máme parametricky dosadit deklaraci datové struktury.

T Úsek typového modulu pro práci s dat typ1.m

```
ws
      1    VT–%1.
=d      2  pic x(80).
```

A Příklad na změnu typu řádkovými parametry PG2.m

```
: szcobol
ws
.     typ1.m     VSTUP
<d
          2  CISLO      pic 999.
          2  KOD        pic x(5).
          2  JMENO      pic x(30).
d>
```

Výsledný vygenerovaný text PG2.cbl

```
working–storage section.
1     VT–VSTUP.
  2  CISLO      pic 999.
  2  KOD        pic x(5).
  2  JMENO      pic x(30).
procedure division.
```

Komentář k příkladu PG2:

- Pokud by nebyla skupina řádků parametru <d ... d> uvedena, dosadil by se implicitní popis „pic x(80)".
- Jednoduchý parametr „%1" je použit pro dosazení přípony „VSTUP", která tak identifikuje jméno datové struktury a tím umožňuje v příkazech programu komunikaci mezi moduly.

Formální definice „řádkového parametru" může být učiněna jedním řádkem s řídícím kódem „=písmeno", anebo skupinou řádků, vymezených kódy „/písmeno ... písmeno/".

Při generování odkazu se pak příslušný řádek nebo výše uvedeným způsobem vymezená skupina nahradí konkretním řádkem nebo skupinou řádků, zadanými při odkazu z nadřízeného modulu.

Řádkové parametry umožňují při generování příslušného odkazu realizovat kompletní změnové řízení, tj. nejen doplněk, změnu nebo zrušení textu v podřízeném modulu, ale i výběr jednoho nebo několika parametrů z předem připravených variant.

Počet řádkových parametrů v programu je dán počtem velkých a malých písmen anglické abecedy, vlastní počet řádků nahrazovaného nebo vkládaného textu v každém parametru není omezen.

6 Lokální jména a odkazy

Ve zdrojovém textu každého modulu, na který bude odkazováno, lze vytvářet programátorská jména, která běžně nejsou ze žádného jiného modulu přístupná. Tato „lokální jména" jsou jako formální parametry definována kombinací „–%" s následujícím nečíselným znakem nebo mezerou a jsou odvozována od pořadového čísla výskytu odkazu v rámci celého programu.

T Úsek typového modulu s lokálními jmény typ2.m

```
ws
      1     POCET-%       pic 999.
pd
      NACITANI-%1.
            add 1 to POCET-%.
```

A Příklad s ukázkou možností lokálních jmen PG3.m

```
: szcobol
pd
            perform NACITANI-A.
            perform NACITANI-B.
```

```
        stop run.

.       typ2.m      A
.       typ2.m      B
```

Výsledný vygenerovaný text — PG3.cbl

```
working-storage section.
1       POCET-002       pic 999.
1       POCET-003       pic 999.
procedure division.
        perform NACITANI-A.
        perform NACITANI-B.
        stop run.
NACITANI-A.
        add 1 to POCET-002.
NACITANI-B.
        add 1 to POCET-003.
```

Komentář k příkladu PG3:

- Při opakovaných odkazech na stejný modul byla jako přípony lokálních jmen postupně dosazena trojčíslí 002 a 003, takže všechna jména zůstala jednoznačná.

- Vzhledem k tomu, že v průběhu ladění a při jakékoliv změně počtu a pořadí odkazů se tato trojčíslí mění, nelze je mimo modul snadno odvodit, takže jejich využívání pro komunikaci nemá praktický smysl.
- Kombinace „pomlčka a 3 číslice" však nesmí být v programech k žádným jiným účelům využívána.
- Z příkladu je též zřejmé vícenásobné vyvolání určité akce, tj. příkazů v odstavci „NACITANI-%1", identifikované pomocí přípon „A" a „B".

Moduly jsou na všech úrovních rovnocenné, tj. v každém z nich mohou být parametrizované odkazy na moduly, podřízené na další úrovni atd. Jedinou formální podmínkou je, že odkaz nesmí být prvním platným řádkem modulu.

V současné verzi je celkový počet odkazů a tím i počet úrovní omezen na 999. V případě potřeby lze přípony lokálních jmen rozšířit na 4 číslice a tím zvýšit počet odkazů, resp. úrovní na 9999.

Pro případ rekurzivního odkazu, který by způsobil zacyklení, činí generátor po zpracování 99 odkazů kontrolní dotaz, zda se nejedná o chybu.

7 Využití pracovních modulů – slovník a definice dat

Poměrně častým případem může být nutnost parametricky předat řádkový parametr na předem neznámou podřízenou úroveň. K tomuto účelu lze výhodně využít pracovní modul, definovaný v hlavním modulu programu.

```
T     Typový modul s f-parametrem pro odkaz                    typ3.m
ws
      1     VT-%.     *> typ3.m – deklarace dat
.     %1
```

```
T     Nejblíže nadřízený modul                                 typ4.m
      * text      typ4.m
.     typ3.m      %1
.     typ3.m      %2
```

```
T     Výše nadřízený modul                                     typ5.m
      * text      typ5.m
.     typ4.m      %1    %2
```

```
A     Příklad hlavního modulu programu                         PG4.m
: szcobol
      * příklad PG4.m
ws
<     XXX.-m
            *> XXX – konkretní deklarace
.     jmeno.-     '2'    -PG4
>
.     typ5.m      XXX.-m     seznam1.-
```

```
A     Příklad modulu definice dat                              DDM.m
/* centrální slovník dat a datových struktur projektu
      libovolně komentovaný definičními popisy
*/
<     cislo.-
      %1    %3CISLO%2      pic 999.
<     kod.-
      %1    %3KOD%2        pic x(5).
<     jmeno.-
      %1    %3JMENO%2      pic x(30).
```

```
<     seznam1.–
          *> seznam jmen dle čísel
.     cislo.–     2
.     jmeno.–     2
<     seznam2.–
      *> seznam jmen dle kódů
.     kod.–       2
.     jmeno–.     2
>
```

Výsledný vygenerovaný text PG4.cbl

```
0002*   příklad PG4                                          PG4.m
0003 working–storage section.                               szcobol
0001*   text typ5.m                                         typ5.m
0001*   text typ4.m                                         typ4.m
0002 1      VT–004.    *> typ3.m – deklarace dat            typ3.m
0005        *> XXX – konkretní deklarace                    PG4.m
0014   2    JMENO–PG4      pic x(30).                       DDM.m
0002 1      VT–005.    *> typ3.m – deklarace dat            typ3.m
0017        *> seznam jmen dle čísel                        DDM.m
0005   2    CISLO          pic 999.                         DDM.m
0014   2    JMENO          pic x(30).                       DDM.m
0006 procedure division.                                    szcobol
```

Komentář k příkladu PG4:

- Typový modul „typ3.m" pro svoji potřebu vyžaduje deklaraci datové struktury, která má být zadána parametricky. Modul je přitom natolik obecný, že jeho zařazení v hierarchii může být různé a předávání konkretních řádkových parametrů by bylo příliš složité. Proto je doplnění potřebné definice realizováno jednoduchým formálním parametrem „. %1", ze kterého po konkretizaci vznikne odkaz na pracovní modul „. XXX.–m", který příslušnou deklaraci obsahuje.
- Příklad dále ukazuje definici datových položek a datových struktur ve formě samostatného modulu „DDM.m", který může sloužit pro celý projekt jako slovník dat.
- Při definování deklarace „XXX.–m" je využito elementární datové položky ze slovníku dat, s doplněním individuální přípony „–PG4".
- V ladícím režimu jsou v okrajových sloupcích výpisu uváděna jména souborů a čísla řádků, ze kterých výsledný programový text vznikl.

8 Rozhraní objektových modulů

Pro každý objektový modul je třeba jednoznačně vymezit jeho „rozhraní", tj. množinu zpráv, pomocí kterých komunikuje se svým okolím. Poněvadž v našem případě jde o skládání programu ze zdrojových textů modulů, musíme do rozhraní zahrnout též pokyny, podle kterých má být provedeno generování.

K popisu rozhraní lze výhodně využít jednopísmenové „dekompoziční kódy", které přesně charakterizují jednotlivé kategorie prvků tohoto rozhraní.

Přehled dekompozičních kódů v rozhraní modulů (8.1)

parametry pro generátor
- N ..... identifikační přípony jmen
- G ..... ostatní parametry pro generování

předávání dat při práci programu
- I ..... data, která jsou modulem čtena
- O ..... data, která jsou modulem vytvářena
- W ..... data, která jsou modulem měněna

předávání řízení při práci programu
- A ..... akce prováděná modulem, vyžádaná mimo modul
- P ..... akce prováděná modulem v místě odkazu na něj
- R ..... požadavek z modulu na akci mimo modul
- C ..... podmínková jména modulu testovatelná mimo modul
- S ..... podmínková jména modulu nastavitelná mimo modul

Každý modul je tedy třeba chápat ve dvou časově rozdílných situacích. V první fázi jako dílec ze stavebnice, který bude spolu s jinými dílci jednou nebo vícekrát využit pro jednorázové vygenerování výsledného zdrojového programu. Po kompilaci a sestavení programu v binární formě pak bude při jeho každém spuštění spolupracovat s ostatními moduly formou vzájemného předávání řízení a dat.

Dále si uvedeme příklad na definici rozhraní typového modulu, který má zprostředkovávat práci s textovým souborem.

```
/*T ------------------------------------------------------------
1     Práce s textovým souborem                          sbtx.m
------------------------------------------------------------------
2
      N     identifikační přípona jmen                   %1
      G     definice datové struktury věty               VT-%1
            /  implicitně „2 pic x(80)."
```

	/ vlastní popis	<d ... d>
I	jméno zpracovávaného souboru	JM–%1
W	věta zpracovávaného souboru	VT–%1
A	sekvenční přečtení souboru	PRECTI–%1
R	předání přečtené věty ke zpracování	VETA–%1
C	rozlišení stavu čtení	
	/ první věta v souboru	PRVNI–%1
	/ další věta v souboru	DALSI–%1
S	možnost předčasného ukončení čtení	KONEC–%1
A	otevření souboru pro výstup	OPEN–O–%1
A	uzavření souboru, otevřeného pro výstup	CLOSE–O–%1
A	zápis věty do výstupního souboru	PIS–%1

3

Modul nevyužívá žádné další moduly.

4

Hlavním cílem činnosti modulu je sekvenčně přečíst textový soubor a jednotlivé věty předat nadřízenému modulu ke zpracování.

Kromě toho modul umožňuje též otevřít soubor pro výstup a zapisovat do něj věty, připravené v nadřízeném modulu.

5

Příklad 1 – přečtení a zobrazení vět ze souboru „text.tx"

```
: szcobol
pd
        move „text.tx" to JM–TEXT.
        perform PRECTI–TEXT.
        stop run.
    VETA–TEXT.
        display VT–TEXT.
        stop „pokračování – Enter".
    sbtx.m      TEXT
```

Komentář k příkladu definice rozhraní modulu „sbtx.m":

- Řádky začínající lomítkem „/" popisují alternativní řešení nebo situace.
- Přečtení textového souboru patří mezi běžné úlohy, takže už sám fakt, že není nutno se stále znovu zabývat celým algoritmem může přinášet úsporu.

- V zájmu zjednodušení je rozhraní definováno pouze pro základní algoritmus. V konkrétní praxi by součástí modulu byla kontrola na existenci souboru, možnost rozlišení nejen první, ale i poslední věty souboru, apod.
- Implicitně je předpokládána deklarace věty vcelku. Pokud je to nutné, je možno ji pomocí řádkového parametru „d" (příklad PG2.m výše), případně s využitím pracovního modulu (příklad PG4.m), nahradit deklarací vlastní.
- Příkaz „perform PRECTI–%1" znamená, že si nadřízený modul vyžádá provedení příslušné akce. V jednom odstavci, ihned za řádkem s příslušným kódem „A", je řádek s kódem „R", který sděluje, že v průběhu akce je příkazem „perform VETA–%1" řízení předáváno zpět do nadřízeného modulu.
- V průběhu čtení je dále možno testovat podmínková jména „PRVNI–%1 a DALSI–%1", která ukazují stav čtení a pomocí příkazu „set KONEC–%1 to true" lze čtení předčasně ukončit.
- Příklad 1, který je součástí definice modulu, ukazuje triviální způsob využití modulu k přečtení souboru po větách. Jeho smyslem je ukázat, jaká je vhodná forma pro umístění odkazu na modul (na konci pd) a celkového uspořádání nadřízeného modulu.

9 Závěr

Ve vymezeném rozsahu byly možnosti využívání generátoru ilustrovány na nejjednodušších příkladech. K účelnosti jeho nasazení při praktickém programování a k náročnosti přechodu na účinné využívání jeho předností lze říci asi toto:

- První podmínkou je zvládnutí techniky práce, tj. seznámení se s řídícími kódy, pravidly pro tvorbu jmen a definováním odkazů s modifikujícími parametry.

  Rozdíly oproti klasickému psaní programů jsou poměrně malé, takže by osvojení formálních pravidel mělo proběhnout rychle.

- Výrazně náročnější je potřeba změny myšlenkového přístupu při rozdělování řešené problematiky na objekty a definování jejich rozhraní. V té souvislosti je třeba rozlišovat, zda se jedná o tvorbu aplikačních nebo typových modulů.

  Pro aplikační využívání dobře dokumentovaných a příklady opatřených typových modulů by mělo stačit krátké zaškolení. Zde je hlavní otázkou spíše znalost rozhraní typových modulů, které jsou k dispozici.

  Návrhy typových modulů ovšem vyžadují důkladné znalosti algoritmizace i možností jazyka. Tady totiž vystupuje do popředí především nutnost správné formulace zadání problému.

Už sama existence metodiky pro tvorbu rozhraní sice pomáhá při formulování takových zadání, ale profesionální zvládnutí vyžaduje určitou dobu zkušeností při práci na konkretních programech.

– Dosavadní výsledky naznačují, že generátor dobře podporuje zásady systémového přístupu k programování, formulované panem Dijkstrou, Langeforsem a dalšími asi v tom smyslu, že „složité problémy je třeba rozdělit na jednoduché, aby byly zvládnutelné".

Objektový modul i ve formě pouhého zdrojového textu totiž vzhledem k možnostem formulovat přesné rozhraní a zajistit potřebné zapouzdření má všechny předpoklady stát se velice praktickou černou skříňkou ve výše uvedeném pojetí.

Literatura

[1] Polák J.: Jak objektově programovat, Programování '91, DT Ostrava, 1991

[2] Polák J., Merunka V.: OOP I a II, Softwarové noviny, č. 2 a 3, Praha 1993

[3] Referáty bloku OOP, Programování '93, DT Ostrava 1993

[4] Čevela V.: Návrh objektově orientovaného generátoru programových textů, Programování '92, str. 69 a dále, DT Ostrava 1992

[5] Generátor programů Cobol GPC 1.1, firemní dokumentace, Programátorské služby Cobol, © Čevela *** Šlursová, 1993

Autor: Ing. Vlastimil Čevela
Benešova 279
664 42 Modřice
tel. 05 – 303 367, fax 05 – 322 504