# Visual Prolog与Visual C++混合编程技术研究

徐彤， 张莉， 李松

(空军工程大学 导弹学院，陕西三原 713800)

**摘 要**：分析了Visual Prolog论域与C数据结构的对应关系以及函数、谓词的调用约定，讨论了Visual Prolog与Visual C++进行混合编程的实现方法，并通过二者相互调用的代码演示了该方法，从而将VC与Visual Prolog的优势有效的结合起来，使智能应用的开发简单而高效。

**关键词**：论域；谓词；动态链接库；堆栈

**中图分类号**：TP312 **文献标识码**：A **文章编号**：1009-3516(2006)03-0058-03

Visual Prolog是Prolog开发中心推出的基于Prolog语言的可视化集成开发环境。由于它非常适合于专家系统、规划和其他AI相关问题的求解，因此，成为目前国际上研究和开发智能化应用的主流工具之一[1]。在我国Visual Prolog的应用也正在逐渐兴起。

在智能应用的开发过程中，Visual Prolog基于描述性的知识表示和完备的用于推理的搜索控制机制，使其对开发者具有强大的吸引力[1]，但仍有许多理由使开发者在应用Visual Prolog开发程序的同时使用其他编程语言。例如，进行复杂的数值计算、中断处理，或者原本就有许多工作已经通过其他编程语言完成[2]。事实上，在智能应用的开发中，根据各种编程语言的优势，同时合理的使用Visual Prolog和其他语言进行混合编程，将能够很好的提高效率和性能。本文讨论的就是利用动态链接库进行Visual Prolog与Visual C++混合编程的方法。

## 1 论域与数据类型

要进行Visual Prolog与Visual C++所生成代码间的互相调用，首先需要明确的就是Visual Prolog论域与Visual C++数据类型的对应关系，这是正确使用内存中所存贮数据的前提。

1)标准论域。Visual Prolog的大部分标准论域与Visual C++中的基本数据类型是一一对应的，以32位程序为例，Visual Prolog的char、byte、(u)short、(u)long、integer、real、word、dword分别对应于Visual C++中的char、byte、(unsigned) short、(unsigned) long、int、double、word、dword，用这些相对应的论域和数据类型声明的变量，在内存中采用了相等的字节数来实现。

标准论域中的string和symbol表示了字符串，但字符的存贮方式有所不同。一个string论域的变量相当于C语言中指向char型数组的指针，而一个symbol论域的变量则相当于一个散列表的指针，该散列表包含了字符串的有关信息。

2)复合论域。复合论域相当于C语言中结构体与共用体的嵌套。复合论域myDomain所对应的C语言实现方法如下：

```
myDomain =                  struct myDomain{
    i(integer);                 unsignedchar functor;
    f(real);        ⇔           union{
    s(string)                   int i;double f;char * s}
```

收稿日期:2005-05-27
基金项目:国防科技预研基金资助项目(51406050301DZ01)
作者简介:徐 彤(1978-),男,山东菏泽人,博士生,主要从事智能决策、自动推理软件研究.

}

由上可见,Visual Prolog复合论域的存贮包括两部分,即1字节的算符描述和一个共用体。算符描述指出了当前变量所属算符的编号,上面代码中算符i、f、s的编号依次为1、2、3;共用体中存贮了当前变量的值。

3)表论域。在Visual Prolog中,无论是标准论域、复合论域还是自定义论域,都可以用来声明一个表论域。表论域的变量在内存中的存储结构相当于C语言的链表。链表中的每一个节点都包含了用于声明表论域的原论域变量所对应的存贮结构,以及一个指向下一个节点的指针。在用简单论域(以integer论域为例)和复合论域(以myDomain论域为例)所声明的表论域中,一个节点的C数据结构表示如下:

表论域声明:(简单论域构成的表论域节点)

```
integer_list = integer *
```

对应的C数据结构:

```
struct node_integer_list
{
    int      value;
    struct node_integer_list * pNext;
};
```

表论域声明:(复合论域构成的表论域节点)

```
myDomain_list = myDomain *
```

对应的C数据结构:

```
struct node_myDomain_list{
    unsignedchar functor;
    union{int i;double f;char * s} ;
    struct node_myDomain_list * pNext;
    };
```

4)引用论域。在Visual Prolog中可以对已有论域声明它的引用论域。引用论域与C语言指针相对应。

## 2　调用约定

为了进行Visual Prolog和VC间的相互调用,除了保证所调用的函数和谓词使用的数据类型与论域相对应外,还要保证Visual Prolog谓词与VC函数遵循相同的调用约定。调用约定主要在两方面进行约定:①函数或谓词在被编译时名称的转换规则;②调用时参数的压栈顺序以及返回时压栈参数是否出栈[3~4]。

VC中__stdcall调用约定与Visual Prolog中language stdcall约定具有相同含义。如果使用它们来约束VC函数和Visual Prolog谓词,则函数和谓词将具有如下特征:

1)编译时它们的名称前都会被添加下划线,名称后被添加符号"@"和一个整数。这个整数表示函数或谓词所带参数的全部字节长度。下面是具体的函数和谓词被编译的结果。

Visual Prolog谓词:myExample(integer,real) language stdcall　　编译结果:
VC函数:　void __stdcall myExample(int,double)　⇒　_myExample@12

2)调用时,参数按从右到左顺序压栈,返回时参数出栈。

由以上可知,只要谓词名与函数名相同,参数个数相等,参数论域与数据类型相对应,那么使用__stdcall和language stdcall对函数和谓词进行调用约定的限制,将能得到相同的编译结果,并能保证按同样的方式调用,从而使Visual Prolog和VC间的相互调用成为可能。

## 3　从Visual Prolog调用VC函数

在Visual Prolog中调用VC函数,首先应把在VC中编写的函数声明为__stdcall调用方式,把函数作为动态链接库的导出函数编译链接到dll文件中,然后在Visual Prolog项目里,加入该动态链接库的导入库文件(.LIB文件),并将动态链接库拷贝到调用VC函数的Visual Prolog程序能够找到的路径下。再为所调用的VC函数在Visual Prolog中声明一个谓词与之对应。这个被声明的谓词应遵循以下约定:

1)必须声明于全局谓词段。

2)指定谓词的语言规范为language stdcall。这个规范使系统能够按照正确的调用约定(包括名称转换规则、参数压栈顺序、返回前压栈的参数是否出栈等)对VC函数进行调用。

3)谓词名与函数名必须相同,谓词参数的论域类型与VC函数参数的数据类型相对应,顺序相同,且谓词参数的流程模式全为输入模式。

4)谓词的确定性模式为procedure。

5)如果所调用的VC函数返回类型不是void,就必须在谓词声明前,加上一个论域,该论域与VC函数返

回类型按照本文第二节中所列举的对应关系进行对应。

下面是两个典型的 C 函数,分别体现了不同的返回类型以及参数为指针和非指针的情况:

```
void__stdcall myFunc_1(int,double,char);
int__stdcall myFunc_2(int *);
```

它们在 Visual Prolog 中的声明方法为

```
Domains
    pointer_int = reference integer
Global Predicates
    procedure myFunc_1(integer,real,char) - (i,i,i) language stdcall
    procedureintegermyFunc_2(pointer_int) - (i)language stdcall
```

按照上述方法进行声明的谓词可以在 prolog 代码中正常使用,就像使用普通的由 prolog 代码声明并在子句段实现的谓词一样。

## 4 从 VC 调用 Visual Prolog 谓词

从 VC 调用 Visual Prolog 谓词,同样要通过动态链接库,具体步骤如下:

1)在所创建的 Visual Prolog 项目中,将项目的目标类型(Target Type)选为 dll,把所编写的 prolog 代码编译链接成一个动态链接库。想要在动态链接库中导出供其它可执行代码使用的谓词,应使用调用约定 language stdcall。并且该谓词不能有多个流程模式,否则就要实现该谓词的多个 C 语言版本。

2)在模块定义文件(.DEF 文件)的 EXPORTS 段说明动态链接库中的哪些谓词可以被导出。例如,若想导出谓词 myPredicate_1 和 myPredicate_2,应在.DEF 文件中进行如下定义:

```
Exports
    myPredicate_1
    myPredicate_2
```

3)将项目的导入库文件(.LIB 文件)加入 VC 项目或在 VC 源代码中显式的加入指导编译器编译导入库文件的代码。例如:

```
#pragma    comment(lib, ". \Lib\MyDll. lib")
```

4)把动态链接库拷贝到 VC 程序能够找到的路径下(如系统目录、当前目录等)。

5)为所调用的谓词在 VC 中声明一个 C 函数与之对应。这个被声明的函数同样须遵循一组约定,具体是:①使用调用约定__stdcall;②使用关键字 extern,指明函数在模块外实现;③函数名与谓词名必须相同。如果谓词参数的流程模式为输入模式,那么它在 VC 函数中相应参数的数据类型与论域类型相对应;如果谓词参数的流程模式为输出模式,那么它在 VC 函数中相应参数的数据类型与用该论域类型声明的引用论域相对应(因为 Visual Prolog 在调用谓词时,把未绑定的变量看作为引用变量)。

下面是两个谓词,所带参数的流程模式分别为输入模式和输出模式:

```
procedure myPredicate_1(integer) - (i) language stdcall
proceduremyPredicate_2(integer) - (o)language stdcall
```

它们对应的 VC 函数声明如下:

```
extern int __stdcall myPredicate_1(int);
extern void __stdcall myPredicate_1(int *);
```

上述步骤声明的函数,虽然实现代码在 Visual Prolog 中,但可以像普通函数一样在 VC 代码中使用。

## 5 结束语

以上讨论了 Visual Prolog 谓词与 VC 函数相互调用的方法,在智能应用的开发中,这是十分有用的技术,开发者可通过该方法将 VC 与 Visual Prolog 的优势有效的结合起来,从而使智能应用的开发简单而高效。

(下转第 68 页)