Visão Artificial Para a Indústria

Manual de Desenvolvimento de Módulos de Comunicações

Luis Fonseca Carvalho de Matos

(luis.matos@ua.pt)

Julho de 2007

Índice de conteúdos

1. Advertência.....1

2. O Sistema de Comunicação.....1

3. O Alto nível e a Camada de abstracção.....1

4. A Camada de Abstracção:.....3

5. Registo de comunicações.....5

Anexo I – Funções de Comunicação.....7

Anexo II – Funções auxiliares de IO's.....8

Enumerações.....8

Funções.....9

1. Advertência

Este manual é um guia para expansão de comunicações suportadas da aplicação VAPI – Visão Artificial Para a Indústria.
A expansão de comunicações indica o aumento do número de elementos com que o VAPI consegue comunicar e que o utilizador pode definir.
O leitor deste manual deve estar familiarizado com a linguagem C, estruturas de dados e funções.

2. O Sistema de Comunicação

Figura 2.1 – Esquema da interface de comunicações

Por forma a adquirir capacidades de comunicação, foi desenvolvida uma arquitectura que permitisse que o VAPI interagisse com o que o rodeia. Assim, nasceu um sub-sistema modular, dividido em 3 camadas (figura 1.1):

- baixo nível;
- camada de abstracção;
- alto nível.

O baixo nível designa as bibliotecas fornecidas por fabricantes de hardware e software. O alto nível são funções para inclusão em operações. A camada de abstracção relaciona as 2 e permite ao programador/utilizador desenvolver de forma independente do objecto que está a utilizar para o efeito.

3. O Alto nível e a Camada de abstracção

O alto nível pretende ser um conjunto de funções que permitem ao utilizador / programador:

- reconhecer o hardware através de um endereço único;
- comunicar com o dispositivo de forma directa e simples;
- facilitar a criação de operações e a sua parametrização.

As funções de alto nível são algo funções do género *function(int comAddress, ...)*, em que o *comAddress* é um número inteiro e os restantes campos os dados necessários para a comunicação.

Para que possuam um endereço único, é necessário configurar o elemento de comunicação. Para tal, existe uma "Lista de Comunicações Configuradas" em que a cada elemento é atribuído um valor inteiro.

Para que o utilizador possa configurar um elemento de comunicação, este tem que estar incluído na "Lista de Comunicações Suportadas". Esta lista contém a relação de funções da camada de abstracção existentes.

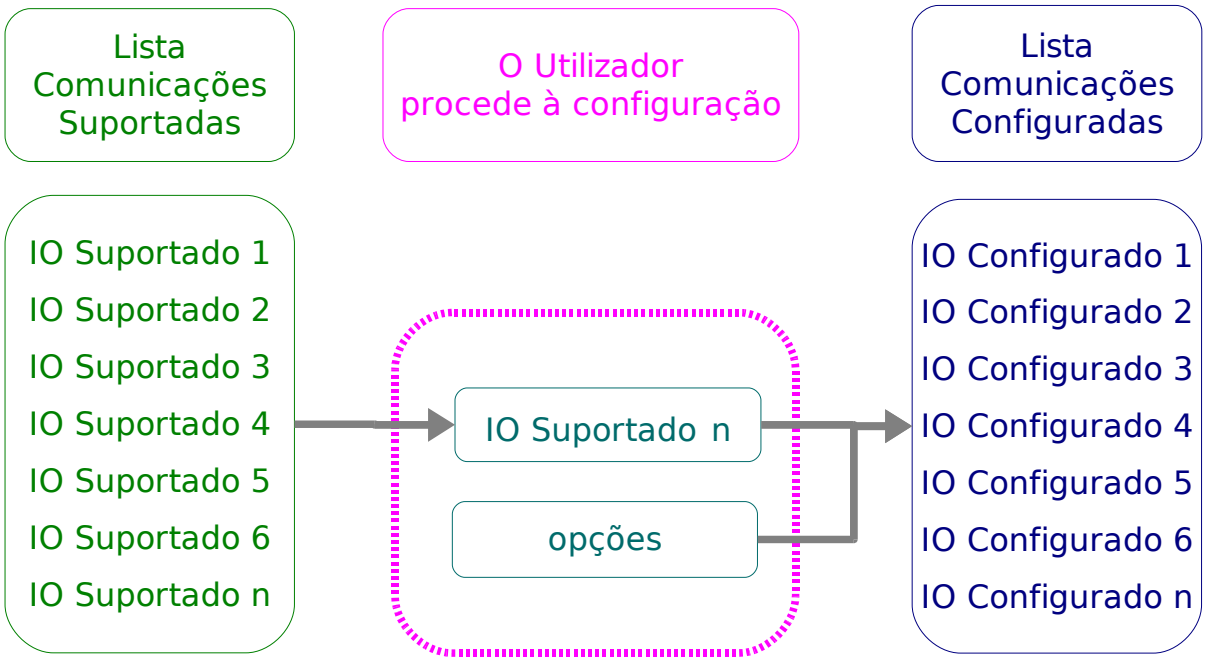

Figura 3.1 – Relação entre Listas: Suportadas e configuradas

A cada função da camada de abstracção corresponde um elemento de comunicação (IO). A relação entre a "Lista de Comunicações Suportadas" e a "Lista de Comunicações Configuradas" é dada pela figura 3.1.

Do lado esquerdo da figura estão as comunicações suportadas. Estas, depois de parametrizadas, consideram-se configuradas.

Exemplo: se existe uma abstracção para se conectar a uma base de dados, é necessário definir, pelo menos, o ***IP*** do computador onde se encontra a base de dados, o seu **nome** e **password**.

O utilizador define estes parâmetros e configura a comunicação. Se esta for a 1ª comunicação configurada, recebe o número ***0*** como endereço na aplicação.

Devido a este esquema de funcionamento, o programador apenas questiona o utilizador, através da operação que desenvolveu, o endereço de comunicação, descurando os restantes elementos.

Dada a camada de abstracção, o programador (de operações) pode também aceder às comunicações de um modo facilitado. Utiliza a função *v_input_char_int(int address, char*str)* para efectuar um ***query*** à base de dados que devolve um valor inteiro.

Exemplo:

V_input_char_int (0,"select id from table where name=Luis"),

O campo ***id*** *é* um campo de valores inteiros.

Para verificar as funções existente, consulte o anexo I.

4. A Camada de Abstracção:

Para obter as funções de Alto Nível, tem que ser criada uma forma de não de depender directamente das bibliotecas dos fabricantes. Cada fabricante e cada elemento de comunicação requerem configurações e acessos diferentes.

A Camada de Abstracção é, ssim, responsável pela tradução entre a camada de baixo nível (bibliotecas de fabricantes e/ou acesso directo ao elemento de comunicação) e a de alto nível. A sua forma é similar às operações:

*void * function (vOptions * Options, vpointer Data)*

Figura 4.1: Prototipo da função da camada de abstracção

Data é um ponteiro para os parâmetros necessários à função, que pode ser de qualquer tipo. Após o desenvolvimento desta função é necessário efectuar o seu registor na aplicação, para que seja reconhecida e disponibilizada na "Lista de Comunicações Suportada".

Deve ser indicado:

- O nome do elemento de comunicação
- Descrição
- Via de comunicação (input, output, vários inputs, etc)
- Tipo de variável de entrada (Tipo do ponteiro Data)
- Tipo de variável de saída (Tipo do retorno)

As opções são definidas aquando o registo. As opções podem assumir 3 tipos:

- inteiros;
- decimais;
- caracteres (strings).

O ponteiro ***Data*** serve qualquer propósito em termos de inputs para a ***function.*** As funções de alto nível seguem o tipo de dados de *input* e *output*. Se um elemento de

comunicação tem de input *int* e output *int*, então a função de alto nível a utilizar será v*_input_int_int()*.

Quanto à camada de abstracção, uma função exemplo é a utilização de um ficheiro como input digital. O objectivo é ler o primeiro caracter de um ficheiro. Se este for "0", será considerado como FALSO, se for 1 será VERDADEIRO:

```
vBoolean *
FileAsDigitalInput (vOptions * Options, vpointer Data)
{
	//Obter o caminho para o ficheiro das opções.
	char *filename = vapiOptionsGetChars (Options, 0);
	int fileReturn;			//Inteiro a obter do ficheiro
	vBoolean *output;		//Definir o output
	FILE *handler;			//O handler do ficheiro

	//Caso O ficheiro não exista ou tenha um quelquer tipo de erro
	if ((handler = fopen (filename, "r")) == NULL)
	{
		return NULL;
	}
	//Obter o primeiro caracter do ficheiro (só esse interessa)
	fileReturn = fgetc (handler);
	fclose (handler);
	switch (fileReturn)
	{
	//Caso Seja End Of File Occoreu um erro
	case EOF:
	{
		return NULL;
	}
	//Caso Seja 1, é verdadeiro
	case (int) '1':
	{
		output = malloc (sizeof (vBoolean));
		output[0] = vTRUE;
		return output;
	}
	//Caso seja 0 é falso
	case (int) '0':
	{
		output = malloc (sizeof (vBoolean));
		output[0] = vFALSE;
		return output;
	}
```

```
//caso não seja nem 1 nem 0, então não interessa (erro);
        default:
                return NULL;
        }
}
```

Figura 4.2 – Relação entre Listas: Suportadas e configuradas

De notar que a função retorna *NULL* caso obtenha erro. O ponteiro **Data** não é utilizado, foi declarado como do tipo *NULL*.

5. Registo de comunicações

O registo de novas comunicações é efectuado através de uma função de registo, constituída por 2 elementos: declaração e definição.

- declaração consiste na inicialização da operação, utilizando a função *vapiIOListAddIO*;
- definição consiste em definir propriedades cuja parametrização é necessária.

De momento estes passos são efectuados na função *vapiIORegisterSupportedHw()* no ficheiro *vapiIOData.c*. Para se registar um novo elemento de comunicação basta adicionaros passos descritos em cima, no final da função.

Exemplo: Adicionar a função *FileAsDigitalInput* à “Lista de Comunicações Suportadas”.

```
position =
        vapiIOListAddIO (GlobalIOList, "Ficheiro Como Input Digital",
                        "Ficheiros",
                        "Considerar um ficheiro como input digital",
                        vInput, vNULL, vBool,
                        (vIOFunction *) & FileAsDigitalInput);

        vapiIOPropertySetChar (position, "Caminho Para o Ficheiro", " ");
```

Figura 5.1 – Exemplo de registo de uma função de abstracção

A N E X O S

Anexo I – Funções de Comunicação

- vBoolean *
v_input_NULL_boolean (int IOindex)

- int *
v_input_str_int (int IOindex, const char *str)

- vBoolean
vIOCheckIO (vIOConfigured *IO, int IOtype, int InData, int OutData)

Anexo II – Funções auxiliares de IO's

Enumerações

enum

- IOData {
  - vStr
  - vInt
  - vDouble
  - vBool
  - vPointer
  - vNULL
    }
  -
- IOtype {
  - vInput
  - vInputN
  - vOutput
  - vOutputN
  - vIOBoth
  - vIOBothN
    }

Funções

- int

vapiIOListAddIO (vIOList *IOList, const char *IOname, const char *GroupName, const char *IOdescription, int IOtype, int InData, int OutData, vIOFunction IOFunc)

- void

vapiIOPropertySetChar (int IOPosition, const char *name, const char *description)

- void

vapiIOPropertySetInt (int IOPosition, const char *name, const char *description, int min, int max, int default_value, int scale)