Controle Automatizado de Bombas para Laboratório

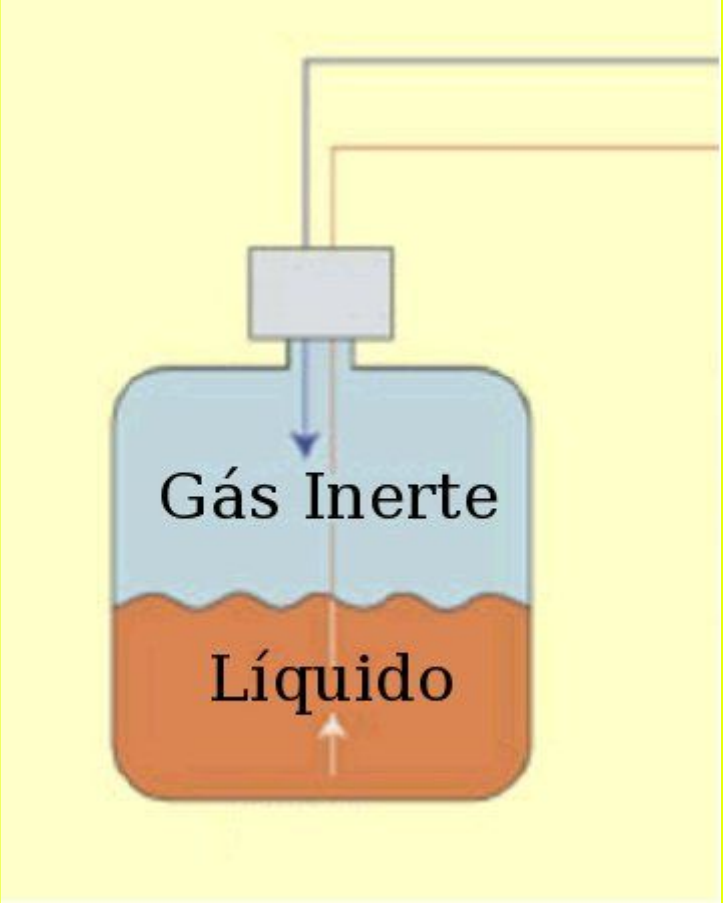
Bombas

Para sistemas de análise em fluxo existem basicamente três tipos de sistemas para o bombeamento de líquidos que oferecem fluxo constante (sem pulsos):

-frasco pressurizado,
-bomba peristáltica,
-bomba de seringa.

Frasco Pressurizado

O princípio de bombeamento com frasco pressurizado consiste na injeção de um gás inerte em um frasco contendo o líquido que se deseja bombear, forçando a saída do líquido pelo aumento da pressão interna.



Frasco Pressurizado

Este conceito foi aplicado de forma criativa com o uso de uma simples bomba de aquário conforme publicado no artigo: “Propulsor pneumático versátil e isento de pulsação para sistemas de análise em fluxo.”
Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-40422001000600017

Frasco Pressurizado com Bomba de Aquário

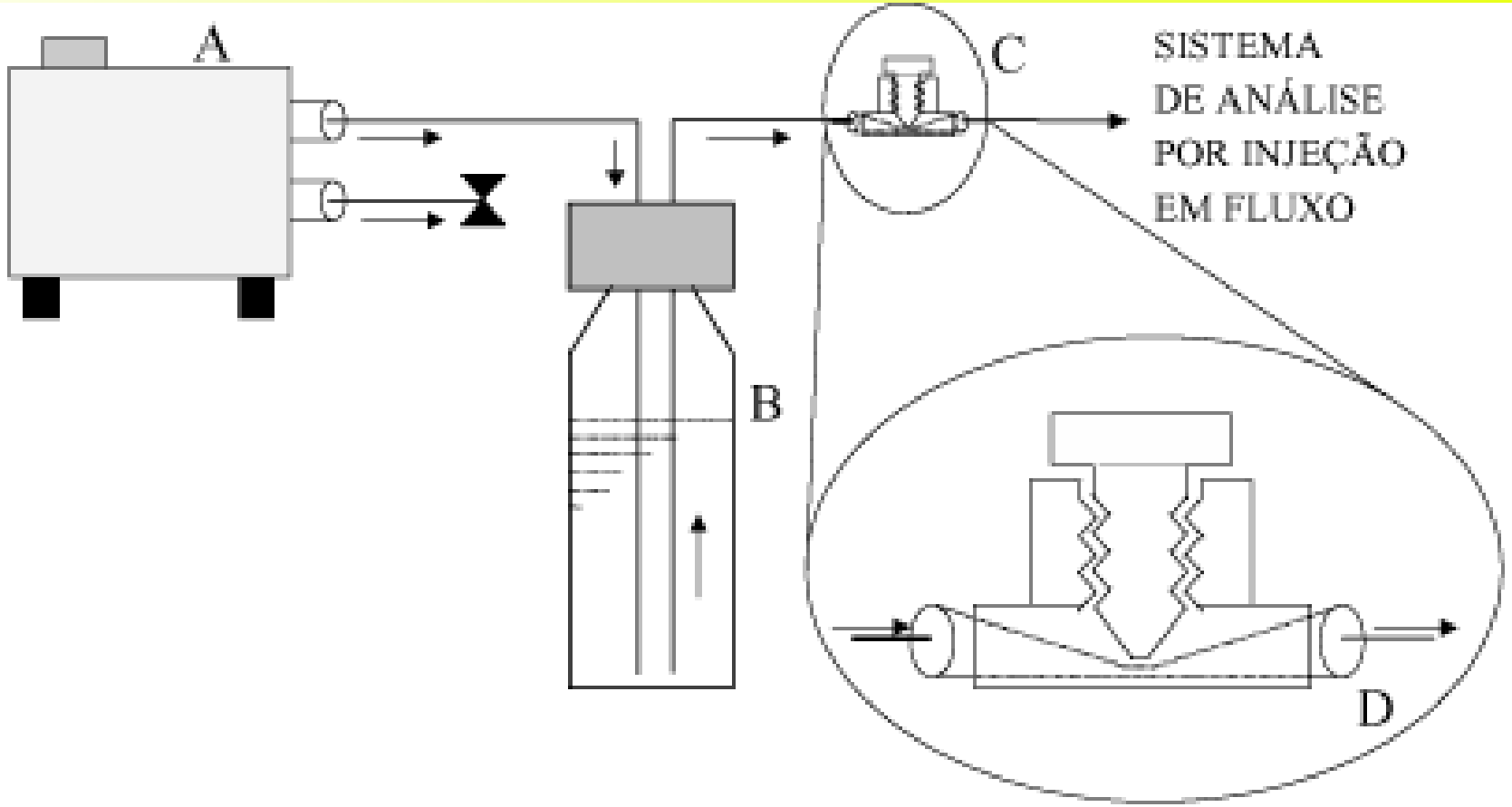
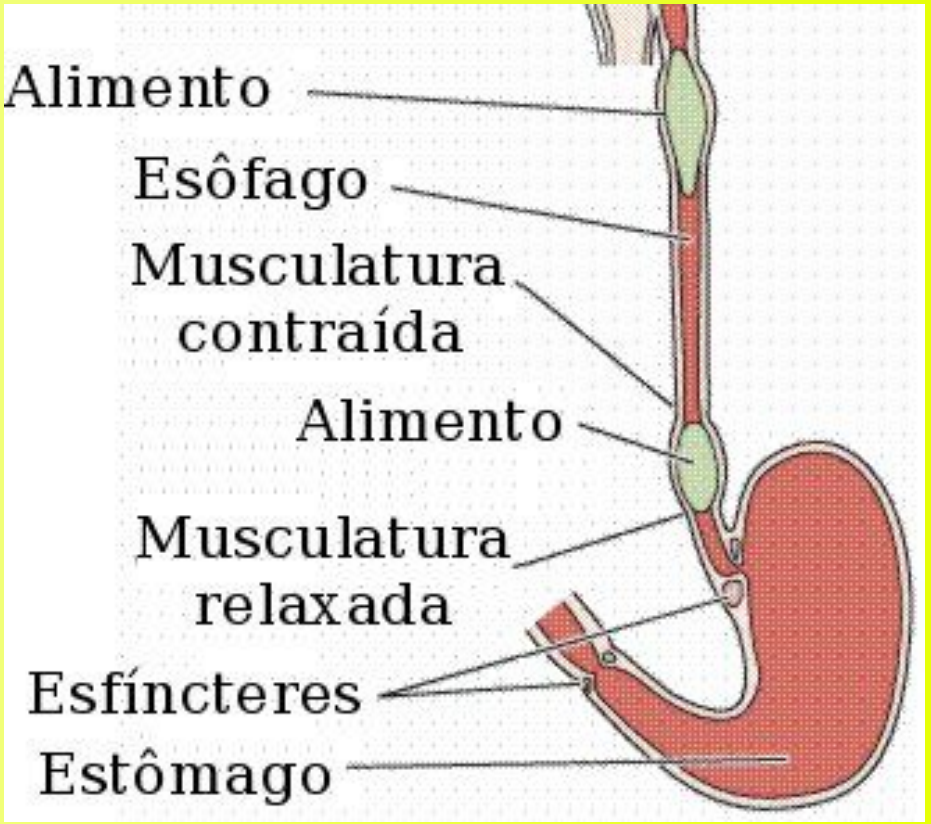


Figura 1. *Representação esquemática do sistema de análise em fluxo utilizando bomba de aquário para a propulsão de líquidos. (A) Mini-compressor de ar (dimensões: ~12 x 6 x 6 cm); (B) Reservatório de eletrólito; (C) Válvula reguladora de vazão; (D) Tubo flexível de Tygon®.*

Sistema Peristáltico

O sistema peristáltico é uma das mais antigas formas de bombeio, inventada pela própria natureza.

Do esôfago ao intestino, nosso aparelho digestivo funciona pelo mesmo princípio



Sistema Peristáltico - Componentes

1. Cabeçote para guiar o tubo flexível
2. Tubo flexível,
3. Roletes montados no braço porta-rolete, que pressionam o tubo flexível contra o cabeçote
4. Eixo central da bomba, que gira o braço porta-rolete
5 e 6. Conexões para o tubo flexível (entrada e saída)
7.Motor elétrico, engrenagens de redução, eletrônica de controle com regulagem, leds, etc.

Bombas de Seringa

Bombas de seringa não são muito usadas em sistemas de análise em fluxo, provavelmente por razões de custo.

A bureta automática é um exemplo de bomba de seringa muito comum nos laboratórios.

Bombas Peristálticas - Modelos

Exemplos de bombas peristálticas para uso em laboratório.

Bomba Masterflex – Mod. 7550-30

Os modelos 7550-30 e 7550-50 da Masterflex permitem o uso de diferentes cabeçotes para diferentes vazões e controle remoto via porta serial RS232C interligada (ou não) com outras unidades (bombas, balanças e misturadores) compatíveis com a configuração “daisy-chain”.

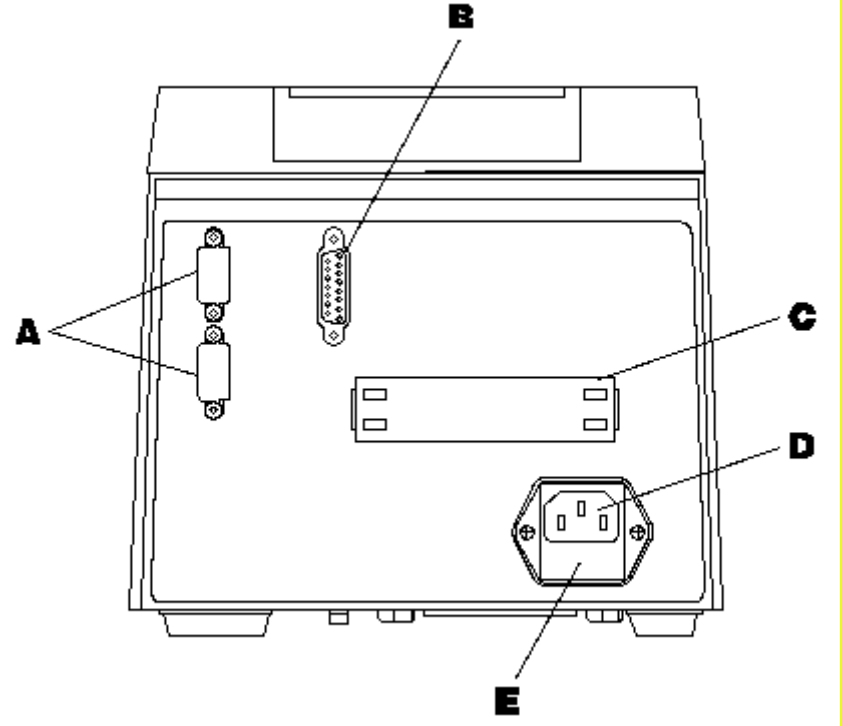
http://www.coleparmer.com/catalog/manual_pdfs/07523-60,-70.pdf

Bomba Masterflex – Mod. 7550-30

À esquerda, o uso do cabeçote para pequenas vazões (7519-15), adequado para análise em fluxo, e à direita o cabeçote para altas vazões.

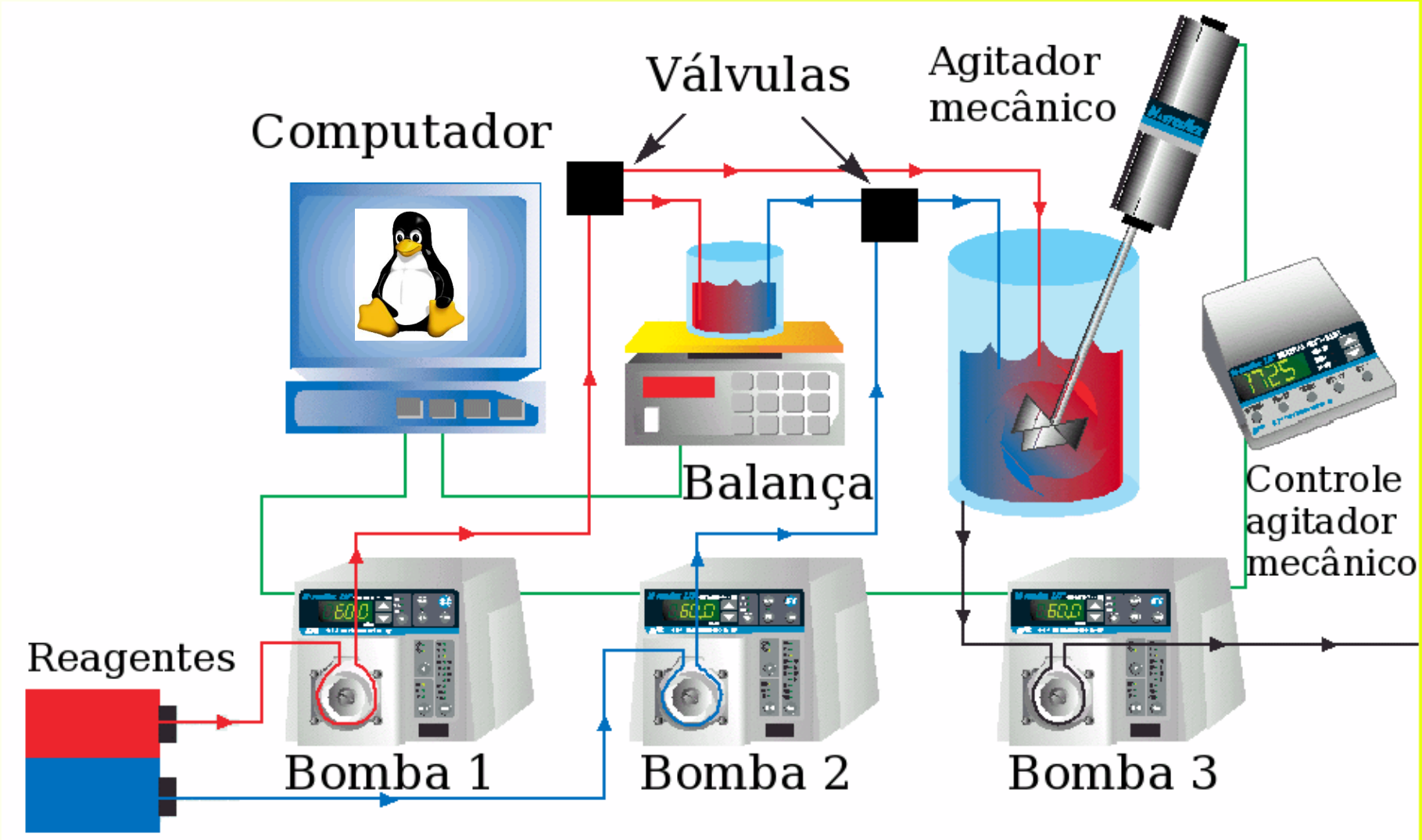
Bomba Masterflex – Mod. 7550-30

A-Conectores RS232 para ligação a um computador e outros equipamentos com o mesmo recurso.
B-Conector para sincronizar o funcionamento com outros dispositivos mediante abertura e fechamento de contatos elétricos (não é comunicação serial!).



http://www.coleparmer.com/catalog/manual_pdfs/07523-60,-70.pdf

Daisy-Chain



Daisy-chain significa dois ou mais equipamentos ligados em série na mesma porta serial recebendo e repassando os comandos enviados pelo computador de controle.

Desenvolvimento

do

Programa de Controle

(Linguagem Tcl/Tk)

Pinagem

DB-25 PLUG ON CONTROL COMPUTER
Pin 2 - Transmitted data to satellite
Pin 3 - Received data from satellite
Pin 5 - Clear to send—RTS from satellite
Pin 7 - Ground

DB-9 PLUG "AT type" ON CONTROL COMPUTER (DTE) AND SATELLITE
Pin 3 - Transmitted data to satellite
Pin 2 - Received data from satellite
Pin 8 - Clear to send—RTS from satellite
Pin 5 - Ground

DB-9 SOCKET ON SATELLITE (DCE)
Pin 3 - Receive signal from the computer
Pin 2 - Transmit signal to the computer
Pin 5 - Ground
Pin 8 - Request to send—(RTS) to the computer

The serial lines between units will be passed from unit to unit by a hardware buffer on the input and connecting it directly to the output driver through a hardware gate. This way any output only sees one input load. If power is turned off on any pump drive, all drives below it in the daisy-chain cannot communicate.

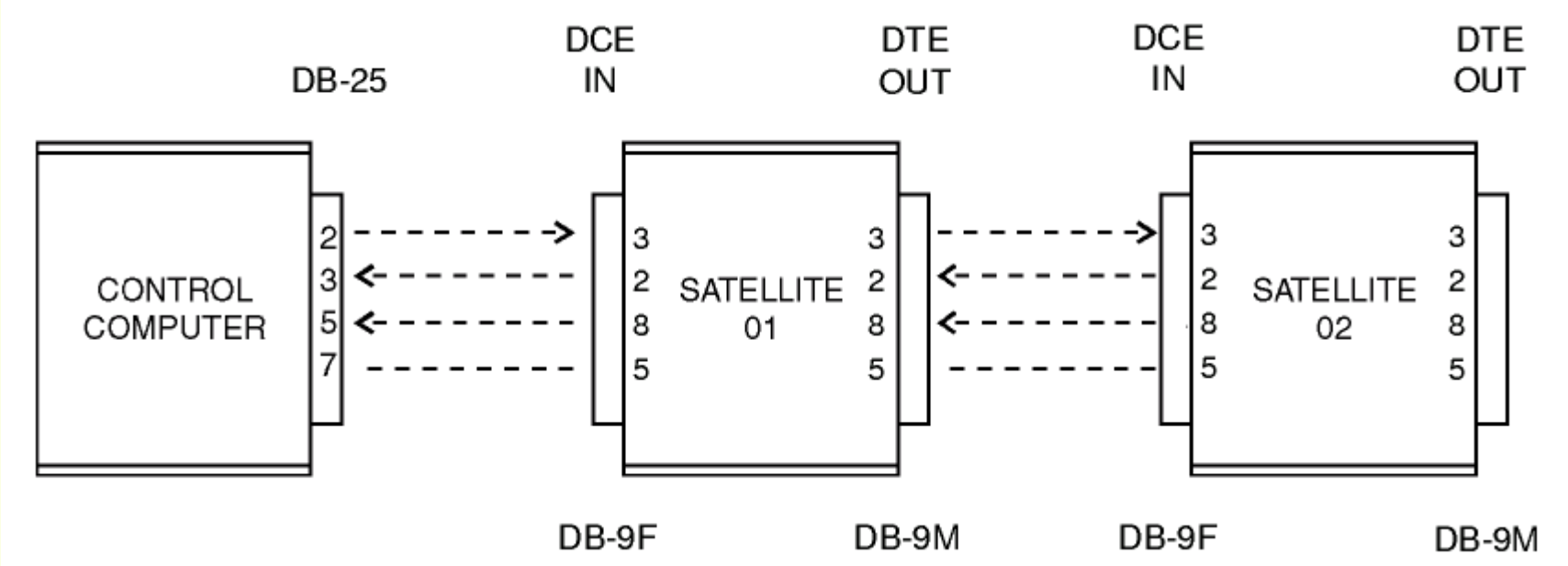
Informações contidas no manual do equipamento com respeito ao nome dos pinos do conector serial.

Cabo Serial

- **Pinagem do cabo serial DB9:**
- **PC Bomba**
- **(Rx) 2 <- 2 (Tx)**
- **(Tx) 3 - > 3 (Rx)**
- **(Gnd) 5 - 5 (Gnd)**
- **(CTS) 8 <- 8 (RTS)**

Pinagem do cabo para comunicação serial.

Configuração “Daisy-Chain”



Os comandos enviados pelo computador são repassados diretamente pelos equipamentos (satélites) seguintes até o último equipamento da série. No entanto cada equipamento tem a capacidade de bloquear a comunicação com os demais assumindo o privilégio de se comunicar com o computador. Todos devem estar ligados para que os comandos possa alcaçar toda a série.

Configuração da Porta Serial

1.2 - SERIAL DATA FORMAT

The serial data format is full duplex (simultaneously transmit and receive), 1 start bit, 7 data bits, one odd parity bit, and one stop bit at 4800 bits per second. All data transmitted will consist of characters from the standard ASCII character set.

Note: Odd parity is defined such that the sum of the eight individual bits is an odd number (1, 3, 5 or 7).

O manual informa que a porta serial deve ser configurada para 4800 baud, paridade “o”, odd (ímpar), 7 bits de dados e 1 bit de parada.

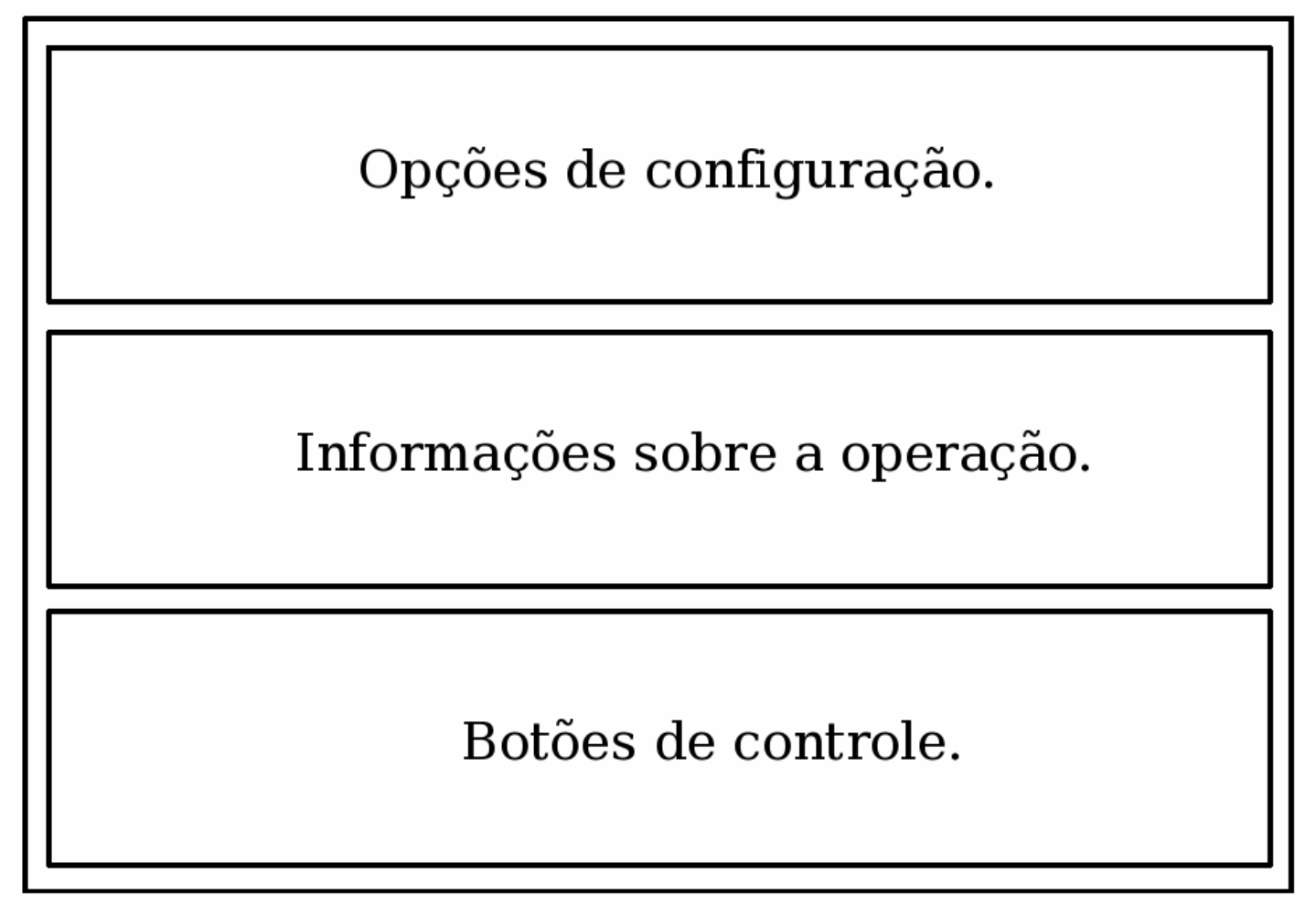
Comandos para Controle da Bomba

COMMAND CHARACTERS FROM CONTROL COMPUTER TO PUMP		PARAMETER FIELD
A	Request auxiliary input status	none
B	Control auxiliary outputs when G command executed	xy, x = aux1, y = aux2, 0 = off, 1 = on
C	Request cumulative revolution counter	none
E	Request revolutions to go	none
G	Go Turn pump on and auxiliary output if preset	none = run for number of revolutions set by V command 0 = continuous run until Halt command
H	Halt (turn pump off)	none
I	Request status data	none
K	Request front panel switch pressed since last K command	none
L	Enable local operation	none
O	Control auxiliary outputs immediately without affecting drive	xy, x = aux1, y = aux2, 0 = off, 1 = on
R	Enable remote operation	none
S	Set motor direction and RPM	+xxx.x, -xxx.x, +xxxx, -xxxx + = CW, –=CCW
S	Request motor direction & RPM	none
U	Change satellite number	nn new satellite number
V	Set number of revolutions to run	xxxxx.xx
Z	Zero revolutions to go counter	none
Z	Zero cumulative revolutions	0
<CAN>	Terminates line of data up to and including STX (used primarily for keyboard input)	none
<ENQ>	Enquire which satellite has activated its RTS line	none

Table 1 - Pump satellite commands

Em destaque os comandos utilizados.

Criando a Interface Gráfica



Vamos pensar inicialmente em uma tela única contendo três frames (campos):
1-ajuste dos parâmetros de configuração,
2-informações sobre a operação
3-botões de controle.

Criando a Interface Gráfica

Parâmetros de configuração:

-**Porta serial**: para facilitar o usuário podemos criar o nome do dispositivo serial automaticamente dependendo do sistema utilizado. Resta ao usuário apenas selecionar o número.

-**Sentido de rotação** da bomba: o sentido pode ser definido apenas como horário ou anti-horário restando ao usuário posicionar as conexões de forma adequada

-**Modelo da bomba**: esta opção irá definir quais os módulos que deverão ser usados

-**Tubo utilizado**: cada tubo deverá ter um valor de vazão/RPM que será usado para calcular o tempo de operação e a velocidade de rotação (RPM) para bombear o volume solicitado pelo usuário com a vazão especificada

-**Volume a ser bombeado**

-**Vazão**

Interface Gráfica – Primeiras linhas

```
#!/usr/bin/env wish

wm title . "Bomba Peristáltica"

if {$tcl_platform(platform) == "unix"} {
        set porta_serial "/dev/ttyS"
    } else {
        set porta_serial "com"
    }
```

A linha “wm title” apenas cria um nome para a janela principal.
Em seguida o programa verifica o conteúdo da variável tcl_plataform(plataform), que é um vetor associativo (array) interno da Tcl com informações sobre o sistema onde o script está rodando. Se for “unix” então a variável porta_serial recebe o conteúdo “/dev/ttyS” senão “com”.

Entendendo Listas e Arrays

Um exemplo, criando uma lista chamada “amostra” contendo as informações para a identificação de uma amostra, ou seja: nome da análise, dia da amostragem e hora da amostragem.
set amostra { “sulfeto” “hoje” “agora” }
Para exibir na tela o nome da análise a ser feita, o dia e a hora:
puts [lindex $amostra 0]
puts [lindex $amostra 1]
puts [lindex $amostra 2]

Fazendo o mesmo com um array (vetor associativo):
set amostra(nome_analise) “sulfeto”
set amostra(dia_amostragem) “hoje”
set amostra(hora_amostragem) “agora”
Para escrever na tela o nome da análise, o dia e a hora:
puts $amostra(nome_analise)
puts $amostra(dia_amostragem)
puts $amostra(hora_amostragem)

Entendeu? :-)

Interface Gráfica – Ajuste, Info e Botões

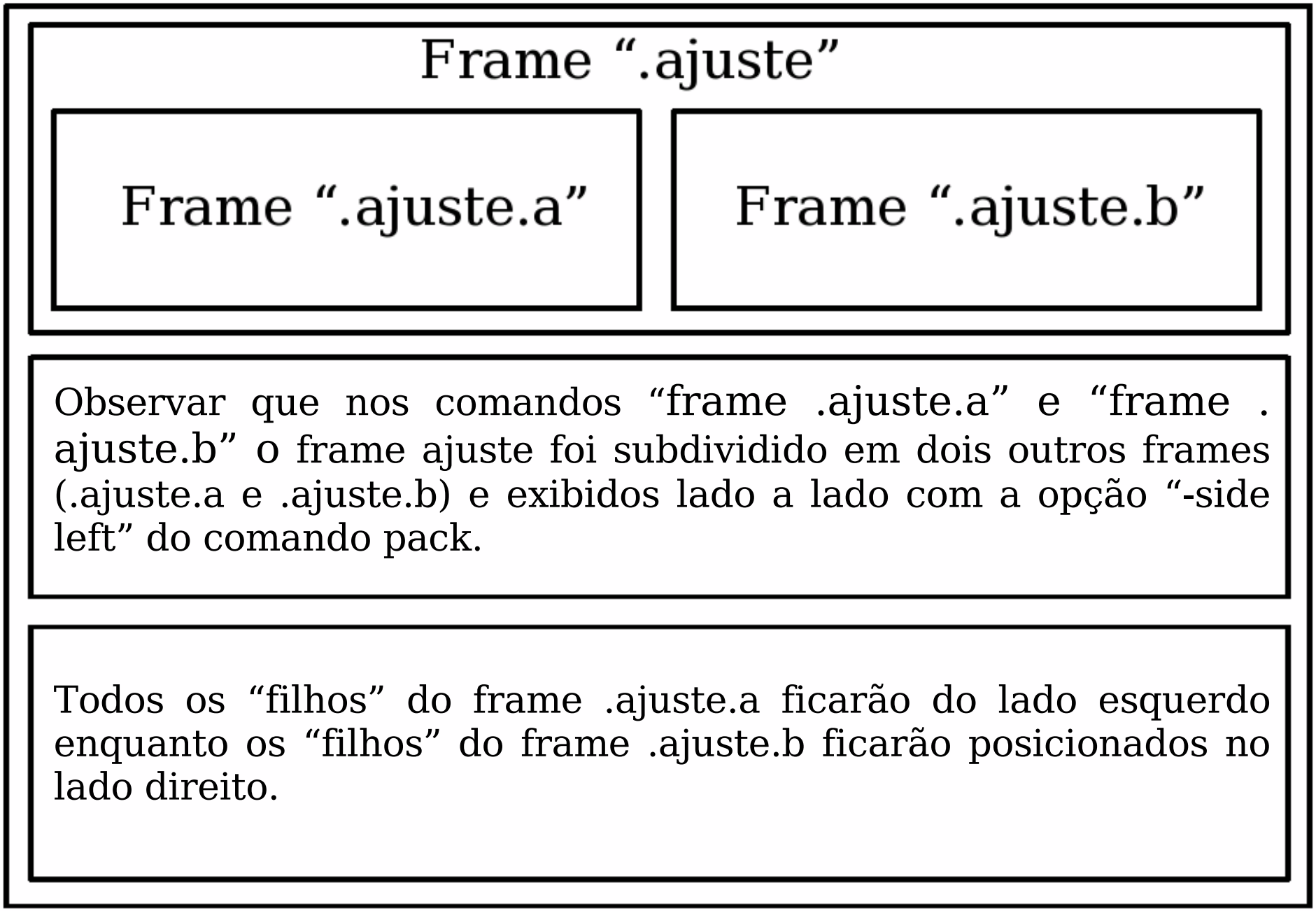
```
frame .ajuste -padx 5 -pady 5
frame .ajuste.a
frame .ajuste.b
labelframe .info -text "Informações" -padx 5 -pady 5
frame .botoes -padx 5 -pady 5
pack .ajuste.b .ajuste.a -side left
pack .ajuste -side top
pack .info
pack .botoes -expand yes -fill both
```

Incluir os comandos acima para a criação dos três frames, “ajuste”, “info” e “botoes” que vão exibir respectivamente as opções de configuração, as informações e os botões de controle.

O comando labelframe é um frame que exibe o nome do frame especificado pela opção “text”.

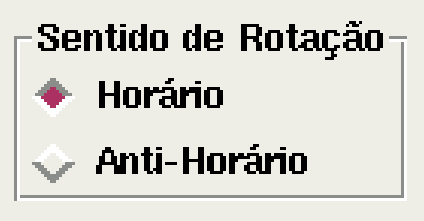
As opções “padx” e “pady” alteram o espaçamento horizontal e vertical do frame respectivamente.

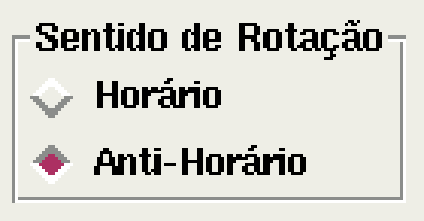
Interface Gráfica – Detalhe do Frame Ajuste



Interface Gráfica – Sentido de Rotação

```
labelframe .ajuste.a.sentido -text "Sentido de Rotação"
radiobutton .ajuste.a.sentido.horario \
-text "Horário" -variable sentido_rotacao  -value "+"
radiobutton .ajuste.a.sentido.antihorario \
-text "Anti-Horário" -variable sentido_rotacao -value "-"
pack .ajuste.a.sentido
pack .ajuste.a.sentido.horario -anchor w
pack .ajuste.a.sentido.antihorario -anchor w
```



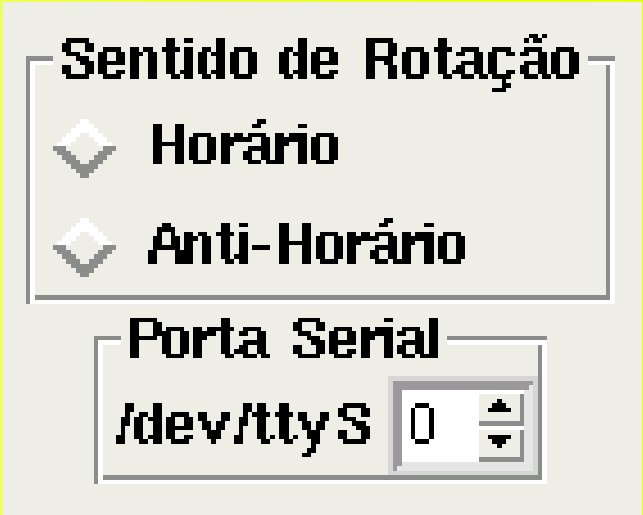


Criando o frame com o título “Sentido de Rotação” dentro do qual foram criados 2 radiobuttons. Observe que os dois radiobuttons estão configurados para a mesma variável “sentido_rotacao” com a opção “variable” e por isso estão vinculados, ou seja, apenas um dos dois poderá ficar selecionado.
Quando o radiobutton de nome “Horário” for selecionado a variável “sentido_rotação” recebe o valor “+” e quando o radiobutton “Anti-Horário” for selecionado, a variável sentido_rotacao” recebe o valor “-”

Interface Gráfica – Porta Serial

```
labelframe .ajuste.a.porta_serial -text "Porta Serial"
label .ajuste.a.porta_serial.rotulo -text $porta_serial
spinbox .ajuste.a.porta_serial.entrada -from 0 -to 30 \
-bg white  -textvariable num_porta_serial -width 2

pack .ajuste.a.porta_serial
pack .ajuste.a.porta_serial.rotulo -side left
pack .ajuste.a.porta_serial.entrada -side right
```

Criando o frame com o título “Porta Serial” dentro do qual foi criado um spinbox. O spinbox facilita a atribuição de valores a variáveis, neste caso valores numéricos de (-from) 0 a (-to) 30. A opção “bg” e “width” definem respectivamente a cor de fundo e a largura do spinbox.
A opção “textvariabel” define que a variável “num_porta_serial” irá receber o valor numérico que o usuário selecionar com o spinbox.



Interface Gráfica – Modelo da Bomba

```
label .ajuste.b.bomba_rotulo -text "Modelo da Bomba"
spinbox .ajuste.b.bomba_entrada  \
-values [array names modelo_bomba] -bg white \
-textvariable bomba_em_uso
grid .ajuste.b.bomba_rotulo -row 0 -column 0 -sticky w
grid .ajuste.b.bomba_entrada -row 0 -column 1 -sticky ew
```

Nos comando acima estamos criando um label e um spinbox para configurar o modelo da bomba que será utilizado.
O objetivo é permitir que um único programa possa ser usado para controlar diferentes modelos de bombas mediante o carregamento dos respectivos módulos (drivers).

Entendendo o Comando “grid”

```
label .ajuste.b.bomba_rotulo -text "Modelo da Bomba"
spinbox .ajuste.b.bomba_entrada  \
   -values [array names modelo_bomba] -bg white \
   -textvariable bomba_em_uso
grid .ajuste.b.bomba_rotulo -row 0 -column 0 -sticky w
grid .ajuste.b.bomba_entrada -row 0 -column 1 -sticky ew
```

O “grid” é um gerenciador de posição assim como o “pack”, mas difere deste por organizar as janelas/widgets em uma grade “invisível”. As janelas/widgets são dispostas na tela usando as opções “row” (linha) e “column” (coluna).

Você não precisa se preocupar com quantas linhas e colunas existem em uma janela, o “grid” cuida disso pra você. :-)

A opção “sticky” é usada para “esticar” o widget/janela nas direção dos pontos cardeais: n (norte), s (sul), e (leste), w (oeste) ou a combinação destes.

A opção “sticky” irá reajustar as dimensões do widget desconsiderando a opção “width” que porventura tenha sido usada na criação do widget.

Array (Vetor associativo) “modelo_bomba”

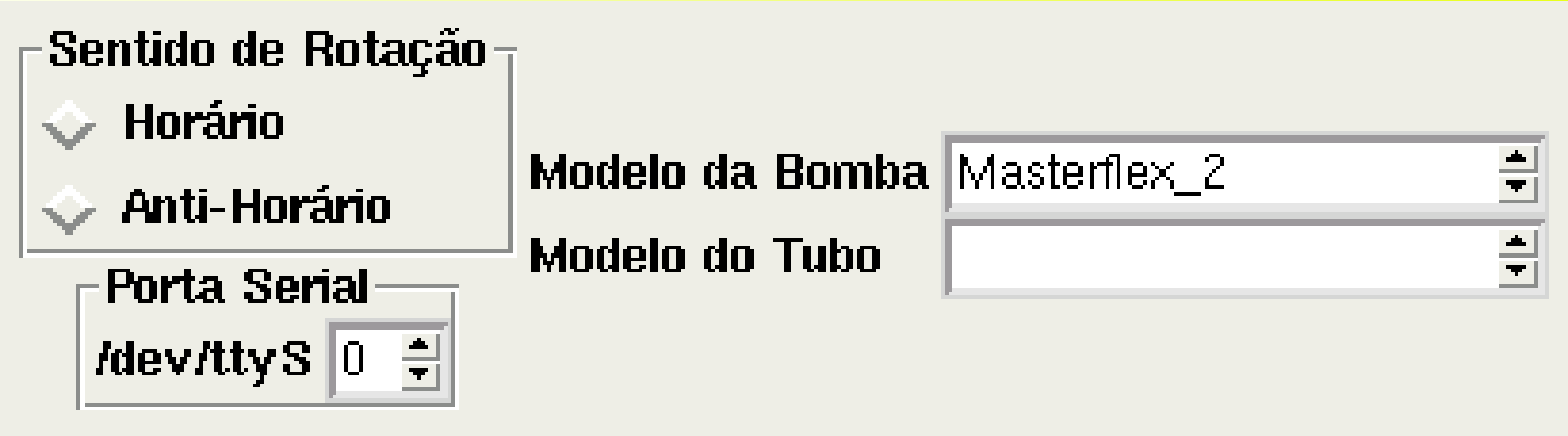
```
set modelo_bomba(Masterflex) "masterflex.tcl"
set modelo_bomba(Ismatec) "ismatec.tcl"
set modelo_bomba(Alitea_S2) “alitea_s2.tcl
```

Estes comandos criam a variável “modelo_bomba” do tipo array para armazenar os diferentes modelos de bombas disponíveis.
O conteúdo desta variável será usado mais tarde para carregar os respectivos módulos para cada modelo de bomba contendo os comandos específicos.

Interface Gráfica – Modelo do Tubo

```
label .ajuste.b.tubo_rotulo -text "Modelo do Tubo"
spinbox .ajuste.b.tubo_entrada  \
    -values $modelo_tubo -width 10 \
    -textvariable tubo_em_uso -bg white
grid .ajuste.b.tubo_rotulo -row 1 -column 0 -sticky w
grid .ajuste.b.tubo_entrada -row 1 -column 1 -sticky ew
```

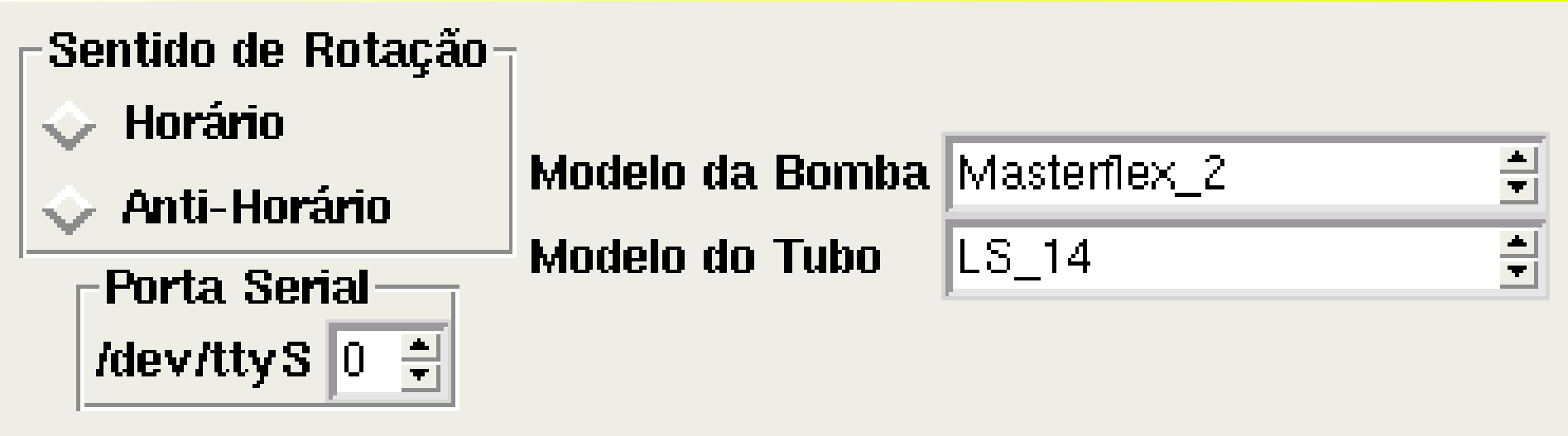
Estes comandos criam a variável “modelo_tubo” do tipo array para armazenar os diferentes modelos de tubo disponíveis e a constante (vazão/rpm) para cada tubo.
O conteúdo desta variável será usado mais tarde para converter os valores de volume solicitados pelo usuário em RPM para os comandos da bomba.



List “modelo_tubo”

```
set modelo_tubo { LS_13 LS_14 }
```

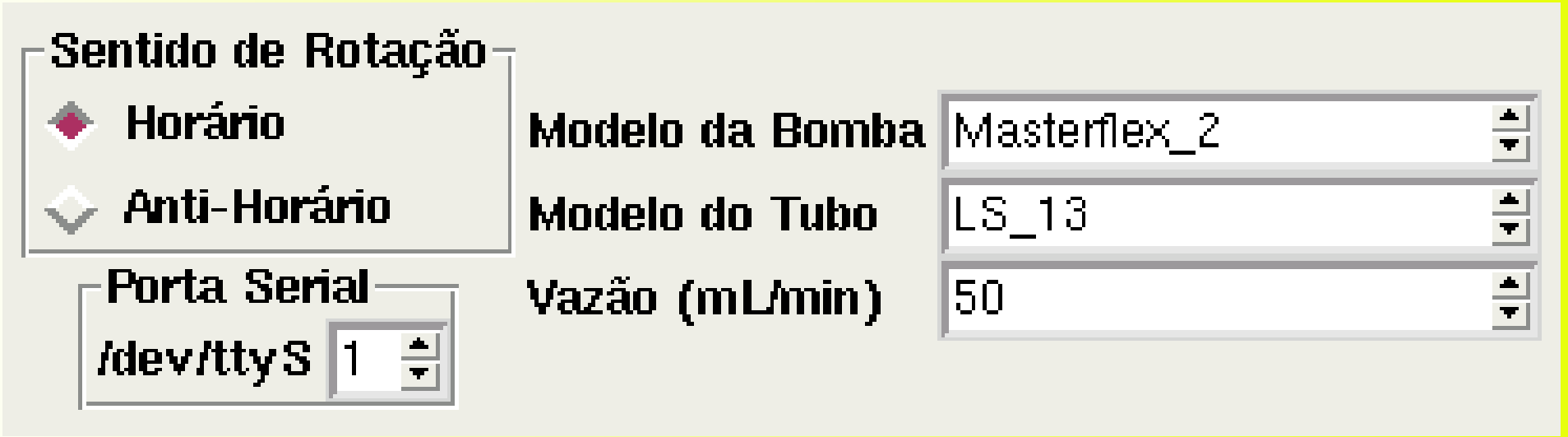
Este comando cria a variável do tipo lista “modelo_tubo” para armazenar os modelos de tubo.
Dentro dos módulos que serão criados para cada modelo de bomba, deverão estar incluídos os valores correspondentes de vazão/rpm. A constante vazão/rpm permitirá calcular a velocidade de rotação e o tempo de operação para bombear o volume solicitado com a vazão especificada pelo usuário.



Interface Gráfica - Vazão

```
label .ajuste.b.vazao_rotulo -text "Vazão (mL/min)"
spinbox .ajuste.b.vazao_entrada  -from 0 -to 200 \
   -increment 5 -bg white  -textvariable vazao -width 4
grid .ajuste.b.vazao_rotulo -row 1 -column 0 -sticky w
grid .ajuste.b.vazao_entrada -row 1 -column 1 -sticky ew
```

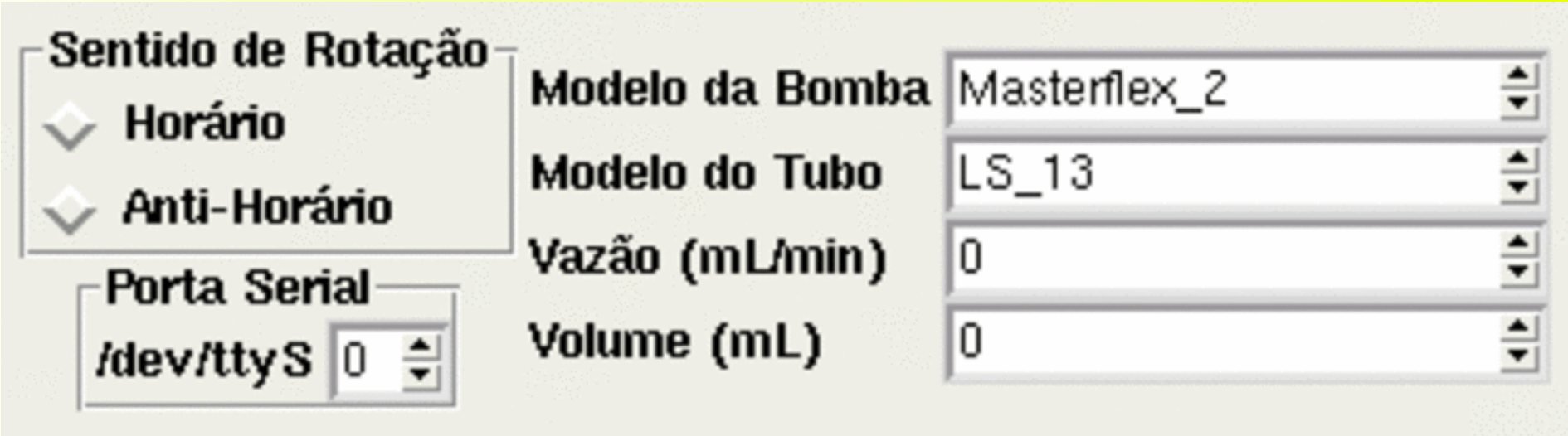
Criando o spinbox ao lado do label "Vazão (ml/min) para configurar a vazão na qual a bomba deve operar.
Observe que agora posicionamos o “label” e o “spinbox” na linha 1 (row).



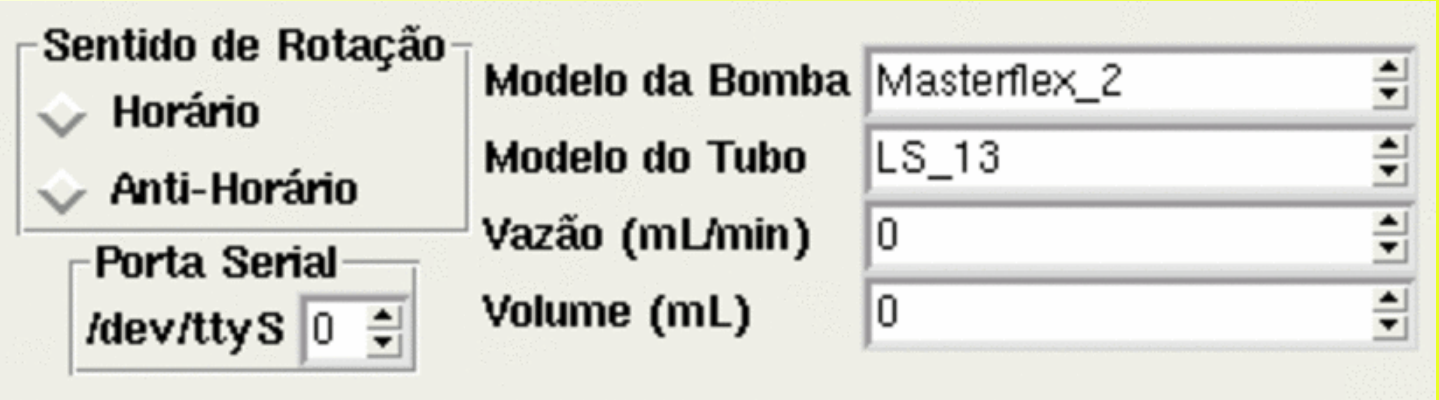
Interface Gráfica - Volume

```
label .ajuste.b.volume_rotulo -text "Volume (mL)"
spinbox .ajuste.b.volume_entrada  -from 0 -to 1000 \
-increment 10 -bg white -textvariable volume -width 4

grid .ajuste.b.volume_rotulo -row 3 -column 0 -sticky w
grid .ajuste.b.volume_entrada -row 3 -column 1 -sticky ew
```

Criando o spinbox ao lado do label "Volume (ml) para configurar o volume que a bomba deve bombear.
O volume que for especificado ficará armazenado dentro da variável “volume”.

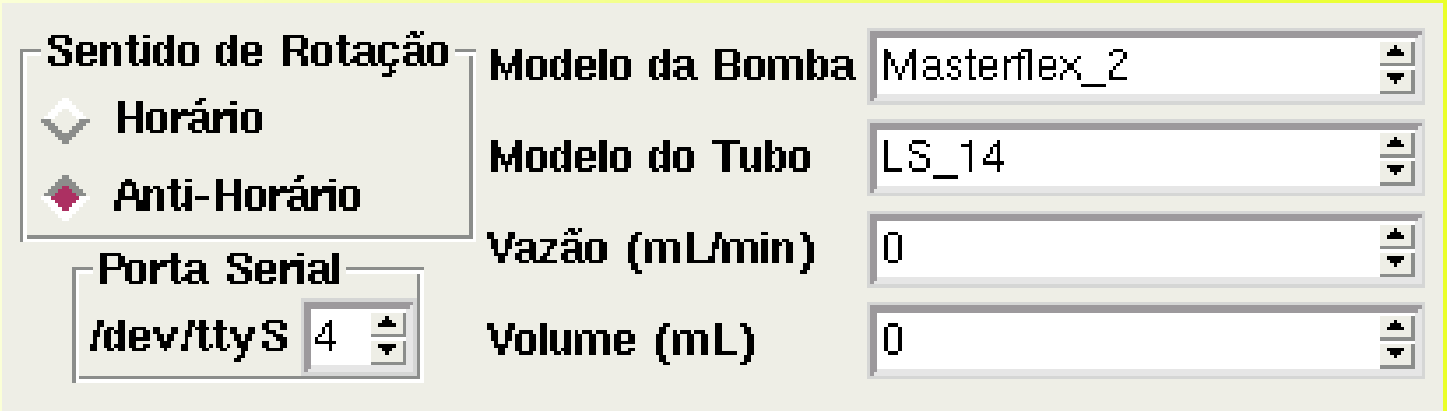


Interface Gráfica – Espaçamento

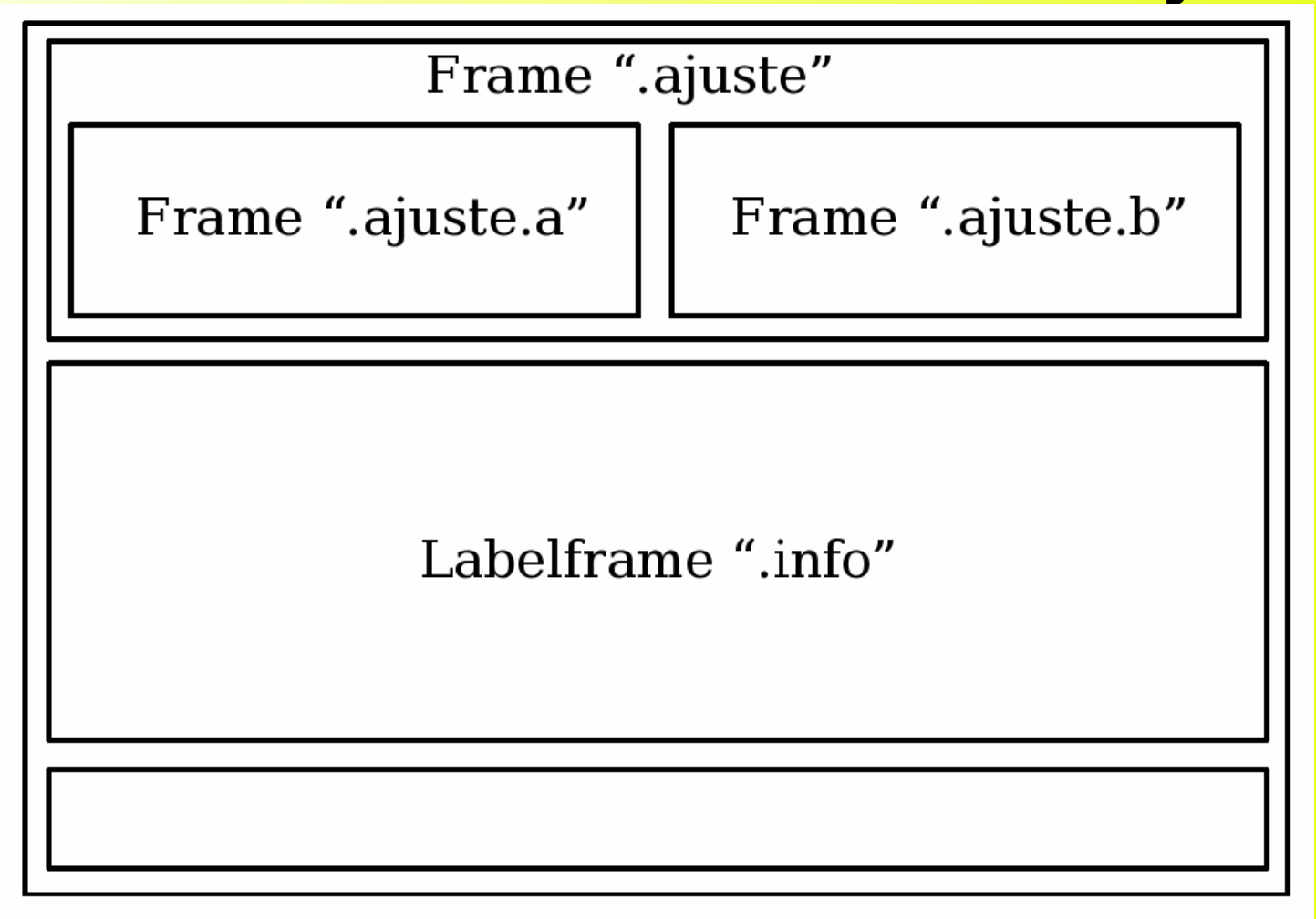


Apenas para melhorar o espaçamento entre as opções de configuração, vamos incluir a opção “pady”, com o valor 2, nos comandos grid dos spinbox.

```
grid .ajuste.b.bomba_entrada -row 0 -column 1 -sticky ew -pady 2
grid .ajuste.b.tubo_entrada -row 1 -column 1 -sticky ew -pady 2
grid .ajuste.b.vazao_entrada -row 2 -column 1 -sticky ew -pady 2
grid .ajuste.b.volume_entrada -row 3 -column 1 -sticky ew -pady 2
```



Interface Gráfica- Informações



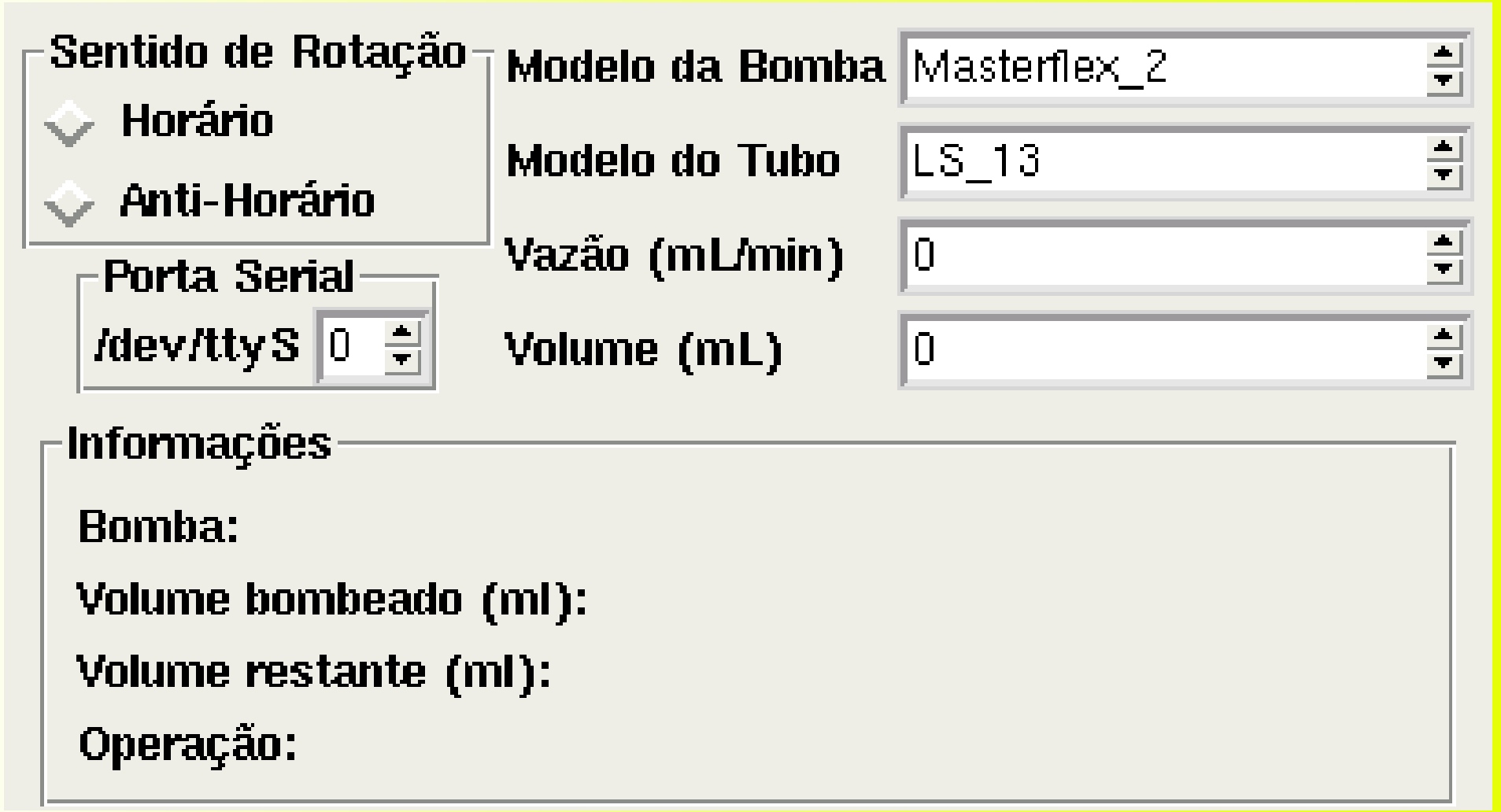
Agora vamos incluir os comandos para criação das janelas/ widgets dentro do labelfame “.info” que devem manter o usuário informado sobre o andamento da operação de bombeamento.

Interface Gráfica - Informações

```
foreach i {"Bomba" "Volume bombeado (ml)" "Volume restante  \
(ml)"  "Operação"} j {status_bomba vol_real vol_rest status_op} {
    set  len [string length $i]
    frame .info.$j
    label .info.$j.nome -text "$i: " -width [expr $len + 2] -anchor w
    label .info.$j.valor -textvar $j -anchor w -width 30
    pack .info.$j.nome -side left
    pack .info.$j.valor -side left -expand 1 -fill x
    pack .info.$j -side top -anchor w -fill x
}
```

Usando um loop foreach, são criados 4 frames, dentro dos quais são criados dois labels (etiquetas) para indicar o nome da variável e seu conteúdo. Notar o uso do conteúdo da variável j em dois contextos. O primeiro label exibe um conteúdo fixo, por isso usamos a opção "text", mas no segundo label usamos a opção "textvar" pois o seu conteúdo é variável.
O comando “string length $i" retorna o número de caracteres da string armazenada dentro da variável i.

Interface Gráfica - Informações



Entendendo o Comando “foreach”

O loop “Foreach” (para cada um de) se aplica a listas, executando um comando ou um script para cada item da lista de itens.
Por exemplo:

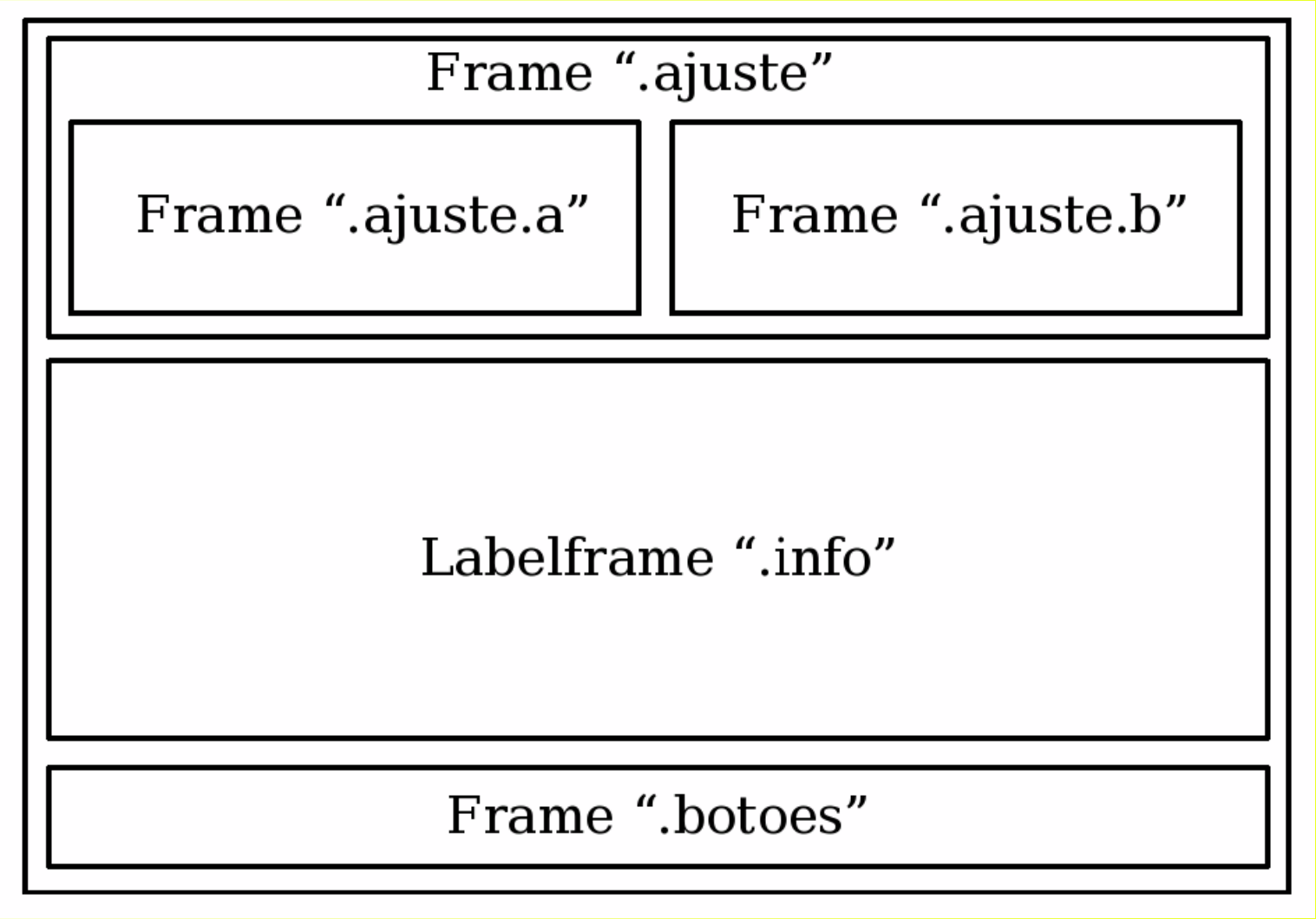
```
set lista_1 {“item_1”  “item_2”  “ item_3”  “ item_4”}
foreach  variavel  $lista_1 {
puts $variavel
}
```

O código acima vai exibir na tela cada um dos itens de “lista_1”
Podemos usar 2 listas dentro de um mesmo foreach como fizemos para montar o widget “.info”.

```
set lista_1 {“item_1”  “item_2”  “ item_3”  “ item_4”}
set lista_2 {“item_1”  “item_2”  “ item_3”  “ item_4”}
foreach  variavel_1  $lista_1 variavel_2 $lista_2 {
puts $variavel_1
puts $variavel_2
}
```

Entendeu? :-)

Interface Gráfica – Botões de Controle



Incluindo o campo “.botoes” na interface gráfica.

Interface Gráfica – Botões de Controle

```
button .botoes.iniciar -text "Iniciar" -command {
iniciar $porta_serial $num_porta_serial $tubo_em_uso \
$sentido_rotacao $vazao $volume }
button .botoes.parar -text "Parar" -command {
parar_op $porta_serial $num_porta_serial }
button .botoes.sair -text "Sair" -command {exit}
pack .botoes.iniciar .botoes.parar .botoes.sair -side right \
 -expand yes -fill both
```

Criação dos 3 botões de controle (Iniciar, Parar e Sair) dentro do frame “.botoes”.
A opção “command” especifica qual comando ou script deve ser executado quando o botão for acionado. No caso do botão “Iniciar” chama o procedimento “iniciar com os argumentos: porta_serial, num_porta_serial, tubo_em_uso, sentido_rotacao, vazao e volume.
O botão “Parar” chama o procedimento “parar_op” com os argumentos: porta_serial e num_porta_serial.
E finalmente o comando “exit” ao se clicar o botão “Sair”

Interface Gráfica – Botões de Controle

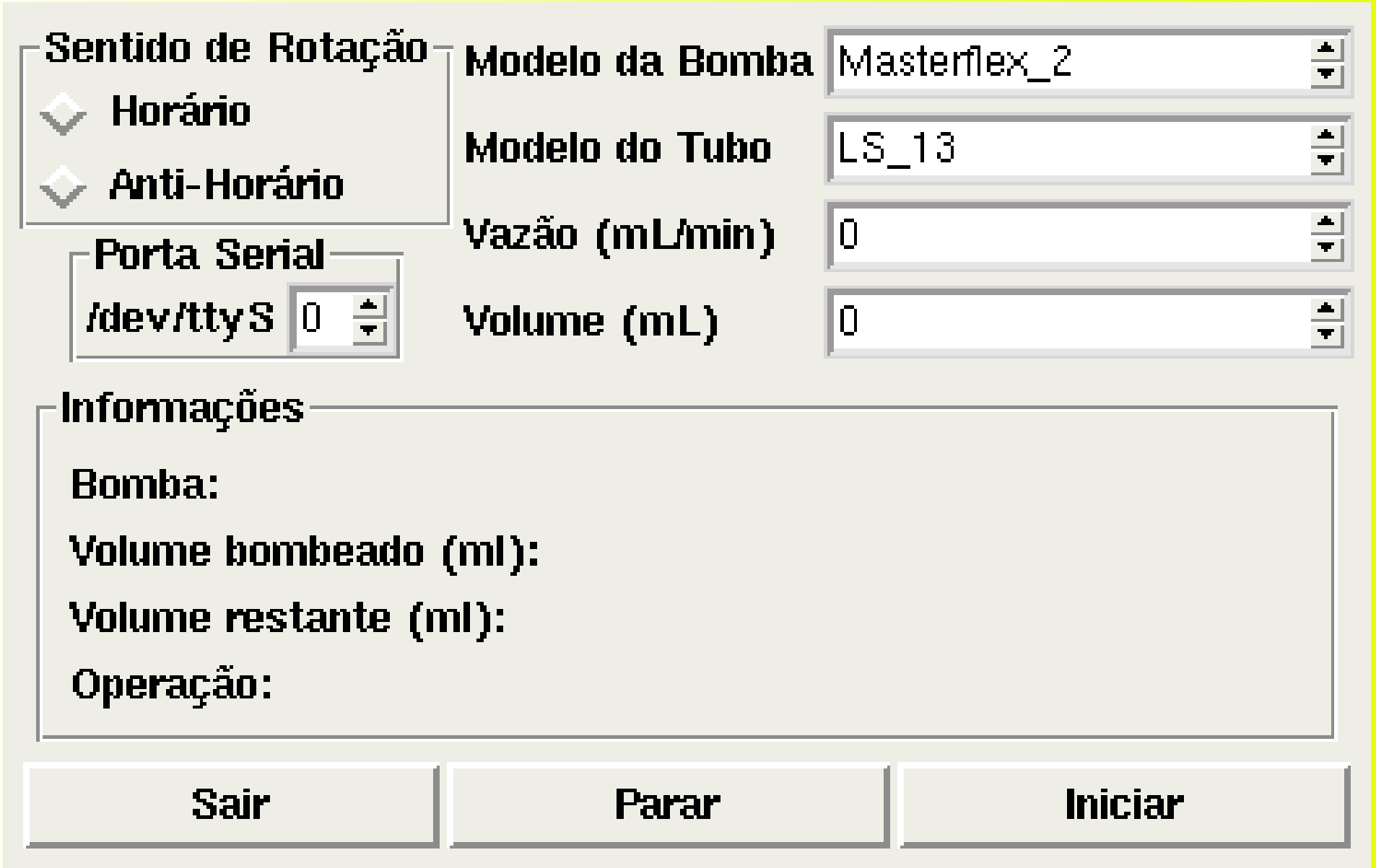


Diagrama Geral

Diagrama de fluxo simplificado com os procedimentos definidos na rotina principal (bomba.tc l) e no módulo (módulo.tcl).
Os números indicam seqüência aproximada de execução dos procedimentos.

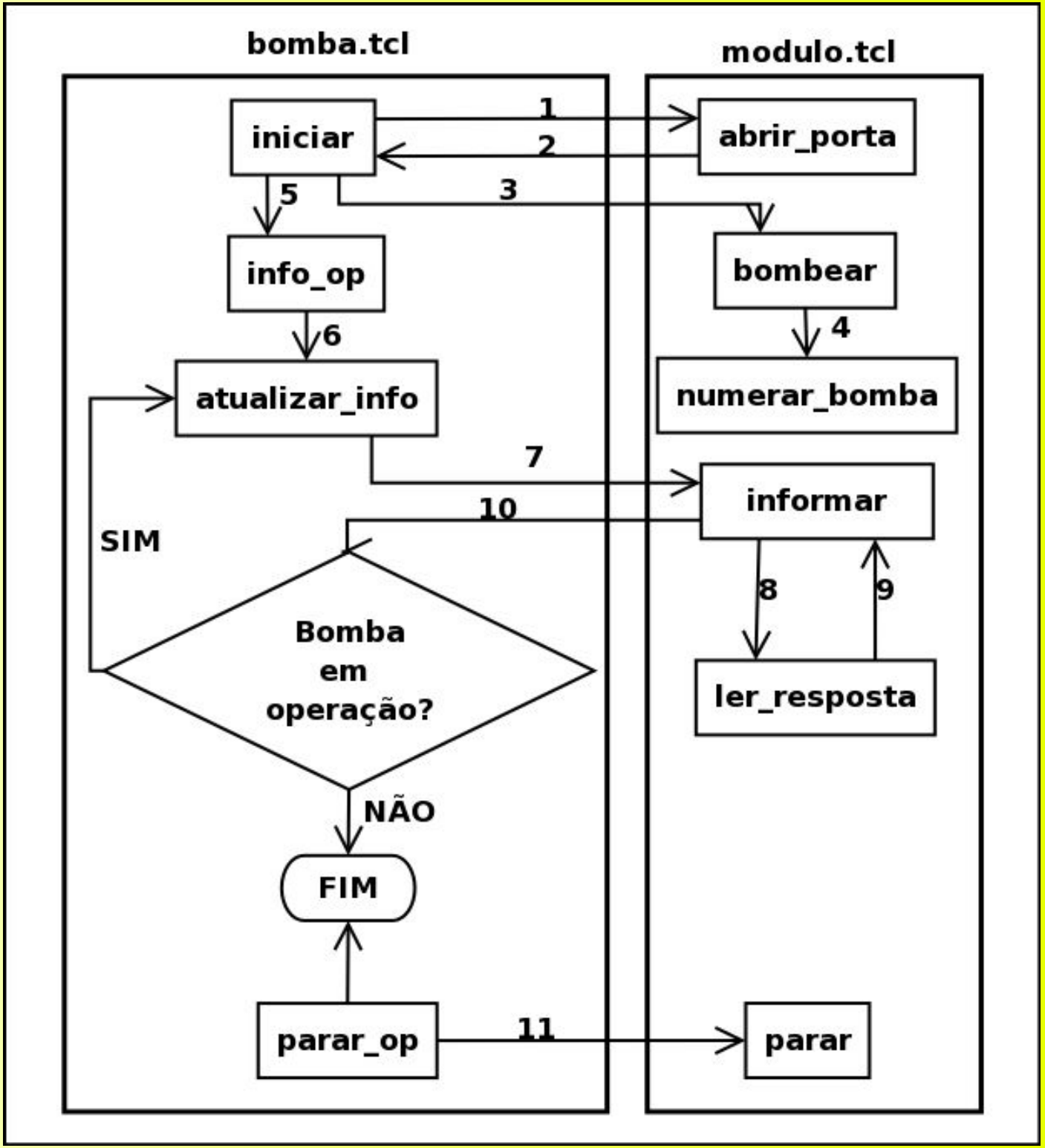
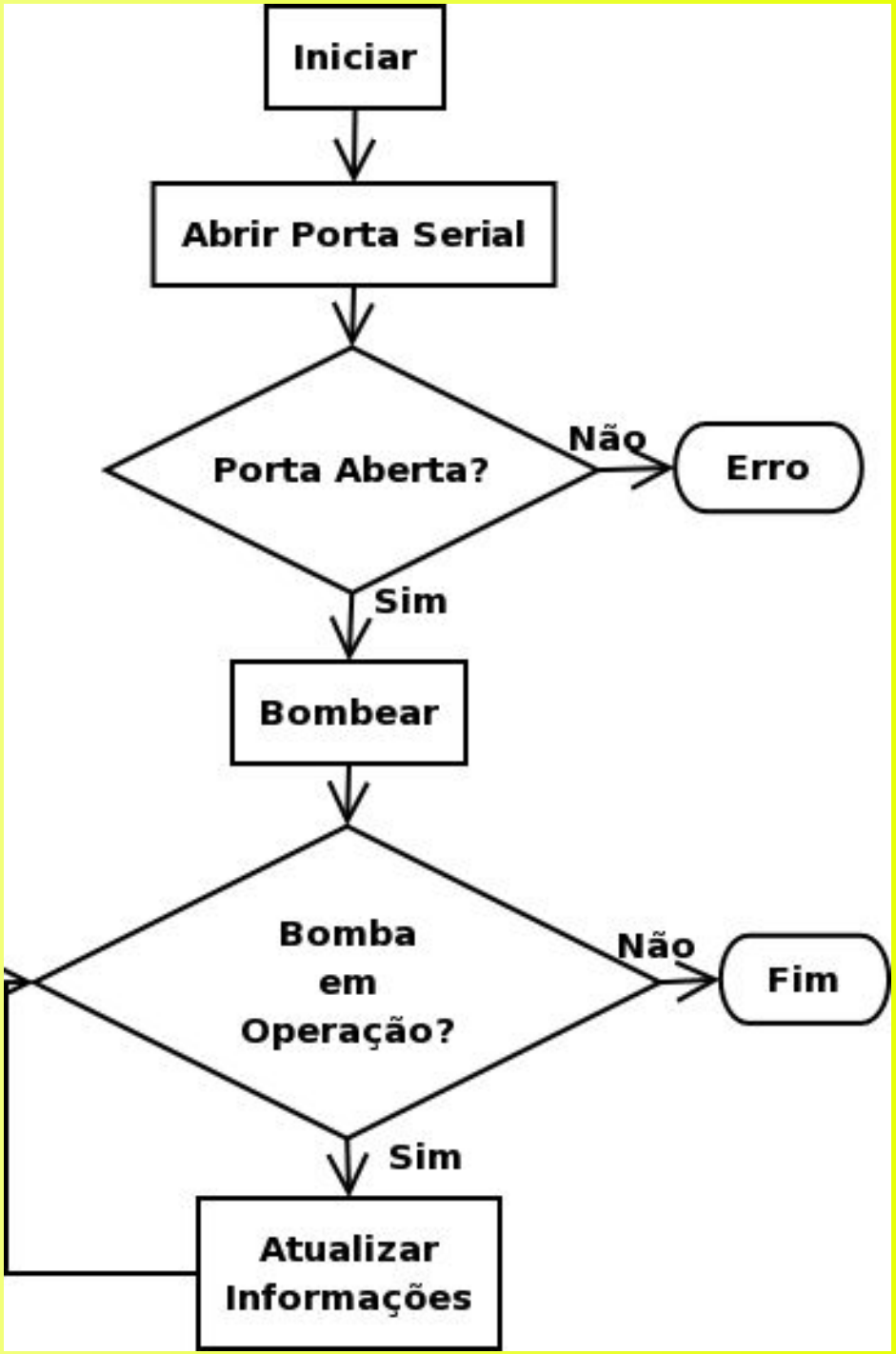


Diagrama de Fluxo

Diagrama de fluxo simplificado com os eventos desencadeados pelo execução do procedimento “Iniciar”.



Procedimento “iniciar”

```
button .botoes.iniciar -text "Iniciar" -command {
    iniciar $porta_serial $num_porta_serial $tubo_em_uso  \
    $sentido_rotacao $vazao $volume
}
proc iniciar { nome numero tubo sentido vazao volume } {
global status_op
    set porta [abrir_porta $nome $numero]
        if {($porta != “SERIAL_ERRO_OPEN”)} {
            bombear $porta $tubo $sentido $vazao $volume
            info_op $porta
        } else {
    set status_op $porta
}
```

Definição do proc “iniciar” que está associado ao botão “Iniciar”. Primeiro declara “status_op” como uma variável global. O proc “iniciar” executa 3 outros procedimentos: abrir_porta, bombear e info_op (informações sobre a operação). Os procedimentos “abrir_porta” e “bombear” devem ser definidos dentro dos módulos, e o procedimento “info_op” será definido dentro própria interface “bomba.tcl”. No entanto os comando “bombear” e “info_op” só serão executados se o conteúdo da variável “porta” for diferente (!=) da string “SERIAL_ERRO_OPEN”

Procedimento “info_op”

```
proc info_op { porta } {
    global status_op
    set status_op "Em andamento"
    .botoes.iniciar configure -state disable
    after 0 [list atualizar_info $porta]
    vwait status_op
    .botoes.iniciar configure -state normal
}
```

Foi incluído o comando after para evitar a alteração de status_op (dentro de atualizar_info) antes do comando vwait, fazendo o comando vwait aguardar indefinidamente! :-(

Inicialmente declara a variável global “status_op” que está vinculada ao label “.info.status_op” do frame “.info”. Dessa forma, o conteúdo desta variável será automaticamente exibido na interface. A opção “-state” em botões pode assumir três valores possíveis: normal, disable e active. No estado disable, o botão fica desabilitado, impossibilitado de ser clicado. Isso impede que o(a) usuário(a) inicie nova operação enquanto a operação atual está em andamento. O comando “after 0 ...” agenda a execução de “atualizar_info $porta” após 0 ms. O comando “vwait status_op” interrompe o andamento do procedimento atual até que a variável “status_op” tenha seu valor alterado ao final da operação. Entretanto isso não impede que outros eventos sejam processados. No final do procedimento o botão “Iniciar” é reativado, volta ao estado “normal”.

Procedimento “atualizar_info”

```
proc atualizar_info { porta } {
      global status_bomba  volume vol_real vol_rest status_op \
        tubo_em_uso
   set status [ informar $porta $tubo_em_uso]
   set status_bomba [ lindex $status 0 ]
   set vol_real [ lindex $status 1 ]
   set vol_rest [expr $volume - $vol_real]
   set status_dados [lindex $status 2]
```

Este procedimento atualiza as informações sobre a operação em andamento. Primeiramente declara as variáveis globais status_bomba (estado da bomba), volume (volume definido pelo usuário), vol_real (volume bombeado), vol_rest (volume restante) e status_op (estado da operação). Em seguida chama o procedimento “informar $porta $tubo_em_uso”, declarado no módulo da bomba, o qual retorna uma lista com três elementos:
{“estado da bomba” “volume bombeado” “estado dos dados”}.
O comando lindex é usado com a sintaxe: “lindex lista índice” e retorna o item da lista “status” na posição informada pelo índice numérico. Lembrar que uma lista é numerada a partir do “0”, por isso o comando: set status_bomba [lindex $status 0]. O mesmo comando lindex é usado para atualizar o conteúdo das variáveis “vol_real” e “status_dados”.

Procedimento “atualizar_info”

```
    if {$status_dados} {
        set status_op "PROBLEMAS DE COMUNICAÇÃO"
        parar $porta
        set status_bomba "Bomba parada"
    } elseif {($status_bomba == "Bomba em funcionamento")} {
        after 2000 [list atualizar_info $porta]
    } elseif {($volume == $vol_real)} {
        set status_op "REALIZADA"
    } elseif {($status_op != "INTERROMPIDA")} {
        set status_op "PROBLEMAS NA OPERAÇÃO"
    }
}
```

O primeiro if verifica se os dados foram recebidos pela bomba corretamente, se o teste for verdadeiro as variáveis status_op e status_bomba são alteradas e o procedimento “parar” é executado. Mas s a bomba ainda está em operação, o comando “after” agenda uma nova chamada do procedimento “atualizar_info” após um intervalo de 3s (3000 ms). O comando “list” é usado para organizar o nome da rotina e seus argumentos. Se a bomba não está em funcionamento verifica-se se o volume solicitado já foi bombeado, neste caso a variável “status_op” é atualizada com a mensagem “REALIZADA”. O último teste “if” verifica se a bomba está parada pelo comando “parar”, em caso negativo houve algum problema na operação de bombeio e a variável status_op é atualizada com “PROBLEMAS NA OPERAÇÃO”.

Procedimento “parar_op”

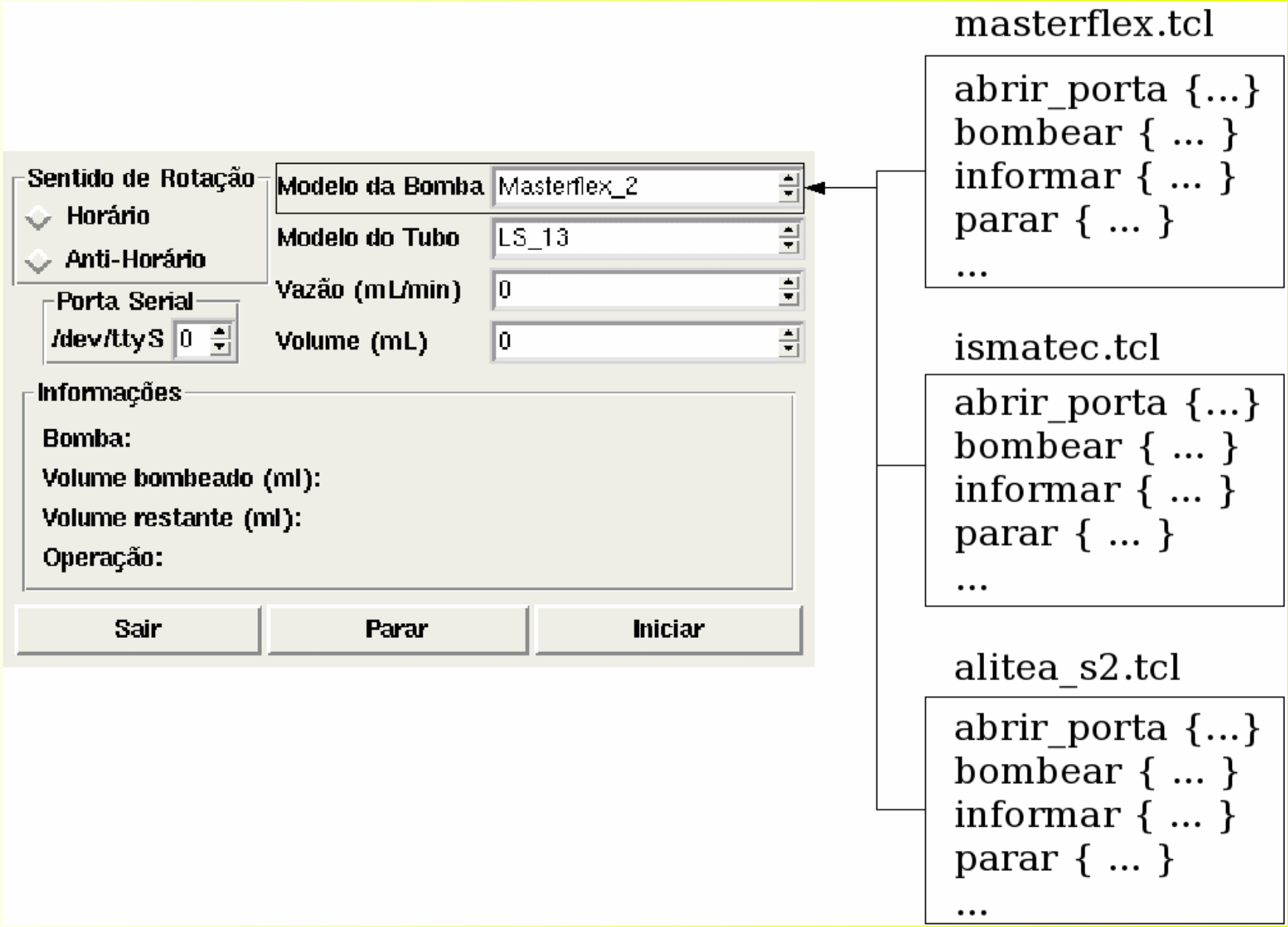
```
proc parar_op { nome numero } {
    global status_op
    set porta [abrir_porta $nome $numero]
    if {($porta != "SERIAL_ERRO_OPEN") && \
        ($status_op == "EM ANDAMENTO")} {
            parar $porta
            set status_op "INTERROMPIDA"
    }
}
```

Finalmente o procedimento “parar_op” que será chamado pelo botão “Parar” (.botoes.parar). Primeiro a declaração da variável global “status_op” que deve ser atualizada no caso de uma interrupção. Em seguida a abertura da porta serial e o teste if para verificar se não houve problemas na abertura e se existe alguma operação em andamento.

Criação dos Módulos

Podemos considerar que a interface está pronta. Resta agora escrever os procedimentos que devem ser executados para o controle da bomba.
Vamos usar uma estrutura modular, onde os diferentes módulos para os diferentes modelos de bombas contêm procedimentos com os mesmos nomes porém com as instruções compatíveis com as respectivas bombas que devem controlar.
O módulo selecionado será “carregado” com o comando “source modulo.tcl” (http://wiki.tcl.tk/1027) disponibilizando os procedimentos para serem usados pela interface.
Vamos definir 4 procedimentos (rotinas) que os módulos deverão conter:
abrir_porta { nome número }
bombear { porta tubo sentido vazao volume }
informar { porta }
parar { porta }

Criação dos Módulos



Carregando os Módulos

Primeiro vamos incluir o comando “source” associado ao widget “spinbox .ajuste.b.tubo_entrada” para carregamento dos módulos pela interface quando o usuário selecionar o modelo da bomba, e criar os respectivos módulos no diretório local apenas com cabeçalho e um comando simples.

```
spinbox .ajuste.b.bomba_entrada  -values [array names modelo_bomba] \
    -bg white -textvariable bomba_em_uso \
    -command { source $modelo_bomba($bomba_em_uso) }
```

Módulo masterflex.tcl:

```
#!/usr/bin/env tclsh
puts "Módulo masterflex.tcl carregado!"
```

Módulo ismatec.tcl:

```
#!/usr/bin/env tclsh
puts "Módulo ismatec.tcl carregado!"
```

Módulo alitea_s2.tcl:

```
#!/usr/bin/env tclsh
puts "Módulo alitea_s2.tcl carregado!"
```

Módulo masterflex.tcl

```
#!/usr/bin/env tclsh

puts "Módulo masterflex.tcl carregado!"

proc abrir_porta { nome numero } {
puts "Procedimento abrir_porta"
return id_porta
}

proc bombear { porta tubo sentido vazao volume } {
puts "Procedimento bombear"
}
```

Inclusão dos procedimentos “abrir_porta” e “bombear” dentro do módulo masterflex.tcl.

Os mesmos procedimentos devem ser definidos nos demais módulos.

O nome dos procedimentos é o mesmo para todos os módulos, mas com conteúdos diferentes de acordo com os comandos de controle do equipamento.

Módulo masterflex.tcl – proc “abrir_porta”

```
#Possíveis conteúdos da variável “status_serial”:
#SERIAL_ABERTA          Porta serial aberta
#SERIAL_ERRO_OPEN       Nao foi possivel abrir a porta serial.
#SERIAL_ERRO_CLOSE      Nao foi possivel fechar a porta serial.
#SERIAL_ERRO_READ       Nao foi possivel ler a porta serial.
#SERIAL_ERRO_WRITE      Nao foi possivel escrever na porta serial.
#SERIAL_ERRO_TIMEOUT Time Out ao tentar ler a porta serial.
```

Muitas informações para os procedimentos de comunicação com a bomba Masterflex foram obtidas no sítio:
http://www.souzamonteiro.com/curso_tcl/aula31.html
A variável "status_serial" será criada para guardar as informações sobre as operações de abertura, fechamento, leitura e escrita da porta serial.
Essa variável é definida como uma variável global para decidir se a porta serial pode ser usada para leitura/escrita.
O evento "Time Out" ocorre quando o tempo de leitura ultrapassa o tempo máximo de espera para operações de leitura, o qual é definido pela opção "timeout"

Procedimento “abrir_porta”

```
proc abrir_porta { nome numero } {
    global id_porta status_serial num_porta_ant
   set baud 4800
   set paridade o
   set bit_dados 7
   set bit_parada 1
   set espera 500

   set fechado [catch {tell $id_porta}]
```

O catálogo da bomba Masterflex L/S 7550 indica:
baud 4800, paridade ímpar (o), 7 bits de dados e 1 bit de parada. Por isso definimos no início do procedimento o valor dessas variáveis que serão usadas para configurar a porta serial, através do comando “fconfigure”.
A linha “set fechado [catch {tell $id_porta}]" serve para verificar se a porta serial já foi aberta anteriormente. Esta informação é importante para evitar a abertura de um novo canal desnecessariamente, sobrecarregando o sistema. A abertura do canal (porta serial) só será feita caso ele esteja fechado ou se o usuário alterou o número da porta serial.

Entendendo o comando “catch” e “tell”

Na linha “set fechado [catch {tell $id_porta}]", descrita anteriormente o comandos "tell" serve para informar a posição atual de um cursor em um canal (id_porta), que pode ter sido aberto para um arquivo, porta serial ou um pipe de comandos.
No caso do canal estar fechado o comando retorna uma mensagem de erro. Aí entra em cena o comando “catch" (catch {tell $id_porta}).
O comando catch executa o comando (ou script) e se este contiver um erro, o comando "catch" retorna um valor diferente de zero que corresponde aos códigos de erro definidos internamente pela Tcl.
Este código é armazenado na variável “fechado” (set [catch {tell $id_porta}]) e portanto qualquer valor diferente de zero armazenado na variável fechado, significa que a porta serial está fechada!
Esta variável será usada dentro do mesmo procedimento "abrir_porta” como condição para um teste "if".

Procedimento “abrir_porta”

```
if { ![info exists num_porta_ant ] } { set num_porta_ant $numero }
```

Dando sequência ao desenvolvimento do procedimento “abrir_porta” incluímos a instrução acima.
O comando “if” executa um comando ou um script se uma condição/teste for verdadeira. Em termos numéricos, “0” significa “falso” e qualquer valor diferente de “0” é “verdadeiro”. O sinal de negação “!” precedendo a expressão serve para inverter (0->1 ou 1->0)
Portanto o teste “if” verifica se a variável “num_porta_ant” já foi definida e se contém algum valor. Se ela não existir, a expressão “[info exists num_porta_ant]” retorna “0”, mas com o sinal de negação “!” passa a valer “1” e o comando “set num_porta_ant $numero” é então executado.
A partir daí, em outras chamadas, a variável “num_porta_ant” passa a armazenar um valor numérico, e a expressão “[info exists num_porta_ant]” passa a retornar “1”, mas é convertido a “0” pelo sinal de negação “!”.

Procedimento “abrir_porta”

```
if {($fechado) || ($numero != $num_porta_ant)} {

        catch {close $id_porta}

        if {[string equal $nome "/dev/ttyS"]} {
        set nome [append nome $numero]
        } else {
        set nome [append nome $numero ":"]
        }
```

Em seguida um comando “if” verifica se a porta serial está fechada, pelo conteúdo da variável “fechada”, **OU** (“||”) se o número da porta serial atual é diferente (“!=”) do número usado na última chamada da função abrir_porta (“num_porta_ant”). Se uma das condições for verdadeira o interpretador Tcl executa os comandos definidos “dentro” do teste if.
O comando “close $id_porta” fecha o antigo canal e no caso de algum erro a mensagem de erro é armazenada pelo comando “catch”, evitando a interrupção do script.
O segundo “if” compara o valor da variável nome com a string “/dev/ttyS” (Linux). Se o teste for verdadeiro (retornar “1”) anexa apenas o número armazenado na variável “numero”, senão (else) acrescenta ainda o caracter “:” (Windows) no valor da variável “nome”, com o uso do comando “append”.

Procedimento “abrir_porta”

```
set resultado_1 [ catch {  set id_porta [open $nome r+]  } ]
```

Na primeira linha foram agrupados 4 comandos:
1- “open $nome r+” abre o dispositivo serial cujo nome está armazenado na variável “nome” no modo leitura e escrita “+r”, ou seja, podendo enviar e receber dados,
2- “set id_porta [...” armazena o nome do canal aberto na variável id_porta,
3- “catch { ... “ evita a interrupção do script e retorna o código de retorno gerado na execução dos comandos precedentes, “0” sem erro, e qualquer valor diferente de “0” se houver erro.
4- “set resultado [...” armazena o código retornado pelo comando “catch” na variável “resultado”

Procedimento “abrir_porta”

```
set resultado_2 [catch { fconfigure $id_porta -translation binary  \
-timeout $espera }]
set resultado_3 [catch {fconfigure $id_porta -mode "$baud,  \
$paridade, $bit_dados,$bit_parada" }]
```

O comando “fconfigure” configura as características de um canal de entrada/saída. A opção “translation” diz como os caracteres de final de linha devem ser traduzidos. Os valores possíveis são: auto, binary, cr, crlf e lf. Para este tipo de equipamento devemos usar a opção “binary”.
O comando “fconfigure” também configura a porta serial com os parâmetros de comunicação definidos no manual do equipamento e armazenados nas respectivas variáveis definidas no início do procedimento “abrir_porta”, ou seja, baud, paridade, bit_dados e bit_parada. A opção “timeout” é usada para determinar o tempo máximo de espera em operações de leitura. Isso impede que o script fique esperando indefinidamente o recebimento de dados, interrompendo a execução do script. Se o comando for executado com sucesso a variável resultado_2 irá armazenar o valor “0”, senão, um valor diferente de “0”.

Procedimento “abrir_porta”

```
if { ($resultado_1 != 0) || ($resultado_2 != 0) || \
($resultado_3 != 0)  } {
     set status_serial "SERIAL_ERRO_OPEN"
     return $status_serial
    } else {
     set status_serial "SERIAL_ABERTA"
     set num_porta_ant $numero
     return $id_porta
    }
```

O teste “if” verifica se houve algum problema na execução do primeiro comando de abertura da porta serial **E** (“&&”) do segundo comando de configuração da porta serial. Em caso afirmativo, armazena a string “SERIAL_ERRO_OPEN” na variável status_serial e retorna o conteúdo desta variável, senão:

1-armazena “SERIAL_ABERTA” em status_serial,
2-atualiza o conteúdo da variável num_porta_ant,
3-e retorna o nome do canal.

Entendendo Procedimentos e Funções

Procedimentos e Funções são praticamente iguais e em Tcl usamos um único comando para criar tanto **procedimentos** como **funções.**
A diferença está no uso que fazemos do valor retornado. Ou seja, em uma função incluímos explicitamente o comando “return $variavel” definindo o valor que deve ser retornado para ser usado em alguma outra operação dentro do programa.
Já os procedimentos não retornam valores.
A rigor, os procedimentos sem o comando “return” também retornam valores. Mas o uso do comando “return” permite controlar o que é retornado pela função.
Apenas como curiosidade, na linguagem C, tudo são funções!

Procedimento “abrir_porta”

```
proc abrir_porta { nome numero } {
    ...
    if {($fechado) || ($numero != $num_porta_ant)} {
    ...
    } else {
    return $id_porta
    }
}
```

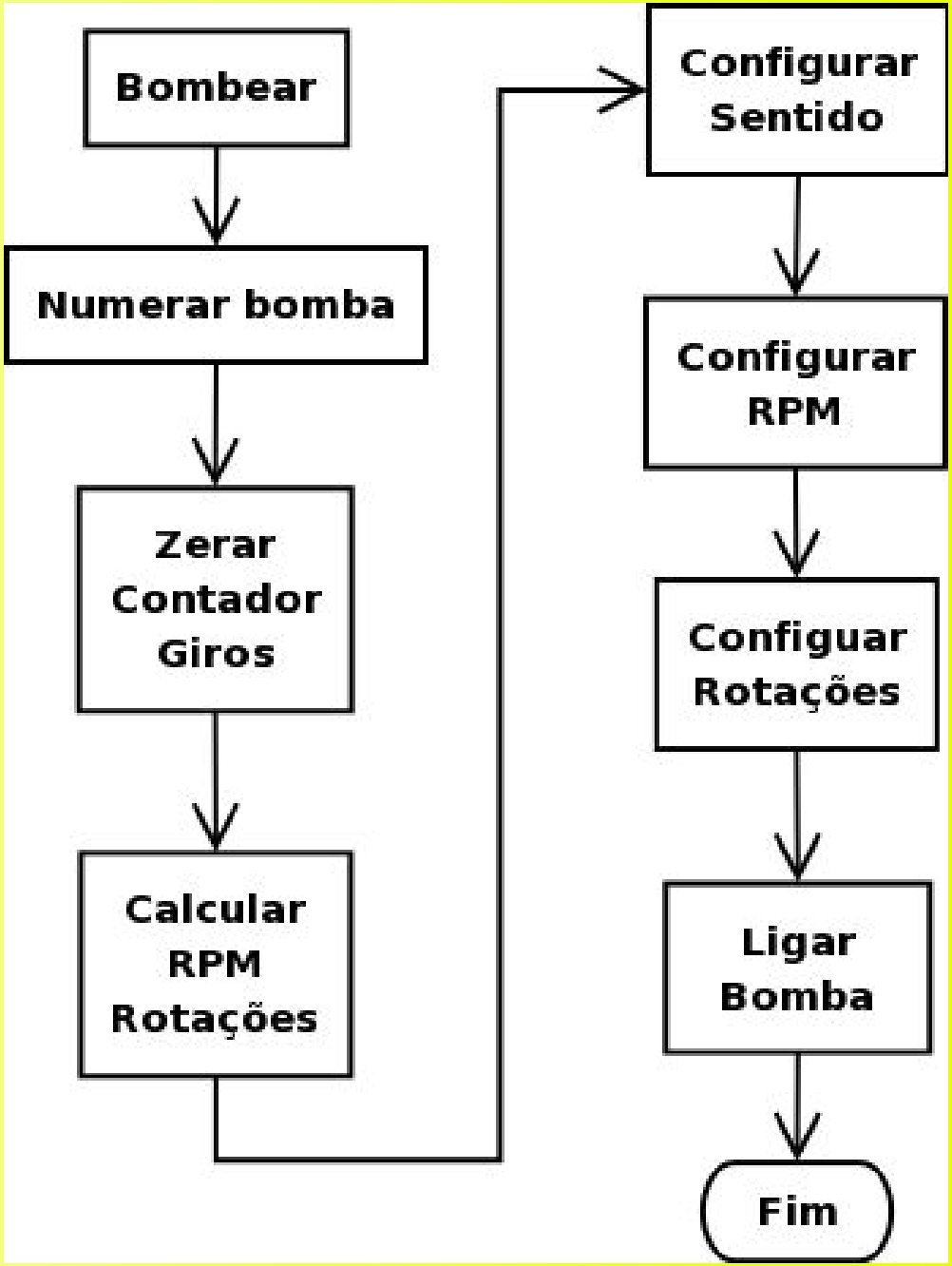
Deixamos em negrito os comandos finais da **função** abrir_porta, fechando o comando if e a função abrir_porta.
Se a porta serial estiver fechada **ou** (“||”) o número do dispositivo serial tiver sido alterado então executar os comandos, que já foram descritos anteriormente, **senão** (“else”) **retornar** (“return”) o mesmo canal que já estava aberto (“$id_porta”).

Procedimento “abrir_porta”

```
proc abrir_porta { nome numero } {
  global   id_porta   status_serial
  global  num_porta_ant
  set baud 4800
  set paridade o
  set bit_dados 7
  set bit_parada 1
  set espera 500
  set fechado [catch {tell $id_porta}]

  if { ![info exists num_porta_ant ] } \
{ set num_porta_ant $numero }

  if {($fechado) || ($numero != \
$num_porta_ant)} {catch \
{close $id_porta}

  if {[string equal $nome \
"/dev/ttyS"]} \
{set nome [append nome $numero]
 } else {
set nome [append nome $numero ":"]
  }
  set resultado_1 [ catch { set id_porta \
[open $nome r+] } ]
  set resultado_2 [catch {fconfigure \
$id_porta -translation binary  \
-timeout $espera}]
  set resultado_3 [catch {fconfigure \
$id_porta  -mode  "$baud, $paridade,  \
$bit_dados,$bit_parada" }]

  if {($resultado_1 != 0) ||  \
($resultado_2 != 0) ||  \
($resultado_3 != 0) } {
  set status_serial "SERIAL_ERRO_OPEN"
  return $status_serial
  } else {
  set status_serial "SERIAL_ABERTA"
  set num_porta_ant $numero
  return $id_porta
  }
  } else {
  return $id_porta
  }
}
```

Procedimento “bombear”

A numeração da bomba é um requisito necessário para identificar um módulo dentro de uma configuração “Daisy Chain”.
O contador de giros armazena o número de rotações da bomba e permite acompanhar o volume bombeado.
A vazão é convertida em RPM (Rotações Por Minuto) e o volume é convertido em número de rotações.
Essas informações são transferidas para a interface da bomba juntamente com o sentido de rotação antes do comando para iniciar.



Procedimento “bombear”

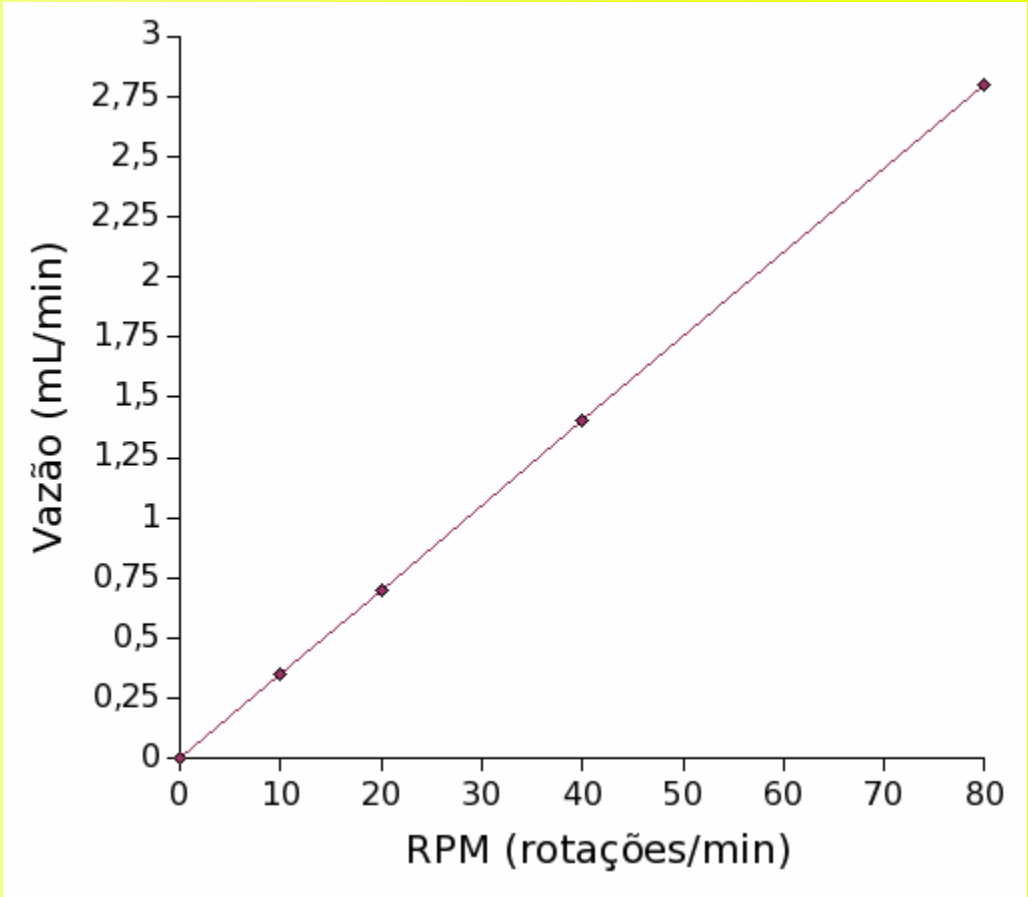
```
proc bombear { porta tubo sentido vazao volume } {

    global cte_tubo

    numerar_bomba $porta 01
```

Iniciando a definição do procedimento “bombear”, com 5 argumentos (porta, tubo, sentido, vazao e volume), declarando a variável “cte_tubo” do tipo array que irá armazenar o valor da constante (vazão/RPM) para cada modelo de tubo disponível.
O comando “numerar_bomba” (definido mais tarde) executa o procedimento de identificação da bomba usando como argumentos o nome do canal e o caractere “01”. Lembrar que este modelo de bomba pode ser usado em série com outros dispositivos (Daisy Chain) e por isso é necessário atribuir um número de identificação para cada equipamento integrante da cadeia. Neste caso estamos usando apenas uma bomba, por isso a numeração 01.

Declarando a variável “cte_tubo”

```
set cte_tubo(LS_13) 0.06
set cte_tubo(LS_14) 0.2166
```

Declarando a variável do tipo array no início do módulo “masterflex.tcl” que vai armazenar os diferentes modelos de tubos usados com as respectivas razões vazão/rpm. Permitindo converter os valores de vazão (ml/min) em RPM para configurar a velocidade da bomba para o modelo de tubo selcionado.



Esta constante deve ser determinada experimentalmente pela inclinação do gráfico de vazão por RPM para cada modelo de bomba utilizado.

Variável “cte_tubo”

No entanto é importante lembrar que alguns modelos de cabeçotes reduzem a velocidade de rotação:
Modelos 07519-10 e 07519-20 não reduzem a velocidade
Modelos 07519-15 e 07519-25 reduzem a velocidade em 5 vezes.
Por isso a constante de cada tubo irá depender também do cabeçote em uso.

Procedimento “bombear”

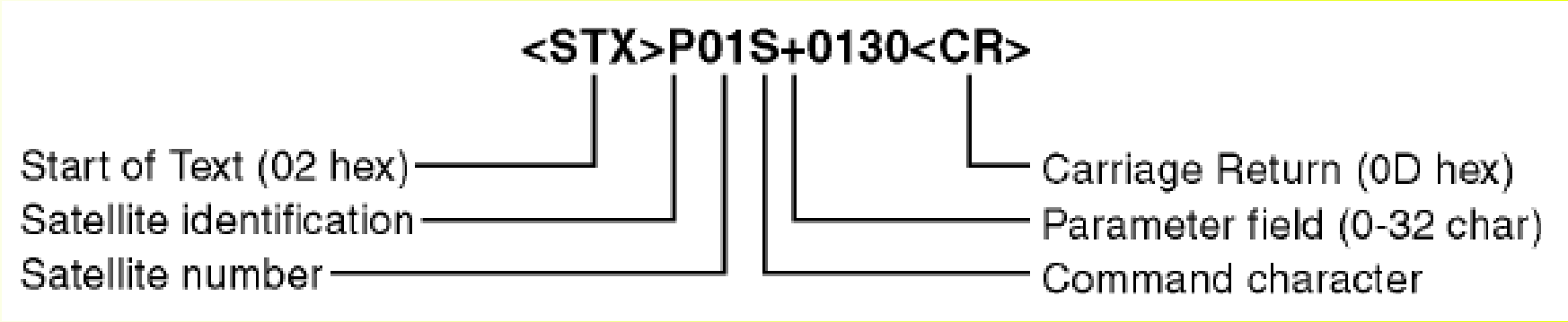
```
puts -nonewline $porta "\x02P01Z0\x0d"
flush $porta
```

Comando para zerar o contador interno da bomba para podermos acompanhar o volume bombeado.
Os comandos enviados para a bomba devem ser precedidos pelo caractere de controle <STX> (Start of Text – Inicio do texto) “02” (hexadecimal) e finalizados pelo <CR> (Carriage Return - mudança de linha) “0d” (hexadecimal) no código ASCII. Na Tcl esses caracteres são enviados respectivamente como "\x02" e "\x0d".
O formato do comando segue a seguinte estrutura:
<STX>PnnZ0<CR>, onde P identifica o tipo de equipamento (Pump), nn identifica o número do equipamento (01), Z0 para zerar o contador.
A opção "nonewline" é usada para evitar a inclusão do caracter "\n" (Nova Linha) pelo comando puts.
E finalmente o comando "flush" força o envio imediato do comando.

Procedimento “bombear”

```
set rpm [format "%.1f" [expr $vazao / $cte_tubo($tubo)]]

puts -nonewline $porta "\x02P01S$sentido$rpm\x0d"

flush $porta
```

A expressão [expr $vazao / $cte_tubo($tubo)] retorna o valor da velocidade de rotação em RPM (k=vazao/RPM -> RPM=vazao/k) a partir da vazão (ml/min) definida pelo usuário.
Para atender a especificação do equipamento, o resultado é formatado com uma casa decimal antes de ser enviado pela porta serial.
E finalment o comando “puts ...” envia o caractere “S”(Speed) no formato especificado, incluindo o sentido de rotação armazenado na variável "sentido", um dos argumentos do procedimento "bombear".



Procedimento “bombear”

```
proc bombear { porta tubo sentido vazao volume } {
...
set rotacoes [expr $volume / $cte_tubo($tubo)]
    set rotacoes  [format "%.2f" $rotações]

    puts -nonewline $porta "\x02P01V$rotacoes\x0d"
    flush $porta

    puts -nonewline $porta "\x02P01G\x0d"
    flush $porta
}
```

E para finalizar o proc “bombear”, os comandos “set rotacoes...” calcula do número de rotações para bombear o volume especificado pelo usuário (rotacoes = volume/k). E envia o caractere “V” seguido do conteúdo da variável “rotacoes”

O último comando “puts ...” envia o caractere “G” para iniciar o bombeamente conforme os parâmetros definidos anteriormente (sentido, vazão e volume).

Procedimento “numerar_bomba”

```
proc numerar_bomba { porta num } {
    global status_serial

    puts -nonewline $id_porta "\x05"
    flush $id_porta
```

Ao ser ligada, a bomba ativa o buffer de envio e recebimento e ativa o sinal do pino 8 (RTS) do conector serial. Neste momento o computador deve enviar o caractere de controle <ENQ> (Enquire), “\x05” em hexadecimal, para interrogar a identidade do dispositivo. Se a bomba não tiver sido identificada, retorna uma sequência de caracteres contendo “P?” e o código referente ao modelo. (“0” para o modelo 7557-30 e “2” para o modelo 7550-50).

<STX>P?0<CR> = 600 RPM 7550 -30
<STX>P?2<CR> = 100 RPM 7550 -50

Procedimento “numerar_bomba”

```
set resposta [ler_resposta $porta]

if {!($status_serial == "SERIAL_ERRO_TIMEOUT")} {

    set numerar [regexp {P\?} $resposta texto]
```

O comando “set resposta [ler_resposta $porta]” executa o procedimento “ler_resposta” (descrito posteriormente) com o argumento “porta” e retorna as mensagens recebidas pela porta serial, armazenando a mensagem recebida na variável “resposta”.
Se o conteúdo de “status_serial” “não” for igual a “SERIAL_ERRO_TIMEOUT” então o comando “regexp” (expressão regular) verifica se na string de resposta existe a seqüência “P?" que é enviada pela bomba enquanto ela ainda não foi numerada.
O comando “regexp” retorna “1” se o padrão "P?" for encontrado no conteúdo da variável "resposta" e zero caso contrário. Este retorno é então armazenado na variável numerar para ser usado posteriormente.

Procedimento “numerar_bomba”

```
proc numerar_bomba {porta num} {
...
        if {$numerar} {
            puts -nonewline $porta "\x02P$num\x0d"
            flush $porta
        }
    }
}
```

E finalizando o procedimento, se o conteúdo de "numerar" for diferente de "0", o comando puts envia o comando "P$num", com os respectivos caracteres de controle (\x02 e \x0d), onde a variável "num" armazena o número do dispositivo que foi passado como argumento para o procedimento "numerar_bomba {porta num}".

Expressões Regulares

Expressões regulares permitem localizar e extrair seqüências de caracteres de um arquivo ou de uma string.

Em Tcl existem dois comando para manipular expressões regulares : "regexp" e "regsub". O comando "regexp" retorna 1 caso uma expressão regular combine com um texto passado para o comando, e zero caso contrário. E o comando "regsub" substitui as ocorrências de uma expressão regular por um texto substituto informado para o comando.

Para criar expressões regulares usamos os "metacaracteres", que são caracteres comuns com significados especiais (., ^, ?, *, +, { etc).
Existem quatro tipos de metacaracteres: Representantes (., [, ^), Quantificadores (?, *, +, {), Âncoras (^, $, \b), e Outros (\, |, (,).

(Extraído do livro: Tcl/Tk, Programação Linux, Alexander Franca)

Expressões Regulares – Metacaracteres Representantes

O metacaractere ponto (.) representa um caractere, seja ele qual for. Por exemplo, a expressão regular:

.omba

combina com: Bomba, bomba, tomba, zomba, ou seja, qualquer coisa que tenha um caractere seguido da seqüência “omba". O ponto casa inclusive com o caractere ponto!

A lista ([]) define uma seqüência de caracteres dentro de colchetes. Por exemplo a expressão regular:

p[ri]a

combina com: pra, pia, ou seja, qualquer palavra que comece com "p", termine com "a" e tenha as letras "r" ou "i" entre as letras "p" e "a".

A lista negada especifica os caracteres que **não** devem combinar com o texto. Por exemplo, a expressão regular:

p[^ri]a

combina com qualquer palavra que comece com "p", termine com "a" e **não** tenha as letras "r" ou "i" entre as letras "p" e "a".

(Extraído do livro: Tcl/Tk, Programação Linux, Alexander Franca)

Expressões Regulares – Metacaracteres Quantificadores

O metacaractere (?), chamado de opcional, determina que o caractere anterior a ele pode combinar zero ou 1 vez. Por exemplo, a expressão:
sai?
combina com sa ou sai.
Podemos associar um metacaracter quantificador a um representante. Por exemplo, a expressão regular:
sa.?
combina com qualquer seqüência que comece com sa, seguida ou não de qualquer caractere: sa, sai, sal, sa#, sa., etc.

O metacaractere asterisco (*) representa zero, 1 ou mais ocorrências do caractere anterior. Por exemplo, a expressão regular:
sai*
combina com: sa, sai, saii, saiiiii...
Combinando um quantificador com um representante. A expressão:
sa.*
combina com: sa, sai, saii, saiiiiiiii, saleiro, salgado, sapo etc, ou seja, qualquer caractere em qualquer quantidade.

(Extraído do livro: Tcl/Tk, Programação Linux, Alexander Franca)

Expressões Regulares – Metacaracteres Quantificadores

O metacaractere (+) determina que o caractere anterior a ele pode combinar uma ou mais vezes. Mas não é possível ter nenhuma ocorrência do caractere anterior Por exemplo, a expressão:
9+1
combina com 91, 9999991, mas não combina com 1.
O metacaractere ({}) permite especificar o número exato de ocorrências do caractere anterior, ou então uma quantidade mínima ou máxima de ocorrências do caractere anterior. Por exemplo:
{1,6} = de uma a seis ocorrências
{2, } = duas ou mais ocorrências
{3} = exatamente 3 ocorrências
Por exemplo, a expressão:
6{1,3}
combina somente com: 6, 66 ou 666.

(Extraído do livro: Tcl/Tk, Programação Linux, Alexander Franca)

Expressões Regulares – Metacaracteres Âncoras

Os metacaracteres do tipo âncora apenas marcam uma posição específica em uma linha. Eles não podem ser quantificados e, portanto, os metacaracteres quantificadores (*, ?, { e +) não surtem qualquer efeito sobre eles.
O meta caractere circunflexo (^) marca o início de uma linha. Por exemplo, a expressão regular:
^#.*
combina com qualquer linha que se inicia com um caracter tralha (#). Pode ser útil para remover as linhas de comentários de um script ou de um arquivo de configuração.
O metacaractere cifrão ($) especifica o final de uma linha. Por exemplo, a expressão regular:
[0-7]+$
combina com uma ou mais ocorrências de números no intervalo de 0 a 7 que esteja no final de uma linha. Essa expressão regular será usada para avaliar a operação da bomba.

(Extraído do livro: Tcl/Tk, Programação Linux, Alexander Franca)

Expressões Regulares – Outros Metacaracteres

São os metacaracteres que não são quantitativos, representativos ou âncoras, mas servem para refinar o controle das expressões regulares. O metacaractere escape (\) é usado quando queremos especificar um caractere literal. Por exemplo, a expressão regular:

P\?

combina com o caractere (P) seguido do caractere interrogação (?) e não o metacaractere quantificador opcional (?). Essa expressão regular permite verificar a ocorrência dessa seqüência na mensagem enviada pela bomba após o receber o comando <ENQ>.

<STX>P?0<CR> = 600 RPM 7550 -30
<STX>P?2<CR> = 100 RPM 7550 -50

```
set numerar [regexp {P\?} $resposta texto]
```

O metacaracter OU (|) permite especificar seqüências possíveis para uma determinada posição. Ex: dia|noite, pode combinar com dia OU noite.

(Extraído do livro: Tcl/Tk, Programação Linux, Alexander Franca)

Procedimento “ler_resposta”

```
proc ler_resposta { porta } {
    global status_serial
    while 1 {
     set leitura [catch {set caractere [read $porta 1]}]
     if {$leitura} {
         set status_serial "SERIAL_ERRO_READ"
         return $status_serial
     }
```

O comando “read $porta 1” é usado para ler 1 byte pela porta serial quando o caractere é enviado sem o caractere “\n” (nova linha). O retorno do comando é atribuido à variável caractere. Se não houver erro na operação de leitura o comando “catch” retorna “0”, caso contrário retorna “1” para a variável “leitura”. O teste “if” verifica se houve erro na operação de leitura. Em caso de erro o teste “if” é verdadeiro e a variável status_serial passa a armazenar a mensagem de erro de leitura. E finalmente o comando “return” encerra a execução do procedimento retornando a variável “status_serial”.

Procedimento “ler_resposta”

```
        if {$caractere == ""} {
            set status_serial "SERIAL_ERRO_TIMEOUT"
            return $status_serial
        }
        if {$caractere == "\x0d"} {
         return $palavra
        }
        append palavra $caractere
    }
}
```

Se nenhum byte for recebido no intervalo de 500 ms, a variável status_serial passa a armazenar a mensagem “SERIAL_ERRO_TIMEOUT” e o procedimento retorna com a variável “status_serial”. O segundo teste “if” verifica se o byte recebido é um caractere de controle <CR> (\x0d), indicador de final da mensagem. Em caso afirmativo encerra o procedimento e retorna o conteúdo da variável “palavra”. Se todos os testes “if” forem falsos o comando “append palavra $caractere” acrescenta o conteúdo da variável “caractere” ao conteúdo da variável “palavra”.

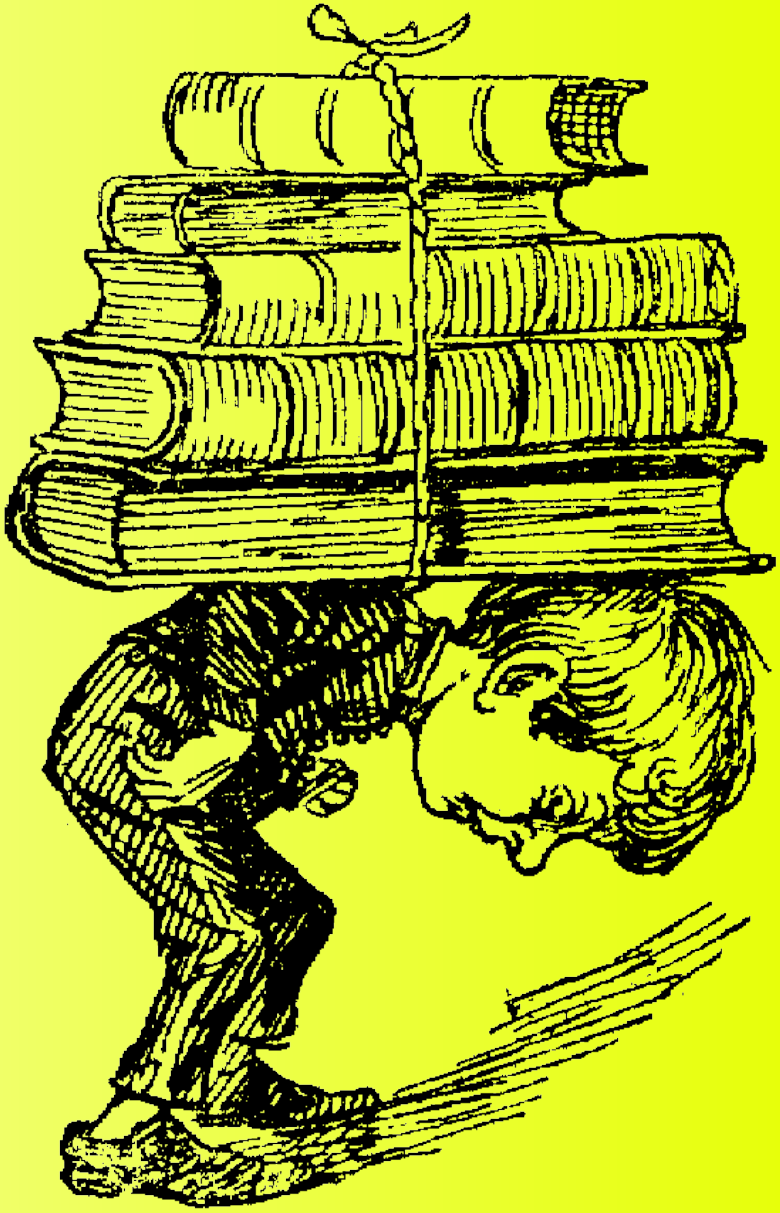
Procedimento “ler_resposta”

```
proc ler_resposta { porta } {
    global status_serial
    while 1 {
        set leitura [catch {set caractere [read $porta 1]}]
        if {$leitura} {
            set status_serial "SERIAL_ERRO_READ"
            return $status_serial
        }
        if {$caractere == ""} {
            set status_serial "SERIAL_ERRO_TIMEOUT"
            return $status_serial
        }
        if {$caractere == "\x0d"} {
            return $palavra
        }
        append palavra $caractere
    }
}
```

Procedimento “ler_resposta”

Exercício. Oba! :-)

Montar um diagrama de fluxo para o procedimento “ler_resposta”.



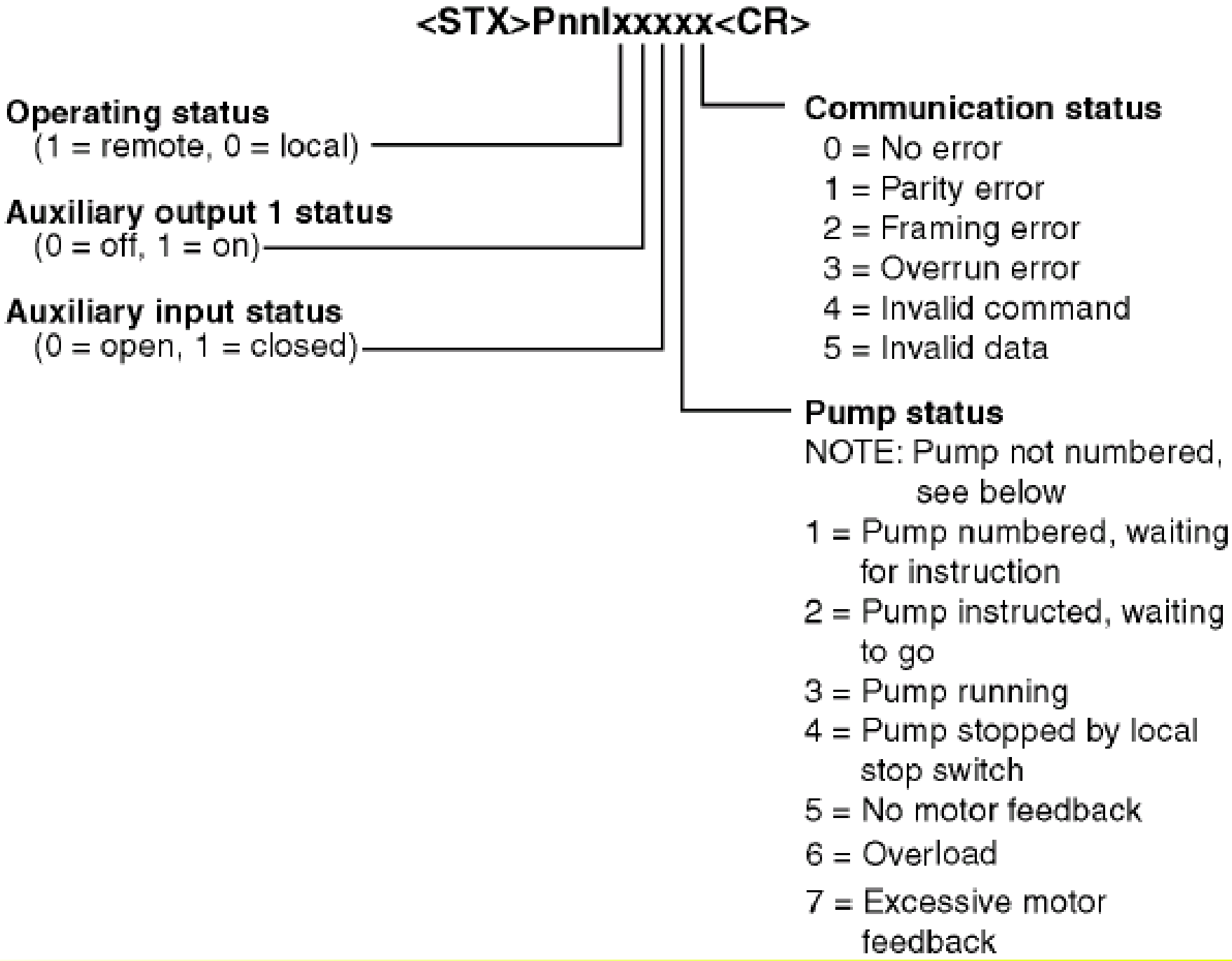
Procedimento “informar”

Procedimento para obter informações sobre as condições de operação e retornar as informações como uma lista
É usada expressão regular para extrair da resposta enviada pela bomba a última seqüência de 5 dígitos contendo várias informações sobre a bomba
<STX>PnnIxxxxx<CR>
| | | | |Comunicação (0,1,2,3,4 ou 5)
| | | |Bomba
| | |Entrada auxiliar (0=aberto, 1=fechado)
| |Saída Auxiliar (0=off, 1=on)
|Operação (1=remota, 0=auxiliar)

Significado dos códigos de Comunicação: 0=sem erro, 1=erro de paridade,
2=erro de framing, 3=estouro de buffer, 4=comando inválido e 5=dado inválido.

Significado dos códigos da bomba: 1"Bomba numerada, Aguardando comando", 2 "Aguardando comando para iniciar", 3 "Bomba em funcionamento", 4 “Interrupção manual", 5 "Sem comunicação com o motor", 6 "Sobrecarga do motor", 7 "Excessivo "Feedback" do motor".

Procedimento “informar”



Procedimento “informar”

```
proc informar { porta tubo } {

    global cte_tubo

    puts -nonewline $porta "\x02P01I\x0d"
    flush $porta

    set resposta [ler_resposta $porta]

    set r1 [regexp {([0-7]+$)} $resposta status_geral]
```

Declara a variável global “cte_tubo” cujo conteúdo será usado para calcular o volume que já foi bombeado.
Em seguida envia o comando “I”, solicitando informações sobre a bomba e armazena a string com as informações na variável status_geral com o uso de uma expressão regular [0-7]+$.
O metacaractere cifrão ($) especifica o final de uma linha e combina com uma ou mais ocorrências de números no intervalo de 0 a 7 que esteja no final de uma linha. Se o comando “regexp” combinar com o conteúdo da variável “resposta”, a variável “r1” armazena o valor “1”, caso contrário armazena “0”.

Procedimento “informar”

```
   if {$r1} {
    set bomba [string index $status_geral 3]
    switch -- $bomba {
        1 { set status_b "Bomba numerada, Aguardando
comando" }
        2 { set status_b "Aguardando comando para iniciar"}
        3 { set status_b "Bomba em funcionamento"          }
        4 { set status_b "Interrupção manual"              }
        5 { set status_b "Sem comunicação com o motor"     }
        6 { set status_b "Sobrecarga do motor"             }
        7 { set status_b "Excessivo \"Feedback\" do motor" }
        default { set status_b "Código desconhecido"       }
    }
```

Se o teste if for verdadeiro (r1 > 0) o comando “string index ...” retira o caractere da quarta posição (3) da variável “status_geral”, contendo informações sobre a bomba, e armazena na variável “bomba”. Em seguida o comando “switch” altera o valor da variável “status_b” conforme o código numérico especificado no manual do equipamento

Procedimento “informar”

```
set dados [string index $status_geral 4]
puts -nonewline $porta "\x02P01C\x0d"
flush $porta
set resposta [ler_resposta $porta]
set r2 [regexp {([0-9]+\.[0-9]+)} $resposta rotacoes_real]
```

O primeiro comando atribui à variável “status_dados” o caractere da quinta posição (4) da variável “status_geral” informando se os comando foram interpretados corretamente pela bomba. Qualquer valor diferente de “0” significa a ocorrência de erro.
Em seguida envia o comando “C” solicitando informações sobre rotações realizadas e armazena a string com as informações na variável resposta. O comando regexp “extrai” a seqüência contida na variável “resposta” e armazena em “rotacoes_real” com o uso da expressão regular [0-9]+\.[0-9]+, ou seja, um ou mais caracteres numéricos no intervalo de 0 a 9 seguido de um ponto "." e outra seqüência de caracteres de 0 a 9.

<STX>PnnC<CR>	<STX>Cxxxxxxx.xx<CR> max revolutions = 9,999,999.99

Procedimento “informar”

```
    if {$r2} {
        set vol [expr $rotacoes_real * $cte_tubo($tubo)]
        set vol [format "%.2f" $vol]
        set retorno [list $status_b $vol $dados]
       return $retorno
    }
}
```

Se “r2” for diferente de zero, então o comando “regexp” anterior conseguiu extrair o número de rotações realizadas e armazenar na variável rotacoes_real. Neste caso o volume bombeado é obtido pelo produto do número de rotações pela constante do tubo em uso (k=ml/rotação). Em seguida o volume é formatado para exibir 2 casas decimais com o comando “format “%.2f” ... ”. Com todas as informações para o retorno da função, o comando “list” organiza as informações sobre a bomba, volume e dados no formato de lista na variável “retorno”, o qual é retornado pelo comando “return”.

Procedimento “informar”

```
proc informar { porta tubo } {
    ...
    set r1 [regexp {([0-7]+$)} $resposta status_geral]
    if {$r1} {
    ...
    }
    set retorno [list $resposta 0 1]
    return $retorno
}
```

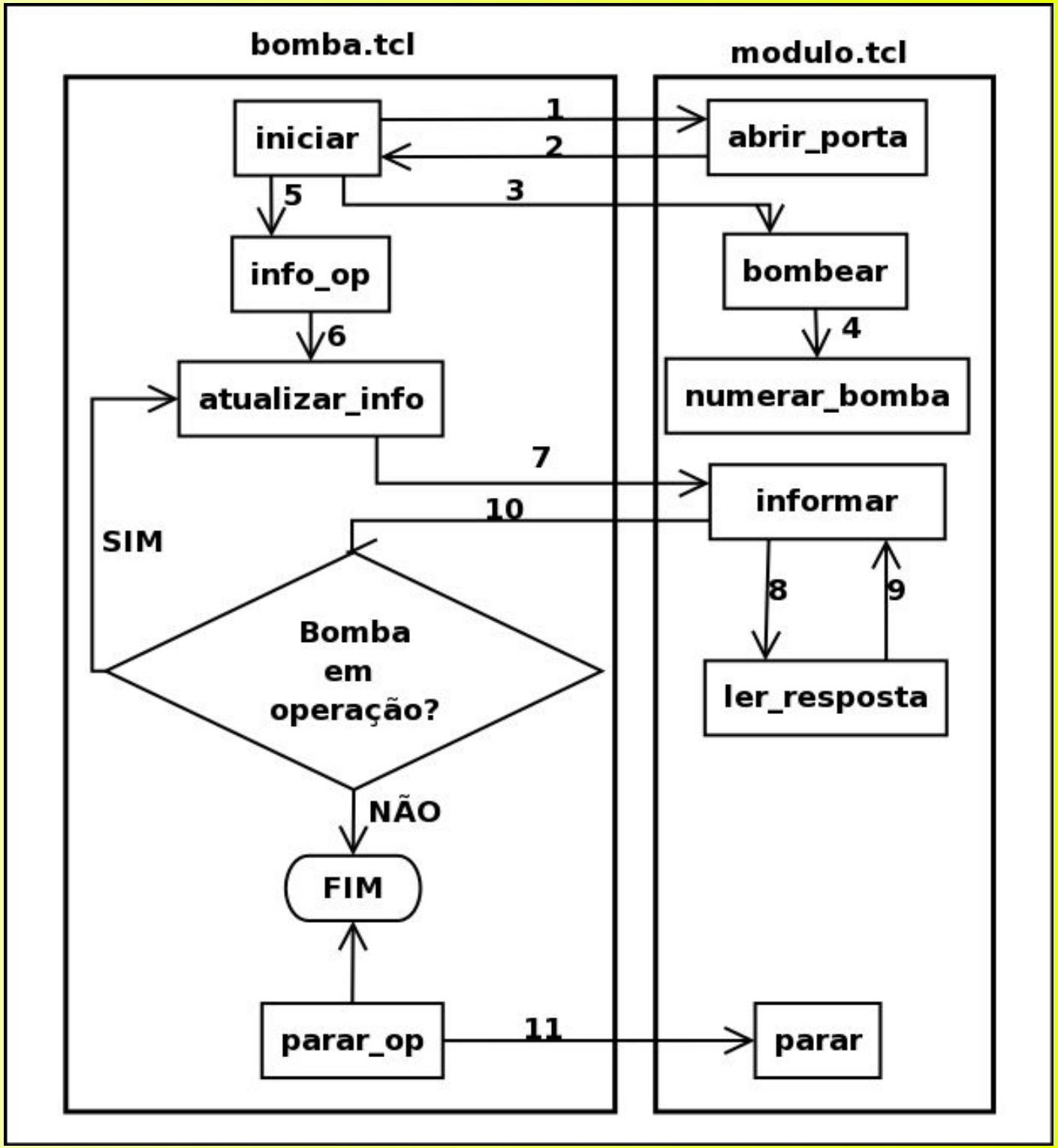
O comandos finais (em negrito) são executados apenas se a variável “r1” for igual a zero, o que irá ocorrer caso a variável “resposta” não contenha nenhuma seqüência numérica. Neste caso o procedimento retorna “0” para a variável “vol” e “1” para a variável “dados” indicando erro na variável dados.

Procedimento “parar”

```
proc parar { porta } {
    puts -nonewline $porta "\x02P01H\x0d"
    flush $porta
}
```

Este procedimento simplesmente envia o comando “H” para parar a bomba, usando como argumento o nome do canal para a porta serial.

Recapitulando!



Referências

http://www.provitec.com.br/produtos/peristaltica.html

Manual da bomba peristaltica Masterflex
http://www.coleparmer.com/catalog/manual_pdfs/07523-60,-70.pdf

Carregamento dinâmico de módulos
http://wiki.tcl.tk/1027

Acesso a porta serial
http://www.souzamonteiro.com/curso_tcl/aula31.html

Programação com Tcl/Tk (Alexander Franca)
http://www.nautae.eti.br/meulivro.php

Obrigado pela atenção!