CBPF-NT-003/00

LabVIEW

Valeriana Gomes Roncero
roncero@cbpf.br

Marcelo Portes de Albuquerque
marcelo@cbpf.br

Centro Brasileiro de Pesquisas Físicas – CBPF
Rua Dr. Xavier Sigaud, 150
22290-180 – Rio de Janeiro, R.J.

Sumário

Introdução

LabVIEW (Laboratory Virtual Instruments Engineering Workbench) é uma linguagem de programação desenvolvida pela National Instruments.

O LabVIEW é diferente das usuais linguagens de programação em um aspecto importante. Ao invés de utilizar linhas de código, ele utiliza uma linguagem gráfica conhecida como linguagem G que é composta de muitos nodos conectados. O LabVIEW tem um compilador gráfico aperfeiçoado para maximizar o desempenho do sistema. O LabVIEW simplifica o desenvolvimento do programa, e também diz imediatamente ao usuário quando um erro foi cometido. Como também produz um código que pode ser reutilizável. LabVIEW é usado como um substituto para as linguagens baseadas em linhas de código, permitindo ao usuário observar o que o programa está fazendo literalmente, deste modo, você pode inserir um pedaço de código esquecido, e pode estudar como o dados estão "viajando". Ele tem extensivas bibliotecas de funções para qualquer programa.

Os programas no LabVIEW são chamados de **Virtual Instruments** (VI's) porque a aparência e as operações simulam instrumentos reais.

1. Apresentação dos Itens

Nesta seção os aspectos necessários são discutidos para se familiarizar com o uso de LabVIEW, inclusive as janelas Painel (Panel) e Diagrama (Diagram), menus do LabVIEW e a janela de hierarquia.

Também são discutidos outros aspectos necessários, como o uso do menu e da barra de ferramentas, criação de objetos e outros.

1.1. Painel Frontal e Diagrama de Fluxo de Dados

Cada VI tem duas janelas separadas, mas relacionadas entre si, que são Painel Frontal (Panel) e o Diagrama de Fluxo de dados (Diagram). Você pode comutar entre ambas as telas com o comando **Show Panel/Show Diagram** (Mostrar Painel/Mostrar Diagrama) do menu **Windows** (Janelas). Usando os comandos **Tile** dentro desse mesmo menu, podemos posicionar as janelas Painel e Diagrama uma ao lado da outra ou uma sobre a outra.

1.1.1. Painel Frontal

Figura 1 – Janela do Painel Frontal

É uma interface interativa entre o usuário e o programa. É aonde o usuário entra com os dados usando o mouse ou o teclado, e então vê os resultados na tela do computador. Quer dizer, o Painel Frontal é uma janela de execução. Como podemos observar na *Figura 1*.

1.1.2. Diagrama de Fluxo de Dados

Figura 2 – Janela do Diagrama de Fluxo de Dados

É a representação de um programa ou algoritmo. É aonde o programador cria o seu programa.

1.2. Menus do LabVIEW

A programação em LabVIEW força utilizar com freqüência os diferentes menus. A barra de menu da parte superior da janela de um VI contém diversos menus **pull-down**. Quando clicamos sobre um item ou elemento desta barra, aparece um menu. Este menu contém elementos comuns a outras aplicações do Windows, como **Open** (Abrir), **Save** (Salvar) e **Paste** (Colar), e muitos outros particulares do LabVIEW.

A seguinte figura mostra a barra de menu quando a janela Painel está ativa. O item **Show Functions Palette** substitue o **Show Controls Palette** quando a janela Diagrama está aberta.

Figura 3 – Menu pop-up do item Windows

O menu do LabVIEW que utilizaremos com mais freqüência é o menu **pop-up** de objetos, o qual habilitamos posicionando o cursor sobre o objeto em questão e clicando o botão direito do mouse. Se o clique for feito sobre um espaço vazio, o menu obtido será o referente à função da ferramenta selecionada.

1.3. Barra de Ferramentas

Podemos criar ou mudar um VI quando este está no modo **Edit**. Nele, as ferramentas de edição são habilitadas, ele se posiciona abaixo da barra de menu da janela, como está indicado abaixo:

Figura 4 – Barras de Menu e Edição

Quando estamos prontos para executar o nosso VI, selecionamos **Change to Run Mode** (Troca de Modo de Execução) a partir do menu **Operate** ou se o que queremos é executar o VI a partir do modo **Edit** sem passar para o modo **Run**, temos que clicar na seta de execução. Fazendo isto compilamos o VI e colocamos no modo **Run**.

1.4. Paleta de Ferramentas (Tools)

Figura 5 – Paleta de Ferramentas

A Paleta de Ferramentas representa os modos de operações especiais do mouse. Nós a usamos para selecionar funções específicas de edição ou execução. É utilizada tanto no Painel Frontal quanto no Diagrama de Fluxo de Dados.

- A ferramenta **Operate Value** (Valor Operativo) administra os controles do Painel Frontal (e os indicadores do modo **Edit**). É a única ferramenta disponível no modo **Run**.

- A ferramenta **Position/Size/Select** (Posição/Tamanho/Seleção) seleciona, move e redimensiona objetos.

- A ferramenta **Edit Text** (Editar Texto) cria e edita textos.

- A ferramenta **Connect Wire** (Conexão de linhas) engloba os objetos do Diagrama de Fluxo de Dados e conecta os conectores dos terminais do VI aos controles e indicadores do painel frontal.

- A ferramenta **Object Popup** (Menu pop-up do objeto) exibe o menu pop-up associado ao objeto. Tem o mesmo efeito que se apertarmos o botão direito do mouse sobre o objeto.

- A ferramenta **Scroll Window** (Deslocamento da tela) desloca a tela na direção que queremos para ver possíveis áreas ocultas.

- A ferramenta **Set/Clear Breakpoint** (Estabelecer/Retirar pontos de parada) permite pôr tantos pontos de ruptura quanto desejamos no diagrama de fluxo de dados. Quando se chega à um deles durante a execução, o LabVIEW comuta automaticamente ao diagrama de fluxo de dados. Usamos esta mesma ferramenta para remover os pontos.

- A ferramenta **Probe Data** (Sonda de dados) checa valores intermediários no VI que está executando e procedimentos questionáveis ou resultados inesperáveis.

- A ferramenta **Get Color** (Captar cor) recolhe uma amostra da cor para ser utilizada posteriormente.

- A ferramenta **Set Color** (Selecionar Cor) colore diversos objetos e fundo.

Você pode mudar a ferramenta fazendo o seguinte:

- Clicando no ícone da ferramenta que será utilizada.
- Usando a tecla TAB para selecionar a ferramenta desejada.
- Pulsando o SPACE para mudar entre a ferramenta **Operate Value** e **Position/Size/Select** quando a janela Painel está ativa, e entre as ferramentas **Writing** e **Position/Size/Select** quando a janela de Diagrama é a ativa.

1.5. Paleta de Controles (Controls)

Figura 6 – Paleta de Controles

É utilizada no Painel Frontal. Representa as variáveis de entrada e saída do programa.

1.6. Paleta de Funções (Functions)

É utilizada no Diagrama de Fluxo de Dados. Representa as variáveis internas, funções e subprogramas que são utilizados no código fonte do programa.

Figura 7 – Paleta de Funções

2. Aplicação

2.1. Criação de objetos

Para elaborar um programa no Painel Frontal, temos que localizar nele os objetos desejados mediante a Paleta de Controles. Criamos os objetos sobre o Diagrama de Fluxo de Dados selecionando-os na Paleta de Funções. Por exemplo, se queremos criar um **knob** (botão giratório) sobre o Painel Frontal, primeiro temos que selecioná-lo desde do subitem **Numeric** (Numérico) a partir da Paleta de Controles.

O objeto aparecerá na janela Painel com um retângulo negro ou cinza que representa uma etiqueta de identificação ou **Label**. Se queremos utiliza-la neste mesmo momento, introduziremos o texto pelo teclado.

Quando criamos um objeto no Painel Frontal, ao mesmo tempo se cria o terminal correspondente no Diagrama de Fluxo de Dados. Este terminal é usado tanto para ler dados desde um controle como para enviá-los a um indicador.

Se selecionar **Show Diagram** (Exibir Diagrama) a partir do menu **Windows**, poderemos ver o diagrama correspondente ao Painel Frontal. Este diagrama conterá terminais para todos os controles e indicadores do Painel Frontal.

Todos os objetos em LabVIEW têm associado menus **pop-up**, os quais podemos obter pressionando o botão direito do mouse sobre o objeto. Mediante a seleção de suas diferentes opções podemos atuar sobre determinados parâmetros como, o aspecto ou comportamento desse objeto.

Por exemplo, se não tivéssemos introduzido o texto na etiqueta do controle anterior, esta teria desaparecido ao clicarmos em qualquer outro lado. Para voltar a visualizá-la teríamos que obter o menu **pop-up** deste controle e selecionar **Label** do menu **Show**.

Para exibir a Paleta de Controles ou de Funções temos que clicar com o botão direito do mouse em qualquer área livre da tela: aparecerá a Paleta de Controles ou de Funções dependendo em qual janela nós estamos, Painel ou Diagrama, respectivamente.

2.2. Tipos de dados em LabVIEW – Controladores e Indicadores

O LabVIEW oferece uma grande variedade de tipos de dados com que podemos trabalhar respondendo as necessidades reais. Um dos aspectos mais significantes no LabVIEW é a diferenciação que se efetua no Diagrama de Fluxo de Dados entre os diferentes tipos de controles e indicadores, baseado em que cada um deles tem uma cor própria.

Deste modo, e como conseqüência de um memorização ou assimilação prática, será muito fácil identificarmos e reconhecermos imediatamente se estamos trabalhando com o tipo de dado apropriado. Distinguimos os seguintes tipos, os quais podem funcionar tanto como controladores como indicadores (entre parênteses está a representação da cor como está representado no Diagrama de Fluxo de Dados):

- Boolean (verde claro)

Os tipos de dados booleanos são inteiros de 16 bits. O bit mais significativo contém o valor booleano. Se no bit 15 se coloca o valor 1, então o valor do controle ou indicador é **true** (verdadeiro); ao contrário, se este bit 15 vale 0, o valor da variável booleana será **false** (falso).

- Numérico: há diferentes tipos:

Extended Precision (laranja)

De acordo com o padrão do computador que estamos usando os números reais com precisão extendida apresentam o seguinte formato:

Macintosh:	96 bits (formato de precisão extendida MC68881 - MC68882)
Windows:	80 bits (formato de precisão extendida 80287)
Sun:	Formato de 128 bits
HP-UX:	São armazenados como os números reais de dupla precisão.

Double Precision (laranja)

Os números reais de dupla precisão cumprem com o formato de dupla precisão IEEE de 64 bits.

Single Precision (laranja)

Os números reais de precisão simples cumprem com o formato de precisão simples IEEE de 32 bits.

Long Integer (azul)

Os números inteiros longos têm um formato de 32 bits, com ou sem sinal.

Word Integer (azul)

Estes números têm um formato de 16 bits, com ou sem sinal.

Byte Integer (azul)

Têm um formato de 8 bits, com ou sem sinal.

Unsigned Long (azul)	Inteiro longo sem sinal.
Unsigned Word (azul)	Word sem sinal.
Unsigned Byte (azul)	Byte sem sinal.
Complex Extended (laranja)	Número complexo com precisão extendida.
Complex Double (laranja)	Complexo com dupla precisão.
Complex Single (laranja)	Complexo com precisão simples.

- Array & Cluster

Array

Um Array (vetor) é um conjunto de tamanho variado de elementos que são do mesmo tipo. Os elementos do vetor são ordenados, e você tem acesso a um elemento individualmente quando se ordena o vetor. O primeiro índice é zero, o qual indica que a dimensão do vetor varia de zero a n-1, onde n é o número de elementos do vetor.

Cluster (marrom ou rosa)

Um **cluster** (agrupamento) armazena diferentes tipos de dados de acordo com as seguintes normas: os dados escalares são armazenados diretamente no agrupamento; e os vetores, strings e caminhos são armazenados indiretamente.

- Strings (rosa)

A string é utilizada quando se tem uma seqüência de caracteres alfanuméricos.

- Paths (verde escuro)

O LabVIEW armazena os componentes, tipo e número, de um **path** (caminho) em palavras inteiras, seguidas imediatamente pelos componentes do caminho.

3. Programação Estrutural

Muitas vezes é necessário executar um mesmo conjunto de sentenças um determinado número de vezes, ou que estes se repitam enquanto são completadas certas condições. Também pode acontecer que desejamos executar uma ou outra sentença

dependendo das condições fixadas ou simplesmente força que algumas executem sempre antes das outras.

Para isto o LabVIEW tem quatro estruturas facilmente diferenciáveis pela sua aparência e que estão disponíveis na opção **Structures** da Paleta de Funções da janela Diagrama:

Figura 8 – Estruturas: Sequence, Case, For Loop e While Loop

3.1. Estrutura Iterativas: For Loop e While Loop

3.1.1. For Loop

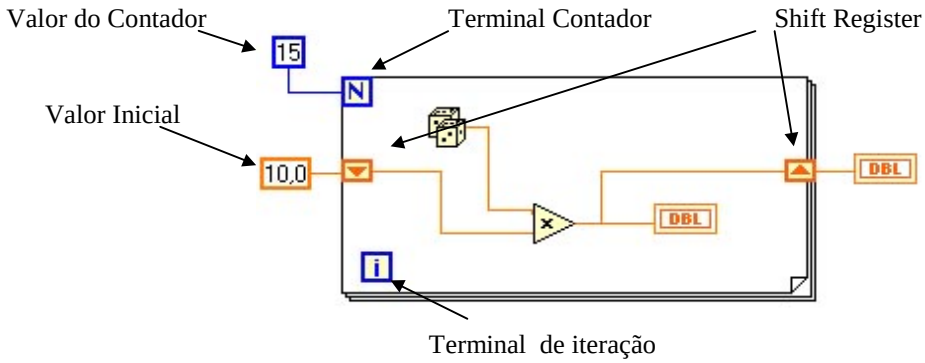

Figura 9 – Exemplo de uma estrutura For Loop

Usaremos **For Loop** quando queremos que uma operação se repita um determinado número de vezes. Seu equivalente a linguagem convencional é:

For i = 0 a N-1
    Executa Subdiagrama

Quando colocamos um **For Loop** na janela Diagrama observamos que temos associado os terminais:

1. Terminal Contador: contém o número de vezes que se executará o subdiagrama criado dentro da estrutura. O valor do contador se fixará externamente.

2. Terminal de Iteração: indica o número de vezes que a estrutura foi executada: 0 durante a primeira iteração, e 1 durante a segunda e assim até N-1. Ambos os terminais são acessíveis do interior da estrutura, quer dizer, seus valores poderão fazer parte do subdiagrama mas em nenhum caso eles poderão ser modificados.

3.1.2. While Loop

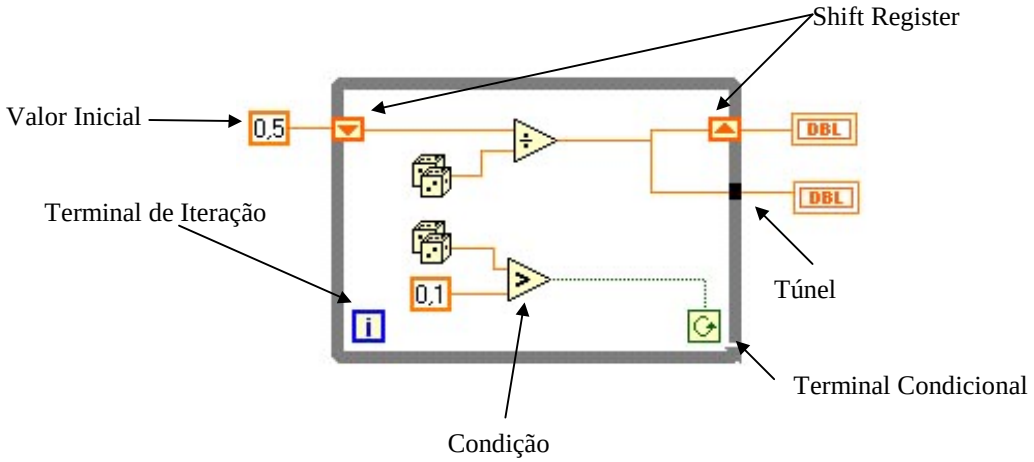

Figura 10 – Exemplo de uma estrutura While Loop

Usaremos **While Loop** quando queremos que uma operação se repita enquanto uma determinada condição é verdadeira. Seu equivalente na linguagem convencional é:

Do
Executar subdiagrama
While condição **is True**

(Embora esta estrutura seja mais semelhante ao comando *Repeat-Until*, já que se repete no mínimo uma vez, independente do estado da condição).

Igual ao **For Loop**, ele contém dois terminais:

1. Terminal Condicional: à ele conectaremos a condição que fará que o subdiagrama seja executado. O LabVIEW conferirá o estado deste terminal ao término de cada iteração, se o seu valor for **true** (verdadeiro), ele continuará, mas se o valor for **false** (falso), parará a execução.

2. Terminal de Iteração: indica o número de vezes que o subdiagrama foi executado e o valor mínimo, sempre será 0.

3.2. Registros de Deslocamento

Os registros de deslocamento ou **shift register** estão disponíveis tanto no **For Loop** como no **While Loop**, que permitem transferir os valores do final de uma iteração ao princípio da mesma.

Inicialmente o **shift register** tem um par de terminais colocados em ambos os lados do **Loop**; o terminal da direita armazena o valor final da iteração até que um novo faça que este valor se desloque para o terminal da esquerda, ficando no da direita o novo valor. Um mesmo registro de deslocamento pode ter mais de um terminal no lado esquerdo; para adicioná-lo escolheremos a opção **Add Element** (adicionar elemento) do menu **pop-up**. Quanto mais terminais tenhamos no lado esquerdo mais valores de iterações anteriores poderemos armazenar.

O menu **pop-up** tem outros dois comandos:

Remove Element – Elimina sempre um terminal do lado esquerdo quando o registro de deslocamento tem mais de um elemento associado.

Remove All – Elimina todos os registros de deslocamento, tanto os terminais da esquerda como os da direita.

Um mesmo **Loop** pode ter vários registros de deslocamentos sendo conveniente inicializá-los, de forma que os terminais da esquerda tenha o valor desejado quando acontecer a primeira iteração. Os registros de deslocamentos podem sempre trabalhar com qualquer tipo de dados sendo eles conectados a cada terminal, sendo esses dados do mesmo tipo.

Ao finalizar a execução de todas as iterações, o último valor ficará no terminal da direita; unindo-o a um indicador de mesmo tipo de dado fora do **Loop**, podendo obter seu valor.

Mas existe outra possibilidade para passar os dados de um modo automático do interior da estrutura para o exterior. Quando uma linha atravessa os limites do **Loop**, aparece na borda um novo terminal chamado túnel que faz a conexão entre o interior e o exterior, de forma que os dados fluem através do túnel depois de cada iteração de **Loop**, podendo armazenar deste modo não somente o último valor de todas as iterações como também os valores intermediários. Esta possibilidade de acumular automaticamente vetores em seus limites se chama **auto-indexing** ou autoindexado.

O LabVIEW habilita por padrão o **auto-indexing** no **For Loop** já que é mais freqüente usar esta estrutura para criar vetores do que no **While Loop**, no qual esta opção é desabilitada por padrão e cuja a utilização poderia causar problemas de memória, já que não sabemos quantas vezes vai ser executado. No entanto, fazendo **pop-up** no túnel podemos habilitar ou desabilitar esta opção.

3.3. Estruturas Case e Sequence

Estes tipos de estruturas se diferem ao mesmo tempo das iterativas porque podem ter múltiplos subdiagramas, dos quais somente um é visível por vez. Na parte superior de cada estrutura existe uma pequena janela que mostra o identificador do subdiagrama que está sendo mostrado. Em ambos os lados desta janela existem dois botões que decrementam ou incrementam o identificador de forma que possamos ver o resto de subdiagramas.

3.3.1.Case

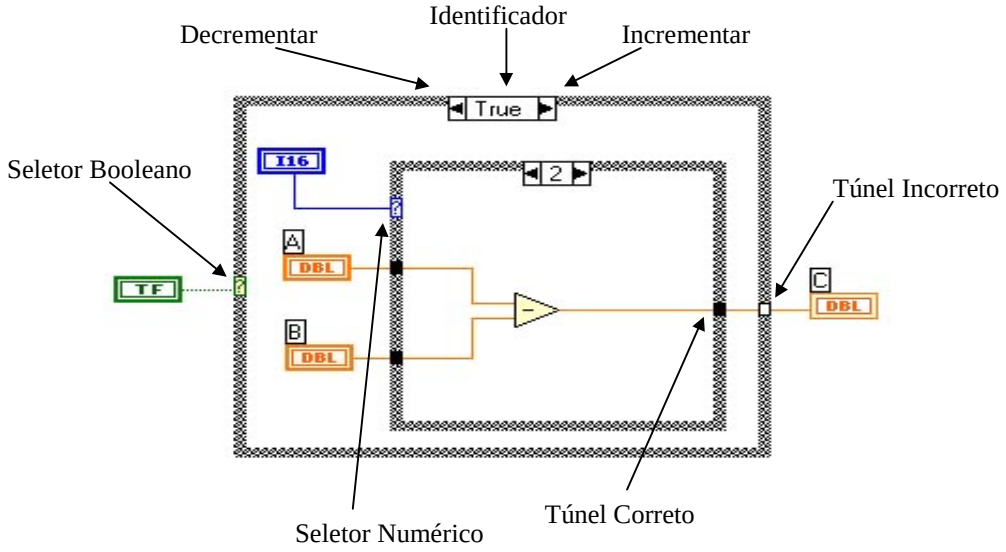

Figura 11- Exemplo de uma estrutura Case

Usaremos a estrutura **Case** naquelas situações em que o número de alternativas disponíveis sejam duas ou mais. Dependendo do valor que tenha o seletor dentro dos n possíveis valores, será executado um dos n subdiagramas correspondente ao valor escolhido.

A estrutura **Case** consiste de um terminal chamado seletor e um conjunto de subdiagramas que estão dentro de um evento e que esteja etiquetado por um identificador do mesmo tipo que o seletor; este pode ser booleano ou numérico. Se conectar um valor booleano ao seletor, a estrutura terá dois eventos: **false** e **true**. Mas se conectar um valor numérico a estrutura poderá ter até 214 eventos. Neste caso a estrutura **Case** engloba duas sentenças diferentes de outras linguagens convencionais:

1. **lf** condição **True**
   Then
   Executar caso **True**
   Else
   Executar caso **False**

2. **Case** seletor **of**
   l: executar caso **1**;
   n: executar caso **n**
   End

A estrutura **Case** não tem os registros de deslocamento das estruturas iterativas mas podemos criar os túneis para retirar ou introduzir dados. Se um caso ou evento provê um dado de saída para uma determinada variável será necessário que todos os demais também façam o mesmo; se não acontecer desta maneira será impossível executar o programa.

3.3.2. Sequence

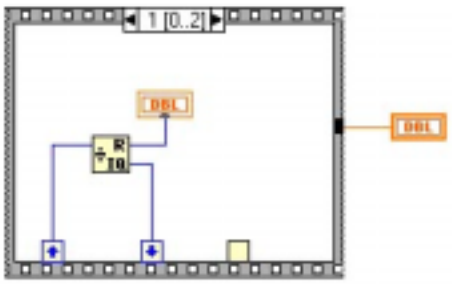

Figura 12 – Exemplo de uma estrutura Sequence

Esta estrutura não tem seu homólogo nas diferentes linguagens convencionais, já que nestes as seqüências são executadas na ordem de aparecimento mas, como já sabemos, no LabVIEW uma função é executada quando tem disponível todos os dados de entrada. Isto produz uma dependência de dados de modo que a função a qual recebe um dado diretamente ou indiretamente de outra função seja sempre executada posteriormente, criando um fluxo de programa.

Mas existem ocasiões em que esta dependência de dados não existe e é necessário que um subdiagrama seja executado antes de outro; é nesses casos que usaremos a estrutura **Sequence** para forçar um determinado fluxo de dados. Cada subdiagrama estará contido em um **frame** ou quadro e este será executado em ordem de aparecimento: Primeiro o *frame 0 ou quadro 0*, depois o *quadro 1* e assim, sucessivamente, até o último *frame*.

Ao contrário do **Case**, se um quadro gera um dado de saída para uma variável os demais não terão por que gerá-lo. Mas teremos que lembrar de que o dado somente estará disponível quando o último quadro for executado e não o quadro que transfere o dado.

Devido à semelhança dos menus **pop-up** das estruturas **Case** e **Sequence** vamos estudá-las de forma conjunta indicando em cada caso as possíveis diferenças que podem existir:

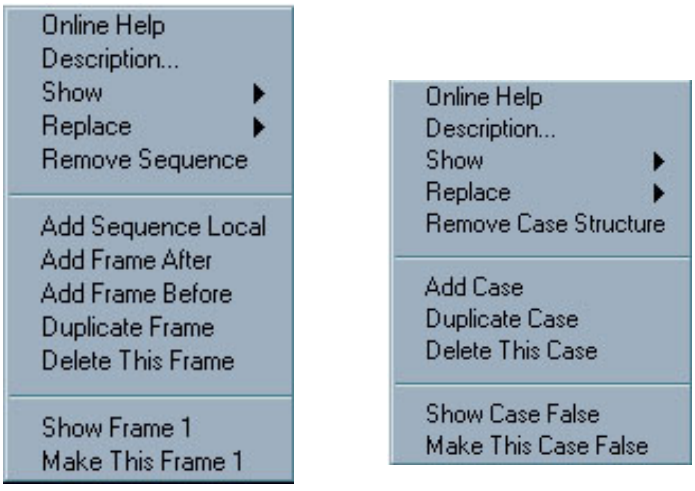

Figura 13 – Menu pop-up das estruturas Case e Sequence

- **Online Help –** Exibe o conteúdo de ajuda sobre a estrutura.

- **Description** – Permite acrescentar comentários.

- **Show –** Oculta ou visualiza a etiqueta de identificação da estrutura e, se não existe, permite colocá-la.

- **Replace** – Altera a estrutura **Case** ou **Sequence** para qualquer outra função da paleta **Structs & Constants**.

- **Remove Case Structure** ou **Sequence –** Remove a estrutura **Case** ou **Sequence** e todos os subdiagramas, menos o que está sendo visualizado no momento da execução deste comando.

- **Add Sequence Local** – Esta opção somente está disponível no menu da estrutura **Sequence** e é usado para passar dados de um quadro para outro. Uma pequena seta apontando para o exterior da estrutura indica o quadro de origem da seqüência local, enquanto uma seta que aponta para o interior indica que a seqüência local contém um dado de saída. Todos os quadros posteriores ao que contém a seqüência local que origina o dado poderá disponibilizá-lo, não sendo assim para os quadros anteriores os quais aparecerá um quadrado vazio que indicará que os dados não estão disponíveis.

- **Add Case –** Esta opção adiciona um evento após o evento que está ativo no momento da ação.

- **Show Case** ou **Show Frame –** Nos ermite ir diretamente ao subdiagrama que queremos visualizar sem ter que passar por todos os eventos ou quadros intermediários que podem existir. Ao clicar nesta opção, um menu contendo todos os identificadores aparecerá e somente teremos que mostrar com o cursor do mouse o qual desejamos ver. Se somente tiver dois subdiagramas aparecerá diretamente o nome do único identificador que podemos visualizar, como é o caso do **Case** como seletor booleano.

- **Add Case After** ou **Add Frame After –** Este comando insere um subdiagrama vazio imediatamente depois do qual se esteja visualizando.

- **Add Case Before** ou **Add Frame Before –** Insere um subdiagrama vazio exatamente um nível depois do qual se esteja visualizando.

- **Duplicate Case** ou **Duplicate Frame –** Insere uma cópia de subdiagrama visível imediatamente depois do mesmo.

- **Make This Case** ou **Make This Frame –** Move um subdiagrama para outra posição.

- **Remove Case** ou **Remove Frame –** Elimina o subdiagrama visível. Este comando não está disponível se somente existir um Case o um Frame.

Na estrutura **Sequence** se somente houver uma seqüência, não aparecerá nenhum identificador no quadro; enquanto existir mais de um, ele nos indicará em qual estamos e

quantos existem. O mesmo acontece com a estrutura **Case,** só que, neste caso temos, como mínimo, dois possíveis estados. Tudo isso podemos observar na figura a seguir:

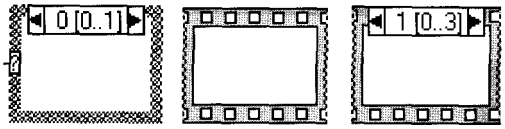

Figura 14 – Estrutura Case e dois modos da estrutura Sequence

3.4. Formula Node

O **Formula Node** é uma função de característica semelhantes as estruturas vistas anteriormente, disponível na paleta **Structs & Constants** da Paleta de Funções, porém, ao invés de conter um subdiagrama, contém uma ou mais fórmulas separadas por um ponto e vírgula. Usaremos **Formula Node** quando quisermos executar fórmulas matemáticas que seriam complicadas de criar utilizando as diferentes ferramentas matemáticas que o LabVIEW incorpora em suas bibliotecas.

Uma vez escrita a fórmula no interior do retângulo somente poderemos adicionar os terminais que farão a função de variáves de entrada ou de saída; para ele aparecerá o menu **pop-up** da estrutura e executaremos o comando **Add Input** (adicionar entrada) ou **Add Output** (adicionar saída).

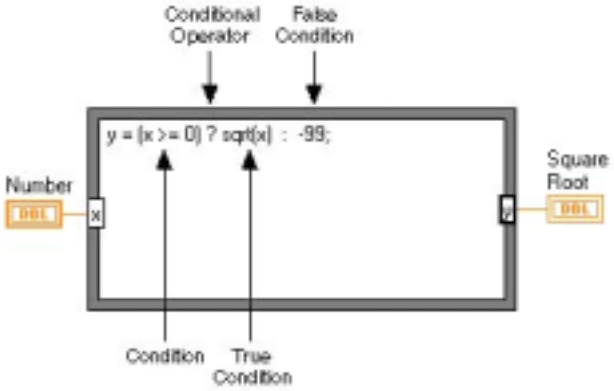

Figura 15 – Exemplo da estrutura Fórmula Node

Cada variável, também, terá outro menu **pop-up** que permitirá defini-la como de saída se anteriormente fora de entrada, ou de entrada se no início tivesse sido de saída (**Change to Output** ou trocar a saída, **Change to Input** ou trocar a entrada). Também podemos eliminá-la mediante o comando **Remove**.

Não há nenhum limite para o número de variáveis ou de fórmulas e nunca poderá haver duas entradas ou duas saídas com o mesmo nome, embora uma saída possa ter o mesmo nome que uma entrada. Todas as variáveis de saída deverão estar assinaladas a uma fórmula pelo menos uma vez.

O tabela mostra para algumas das funções do **Formula Node**:

abs(x)	Retorna o valor absoluto de *x*.
acos(x)	Calcula o coseno inverso de *x* em radianos.
acosh(x)	Calcula o coseno hiberbólico inverso de *x* em radianos.
asin(x)	Calcula o seno inverso de *x* em radianos.
asinh(x)	Calcula o seno hiperbólico inverso de *x* em radianos.
atan(x,y)	Calcula a tangente inversa de *y/x* em radianos.
atanh(x)	Calcula a tangente hiperbólica inversa de *x* em radianos.
cos(x)	Calcula o coseno de *x* em radianos.
cosh(x)	Calcula o coseno hiperbólico de *x* em radianos.
cot(x)	Calcula a cotangente de *x* em radianos.
csc(x)	Calcula a cosecante de *x* em radianos.
exp(x)	Calcula o valor de *e* elevado a *x*.
Ln(x)	Calcula o logaritmo natural de *x*.
Log(x)	Calcula o logaritmo de *x* na base 10.
Log2(X)	Calcula o logaritmo de *x* na base 2.
max(x,y)	Compara *x* com *y*, e retorna o maior valor.

min(x,y)	Compara *x* com *y*, e retorna o menor valor.
mod(x,y)	Calcula o conciente de *x/y*.
rando	Gera um número aleatório entre 0 e 1.
sic(x)	Calcula a secante de *x* em radianos.
sign(x)	Retorna 1 se *x* é maior que 0, 0 se *x* é igual a 0 e -1 se *x* é menor que 0.
sin(x)	Calcula o seno de *x* em radianos.
sinc(x)	Calcula o seno de *x* dividido por *x* em radianos.
sinh(x)	Calcula o seno hiperbólico de *x* em radianos.
sqrt(x)	Calcula a raiz quadrada de *x*.
tan(x)	Calcula a tangente de *x* em radianos.
tanh(x)	Calcula a tangente hiperbólica de *x* em radianos.

3.5. Variáveis Locais e Globais

As variáveis são indispensáveis em qualquer tipo de problemas, já que permitem armazenar a informação necessária para a resolução de problemas.

No LabVIEW todos os controles introduzidos no Painel Frontal que geram um terminal na janela Diagrama serão variáveis identificadas pelo nome representado na etiqueta. Mas pode ocorrer que queiramos usar o valor de certa variável em outro subdiagrama ou em outro VI ou, simplesmente, que queremos armazenar um resultado intermediário. A forma mais simples de fazê-lo é gerando variáveis locais e/ou globais dependendo da aplicação.

3.5.1. Variáveis Locais

Nas variáveis locais os dados são armazenados em alguns dos controladores ou indicadores existentes do Painel Frontal do VI criado; é por isso que estas variáveis não servem para trocar dados entre VI's. A principal utilidade destas variáveis reside no fato de que uma vez criada a variável local não importa o que possa acontecer entre indicador ou um controlador, desde que poderá usar o mesmo diagrama tanto de entrada como saída.

As variáveis locais estão disponíveis no menu **Structs & Constants** da Paleta de Funções e dispõem de seguinte menu **pop-up**:

Figura 16 – Menu pop-up de uma variável local

3.5.2. Variáveis Globais

As variáveis globais são um tipo especial de VI, que unicamente dispõem do Painel Frontal**,** no qual se define o tipo de dado da variável e o nome de identificação indispensável para que possamos nos referir a ela depois.

Quando escolhemos a função **Global** do menu **Structs & Constants** criamos um novo terminal no Diagrama; este terminal corresponde a um VI que inicialmente não contém nenhuma variável. Para poder adiciona-lo daremos um duplo clique no terminal e se abrirá o Painel Frontal. Uma vez aberto, as variáveis são definidas igual a qualquer controle ou indicador de um VI normal. Podemos criar um VI para cada variável global ou defini-las no mesmo VI, que é a opção mais indicada para qualquer aplicação. Quando terminarmos de colocar todas as variáveis gravaremos o VI e o fecharemos. Se uma vez fechado queremos adicionar variáveis novas, bastará voltar a abri-lo e adicionar as mudanças necessárias. Para adicionar novos terminais que fazem referência às variáveis globais criadas, não executamos a função **Global** já que isto criaria um novo VI então abriremos o VI existente por meio do comando **VI..** da Paleta de Funções e selecionaremos a variável através do comando **Select ltem** do menu **pop-up**. Também, este mesmo menu tem outra opção que nos permite usar uma variável já criada para ler dados ou armazená-los, se trata do comando **Change To Read Global** ou **Change To Write Global**.

3.6. Attribute Node

Os **Attribute Node** ou nodos de atributos podem ser considerados como variáveis que dependem unicamente do terminal o qual foram criados e que permitem ler ou modificar atributos do Painel Frontal de um controle ou indicador como, por exemplo, mudar a cor, torná-lo invisível, desativá-lo, ler as posições do cursor, mudar escalas, etc..

Para criar um nodo de atributo basta abrir o menu **pop-up** do objeto e selecionamos a opção **Create**. Podemos criar um nodo de atributo ou uma variável local. Uma vez criado aparece no Diagrama um novo nodo que pode ser tanto de escrita como de leitura. Uma pequena seta à esquerda do nodo indica que isto é de escrita, enquanto uma seta à direita indica que é de leitura. Os nodos de atributos também têm seus próprios menus **pop-up** como está indicado.

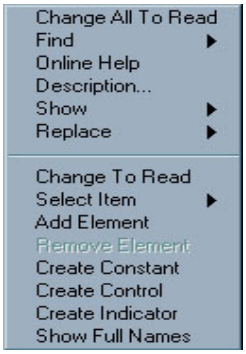

Figura 17 – Menu pop-up de um nodo de atributo

Os atributos para um controle numérico são:

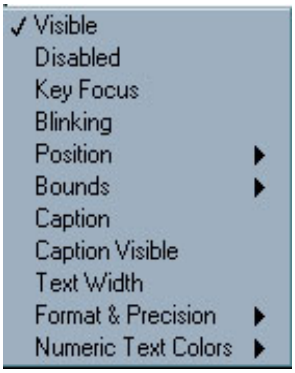

Figura 18 – Menu pop-up com os atributos de um controle numérico

E para uma string:

- **Remove Element** – Elimina o terminal selecionado.

- **Add Element** – Adiciona um novo terminal.

Alguns controles como o **Graph** têm um grande número de atributos. Muitos destes atributos se agrupam em categorias, como é o caso de **Y Scale Info** para um indicador **XY Graph**. Uma pequena seta à direita indica que se trata de uma categoria.

Podem ser selecionados todos os atributos de uma categoria de uma única vez mediante ao comando **All Elements** (todos os elementos), embora também podemos selecioná-los individualmente escolhendo o atributo específico.

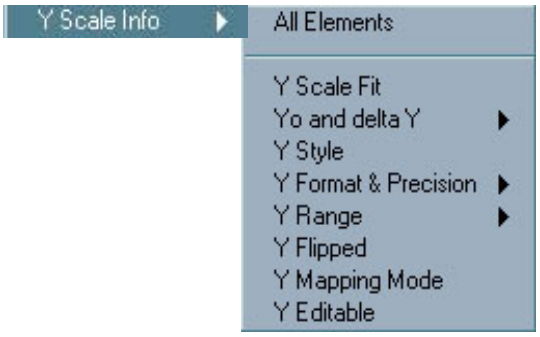

Figura 19 – Atributo Y Scale Info com as suas categorias

3.7. Indicadores Gráficos

Em muitas ocasiões é necessário para uma maior compreensão dos resultados obtidos representá-los graficamente. Para isto o LabVIEW dispõem de cinco tipos de gráficos acessíveis a partir da Paleta de Controles do Painel Frontal e selecionar o item **Graph**, que está dividido em dois grupos: indicadores **Chart** e indicadores **Graph**.

Figura 20 – Exemplo de 5 tipos de gráficos: 1. Waveform Chart, 2. Waveform Graph, 3. XY Graph, 4. Intensity Chart e 5. Intensity Graph.

Um indicador **Graph** ou indicador gráfico é uma representação bidimensional de um ou mais gráficos, o gráfico recebe os dados como um bloco. Um indicador **Chart** de linhas também exibe gráficos, mas este recebe os dados e os exibe ponto por ponto ou vetor por vetor, retendo um certo número de pontos na tela por meio de um buffer disponível para ele.

3.7.1. Indicadores Chart

3.7.1.1. Waveform Chart

Waveform Chart é um tipo especial de indicador numérico que mostra um ou mais gráficos, retendo na tela um certo número dados definidos por nós mesmos. Os novos dados são adicionados ao lado dos existentes, de forma que eles possam ser comparados entre eles.

Os dados podem ser passados um a um para o **Chart** ou mediante vetores. Evidentemente é mais conveniente passar múltiplos pontos ao mesmo tempo já que desta maneira somente é necessário desenhar o gráfico uma vez e não um por cada ponto.

É possível desenhar vários gráficos em um mesmo **Chart**, unindo os dados de cada gráfico em um **cluster** de escalares numéricos de forma que cada escalar que contém o **cluster** é considerado como um ponto de cada um dos gráficos para uma mesma abcissa. Se pode economizar tempo unindo o **cluster** em vetores e depois transferindo todo o vetor para o gráfico.

Exibindo o menu pop-up temos acesso as seguintes opções:

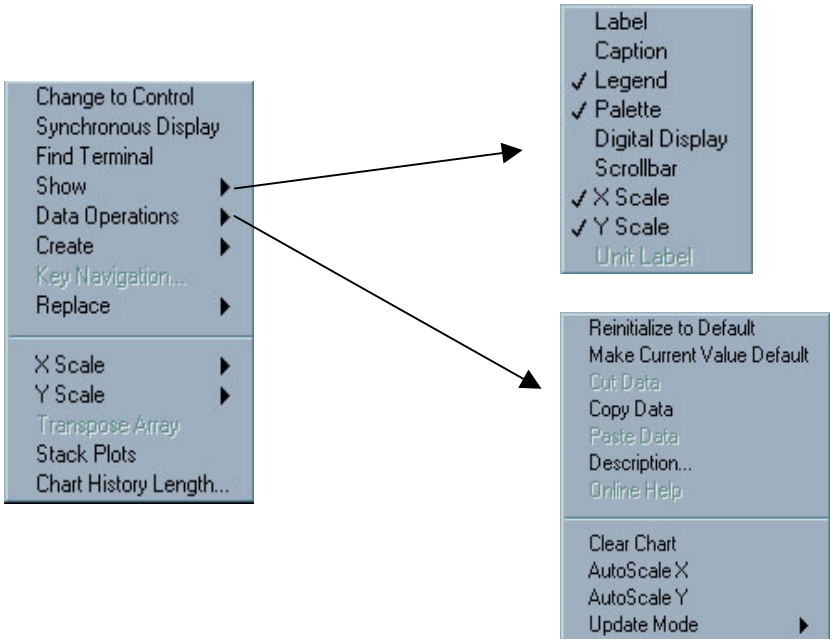

Figura 21 –Menu pop-up do Waveform Chart

3.7.1.2. Intensity Chart

Por meio do **Intensity Chart** podemos mostrar dados tridimensionais colocando blocos de cores sobre planos cartesianos. Para eles criaremos vetores bidimensionais de números onde os índices de um elemento corresponderão às coordenadas X e Y, e o conteúdo da coordenada Z, que terá associado uma cor para cada valor possível. Previamente será necessário definir a escala de cores que vamos utilizar através do nodo de atributo mediante o item **Z Scale lnfo**: **color, array** ou **color table**, ou através da coluna de cores visualizada próximo ao gráfico.

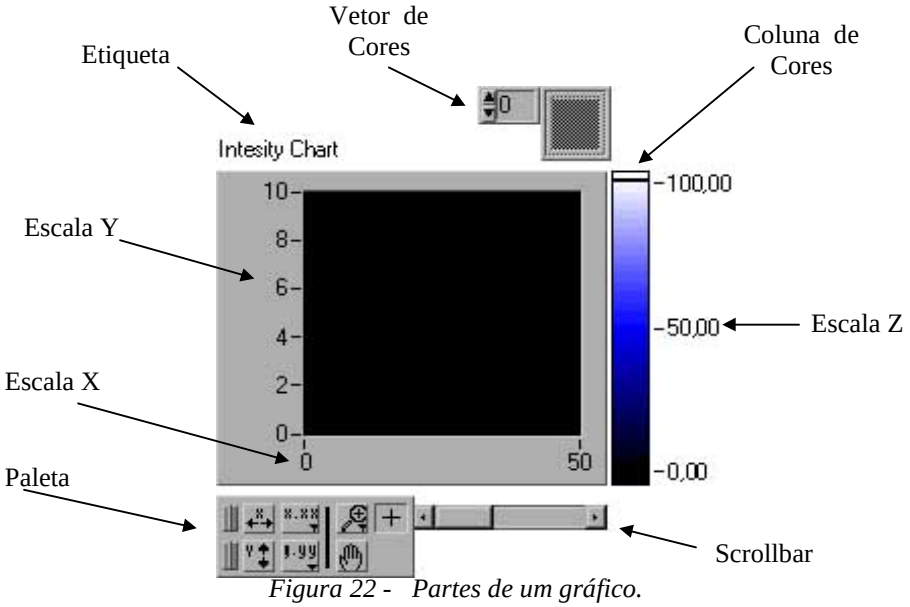

Figura 22 - Partes de um gráfico.

Evidentemente, a escala de cores que podemos visualizar dependerá da resolução do monitor.

Cada vez que é enviado um novo conjunto de dados, estes aparecerão representados à direita dos já existentes. **Intensity Chart** suporta os três modos de visualização do **Waveform Chart** e também dispõem de um buffer cujo o tamanho é, por padrão, 128 pontos. As opções disponíveis para **Intensity Chart** são praticamente as mesmas que para **Waveform Chart,** unicamente, porque existe uma nova coordenada. Aparecem no menu as opções para esta coordenada, que são:

- **Show Ramp –** Visualiza ou oculta a coluna de cores.
- **Show Color Array –** Permite fixar as cores à coluna de cores.
- **Show Z Scale –** Visualiza ou oculta a escala Z.
- **AutoScale Z –** Ajusta de forma automática o eixo de valores de Z para a escala de cores.
- **Z Scale –** Permite escolher o estilo da escala, tipo de grade, ponto inicial, incremento entre ponto e ponto, formato e precisão destes pontos.

3.7.2. Indicadores Graph

3.7.2.1. Waveform Graph

Waveform Graph representa uma série de valores Y igualmente distribuídos dado sempre uma distância delta de X (ÄX) começando a partir de um valor inicial Xo. Para um mesmo ponto X, somente pode corresponder a um valor de Y. Quando é representada uma nova série de dados, ao contrário do que aconteceu nos indicadores **Chart**, estes dados substituem os já existentes em vez de serem adicionados ao lado, e perdem os valores representados previamente.

Existem duas possibilidades para representar um único gráfico em um **Waveform Graph**. A primeira consiste em unir um vetor de valores numéricos diretamente ao gráfico de forma que este interprete cada valor como um ponto novo começando em X = 0 e incrementando X em 1 para cada ponto.

A segunda consiste em criar um **cluster,** o qual, junto com o vetor de valores, se indica o valor inicial Xo e o incremento ÄX.

Existe a possibilidade de representar mais de um gráfico em um mesmo **Waveform Graph**. É necessário unir os dados de diferentes gráficos em um formato que o LabVIEW saiba interpretar. Usar um formato ou outro, virá determinado principalmente pelas características dos gráficos à serem exibidos. Deste modo, se todos os gráficos têm uma mesma escala X e um mesmo número de pontos, bastará criar um vetor bidimensional de valores numéricos onde cada linha de dados é um único gráfico. O LabVIEW interpretará estes dados como pontos no gráfico começando em X = 0 e incrementando-o em 1. Se nos interessa mudar o ponto inicial ou o incremento de X, criaremos um **cluster** que conterá o vetor bidimensional e os valores de Xo e Äx.

Por meio do comando **Transpose Array** do menu **pop-up** podemos fazer com que o LabVIEW interprete as colunas como diferentes gráficos em vez das linhas.

Pode acontecer que o número de elementos de cada gráfico seja diferente. Neste caso é necessário criar um **cluster** para cada vetor de dados e depois unir todos os **clusters** em um vetor. Isto é necessário porque o LabVIEW não permite criar vetores de vetores. Se igual previamente nos interessa que o ponto inicial seja diferente de zero ou que o incremento seja diferente de 1, criaremos um **cluster** que contenha um vetor de **clusters** de vetor e os novos valores de Xo e ÄX.

Finalmente, se nem a escala e o número de pontos do gráfico é o mesmo para todos eles, o que faremos será criar um **cluster** para cada gráfico que conterá um vetor de dados, um valor Xo e um valor ÄX. E com todos os **clusters** dos diferentes gráficos criaremos um vetor. Este último formato é o mais completo de todos porque permite fixar um valor Xo e um valor ÄX diferente para cada gráfico.

3.7.2.2. XY Graph

Em **XY Graph** um ponto X, pode ter vários valores Y, o que permite, por exemplo, desenhar funções circulares. **XY Graph** representa uma coordenada (X,Y) onde os valores de X não têm porque estarem igualmente distribuídos como acontecia no **Waveform Graph**.

Para representar um único gráfico em um **XY Graph** existem duas possibilidades. A primeira consiste em criar um **cluster** que contém um vetor de dados X e um vetor de dados Y. A segunda consiste em criar um vetor de **clusters**, onde cada **cluster** contém um valor de X e um valor de Y.

Igual ao **Waveform Graph** existe a possibilidade de representar mais de um gráfico em um mesmo **XY Graph**. Mas, neste caso, somente existem dois possíveis formatos derivados dos dois formatos vistos previamente um único gráfico. O primeiro formato é um vetor de gráficos, onde cada gráfico é um **cluster** de um vetor X e um vetor Y. E o segundo formato é um vetor de **clusters** de gráficos, onde cada gráfico é, por sua vez, outro vetor de **clusters** contendo um valor X e um valor Y.

3.7.2.3. Intensity Graph

Intensity Graph é exatamente igual ao **Intensity Chart**, só que o **Intensity Graph** não retém valores anteriores quando um novo bloco de valores é carregado, este substitui ao já existente.

Os comandos disponíveis no menu **pop-up** dos indicadores **Graph** têm as mesmas utilidades que os descritos nos indicadores **Chart**. Somente existe uma diferença importante e é que os indicadores **Graph** dispõem cursores que nos permitem mover pelo gráfico.

3.7.3. Graph Cursors

A paleta de cursores está disponível na opção **Show Cursor Display** do menu **pop-up.**

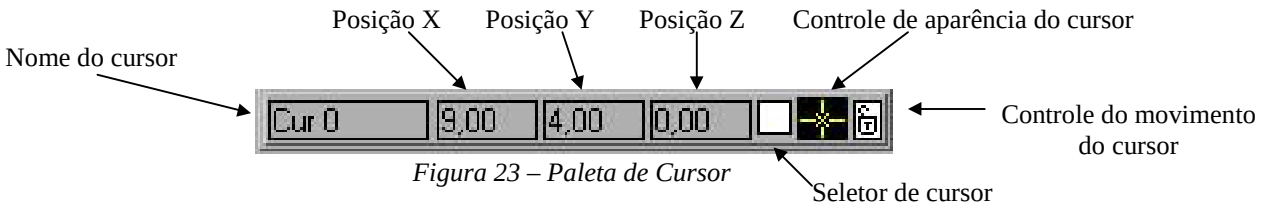

Figura 23 – Paleta de Cursor

Nome do cursor – Permite introduzir uma etiqueta de identificação do cursor. Podemos ter tantos cursores como desenhos.

Posição X, **Posição Y –** Indica as coordenadas em que se encontra o cursor; nos indicadores **Intensity Graph** aparece também a coordenada Z. Podemos mover diretamente o cursor para uma posição concreta introduzindo as coordenadas no ponto desejado.

Seletor do cursor – Seleciona o cursor a mover. Pode ser selecionado por vez tantos cursores como desejamos.

Controle da aparência do cursor – Abrindo o menu abaixo por meio do botão esquerdo do mouse podem modificar algumas características do cursor:

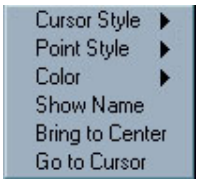

Figura 24 – Menu pop-up com algumas características do cursor

Cursor Style – Seleciona a forma com que se indica o ponto o qual se encontra o cursor.

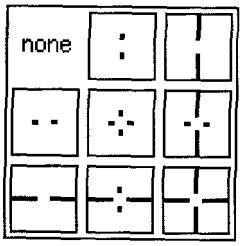

Figura 25 – Menu Cursor Style

Point Style – Seleciona o estilo do ponto que marca a posição do cursor.

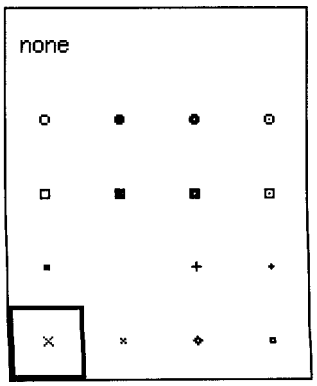

Figura 26 – Menu Point Style

Color – Seleciona a cor do cursor.

Show Name – Exibe o nome do cursor sobre o gráfico.

Bring to Center – Move o cursor até o centro da tela trocando as coordenadas deste.

Go to Cursor – Modifica as escalas X e Y de forma que possamos ver o cursor, mas sem mudar as coordenadas dele.

Controle do movimento do cursor – Um cadeado fechado indica que o cursor se moverá seguindo o gráfico (opções **Look to plot** e **Snap to point**), e se o cadeado estiver aberto indica que o cursor moverá livremente (opção **Free**). Se existisse mais de um gráfico o menu nos permitirá escolher qual dos modos queremos que o cursor se mova. O comando **Allow Drag**, quando está ativo, permite deslocar o gráfico diretamente com o ponteiro do mouse.

Controle da direção do cursor – Move os cursores selecionados ponto a ponto para a direção indicada.

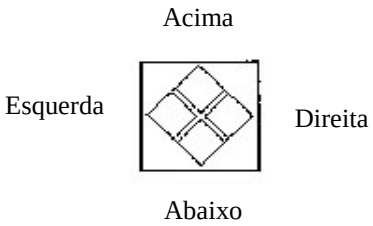

Figura 27 – Controle de direção do cursor

3.8. SubVI

É um subprograma. É quando um VI é usado dentro de outro VI.

Você pode converter uma parte de um VI em um subVI para ser chamado de outro VI. Você seleciona uma parte de um VI, seleciona **Edit** e o subitem **Create SubVI**, e a parte selecionada se torna um subVI. São criados automaticamente controles e indicadores para o novo subVI, o subVI é automaticamente conectado às linhas existentes, e um ícone do subVI substitui a parte selecionada no Diagrama de Fluxo de Dados no VI original.

4. Programas Cliente/Servidor

Nesta seção será explicado, de um modo geral, como funciona os programas Cliente e Servidor, e também será feito uma comparação de um exemplo de troca de mensagens entre vários clientes e um servidor em três diferentes linguagens de programação.

4.1. Comunicação entre Clientes e Servidor

Para ocorrer uma comunicação entre processos, é necessário que os processos utilizem uma linguagem de comunicação comum, que é conhecida como Protocolo.

Um Protocolo de comunicação permite especificar o dado que se pode enviar ou receber e a localização do destino ou código, sem se preocupar como o dado é transferido. O Protocolo traduz os comandos em dados que os drivers da rede podem aceitar. Os drivers transferem o dado através da rede.

Muitos Protocolos aceitam padrões de comunicação. Em geral, um protocolo não é compatível com o outro. Conseqüentemente, quando se inicia uma aplicação de comunicação, primeiro temos que decidir qual o Protocolo vamos utilizar.

Quando criamos uma aplicação, temos mais flexibilidade para escolher o Protocolo. Fatores que afetam a escolha de um Protocolo são: os tipos de máquinas em que os processos vão rodar, o tipo de hardware para rede que temos disponível, e qual complexidade de comunicação que a aplicação requer.

Muitos Protocolos estão incluídos no LabVIEW, alguns deles são específicos para um determinado tipo de computador. Os Protocolos seguintes são utilizados para a comunicação entre computadores.

- Transmission Control Protocol (TCP) – Windows, Macintosh, e UNIX
- User Datagram Protocol (UDP) – Windows, Macintosh, e UNIX
- Active X – somente Windows 95/NT
- Dynamic Data Exchange (DDE) – somente Windows
- AppleEvents – somente Macintosh
- Program-to-Program Communication (PPC) – somente Macintosh

Uma utilização primária para aplicação de softwares na rede é quando temos uma ou várias aplicações utilizando os serviços de outras aplicações. Por exemplo, a comunicação entre Clientes e Servidor ocorre da seguinte forma: o Servidor tem que estar funcionado, conectado à rede, e executando a aplicação que estará disponível para o Cliente. Em um primeiro momento, o Servidor está aguardando qualquer conexão remota de qualquer Cliente. O Cliente tem que saber, neste exemplo, qual o IP do Servidor e a Porta em que a aplicação que ele deseja está sendo executada. E então ele tenta estabelecer a conexão com o Servidor, se obtiver sucesso, o Servidor aceita e libera o cliente. O Cliente fica bloqueado no momento em que está aguardando a autorização do Servidor. O Cliente

então tenta conectar-se à aplicação, que o libera para fazer o seu trabalho, podendo assim o Cliente enviar e receber dados.

Quando o cliente não necessitar mais da conexão ele se desconecta, encerrando a conexão com o Servidor.

Ás vezes não precisamos nos preocupar com a parte do estabelecimento da conexão, pois em algumas aplicações os programas já tratam do estabelecimento da conexão, fazendo com que o usuário só necessite executar o programa Cliente indicando provavelmente a porta e o endereço IP do Servidor.

4.1.1. Ferramentas de Comunicação

Nesta parte será explicado somente o Protocolo TCP, pois ele é o protocolo mais utilizado. O Labview possui algumas funções para comunicação em rede. Como podemos observar na paleta de funções no subitem comunicação, podemos observar os protocolos para comunicação como TCP, UDP e outros, que foram citados na seção 4.1.. Ao clicarmos na opção TCP, aparecerá outra janela com as funções do protocolo TCP, que estão exibidas a seguir:

- **TCP Listen.vi** – Cria um ouvinte e espera por uma conexão TCP que chamará o processo que está rodando no servidor em uma porta especificada. Porta, na programação, é uma "conexão lógica" e específica, que utiliza o protocolo da Internet , TCP/IP, e é o modo, pelo qual o programa cliente especifica um programa servidor particular em um computador que está na rede.

- **TCP Open Connection** – Tenta abrir a conexão TCP com o endereço IP e a porta especificada.

- **TCP Read** – Recebe os bytes que estão sendo enviados por uma conexão TCP.

- **TCP Write** – Envia dados do tipo string para uma conexão TCP especificada.

- **TCP Close Connection** – Encerra a conexão.

- **TCP Create Listener** – Cria um ouvinte para a conexão TCP.

- **TCP Wait on Listener** – Espera por uma conexão TCP na porta especificada.

- **IP To String** – Converte um endereço IP em string.

- **String To IP** – Converte uma string em um endereço IP.

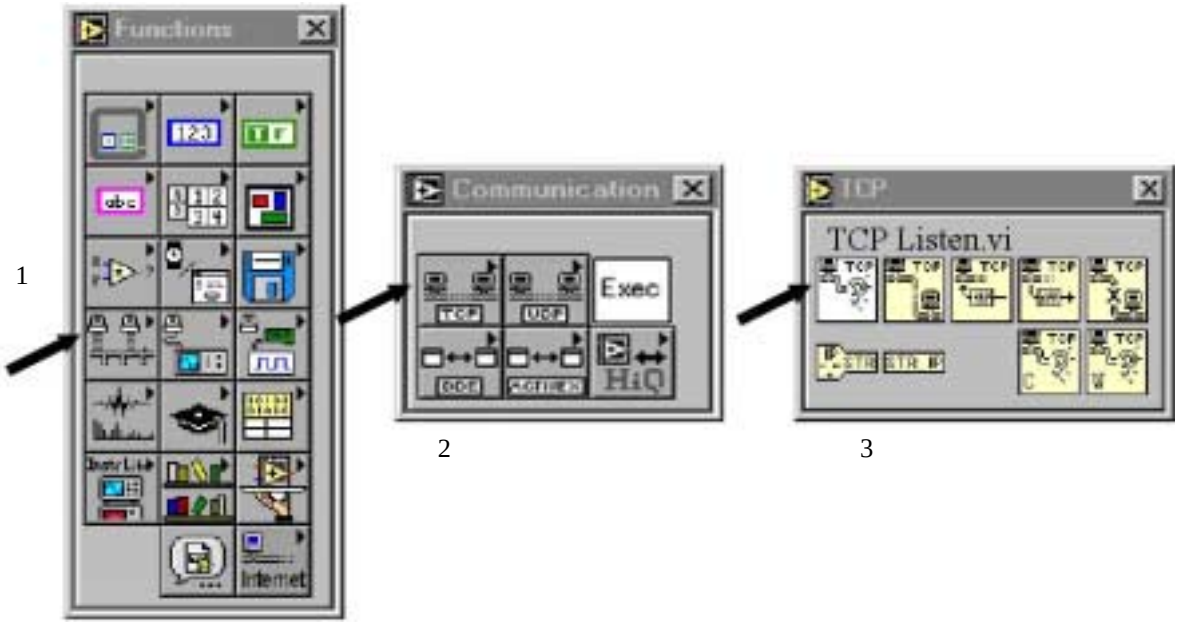

Figura 28 – Localização das ferramentas do TCP. 1. Paleta de Funções; 2.Subitem Comunicação; e 3. Subitem TCP.

Podemos observar que o ícone **TCP Listen.vi** tem o fundo branco. O fundo branco é a representação utilizada para mostrar que o programa pode ser alterado pelo programador. Ao contrário, os outros ícones tem o fundo amarelo, diferente do ícone do **TCP Listen.vi**. Quando o ícone tiver o fundo amarelo, quer dizer, que o LabVIEW não permite acesso ao código-fonte deste VI.

4.2. Comparação entre as linguagens de programação

Nesta parte será exemplificado dois programas, Cliente e Servidor, em cada uma das seguintes linguagens: C, Java e LabVIEW. Estes programas irão trocar mensagens, sendo que o Servidor poderá estar conectado à vários clientes ao mesmo tempo.

4.2.1. Linguagem C

A linguagem C utiliza sockets para a conexão entre o cliente e o servidor. Os sockets são os programas responsáveis pela comunicação ou interligação de outros programas na internet.

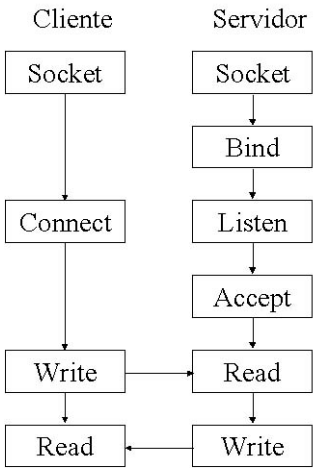

Figura 29 – Funcionamento de uma conexão Cliente/Servidor na Linguagem C.

Na *Figura 29* podemos observar o funcionamento de um programa Cliente/Servidor na Linguagem C, com as funções mais importantes.

Primeiramente, o Servidor cria uma nova conexão ponto-a-ponto através da função socket. Depois, na função Bind, ele associa o socket a um processo local, com informações como endereço e número de porta. Enquanto o Servidor está executando a função Listen, que anuncia que está aguardando alguma conexão e especifica o tamanho de fila de usuários no receptor de mensagens, o Cliente executa a função socket da mesma maneira que o Servidor e a função Connect, que faz um pedido para estabelecer associação com processo remoto. Agora, o Servidor executa a função Accept que bloqueia o Cliente até a conexão ser aceita. Se obtiver sucesso na execução de todas as funções a conexão estará estabelecida e então os dois programas poderão trocar informações entre si. A função Write é utilizada para enviar dados e a função Read para receber dados.

Programa Cliente

```
/* Bibliotecas */
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>
#include<arpa/inet.h>

/* Variaveis Globais */
#define sockport 2780       /* porta reservada para a conexao */
#define tammens 80          /* tamanho maximo da mensagem     */

/* Programa Principal */
void main (int argc, char **argv) {
  char *serv;
  int sockfd;
  struct sockaddr_in endserv;
```

```
  struct sockaddr_in endcliente;       /* armazena em hexadecimal o end do cliente */
  int tamendclie,nwritten,nleft,nread,tamendserv,tamanho,atual;
  char *mensagem,*msg,*buffer;

  /* alocando área para os ponteiros */
  serv = (char*)calloc(16, sizeof(char));
  msg = (char*) malloc(80);
  mensagem = (char*)malloc(80);
  buffer = (char*) malloc(80);

  sockfd = socket(PF_INET,SOCK_STREAM, 0);    /* passando dominio, tipo, protocolo */
  tamendclie = sizeof(endcliente);   /* variavel recebe o tamanho da variavel endcliente */
  bzero ((char*)&endcliente,sizeof(endcliente));

  endcliente.sin_family= AF_INET;
  endcliente.sin_addr.s_addr = htonl(INADDR_ANY);          /* pega o endereco da maquina */
  endcliente.sin_port=htons(sockport);

  /* testa se tem menos de 2 parametros quando manda executar, ex: cliente endservidor porta */
  if (argc == 2)
    strncpy (serv,argv[1],strlen(argv[1]));
  else
    strncpy (serv,"127.0.0.1",strlen("127.0.0.1"));

  bzero ((char*)& endserv,sizeof(endserv));

  endserv.sin_family=AF_INET; /*endereco  da familia Internet */
  endserv.sin_addr.s_addr=inet_addr(serv);
  endserv.sin_port=htons(sockport);
  tamendserv= sizeof(endserv);

  if (connect (sockfd, (struct sockaddr *)&endserv, tamendserv) < 0)    {
     perror ("Erro no connect");
     close (sockfd);
     return;
    }
  while(1)          {
     printf ("* Programa Cliente *\n");
     printf ("\n Mensagem a enviar: " );
     fgets (mensagem,80,stdin);
     tamanho = strlen(mensagem);
     nwritten = write(sockfd,mensagem,tamanho);
     if (nwritten < 0)          {
         perror ("erro do write");
        close (sockfd);
        return;
       }
     nread = read(sockfd,msg,1024);
     if (nread < 0)
       perror("erro no read");
     else
       printf("\n Reposta do servidor: %s\n",msg);
    }
}
```

Programa Servidor

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>
#include<arpa/inet.h>
#include<pthread.h>
#include<signal.h>

#define max_client 5  /* 5 - numero maximo de clientes */
#define tammens 80
#define serv "127.0.0.1"
#define PORT 2780

typedef struct      {
   int sockfd;
   struct sockaddr_in endserv;
  } Socket;
Socket *server;

struct client         {
   int sockfd;        /* socket id para o cliente */
   int client;        /* numero do cliente */
   pthread_t t;       /* thread id */
   pthread_attr_t attr;         /* thread atributos */
   struct sockaddr_in endclient;       /* ip, porta e protocolo */
  };

struct client listofclient[max_client];            /* lista de clientes conectados */
int current_client = 0;       /* numero de clientes conectados */

Socket *create_server_socket();
void start_server( Socket *s );
void send_receive( struct client *c );

void main (void) {
   printf ("Creating server socket...");
   server = (Socket*)create_server_socket( );
   if (server == NULL)      {
      printf("* Failed * \n");
      exit(0);
     }
   printf ("* Done *\n");
   start_server (server);
   while(1);
  }

Socket *create_server_socket( )     {
   Socket *s;
   s = (Socket *)malloc(sizeof(Socket));
```

```
  memset((void *)&s->endserv,0,sizeof(struct sockaddr_in)); /* Limpando a estrutura de socket 's'   */
  s->endserv.sin_port = htons (PORT);        /* Configurando a porta de comuinicação */
  s->endserv.sin_family = AF_INET;           /* Tipo da familia (TCP/IP) */
  s->endserv.sin_addr.s_addr = INADDR_ANY;      /* Conexão remota */

  /* Cria o socket - tipo(sock_stream)-tcp */
  if((s->sockfd = socket(s->endserv.sin_family,SOCK_STREAM,0)) == 0)      {
    perror ("* Socket Failed *");
    free(s);
    exit(0);
   }

  /* Associa o socket a uma porta definida (port) */
  if (bind(s->sockfd, (struct sockaddr *)&s->endserv,sizeof(struct sockaddr_in)) < 0)     {
    perror("* Erro no bind *");
    free(s);
    exit(0);
   }

  /* Escuta por novas conexões (max de 5 conexoes simultaneas (backlog)) */
  if (listen(s->sockfd,max_client) < 0) {
    perror("Erro no listen ");
    free(s);
    exit(0);
   }
  return s;
 }

void start_server(Socket *s)           {
  pthread_t t;
  pthread_attr_t attr;
  int sockfd,size;

  printf ("* Starting Server... *\n");
  printf ("* Done *\n");
  while (1)          {
    printf ("* Accept *\n");
    size = sizeof(struct sockadrr_in *);
    sockfd = accept(s->sockfd,(struct sockaddr*)&listofclient[current_client].endclient,&size);
    if (current_client > 5)
      close(sockfd);
    else
      {
       listofclient[current_client].sockfd = sockfd;
       listofclient[current_client].client = current_client;
       pthread_attr_init(&listofclient[current_client].attr);
       pthread_create(&listofclient[current_client].t,&listofclient[current_client].attr,(void *)send_receive,
                                  &listofclient[current_client]);
       sleep(2);
       current_client++;
      }
    }
 }

void send_receive (struct client *c) {
  int atual,nread,nwritten,nleft,tamanho;
```

```
  char *msg;
  msg = (char*)malloc(1024);

  while(1)          {
    bzero(msg,sizeof(msg));
    nread = 0;
    while (nread <= 0)
      nread = read (c->sockfd,msg,sizeof(msg));
    printf ("\n Servidor recebeu:%s ",msg);
    tamanho = strlen(msg);
    write (c->sockfd,msg,sizeof(msg));
   }
 }
```

4.2.2. Linguagem Java

A Linguagem Java também utiliza socket para se comunicar em um modelo Cliente/Servidor. Observe que em ambos os programas estamos importando a classe java.net (a maioria das capacidades de rede de Java pode ser encontrada nesse pacote) e pegando quaisquer erros de entrada/saída porque o código está cercado por um bloco try/catch.

Quanto ao código em si, as principais linhas são:

```
Socket s = new Socket(args[0], port);
DataInputStream sin = new DataInputStream(s.getInputStream());
```

A primeira linha abre um socket, que é uma abstração para o software da rede que permite a comunicação para fora e para dentro do programa. Passamos o endereço remoto e o número da porta para o construtor do socket. Quando o socket é aberto, o método getInputStream em java.net.Socket retorna um objeto InputStream, que você pode usar como qualquer outro arquivo.

Programa Cliente

```
import java.io.*;
import java.net.*;

public class client {
   public static final int DEFAULT_PORT = 6789;
   public static void usage( ) {
      System.out.println("Sintaxe: java Client <hostname> [<port>]");
      System.exit(0);
   }
   public static void main(String[ ] args) {
      int port = DEFAULT_PORT;

      if ((args.length != 1) && (args.length != 2)) usage( );
      if (args.length == 1) port = DEFAULT_PORT; // Especificacao da porta
      else {
         try { port = Integer.parseInt(args[1] ); }
         catch (NumberFormatException e) { usage( ); }
           }
```

```
        try {
            // Cria um socket para se comunicar com um host e uma porta especificada
            Socket s = new Socket(args[0], port);

            // Cria streams (fluxo de dados) para leitura e gravacao de linhas de texto para/de um socket.
            DataInputStream sin = new DataInputStream(s.getInputStream( ));
            PrintStream sout = new PrintStream(s.getOutputStream( ));

            // Cria um stream para leitura de linhas de texto do console
            DataInputStream in = new DataInputStream(System.in);

            // Diz ao usuario que estamos conectados
            System.out.println ("Connected to " + s.getInetAddress( ) + ":"+ s.getPort());

            String line;
            while(true) {

                // Imprime um prompt
                System.out.print ("Cliente:\> ");

                System.out.flush( );
                // Le uma linha do console; checa por EOF
                line = in.readLine( );

                if (line == null) break;
                // Envia a linha para o servidor
                sout.println(line);

                // Le uma linha do servidor
                line = sin.readLine( );

                // Checa se a conexao esta fechada
                if (line == null) {
                    System.out.println ("Conexao encerrada pelo servidor.");
                    break;
                }

               // Exibe a linha no console
               System.out.println ("Servidor: "+line);
            }
        }
        catch (IOException e) { System.err.println(e); }

        // Fechar o socket
        finally {
            try { if (s != null) s.close( ); } catch (IOException e2) { ; }
        }
    }
}
```

Programa Servidor

```
import java.io.*;
import java.net.*;

public class server extends Thread {
   public final static int DEFAULT_PORT = 6789;
   protected int port;
   protected ServerSocket listen_socket;

   // Sai com uma mensagem de erro, quando uma excecao ocorre
   public static void fail(Exception e, String msg) {
      System.err.println(msg + ": " +  e);
      System.exit(1);
   }

   // Cria um ServerSocket para escutar por conexoes;  incializa o thread
   public server(int port) {
      if (port == 0) port = DEFAULT_PORT;
      this.port = port;
      try { listen_socket = new ServerSocket(port); }
      catch (IOException e) { fail(e, "Excecao ao criar o socket do servidor"); }
      System.out.println("Servidor: Escutando a porta " + port);
      this.start( );
   }

   // O corpo do servidor thread.  Loop infinito, escutando por  e aceitando conexoes de clientes
  // Para cada conexao, cria um objeto de conexao para manter a comunicacao atraves do new socket
   public void run( ) {
      try {
         while(true) {
            Socket client_socket = listen_socket.accept( );
            Connection c = new Connection(client_socket);
         }
      }
      catch (IOException e) {
         fail(e, "Excecao enquanto escuta por conexoes");
      }
   }

   // Inicia o servidor, escutando a porta especificada
   public static void main(String[ ] args) {
      int port = 0;
      if (args.length == 1) {
         try { port = Integer.parseInt(args[0]);  }
         catch (NumberFormatException e) { port = 0; }
      }
      new server(port);
   }
}

// Esta classe é um thread que mantem toda a comunicacao com o cliente
class Connection extends Thread {
   protected Socket client;
   protected DataInputStream in;
   protected PrintStream out;
```

```
   // Inicializa as streams e ativa o thread
   public Connection(Socket client_socket) {
      client = client_socket;
      try {
         in = new DataInputStream(client.getInputStream());
         out = new PrintStream(client.getOutputStream());
      }
      catch (IOException e) {
         try { client.close(); } catch (IOException e2) { ; }
         System.err.println ("Excecao enquanto recebe socket streams: " + e);
         return;
      }
      this.start( );
   }

   // Prove o servico.
   // Le uma linha, e a envia para o servidor
   public void run( ) {
      String line;
      int len;
      try {
         for( ; ;) {

            // Le uma linha
            line = in.readLine( );
            if (line == null) break;

            // Exibe a linha que recebeu
            System.out.println ("Servidor recebeu: " +line);

           // Envia a linha para o cliente
           out.println (line);
         }
      }
      catch (IOException e) { ; }
      finally { try {client.close( );} catch (IOException e2) {;} }
   }
}
```

4.2.3. Linguagem LabVIEW

Na Linguagem LabVIEW os programas Cliente e Servidor não utilizam socket. O Cliente troca informações normalmente com o Servidor. O programa Servidor está dividido em duas etapas:

- a primeira é que a variável Listen deve ser incializada em falso, quer dizer, que ela não tem nenhum problema. Caso ocorra algum erro ela irá mudar para verdadeiro;
- a segunda etapa é o estabelecimento da conexão e o fornecimento do serviço. Esta segunda etapa é criada uma fila de clientes com o número de identificação da conexão, para que o servidor qual é a mensagem respectiva a cada cliente.

O LabVIEW 5.0 foi utilizado para desenvolver este programa, e nesta versão não está disponível a função socket.

Programa Cliente

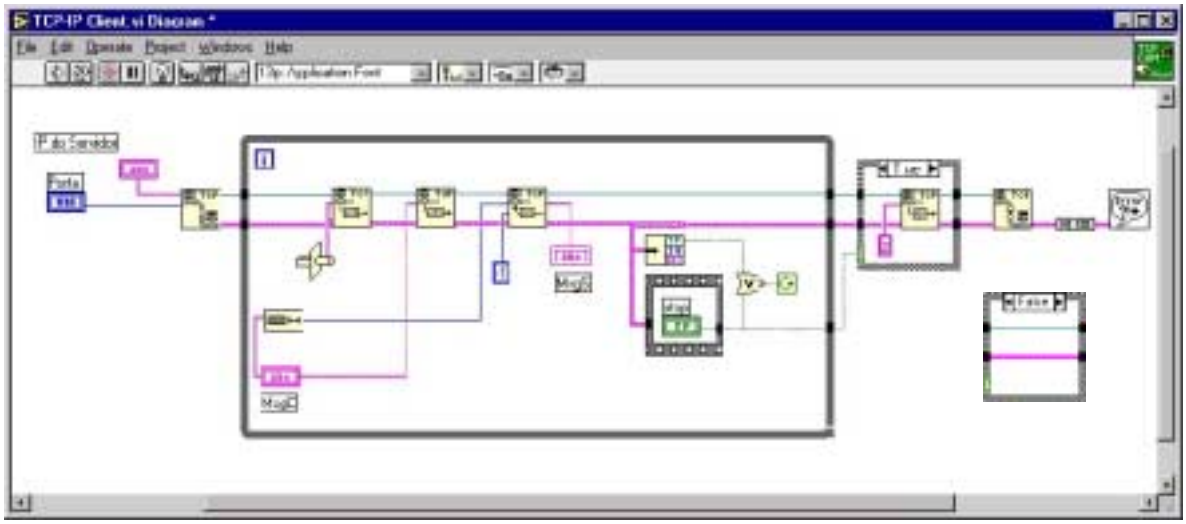

Figura 30 – Programa cliente – TCP-IP Client.vi Diagram

Programa Servidor

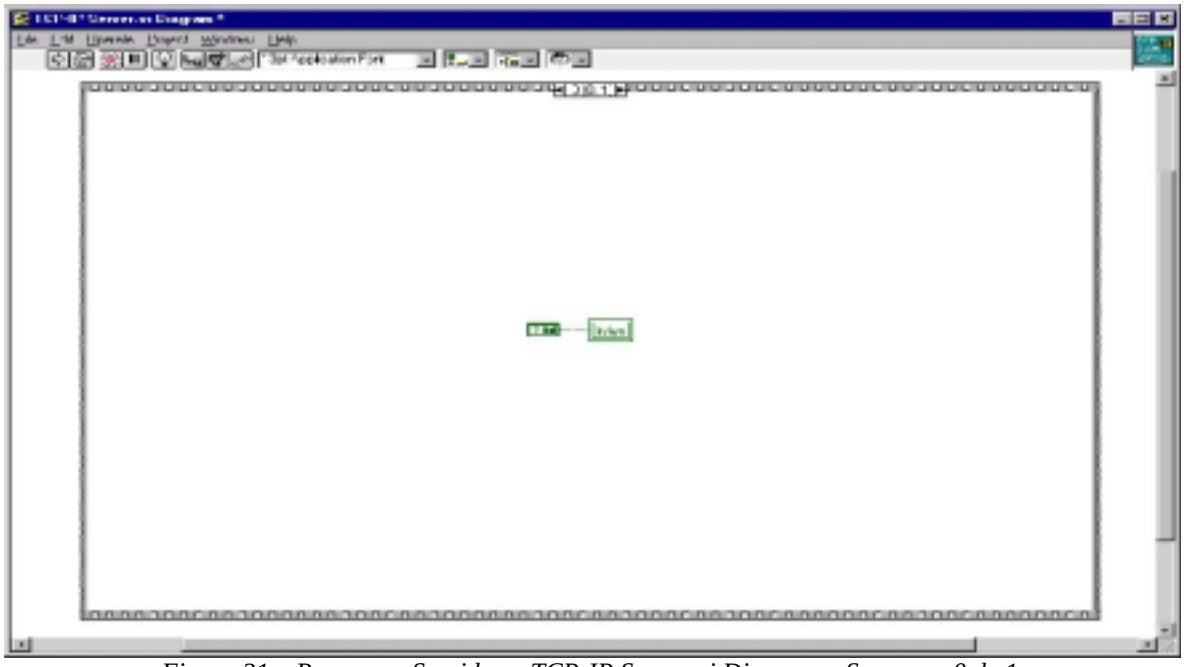

Figura 31 – Programa Servidor – TCP-IP Server.vi Diagram – Sequence 0 de 1

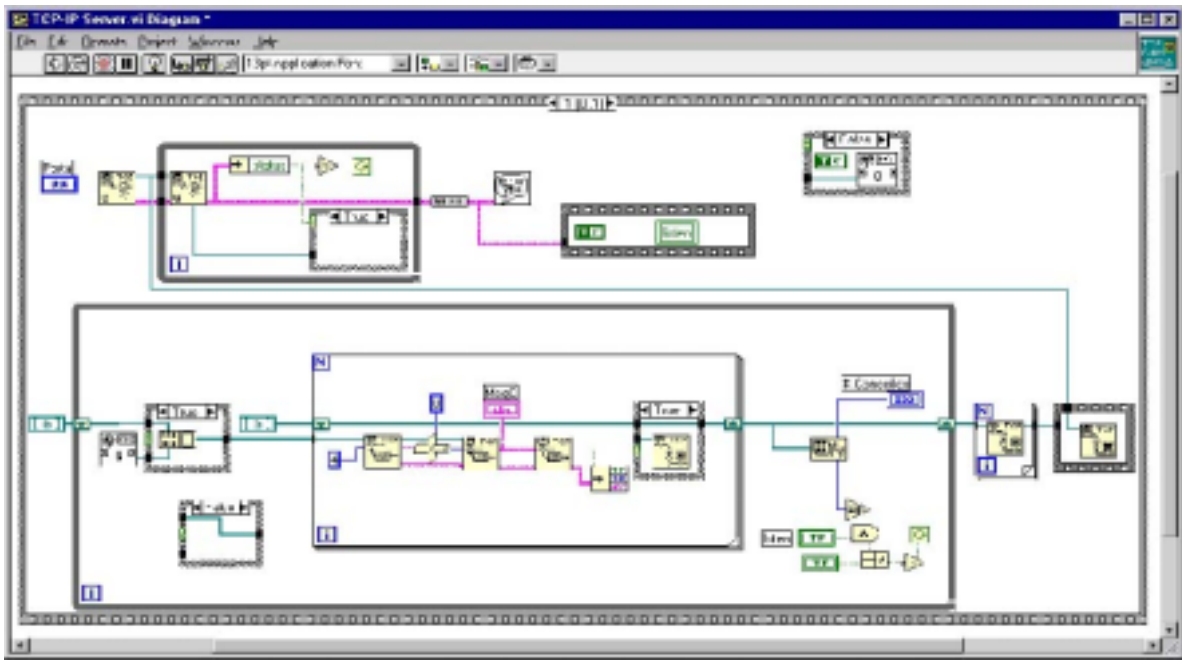

Figura 32 - Programa Servidor – TCP-IP Server.vi Diagram – Sequence 1 de 1

Referências:

[1] – “G Programming Referece Manual”, National Instruments Corporation, January 1998 Edition, Part Number 321296B-01.

[2] – Online Reference, LabVIEW™ 5.1, National Instruments Corporation, February 1999, Part Number 321782B-01.