PSI -2222 - Práticas de Eletricidade e Eletrônica

Instrumentação Virtual

João E. Kögler Jr
www.lsi.usp.br / ~kogler
kogler@ lsi.usp.br

outubro 2004

INSTRUMENTAÇÃO VIRTUAL

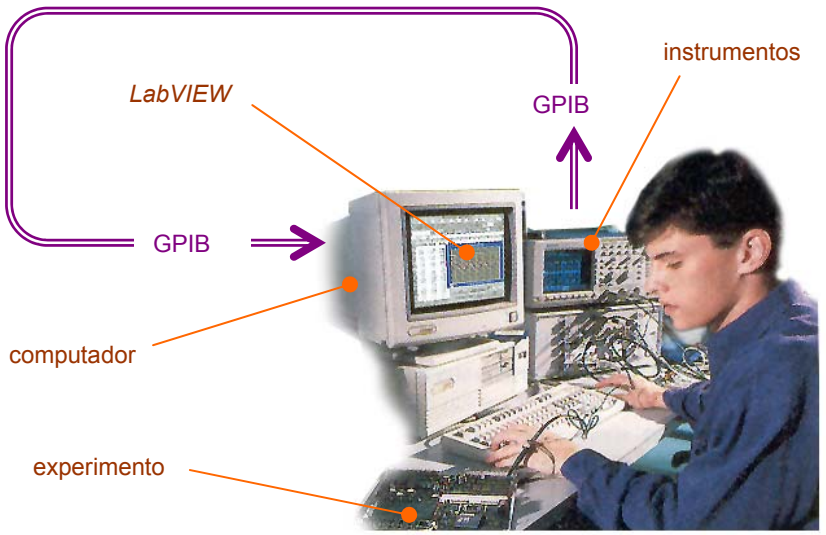
1. Objetivos

- Apresentar o conceito de instrumentação virtual, seus aspectos e suas vantagens
- Associar a instrumentação virtual à programação visual (uso de componentes de *software* reutilizáveis, implementados graficamente) orientada ao fluxo de dados
- Apresentar as principais interfaces e padrões de comunicação usados na instrumentação virtual

2. Introdução

Você descobrirá, no decorrer do seu curso de engenharia elétrica que, nas disciplinas de Laboratório de Eletricidade e de Eletrônica, são realizadas diversas experiências em que os equipamentos usados para geração e medida de sinais são conectados ao computador através de cabos GPIB, controlando-se o experimento todo via *software* através de um sistema denominado LabVIEW ®[1].

No parágrafo acima, você deve interpretar a palavra *experimento* como sendo uma seqüência de ações de medida de valores instantâneos e de parâmetros de sinais elétricos. Esses valores são fornecidos pelos instrumentos de medida. Além das ações de medida, também faz parte do experimento o registro e a documentação das medições.



Essa abordagem adotada nesses cursos visa prepará-lo para que você possa sentir-se à vontade com uma das mais modernas e revolucionárias formas de se realizar projetos, testes e simulações em engenharia: a denominada engenharia virtual. Esse paradigma possui duas grandes componentes: a instrumentação virtual e a simulação por computador. A primeira se refere ao emprego do computador para ampliar a capacidade funcional dos instrumentos. A segunda corresponde ao uso do computador para simular o comportamento de processos, sistemas, dispositivos, meios e materiais. A instrumentação virtual e a simulação por computador nasceram e evoluíram independentemente uma da outra. Nos últimos anos, elas convergiram para esse novo conceito de engenharia virtual. Agora, pode-se falar na instrumentação virtual enviando dados reais em tempo real para um modelo simulado, que processa esses dados e retorna ao sistema físico, através de interfaces adequadas, estímulos, sinais e efeitos.

[1] *LabVIEW* é marca registrada da National Instruments, Texas, Estados Unidos da América.

A tecnologia de instrumentação utilizada nesses cursos procura automatizar os experimentos, permitindo que os fenômenos que se deseja estudar possam ser explorados intensivamente. Isso requer que se realize as seguintes atividades, de forma automática:

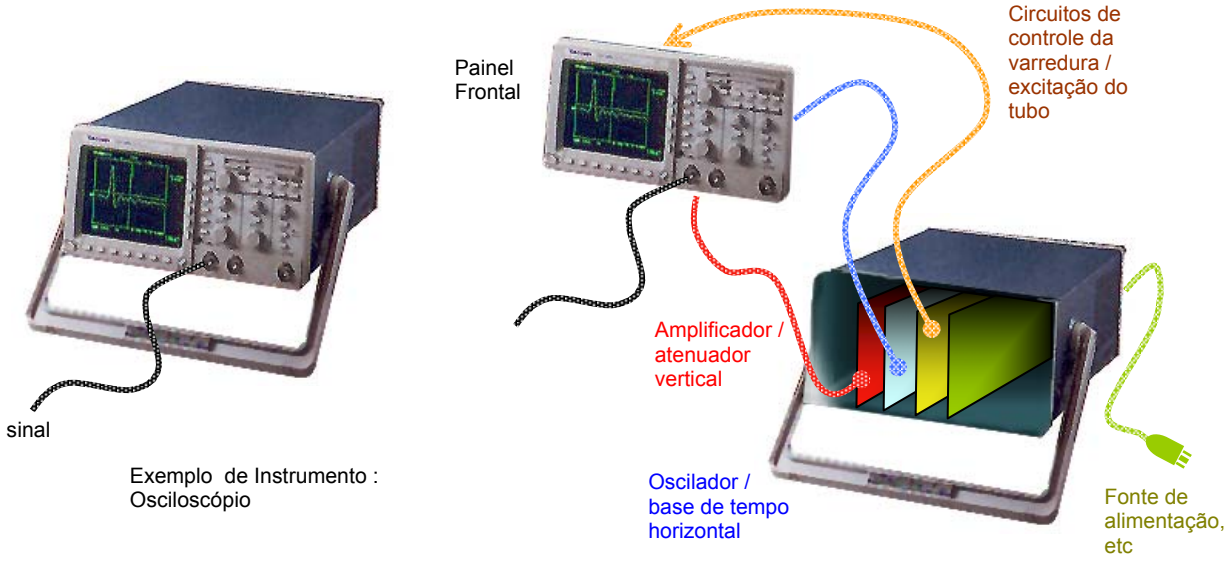
- Monitorar diversas variáveis simultaneamente
- Coletar grande volume de dados de cada uma delas
- Repetir o experimento com a variação de parâmetros e /ou situações experimentais
- Colecionar os resultados e visualizá-los de forma conclusiva e sinóptica

Essa automatização é alcançada através do emprego de geradores de sinal, osciloscópios e demais instrumentos capazes de serem controlados e monitorados via computador. O programa de computador é o responsável pela condução do experimento. Ele realiza as seguintes ações:

I. pré-condiciona todos os equipamentos,
II. interpreta os comandos do experimentador,
III. dispara as excitações e aquisições de sinais,
IV. administra a seqüência de eventos,
V. promove a coordenação e sincronização dos processos de acionamento e monitoração e, finalmente,
VI. encarrega-se do armazenamento e apresentação dos resultados em uma forma que proporcione boa visualização para o experimentador.

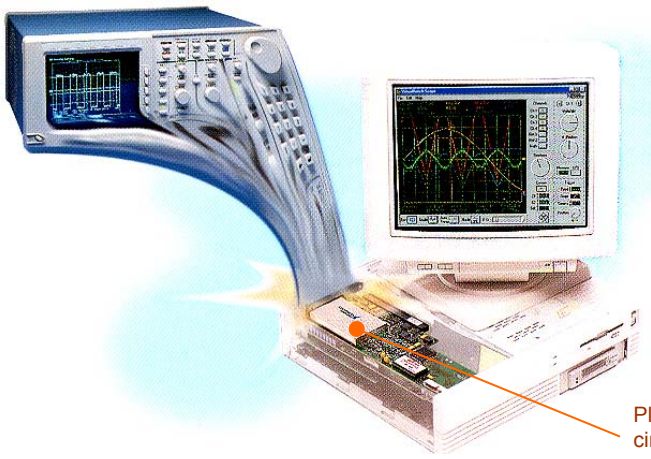
O *programa de computador* é, portanto, um elemento-chave da automatização. É o responsável por administrar toda a complexidade dessa tarefa e realizar os cálculos numéricos. Outro elemento-chave é a *monitoração e controle do instrumento pelo computador*, que devem ser feitos via comunicação digital de dados, que requer o uso de interfaceamento e transmissão de sinais digitais. Esses ingredientes compõem o paradigma de engenharia auxiliada por computador (CAE - *Computer-Aided Engineering*) denominado ***Instrumentação Virtual***, acima mencionado e que será explorado e estudado neste seminário.

A idéia é basicamente a seguinte: um equipamento elétrico / eletrônico destinado a medir ou controlar algum tipo de variável, essencialmente tem um painel frontal, por onde o usuário opera o equipamento e um conjunto de circuitos que realizam as operações necessárias. Considere por exemplo o caso de um osciloscópio: ele contém circuitos para receber o sinal medido, condicionando-o adequadamente. Esse condicionamento corresponde ao acoplamento e à atenuação / pré-amplificação do sinal. O sinal assim condicionado é injetado no amplificador vertical para gerar a deflexão vertical do feixe na tela. Outros circuitos existentes correspondem à geração de varredura, de acordo com uma base de tempo selecionada e disparando segundo informações de sincronização e disparo retiradas eventualmente do sinal de entrada.



Exemplo de Instrumento :
Osciloscópio

Oras, o monitor do computador também tem um tubo de raios catódicos e circuitos que controlam a varredura do vídeo. A idéia da instrumentação virtual surgiu, então, pensando-se em colocar em uma placa ligada ao barramento do computador os circuitos necessários para receber o sinal, condicioná-lo e convertê-lo em seguida à forma digital. A forma de onda será então exibida na tela, sendo a base de tempo produzida por software. Com o advento das interfaces gráficas de usuário, tornou-se possível sintetizar na tela o próprio painel do osciloscópio, tal como se o computador emulasse o instrumento. Daí o termo instrumento virtual.



O computador como instrumento virtual, no caso, emulando um osciloscópio.

Placa de aquisição de dados com os circuitos de acoplamento e amplificação

Diversos outros instrumentos podem ser emulados no computador dotado de uma placa adequada ao tratamento do sinal específico. Também sensores e transdutores podem ser conectados ao computador, através de placas de condicionamento de sinais e de aquisição de sinais, como ilustrado abaixo.

notebook

O computador notebook é dotado de uma placa tipo PCMCIA que realiza a aquisição do sinal condicionado no módulo indicado.

Bloco de terminais e condicionamento de sinal.

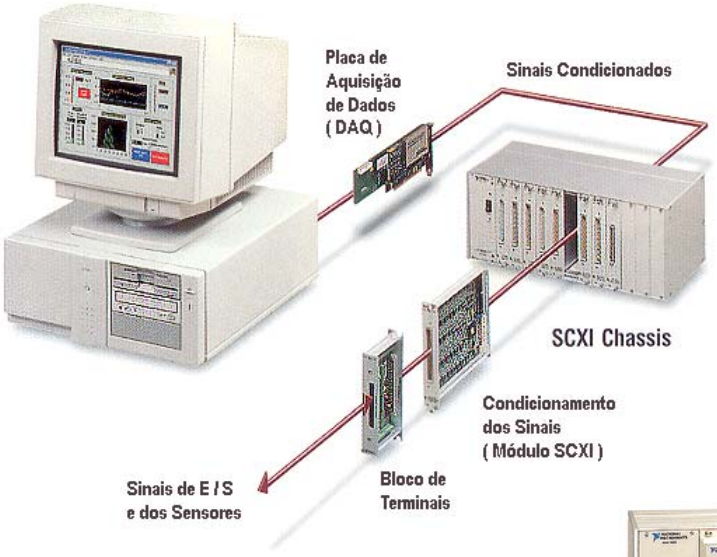
Sensor termopar

Podem-se construir bastidores para acomodarem diversos blocos de terminais e circuitos de condicionamento de sinais provenientes de sensores, como ilustrado a seguir. A figura mostra um bastidor tipo SCXI, da National Instruments. Nos exemplos ilustrados pelas figuras até aqui, os sinais chegam diretamente ao computador, provenientes de um sensor ou sonda (ponta de prova). Uma vez na memória, os sinais podem ser processados, pois passam a ser dados numéricos que podem ser manipulados à vontade, bastando-se que se aplique os algoritmos adequados.

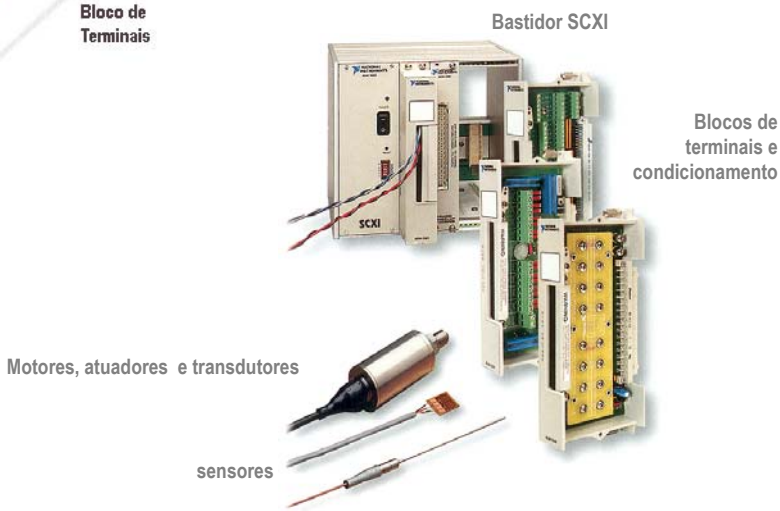
.

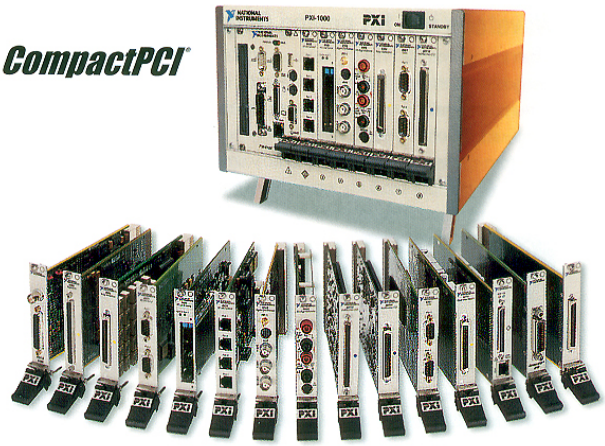
Esse expediente de adquirir os sinais de armazená-los de forma numérica para posterior processamento e análise, também inspirou os projetistas de instrumentos que criaram equipamentos que já fornecem o sinal em forma digital e transferindo-os ao computador.

A vantagem de se fazer isso encontra-se na possibilidade de se estender a funcionalidade do instrumento. Assim, por exemplo, o osciloscópio pode ser transformado em algum tipo de analisador de sinais, por exemplo, realizar sua análise espectral através da transformada de Fourier, calculada no computador. O resultado poderia ser enviado de volta ao osciloscópio ou exibido na tela do computador. Para realizar essas transferências entre instrumento e computador, criou-se um protocolo e um barramento padronizados, constituindo o padrão GPIB, que mencionamos atrás e que estudaremos em maior detalhe adiante.

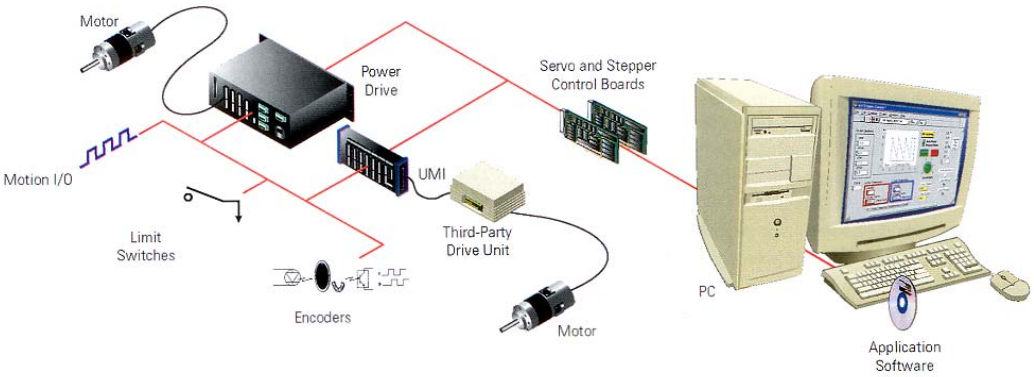


Os sinais de dispositivos de entrada e saída e de sensores e atuadores podem ser transferidos computador.





Surgiram também computadores dotados de padrões elétricos de maior confiabilidade, com maior robustez. Exemplo dessa tecnologia são os computadores compactos de barramento PCI, chamados de PXI (Compact PCI). A figura ilustra uma série de placas que podem ser acopladas ao barramento PXI .



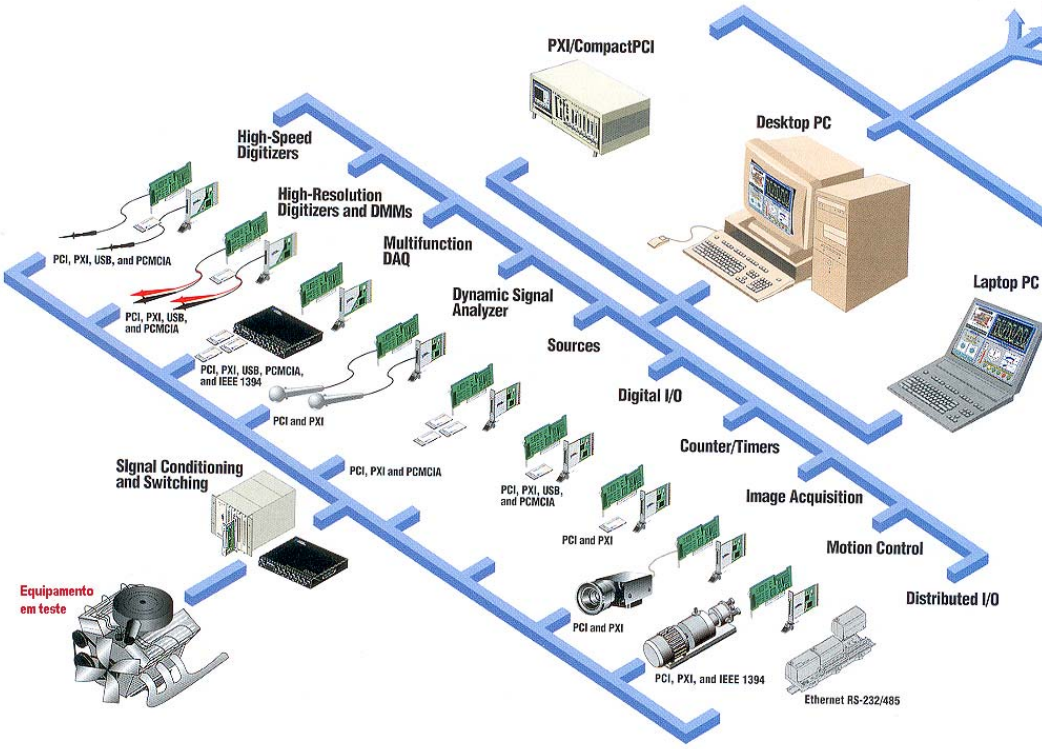
Outro exemplo de instrumentação virtual, acima ilustrado, corresponde ao uso de interfaces de acionamento de motores de passo ou servo.

Outros dispositivos, além de computadores no estrito senso da palavra, podem ser utilizados como instrumentos virtuais, como os PDAs (personal device assistant) como o Palm e o Pocket PC da HP / Compaq.

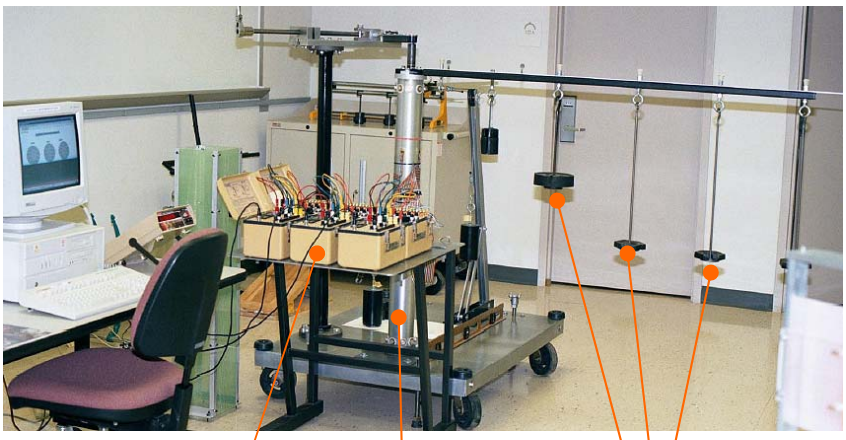
Placa PCMCIA adaptada ao Pocket PC

A vantagem de se utilizar o Pocket PC como instrumento virtual é o fato de ele emular um equipamento portátil que pode ser programado para realizar uma variedade de análises nos dados.

A instrumentação virtual tornou-se uma ferramenta poderosa para a realização de testes e controle de qualidade na indústria. A figura a seguir mostra uma variedade de testes que podem ser realizados em um equipamento e que podem, eventualmente, ser concentrados em um único computador.

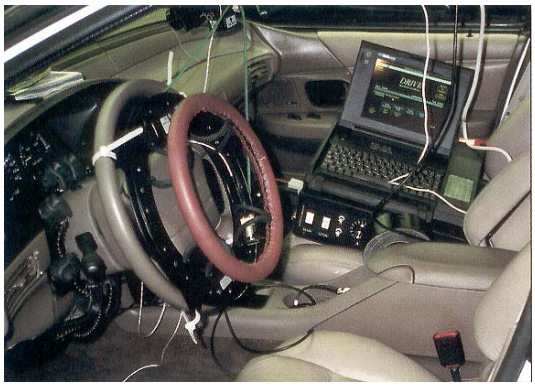


A figura a seguir mostra um ensaio de resistência mecânica de materiais metálicos, sendo realizado através de controle e monitoração via instrumentação virtual.



Corpo de prova (detalhe), mostrando os sensores (strain gages)

A figura ao lado mostra um teste empregando instrumentação virtual, realizado na indústria automobilística.



3. Instrumentação Virtual

Um instrumento real genérico pode ser visto como um aparelho dotado dos seguintes componentes:

- um elemento sensor ou atuador
- um transdutor
- um painel de controle e medição
- um painel de conexões

Outros componentes eventualmente fazem parte do instrumento (por exemplo, circuitos eletrônicos, etc) porém os elementos acima apresentados são suficientes para construir um modelo genérico de instrumento, satisfatório para a discussão a seguir.

O sensor é o componente que aparece em um instrumento de medida, que o habilita a "sentir" uma dada grandeza física (uma tensão, uma temperatura, uma pressão, etc).

O atuador aparece no caso de um instrumento de controle ou comando (por exemplo, um robô, um posicionador pneumático, um controlador lógico, um elemento aquecedor, uma fonte de tensão, etc). O atuador permite alterar uma variável física (a temperatura, a posição de um objeto, a tensão na entrada de um circuito, o estado de uma chave, etc).

O transdutor é um elemento que converte uma grandeza física de uma dada natureza em uma de outro tipo. Por exemplo, um microfone transforma som em um sinal elétrico, enquanto um alto-falante faz o oposto - são ambos transdutores eletromecânicos.

O painel de controle e medição (geralmente frontal - *front panel*) contém botões, chaves, indicadores e mostradores que permitem operar o instrumento.

O painel de conexões (geralmente traseiro - *back panel*) contém os terminais aos quais se conecta os elementos sensores ou atuadores, por exemplo, através de cabos ou fios.

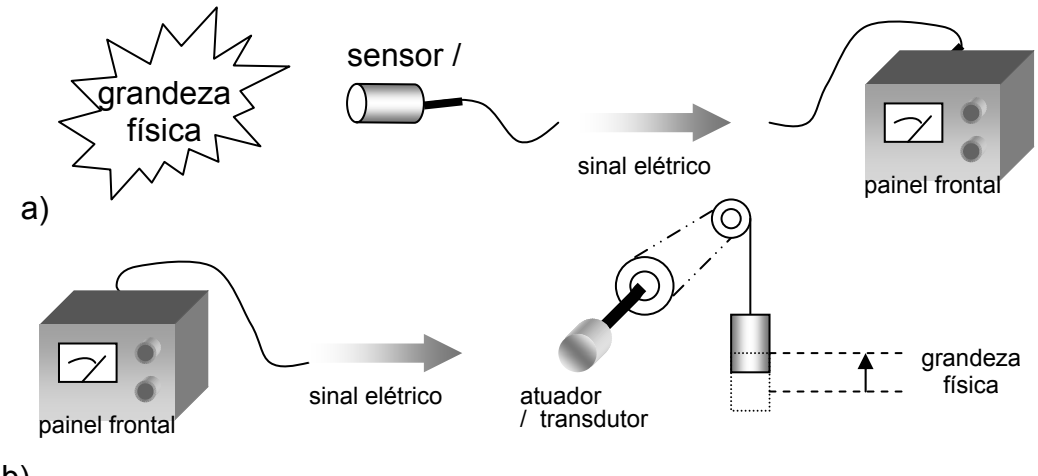


Figura 1 - Modelo de instrumentos de (a) medida e (b) comando.

A transdução da grandeza física genérica para a forma elétrica é requerida em instrumentos cujo funcionamento é eletrônico (atualmente, a maioria). Em geral o painel de controle opera eletronicamente, bem como o processamento do sinal. A figura 1 mostra esquematicamente os dois tipos básicos de instrumento: o de *medida* e o de *controle*. O operador humano usa esses instrumentos fazendo as conexões dos cabos de ligação com os demais dispositivos através do *painel de conexões* e opera o instrumento através do *painel frontal*. Dizemos então que o conjunto desses painéis constitui a *interface conceitual* entre o instrumento (real) e o usuário.

O *instrumento virtual* é um sistema formado por um computador mais um instrumento de medida ou equipamento de comando (reais), colocados em comunicação. Um programa executando no computador torna o instrumento ou o controlador acessível ao operador através de uma interface gráfica de software. Essa interface é dotada de botões, chaves, mostradores, indicadores, painéis de exibição de gráficos, etc, apresentados como objetos interativos, animados sob ação do operador através do apontador do *mouse*. O instrumento conectado ao computador pode ser desde um equipamento completo, como um osciloscópio, frequencímetro etc, ou mesmo um simples sensor como um termopar, um extensômetro etc.

Os botões e indicadores que aparecem na tela do instrumento virtual podem não corresponder a controles reais do instrumento que está conectado ao computador. Isto é, usando o computador, podemos ampliar a funcionalidade de um instrumento acrescentando-lhe novas funções executadas pelo computador com as medidas fornecidas pelo instrumento. Por exemplo, se acoplarmos ao computador um osciloscópio digital que não tenha a função de análise espectral, podemos obter os dados do sinal adquiridos pelo osciloscópio, processá-los no computador usando-se um algorítmo de FFT (Transformada Rápida de Fourier) e assim realizar uma análise espectral do sinal. O conjunto osciloscópio digital + computador executando a FFT, forma um Analisador de Espectro Virtual.

A interface conceitual entre um computador e seu usuário é provida pelos seus *dispositivos de entrada/saída*: *a tela do monitor, o teclado, o apontador* (*mouse, trackball, tablets, data gloves*) e tantos outros dispositivos que vão surgindo à medida que se desenvolve novas interfaces humano-máquina. Além do interfaceamento com o usuário, deve-se considerar aquele realizado com outros computadores e equipamentos periféricos. Através da *interface de rede*, o computador pode comunicar-se com outros computadores e, dessa forma, com usuários dos mesmos, situados remotamente. Outros equipamentos periféricos são ligados ao computador através de *adaptadores* adequados (porta serial, porta paralela e placas dedicadas, conectadas ao barramento). Esses equipamentos são, por exemplo, câmeras de vídeo, microfones, alto-falantes, instrumentos de medida e acionamento etc.

No paradigma de instrumentação virtual, o computador é usado tanto para operar do instrumento, quanto para conduzir o experimento, conforme for conveniente. Nessa visão, a interface conceitual associada ao computador deve prover a mesma funcionalidade que a interface conceitual que um instrumento real apresentaria ao usuário. Ou seja, através da tela do computador, de seu teclado e apontador, o usuário deve ser capaz de operar os instrumentos ou conduzir o experimento, tal qual faria usando os controles do instrumento real. Evidentemente que essa funcionalidade será limitada pelas características do software de instrumentação virtual. A seguir, discutiremos essas características e faremos uma comparação entre alguns tipos de implementações dessa idéia.

Para se operar um instrumento via computador, é necessário que o instrumento disponha de uma interface eletrônica de comnicação que possa ser adaptada ao computador. Existem diversas formas de se fazer isso, como será visto na seção 6, mais adiante. Por ora, vamos admitir que os instrumentos a que nos referirmos permitem essa conexão com o computador. A concepção mais elementar de se implementar a operação do instrumento via computador é construir um programa em alguma linguagem declarativa (como C, Java, Pascal, BASIC) e executá-lo, capturando via teclado as entradas do usuário e apresentando os resultados na tela do monitor em forma numérica ou de um gráfico. Essa foi a abordagem tradicional durante vários anos e, ainda atualmente é largamente utilizada. Todavia, esse método requer o conhecimento de programação de computadores e de transmissão digital de dados.

Com o advento dos ambientes gráficos de programação e visando simplificar a tarefa de quem desenvolve aplicações que manipulam dados de instrumentos, surgiram as chamadas linguagens de *programação visual*. Uma dessas linguagens é usada pelo *LabVIEW*, que estudaremos nesta experiência. Ela é denominada "G" [2] e tem a mesma potencialidade de uma linguagem textual, como C ou Pascal, por exemplo. Entretanto, seus comandos são apresentados de forma gráfica, como ícones interconectados através de ligações, formando o programa. O programa escrito em G constitui a base do instrumento virtual do LabVIEW.

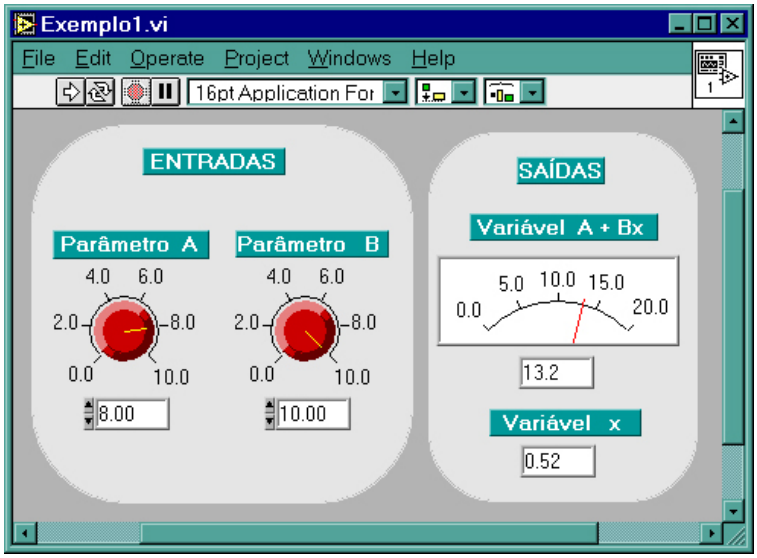
[2] O G vem de "**G**raphics", imitando o nome da linguagem C, porém com um apresentação totalmente gráfica.

O instrumento virtual é composto de duas partes:

- Um painel frontal
- Um diagrama de blocos

O painel frontal é uma janela apresentada na tela do monitor, na qual são desenhados ícones com formatos que lembram os componentes de um painel de instrumento (botões, chaves, indicadores, oscilogramas, etc). Esses componentes estão associados a variáveis e parâmetros, cujos valores são medidos ou ajustados. O diagrama de blocos[3] representa graficamente os processos aos quais são submetidas as variáveis e parâmetros apresentados no painel frontal.

a)



b)

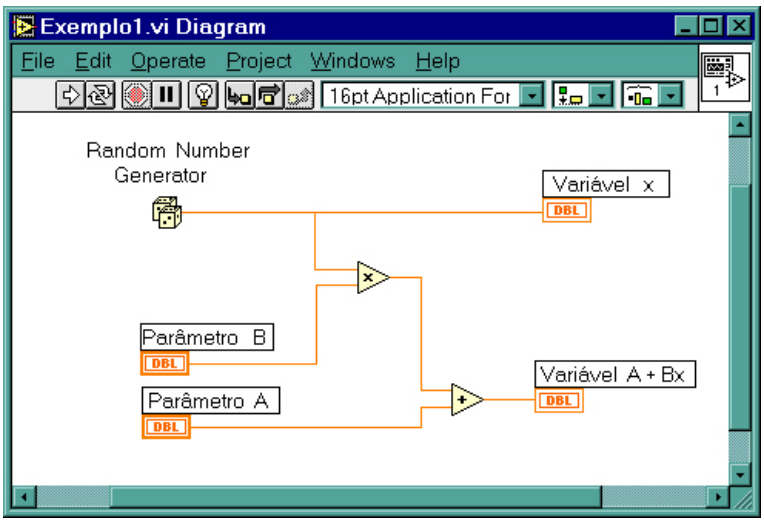


Figura 2 - Interface conceitual do instrumento virtual (LabVIEW) : (a) painel frontal e (b) diagrama de blocos.

A figura 2 mostra uma situação hipotética em que uma variável **x** medida é combinada com os parâmetros **A** e **B** ajustados no painel frontal, produzindo o valor **A + Bx**. No diagrama de blocos, mostra-se que a medida da variável **x** é simulada por um gerador de números aleatórios, que produz números ao acaso, uniformemente distribuídos no intervalo

[3] Realmente denominado "diagrama de fiação", como tradução de *wiring diagram*. Entretanto, para maior clareza didática foi aqui chamado "diagrama de blocos".

[0,1]. A variável e os parâmetros são processados pelos blocos de multiplicação e adição mostrados no diagrama de blocos. Como saída, são fornecidos simultaneamente os valores de **x** e de **A + Bx**.

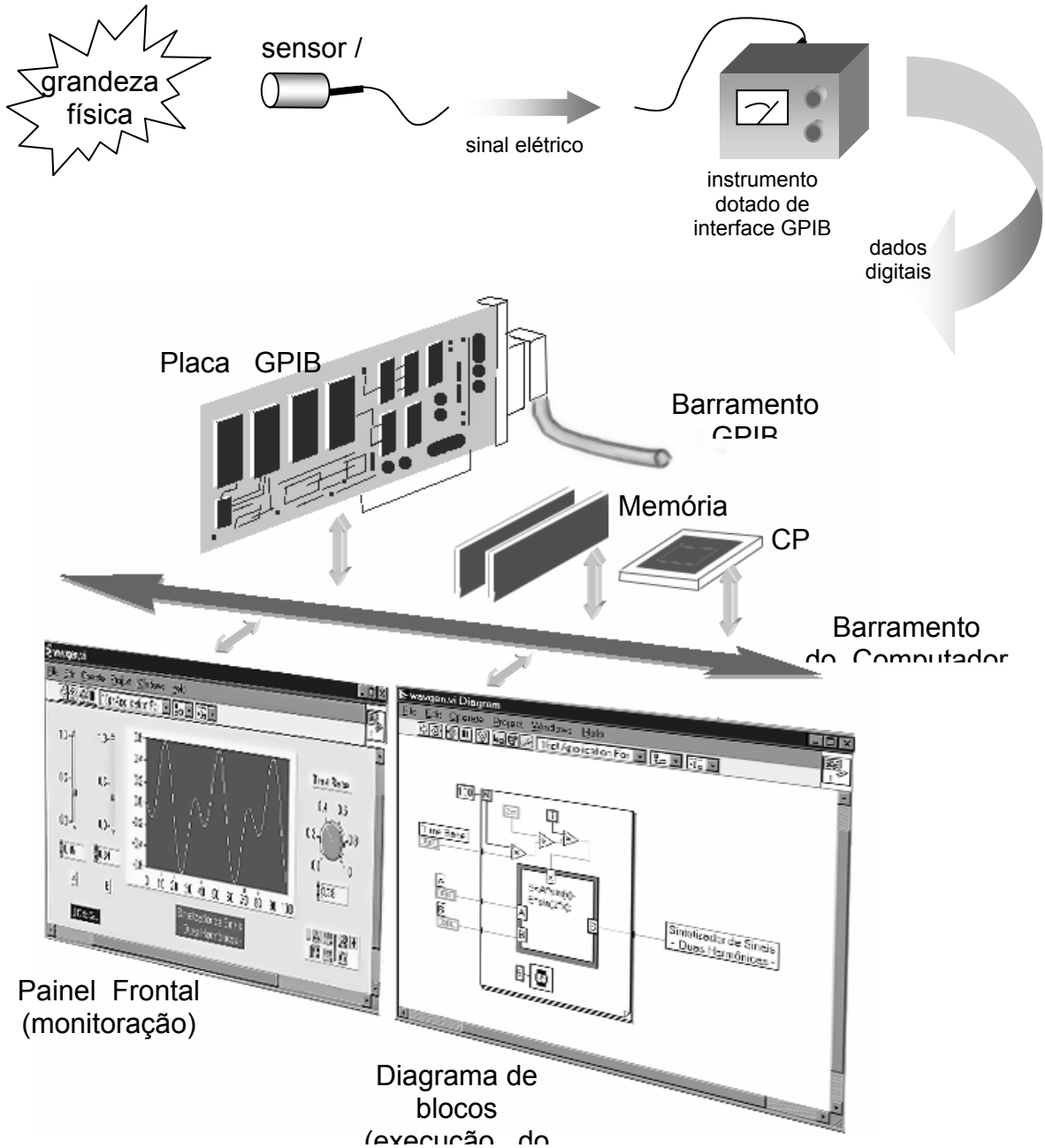


Figura 3 - Estrutura de um instrumento virtual

A disponibilidade de interfaces gráficas permite a apresentação da informação em uma forma mais ergonômica, isto é, segundo um arranjo visual adequado à tarefa que se pretende realizar. Assim sendo, em vez de usar somente o teclado, a captura das entradas do usuário (comandos, ações, ajuste de parâmetros) pode ser feita de modo mais natural, atuando-se com o apontador do *mouse* em um ícone mnemônico (por exemplo, desenhado com o aspecto de um botão ou chave). Além disso, podem-se representar as conexões, entre os diversos equipamentos que compõem o experimento, através do diagrama de blocos. Com esses ingredientes, a interface conceitual do computador pode prover o

instrumento virtual com os mesmos compoentes funcionais encontrados nos painéis frontais e painéis de conexões dos instrumentos.

A figura 3 mostra a estrutura de um instrumento virtual, tal como implementado no *LabVIEW*. Sua interface conceitual é composta dos elementos que anteriormente mencionamos: (i) o painel frontal e (ii) o diagrama de blocos. O processo de monitoração ou controle dá-se fisicamente no instrumento real, tal como mostrado na figura 1. Só que agora, o instrumento real é conectado ao computador via um enlace de comunicação digital (no nosso caso será GPIB) e a operação do instrumento real se faz através da interface gráfica do computador com o usuário. O instrumento virtual pode fornecer ao usuário um conjunto de valores melhor caracterizado do que o que seria fornecido pelo instrumento real que de fato os mediu (por exemplo, poderá fornecer esses dados filtrados de ruído, organizados em uma escala mais adequada, agrupados em classes, etc). Além disso pode valer-se de recursos de computação gráfica para produzir uma visualização dos dados mais compreensível.

4. Programação Visual com Componentes - *Componentware*

Um programa de computador é um conjunto de declarações e instruções expressos em uma linguagem adequada, que descreve que tarefas devem ser executadas pela máquina. Usualmente, a linguagem usada é *textual* e o programa é, portanto, um texto contendo as tais declarações e instruções. Para se alcançar uma produtividade e eficiência elevadas na atividade de programação, foram desenvolvidos diversos métodos de se programar. Estes recomendam formas de se conceber e estruturar o *código fonte* do programa, aquele texto que o programador escreve em linguagem de alto nível, antes de ser compilado. Com o advento das interfaces gráficas, passou-se a incorporar nos programas trechos que descrevem os componentes visuais das mesmas: janelas, botões, cursores, etc. Como esses componentes em geral são padronizados para cada ambiente gráfico, eles passaram a ser fornecidos ao programador como bibliotecas pré-fabricadas, que podem ser ligadas ao programa durante a compilação (estáticas) ou durante a execução do programa, à medida em que são requisitadas (dinâmicas). As bibliotecas dinâmicas (*DLLs - Dynamic Linked Libraries*) são cada vez mais utilizadas pois só são incorporadas no instante de sua utilização, levando à economia de espaço de armazenagem.

O chamado *ambiente gráfico* é uma convenção usada para apresentar os elementos que compõem a interface gráfica. Essa convenção regulamenta o aspecto das janelas, botões, menus, etc, sua organização e disposição, seu acabamento visual. Exemplos de ambientes gráficos: *openlook, motif, athena Xtoolkit, microsoft Windows.*

A idéia de *componente de software* deve ser entendida como sendo um trecho de programa que tem uma funcionalidade bem específica. Certos componentes são utilizados muitas vezes em um mesmo programa ou em diferentes programas. Por exemplo, o trecho de programa que implementa um botão ou um menu em uma interface gráfica. Por uma questão de eficiência, os componentes têm seu código bastante otimizado e são fornecidos pré-compilados, em bibliotecas, como mencionamos antes. No caso da instrumentação virtual, pode-se pressupor uma série de componentes que vão traduzir a funcionalidade dos componentes correspondentes em instrumentos reais. Assim, encontraremos os seguintes componentes em um instrumento virtual: botões, mostradores, indicadores luminosos ("LEDs"), janelas oscilográficas (como a tela de um osciloscópio), entre outros.

Em uma linguagem de programação visual, o programa é expresso através de componentes visuais interconectados de modo a compor a lógica das tarefas a serem executadas. Os componentes representam não só os elementos da interface gráfica, mas também os blocos construtivos de uma linguagem de programação: blocos iterativos (*for, while*) , estruturas de decisão (*case, if-then-else* [4]), sub-rotinas e procedimentos, estruturas de dados (variáveis elementares, cadeias de caracteres, matrizes, estruturas híbridas), etc.

[4] A estrutura *if-then-else* é implementada como caso particular de *case* no *LabVIEW*.

Os exemplos a seguir mostram três implementações de um programa para gerar um conjunto aleatório de dados e encontrar o valor máximo desse conjunto. A primeira implementação é feita em linguagem C, textual. A segunda e a terceira são feitas usando-se programação visual, com dois produtos comerciais: *HP-VEE* [5] e *LabVIEW*.

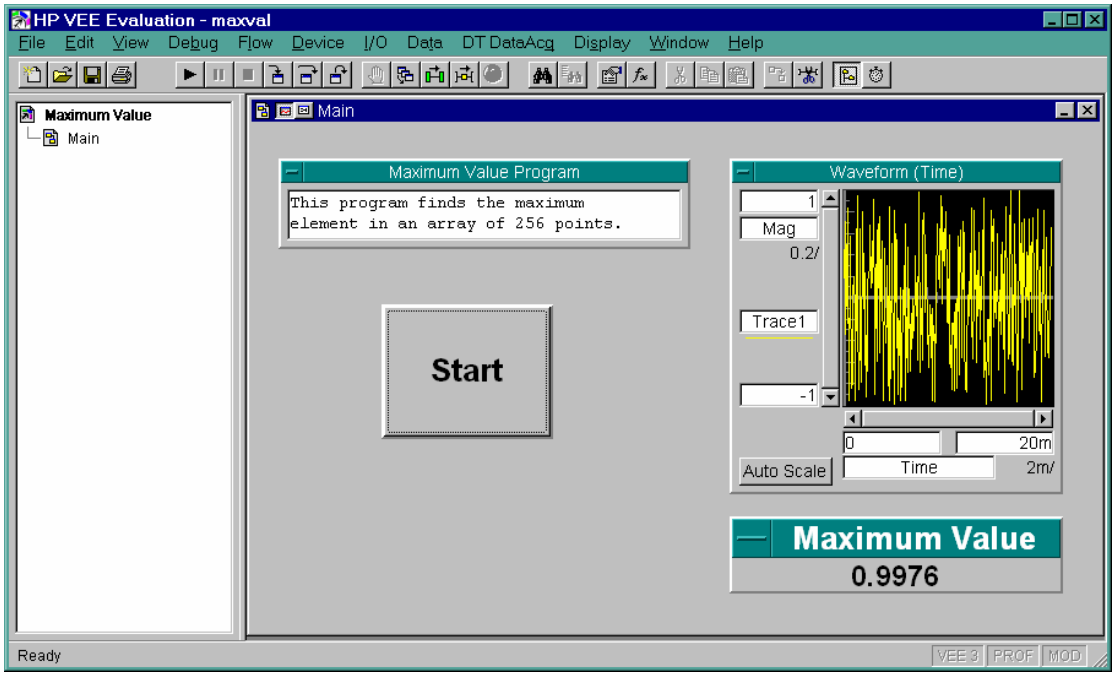
```
/* Programa  em C para gerar um vetor de números
aleatórios
   e encontrar o máximo dentre eles */

#include <stdio.h>
#include <math.h>

main( )
{
 double num[256], max;
 int i;

 printf("O vetor de numeros aleatorios:\n\n");
 for(i=0; i<256, i++){
        num[i]=((double) rand( ) / 32768 * 2) - 1;
        printf("%f\n", num[i]);
        }

 max=num[0];
 for(i=1; i<256, i++){
        if(num[i] > max) max = num[i];
```

Figura 4 - Implementação da solução em C

A figura 4 mostra a implementação do programa em C. Nele é criado um vetor *num[ ]* de 256 entradas em ponto flutuante de dupla precisão. Essas entradas são preenchidas com números aleatórios através da função *rand( )* repetida 256 vezes no primeiro laço *for.* O programa imprime as entradas do vetor, uma em cada linha. No segundo laço *for*, varre-se o vetor buscando o valor máximo contido no mesmo, que então é impresso.

[5] *HP-VEE* é marca registrada da Hewlett-Packard, Estados Unidos da América.

a)



b)

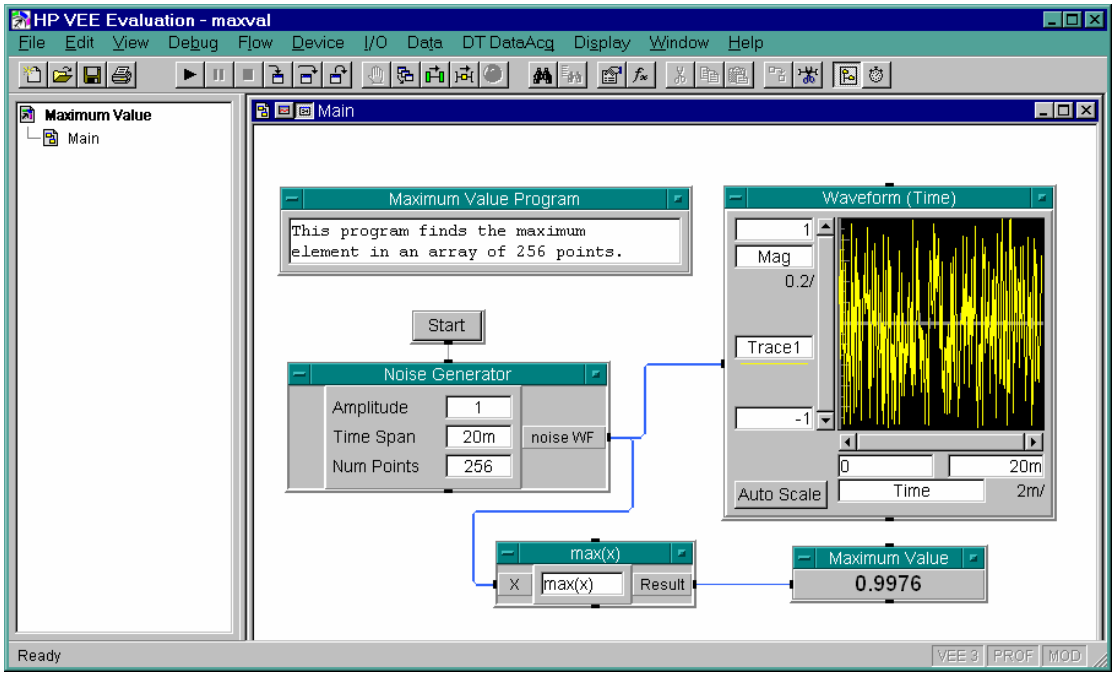
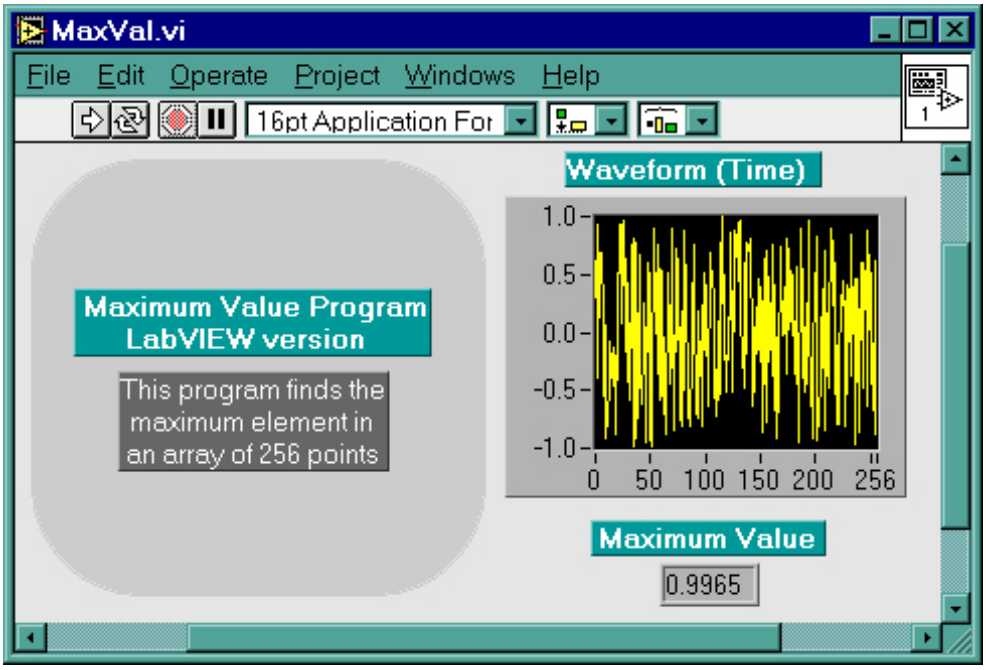


Figura 5 - Solução implementada com *HP-VEE*.

A figura 5 mostra a solução obtida com o programação visual usando o ambiente de desenvolvimento HP-VEE. Na fig. 5.a vê-se o painel frontal e, na 5.b, o diagrama de blocos. Pode-se identificar os seguintes componentes: uma área de texto contendo o título do programa, um botão de partida (*start*), um bloco contendo o gerador de ruído que preenche o vetor, um bloco contendo a função *max(x)* que obtém o máximo do vetor *x* fornecido como entrada, um bloco contendo uma tela oscilográfica e um bloco que mostra um valor numérico que lhe é fornecido como entrada. A tela oscilográfica faz o papel da impressão dos valores do vetor. Ela poderia ter sido substituída por um bloco contendo uma tabela que mostraria os valores do vetor.

a)



b)

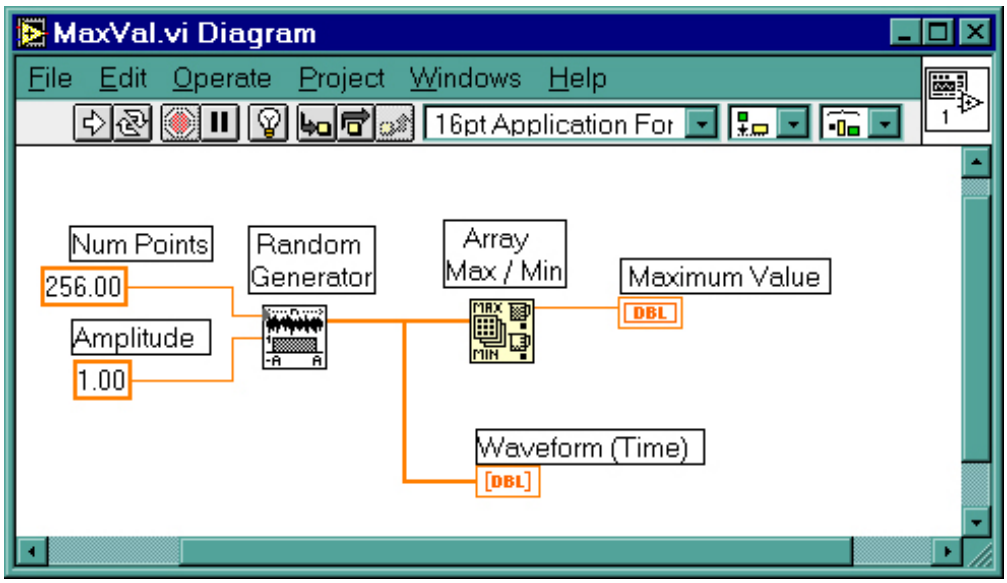


Figura 6 - Solução implementada com *LabvIEW.*

A figura 6 mostra a implementação do programa com o ambiente de desenvolvimento *LabVIEW*. Em (a) vê-se o painel frontal e em (b) o diagrama de blocos. O painel frontal contém os seguintes componentes: um painel decorativo de apresentação com o título da aplicação, uma tela oscilográfica, exibindo o vetor de números aleatórios de forma gráfica e uma caixa contendo o valor máximo. Estes dois últimos blocos, a tela e a caixa apresentam componentes correspondentes no diagrama de blocos, que podem ser identificados pelos rótulos idênticos aos que aparecem no painel frontal, respectivamente. A implementação foi feita de modo a se aproximar o máximo da feita com o *HP-VEE*.

Essencialmente a programação visual é realizada escolhendo-se em uma caixa de ferramentas cada componente, que é exibido no local do diagrama de blocos que o usuário

selecionar com o *mouse*. Os componentes são então conectados selecionando-se com o mouse as terminações dos componentes que devem ser interligadas e desenhando-se o trajeto da ligação sobre o diagrama de blocos, com o apontador do mouse. No caso do *LabVIEW*, a cor e espessura da ligação se ajustam automaticamente ao contexto, de acordo com o tipo de dado que está sendo passado de um componente para outro.

A utilização de componentes de *software* em programação (visual ou textual) é uma prática possível graças à orientação a objetos, que tem peculiaridades conceitualmente diferentes da tradicional programação estruturada. Em instrumentação virtual, os programas são denominados instrumentos virtuais (*virtual instruments* - **VI**s). Os *componentes* que são os blocos construtivos usados na elaboração de um instrumento virtual, são *objetos,* no sentido da programação orientada a objetos, que pode ser encontrada no *apêndice A*. Alguns usam o termo *componentware* para designar *software* baseado em componentes. Os componentes são módulos de *software* reutilizáveis (geralmente pré-compilados).

5. Execução simultânea (*Multitasking* e *Multi-threading*)

O computador tem seu funcionamento administrado pelo *sistema operacional*, como o Linux, o Windows e o MacOS por exemplo. Ele é uma peça de software que decide quem usa qual parte da memória, que programas são executados, quais tarefas o computador está realizando e de que forma. Conceitualmente, os componentes e objetos são peças de programação que podem ter existência independente umas das outras. Isto é, se o sistema operacional permitir, os diversos objetos e componentes de um programa poderão estar sendo executados paralelamente (ou concorrentemente), de modo assíncrono. Seu funcionamento conjunto, traduzindo a funcionalidade do programa, dar-se-á pela troca de informações, isto é, dados. Dessa forma, os objetos trocam mensagens uns com os outros e, assim, o estado do sistema vai evoluindo, produzindo a execução da tarefa. Diz-se, então, que a execução dos objetos é disparada de forma *simultânea*, isto é, independentemente uns dos outros, de acordo com a capacidade do sistema.

O tipo de simultaneidade de que os objetos poderão desfrutar, na prática, depende da plataforma de execução (*hardware* e sistema operacional). Uma plataforma pode ser multiprocessada (paralela) ou multitarefa. As plataformas multiprocessadas dispõem de mais de um processador, podendo de fato executar simultaneamente (do ponto de vista de tempo físico) mais de um processo [6]. As plataformas multitarefa são dotadas de sistemas operacionais capazes de administrar vários processos executando em tempo partilhado, isto é, cada processo é partido em trechos e cada trecho é executado durante um intervalo de tempo curto, parando em seguida para dar vez à execução do trecho de um outro processo e assim por diante. Com isso o sistema funciona como se vários processos parecessem estar sendo executados ao mesmo tempo[7].

Quando um dado processo contém vários objetos, pode-se executá-los de forma "simultânea" repartindo o uso do processador entre eles no modo de tempo partilhado. Esse tipo de multitarefa é denominado *multithreading* e cada módulo do processo que executa em "paralelo" denomina-se um *thread* de execução. Não entraremos em maiores detalhes quanto a esses termos e vamos entender os objetos como se fossem trechos de um processo que podem executar quase-paralelamente em uma plataforma multitarefa. Guarde bem o seguinte:

- *multitasking* refere-se a executar vários programas simultânemanente
- *multithreading* refere-se a executar vários trechos (objetos) de um mesmo programa simultâneamente.

Os módulos do LabVIEW (as “caixinhas” constituidas pelos ícones de sua linguagem gráfica) são objetos com *threads*, que podem executar simultâneamente. A forma de administrar a repartição do tempo de processador entre esse objetos pode levar à execução assíncrona ou síncrona. No caso do LabVIEW, há módulos que devem executar de forma

[6] Entenda-se por *processo* um programa. Um processo pode conter um ou vários objetos.
[7] Tempo aqui é do ponto de vista computacional, isto é, com base no relógio (*clock*) da máquina.

síncrona sempre. Um módulo é dito *síncrono* quando sua execução, uma vez iniciada, deve seguir até o fim, antes que outro módulo possa ser executado. Ou seja, módulos síncronos afetam o "quase paralelismo" de execução entre os objetos. Voltaremos a esse ponto na seção 7.

Esses conceitos são essenciais para permitir a concepção dos ambientes de programação visual que compõem o *LabVIEW* e o *HP-VEE*. Os componentes desses ambientes são objetos, instâncias de classes que se comunicam. As mensagens que trocam uns com os outros são especificadas pelas conexões presentes no diagrama de blocos. Seus blocos são disparados em tempo compartilhado quando o programa é executado. O encapsulamento dos dados é essencial, para garantir que os componentes possam existir com base na sua funcionalidade, sem nos preocuparmos com detalhes de sua implementação. O mecanismo de passagem de mensagens entre objetos requer que se defina interfaces padronizadas entre os objetos e protocolos que indiquem como um bloco genérico passa informação para outro. As interfaces especificam a forma como os dados são apresentados para entrada e saída de um bloco. Os protocolos especificam a forma como os dados devem ser transferidos de um bloco para outro, através de suas interfaces. A figura 7 ilustra esses conceitos.

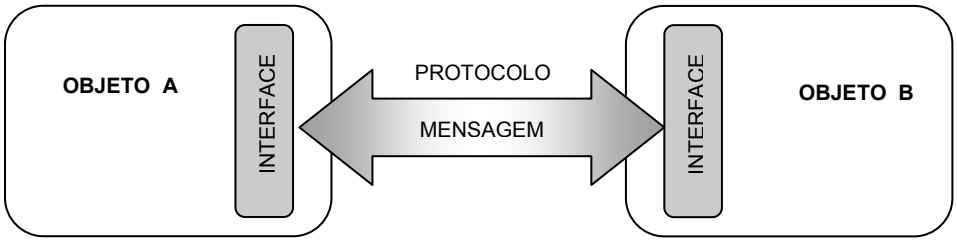


Figura 7 - Interfaces e protocolos de troca de mensagens (dados, sinais) entre os objetos. (*Observação* - o termo "sinal" foi usado significando um tipo particular de dado que comunica um evento ao outro objeto.)

Os instrumentos reais acoplados ao computador aparecem nesse ambiente de programação visual como blocos especializados. Esses blocos recebem as informações vindas através das interfaces eletrônicas com os instrumentos reais e as transformam em variáveis, passadas aos demais blocos como mensagens contendo os dados. Resta discutir o que são esses blocos especializados e que tipo de comunicação eles executam com os instrumentos reais.

6. Interfaces de Instrumentação

Há dois aspectos a se considerar quanto ao interfaceamento dos instrumentos reais com o ambiente de objetos da instrumentação virtual: (i) o componente de interfaceamento e (ii) o padrão de comunicação. O componente de interfaceamento refere-se ao módulo de *software* ou objeto que representa o instrumento real dentro do ambiente de programação. O padrão de comunicação determina a forma como os dados são transferidos (tipo de barramento de dados e protocolo de comunicação). O componente de interface deve ser compatível com o protocolo de comunicação usado.

Os tipos de barramentos de dados mais comuns para uso com microcomputadores são: via serial e via paralela. Nos barramentos seriais, há uma única via física de comunicação, por onde passam os bits dos dados serialmente. Nos barramentos paralelos, há várias vias físicas para transmissão dos bits dos dados paralelamente. Além das vias de dados, esses barramentos têm outras vias dedicadas a sinais de controle e aterramento. Existem normas que padronizam os parâmetros dessas vias e a forma como são inseridos os dados e sinais nas vias (protocolos). Os protocolos seriais mais utilizados são o RS-232C , o

RS-449A, o *Universal Serial Bus* (USB) e o IEEE-1394 (*FireWire*). Os protocolos paralelos mais conhecidos são o IEEE-1284 (Centronix) e o IEEE-488 (GPIB). Discutiremos aqui apenas este último.

6.1. A interface GPIB (IEEE - 488)

Em 1965, a companhia americana Hewlett-Packard projetou uma interface e barramento de comunicação para conectar seus instrumentos programáveis, então denominado HP-IB (Hewlett-Packard Interface Bus). Essa interface se popularizou rapidamente devido à alta taxa de transferência que suportava (1Mbyte/s nominal). Isso levou a sua padronização pelo IEEE [8] , em 1975, com o nome GPIB - General Purpose Interface Bus - ou IEEE488. Em 1992 foi editada a segunda revisão dessa norma, que vigora até hoje, a IEEE-488.2.

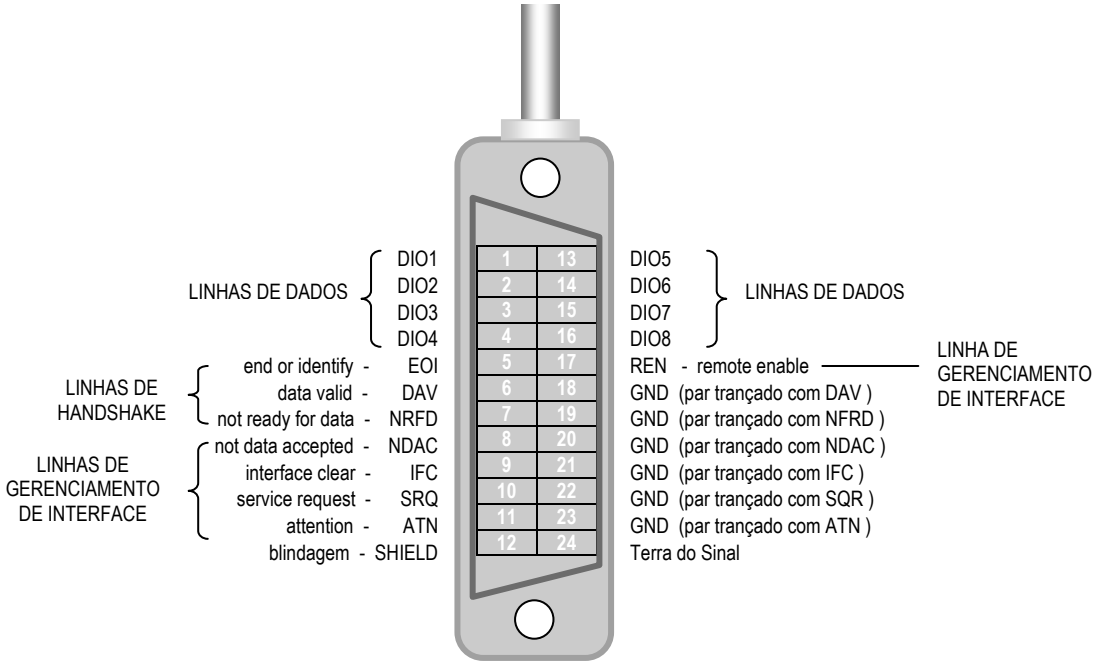


Figura 8 - Conector GPIB e sinais correspondentes

A figura 8 mostra o conector padronizado para a interface GPIB e os sinais correspondentes a cada pino. Os dispositivos GPIB comunicam-se uns com os outros enviando dois tipos de mensagens:

- Mensagens independentes de dispositivo - são os dados (resultados de medidas, instruções de programação, *status* da máquina, arquivos de dados)
- Mensagens de comando da interface - são destinadas à execução de funções tais como: endereçamento de dispositivos, programação da interface, inicialização do barramento de interface, etc.

O padrão GPIB prevê três tipos de entidades conceituais correspondentes a certas classes de ações: as que falam (*talkers)* , as que escutam (*listeners)* e as de controle (*controllers)*. O *talker* envia dados para um ou mais *listeners*, que recebem os dados. O *controller* gerencia o fluxo de informações no GPIB enviando comandos a todos os dispositivos. Um voltímetro digital, por exemplo, é um *talker* e um *listener*, enquanto um computador com uma placa GPIB pode ser os três tipos de entidade.

Pode-se usar dois tipos de configuração das conexões entre os dispositivos ligados em um GPIB: linear e estrela, como mostrado na figura 9.

[8] IEEE - Institute of Electrical and Electronic Engineers – veja mais em : www.ieee.org

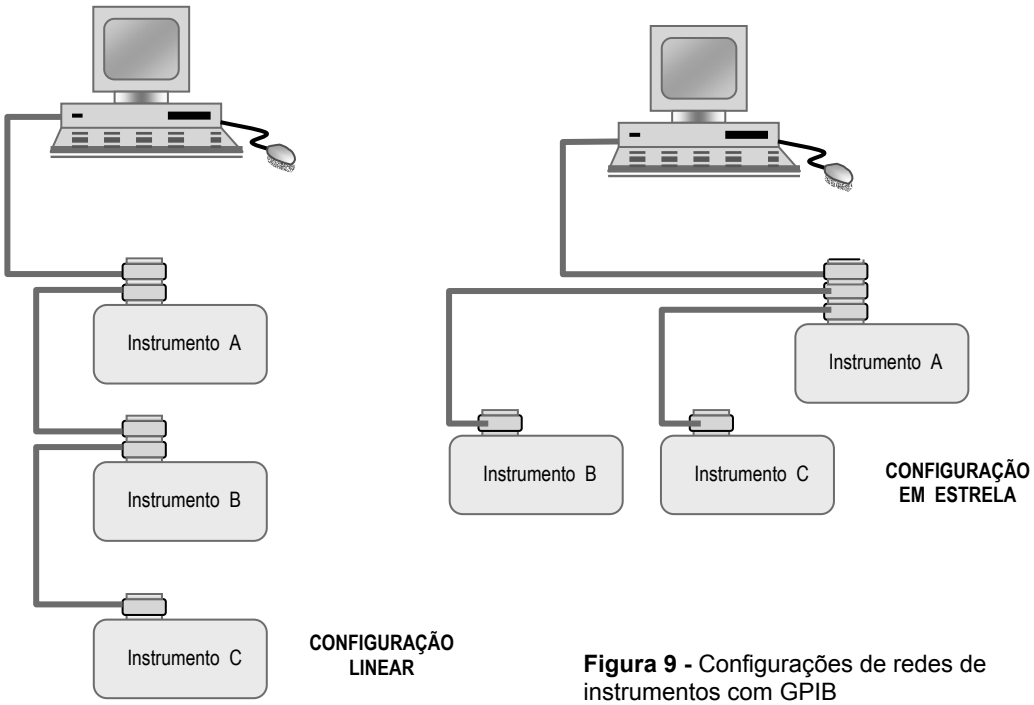


Figura 9 - Configurações de redes de instrumentos com GPIB

O GPIB apresenta as seguintes restrições de montagem:

- Máxima separação entre instrumentos: 4 m
- Separação média máxima entre instrumentos: 2 m
- Máximo comprimento de cabo: 20 m
- Número máximo de instrumentos: 15 (até 10 ligados)

6.2. Interfaceamento de sensores e atuadores analógicos

Sensores e atuadores que utilizam diretamente sinais analógicos podem ser interfaceados com um ambiente de instrumentação virtual através de dispositivos de aquisição de dados (DAQs). Os DAQs são dotados de conversores analógico-digitais e podem ser placas conectadas diretamente ao barramento do computador ou dispositivos externos conectados via serial, paralela ou outros (SCSI, TCP/IP, etc).

6.3. Componentes de interfaceamento de instrumentos (*instrument drivers*)

Os instrumentos reais são vistos pelo ambiente através de um componente de *software* apresentado como um bloco no diagrama de blocos. Esses componentes são denominados *instrument drivers* e devem ser compatíveis com a forma de comunicação utilizada (GPIB, serial, barramento interno, etc). Dentro desse bloco encontra-se o programa capaz de se comunicar com o *hardware* do instrumento e controlá-lo. Esse módulo pode ser implementado seguindo um padrão de arquitetura de *software* que permite tratar os dispositivos de maneira bastante uniforme e simplificada, denominado VISA - Virtual Instrument Software Architecture. Esse padrão inclui métodos para tratamento de eventos e de erros. O VISA cobre uma extensa classe de tipos de interfaceamento: GPIB, serial, barramentos PCI (PCI extendido e compacto - PXI), VME (VME extendido e compacto - VXI), PCMCIA, etc.

7. Ambientes comerciais de instrumentação virtual

Vamos discutir aqui alguns aspectos dos dois mais populares ambientes de instrumentação virtual: o *HP-VEE* e o *LabVIEW*.

7.1. Hewlett-Packard Virtual Engineering Environment

O HP-VEE é o ambiente de instrumentação virtual criado pela Hewlett-Packard para trabalhar com seus próprios equipamentos e demais instrumentos compatíveis. A figura 5 mostra o aspecto e os elementos essenciais desse ambiente. O HP-VEE foi implementado usando o paradigma de programação orientada a objetos. Entretanto, não é um ambiente de desenvolvimento de programas orientados a objetos (Baroth e Hartsough, 1995 em [1]). Essencialmente oferece ao usuário módulos cujos parâmetros e aspecto podem ser ajustados a gosto e conectados para compor uma tarefa complexa. Esses módulos são objetos, instâncias de classes como *geradores, interfaces de operador, mostradores, instrumentos, objetos matemáticos* e outros. Os módulos são, em princípio, disparados assincronamente. Todavia, permite a execução síncrona, através da especificação da ordem de execução feita através da conexão de terminais especiais existentes em todos os blocos de módulos - os terminais de *entrada e de saída de ordem sequencial* (vide figura 5: o terminal onde se liga o botão *start* é a entrada de ordem de sequência do bloco do gerador de ruído e o correspondente no bloco do botão de *start* é o seu terminal de ordem de saída). A especificação da sequência por esse método pode vir a se tornar uma tarefa difícil em sistemas muito grandes. O HP-VEE também permite que DLLs geradas a partir de outras linguagens sejam ligadas ao ambiente. O foco de desenvolvimento e comercialização do HP-VEE tem sido basicamente atender às necessidades de uso de instrumentos produzidos pela própria HP. O HP-VEE era originalmente interpretado, porém sua versão mais recente já produz código compilado.

7.2. Laboratory Virtual Instrument Engineering Workbench

O LabVIEW foi criado pela National Instruments como um ambiente de programação voltado ao desenvolvimento de aplicações, realizado através de programação visual orientada pelo fluxo de dados, com o foco em instrumentação virtual. O nome LabVIEW designa o ambiente de desenvolvimento e a linguagem de programação visual denomina-se **G** (de *G*raphics, numa acepção à linguagem **C**). Historicamente, a linguagem G foi primeiramente desenvolvida e, com ela, o ambiente LabVIEW foi implementado. A linguagem G tem expressividade equivalente à de uma linguagem declarativa textual, como o C e o Pascal, por exemplo. Ela é dotada de blocos especiais para controle de fluxo, laços iterativos, etc, como discutido na seção 4. O programa visual criado com esses blocos é compilado e executado dentro do ambiente LabVIEW, que também oferece mecanismos para detectar, rastrear e diagnosticar erros de programação. Durante a execução, os objetos são disparados assincronamente, como *threads* independentes. Há duas exceções a essa regra, que forçam a sincronização (isto é, são módulos executados até o fim): (i) os blocos dentro de uma *sequência* , definidos como explicitamente sequenciais e (ii) os CINs. Estes, os *Code Interface Nodes* são blocos que contêm código desenvolvido em C, por exemplo, pelo usuário. Como não é possível em princípio prever o comportamento desses módulos, os desenvolvedores do LabVIEW optaram por uma postura conservadora, forçando esses blocos a serem síncronos por princípio.

Os programas desenvolvidos em G são genericamente designados por *VIs* (*Virtual Instruments*), mesmo que não estejam implementando uma tarefa de instrumentação. Um VI pode ser encapsulado em um bloco e chamado como sub-rotina de um outro VI. O VI encapsulado passa a se chamar então *Sub-VI*, embora possa ser executado independentemente, desde que esteja inicializado corretamente (pré-condicionado completamente). O objetivo desta experiência de PSI-2315 é obter alguma prática com desenvolvimento de programas em G, construindo alguns VIs simples.

Sumário

O que essencialmente se deve ter aprendido aqui é que:

- Um instrumento virtual oferece uma interface gráfica com a mesma funcionalidade que aquela oferecida pelos mostradores e painéis de instrumentos reais.
- Essa interface é realizável em computadores dotados de ambiente gráfico e explora conceitos de programação orientada a objetos.
- Os *componentes* de um instrumento virtual são módulos reutilizáveis de *software* que podem sem implementados como bibliotecas incluídas dinamicamente -- as DLLs.
- Os instrumentos reais são interfaceados com o ambiente de programação através de componentes especializados, denominados *instrument drivers,* implementados segundo um padrão que facilita seu uso de um modo mais uniforme -- o VISA. Os *instrument drivers* devem ser compatíveis com o protocolo de comunicação usado para conectar o instrumento real ao computador.
- O padrão de comunicação utilizado nos instrumentos existentes no laboratório de eletricidade é o GPIB, ou IEEE-488. Esse padrão usa um barramento de interface paralelo de 8 vias de dados, 3 vias de *handshake* e 5 vias de gerenciamento da interface. Cada instrumento recebe um endereço particular no barramento GPIB.

Agradecimentos

O autor agradece pela colaboração prestada através de revisão e sugestões feitas por Edson T. Midorikawa, Maurício O. P. Lisboa, Ivandro Sanches e Denise Consonni.

Também agradece à National Instruments e seus engenheiros, Leandro Fonseca e Paulo Sérgio Pereira, pela apresentação de parte do seminário e pelas demonstrações de aplicações.

Bibliografia

1. M.M. Burnett, A. Goldberg e T.G. Lewis (editores) - *Visual Object-Oriented Programming - Concepts and Environments* - Manning, Greenwich, 1995

2. S. Sigfried - *Understanding Object-oriented Software Engineering* - IEEE Press - New York, 1996

3. A.S. Tanenbaum - *Modern Operating Systems* - Prentice-Hall, Englewood Cliffs, 1992

4. G.W. Johnson - *LabVIEW Graphical Programming* - McGraw-Hill, New York, segunda edição, 1997

5. Hewlett-Packard - *Exploring HP-VEE* - 1995

6. National Instruments - *LabVIEW User Manual* - 1996

7. National Instruments - *Instrumentation Catalogue* - 1998

Questionário de Auto-avaliação

As questões a seguir permitem que você avalie se entendeu o texto e captou os principais aspectos conceituais:

1. Quais as partes de um instrumento ?
2. Qual a diferença entre *sensor, transdutor e atuador* ?
3. O quê é uma *interface conceitual* ?
4. Quais são os componentes da interface conceitual de um osciloscópio ?
5. Quais são os componentes da interface conceitual de um computador ?
6. Dê um exemplo de um ambiente janelado de um computador.
7. O quê são *componentes* de *software* ?
8. O quê significa a sigla *VI* ?
9. O quê é o painel frontal de um VI ?
10. O quê é o diagrama de blocos de um VI ?
11. O quê é o *sistema operacional* ? Dê um exemplo ?
12. O que é um *processo* (em um computador) ?
13. O quê é um sistema operacional *multi-tarefa* ?
14. O quê é um sistema operacional *multi-threaded* ?
15. O quê é um protocolo de comunicação ?
16. O quê é o protocolo GPIB ? Quais suas particularidades interessantes ?
17. O quê é a linguagem *G* ?

APÊNDICE A

Este apêndice visa satisfazer a curiosidade e complementar alguns conceitos mencionados na apostila. Sua leitura não é indispensável, porém esclarece alguns pontos.

Programação Orientada a Objetos

O método de programação estruturada procura dividir o programa em blocos que guardam entre si uma relação hierárquica perfeita, isto é, não há desvios incondicionais de uma seção para outra. Uma forma especialmente interessante de programação estruturada é a *programação modular*. Este método consiste em dividir a aplicação em *módulos*, que são trechos de programa que se relacionam exclusivamente através de interfaces por onde trocam dados (em geral os argumentos de funções e procedimentos). Isto é, para entrar e sair de um módulo a via é sua interface e passa-se de um módulo A para um módulo B realizando-se no primeiro uma chamada do segundo, retornando-se ao módulo A quando a execução do B encontrar uma instrução de retorno. Em programação modular pode-se usar os chamados *tipos abstratos de dados* (que envolvem por exemplo, estruturas híbridas onde se mistura variáveis inteiras, em ponto flutuante, caracteres, etc).

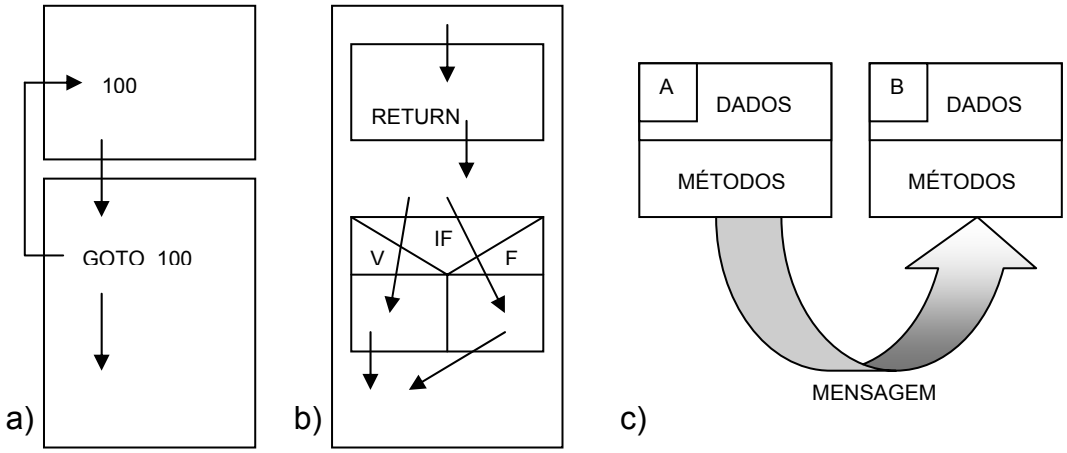


Figura 10 - Paradigmas de programação - (a) *programação não-estruturada* - o fluxo do programa pode sofrer desvios incondicionais; (b) *programação estruturada* - o fluxo segue a hierarquia de blocos e de condições (V = verdadeiro, F = falso); (c) *programação orientada a objetos* - os módulos são executados "simultaneamente" e se comunicam trocando mensagens. O tipo de simultaneidade irá depender do sistema (plataforma de execução e implementação do ambiente de programação).

Na programação modular os módulos relacionam-se uns com os outros trocando dados. Entretanto, cabe ao programador definir a forma correta de passar um dado de um módulo para o outro. Isto é, o programador precisa conhecer como foi implementada internamente a estrutura do dado, para poder usá-lo.

Na programação orientada a objetos os módulos passam a se chamar *objetos*. Mas é mais que uma simples mudança de nomes. A diferença agora é que os dados de cada módulo só podem ser vistos externamente ao módulo através de funções especiais chamadas *métodos*. Os métodos sabem como manipular os tipos de dados do objeto correspondente de forma adequada. Dessa maneira, usando essas funções chamadas métodos, o programador consegue usar os dados de um objeto, sem precisar saber como a estrutura de dados está implementada internamente. Diz-se que os dados estão *escondidos* (*data hidding*). Essa propriedade dos objetos denomina-se *encapsulamento.* O seguinte exemplo ajudará a entender melhor esses conceitos.

a)

```
typedef struct{
        float   parte_real,
        float   parte_imag}
complexo;

/* exemplo de uso: */

complexo Z1, Z2, Z3;

Z3.parte_real = Z1.parte_real  +
                Z2.parte_real;
```

b)

```
typedef float complexo[2];


/* exemplo de uso: */

complexo Z1, Z2, Z3;

Z3[0] = Z1[0] + Z2[0];

Z3[1] = Z1[1] + Z2[1];
```

Figura 11 - Implementação do tipo de dado *complexo* para representar números complexos em linguagem C . Em (a) o tipo complexo é implementado como uma *struct*. Em (b) a implementação é feita com um *array* de duas entradas, onde a primeira é a parte real e a segunda a parte imaginária.

Suponhamos que se deseja implementar um programa que manipula números complexos, por exemplo, para aplicação em análise de circuitos em regime permanente senoidal. Sem usar conceitos de programação orientada a objetos, a concepção usual seria construir um tipo abstrato de dado e batizá-lo como *complexo* por exemplo. Em linguagem C há duas formas de se fazer isso, como mostra a figura 11.

Na programação modular, para realizar a adição dos números complexos Z1 e Z2, o programador precisa saber como foi implementado o tipo de dado complexo. Se a implementação mudar por alguma razão, de uma forma para a outra, o programador será obrigado a alterar a forma de calcular a adição. Já no caso de se usar programação orientada a objetos esse problema não ocorre.

Na figura 12 apresenta-se a solução baseada em objetos. Para implementar números complexos como objetos, cria-se uma *classe* para acomodá-los. A classe contém as declarações especificando a estrutura e as propriedades dos objetos que compreende. Na figura 12 não é mostrado como o tipo complexo está implementado na classe *complexo*; não foi mostrado se foi feito como um *array* de duas entradas ou uma *struct*. A figura 12 mostra a realização da adição quando o tipo complexo é um objeto. Tudo que o programador precisa saber é o nome da função que manipula suas partes real e imaginária. Ele não precisa saber como o tipo complexo está implementado internamente ao objeto. Entretanto, sabemos que dispomos de duas funções (os métodos) que retornam a parte real e a parte imaginária do complexo (também não foi mostrado como essas funções foram implementadas, pois também é irrelevante).

Nas linguagens orientadas a objeto, os objetos são agrupados em *classes*. Quando se especifica um tipo de objeto, define-se de fato a sua classe. No exemplo, define-se *a classe complexo*. Quando se deseja criar objetos de uma classe (no exemplo, os números complexos Z1, Z2 e Z3), basta-se declarar o nome da classe e especificar o nome dos objetos, que são denominados *instâncias* dessa classe . No exemplo, Z1, Z2 e Z3 são instâncias da classe *complexo*.

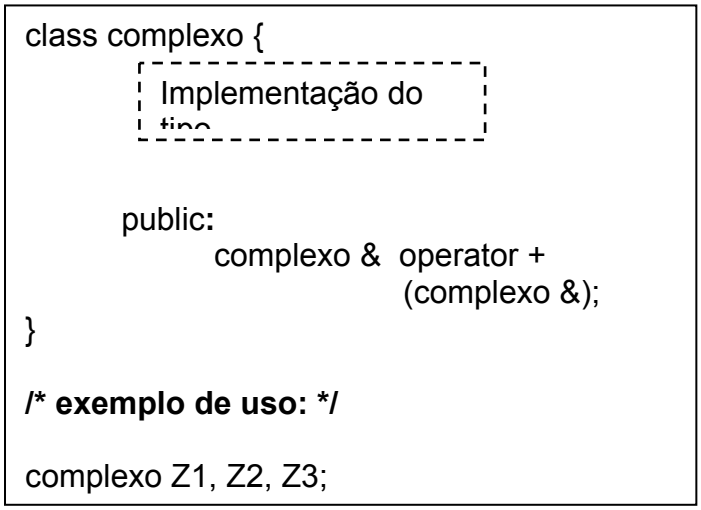


Figura 12 - Adição de números complexos implementados como objetos. A classe *complexo* compõe-se de uma parte privada, protegida, que é a implementação do tipo de dado, que está escondida, não sabemos se foi usado *struct* ou *array*. A parte pública mostra o protótipo da função (método) que manipula o tipo complexo tal como criado na parte privada. Para quem usa o objeto *complexo* assim criado, a parte pública é a única parte visível e relevante. Tal como implementado nesse exemplo (em C++) o operador **+** foi modificado para ser capaz de realizar a adição complexa corretamente (sobrecarga de operador).

```
class complexo {

                float Real;
                float Imag;

        public:

                complexo & operator +  (complexo &);  /* prototipo do metodo */

        }


// implementacao do metodo ( via sobregarga do operador de adicao )

        complexo &          complexo : :  operator + (complexo & arg )

        {
                float    re,   im ;

                re =  Real +  arg.Real ;
                im = Imag + arg.Imag ;

                return  complexo ( re, im ) ;  /* chama construtor */

        }
```

Figura 12 B - Implementação detalhada do tipo *complexo* como um objeto, em C++

Sobrecarga de operador - na figura 12 foi apresentado o método que realiza a adição dos números complexos através da *sobrecarga do operador* de adição. Esse expediente consiste em usar o símbolo de um operador já existente para designar uma nova operação "similar" (no sentido algébrico).

Os objetos e classes possuem a propriedade de *herança*. Isto é, podemos usar uma classe já definida como elemento para criar uma nova classe. Feito isso, as instâncias dessa nova classe herdam as propriedades da classe original também. Por exemplo, se definirmos

uma nova classe e a chamarmos *espectro* e usarmos a classe *complexo* em sua definição, a nova classe *espectro* terá as propriedades da classe *complexo*, juntamente com as novas propriedades que especificarmos para a classe *espectro*.

Programação Orientada pelo Fluxo de Dados

Já dissemos que na linguagem G usada pelo LabVIEW, os módulos correspondentes aos ícones (ou "caixinhas") são objetos e contêm *threads* independentes, isto é, podem ser executados "simultâneamente". Entretanto, a programação em si do LabVIEW, até a versão 6i, de 2001, não é a rigor orientada a objetos, ainda que seus componentes sejam objetos e que tenha sido desenvolvida em C++. A forma com que o usuário programa o LabVIEW é denominada programação orientada a fluxo de dados.

Nesse paradigma de programação, os diversos módulos executam condicionados pelo fluxo de dados. Isto é, se um módulo B depende dos dados gerados por um outro módulo A, aquele só pode executar quando este já lhe houver fornecido os dados. Ao conectarmos duas "caixinhas" do LabVIEW, estamos condicionando a execução de um módulo pelo fluxo dos dados provenientes do outro que metaforicamente passam de um para o outro através do "fio" de conexão.

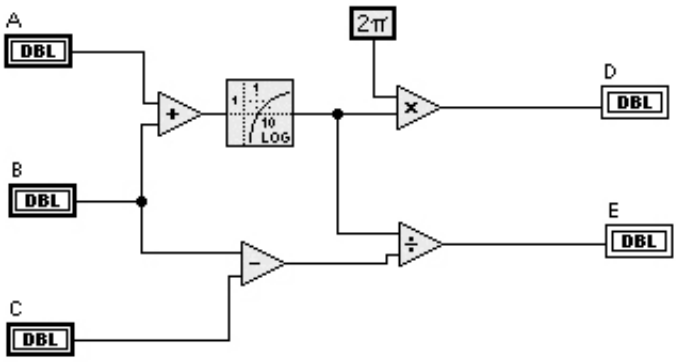


Figura 13 – Controle pelo fluxo de dados.

Todas as "caixinhas" na figura 13 têm *threads* independentes e podem executar simultâneamente, de forma assíncrona, em princípio. Entretanto as conexões entre elas criam dependência pelo fluxo de dados. Portanto, a adição ea subtração podem executar "simultâneamente" porém os demais módulos ficam na dependência da disponibilidade dos dresultados produzidos pela execução dos módulos anteriores.

APÊNDICE B

GLOSSÁRIO

1.*Metáfora computacional* - é uma forma de abstração da relação semântica entre o programa, o processo computacional subjacente e o domínio sobre o qual agem. Em uma metáfora o significado da entidade à qual se refere (referido) é caracterizado pelo nome que a referencia (referente). Assim, a metáfora *instrumento virtual* refere-se a um conjunto de programas que é interpretado como se fosse um instrumento. Exemplos de outras metáforas computacionais: *componente reusável* e *agente inteligente*.

2. *Aparelho, equipamento, instrumento* - em diversas ocasiões empregamos esses termos sem maior preocupação com seu significado. Um *aparelho* é um conjunto de dispositivos que opera de forma harmoniosa de modo a conferir ao sistema todo a característica de unidade. O termo *equipamento* pode referir-se a um ou mais aparelhos compondo um sistema destinado à execução de um conjunto tarefas. O *instrumento* é um aparelho destinado à intervenção de um operador em um determinado meio, seja para medir ou para controlar suas variáveis e parâmetros.

3. *Sensor, atuador e trandutor* - em muitos textos de instrumentação, o termo transdutor é utilizado para designar a classe que engloba os sensores e os atuadores. No presente texto consideramos como *transdutor* uma abstração da função de transformar uma grandeza física de um tipo em um outro mais adequado (por exemplo, transformar temperatura em sinal elétrico). Da mesma forma, designamos por *sensor* a abstração da função de responder a uma variável física (por exemplo, a temperatura). Todavia, em termos práticos, geralmente não é possível separar a parte sensorial da transdução em um dispositivo sensor real. O mesmo raciocínio vale para o *atuador*, que no presente texto designa a função de alterar uma dada variável física.

4. *DLL - Dinamic Linked Library* - é um módulo de programa pré-compilado que pode ser utilizado por um outro programa. A passagem de parâmetros e dados entre a DLL e o programa que a utiliza se faz através de uma interface de programação pré-estabelecida. O programa pode conter funções que são definidas na DLL.

5. *Componente* - é um módulo de programação implementado como objeto pré-compilado na forma de uma DLL. Os componentes que formam os blocos do LabVIEW são dessa categoria. À medida com que novos blocos de um instrumento virtual vão sendo criados, eles vão sendo imediatamente compilados para serem guardados em uma biblioteca particularmente criada para aquele VI. Quando o VI é executado, portanto, a maior parte de seu "código" já está pré-compilada.