UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE MESTRADO EM ENGENHARIA ELÉTRICA

MÓDULOS: GERENCIADOR DE TELAS E GERENCIADOR DE PERIFÉRICOS DE UM SISTEMA DE PROTOTIPAGEM RÁPIDA USUÁRIO-COMPUTADOR

RINALDO SANTOS JÚNIOR

CAMPINA GRANDE
ABRIL - 1992

RINALDO SANTOS JÚNIOR

MÓDULOS: GERENCIADOR DE TELAS E GERENCIADOR DE PERIFÉRICOS DE UM SISTEMA DE PROTOTIPAGEM RÁPIDA USUÁRIO-COMPUTADOR

Dissertação apresentada ao Curso de MESTRADO EM ENGENHARIA ELÉTRICA da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do Grau de Mestre.

ÁREA DE CONCENTRAÇÃO: **PROCESSAMENTO DA INFORMAÇÃO**

MARIA DE FÁTIMA Q. VIEIRA TURNELL
(Orientador)

CAMPINA GRANDE
ABRIL - 1992

S237m Santos Junior, Rinaldo
Modulos : gerenciador de telas e gerenciador de perifericos de um sistema de prototipagem rapida usuario-computador / Rinaldo Santos Junior. - Campina Grande, 1992.
131 f. : il.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal da Pariaba, Centro de Ciencias e Tecnologia.

1. Perifericos 2. Software - 3. Dissertacao I. Turnell, Maria de Fatima Q. Vieira, Dra. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.35(043)

MODULOS: GERENCIADOR DE TELAS E
GERENCIADOR DE PERIFERICOS DE UM SISTEMA DE
PROTOTIPAGEM RAPIDA DE INTERFACES USUARIO-COMPUTADOR

RINALDO SANTOS JUNIOR

DISSERTAÇÃO APROVADA EM 30.04.92

Maria de Fatima Q. V. Turnell

MARIA DE FATIMA Q. VIEIRA TURNELL, Ph.D., UFPB
Orientadora

JOSE ANTAO BELTRÃO MOURA, Ph.D., UFPB
Componente da Banca

MISAEL ELIAS DE MORAIS, Dr.-Ing., UFPB
Componente da Banca

CAMPINA GRANDE - PB
ABRIL - 1992

A todos que, de alguma forma, ajudaram na realização e êxito deste trabalho, agradeço.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

ABSTRACT

The objective of this dissertation is to present the functional characteristics, and detail the implementation, of a software tool for management of input and output peripheral devices called Presentation Manager.

At first, implemented as part of a Computer-User Interface Rapid Prototyping System, this Presentation Manager can be easily integrated to other applications which run under MS-DOS for the IBM-PC systems. This integration can occur at different layers, each demanding different levels of background knowledge by the designer.

One of the main characteristics of this work is the transparency achieved during the management of the peripheral I/O devices (where physical devices can be regarded and treated as logic devices) and the resulting ease of adding new devices to an application which uses this Presentation Manager.

RESUMO

Esta dissertação tem o propósito de apresentar as características funcionais e detalhar a implementação de uma ferramenta de software para gerenciamento de dispositivos periféricos de entrada e saída chamada Gerenciador de Apresentação.

Inicialmente implementado para um Sistema de Prototipagem Rápida de Interfaces Usuário-Computador, este Gerenciador de Apresentação pode ser facilmente integrado a outros aplicativos que trabalhem em ambiente MS-DOS do IBM-PC. Esta integração pode ocorrer em diferentes camadas, cada uma delas exigindo graus diferentes de conhecimento por parte do projetista.

Uma das principais características deste trabalho é a transparência obtida na gerência de dispositivos periféricos de entrada e saída (onde dispositivos físicos são tratados como dispositivos lógicos) e a consequente facilidade de acréscimo de novos dispositivos, a um aplicativo que se utilize deste gerenciador.

INTRODUÇÃO

Em 1962, Ivan E. Sutherland defendendo sua tese de doutorado pelo Massachussets Institute of Technology, apresentou o *Sketchpad*, considerada a pioneira das interfaces usuário-computador. No entanto, só em 1984 com o lançamento do MacIntosh, a Apple Computer Inc. tornou conhecida a expressão *interface amigável* a milhares de pessoas em todo o mundo [PERRY 89]. Entenda-se por *interface amigável*, uma interface consistente e agradável ao usuário.

Esta evolução dos conceitos na área de interface combinada com computadores cada vez mais rápidos e com maior capacidade de armazenamento, fez surgir uma nova sub-divisão dentro da tecnologia CASE (*Computer Aided Software Engineering*): Sistemas de Gerenciamento de Interface do Usuário - SGIU [BRANCO 89].

Computadores com capacidade gráfica de alta definição, a um custo cada vez mais baixo, não são por si só a solução para construção de interfaces usuário-computador eficientes. Os sistemas de prototipagem rápida [LUQI 89, LEWIS 89] têm como objetivo aumentar esta eficiência, ao permitir uma maior participação do usuário no processo de definição da interface antes de sua implementação e integração com a aplicação.

Para tanto, uma vez determinadas as necessidades do usuário, constroi-se um protótipo, o qual consiste de um conjunto de informações que define uma interface, tais como, características das janelas, tipos de interação, sequenciamento dos diálogos. Esta definição pode ser feita através de uma linguagem de especificação ou declarativa. O passo seguinte consiste em fazer uma simulação da interface prototipada, oferecendo ao usuário condições de validar ou não o protótipo (total ou parcialmente). Se o usuário não aprovar o protótipo (ou parte dele), este pode ser redefinido sem muito esforço, para mais uma vez ser simulado e submetido a uma nova avaliação.

Estando o usuário satisfeito com a interface, esta é codificada à partir das especificações resultantes deste processo. Portanto, a finalidade principal de um sistema de prototipagem rápida é auxiliar e agilizar a especificação, simulação, validação e documentação de Interfaces Usuário-Computador.

Na versão inicial do Sistema de Prototipagem Rápida AGILE, a validação é baseada em critérios subjetivos aplicados pelo usuário durante sua interação com o protótipo. Pretende-se, no entanto, em versões futuras, utilizar uma métrica baseada no registro de aspectos críticos da interação, tais como: taxa de erro, solicitação de ajuda, tempo de resposta do usuário, etc. com o propósito de ampliar esta avaliação.

Justificativa

O módulo Gerenciador de Apresentação foi concebido a partir da necessidade do Sistema de Prototipagem Rápida para Interfaces Usuário-Computador AGILE [BRANCO 89] centralizar a gerência das informações de entrada e saída em um módulo isolado que garantisse o funcionamento do AGILE independente da configuração do hardware disponível no ambiente IBM-PC.

Uma das opções possíveis teria sido utilizar um pacote comercial, como por exemplo o *Windows* da *Microsoft*. Na época, ainda não havia sido lançada a versão 3.0, e a versão 2.03 não parecia satisfatória. Além disso, o *SDK* (*Software Development Kit*), que é o conjunto de programas e bibliotecas para auxílio na criação de programas para o *Windows*, é complexo o suficiente para que hoje existam vários outros fornecedores de software criando seus próprios '*SDKs*' para *Windows* ou para o próprio *SDK* da *Microsoft*.

Outro inconveniente com pacotes comerciais é a não disponibilidade do código (total ou parcialmente), que permita sua integração com sistemas em desenvolvimento. Além do que, qualquer que fosse o pacote escolhido, seria necessário o desenvolvimento de uma Interface Aplicação-Programa que o conectasse aos demais módulos do Sistema AGILE.

Por outro lado, dentre os objetivos do trabalho, que serão discutidos mais adiante, destacamos ainda a necessidade de um módulo para tratamento de dispositivos periféricos que permitisse uma maior facilidade no tratamento e expansão destes dispositivos.

A opção escolhida foi então, o desenvolvimento do Gerenciador de Apresentação de forma a atender as necessidades anteriormente citadas, embora seja limitada a sua capacidade de representação gráfica quando comparado a pacotes comerciais análogos. Esta decisão, apesar do maior tempo necessário e da limitação de recursos citada, nos daria maior flexibilidade e controle sobre as características do gerenciador, além de tornar desnecessário ao usuário pagar *royalties* ou ter que comprar um pacote *'Windows like'*.

Posteriormente, este Gerenciador de Apresentação, desenvolvido inicialmente para ser parte do Sistema de Prototipagem Rápida AGILE, atingiu um grau de independência suficiente para que sua utilização integrasse outros aplicativos. Detalhes sobre como usá-lo desta forma são encontrados no Capítulo 2 (Gerenciador de Apresentação).

Escopo do Trabalho

Como já foi mencionado anteriormente, a motivação principal deste trabalho foi prover o Sistema AGILE com recursos para tratamento das informações geradas na comunicação com o usuário. Já um dos principais propósitos do Sistema AGILE é a avaliação das interfaces geradas pelo mesmo e, para tanto, é necessária uma maior variedade de recursos gráficos que aumente o grau de confiabilidade desta avaliação. Esta gama de recursos foge ao escopo do presente trabalho, o qual, limitou-se a fornecer um conjunto mínimo de recursos aceitável para um sistema de prototipagem. Este conjunto mínimo, no entanto, pode vir a ser adequado para outras aplicações.

Se por um lado, o Gerenciador de Apresentação oferece um número reduzido de recursos gráficos, por outro, atende a um número maior de usuários, devido a simplicidade de sua utilização.

A partir da proposta original de especificação do Sistema AGILE [BRANCO 89], foram impostas limitações e definidos os objetivos do presente trabalho. Uma destas limitações foi a implementação do sistema para o ambiente MS-DOS do PC. Esta decisão foi tomada devido ao maior número de aplicativos que rodam em ambiente MS-DOS.

Como um dos objetivos, podemos citar a portabilidade do produto final para outros ambientes, que não os da linha IBM-PC. Esta, apesar de não estar disponível de imediato, é facilitada pela linguagem escolhida para implementação (linguagem C) e pelos cuidados tomados com a modularização e documentação.

Um outro objetivo consistiu em oferecer um maior número de recursos para o projeto de interfaces, para tanto, optou-se por desenvolver o sistema para trabalhar no ambiente gráfico do IBM-PC. Um editor foi previsto para ser utilizado como expansão dos recursos gráficos existentes para apoio aos projetos de telas e diálogos. Para tal, está disponível no código e nas estruturas de dados suporte para sua integração.

Um dos principais objetivos do trabalho, no entanto, foi facilitar a integração de dispositivos periféricos no projeto de suas aplicações. A forma adotada para tratar os dispositivos periféricos baseia-se em duas matrizes de funções (uma para dispositivos de entrada e outra para dispositivos de saída), cujos detalhes podem ser encontrados no Capítulo 4 - Módulo Gerenciador de Periféricos. As funções de tratamento dos dispositivos de vídeo, não foram introduzidos na matriz de saída, mas isto deve ser feito em versões futuras.

Finalmente, um fator crítico para sua aceitação, é a velocidade. Apesar de um sistema de entrada e saída em coordenadas normalizadas facilitar o acréscimo e tratamento de dispositivos periféricos, essa forma de coordenadas não foi utilizada no presente trabalho, devido aos prejuízos de velocidade que traria, já que operações de ponto flutuante seriam inevitavelmente usadas. A solução adotada foi trabalhar com as coordenadas dos dispositivos.

A presente dissertação, foi dividida em quatro capítulos. O primeiro capítulo trata de introduzir o leitor no ambiente AGILE - Sistema de Prototipagem Rápida Usuário-Computador, descrevendo sucintamente sua filosofia de funcionamento. Isto é necessário, para que se possa posicionar o Gerenciador de Apresentação dentro do contexto deste ambiente, para o qual foi originalmente desenvolvido.

No segundo capítulo, é dada uma visão geral do Gerenciador de Apresentação e descrita, em detalhes, a Interface Aplicação-Programa do mesmo. A Interface Aplicação-Programa é a principal porta de entrada do Gerenciador de Apresentação, existindo porém, outros meios de acesso as funções mais internas deste módulo.

Os terceiro e quarto capítulos, descrevem o Módulo Gerenciador de Telas e o Módulo Gerenciador de Periféricos, respectivamente. Estes módulos correspondem as camadas mais internas do Gerenciador de Apresentação. Como será mostrado, também podem ser utilizados de forma independente.

CAPÍTULO 1

O Ambiente AGILE

Neste Capítulo são ilustradas algumas das características do Sistema AGILE, motivo do desenvolvimento do Gerenciador de Apresentação. Maiores detalhes sobre o Sistema AGILE e a fundamentação teórica podem ser encontrados em [BRANCO 89] e [PROCÓPIO 92].

1.1. Filosofia de Funcionamento

A arquitetura do Sistema AGILE foi definida baseando-se no modelo de Seeheim [LÖWGREN 88][GREEN 85]. Este modelo representa um dos tipos de arquitetura de Sistemas Gerenciadores de Interface de Usuário e foi proposto na *workshop*, realizada em Seeheim em 1983. Entre o programa aplicativo e o usuário, este modelo interpõe três módulos, como pode ser visto na Figura 1.1.

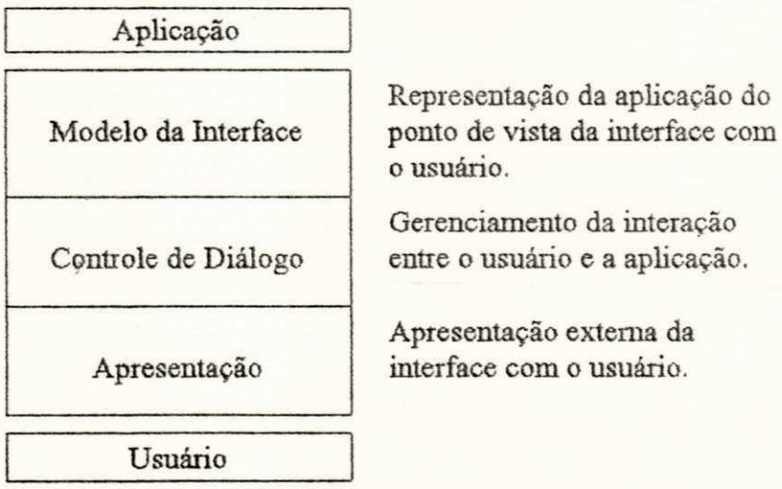


Figura 1.1 - Modelo de Seeheim.

1.1.1. Arquitetura AGILE

A Figura 1.2, ao lado, ilustra a estrutura geral do Sistema de Prototipagem Rápida AGILE.

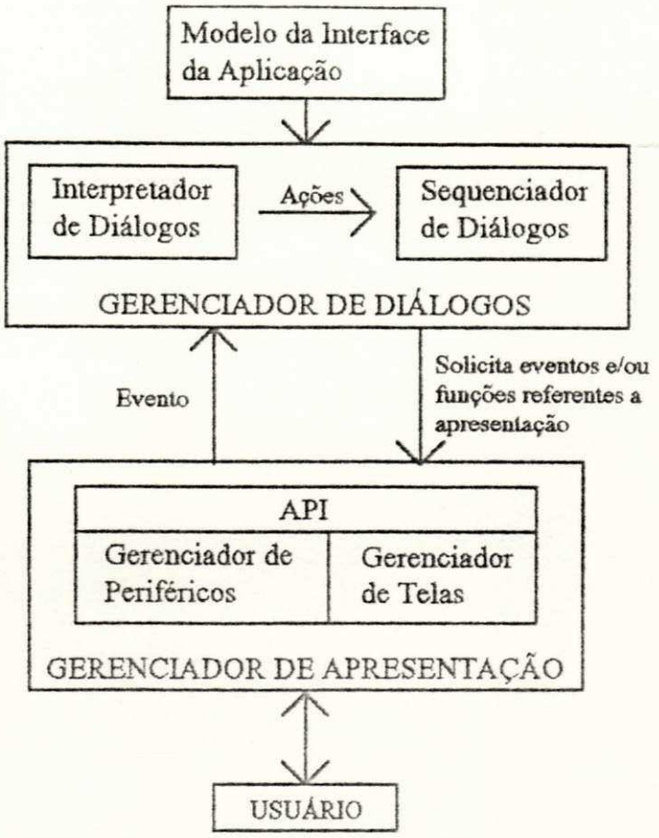


Figura 1.2 - Arquitetura do AGILE.

Como pode ser visto, os componentes do Modelo de Seeheim - Módulo de Apresentação e Módulo de Controle de Diálogo correspondem, respectivamente ao Gerenciador de Apresentação e ao Gerenciador de Diálogos. Observamos, ainda, que o bloco referente a aplicação não está presente. Isto se deve ao fato de que o Sistema AGILE é um sistema de prototipagem vazia, i.e., não incorpora as funções da aplicação,e não um sistema de gerenciamento de interfaces.

O funcionamento do AGILE é baseado na filosofia de eventos que ativam ações. Os eventos são gerados pelo usuário através dos periféricos de entrada, como por exemplo, o *click* de um *mouse*, uma tecla pressionada, etc.

Ações são rotinas que devem ser chamadas para executarem determinada função, que por sua vez foi definida como consequência de um evento. Esta associação entre o evento e a ação se dá durante a especificação do Modelo da Interface.

Ambientes de Interação

Estão disponíveis dois ambientes de interação para o projetista, um deles, o ambiente de prototipagem, é onde se define o protótipo de uma interface. No outro, o ambiente de simulação, se pode fazer uma simulação de execução da interface definida.

1.1.2. Modelo de uma Interface

No sistema AGILE, os dados necessários no momento da simulação são trazidos junto com os demais dados do protótipo simulado. Estes dados constituem o modelo de uma interface, e são um conjunto de especificações definidas através do editor de interfaces (de forma gráfica e interativa) presente no ambiente [PROCÓPIO 92], ou por qualquer outro editor.

1.1.3. Gerenciador de Diálogos

O Gerenciador de Diálogos do AGILE suporta quatro classes de diálogo, que representam os métodos de interação com o usuário: menu, formulário, comando, pergunta/resposta. Estas quatro classes de diálogo não são uma restrição a introdução de novas classes que venham a surgir, elas apenas refletem o estado atual do trabalho desenvolvido.

O acréscimo de novas classes de diálogo, poderá ser feito a partir da introdução das funções correspondentes no Módulo Gerenciador de Telas. Um exemplo de expansão seria o acréscimo da classe *Manipulação Direta* [SCHNEIDERMAN 87], que não foi abordada no trabalho por falta de recursos gráficos para tratamento de ícones.

O Gerenciador de Diálogos tem como função controlar todo o sequenciamento da interação, a partir do modelo da interface e dos eventos gerados pelos usuários. É formado por dois blocos: o Interpretador de Diálogos, que é responsável pela análise de eventos, e o Sequenciador de Diálogos, que tem a função de ativar ações decorrentes da análise de eventos.

1.1.4. Gerenciador de Apresentação

O Gerenciador de Apresentação foi projetado inicialmente para servir como uma interface inteligente entre o Gerenciador de Diálogos do AGILE [PROCÓPIO 92] e os periféricos. Suas funções podem, no entanto, ser utilizadas por quaisquer programas que necessitem realizar manipulação e tratamento de entrada e saída, de forma análoga a do AGILE. A idéia é criar um canal "imaginário (virtual)" de comunicação, que chamaremos diálogo, através do qual é processada toda a informação de entrada e saída.

1.2. Ferramentas de Prototipagem

Estão disponíveis, atualmente, no Sistema AGILE duas ferramentas de prototipagem:

- Um editor de interfaces, responsável pela definição do modelo da interface. Esta definição é feita de forma interativa.

- Um gerador de documentação, responsável pela geração de um relatório com a especificação do modelo definido com o editor de interfaces.

1.3. Ambiente de Instalação

A configuração de hardware mínima necessária para a instalação do AGILE é a seguinte:

- PC-XT, PC- AT, ou compatível. Um PC-AT é recomendado.
- 640K de memória RAM. Uma maior quantidade de memória é aconselhável, caso o projetista use uma placa EGA ou VGA.
- *Hard disk*
- Dispositivos periféricos de entrada e saída.

Caso o Sistema AGILE, venha a ser utilizado em um PC-XT, é provável que a velocidade seja um fator incômodo. Este problema é acentuado se o PC-XT estiver equipado com uma placa gráfica de maior resolução, e.g., EGA ou VGA.

O estado atual do Sistema AGILE permite o uso do teclado, *mouse*, e *joystick* como periféricos de entrada. Como dispositivos de saída, temos o monitor de vídeo e uma impressora.

CAPÍTULO 2

Gerenciador de Apresentação

Todo aplicativo que interage com um usuário, necessita de rotinas que manipulem a entrada e saída de informação. Deve ser capaz, portanto, de estabelecer uma comunicação formal entre o homem e a máquina. A forma como esta comunicação se realiza é, neste trabalho, chamada de *diálogo*. Hoje, observa-se que a tendência mundial é a utilização de interfaces gráficas para estabelecer esta comunicação.

Os vários aspectos que podem ser visualizados em um microcomputador MacIntosh nos dão uma definição de uma interface gráfica usuário-computador. Entre estes aspectos podemos destacar os que se seguem [HAYES 89]:

- Um dispositivo apontador, tipicamente um mouse;
- Menus na tela que podem aparecer e desaparecer sob controle do dispositivo apontador;
- Janelas que mostram graficamente o que o computador está fazendo;
- Ícones que representam arquivos, diretórios, etc;
- *Dialog boxes*, *buttons*, *sliders*, *check boxes*, e um conjunto de outros artifícios que deixam você dizer ao computador o que e como fazer.

Nem todas as características do MacIntosh estão presentes nas interfaces atuais. É natural que com o passar do tempo variações sejam introduzidas. Por exemplo, algumas não usam ícones. Em outras os ícones são opcionais ou disponíveis somente algumas vezes. Algumas necessitam do mouse, enquanto outras permitem você usar o teclado.

Apesar de existirem híbridos, a maioria das interfaces gráficas consiste de três componentes principais: um *windowing system* (sistema de gerenciamento de janelas), um *imaging model* (modelo de imageamento) e uma *API* (interface aplicação-programa). As Figuras 2.1a e 2.1b ilustram esta subdivisão para as interfaces gráficas mais conhecidas na atualidade, segundo Hayes [HAYES 89].

	NewWave	*Windows*	*Presentation Manager*	*CXI*	*Motif*	*DEC Windows*	*OpenDesktop*	*Open Look*
API	*		*User Interface Controls API*	*HP X Widgets*		*XUI*		*X View*
Windowing system	*Graphics Device Interface*		*Windows API*	*X Windows*				
Imaging model	*GDI output functions*		*Graphics API (GPI)*	*	*Not yet decided*	*Display PostScript*	*	*X11 / NeWS*
Operating system	*MS-DOS*		*OS/2*	*Unix*				
CPU	*Intel 8088/80286/80386/80486*							

Figura 2.1a - Interfaces Gráficas mais conhecidas.

Fonte: BYTE / July 89.

O *windowing system* (sistema de gerenciamento de janelas) é um conjunto de ferramentas de programação e comandos para construir as janelas, menus e *dialog boxes* que aparecem na tela. Ele controla como as janelas são criadas, dimensionadas e movidas na tela, e como o usuário move-se de uma janela para outra, entre suas funções.

	NextStep	*MacIntosh*	*Intuition*	*GEM*
API	*Kits*	*Mac Interface*	*Workbench*	*Application Environment Services*
Windowing system	*Window server*	*Window Manager*	*Intuition Library*	
Imaging model	*Display PostScript*	*Quick Draw*	*Graphics library*	*Virtual Device Interface*
Operating system	*Unix*	*Mac Os*	*Amiga OS*	*TOS*
CPU	*Motorola 68000/68010/68020/68030/68040*			

Figura 2.1b - Interfaces Gráficas mais conhecidas (continuação).

Fonte: BYTE / July 89.

Um exemplo de um *windowing system* é o *X Windows*, o qual não é uma interface gráfica completa. Ele é apenas um sistema de gerenciamento de janelas compartilhado por um grupo de diferentes interfaces.

O *imaging model* define como as fontes e gráficos são realmente criadas na tela. Provavelmente, *PostScript* é o mais conhecido *imaging model*, familiar das impressoras a laser. Outros exemplos são o *QuickDraw* do MacIntosh e o GPI (*Graphic Programming Interface*) do PM para o OS/2.

A Interface Aplicação-Programa (*API Application-Program Interface*) é o conjunto de chamadas a funções para uma determinada linguagem. É através delas que o programador especifica quais janelas, menus, *scroll bars* e ícones aparecerão na tela.

O Gerenciador de Apresentação fornece uma alternativa para facilitar a implementação da interface com o usuário, fornecendo suporte para as seguintes classes de diálogo:

- *menu*, uma lista de opções apresentadas na tela de vídeo, na qual o usuário deve fazer uma escolha. O resultado da escolha inicial é frequentemente, porém nem sempre, um outro menu de opções.

- *formulário*, um conjunto de informações que contém alguma relação entre si, apresentadas no vídeo sob a forma de campos. Cada campo é constituído por duas áreas distintas: o nome da informação desejada e o espaço destinado ao preenchimento por parte do usuário.

- *comando*, um *prompt* apresentado no vídeo, após o qual o usuário deve digitar um *string* a ser interpretado. Normalmente é utilizada a tecla *return* para indicar a finalização da digitação.

- *pergunta&resposta*, uma pergunta apresentada na tela, para qual é esperada uma resposta que deve ser digitada pelo usuário. Da mesma forma que na classe *comando*, normalmente é utilizada a tecla *return* para indicar a finalização da digitação.

- *mensagem*, uma forma de passar informações ao usuário. Esta informação é apresentada no vídeo e, posteriormente, retirada por fechamento da janela na qual se encontra. O uso da função *LeMens*, permite realizar um *timeout* ou verificar o acionamento de uma tecla pré-definida, antes da janela ser fechada.

2.1. Níveis de Acesso às Camadas Internas

Composto pelos Módulo Gerenciador de Telas, Módulo Gerenciador de Periféricos e pela Interface Aplicação-Programa (API), o Gerenciador de Apresentação permite o acesso em duas diferentes camadas, de acordo com as necessidades e o grau de conhecimento do projetista da interface (vide Figura 2.2).

A API é o principal meio de acesso ao Gerenciador de Apresentação, e constitui sua primeira camada. A segunda camada permite o acesso direto às funções do Módulo Gerenciador de Telas e Módulo Gerenciador de Periféricos. Estas funções encontram-se descritas nos Capítulos 3 e 4, respectivamente.

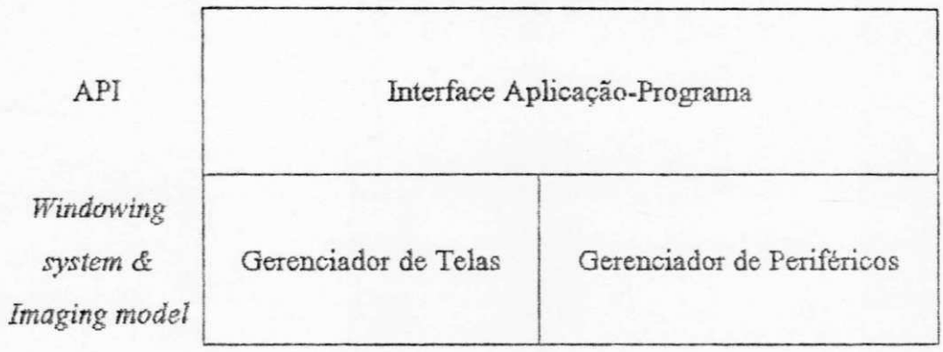


Figura 2.2 - Camadas do Gerenciador de Apresentação.

2.2. Descrição Funcional

A integração entre o Gerenciador de Apresentação e outros aplicativos deverá utilizar a mesma técnica que a sua integração com o AGILE. É permitido também, que se façam algumas chamadas "diretas" ao Módulo Gerenciador de Telas e ao Módulo Gerenciador de Periféricos.

As funções que fazem a interface com o AGILE ou programa aplicativo, subdividem-se internamente em chamadas às funções análogas presentes em cada classe de diálogo.

API - Interface Aplicação-Programa

A porta de entrada do Gerenciador de Apresentação é uma interface bastante simples, composta apenas por cinco funções para tratamento dos diálogos. Estas funções baseiam-se nas classes de diálogo existentes no Sistema AGILE, para o qual este gerenciador foi desenvolvido. O uso destas funções permite *abrir*, *ler*, *fechar*, *mover*, e *redimensionar* uma classe de diálogo.

Além das funções de tratamento dos diálogos existem aquelas responsáveis pelo preenchimento das estruturas de dados do Gerenciador de Apresentação. As informações relevantes, ou seja, todos os dados referentes às janelas e aos elementos das classes de diálogo podem ser obtidas através destas funções por um dos seguintes métodos:

- Os dados são lidos da unidade de disco para a estrutura de dados em memória do Gerenciador de Apresentação através da função *LeModelo*. O arquivo em disco que contém o modelo da interface pode ser criado manualmente ou via Sistema AGILE. Uma descrição mais detalhada sobre este arquivo pode ser encontrada em [PROCÓPIO 92].

- Os dados são criados um a um usando-se as funções disponíveis para tal. Estas funções fazem parte do Módulo Gerenciador de Telas.

Uma vez estando os dados presentes na memória, o projetista necessita apenas sequenciar as atividades desejadas. Os exemplos que se encontram na seção 2.4 (Sintaxe das Funções da API), podem ser usados para melhor visualizar o uso do Gerenciador de Apresentação.

2.3. Tratamento das Funções da API

As funções descritas a seguir utilizam dois parâmetros básicos. O primeiro, é uma variável inteira que identifica a classe de diálogo. O segundo, também uma variável inteira, identifica qual o elemento desta classe que deve ser aberto.

O primeiro parâmetro, chamado *classe*, pode assumir um dos seguintes valores:

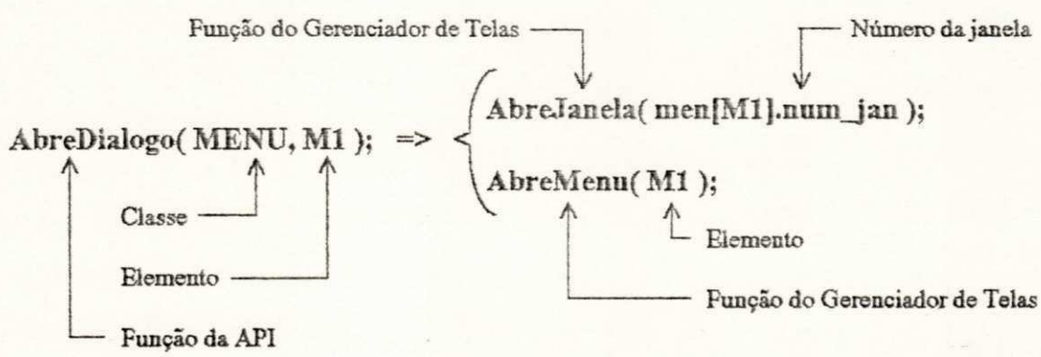
```
enum dialogos {
        MENU, FORMULARIO, COMANDO, PERG_RESP, MENSAGEM
};
```

O segundo parâmetro, chamado *elemento*, assume um valor entre 0 e o número máximo de elementos de sua classe. Este número máximo é função da quantidade de memória disponível em seu sistema. No Sistema AGILE, e.g., usou-se inicialmente o valor 120.

AbreDialogo

AbreDialogo é a chamada ao Gerenciador de Apresentação responsável pela inicialização de um elemento de uma classe de diálogo na tela. Uma chamada a esta função é transformada em uma chamada para abrir uma janela e uma chamada à função abrir específica à classe de diálogo desejada.

Por exemplo, para abrir um menu na tela é feita uma chamada ao Gerenciador de Apresentação, via a função *AbreDialogo*. Esta é subdividida em duas chamadas ao Módulo Gerenciador de Telas. A primeira *AbreJanela*, abre a janela necessária; a segunda *AbreMenu*, desenha o menu na tela de vídeo. Representando de forma gráfica,



AbreDialogo acessa as informações necessárias e chama as funções para proceder a preparação de tela para o uso da função *LeDialogo*. Convém observar que cada elemento de uma classe de diálogo está associado a uma única janela, a qual é aberta automaticamente, caso já não esteja,

através da chamada a função *AbreJanela*. Se a janela já se encontrar aberta, ela passará a ser a janela ativa, i.e, a janela que é vista em frente às outras.

O algoritmo da função *AbreDialogo*, pode ser visto no diagrama a seguir:

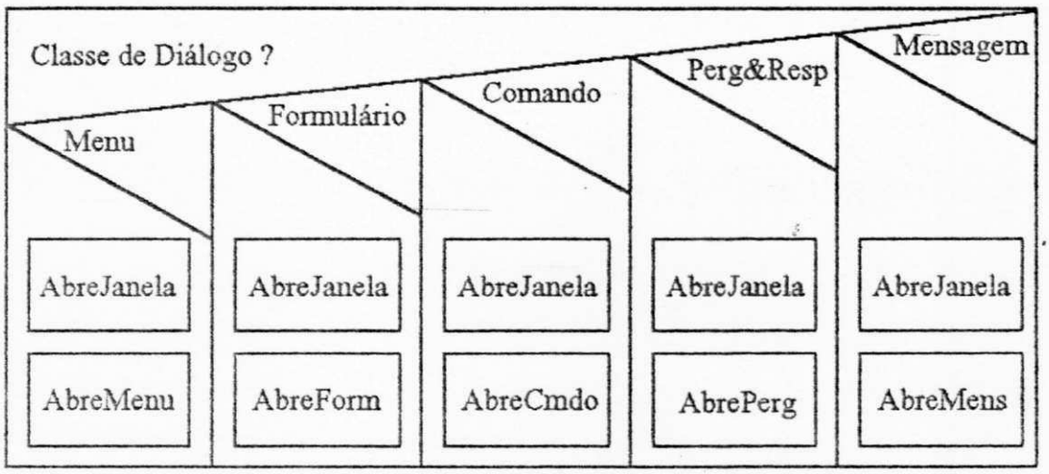
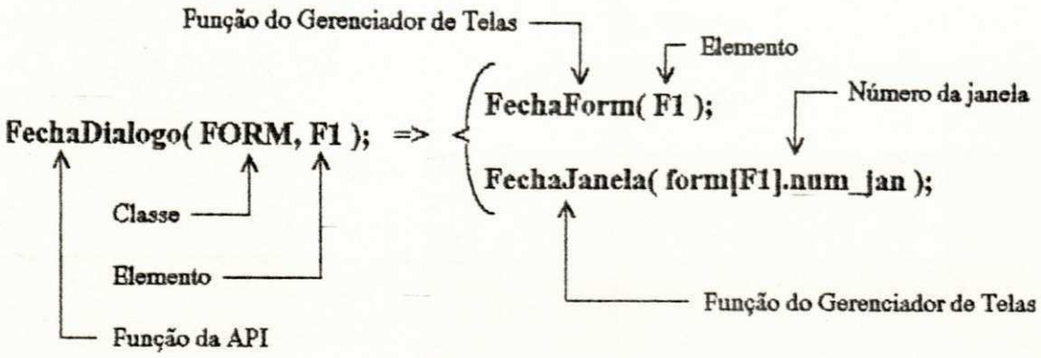


Figura 2.3 - Diagrama Nassi-Schneirdman da função *AbreDialogo*.

FechaDialogo

Para encerrar as atividades com um elemento de uma classe de diálogo, faz-se uma chamada ao Gerenciador de Apresentação através da função *FechaDialogo*, que fecha o canal aberto pela função *AbreDialogo*. Esta função não fecha a janela se a mesma estiver sendo usada por outro elemento de uma classe de diálogo. Neste caso, é dito que a janela tem mais de um *filho*.

Por exemplo, para fechar um formulário na tela é feita uma chamada ao Gerenciador de Apresentação, via a função *FechaDialogo*. Esta é subdividida em duas chamadas ao Módulo Gerenciador de Telas. A primeira *FechaForm*, encerra as atividades do formulário especificado e retira-o da tela; a segunda *FechaJanela*, fecha a janela associada ao formulário caso este seja seu último filho. Semelhantemente à função *AbreDialogo*, representamos graficamente a seguir a operação sendo realizada:



O algorítmo da função *FechaDialogo*, pode ser visto no diagrama a seguir:

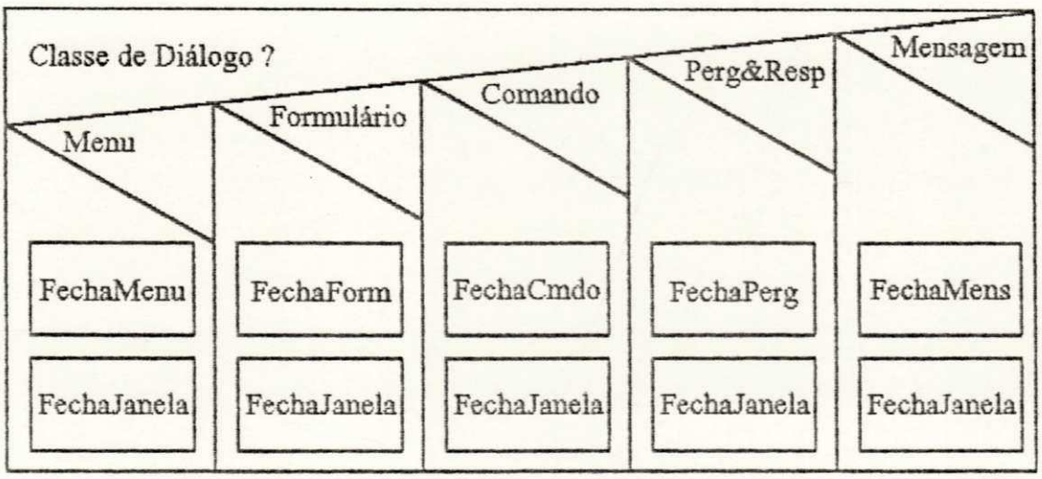


Figura 2.4 - Diagrama Nassi-Schneirdman da função *FechaDialogo*.

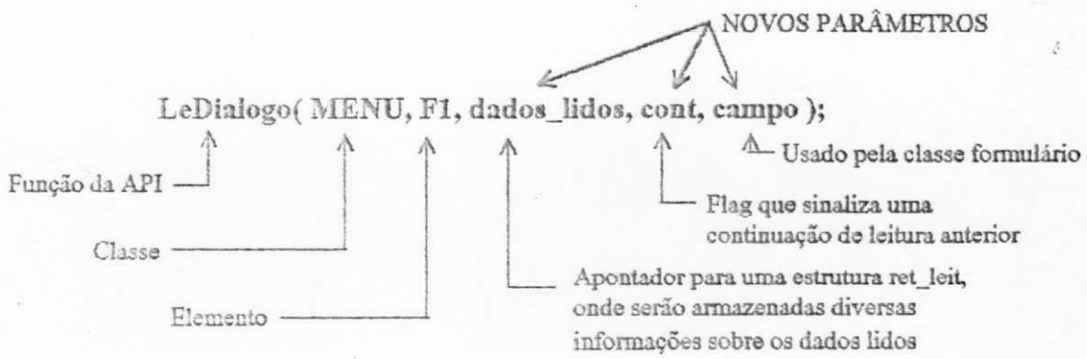
LeDialogo

Uma vez aberto um elemento de uma classe de diálogo, é possivel através de uma nova chamada ao Gerenciador de Apresentação, verificar os eventos gerados pelo usuário, analisá-los e caso necessário, repassá-los para o Gerenciador de Diálogos do AGILE.

Algumas das informações obtidas pela função de leitura de diálogo não precisam ser repassadas ao Gerenciador de Diálogos do AGILE ou programa aplicativo similar. Entre estas funções, podemos citar o deslocamento de uma barra sobre os ítens de um menu. Este deslocamento é

tratado na sua totalidade pelo Gerenciador de Apresentação e apenas após a seleção do ítem desejado o controle é repassado ao Gerente de Diálogos do AGILE. Esta autonomia, em algumas situações, da função *LeDialogo* possibilita uma melhor *performance* do Gerenciador de Apresentação.

Aos parâmetros inteiros *classe* e *elemento*, comuns a todas as funções da API, são acrescentados os três novos parâmetros: um apontador para uma estrutura *ret_leit* e duas variáveis inteiras.



A estrutura *ret_leit* é a forma padrão de retorno dos dados da função *LeDialogo*. Sua sintaxe é visualizada abaixo,

```
typedef struct ret_leit {
        int flag_ret;
        int tipo_ret;
        TECLAS ult_tecla;
        union {
                int ret_int;
                char ret_char;
                char *ret_str;
                TECLAS ret_tecla;
        } val_ret;
} RETLEITURA;
```

A Tabela 2.1, descreve os seus membros.

MEMBRO	DESCRIÇÃO
flag_ret	Informa se houve sucesso ou não na leitura do diálogo.
tipo_ret	Informa o tipo de dado que está sendo retornado. Este dado é usado em conjunto com a *union* val_ret.
ult_tecla	Última tecla digitada.
val_ret	Dado lido. O tipo de dado retornado é definido pela variável *tipo_ret*.

Tabela 2.1 - Descrição dos membros da estrutura *ret_leit*.

O algoritmo da função *LeDialogo*, pode ser visto no diagrama a seguir:

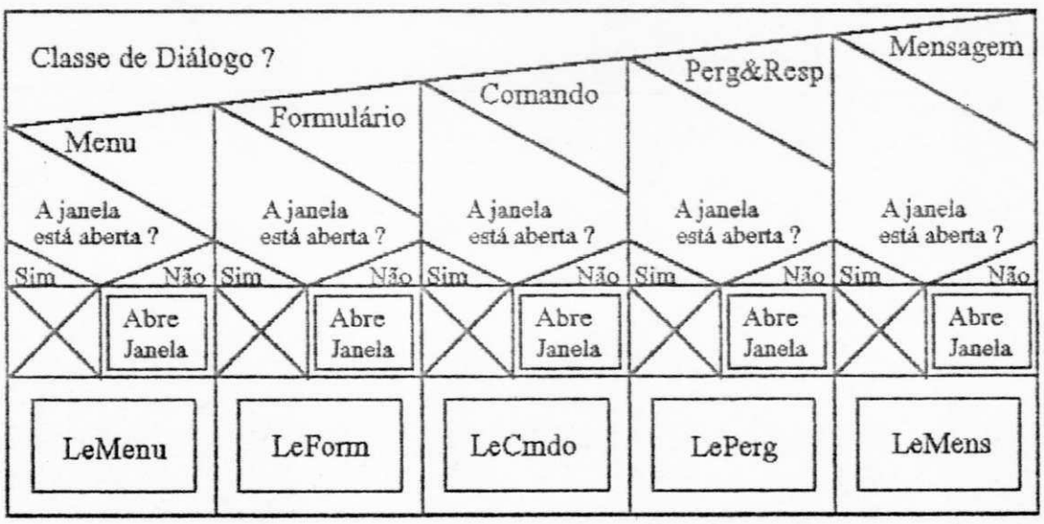


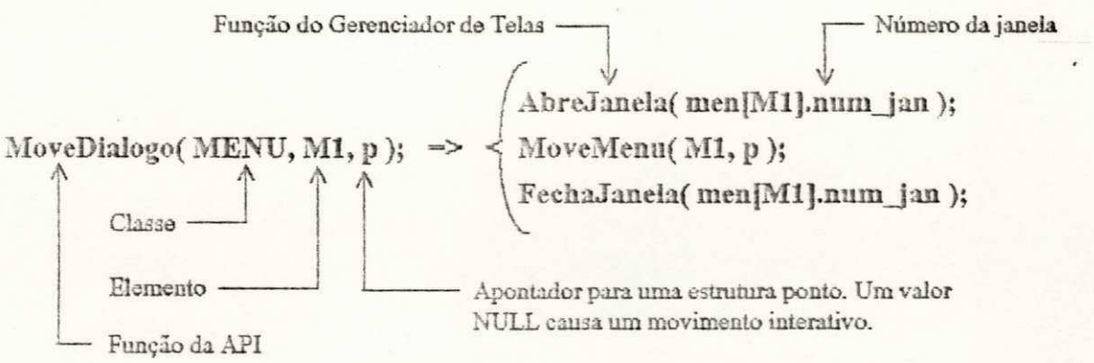
Figura 2.5 - Diagrama Nassi-Schneirdman da função *LeDialogo*.

MoveDialogo

A chamada ao Gerenciador de Apresentação que causa o movimento de um elemento de uma classe de diálogo, pode ser feita de duas formas: direta ou interativa. Na forma direta, um novo

par de coordenadas é fornecido, sob a forma de parâmetro, e este par passa a representar o canto superior esquerdo da caixa que limita o diálogo. Na forma interativa, o usuário usa as teclas de navegação, de qualquer dispositivo de entrada, para posicionar o diálogo no local desejado dentro da janela. Observar que em ambas as operações os limites da tela não são ultrapassados.

A API do Gerenciador de Apresentação transforma um pedido para mover um elemento de uma classe de diálogo, da seguinte forma:



O algorítmo implementado pode ser visto na Figura 2.6, a seguir:

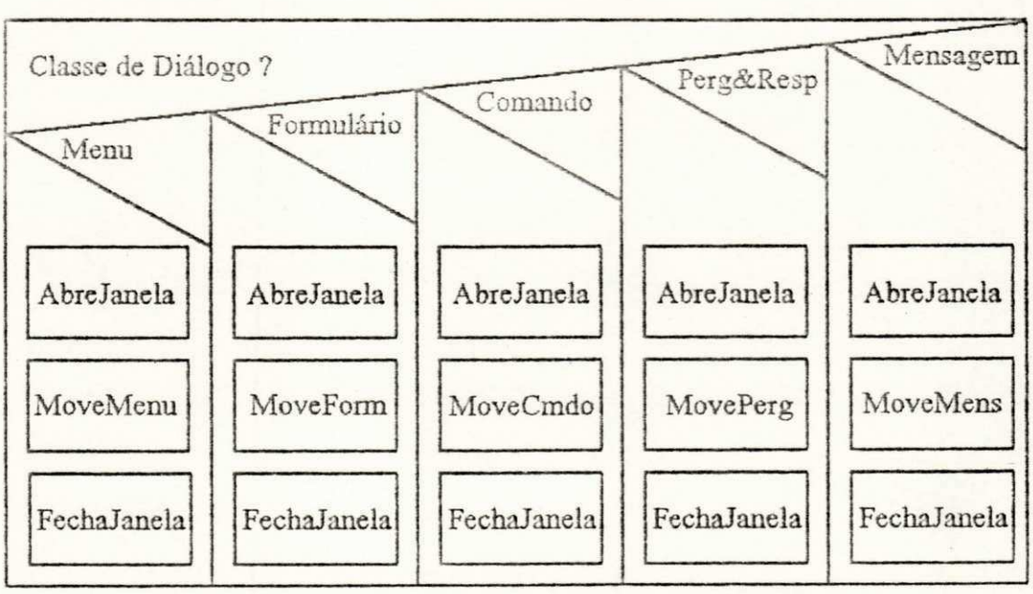
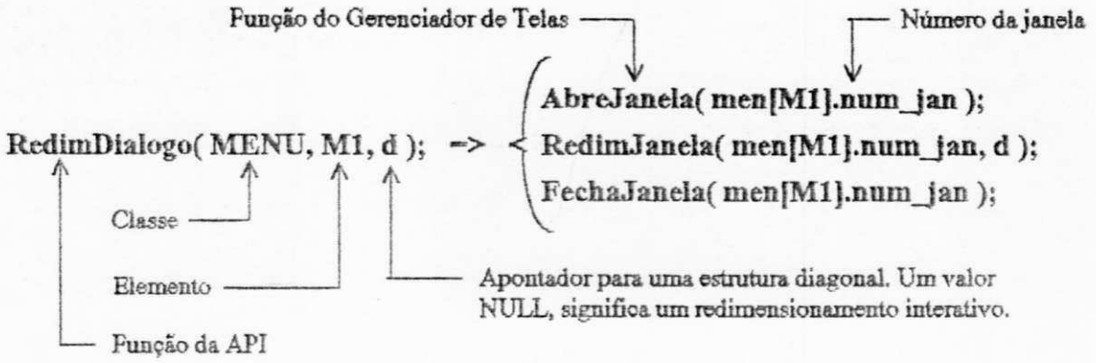


Figura 2.6 - Diagrama Nassi-Schneirdman da função *MoveDialogo*.

RedimDialogo

Esta função, na realidade, redimensiona a janela em que se encontra o diálogo. De forma análoga a função *MoveDialogo*, existem duas formas de usá-la: direta ou interativa. Na forma direta, uma nova diagonal (que define um retângulo), é fornecida e esta passa a ser a nova dimensão da janela. Na forma interativa, o usuário usa as teclas de navegação, de qualquer dispositivo de entrada, para redimensionar a janela. Observar que em ambas as operações os limites da tela não são ultrapassados. A API do Gerenciador de Apresentação transforma um pedido para redimensionar um elemento de uma classe de diálogo, da seguinte forma:



O algorítmo implementado é mostrado na Figura 2.7, a seguir:

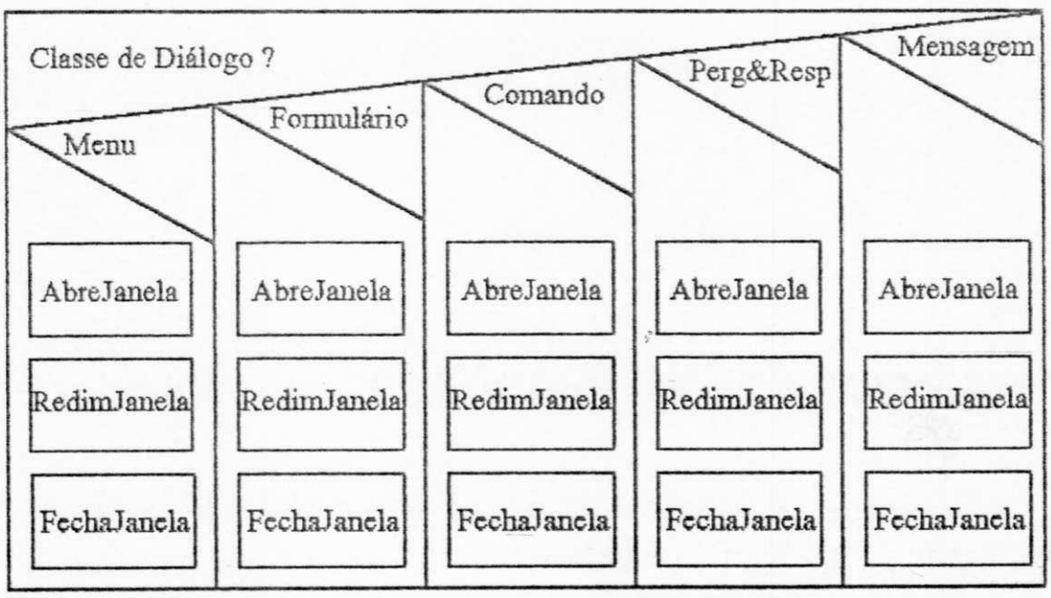


Figura 2.7 - Diagrama Nassi-Schneirdman da função *RedimDialogo*.

2.4. Sintaxe das Funções da API

• IniciaGA

Função: Inicializa o Gerenciador de Apresentação.

Sintaxe: *#include "gapres.h"*

int IniciaGA(int gdriver, int gmode);

Protótipo: *gapres.h*

Comentários: Esta função é responsável pela inicialização do ambiente necessário para as demais funções do Gerenciador de Apresentação. Entre outras coisas, inicializa o modo gráfico para a placa (*gdriver*) e resolução (*gmode*) especificadas. Os valores permitidos para estas variáveis são mostrados na tabela abaixo.

gdriver	*gmode*	Resolução	Cores
CGA	CGAC0	320 x 200	4
	CGAC1	320 x 200	4
	CGAC2	320 x 200	4
	CGAC3	320 x 200	4
	CGAHI	640 x 200	2
EGA	EGALO	640 x 200	16
	EGAHI	640 x 350	16
HERC	HERCMONO	720 x 348	2
VGA	VGALO	640 x 200	16
	VGAMED	640 x 350	16
	VGAHI	640 x 480	16

Obs.: A função *LeModelo* pode alterar o modo gráfico a ser utilizado.

Retorna: Em caso de sucesso na inicialização, retorna o valor lógico VERDADE; caso contrário, retorna FALSO.

Exemplo:
```
#include <dos.h>          /* apenas pelo o sleep */
#include "gapres.h"       /* definicoes e declaracoes globais do GA */

main()
{
        IniciaGA( CGA, CGAHI );   /* inicializa o GA */
        LeModelo( "teste" );      /* le o modelo do arquivo em disco */
        AbreDialogo( MENU, 0 );   /* abre o elemento 0 da classe MENU */
        sleep( 5 );               /* aguarda 5 segundos */
        FechaDialogo( MENU, 0 );  /* fecha o dialogo */
        TerminaGA();              /* encerra as atividades com o GA */
}
```

• **TerminaGA**

Função: Encerra as atividades com o Gerenciador de Apresentação.

Sintaxe:
```
#include "gapres.h"
void TerminaGA();
```

Protótipo: *gapres.h*

Comentários: Ao encerrar as atividades, retorna ao prompt do DOS no modo texto.

Retorna: Nada.

Exemplo: *Vide o exemplo da função IniciaGA.*

• **LeModelo**

Função: Lê do arquivo em disco especificado, o modelo da interface a ser usada.

Sintaxe:
```
#include "gapres.h"
int LeModelo( char *arquivo );
```

Protótipo: *gapres.h*

Comentários: Após a leitura, é verificado se as condições gráficas necessárias ao modelo são satisfeitas pelo ambiente atual.

Retorna: Um valor lógico VERDADE é retornado caso as condições necessárias para rodar o modelo estejam satisfeitas. Em caso contrário, um valor FALSO é retornado.

Exemplo: *Vide o exemplo da função IniciaGA*

- **AbreDialogo**

Função: Inicia um elemento de uma classe de diálogo.

Sintaxe: *#include "gapres.h"*

int AbreDialogo(int classe, int elemento);

Protótipo: *gapres.h*

Comentários: A variável global *Erro* (unsigned long), contém o código de erro.

Retorna: Em caso de sucesso na abertura do diálogo, retorna o valor lógico VERDADE; caso contrário, retorna FALSO.

Exemplo: *Vide exemplo da função IniciaGA.*

- **FechaDialogo**

Função: Encerra o elemento da classe de diálogo especificada.

Sintaxe: *#include "gapres.h"*

int FechaDialogo(int classe, int elemento);

Protótipo: *gapres.h*

Comentários: A variável global *Erro* (unsigned long), contém o código de erro.

Retorna: Em caso de sucesso no fechamento do diálogo, retorna o valor lógico VERDADE; caso contrário, retorna FALSO.

Exemplo: *Vide exemplo da função IniciaGA.*

• **LeDialogo**

Função: Lê um elemento da classe de diálogo

Sintaxe: *#include "gapres.h"*

*void LeDialogo(int classe, int elemento, struct ret_leit *p, int cont_dial,*
int campo);

Protótipo: *gapres.h*

Comentários: Nesta função, três novos parâmetros foram introduzidos:

- A variável inteira *campo* é usada apenas pela classe *formulario*.
- A variável *cont_dial*, também inteira, é um flag que indica se é para iniciar ou continuar a leitura do elemento.
- O apontador para uma estrutura *ret_leit* é o meio pelo qual os parâmetros lidos são retornados.

A variável global *Erro* (unsigned long) contém o código de erro.

Retorna: Nada.

Exemplo:

```
#include "gapres.h"          /* definicoes e declaracoes globais do GA */

main()
{
    struct ret_leit dados;

    IniciaGA( CGA, CGAHI );     /* inicializa o GA */
    LeModelo( "teste" );        /* le o modelo do disco */
    AbreDialogo( MENU, 1 );     /* abre o elemento 1 da classe MENU */
    /* le a opção do menu, permanecendo em loop enquanto não for */
    /* o quarto item  o escolhido */
    do
        LeDialogo( MENU,        /* classe de dialogo menu */
            1,                  /* elemento desta classe */
```

```
                &dados,         /* apontador para a variavel que contera'
                                   os dados lidos */
                0 );            /* leitura inicial */
        while( dados.val_ret.ret_int != 3 );
        FechaDialogo( MENU, 1 ); /* fecha o dialogo */
        TerminaGA();             /* encerra as atividades com o GA */
}
```

• MoveDialogo

Função: Move um elemento de uma classe de diálogo.

Sintaxe: *#include "gapres.h"*

*void MoveDialogo(int classe, int elemento, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: Desloca o elemento de uma classe de diálogo de forma interativa pela tela, caso o apontador *p* tenha valor NULL. Se o valor de *p* for diferente de NULL, a estrutura apontada especifica a nova coordenada do canto superior esquerdo do diálogo.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>          /* define o NULL */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
        IniciaGA( CGA, CGAHI );  /* inicializa o GA */
        LeModelo( "teste" );     /* le o modelo do disco */
        AbreDialogo( MENU, 1 );  /* abre o elemento 1 da classe MENU */
        MoveDialogo( MENU,       /* classe de dialogo menu */
                1,               /* elemento desta classe */
                NULL );          /*move de forma interativa */
        FechaDialogo( MENU, 1 ); /* fecha o dialogo */
```

```
        TerminaGA();        /* encerra as atividades do GA */
}
```

• **RedimDialogo**

Função: Redimensiona um elemento de uma classe de diálogo.

Sintaxe: *#include "gapres.h"*

*void RedimDialogo(int classe, int elemento, struct diagonal *d);*

Protótipo: *gapres.h*

Comentários: Redimensiona a janela na qual se encontra o diálogo. Se *d* igual a NULL, o redimensionamento é interativo. Caso contrário, *d* aponta para uma estrutura *diagonal* que define as novas dimensões da janela.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>          /* define o NULL */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
    IniciaGA( CGA, CGAHI );     /* inicializa o GA */
    LeModelo( "teste" );        /* le o modelo do disco */
    AbreDialogo( MENU, 1 );     /* abre o elemento 1 da classe MENU */
    RedimDialogo( MENU,         /* classe de dialogo */
        1,                      /* elemento desta classe */
        NULL );                 /*move de forma interativa */
    FechaDialogo( MENU, 1 );    /* fecha o dialogo */
    TerminaGA();                /* encerra as atividades com o GA */
}
```

CAPÍTULO 3

Módulo Gerenciador de Telas

No contexto do Gerenciador de Apresentação desenvolvido, o vídeo, principal meio de passagem de informações do aplicativo para o usuário, é tratado como um periférico de saída especial. A quantidade de informação e seu carater dinâmico, fizeram necessário o Módulo Gerenciador de Telas.

3.1. Descrição Funcional

Este módulo é responsável pela manipulação e tratamento de vídeo, ajudando o usuário a monitorar e controlar os diferentes contextos da interação, separando-os fisicamente em diferentes áreas da tela de vídeo. Este gerenciamento é feito através de uma interface composta por dois grupos de funções.

O primeiro grupo permite *criar*, *destruir*, *mover* e *redimensionar* janelas, onde as atividades de interação subsequentes passarão a se desenrolar. O segundo grupo permite *criar*, *destruir*, *mover*, *redimensionar* e ler elementos das classes de diálogo.

Algo importante que devemos saber sobre as telas de vídeo é a variedade de modos em que elas podem trabalhar. No topo da hierarquia dos modos de operação estão o modo texto e o modo gráfico. No modo texto, tudo o que é mostrado na tela de vídeo vem de um gerador de caracteres

fixos. Estes caracteres no IBM-PC, em número de 256, estão gravados em um *chip* de memória ROM e não podem ser alterados sem a substituição da mesma.

Placas de vídeo mais recentes, como a VGA, permitem realizar a cópia desta ROM para uma RAM, possibilitando a alteração da forma dos caracteres mostrados no vídeo. Isto torna possível a redefinição de caracteres *(code pages)* para se acomodarem aos existentes em países estrangeiros.

Via de regra, a tela do PC no modo texto é dividida em 80 colunas por 25 linhas.

O modo gráfico trata a tela como uma série de pontos conhecidos como *pixels*, do inglês *picture elements*. A diferença entres os vários modos gráficos fica por conta das dimensões da tela, i.e., do número de pontos que podem ser representados. Qualquer coisa que possa ser representada no modo texto pode também o ser no modo gráfico a partir da montagem dos *pixels*.

A grande desvantagem do modo gráfico sobre o modo texto é a velocidade. Devido a forma como são tratados os caracteres e como são tratados os pixels, estes levam muito mais tempo e mais memória para serem processados. Estas desvantagens estão a cada dia diminuindo devido a grande capacidade de processamento dos microcomputadores atuais.

Nosso trabalho foi implementado baseado nos modos gráficos de um IBM-PC.

Cursor Gráfico

No modo texto de display de vídeo o hardware é responsável pelo *blinking* do cursor. Esta facilidade não está disponível no modo gráfico. Portanto, foi necessário criar uma função responsável pelo aparecimento e *blinking* do cursor gráfico. O artifício utilizado foi interceptar a interrupção de relógio para realizar o *blinking* do cursor.

O PC tem um temporizador que provoca uma interrupção (interrupção 0x08) 18,2065 vezes por segundo, o que dá um intervalo de quase 55 ms entre interrupções. O vetor original que aponta

para rotina de tratamento desta interrupção é armazenado. Um novo vetor é colocado, apontando para a nossa rotina de *blinking* do cursor. Uma vez realizado este tratamento, a rotina original é chamada.

A posição do cursor é tratada e mantida separadamente para cada janela. A posição é especificada por um par de números, x e y, com origem no canto superior esquerdo da janela. A função *JanPosCursor*, posiciona o cursor na coordenada especificada, limitando-o a área útil da janela.

Padrões de Vídeo

Os microcomputadores IBM-PC, XT, AT e compatíveis dispõem de uma grande variedade de placas para realizarem a saída de vídeo. A Tabela 3.1 ilustra as placas que são atualmente consideradas padrões.

PLACA	DESCRIÇÃO
CGA	*Color Graphics Adapter*, dispõe de resolução máxima de 640 *pixels* (*picture elements*) na horizontal, 200 *pixels* na vertical e 2 cores.
Hercules	Dispõe de resolução máxima de 720 *pixels* na horizontal, 348 *pixels* na vertical e 2 cores.
EGA	*Enhanced Graphics Adapter*, dispõe de resolução máxima de 640 *pixels* na horizontal, 350 na vertical e 16 cores de um *palette* de 64 cores.
VGA	*Video Graphics Array*, dispõe de resolução máxima de 640 *pixels* na horizontal, 480 na vertical e 16 cores de um *palette* de 256 cores. Pode ainda emular todas as placas anteriores.

Tabela 3.1 - Padrões Gráficos no IBM-PC

PLACA	DESCRIÇÃO
SuperVGA	Apesar de não poder ser considerada ainda um padrão, já que diversos fabricantes fazem uso de formas diferentes para acessar as informações nesta placa, existe um movimento para tal. Dispõe de resolução máxima de 800 a 1024 *pixels* na horizontal, 600 a 768 *pixels* na vertical e 16 a 256 cores. É capaz de emular todas as placas anteriores.

Tabela 3.1 - Padrões Gráficos no IBM-PC (continuação)

No caso específico do Sistema AGILE, o ideal é que este rode em um ambiente que disponha de uma placa de vídeo que possa emular as demais, e.g., uma VGA ou SuperVGA, caso contrário, só será possível a simulação de protótipos cujo ambiente alvo tenha placa de vídeo com resolução igual ou inferior àquela onde o protótipo foi gerado.

Durante a fase de prototipagem o usuário pode especificar que sua interface irá rodar num determinado padrão de vídeo. A fase de simulação se encarrega de realizar as críticas, adaptações e correções necessárias a esse padrão.

Para garantir uma maior portabilidade do sistema, existe um modelo da interface do AGILE para cada placa gráfica, excetuando-se a placa SuperVGA, devido aos problemas ainda existentes de compatibilidade de registros internos e chamadas ao BIOS da placa.

Layout de uma janela

As janelas definidas pelo Módulo Gerenciador de Telas tem formato retangular e são compostas por três áreas denominadas: área de título, área útil e área de rodapé (vide Figura 3.1).

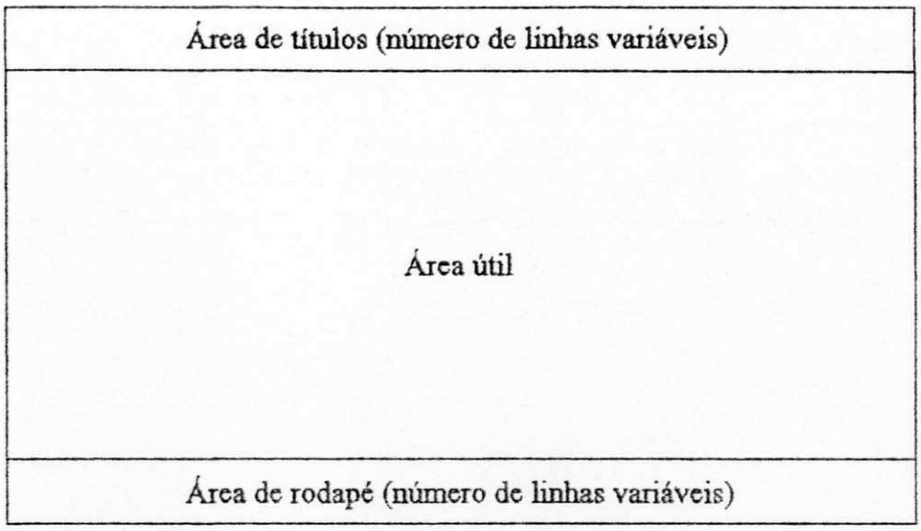


Figura 3.1 - Layout de uma janela.

O número de linhas da área de título e da área de rodapé é variável, tendo cada uma destas áreas atributos próprios de cor, fonte e tamanho de letras. A área útil tem como atributos cor e padrão de preenchimento para o *background*. A borda é definida em função de sua cor.

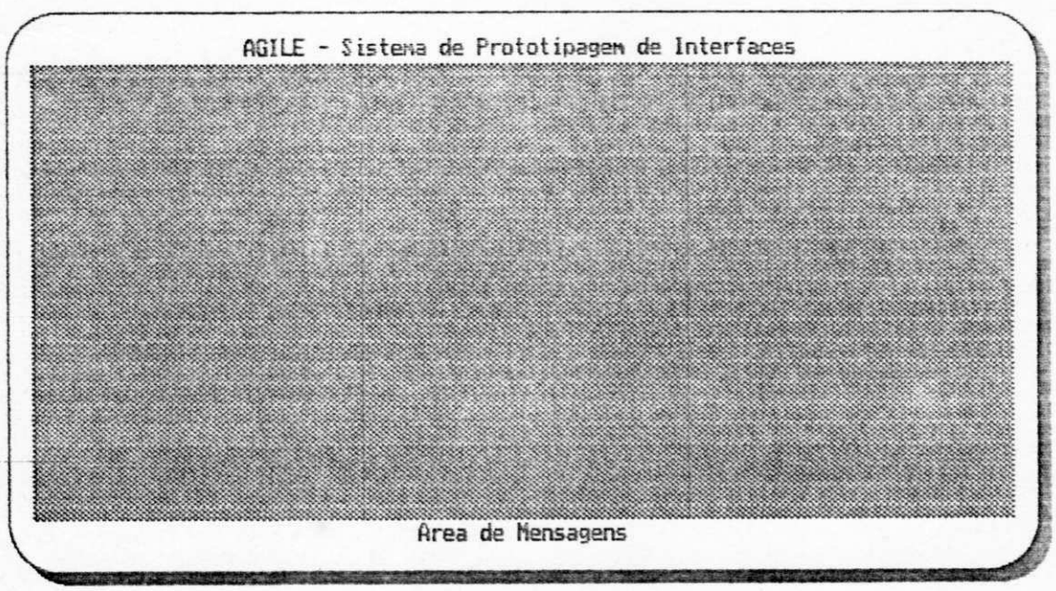


Figura 3.2 - Exemplo de uma janela.

3.2. Tratamento de Janelas

Existe basicamente duas alternativas sobre como as janelas podem ser posicionadas em relação umas as outras na tela: com ou sem sobreposição. A primeira alternativa que permite a sobreposição de janelas será chamada *sobrepostas*. A outra alternativa que consiste em colocá-las lado a lado é chamada *mosaicos*.

Um estudo [MYERS 88] revelou que apesar das pessoas consultadas dizerem preferir os sistemas que permitem sobreposição de janelas, estas pessoas gastam menos tempo realizando as operações de gerenciamento de janelas em um sistema de mosaico do que em um sistema sopreposto. O tempo total para completar uma tarefa, no entanto, é inconclusivo. O tamanho da tela também afeta a preferência entre os dois sistemas.

Podemos considerar que a melhor alternativa resume-se a uma questão pessoal. No entanto, é óbvio que um sistema que permite sobreposição também pode ser usado como um sistema de mosaico. A grande maioria dos sistemas comerciais hoje em uso permite a sobreposição. Vale ressaltar que a versão original do *Microsoft Windows* usava mosaicos.

O nosso Módulo Gerenciador de Telas suporta janelas soprepostas, cabendo ao projetista ao utilizá-lo fazer o posicionamento das janelas como o desejar, inclusive no modo mosaico.

Todas as tarefas associadas a manipulação de janelas, tais como preservar o *background*, lembrar a posição do cursor, confinar os dados dentro da janela, etc. são controladas automaticamente pelo Módulo Gerenciador de Telas.

A seguir passamos a descrever algumas convenções no tratamento das janelas:

- Cada elemento de uma classe de diálogo tem associado a si uma única janela. No entanto, uma janela pode ser usada por vários elementos de uma mesma classe ou que pertençam a várias classes.

- Uma janela é aberta pela primeira vez por um elemento de uma classe de diálogo, é dito então que a janela tem um filho. A necessidade de mais elementos desta ou de outras classes de diálogo usarem a mesma janela já aberta implica no incremento do número de seus filhos. Uma janela também pode ser aberta usando-se diretamente as funções para *abrir*, *fechar*, *mover* e *redimensionar* janelas. Este procedimento porém não é recomendado por aumentar a complexidade do código da aplicação. Esta complexidade implica em obrigar o usuário a ter um maior controle sobre o sequenciamento de suas chamadas ao Gerenciador de Telas.

- O pedido por parte de um elemento de uma classe de diálogo para fechar uma janela só é atendido caso este elemento seja seu último filho, caso contrário apenas o número de filhos é decrementado.

As próximas seções descrevem os recursos disponíveis para tratamento de janelas.

3.2.1. Criando Janelas

Tal como para arquivos em disco, existe uma função para abrir uma janela. A função *AbreJanela* permite criar janelas e para tanto ela obtém os dados necessários de uma estrutura de dados que deve ser previamente preenchida.

A janela tem como dados principais, as coordenadas da diagonal do retângulo que a define, a cor de fundo, padrão de preenchimento, dados sobre o texto e cor dos títulos e rodapés.

Todas estas informações que definem a janela especificada, estão disponíveis no arquivo de protótipo e são carregadas para a memória via a função *LeModelo* da API (vide Capítulo 2).

A seguir, algumas das estruturas de dados utilizadas pelo Módulo Gerenciador de Telas são apresentadas em sua sintaxe original, i.e., em Linguagem C.

viewporttype

A estrutura de dados *viewporttype* define as dimensões da janela e se os dados endereçados a esta janela devem sofrer ou não recorte.

```
typedef struct viewporttype {
        int left;
        int top;
        int right;
        int bottom;
        int clip;
} vport_t;
```

Os membros da estrutura *viewport* são descritos na Tabela 3.2, a seguir:

MEMBRO	DESCRIÇÃO
left	Especifica a coordenada *x* esquerda da área total da janela.
top	Especifica a coordenada *y* superior da área total da janela.
right	Especifica a coordenada *x* direita da área total da janela.
bottom	Especifica a coordenada *y* inferior da área total da janela.
clip	Especifica se deve ou não haver recorte na janela.

Tabela 3.2 - Descrição dos membros da estrutura *viewport*.

texto

A estrutura de dados *texto* define o modelo utilizado para encadear as linhas de texto das áreas de título e rodapé.

```
typedef struct texto {
        char *txt;
        int font;
        int direction;
        int charsize;
        int cor;
        struct text *prox;
} texto_t;
```

Os membros da estrutura *texto* são descritos na Tabela 3.3., a seguir:

MEMBRO	DESCRIÇÃO
txt	Cadeia de caracteres que forma o texto.
font	Define a fonte a ser usada. Os seguintes valores são aceitos: 0 Default_Font 1 Triplex_Font 2 Small_Font 3 Sans_Serif_Font 4 Gothic_Font
direction	Define a direção de escrita. Use o valor 0 (zero) para o texto ser escrito na horizontal e 1 (um) para vertical.
charsize	Define o tamanho do caractere.

Tabela 3.3 - Descrição dos membros da estrutura *texto*.

MEMBRO	DESCRIÇÃO
cor	Define a cor do caractere. O número que define a cor dos caracteres do texto depende do tipo de placa gráfica que está sendo usada. Se for especificado uma cor não disponível para a placa em uso, a mesma é visualizada como a cor BRANCO. Valores de cor para uso genérico: 0 PRETO 1 AZUL 2 VERDE 3 CIANO 4 VERMELHO 5 MAGENTA 6 MARROM 7 CINZA_CLARO 8 CINZA_ESCURO 9 AZUL_CLARO 10 VERDE_CLARO 11 CIANO_CLARO 12 VERMELHO_CLARO 13 MAGENTA_CLARO 14 AMARELO 15 BRANCO
prox	Apontador para próxima *string*

Tabela 3.3 - Descrição dos membros da estrutura *texto* (continuação).

janela

A estrutura de dados *janela* define os atributos de uma janela. Seus campos incluem o seguinte:

```
typedef struct janela {
    vport_t vp;
    char *bbuf;
    char *fbuf;
    int trava;
    int num_filhos;
    int borda;
    int jlt;
    int jtp;
    int jrt;
```

```
    int jbm;
    int curx;
    int cury;
    int jstdfill;
    int jcorfill;
    texto_t *titulo;
    int tstdfill;
    int tcorfill;
    int taltura;
    int tlargura;
    texto_t *msg;
    int mstdfill;
    int mcorfill;
    int maltura;
    int mlargura;
    struct janela *prev;
    struct janela *next;
} janela_t;
```

Os membros da estrutura são descritos na Tabela 3.4., abaixo:

MEMBRO	DESCRIÇÃO
vp	Especifica a área total ocupada pela janela.
bbuf	Apontador para região de memória ou disco onde será salvo a área de vídeo encoberta pela janela.
fbuf	Apontador para a região de memória ou disco onde será salva o conteúdo da janela.
trava	Especifica se a janela pode ser fechada (trava = FALSO) ou não (trava = VERDADE).

Tabela 3.4 - Descrição dos membros da estrutura *janela*.

MEMBRO	DESCRIÇÃO
num_filhos	Guarda a informação do número de elementos usando a janela.
borda	Especifica a presença de um quadro circundando a janela.
jlt	Especifica a coordenada *x* esquerdo da área útil da janela.
jtp	Especifica a coordenada *y* superior da área útil da janela.
jrt	Especifica a coordenada *x* direito da área útil da janela.
jbm	Especifica a coordenada *y* inferior da área útil da janela.
curx	Guarda a informação da coordenada *x* atual do cursor.
cury	Guarda a informação da coordenada *y* atual do cursor.
jstdfill	Especifica o padrão de preenchimento da área de útil, de acordo com os valores abaixo: 0 EMPTY_FILL 1 SOLID_FILL 2 LINE_FILL 3 LTSLASH_FILL 4 SLASH_FILL 5 BKSLASH_FILL 6 LTBKSLASH_FILL 7 HATCH_FILL 8 XHATCH_FILL 9 INTERLEAVE_FILL 10 WIDE_DOT_FILL 11 CLOSE_DOT_FILL
jcorfill	Define a cor de preenchimento da área de trabalho.
titulo	Apontador para a estrutura com a lista de *strings* do título.
tstdfill	Especifica o padrão de preenchimento da área do título.
tcorfill	Define a cor de preenchimento da área do título.
taltura	Altura em *pixels* do título.
tlargura	Largura em *pixels* do título.
msg	Apontador para a estrutura com a lista de *strings* do rodapé.
mstdfill	Especifica o padrão de preenchimento da área de rodapé.
mcorfill	Define a cor de preenchimento da área de rodapé.

Tabela 3.4 - Descrição dos membros da estrutura *janela* (continuação).

MEMBRO	DESCRIÇÃO
maltura	Altura em *pixels* do rodapé.
mlargura	Largura em *pixels* do rodapé.
prev	Apontador para a janela anterior.
next	Apontador para a próxima janela.

Tabela 3.4 - Descrição dos membros da estrutura *janela* (continuação).

3.2.2. Empilhando as Janelas

O Módulo de Gerenciador de Telas usa o conceito de *janela ativa*. Entre todas as janelas, apenas uma, a janela ativa, é o alvo das atividades naquele instante. Quando uma janela é aberta (ou reaberta) ela se torna a janela ativa, permanecendo assim até que seja fechada ou uma nova janela seja aberta.

Para manter a ordem no tratamento das janelas, uma lista duplamente encadeada, foi usada para implementar uma estrutura semelhante a uma pilha, conforme nos mostra a Figura 3.3.

A variável *topwin* aponta para o topo da pilha. Uma variável similar, *lastwin*, aponta para a última janela da pilha. Os apontadores contidos na estrutura das janelas mantém o controle da pilha em ambas as direções. O apontador *next* (em direção ao fundo da pilha) e o apontador *prev* (em direção ao topo da pilha) contém as ligações para as janelas adjacentes.

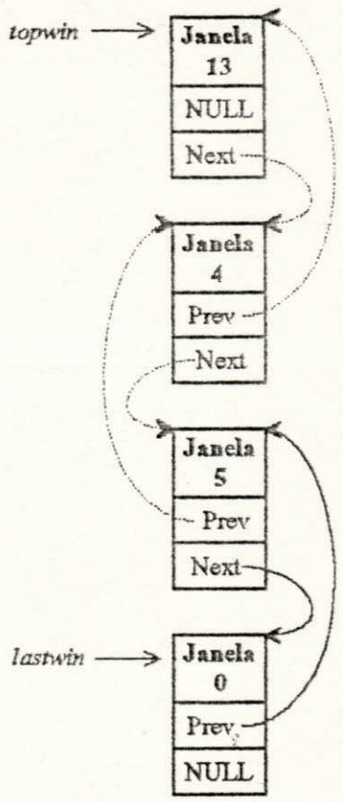


Figura 3.3 - Estrutura da pilha de janelas

3.2.3. *Movendo e Redimensionando as Janelas*

MoveJanela move a janela ativa para uma nova posição na tela. *RedimensionaJanela* altera as dimensões da janela ativa, sem no entanto efetuar operações de *zoom* nos dados contidos na mesma. Em ambos os casos, as informações contidas na janela permanecem com suas posições, em relação ao canto superior esquerdo, inalteradas.

3.3. *Descrição das Funções de Tratamento de Janelas*

As quatro funções que fazem a interface do Módulo Gerenciador de Telas têm como parâmetro um único número, que é o de identificação da janela sobre a qual será realizada a operação desejada.

AbreJanela

A função AbreJanela, recebe um parâmetro que especifica o número da janela a ser aberta. Na medida em que janelas são abertas, elas são empilhadas uma sobre as outras.

Alguns cuidados são levados em consideração pelo código de gerenciamento ao ser aberta uma nova janela. É verificado se existe definição da janela, cujo número foi passado como parâmetro. Havendo ausência de definição para uma janela, o gerenciador devolve o controle a rotina que o chamou e informa o erro.

Abrir uma janela sempre implica em torná-la a janela ativa. Isto significa que a janela será aberta no topo, ou seja, na frente das demais janelas. Tentar abrir uma janela já aberta implica em remove-la da sua atual posição na pilha para sua nova posição no topo das demais janelas. Por exemplo, abrir um novo elemento de uma classe de diálogo em uma janela previamente aberta.

FechaJanela

É a função complementar a função AbreJanela. Recebe como parâmetro o número de identificação da janela a ser fechada. Apesar do nome, esta função, não fecha necessariamente a janela especificada.

Verifica se a janela especificada esta aberta. Esta verificação é feita consultando as informações que se encontram na pilha das janelas.

Independente da posição atual na pilha o procedimento para apagar uma janela é o mesmo. Inicialmente é verificado se a janela não possui mais filhos, o que significaria está sendo usada por outros elementos de uma classe de dialogo. Em seguida a mesma é trazida para o topo da pilha e então apagada.

MoveJanela

Permite a movimentação de uma janela pela tela. A janela a ser movimentada é aquela cujo número é passado como parâmetro. Apenas a moldura do retângulo é movimentada até se completar a operação, quando então o conteúdo é atualizado.

Um método de ativação sugerido para esta rotina consiste em se usar uma tecla pré-definida. A partir de então, a movimentação da janela seria feita, de forma interativa, através das teclas de navegação.

RedimJanela

Permite o redimensionamento da janela especificada. Apenas a moldura do retângulo é alterada até se completar a operação.

De forma análoga a MoveJanela, um método de ativação sugerido para esta rotina consiste em usar uma tecla pré-definida. A partir de então, o redimensionamento da janela seria feito de forma interativa, através das *teclas de navegação*, i.e., das *setas* existentes no teclado reduzido do PC.

3.4. Sintaxe das Funções de Tratamento de Janela

- **AbreJanela**

Função: Abre no vídeo a janela especificada.

Sintaxe: *#include "gapres.h"*

unsigned long AbreJanela(int num);

Protótipo: *gapres.h*

Comentários: Caso a janela já se encontre aberta, passa a ser a janela ativa.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
        int i;

        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo da interface */
        for( i=0; i<5; i++ )        /* abre as janelas 0 a 4 */
                AbreJanela( i );
        sleep( 5 );                 /* aguarda 5 segundos */
        while( --i )                /* fecha as janelas 4 a 0 */
                FechaJanela( i );
        TerminaGA();                /* encerra as atividades com o GA */
}
```

• **FechaJanela**

Função: Fecha no vídeo a janela especificada.

Sintaxe: *#include "gapres.h"*

unsigned long FechaJanela(int num);

Protótipo: *gapres.h*

Comentários: Caso a janela tenha outros filhos, a mesma não é fechada.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
        int i;

        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo da interface */
        for( i=0; i<3; i++ )        /* abre as janelas 0, 1 e 3 */
                AbreJanela( i );
        FechaJanela( 1 );           /* fecha a janela 1 */
        sleep( 5 );                 /* aguarda 5 segundos */
        FechaJanela( 2 );           /* fecha a janela 2 */
        FechaJanela( 0 );           /* fecha a janela 0 */
        TerminaGA();                /* encerra as atividades com o GA */
}
```

• **MoveJanela**

Função: Move no vídeo a janela especificada.

Sintaxe: *#include "gapres.h"*

*void MoveJanela(int num, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: Se o apontador *p* for igual a NULL, desloca a janela de forma interativa pela tela, através das teclas de navegação. Caso contrário, move a janela para o ponto especificado, respeitando os limites da janela.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>          /* define NULL */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
    IniciaGA( CGA, CGAHI );   /* inicializa o GA */
    LeModelo( "teste" );      /* le o modelo da interface */
    AbreJanela( 1 );          /* abre a janela número 1 */
    MoveJanela( 1, NULL );    /* move a janela 1 de forma interativa */
    FechaJanela( 1 );         /* fecha a janela 1 */
    TerminaGA();              /* encerra as atividades com o GA */
}
```

- **RedimJanela**

Função: Redimensiona no vídeo a janela especificada

Sintaxe:
```
#include "gapres.h"
void RedimJanela( int num, struct diagonal *d );
```

Protótipo: *gapres.h*

Comentários: Se o apontador *d* for igual a NULL, redimensiona a janela de forma interativa pela tela, através das teclas de navegação. Caso contrário, redimensiona a janela para o tamanho da diagonal especificada, respeitando os limites da janela.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>          /* define NULL */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */
```

```
main()
{
        struct diagonal d;

        d.left = d.top = 0;
        d.right = d.bottom = 50;
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo da interface */
        AbreJanela( 1 );            /* abre a janela número 1 */
        RedimJanela( 1, &d );       /* redimensiona para os valores de d */
        FechaJanela( 1 );           /* fecha a janela 1 */
        TerminaGA();                /* encerra as atividades com o GA */
}
```

3.5. *Tratamento das Classes de Diálogo*

Do ponto de vista do Módulo Gerenciador de Telas, as classes de diálogo representam as várias formas de interação com o usuário. Cada classe de diálogo engloba características próprias. Estas características são encapsuladas pelas funções do Módulo Gerenciador de Telas.

Numa visão macro deste módulo, a parte que faz o tratamento das classes de diálogo tem em comum cinco tipos de funções:

Abre*Classe*, Le*Classe*, Move*Classe*, Redim*Classe* e Fecha*Classe*.

Na descrição acima, onde existir a palavra *Classe* esta deverá ser substituída por uma das seguintes palavras: Menu, Form, Cmdo, Perg. Por exemplo, *AbreMenu*, *FechaForm*, etc.

A representação na tela de cada uma das classes de dialogo, bem como sua estrutura de dados é mostrada a seguir.

Layout Menu

A Figura 3.4, ilustra o layout de um menu vertical do Gerenciador de Apresentação. Nela podem-se visualizar os diversos atributos que podem ser usados para melhorar o aspecto dos menus. Estes atributos são individuais, i.e., o menu tem atributos gerais, mas cada ítem pode ser modificado individualmente.

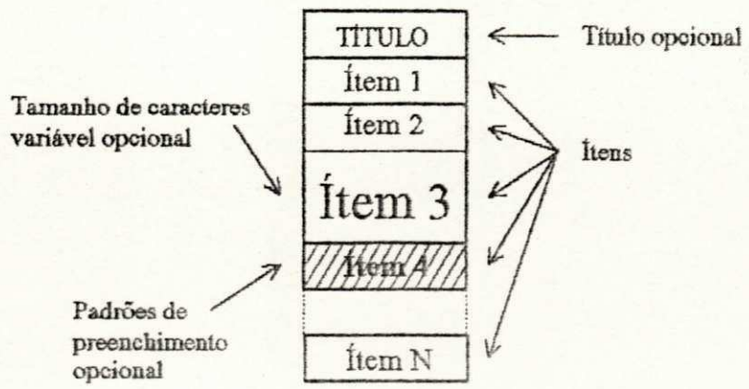


Figura 3.4 - Layout de um menu.

Dentre as idéias previstas, mas não implementadas, está a possibilidade de menus horizontais. A implementação deste tipo de menu facilitará o projeto de sistemas de menus *pull-down*.

Uma outra melhoria prevista é permitir o rolamento dos ítens de um menu. Esta facilidade implica em menus que não estariam restritos ao tamanho da janela.

A.Figura 3.5, a seguir, mostra um exemplo de um dos menus do AGILE:

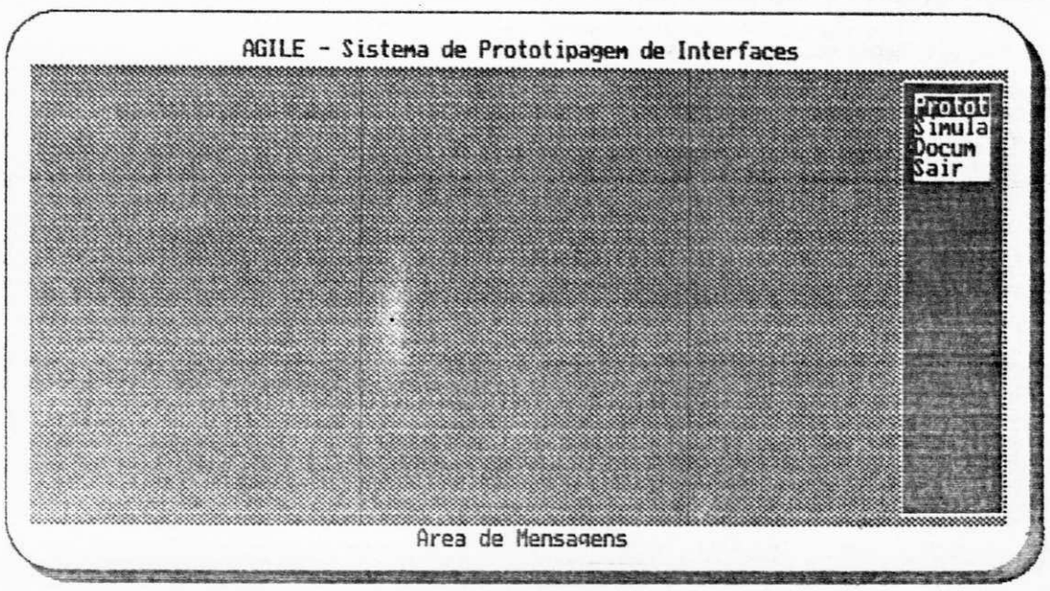


Figura 3.5 - Exemplo de um elemento da classe de diálogo *menu*.

op

A estrutura de dados *op* define a lista das opções válidas para o menu.

```
typedef struct op {
        char *str;
        struct op *prox;
} op_t;
```

Os membros desta estrutura são definidos como mostra a Tabela 3.5, a seguir:

MEMBRO	DESCRIÇÃO
str	Cadeia de caracteres que define a opção. Normalmente um único caractere.
prox	Apontador para a próxima estrutura *op*.

Tabela 3.5 - Descrição dos membros da estrutura *op*.

opcao

A estrutura *opcao* define o tipo de seleção a ser usado pelo menu.

```
typedef struct opcao {
        int tipo_sel;
        TECLAS term;
        op_t *id_op;
} opcao_t;
```

A definição dos membros da estrutura *opcao* pode ser vista na Tabela 3.6, a seguir:

MEMBRO	DESCRIÇÃO
tipo_sel	Define o tipo de seleção a ser usado pelo menu. Esta seleção pode ser por barra, por string ou por barra e string.
term	Estrutura que define o terminador usado para informar que a seleção já foi efetuada.
id_op	Apontador para a lista da opções válidas.

Tabela 3.6 - Descrição dos membros da estrutura *opcao*.

menu

A estrutura *menu* define as características e atributos do menu.

```
typedef struct menu {
        char *buf;
        int num_jan;
        int num_itens;
        int direcao;
        int wrap;
```

```
    int borda;
    int lt;
    int tp;
    int rt;
    int bm;
    int curx;
    int cury;
    int stdfill;
    int corfill;
    texto_t *titulo;
    int tstdfill;
    int tcorfill;
    int altura;
    texto_t *itens;
    opcao_t *opcoes;
} menu_t;
```

Os membros da estrutura *menu* são definidos como mostrado na Tabela 3.7, a seguir:

MEMBRO	DESCRIÇÃO
buf	Apontador para região de memória, onde será salva a região do vídeo encoberto.
num_jan	Número da janela onde o menu será desenhado.
num_itens	Número de ítens do menu.
direcao	Especifica se o menu é horizontal ou vertical.
wrap	Especifica se o menu permite a passagem cíclica, do último ítem para o primeiro, e vice versa.

Tabela 3.7 - Descrição dos membros da estrutura *menu*.

MEMBRO	DESCRIÇÃO
borda	Tipo de borda do menu.
lt	Coordenada *x* esquerda da área ocupada pelo menu.
tp	Coordenada *y* superior da área ocupada pelo menu.
rt	Coordenada *x* direita da área ocupada pelo menu.
bm	Coordenada *y* inferior da área ocupada pelo menu.
curx	Posição X atual do cursor na área ocupada.
cury	Posição Y atual do cursor na área ocupada.
stdfill	Padrão de preenchimento da área ocupada (vide valores na seção sobre janelas).
corfill	Cor de preenchimento da área ocupada (vide valores na seção sobre janelas)..
titulo	Lista das linhas de título.
tstdfill	Padrão de preenchimento da área do título.
tcorfill	Cor de preenchimento da área do título.
altura	Altura total do título.
itens	Lista dos itens.
opcoes	Apontador para estrutura que define o tipo de seleção e lista das opções válidas.

Tabela 3.7 - Descrição dos membros da estrutura *menu* (continuação).

Layout Formulário

Um formulário é formado por um ou mais campos, distribuídos em uma janela. Cada campo é subdividido em duas partes, como pode ser visto na Figura 3.6.

Figura 3.6 - Layout de um campo de um *formulario*.

Cada parte tem seus próprios atributos de cor, tipo e tamanho de letra, e padrão de preenchimento.

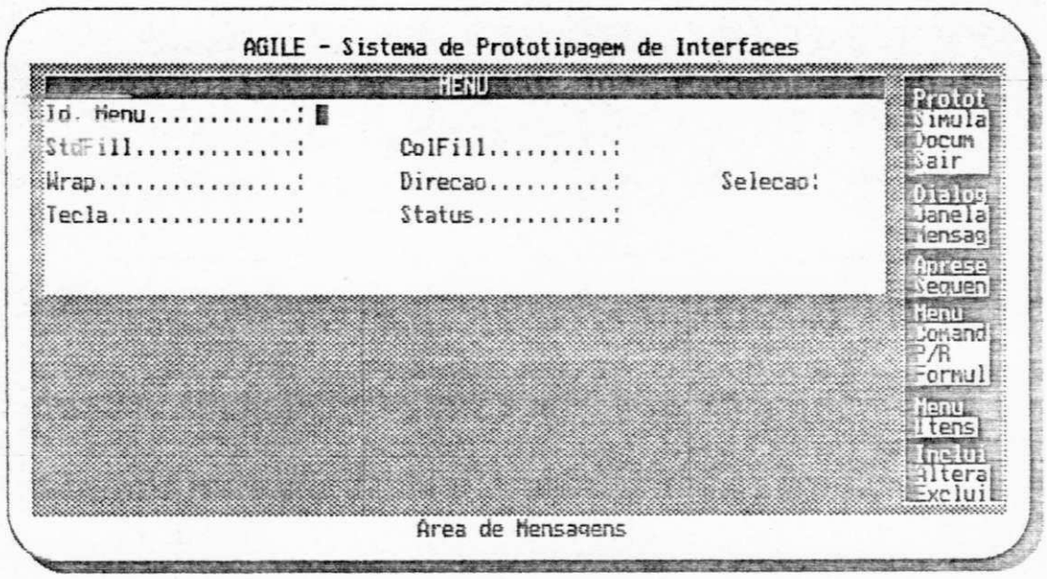


Figura 3.7 - Exemplo de um elemento da classe de diálogo *formulario*.

campo

A estrutura *campo* define as características de um campo de um formulário.

typedef struct campo {

int lin;

```
    int col;
    texto_t *nome;
    int dlin;
    int dcol;
    char dlarg;
    texto_t *dado;
} campos_t;
```

A definição dos membros da estrutura *campo* pode ser vista na Tabela 3.8, a seguir:

MEMBRO	DESCRIÇÃO
lin	Posicionamento y do nome do campo.
col	Posicionamento x do nome do campo.
nome	Apontador para a estrutura que define o nome do campo.
dlin	Posicionamento y do dado lido no campo.
dcol	Posicionamento x do dado lido no campo.
dlarg	Largura do campo de dados.
dado	Apontador para a estrutura que contém o dado.

Tabela 3.8 - Descrição dos membros da estrutura *campo*.

formulario

A estrutura *formulario* define as características do formulário.

```
typedef struct formulario {
    char *buf;
    int num_jan;
    int tipo;
    int num_campos;
```

```
    TECLAS term;
    campos_t *p;
} form_t;
```

A definição dos membros da estrutura *formulario* pode ser vista na Tabela 3.9, a seguir:

MEMBRO	DESCRIÇÃO
buf	Apontador para a região de memória, onde será salva a área encoberta pelo formulário.
num_jan	Número da janela onde será colocado o formulário.
tipo	Define o tipo de formulário: form ou tabela.
num_campos	Número de campos no formulário.
term	Terminador.
p	Apontador para a lista de comandos.

Tabela 3.9 - Descrição dos membros da estrutura *formulario*.

Layout Comando

Duas partes compõem um comando: o *prompt*, e a área de digitação do comando a ser executado. A área de digitação do comando é limitada em 255 caracteres. A Figura 3.8 ilustra o layout de um comando.

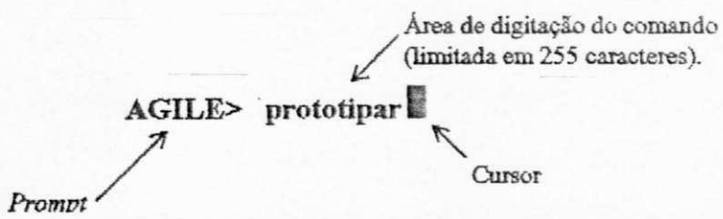


Figura 3.8 - Layout de um *comando*.

O uso da tecla terminadora, usualmente a tecla *return*, sem que nenhum comando tenha sido digitado, causa o *display* de um novo *prompt* na linha seguinte. Caso o limite da janela seja atingido, haverá um rolamento da mesma. Devido a esta característica é aconselhável que cada comando tenha sua janela exclusiva. A Figura 3.9 mostra um exemplo de um elemento da classe *comando*.

Figura 3.9 - Exemplo de um elemento da classe de diálogo *comando*.

comando

A estrutura *comando* define as características do comando.

```
typedef struct comando {
    char *buf;
    int num_jan;
    int x_inic;
    int y_inic;
    int curx;
```

```
    int cury;
    int cursor;
    char *prompt;
    TECLAS term;
} cmd_t;
```

Os membros da estrutura *comando* são definidos como mostrados na Tabela 3.10, abaixo:

MEMBRO	DESCRIÇÃO
buf	Apontador para a região de memória onde será salva a área a ser encoberta pelo diálogo comando.
num_jan	Número da janela onde será desenhado o comando.
x_inic	Posição X inicial.
y_inic	Posição Y inicial.
curx	Posição X atual do cursor na área ocupada.
cury	Posição Y atual do cursor na área ocupada.
cursor	Tipo de cursor.
prompt	Prompt.
term	Terminador.

Tabela 3.10 - Descrição dos membros da estrutura *comando*.

Layout Pergunta&Resposta

O layout da classe de diálogo *pergunta&resposta* é semelhante ao de *comando*. Composto por duas partes: a pergunta, e a área de digitação da resposta. A área de digitação da resposta é limitada em 255 caracteres. A Figura 3.10 ilustra o layout desta classe.

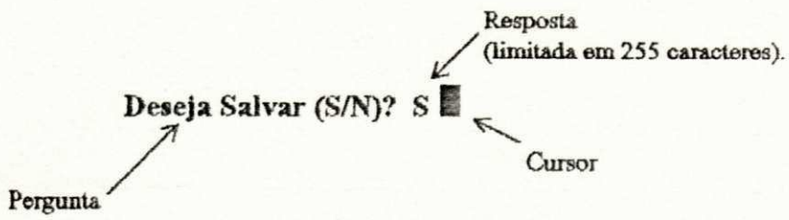


Figura 3.10 - Layout de uma *pergunta&resposta*.

O uso da tecla terminadora, usualmente a tecla *return*, sem que nenhuma resposta tenha sido digitada, causa o *display* da pergunta na linha seguinte. Caso o limite da janela seja atingido, haverá um rolamento da mesma. Devido a esta característica é aconselhável que cada pergunta tenha sua janela exclusiva. A Figura 3.11, mostra um exemplo de um elemento da classe *pergunta&resposta*.

Figura 3.11 - Exemplo de um elemento da classe de diálogo *pergunta*.

pergunta

A estrutura *pergunta* define as características da pergunta&resposta.

```
typedef struct pergunta {
        char *buf;
        int num_jan;
        int x_inic;
        int y_inic;
        int curx;
        int cury;
        int cursor;
        char *prompt;
        texto_t *pr;
        TECLAS term;
} perg_t;
```

Os membros da estrutura *pergunta* são definidos conforme mostra a Tabela 3.11.

MEMBRO	DESCRIÇÃO
buf	Apontador para a região de memória onde será salva a região do vídeo encoberto.
num_jan	Número da janela onde será colocada a pergunta.
x_inic	Posição X inicial.
y_inic	Posição Y inicial.
curx	Posição X atual do cursor na área ocupada.
cury	Posição Y atual do cursor na área ocupada.

Tabela 3.11 - Descrição dos membros da estrutura *pergunta*.

MEMBRO	DESCRIÇÃO
cursor	Tipo de cursor.
prompt	Prompt.
pr	Apontador para a cadeia de caracteres que contém a pergunta.
term	Terminador.

Tabela 3.11 - Descrição dos membros da estrutura *pergunta* (continuação).

Layout Mensagem

O *layout* para a classe *mensagem,* consiste em uma série de frases, armazenadas em uma estrutura do tipo *texto*, que são escritas em uma janela especificada. A coordenada inicial, i.e, o local a partir de onde é escrito o texto é definido pelas variáveis *x_inic* e *y_inic*.

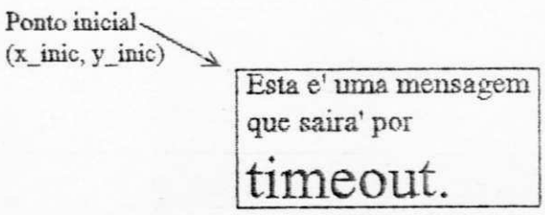


Figura 3.12 - Layout de uma *mensagem*.

Como pode ser visto na Figura 3.12, é possível ter tamanhos de letras diferentes em uma mesma *mensagem*. Também são permitidos tipos de letras diferentes e padrões de preenchimento.

A Figura 3.13, mostra um exemplo de uma mensagem que alerta o usuário para uma entrada de comando inválida.

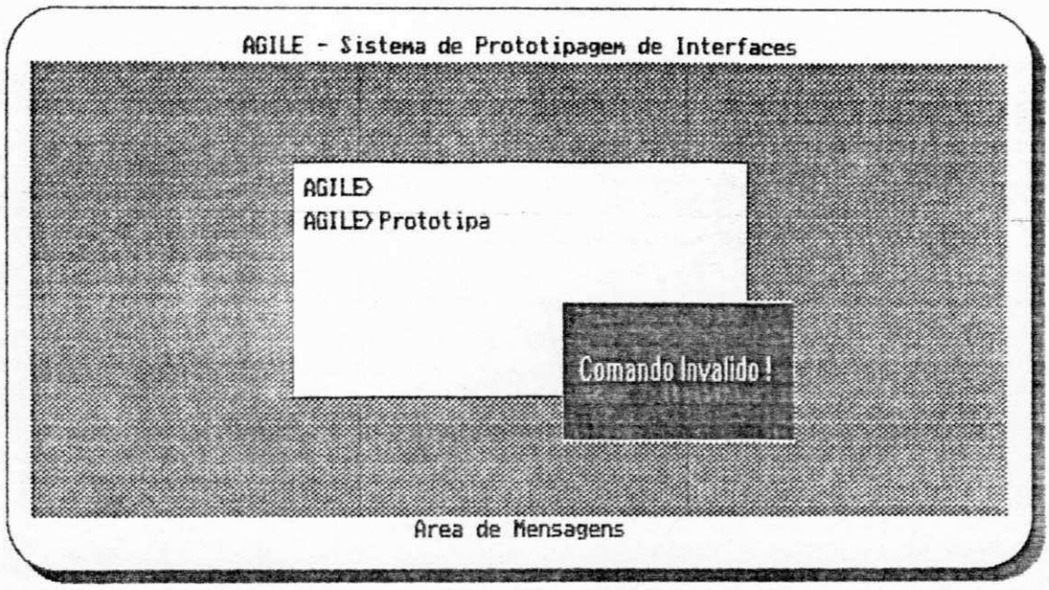


Figura 3.13 - Exemplo de um elemento da classe de diálogo *mensagem*.

mensagem

A estrutura *mensagem* define as características da mensagem.

```
typedef struct mensagem {
        char *buf;
        int num_jan;
        int x_inic;
        int y_inic;
        int timeout;
        TECLAS term;
        texto_t *texto;
} mens_t;
```

Os membros da estrutura *mensagem* são definidos como mostra a Tabela 3.12, a seguir:

MEMBRO	DESCRIÇÃO
buf	Apontador para a região de memória onde será salvo a área a ser encoberta pelo diálogo *mensagem*.
num_jan	Número da janela onde será escrita a mensagem.
x_inic	Posição X inicial.
y_inic	Posição Y inicial.
timeout	Um valor diferente de zero, especifica que deve ser utilizado *timeout*. A unidade é o segundo.
term	Tecla usada como terminador, se o valor de *timeout* for igual a zero.
texto	Apontador para a estrutura com a lista de *strings* da mensagem.

Tabela 3.12 - Descrição dos membros da estrutura *mensagem*.

3.6. Sintaxe das Funções de Tratamento de Classes de Diálogo

◦ AbreMenu

Função: Abre no vídeo o menu especificado.

Sintaxe:
```
#include "gapres.h"
unsigned long AbreMenu( int num );
```

Protótipo: *gapres.h*

Comentários: Antes desta função ser chamada, o Gerenciador de Apresentação deve ter sido inicializado, como também, a janela correspondente deve estar aberta.

Retorna: Retorna um código de erro.

Exemplo:
```
#include <dos.h>                    /* apenas pelo prototipo do sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
```

```
    IniciaGA( CGA, CGAHI );          /* inicializa o GA */
    LeModelo( "teste" );             /* le um modelo para teste */
    AbreJanela( men[0].num_jan );    /* abre a janela */
    AbreMenu( 0 );                   /* abre o menu zero */
    sleep( 5 );                      /* aguarda 5 segundos */
    FechaMenu( 0 );                  /* fecha o menu zero */
    FechaJanela( men[0].num_jan );   /* fecha a janela */
    TerminaGA();                     /* encerra as atividades c/ o GA */
}
```

• **FechaMenu**

Função: Fecha no vídeo o menu especificado.

Sintaxe: *#include "gapres.h"*

unsigned long FechaMenu(int num);

Protótipo: *gapres.h*

Comentários: O menu especificado deve ter sido previamente aberto.

Retorna: Retorna um código de erro.

Exemplo: *Vide exemplo da função AbreMenu.*

• **LeMenu**

Função: Lê a opção do menu especificado.

Sintaxe: *#include "gapres.h"*

*unsigned long LeMenu(int num, struct ret_leit *p, int cont_dial);*

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                 /* prototipo da funcao sleep */
#include "gapres.h"              /* definicoes e declaracoes do GA */
```

```
main()
{
        struct ret_leit p;                  /* retorno dos dados */

        IniciaGA( CGA, CGAHI );             /* inicializa o GA */
        LeModelo( "teste");                 /* le o modelo */
        AbreJanela( men[0].num_jan );       /* abre a janela */
        AbreMenu( 0 );                      /* abre o menu zero */
        LeMenu( 0, &p, 0 );                 /* le o menu zero */
        if( p.flag_ret == VERDADE ){        /* certifica-se que ha' dado */
                /* abre o menu correspondente a opcao */
                AbreJanela( men[p.val_ret.ret_int].num_jan );
                AbreMenu( p.val_ret.ret_int );
                sleep( 5 );
                FechaMenu( p.val_ret.ret_int );
                FechaJanela( men[p.val_ret.ret_int].num_jan );
        }
        FechaMenu( 0 );                     /* fecha o menu zero */
        FechaJanela( men[0].num_jan );      /* fecha a janela */
        TerminaGA();                        /* encerra as atividades c/ o GA */
}
```

- **MoveMenu**

Função: Move o menu especificado.

Sintaxe: *#include "gapres.h"*

*unsigned long MoveMenu(int num, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: Caso o apontador *p* tenha valor NULL, desloca o menu de forma interativa dentro da janela. Caso contrário, o valor apontado por *p* especifica a nova coordenada do canto superior esquerdo do menu.

Retorna: Retorna um código de erro.

Exemplo:
```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo */
        AbreJanela( men[0].num_jan ); /* abre a janela */
        AbreMenu( 0 );              /* abre o menu zero */
        MoveMenu( 0, NULL );        /* move o menu 0 interativamente */
        FechaMenu( 0 );             /* fecha o menu zero */
        FechaJanela( men[0].num_jan ); /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

• AbreForm

Função: Abre no vídeo o formulário especificado.

Sintaxe:
```
#include "gapres.h"
unsigned long AbreForm( int num );
```

Protótipo: *gapres.h*

Comentários: A janela na qual o formulário deve ser desenhado, deve estar aberta.

Retorna: Retorna um código de erro.

Exemplo:
```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo */
```

```
    AbreJanela(form[0].num_jan );      /* abre a janela */
    AbreForm( 0 );                     /* abre o formulario  zero */
    sleep( 5 );                        /* aguarda 5 segundos */
    FechaForm( 0 );                    /* fecha o formulario zero */
    FechaJanela(form[0].num_jan );     /* fecha a janela */
    TerminaGA();                       /* encerra as atividades c/ o GA */
}
```

• FechaForm

Função: Fecha no vídeo o formulário especificado.

Sintaxe: *#include "gapres.h"*

unsigned long FechaForm(int num);

Protótipo: *gapres.h*

Comentários: O formulário deve ter sido previamente aberto com a função *AbreForm*.

Retorna: Retorna um código de erro.

Exemplo: *Vide exemplo da função AbreForm.*

• LeForm

Função: Lê o campo especificado de um formulário especificado.

Sintaxe: *#include "gapres.h"*

*unsigned long LeForm(int num, struct ret_leit *p, int cont_dial, int campo);*

Protótipo: *gapres.h*

Comentários: A variável *num* especifica o formulário. A variável *campo* especifica o campo deste formulário a ser lido.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <stdio.h>                 /* funcoes E/S do C */
#include <dos.h>                   /* prototipo da funcao sleep */
#include "gapres.h"                /* definicoes e declaracoes do GA */
```

```
main()
{
        FILE *fp;
        struct ret_leit p;                      /* retorno dos dados */
        int campo;                              /* indica o campo a ser lido */
        form_t *f;

        /* abre um arquivo para armazenar os dados lidos do formulario */
        if( (fp = fopen( "arqsaida.dat", "w" )) == NULL ){
                puts( "Nao foi possivel abrir o arquivo." );
                exit( 1 );
        }
        IniciaGA( CGA, CGAHI );                 /* inicializa o GA */
        LeModelo( "teste" );                            /* le o modelo */
        f = form[2];                                    /* comodidade */
        AbreJanela( f->num_jan );                       /* abre a janela */
        AbreForm( 2 );                                  /* abre o formulário 2 */
        /* le todos os campos do formulario */
        for( campo = 0; campo < f->num_campos; campo++ ){
                LeForm( 2, &p, 0, campo );              /* le o campo especificado*/
                if( p.flag_ret == VERDADE ){            /* certifica-se que ha dado */
                        fprintf( fp, "%s\n", p.val_ret.ret_str );  /* salva em disco */
                } else {
                        fprintf( fp, "ERRO!\n" );
                }
        }
        FechaForm( 2 );                         /* fecha o formulario 2 */
        FechaJanela( f->num_jan );              /* fecha a janela */
        TerminaGA();                            /* encerra as atividades c/ o GA */
        fclose( fp );
}
```

• MoveForm

Função: Move o formulário especificado.

Sintaxe: *#include "gapres.h"*

*unsigned long MoveForm(int num, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: O formulário especificado deve ter sido previamente aberto. Caso o apontador *p* tenha valor NULL, desloca o formulário de forma interativa dentro da janela. Caso contrário, o valor apontado por *p* especifica a nova coordenada do canto superior esquerdo do formulário.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        struct ponto p;

        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo */
        AbreJanela(form[0].num_jan ); /* abre a janela */
        AbreForm( 0 );              /* abre o formulario zero */
        p.left = p.top =  15;
        MoveForm( 0, &p );          /* move  para a nova posicao */
        FechaForm( 0 );             /* fecha o formulario zero */
        FechaJanela(form[0].num_jan ); /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

• **AbreCmdo**

Função: Abre no vídeo o comando especificado.

Sintaxe: *#include "gapres.h"*

unsigned long AbreCmdo(int num);

Protótipo: *gapres.h*

Comentários: Antes desta função ser chamada, o Gerenciador de Apresentação deve ter sido inicializado, como também, a janela correspondente deve estar aberta.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* apenas pelo prototipo do sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste");         /* le um modelo para teste */
        AbreJanela( cmd[0].num_jan ); /* abre a janela */
        AbreCmdo( 0 );              /* abre o comando zero */
        sleep( 5 );                 /* aguarda 5 segundos */
        FechaCmdo( 0 );             /* fecha o comando zero */
        FechaJanela( cmd[0].num_jan ); /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

• **FechaCmdo**

Função: Fecha no vídeo o comando especificado.

Sintaxe: *#include "gapres.h"*

unsigned long FechaCmdo(int num);

Protótipo: *gapres.h*

Comentários: O comando especificado deve ter sido previamente aberto.

Retorna: Retorna um código de erro.

Exemplo: *Vide exemplo da função AbreCmdo.*

• LeCmdo

Função: Lê os campos do comando especificado.

Sintaxe: *#include "gapres.h"*

*unsigned long LeCmdo(int num, struct ret_leit *p, int cont_dial);*

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                        /* prototipo da funcao sleep */
#include "gapres.h"                     /* definicoes e declaracoes do GA */
main()
{
        struct ret_leit p;              /* retorno dos dados */

        IniciaGA( CGA, CGAHI );         /* inicializa o GA */
        LeModelo( "teste");             /* le o modelo */
        AbreJanela( cmd[0].num_jan );   /* abre a janela */
        AbreCmdo( 0 );                  /* abre o comando zero */
        LeCmdo( 0, &p, 0 );             /* le o comando zero */
        if( p.flag_ret == VERDADE ){    /* certifica-se que ha' dado */
                /* se o que foi digitado e' "menu" */
                if( !strcmp( p.val_ret.ret_str, "menu" ) ){
                        AbreJanela( men[0].num_jan );   /* abre a janela */
                        sleep( 5 );                     /* aguarda 5 s */
                        FechaJanela( men[0].num_jan );  /* fecha a janela */
                }
        }
        FechaCmdo( 0 );                 /* fecha o comando zero */
        FechaJanela( cmd[0].num_jan );  /* fecha a janela */
```

```
        TerminaGA();                    /* encerra as atividades c/ o GA */
}
```

• **MoveCmdo**

Função: Move o comando especificado.

Sintaxe: *#include "gapres.h"*

*unsigned long MoveCmdo(int num, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: O comando especificado deve ter sido previamente aberto. Caso o apontador *p* tenha valor NULL, desloca o comando de forma interativa dentro da janela. Caso contrário, o valor apontado por *p* especifica a nova coordenada do canto superior esquerdo do comando.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo */
        AbreJanela( cmd[0].num_jan ); /* abre a janela */
        AbreCmdo( 0 );              /* abre o comando zero */
        MoveCmdo( 0, NULL );        /* move  interativamente */
        FechaCmdo( 0 );             /* fecha o comando zero */
        FechaJanela( cmd[0].num_jan ); /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

• **AbrePerg**

Função: Abre no vídeo a pergunta especificada.

Sintaxe: *#include "gapres.h"*

unsigned long AbrePerg(int num);

Protótipo: *gapres.h*

Comentários: Antes desta função ser chamada, o Gerenciador de Apresentação deve ter sido inicializado, como também, a janela correspondente deve estar aberta.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* apenas pelo prototipo do sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste");         /* le um modelo para teste */
        AbreJanela( perg[0].num_jan );  /* abre a janela */
        AbrePerg( 0 );              /* abre a pergunta zero */
        sleep( 5 );                 /* aguarda 5 segundos */
        FechaPerg( 0 );             /* fecha a pergunta zero */
        FechaJanela( perg[0].num_jan );  /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

• **FechaPerg**

Função: Fecha no vídeo a pergunta especificada.

Sintaxe: *#include "gapres.h"*

unsigned long FechaPerg(int num);

Protótipo: *gapres.h*

Comentários: A pergunta especificada deve ter sido previamente aberta.

Retorna: Retorna um código de erro.

Exemplo: *Vide exemplo da função AbrePerg.*

• LePerg

Função: Lê a resposta para a pergunta especificada.

Sintaxe: *#include "gapres.h"*

*unsigned long LePerg(int num, struct ret_leit *p, int cont_dial);*

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */
main()
{
        struct ret_leit p;          /* retorno dos dados */

        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste");         /* le o modelo */
        AbreJanela( perg[0].num_jan );  /* abre a janela */
        AbrePerg( 0 );              /* abre a pergunta zero */
        LePerg( 0, &p, 0 );         /* le a pergunta zero */
        if( p.flag_ret == VERDADE ){    /* certifica-se que ha' dado */
                /* se a resposta foi "sim" */
                if( !strcmp( p.val_ret.ret_str, "sim" ) ){
                        AbreJanela( men[0].num_jan );
                        sleep( 5 );
                        FechaJanela( men[0].num_jan );
                }
        }
        FechaPerg( 0 );             /* fecha a pergunta zero */
```

```
        FechaJanela( perg[0].num_jan );     /* fecha a janela */
        TerminaGA();                        /* encerra as atividades c/ o GA */
}
```

• MovePerg

Função: Move a pergunta especificada.

Sintaxe: *#include "gapres.h"*

*unsigned long MovePerg(int num, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: A pergunta especificada deve ter sido previamente aberta. Caso o apontador *p* tenha valor NULL, desloca a pergunta de forma interativa dentro da janela. Caso contrário, o valor apontado por *p* especifica a nova coordenada do canto superior esquerdo da pergunta.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                        /* prototipo da funcao sleep */
#include "gapres.h"                     /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );         /* inicializa o GA */
        LeModelo( "teste" );            /* le o modelo */
        AbreJanela( perg[0].num_jan );  /* abre a janela */
        AbrePerg( 0 );                  /* abre a pergunta zero */
        MovePerg( 0, NULL );            /* move  interativamente */
        FechaPerg( 0 );                 /* fecha a pergunta zero */
        FechaJanela( perg[0].num_jan );  /* fecha a janela */
        TerminaGA();                    /* encerra as atividades c/ o GA */
}
```

• **AbreMens**

Função: Abre no vídeo a mensagem especificada.

Sintaxe: *#include "gapres.h"*

unsigned long AbreMens(int num);

Protótipo: *gapres.h*

Comentários: Antes desta função ser chamada, o Gerenciador de Apresentação deve ter sido inicializado, como também, a janela correspondente deve estar aberta.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                        /* apenas pelo prototipo do sleep */
#include "gapres.h"                     /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );         /* inicializa o GA */
        LeModelo( "teste");             /* le um modelo para teste */
        AbreJanela( mens[0].num_jan );  /* abre a janela */
        AbreMens( 0 );                  /* abre a mensagem zero */
        sleep( 5 );                     /* aguarda 5 segundos */
        FechaMens( 0 );                 /* fecha a mensagem zero */
        FechaJanela( mens[0].num_jan ); /* fecha a janela */
        TerminaGA();                    /* encerra as atividades c/ o GA */
}
```

• **FechaMens**

Função: Fecha no vídeo a mensagem especificada.

Sintaxe: *#include "gapres.h"*

unsigned long FechaMens(int num);

Protótipo: *gapres.h*

Comentários: A mensagem especificada deve ter sido previamente aberta.

Retorna: Retorna um código de erro.

Exemplo: *Vide exemplo da função AbrePerg.*

LeMens

Função: Lê a mensagem especificada.

Sintaxe: *#include "gapres.h"*

*unsigned long LeMens(int num, struct ret_leit *p, int cont_dial);*

Protótipo: *gapres.h*

Comentários: Esta função é usada para realizar o mecanismo de *timeout* para a mensagem aberta, ou para aguardar o pressionamento de uma tecla pré-determinada.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        struct ret_leit p;          /* retorno dos dados */

        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste");         /* le o modelo */
        AbreJanela( mens[0].num_jan );  /* abre a janela */
        AbreMens( 0 );              /* abre a mensagem zero */
        LeMens( 0, &p, 0 );         /* le a mensagem zero */
        FechaMens( 0 );             /* fecha a mensagem zero */
        FechaJanela( mens[0].num_jan );  /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

• **MoveMens**

Função: Move a mensagem especificada.

Sintaxe: *#include "gapres.h"*

*unsigned long MoveMens(int num, struct ponto *p);*

Protótipo: *gapres.h*

Comentários: A mensagem especificada deve ter sido previamente aberta. Caso o apontador *p* tenha valor NULL, desloca a mensagem de forma interativa dentro da janela. Caso contrário, o valor apontado por *p* especifica a nova coordenada do canto superior esquerdo da mensagem.

Retorna: Retorna um código de erro.

Exemplo:

```
#include <dos.h>                    /* prototipo da funcao sleep */
#include "gapres.h"                 /* definicoes e declaracoes do GA */

main()
{
        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        LeModelo( "teste" );        /* le o modelo */
        AbreJanela( mens[0].num_jan );  /* abre a janela */
        AbreMens( 0 );              /* abre a mensagem zero */
        MoveMens( 0, NULL );        /* move  interativamente */
        FechaMens( 0 );             /* fecha a mensagem zero */
        FechaJanela( mens[0].num_jan );  /* fecha a janela */
        TerminaGA();                /* encerra as atividades c/ o GA */
}
```

3.7. Interface com outros Aplicativos

Inicialmente pretendia-se utilizar o Gerenciador de Apresentação para compor o Sistema de Prototipagem Rápida Usuário-Computador - AGILE. Com o desenvolvimento do trabalho, percebeu-se que o acréscimo de algumas funções, tornaria possível extender o uso do

Gerenciador de Apresentação a outros aplicativos. Algumas destas funções foram implementadas e são mostradas a seguir. Outras, especificamente, funções de tratamento de janelas mais genéricas, são deixadas como sugestões para trabalho complementar.

Entre as funções importantes para tornar o Gerenciador de Apresentação mais independente do AGILE, estão as funções para preenchimento das estruturas usadas pelo mesmo. Enquanto no Sistema AGILE o projetista dispõe do editor de interfaces que assegura o preenchimento consistente das estruturas, em outras aplicações do Gerenciador de Apresentação, onde o preenchimento das estruturas é feito diretamente através destas funções, esta consistência é deixada a cargo do projetista. Sendo portanto, necessário usa-las com cautela. Seu uso indevido pode ocasionar o funcionamento errático do Gerenciador de Apresentação. Em alguns casos, pode inclusive haver um '*system halt*'. Por exemplo, abrir uma janela não criada.

Observar ainda, que a maioria das funções usadas para preenchimento ou alteração dos dados das estruturas do Gerenciador de Apresentação, devem ser chamadas quando o objeto do preenchimento ou alteração não estiver sendo utilizado por nenhuma classe de diálogo.

•AcrJan

Função: Acrescenta, ou modifica, uma janela na estrutura de dados do Gerenciador de Apresentação.

Sintaxe:
```
#include "gapres.h"
int AcrJan( int num_jan, int jlt, int jtp, int jrt, int jbm, int jstdfill, int jcorfill,
        int borda );
```

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista na Tabela 3.4.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:
```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
        IniciaGA( CGA, CGAHI );          /* inicializa o GA */
```

```
        AcrJan( 0,              /* numero da janela */
                0, 0,           /* coordenadas superior esquerda */
                100, 50         /* coordenadas inferior direita */
                SOLID_FILL,     /* padrao de preenchimento */
                BRANCO,         /* cor de preenchimento */
                BRANCO );       /* borda branca */
        AbreJanela( 0 );
        sleep( 5 );             /* aguarda 5 segundos */
        FechaJanela( 0 );
        TerminaGA();            /* encerra as atividades com o GA */
}
```

•AcrJanTit

Função: Acrescenta, ou modifica, uma linha de título na estrutura de dados da janela especificada.

Sintaxe:
```
#include "gapres.h"
int AcrJanTit( int num_titulo, int num_jan, int fonte, int charsize, int cor,
        int tstdfill, int tcorfill, char *txt );
```

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:
```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
        IniciaGA( CGA, CGAHI );         /* inicializa o GA */
        AcrJan( 0,              /* numero da janela */
                0, 0,           /* coordenadas superior esquerda */
```

```
            200, 50         /* coordenadas inferior direita */
            SOLID_FILL,     /* padrao de preenchimento */
            BRANCO,         /* cor de preenchimento */
            BRANCO );       /* borda branca */
        AcrJanTit( 0,               /* numero do titulo */
            0,                      /* numero da janela */
            DEFAULT_FONT,           /* tipo de fonte do titulo */
            1,                      /* tamanho da fonte (normal) */
            BRANCO,                 /* cor da fonte */
            SOLID_FILL,             /* padrao de preenchimento */
            PRETO,                  /* cor do preenchimento */
            "Exemplo Titulo" );     /* string do titulo */
        AbreJanela( 0 );
        sleep( 5 );             /* aguarda 5 segundos */
        FechaJanela( 0 );
        TerminaGA();            /* encerra as atividades com o GA */
}
```

•AcrJanRod

Função: Acrescenta, ou modifica, uma linha de rodapé na estrutura de dados da janela especificada.

Sintaxe:
```
#include "gapres.h"
int AcrJanTit( int num_rodape, int num_jan, int fonte, int charsize, int cor,
        int mstdfill, int mcorfill, char *txt );
```

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:
```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */
```

```
main()
{
        IniciaGA( CGA, CGAHI );    /* inicializa o GA */
        AcrJan( 0,                 /* numero da janela */
                0, 0,              /* coordenadas superior esquerda */
                200, 50            /* coordenadas inferior direita */
                SOLID_FILL,        /* padrao de preenchimento */
                BRANCO,            /* cor de preenchimento */
                BRANCO );          /* borda branca */
        AcrJanTit( 0,              /* numero do titulo */
                0,                 /* numero da janela */
                DEFAULT_FONT,      /* tipo de fonte do titulo */
                1,                 /* tamanho da fonte (normal) */
                BRANCO,            /* cor da fonte */
                SOLID_FILL,        /* padrao de preenchimento */
                PRETO,             /* cor do preenchimento */
                "Exemplo Titulo" ); /* string do titulo */
        AcrJanRod( 0,              /* numero do rodape */
                0,                 /* numero da janela */
                DEFAULT_FONT,      /* tipo de fonte do rodape */
                1,                 /* tamanho da fonte (normal) */
                PRETO,             /* cor da fonte */
                SOLID_FILL,        /* padrao de preenchimento */
                BRANCO,            /* cor do preenchimento */
                "Rodape" );        /* string do titulo */
        AbreJanela( 0 );
        sleep( 5 );            /* aguarda 5 segundos */
        FechaJanela( 0 );
        TerminaGA();           /* encerra as atividades com o GA */
}
```

•AcrMenu

Função: Acrescenta, ou modifica, um menu na estrutura de dados do Gerenciador de Apresentação.

Sintaxe: *#include "gapres.h"*

int AcrMenu(int num_menu, int num_jan, int lt, int tp, int stdfill, int corfill, int wrap, int tipo_sel, int tecla, int stat);

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista na tabela Tab.3.7 e no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo: *Vide exemplo da função AcrMenuItem.*

•AcrMenuItem

Função: Acrescenta, ou modifica, um item na estrutura de dados do menu especificado.

Sintaxe: *#include "gapres.h"*

*int AcrMenuItem(int num_item, int num_menu, char *txt, char *op, int fonte, int charsize, int cor);*

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:

```
#include <dos.h>          /* apenas pelo prototipo da função sleep */
#include "gapres.h"       /* definicoes e declaracoes globais do GA */

char *itens[] = {
        "Item 1",
        "Numero 2",
        "3. Item"
        "Ultimo"
```

```
};

main()
{
        struct ret_leit *d;
        int i;

        IniciaGA( CGA, CGAHI );     /* inicializa o GA */
        AcrJan( 0,                  /* numero da janela */
                0, 0,               /* coordenadas superior esquerda */
                200, 50             /* coordenadas inferior direita */
                SOLID_FILL,         /* padrao de preenchimento */
                BRANCO,             /* cor de preenchimento */
                BRANCO );           /* borda branca */
        AcrMenu( 0,                 /* numero do menu */
                0,                  /* numero da janela */
                10, 5,              /* canto superior esquerdo do menu */
                SOLID_FILL,         /* padrao de preenchimento */
                PRETO,              /* cor de preenchimento */
                VERDADE,            /* permite o wrap no menu */
                BARRA,              /* tipo de selecao */
                CR,                 /* return como terminador */
                0 );                /* status default */
        for( i=0; i<sizeof(itens)/sizeof(itens[0]); i++ ){
                AcrMenuItem( i,             /* numero do item */
                        0,                  /* numero do menu */
                        itens[i],           /* string do item */
                        "",                 /* nao usado no tipo BARRA */
                        DEFAULT_FONT,       /* tipo de fonte */
                        1,                  /* tamanho da fonte */
                        BRANCO );           /* cor da fonte */
        }
```

```
    AbreDialogo( MENU, 0 );     /* abre o dialogo MENU, elemento 0 */
    LeDialogo( MENU, 0, &d, 0 );        /* le o dialogo */
    FechaDialogo( MENU, 0 );    /* fecha o dialogo */
    TerminaGA();            /* encerra as atividades com o GA */
}
```

•AcrForm

Função: Acrescenta, ou modifica, um formulário na estrutura de dados do Gerenciador de Apresentação.

Sintaxe:
```
#include "gapres.h"
int AcrForm( int num_form, int num_jan, int tipo, int tecla, int stat,
        int num_campos );
```

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo: *Vide exemplo da função AcrFormCampo.*

•AcrFormCampo

Função: Acrescenta, ou modifica, um campo na estrutura de dados do formulário especificado.

Sintaxe:
```
#include "gapres.h"
int AcrFormCampo( int num_form, int num_campo, char *txt, int lin, int col,
        int fonte, int charsize, int cor, int dlin, int dcol, int dfonte, int dcharsize,
        int dcor, int dlarg );
```

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:

```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

char *campos[] = {
        "Nome:",            /* nome do campo 0 */
        "Telefone:",        /* nome do campo 1 */
        "FAX:"              /* nome do campo 2 */
};

int tams[] ={
        40,                 /* tamanho max dos dados, campo 0 */
        15,                 /* tamanho max dos dados, campo 1 */
        15                  /* tamanho max dos dados, campo 2 */
};

main()
{
        struct ret_leit *d;
        int i;

        IniciaGA( CGA, CGAHI );          /* inicializa o GA */
        AcrJan( 0,              /* numero da janela */
                0, 0,           /* coordenadas superior esquerda */
                400, 199        /* coordenadas inferior direita */
                SOLID_FILL,     /* padrao de preenchimento */
                BRANCO,         /* cor de preenchimento */
                BRANCO );       /* borda branca */
        AcrForm( 0,             /* numero do formulario */
                0,              /* numero da janela */
                FORM,           /* tipo formulario */
```

```
        CR,             /* return como terminador */
        0,              /* status default */
        sizeof(campos)/sizeof(campos[0])    /* num de campos */
        );
    for( i=0; i<sizeof(campos)/sizeof(campos[0]); i++ ){
        AcrFormCampo( i,            /* numero do campo */
            0,                      /* numero do formulário */
            campos[i],              /* string do campo */
            10 * (i+1),             /* coordenada y do campo*/
            10,                     /* coordenada x do campo */
            DEFAULT_FONT,           /* tipo de fonte do campo*/
            1,                      /* tamanho da fonte do campo*/
            PRETO,                  /* cor da fonte do campo*/
            10 * (i+1),             /* coordenada y do dado */
            strlen( campos[i] ) + 2,        /* coordenada x do dado */
            DEFAULT_FONT,           /* tipo de fonte do dado*/
            1,                      /* tamanho da fonte do dado*/
            PRETO,                  /* cor da fonte do dado*/
            tams[i] );              /* largura max do dado */
    }
    AbreDialogo( FORM, 0 );
    for( i=0; i<sizeof(campos)/sizeof(campos[0]); i++ )
        LeDialogo( FORM, 0, &d, 0, i );
    FechaDialogo( FORM, 0 );
    TerminaGA();            /* encerra as atividades com o GA */
}
```

•AcrCmdo

Função: Acrescenta, ou modifica, um comando na estrutura de dados do Gerenciador de Apresentação.

Sintaxe: *#include "gapres.h"*

*int AcrCmdo(int num_cmdo, int num_jan, int x, int y, int tecla, int stat, char *txt, int fonte, int charsize, int cor);*

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:

```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definicoes e declaracoes globais do GA */

main()
{
        struct ret_leit *d;
        int i;

        IniciaGA( CGA, CGAHI );          /* inicializa o GA */
        AcrJan( 0,              /* numero da janela */
                0, 0,           /* coordenadas superior esquerda */
                200, 50         /* coordenadas inferior direita */
                SOLID_FILL,     /* padrao de preenchimento */
                BRANCO,         /* cor de preenchimento */
                BRANCO );       /* borda branca */
        AcrCmdo( 0,             /* numero do comando */
                0,              /* numero da janela */
                2, 2,           /* canto superior esquerdo do comando */
                CR,             /* return como terminador */
                0,              /* status default */
                "AGILE>",       /* prompt */
                DEFAULT_FONT,   /* fonte */
```

```
            1,              /* tamanho da fonte dos caracteres */
            PRETO );        /* cor da fonte */
        AbreDialogo( COMANDO, 0 );
        LeDialogo( COMANDO, 0, &d, 0 );
        FechaDialogo( COMANDO, 0 );
        TerminaGA();        /* encerra as atividades com o GA */
}
```

•AcrPerg

Função: Acrescenta, ou modifica, uma pergunta na estrutura de dados do Gerenciador de Apresentação.

Sintaxe:
```
#include "gapres.h"
int AcrPerg( int num_perg, int num_jan, int x, int y, int tecla, int stat, char *txt,
        int fonte, int charsize, int cor );
```

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:
```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
#include "gapres.h"         /* definições e declarações globais do GA */

main()
{
        struct ret_leit *d;
        int i;

        IniciaGA( CGA, CGAHI );         /* inicializa o GA */
        AcrJan( 0,              /* numero da janela */
                0, 0,           /* coordenadas superior esquerda */
                200, 50         /* coordenadas inferior direita */
```

```
            SOLID_FILL, /* padrao de preenchimento */
            BRANCO,     /* cor de preenchimento */
            BRANCO );   /* borda branca */
        AcrPerg( 0,             /* numero da pergunta */
            0,                  /* numero da janela */
            2, 2,               /* canto superior esquerdo da pergunta */
            CR,                 /* return como terminador */
            0,                  /* status default */
            "Nome de login: ",  /* pergunta */
            DEFAULT_FONT,       /* fonte */
            1,                  /* tamanho da fonte dos caracteres */
            PRETO );            /* cor da fonte */

        AbreDialogo( PERGUNTA, 0 );
        LeDialogo( PERGUNTA, 0, &d, 0 );
        FechaDialogo( PERGUNTA, 0 );
        TerminaGA();            /* encerra as atividades com o GA */
    }
```

•AcrMens

Função: Acrescenta, ou modifica, uma mensagem na estrutura de dados do Gerenciador de Apresentação.

Sintaxe: *#include "gapres.h"*

*int AcrMens(int num_mens, int num_jan, int x, int y, int tecla, int stat, char *txt, int fonte, int charsize, int cor);*

Protótipo: *gapres.h*

Comentários: Uma descrição dos parâmetros pode ser vista no exemplo a seguir.

Retorna: Caso a função obtenha sucesso, retorna VERDADE. Caso contrário, retorna FALSO.

Exemplo:

```
#include <dos.h>            /* apenas pelo prototipo da função sleep */
```

```
#include "gapres.h"          /* definicoes e declaracoes globais do GA */

main()
{
        struct ret_leit *d;
        int i;

        IniciaGA( CGA, CGAHI );          /* inicializa o GA */
        AcrJan( 0,              /* numero da janela */
                0, 0,           /* coordenadas superior esquerda */
                200, 150        /* coordenadas inferior direita */
                XHATCH_FILL,    /* padrao de preenchimento */
                AZUL,           /* cor de preenchimento */
                BRANCO );       /* borda branca */
        AcrMens( 0,             /* numero da mensagem*/
                0,              /* numero da janela */
                2, 2,           /* canto superior esquerdo da mensagem */
                CR,             /* return como terminador */
                0,              /* status default */
                "Testando",     /* mensagem */
                TRIPLEX_FONT,   /* tipo de fonte */
                4,              /* tamanho da fonte */
                PRETO );        /* cor da fonte */
        AbreDialogo( MENSAGEM, 0 );
        LeDialogo( MENSAGEM, 0, &p, 0 );
        FechaDialogo( MENSAGEM, 0 );
        TerminaGA();            /* encerra as atividades com o GA */
}
```

CAPÍTULO 4

Módulo Gerenciador de Periféricos

Os projetistas de hardware continuam a ser generosos em sua produção de novos dispositivos periféricos de entrada e saída. O número e a variedade de dispositivos hoje disponíveis possibilitam o projeto de sistemas com consideravel liberdade, levando a interfaces mais amigáveis entre o usuário e o computador. Contudo, esta diversidade de dispositivos também tende a confundir os projetistas de software. A dificuldade reside na falta de formalismos suficientes para descrever e usar estes dispositivos. [ANSON 79][VAN DEN BOS 78]

A falta de formalismos é constatada se observarmos as variações nas especificações de fabricantes de um mesmo "tipo" de dispositivo. A quantidade de *drivers* de entrada e saída que acompanha certos programas aplicativos chega, as vezes, a suplantar o próprio programa.

Por exemplo, a tarefa de escrever um editor de textos com recursos de impressão em negrito, itálico, etc. pode ser bastante árdua. Quando se pensa em possibilitar a impressão em diversas impressoras enfrenta-se a inexistência de padrões. Inclusive, nas impressoras ditas compatíveis com um padrão encontram-se distorsões: por exemplo, uma determinada impressora "compatível" com o padrão EPSON, aceitando os comandos para impressão de um gráfico mas o imprimindo com densidade diferente daquela do padrão, resultando em uma imagem que ocupava apenas 2/3 da largura da página impressa.

Um dos maiores desafios da Engenharia de Software tem sido desenvolver meios de padronização de programas de modo a torná-los menos dependentes do hardware, em especial para aqueles que fazem uso de recursos gráficos. O modelo de dispositivo virtual tem sido usado

largamente em sistemas gráficos *device-independent*. Pelo menos, dois aspectos devem ser considerados quando dispositivos virtuais são usados em gerenciadores de interface gráfica:

- *Consistência*. Dispositivos virtuais são consistentes em comportamento; na visão do programador eles diferem apenas no tipo de dado retornado, e nos detalhes de "*prompting and echoing*". As condições sob as quais retornam os dados são as mesmas.

- *Isolação*. O programador não consegue distinguir entre dispositivos físicos que retornam o mesmo dado, ou aqueles que são simulados para o fazerem. Um *track ball*, um *tablet*, e um *joystick* todos usados como *LOCATORS* são indistinguíveis.

A consistência, claramente beneficia o programador pela simplificação do acesso a uma variedade de dispositivos de entrada. Uma analogia pode ser feita ao sistema operacional UNIX, que faz acessos a arquivos de dados, diretórios, dispositivos físicos de entrada e saída, de uma forma consistente, como arquivos. Esta consistência, para o programador, pode ser conseguida sem sacrifícios para a interface homem-máquina.

A isolação, no entanto, tem um efeito adverso na qualidade das interfaces do usuário. Torna-se difícil para programadores distinguir entre as características da interface de usuário que se originam do dispositivo físico, daquelas originárias do dispositivo lógico simulado por aquele dispositivo físico. Os detalhes de comportamento de dispositivos individuais, cujo acesso é negado ao programador, são justamente aqueles detalhes os quais determinam a qualidade da interação. Uma interface que funciona bem usando um dispositivo físico, pode tornar-se difícil ou impossível de usar quando este dispositivo é substituído por outro. [THOMAS 83]

No Módulo Gerenciador de Periféricos usa-se uma abordagem para descrever dispositivos periféricos de entrada e de saída que permite a total utilização de suas características e prove um alto grau de independência entre o programa aplicativo e os diversos tipos de dispositivos periféricos que poderiam estar acoplados.

Consideremos que qualquer dispositivo físico de entrada ou saída pode ser representado por uma estrutura de dados, a qual pode ser modificada temporalmente por ações em resposta a certos

eventos, e que estes dispositivos físicos também tem a habilidade de causar certos eventos como parte de seu repertório de ações.

Consideremos ainda que, partes do estado de um dispositivo podem ser visíveis para outros dispositivos de uma forma controlada, enquanto o restante permanece invisível. Um dispositivo pode, então, fazer uso das partes visíveis do estado de um outro dispositivo. Isto possibilita a intercambialidade de um único dispositivo por um grupo de dispositivos e permite ainda que um único dispositivo suporte a função de vários outros. Portanto, um grupo de dispositivos assim definidos pode simular qualquer grupo de dispositivos definidos da maneira usual. A partir destes conceitos, a independência dos aplicativos em relação aos dispositivos é alcancada sem o usual sacrifício das considerações sobre fatores homem-máquina. [ANSON 79]

Com base nestas considerações foi concebido o Módulo Gerenciador de Periféricos que é o objeto deste capítulo. Assim serão apresentadas neste capítulo, as rotinas implementadas para leitura do teclado, *mouse* e *joystick*, de forma a permitir que uma aplicação não sofra alterações caso um ou outro dispositivo seja utilizado. Por exemplo, a movimentação do cursor, pode ser feita tanto pelo deslocamento do mouse quanto pelas teclas de "navegação".

Em [OSÓRIO 88] encontramos uma abordagem semelhante, porém limitada ao *mouse* e teclado.

4.1. Descrição Funcional

O Módulo Gerenciador de Periféricos é responsável pelas funções de entrada e saída de dados. É também responsável pela padronização da entrada e saída de dados, deixando sempre que possível transparente para os demais módulos o tipo de dispositivo periférico de entrada ou saída que está sendo usado naquele momento.

A operação deste Módulo é baseada na Matriz de Funções de Periféricos. Na realidade, são duas matrizes: uma matriz é responsável pelos dispositivos de entrada e a outra pelos de saída. As

informações que contém são análogas e possibilitam a transparência de periféricos pretendida, além de expandirem-se facilmente para acomodar novos dispositivos periféricos e/ou novas funções.

Matriz de Funções de Periféricos

De forma genérica, a Matriz de Funções de Periféricos consiste de um conjunto de funções padrão, onde cada função corresponde a uma tarefa específica de entrada ou saída de dados.

Seu princípio de funcionamento pode ser melhor visualizado através da Figura 4.1, a seguir.

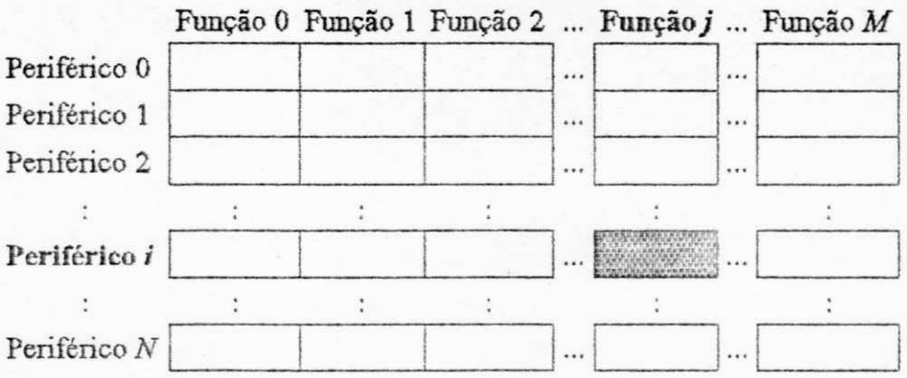


Figura 4.1 - Matriz de Funções de Periféricos.

Nesta figura, cada linha *i* representa um dispositivo periférico e cada coluna *j* uma função padrão. O conteúdo do elemento *ij* da matriz aponta para a rotina do periférico *i* que realiza ou emula a função *j*.

Apesar de ser quase sempre possível emular as funções de um dispositivo físico a partir de outro, nem sempre é prático faze-lo. Como exemplo podemos citar a entrada de caracteres via mouse.

A seguir apresentamos a sintaxe da Matriz de Funções de Periféricos, implementada em Linguagem C.

```
int (*MatFuncEnt[MAX_PERI_ENT][MAX_FUNC_ENT])() = {
        { FEnt_0_peri_0, FEnt_1_peri_0, FEnt_2_peri_0, ... },
        { FEnt_0_peri_1, FEnt_1_peri_1, FEnt_2_peri_1, ... },
                .
                .
                .
};
```

onde, MAX_PERI_ENT é o número máximo de periféricos disponíveis;
MAX_FUNC_ENT é o número máximo de funções.

Associado à Matriz de Funções de Periféricos de Entrada temos um mecanismo de *polling*. Durante a inicialização do Módulo Gerenciador de Periféricos de Entrada, todos os periféricos têm sua função 0 (zero) ativada. A função 0 (zero) é a função padrão para inicialização, ela retorna o status do periférico indicando a sua presença ou não. Caso o periférico esteja presente, ele é inicializado e sua presença sinalizada para o mecanismo de *polling*, que então o coloca em sua tabela de varredura.

Na forma de pseudo-código, mostramos na Figura 4.2, a seguir, o fluxo do programa de inicialização dos periféricos de entrada.

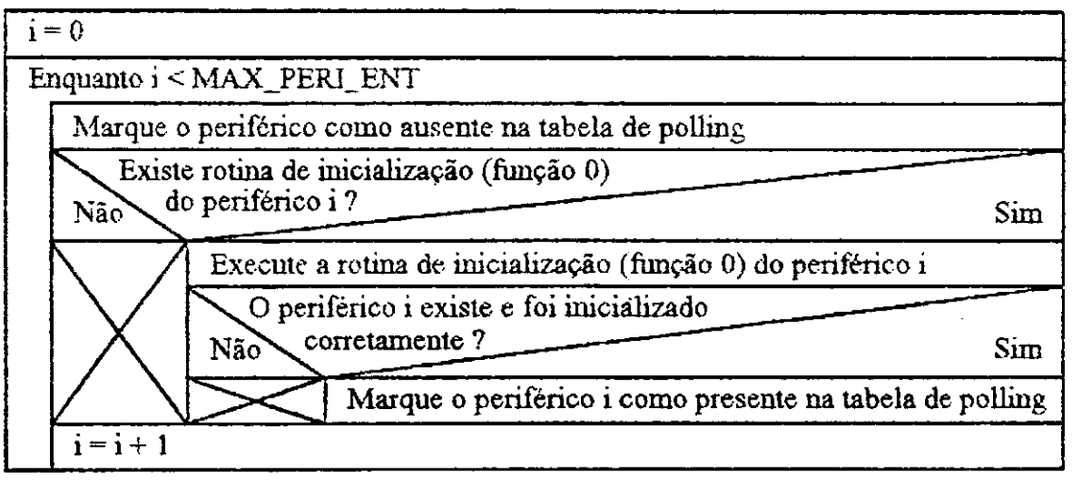


Figura 4.2 - Diagrama Nassi-Schneirdman da função de inicialização dos periféricos de entrada.

Após este procedimento, realizado uma única vez no início do programa aplicativo, os periféricos existentes passam a compor a tabela de *polling*. Cada entrada nesta tabela, pode assumir um valor lógico VERDADE ou FALSO, significando a presença ou ausência, respectivamente, do periférico. A tabela, e sua sintaxe em Linguagem C, são mostradas na Figura 4.3, a seguir:

Periférico 0	V
Periférico 1	V
Periférico 2	F
:	:
Periférico *i*	**V**
:	:
Periférico *N*	F

Sintaxe: int TabPolling[MAX_PERI_ENT];

Figura 4.3 - Tabela de *polling*.

Através do mecanismo de *polling* os dispositivos são "varridos ou pesquisados" periodicamente, para verificar a existência de algum dado disponível. A ordem desta busca é determinada pela ordem crescente das linhas da Matriz de Funções de Periféricos de Entrada (vide Figura 4.1).

Por exemplo, no caso da função *LeTecla*, temos,

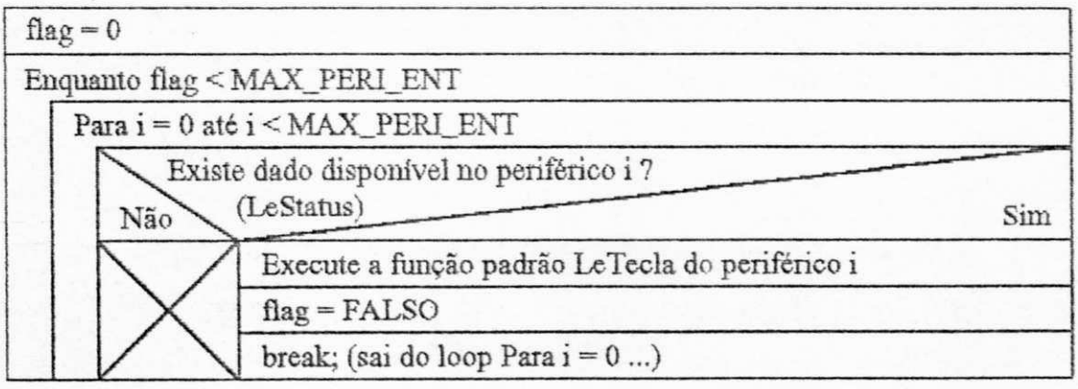


Figura 4.4 - Diagrama Nassi-Schneirdman do mecanismo de *polling* da função *LeTecla*

O uso da Matriz de Funções de Periféricos de Entrada e do mecanismo de *polling* explicados acima permitem que o usuário faça a entrada de informações com o periférico desejado, sem que para isso tenha que reconfigurar o sistema. Permite ainda a facilidade de inclusões de novos periféricos.

Inclusões de Novos Dispositivos Periféricos

Para exemplificar, vamos supor que um usuário disponha de um aplicativo que use um sistema de menus para selecionar as tarefas desejadas. A seleção do ítem desejado é feita através da primeira letra de seu nome. Este usuário adquire um periférico para reconhecimento de voz e quer usá-lo como meio de escolha dos ítens.

Observar que o acréscimo de um novo periférico *i* à Matriz de Funções de Periféricos é efetuado pelo acréscimo de uma linha de funções *j* com os apontadores para as rotinas deste periférico que realizam ou emulam as funções previstas na matriz. Implica, também, em recompilar o Módulo Gerenciador de Periféricos e realizar o *linking* com a aplicação.

A partir deste acréscimo, o usuário pode então fazer a seleção no menu via teclado ou voz, como desejado.

4.2. Dispositivos Periféricos de Entrada

Os dispositivos periféricos de entrada que fizeram parte desta versão do Módulo Gerenciador de Periféricos foram o teclado, o mouse e o joystick. Baseados nestes dispositivos foram codificadas as primeiras funções padrão. Estas funções são responsáveis pela entrada de dados e também pela definição e uso de parâmetros relacionados com os dispositivos de entrada, tais como o tipo de cursor.

Apesar de ser possível, nem todas as funções foram implementadas para todos os periféricos. Este fato é de carater estritamente prático. Citamos por exemplo, a necessidade de um programa aplicativo ler uma cadeia de caracteres alfabéticos. A forma natural para tal entrada de dados é o teclado. Eventualmente, poderia-se emular a entrada de caracteres alfabéticos via um outro dispositivo, mas esta solução seria pouco prática. Esta emulação poderia, por exemplo, ocorrer através de um mouse, usando-se um dos botões para a cada "click" passar para a próxima letra do alfabeto; um outro botão para confirmar a tecla e passar para a próximo campo; e dois "clicks" rápidos significando o fim da cadeia. Tal emulação é pouco prática e funções semelhantes não foram codificadas. As funções padrão de entrada podem ser subdivididas em dois grupos:

Grupo 1 Inicialização e Controle	
IniciaEnt	DefineFaixaH
LeStatusEnt	DefineFaixaV
CursorOn	EstiloCursor
CursorOff	LeSensivel
LePosXCursor	DefineSensivel
LePosYCursor	DesCursorFaixa
PosiçãoCursor	

Grupo 2 Leitura de Dados
LeTecla
LeLocator
LeStroke
LeValuator
LeChoice
LeString

A Figura 4.5, ilustra como a Matriz de Funções de Periféricos de Entrada implementada está preenchida. Esta é uma visão parcial, onde apenas três funções estão representadas.

	Função 0 *IniciaEnt*	Função 1 *LeStatusEnt*	Função 2 *LeTecla*	...
Teclado	*inic_tecla*	*status_tecla*	*le_tecla*	...
Mouse	*inic_mouse*	*status_mouse*	*le_tecla_mouse*	...
Joystick	*inic_joy*	*status_joy*	*le_tecla_joy*	...
:	:	:	:	
Periférico *N*	*inic_N*	*status_N*	*le_tecla_N*	...

Figura 4.5 - Visão Parcial da Matriz de Funções de Periféricos de Entrada

4.2.1. Sintaxe das Principais Funções Padrão de Entrada

Estas funções, em princípio não devem ser chamadas pelo meio exterior uma vez que desta forma a transparência dos dispositivos periféricos deixa de existir. Para tanto, o usuário deve usar as funções descritas no ítem 4.2.2.

Nos exemplos da sintaxe mostrados abaixo, o TECLADO será usado como periférico *default*, no entanto esta sintaxe permanece válida para os demais dispositivos: MOUSE e JOYSTICK. Esta sintaxe se baseia na Linguagem C.

•Função Padrão 0

Função: Inicializa o periférico de entrada.

Sintaxe:
```
#include "gapres.h"
int (*MatFuncEnt[MAX_PERI_ENT] [MAX_FUNC_ENT])();
int TabEntrada[MAX_PERI_ENT];
```

Protótipo: *gapres.h*

Comentários: O programa que chama a Função Padrão 0 deve utilizar seu valor de retorna para preencher a tabela de *polling* TabEntrada.

Retorna: Em caso de sucesso na inicialização retorna o valor lógico VERDADE; em caso de erro retorna FALSO.

Exemplo:
```
#include "gapres.h"

main()
{
        int peri;

        for( peri = 0; peri < MAX_PERI_ENT; peri++ ){
                /* inicializa o periferico, e marca-o na tabela de polling */
                TabEntrada[peri] = (*MatFuncEnt[peri] [E_INICIALIZA])();
```

```
            if( TabEntrada[peri] == VERDADE )
                printf( "Periférico %d OK\n", peri );
        }
    }
```

•Função Padrão 1

Função: Informa status do periférico

Sintaxe: *#include "gapres.h"*

*int (*MatFuncEnt[MAX_PERI_ENT][MAX_FUNC_ENT])();*

Protótipo: *gapres.h*

Comentários: Esta função deve ser usada para todos os periféricos na ordem da tabela de *polling*, com a finalidade de encontrar o periférico que dispõe de dados a serem lidos.

Retorna: Retorna VERDADE, indicando que a dado disponível para leitura; FALSO, indicando a ausência de dados.

Exemplo:

```
#include "gapres.h"

main()
{
    int stat;

    (*MatFuncEnt[TECLADO][E_INICIALIZA])();     /* inicializa */
    stat = (*MatFuncEnt[TECLADO][E_STATUS])();  /* le o status */
    if( stat == VERDADE )
        puts( "Há dados disponíveis." );
}
```

•Função Padrão 2

Função: Lê uma tecla do dispositivo periférico indicado

Sintaxe: *#include "gapres.h"*

*int (*MatFuncEnt[MAX_PERI_ENT][MAX_FUNC_ENT])(int *tecla);*

Protótipo: *gapres.h*

Comentários: O parâmetro *int *tecla* é um apontador para um vetor de duas posições inteiras, onde será armazenada a tecla lida e o status das teclas *shift*, *ctrl*, *alt*, *caps lock*, *scroll lock* e *num lock*;

Retorna: Retorna VERDADE, indicando que o dado disponível foi lido corretamente; FALSO, indicando a ausência de dados.

Exemplo:

```
#include "gapres.h"

main()
{
        int tecla[2], stat;

        (*MatFuncEnt[TECLADO][E_INICIALIZA])();      /* inicializa */
        stat = (*MatFuncEnt[TECLADO][E_LETECLA])( tecla ); /* le a tecla */
        if( stat == VERDADE )
                printf( "Tecla: %x %x\n",tecla[0], tecla[1] );
}
```

4.2.2. *Interface com Outros Módulos*

O meio exterior deve utilizar as funções descritas a seguir para interagir com o Módulo Gerenciador de Periféricos. Ao contrário do ítem anterior, estas funções permitem a transparência dos dispositivos físicos utilizados.

A sintaxe usada baseia-se na Linguagem C. Os programas aplicativos devem ser *linkados* com a biblioteca GA.LIB.

•IniciaEnt

Função: Inicializa periféricos de entrada.

Sintaxe: *#include "gapres.h"*

int IniciaEnt(void);

Protótipo: *gapres.h*

Comentários: Inicializa todos os periféricos de entrada. Os periféricos que obtiverem sucesso na sua inicialização tem o valor lógico VERDADE atribuido a sua posicao na tabela de *polling*.

Retorna: O número de periféricos inicializados.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int np;

        np = IniciaEnt();
        printf( "Nº Periféricos Inicializados: %d\n", np );
}
```

•LeStatusEnt

Função: Lê o status do dispositivo.

Sintaxe: *#include "gapres.h"*

int LeStatusEnt(void);

Protótipo: *gapres.h*

Comentários: Informa a existência de dados disponíveis para leitura. A busca de dados é feita seguindo a ordem da tabela de *polling*. A busca para ao encontrar o primeiro periférico com dado disponível ou ao chegar ao fim da tabela.

Retorna: LeStatusEnt retorna o valor lógico VERDADE caso exista dados disponíveis e o valor lógico FALSO caso contrário.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int stat;

        IniciaEnt();
        stat = LeStatusEnt();
        if( stat == VERDADE )
                puts( "Há dado para ser lido." );
}
```

•LeTecla

Função: Lê uma tecla.

Sintaxe:

```
#include "gapres.h"
int LeTecla( int *tecla );
```

Protótipo: *gapres.h*

Comentários: Através do apontador, retorna dois inteiros que representam a tecla digitada e o estado atual das teclas *shift, alt, caps lock, insert, num lock* e *scrl lock.*

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:

```
#include <stdio.h>
```

```
#include "gapres.h"

main()
{
        int tecla[2];

        IniciaEnt();                    /* inicializa os periféricos de entrada */
        while( !LeTecla( tecla ) );
        printf( "Tecla Lida: %x %x", tecla[0], tecla[1] );
}
```

•LeLocator

Função: Lê um par de coordenadas.

Sintaxe: *#include "gapres.h"*

*void LeLocator(int *x, int *y);*

Protótipo: *gapres.h*

Comentários: Retorna via dois apontadores para inteiros uma posição em coordenadas do dispositivo.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int x, y;

        IniciaGA( CGA, CGAHI );
        LeLocator( &x, &y );
        gprintf( "X = %d, Y = %d\n", x, y );
```

```
        TerminaGA();
}
```

•LeStroke

Função: Lê uma sequência de coordenadas.

Sintaxe: *#include "gapres.h"*

void LeStroke(int x[], int y[], n);

Protótipo: *gapres.h*

Comentários: Retorna via dois vetores uma sequência de n pontos em coordenadas do dispositivo.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int x[10], y[10], i;

        IniciaGA( CGA, CGAHI );
        LeStroke( x, y, 10 );
        for( i=0; i<10; i++ )
                printf( "X(%d)=%d, Y(%d)=%d\n", i, x, i, y );
        TerminaGA();
}
```

•LeValuator

Função: Lê um valor real.

Sintaxe: *#include "gapres.h"*

double LeValuator(void);

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Retorna um número de dupla precisão.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        double d;

        IniciaGA( CGA, CGAHI );
        d = LeValuator();
        gprintf( "Número real lido: %lf\n", d );
        TerminaGA();
}
```

•LeChoice

Função: Lê uma opção.

Sintaxe: *#include "gapres.h"*

unsigned int LeChoice(void);

Protótipo: *gapres.h*

Comentários: Retorna um inteiro não negativo o qual representa uma opção entre várias alternativas.

Retorna: Em caso de sucesso, retorna o valor da opção escolhida; em caso de erro retorna 0.

Exemplo:
```
#include <stdio.h>
#include "gapres.h"

main()
{
        unsigned int u;

        IniciaEnt();
        u = LeChoice();
        printf( "Opção Escolhida: %u\n", u );
}
```

LeString

Função: Lê uma cadeia de caracteres

Sintaxe:
```
#include "gapres.h"
char *LeString( char *str );
```

Protótipo: *gapres.h*

Comentários: LeString lê uma cadeia de caracteres terminada por um *return* e a coloca no local apontado por str. O *return* é substituído por um caractere nulo (\0) em str.

Retorna: Em caso de sucesso, retorna o próprio argumento da função; retorna NULL em caso de erro.

Exemplo:
```
#include <stdio.h>
#include "gapres.h"

main()
{
        char str[80];

        IniciaGA( CGA, CGAHI );
```

```
        LeString( str );
        gprintf( "String Lida: %s\n", str );
        TerminaGA();
}
```

•CursorOn

Função: Habilita o display do cursor na tela.

Sintaxe: *#include "gapres.h"*

void CursorOn(void);

Protótipo: *gapres.h*

Comentários: CursorOn ativa o interceptação da interrupção de relógio para realizar o *blinking* do cursor.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        IniciaGA( CGA, CGAHI );
        CursorOn();
        gprintf( "O cursor está piscando em 100,100\n" );
        PosicionaCursor( 100, 100 );
        sleep( 5 );
        CursorOff();
        gprintf( "O cursor foi desabilitado\n" );
        TerminaGA();
}
```

•CursorOff

Função: Desabilita o display do cursor na tela.

Sintaxe: *#include "gapres.h"*

void CursorOff(void);

Protótipo: *gapres.h*

Comentários: CursorOff desativa o interceptação da interrupção de relógio para realizar o *blinking* do cursor. Esta função deve ser usada antes de qualquer alteração na área da tela que contem o cursor. Isto evita problemas tais como o cursor interferindo nos dados de tela.

Retorna: Nada.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        IniciaGA( CGA, CGAHI );
        CursorOn();
        gprintf( "O cursor está piscando em 100,100\n" );
        PosicionaCursor( 100, 100 );
        sleep( 5 );
        CursorOff();
        gprintf( "O cursor foi desabilitado\n" );
        TerminaGA();
}
```

•LePosXCursor

Função: Retorna a coordenada x atual do cursor.

Sintáxe: *#include "gapres.h"*

int LePosXCursor(void);

Protótipo: *gapres.h*

Comentários: LePosXCursor encontra a atual coordenada gráfica x do cursor. O valor é relativo a janela ativa.

Retorna: Retorna a coordenada x atual do cursor.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int x, y;

        IniciaGA( CGA, CGAHI );
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        TerminaGA();
}
```

•LePosYCursor

Função: Retorna a coordenada y atual do cursor.

Sintaxe:

```
#include "gapres.h"
int LePosYCursor( void );
```

Protótipo: *gapres.h*

Comentários: *LePosYCursor* encontra a atual coordenada gráfica y do cursor. O valor é relativo a janela ativa.

Retorna: Retorna a coordenada y atual do cursor.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"
```

```
main()
{
        int x, y;

        IniciaGA( CGA, CGAHI );
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        TerminaGA();
}
```

PosicionaCursor

Função: Posiciona o cursor na janela atual.

Sintaxe: *#include "gapres.h"*

int PosicionaCursor(int x, int y);

Protótipo: *gapres.h*

Comentários: *PosicionaCursor* move o cursor para as coordenadas (x,y) na janela atual. Os valores especificados devem estar dentro da área definida como permitida para o cursor. Caso contrário, os valores serão convertidos para os mais próximos dentro desta área. Neste caso um valor lógico FALSO é retornado.

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int x, y;
```

```
        IniciaGA( CGA, CGAHI );
        PosicionaCursor( 50, 10 );
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        TerminaGA();
}
```

•DefineFaixaH

Função: Define a faixa horizontal onde é permitido o cursor movimentar-se.

Sintaxe: *#include "gapres.h"*

int DefineFaixaH(int xl, int xr);

Protótipo: *gapres.h*

Comentários: *DefineFaixaH* limita o movimento do cursor em determinada faixa horizontal. Os valores especificados devem estar dentro da área definida como permitida para o cursor. Caso contrário, os valores serão convertidos para os mais próximos dentro desta área. Neste caso um valor lógico FALSO é retornado.

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"

main()
{
        int x, y;

        IniciaGA( CGA, CGAHI );
        DefineFaixaH( 50, 100 );
        PosicionaCursor( 50, 10 );
```

```
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        PosicionaCursor( 150, 10 );
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        TerminaGA();
}
```

•DefineFaixaV

Função: Define a faixa vertical onde é permitido o cursor movimentar-se.

Sintaxe:
```
#include "gapres.h"
int DefineFaixaV( int yt, int yb );
```

Protótipo: *gapres.h*

Comentários: *DefineFaixaV* limita o movimento do cursor em determinada faixa vertical. Os valores especificados devem estar dentro da área definida como permitida para o cursor. Caso contrário, os valores serão convertidos para os mais próximos dentro desta área. Neste caso um valor lógico FALSO é retornado.

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:
```
#include <stdio.h>
#include "gapres.h"

main()
{
        int x, y;

        IniciaGA( CGA, CGAHI );
```

```
        DefineFaixaV( 50, 100 );
        PosicionaCursor( 50, 10 );
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        PosicionaCursor( 150, 70 );
        x = LePosXCursor();
        y = LePosYCursor();
        gprintf( "O cursor está em %d,%d\n", x, y );
        TerminaGA();
}
```

•EstiloCursor

Função: Define o estilo do cursor em termos de cor e forma.

Sintaxe: *#include "defglob.h"*

int EstiloCursor(int forma, int cor);

Protótipo: *gapres.h*

Comentários: Colocar os códigos de cores e forma do cursor.

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:

```
#include "gapres.h"

main()
{
        IniciaGA( CGA, CGAHI );
        EstiloCursor( NORMAL, BRANCO );
        PosicionaCursor( 50, 10 );
        while( !LeStatusEnt );
        TerminaGA();
```

```
}
```

•LeSensivel

Função: Lê a informação sobre os incrementos horizontais e verticais causados pelo deslocamento do periférico.

Sintaxe: *#include "gapres.h"*

*void LeSensivel(int *dx, int *dy);*

Protótipo: *gapres.h*

Comentários: Este deslocamento pode ser físico, por exemplo o deslocamento de um mouse. Ou pode ser lógico, por exemplo o uso das setas do teclado.

Retorna: Nada.

Exemplo:

```
#include "gapres.h"

main()
{
        int dx, dy;

        IniciaEnt();
        LeSensivel( &dx, &dy );
        printf( "Sensibilidade Horizontal: %d\n", dx );
        printf( "Sensibilidade Vertical  : %d\n", dy );
}
```

•DefineSensivel

Função: Define os novos valores para incrementos horizontais e verticais causados pelo deslocamento do dispositivo periférico de entrada.

Sintaxe: *#include "gapres.h"*

int DefineSensivel(int dx, int dy);

Protótipo: *gapres.h*

Comentários: Este deslocamento pode ser físico, por exemplo o deslocamento de um mouse. Ou pode ser lógico, por exemplo o uso das setas do teclado.

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:

```
#include "gapres.h"

main()
{
        int dx, dy;

        IniciaEnt();
        DefineSensivel( 4, 2 );
        LeSensivel( &dx, &dy );
        printf( "Sensibilidade Horizontal: %d\n", dx );
        printf( "Sensibilidade Vertical  : %d\n", dy );
}
```

•DesCursorFaixa

Função: Define uma área de exclusão do display do cursor.

Sintaxe:

```
#include "gapres.h"
int DesCursorFaixa( int xl, int yt, int xr, int yb );
```

Protótipo: *gapres.h*

Comentários: O cursor desaparece ao entrar na área definida.

Retorna: Em caso de sucesso, retorna o valor lógico VERDADE; retorna FALSO em caso de erro.

Exemplo:

```
#include <stdio.h>
#include "gapres.h"
```

```
main()
{
        int x, y;

        IniciaGA( CGA, CGAHI );
        DesCursorFaixa( 10, 10, 100, 100 );
        PosicionaCursor( 50, 50 );
        gprintf( "O cursor não é visivel aqui\n" );
        sleep( 5 );
        PosicionaCursor( 150, 70 );
        gprintf( "O cursor é visivel aqui\n" );
        TerminaGA();
}
```

4.3. Dispositivos Periféricos de Saída

Os dispositivos periféricos de saída têm um tratamento análogo àquele dos dispositivos de entrada. Existe, portanto uma Matriz de Funções de Periféricos de Saída, onde cada linha *i* representa um dispositivo periférico de saída e cada coluna *j* uma função.

Já que a aplicação deve informar para onde deve ir a saída dos dados, transparência nos dispositivos de saída implica em dispor dos mesmos comandos para, por exemplo, imprimir uma palavra em negrito, independente do dispositivo de saída utilizado.

A Matriz de Funções de Periféricos de Saída tem a seguinte sintaxe:

```
int (*MatFuncSai[MAX_PERI_SAI][MAX_FUNC_SAI])() = {
        { FSai_0_peri_0, FSai_1_peri_0, FSai_2_peri_0, ... },
        { FSai_0_peri_1, FSai_1_peri_1, FSai_2_peri_1, ... },
```

```
};
```

onde, MAX_PERI_SAI é o número máximo de periféricos disponíveis,
MAX_FUNC_SAI é o número máximo de funções.

A Figura 4.6, abaixo, ilustra como a Matriz de Funções de Periféricos de Saída implementada está preenchida. Esta é uma visão parcial, onde apenas três funções estão representadas.

	Função 0 IniciaSai	Função 1 LeStatusSai	Função 2 SaiCaractere	...
IMP1	inic_imp1	status_imp1	sai_carac_imp1	...
IMP2	inic_imp2	status_imp2	sai_carac_imp2	...
:	:	:	:	
Periférico N	inic_N	status_N	sai_carac_N	...

Figura 4.6 - Visão Parcial da Matriz de Funções de Periféricos de Saída

4.3.1. Sintaxe das Principais Funções Padrão de Saída

Estas funções, em princípio não devem ser chamadas pelo meio exterior uma vez que desta forma a transparência dos dispositivos periféricos deixa de existir. Para tanto, o usuário deve usar as funções descritas no ítem 4.3.2.

Nos exemplos da sintaxe mostrados abaixo, a impressora IMP1 será usada como periférico *default*, no entanto esta sintaxe permanece válida para os demais dispositivos: impressora IMP2. Esta sintaxe se baseia na Linguagem C.

•Função Padrão 0

Função: Inicializa o periférico de saída.

Sintaxe: *#include "gapres.h"*

*int (*MatFuncSai[MAX_PERI_SAI][MAX_FUNC_SAI])();*

int TabSaida[MAX_PERI_SAI];

Protótipo: *gapres.h*

Comentários: O programa que chama a Função Padrão 0 deve utilizar seu valor de retorna para preencher a tabela TabSaida.

Retorna: Em caso de sucesso na inicialização retorna o valor lógico VERDADE; em caso de erro retorna FALSO.

Exemplo:

```
#include "gapres.h"

main()
{
        int peri;

        for( peri = 0; peri < MAX_PERI_SAI; peri++ )
                TabSaida[peri] = (*MatFuncSai[peri][S_INICIALIZA])();
        if( TabSaida[peri] == VERDADE )
                printf( "Periférico %d OK\n", peri );
}
```

•Função Padrão 1

Função: Informa *status* do periférico

Sintaxe: *#include "gapres.h"*

*int (*MatFuncSai[MAX_PERI_SAI][MAX_FUNC_SAI])();*

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Retorna VERDADE, indicando que o periférico esta disponível para receber dados; caso contrário retorna FALSO, indicando que esta *busy*.

Exemplo:

```
#include "gapres.h"

main()
{
        int stat;

        (*MatFuncSai[IMP1][S_INICIALIZA])();
        stat = (*MatFuncSai[IMP1][S_STATUS])();
        if( stat == VERDADE )
                puts( "O periférico está pronto." );
}
```

4.3.2. Interface com Outros Módulos

•IniciaSai

Função: Inicializa periféricos de saída.

Sintaxe:
```
#include "gapres.h"
int IniciaSai( void );
```

Protótipo: *gapres.h*

Comentários: Inicializa todos os periféricos de saida. Os periféricos que obtiverem sucesso na sua inicialização tem o valor lógico VERDADE atribuido a sua posição na tabela de saída.

Retorna: Em caso de sucesso na inicialização de todos os periféricos, retorna o valor lógico VERDADE; retorna FALSO em caso de erro na inicialização de algum periférico.

Exemplo:
```
#include <stdio.h>
```

```
#include "gapres.h"

main()
{
        int np;

        np = IniciaSai();
        printf( "Nº Periféricos Inicializados: %d\n", np );
}
```

•LeStatusSai

Função: Lê o *status* do dispositivo indicado.

Sintaxe:
```
#include "gapres.h"
int LeStatusSai( int periferico );
```

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Retorna VERDADE, indicando que o periférico esta disponível para receber dados; caso contrário retorna FALSO, indicando que esta *busy*.

Exemplo:
```
#include <stdio.h>
#include "gapres.h"

main()
{
        int stat;

        IniciaSai();
        stat = LeStatusSai( IMP1 );
        if( stat == VERDADE )
                puts( "A impressora IMP1 está livre." );
```

```
}
```

•SaiCaractere

Função: Envia um caractere para o dispositivo indicado.

Sintaxe: *#include "gapres.h"*

int SaiCaractere(int periferico, char c);

Protótipo: *gapres.h*

Comentários: SaiCaractere envia um caractere para o dispositivo indicado e usa o atributo atual.

Retorna: Em caso de sucesso na retorna o valor lógico VERDADE; retorna FALSO em caso contrário.

Exemplo:

```
#include "gapres.h"

main()
{
        IniciaSai();
        SaiCaractere(IMP1, 'A');
}
```

•SaiString

Função: Envia uma cadeia de caracteres para o dispositivo indicado.

Sintaxe: *#include "gapres.h"*

*int SaiString(int periferico, char *s);*

. Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Em caso de sucesso na retorna o valor lógico VERDADE; retorna FALSO em caso contrário.

Exemplo: *#include "gapres.h"*

```
main()
{
        IniciaSai();
        SaiString( IMP1, "Gerenciador de Periféricos" );
        SaiCaractere( IMP1, '\n' );
}
```

•SaiCaractereAtrib

Função: Envia um caractere com o atributo especificado para o dispositivo indicado.

Sintaxe: *#include "gapres.h"*

int SaiCaractereAtrib(int peri, char c, int atrib);

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Em caso de sucesso na retorna o valor lógico VERDADE; retorna FALSO em caso contrário.

Exemplo:

```
#include "gapres.h"

main()
{
        IniciaSai();
        SaiCaractereAtrib( IMP1, 'A', NEGRITO );
}
```

•SaiStringAtrib

Função: Envia uma cadeia de caracteres com um atributo especificado para o dispositivo indicado.

Sintaxe: *#include "gapres.h"*

int SaiStringAtrib(int peri, char c, int atrib);

Protótipo: *gapres.h*

Comentários: Nenhum.

Retorna: Em caso de sucesso na retorna o valor lógico VERDADE; retorna FALSO em caso contrário.

Exemplo:

```
#include "gapres.h"

main()
{
        IniciaSai();
        SaiStringAtrib( IMP1, "AGILE", ITALICO );
        SaiCaractere( IMP1, '\n' );
}
```

•DefineAtributo

Função: Define o atributo a ser usado pelo dispositivo de saída.

Sintaxe: *#include "gapres.h"*

int DefineAtributo(int periferico, int atributo);

Protótipo: *gapres.h*

Comentários: A variável *atributo,* é um número que mapeia uma função que realiza a implementação do atributo.

Retorna: Em caso de sucesso na retorna o valor lógico VERDADE; retorna FALSO em caso contrário.

Exemplo:

```
#include "gapres.h"

main()
{
```

```
    IniciaSai();
    DefineAtributo( IMP1, CAPSLOCK );
    SaiString( IMP1, "Tudo maiusculo" );
}
```

Mapeamento de teclas / Reprogramação do Teclado

Quando usamos um computador de forma interativa existe a necessidade de especificar algumas teclas para edição e "navegação". As teclas definidas pelo Módulo Gerenciador de Periféricos para esta finalidade podem eventualmente tornarem-se conflitantes com as teclas usadas pelo programa aplicativo, ou mesmo fugirem a um padrão existente ou estipulado pelo usuário do programa aplicativo.

Pensando nesta possibilidade, pensou-se em criar uma tabela de mapeamento entre a função de edição e/ou navegação e o código da tecla que a habilita. Isto permite ao projetista redefinir as teclas de edição e navegação da forma que achar mais conveniente (vide Figura 4.7). Esta técnica não foi implementada até o presente estágio do trabalho, mas fica proposta como sugestão para versões futuras.

Função	Código	Símbolo
Deslocar o cursor uma posição a esquerda	4B00H	←
Deslocar o cursor uma posição a direita	4D00H	→
Descer uma linha	5000H	↓
Subir uma linha	4800H	↑
Ajuda *On-Line*	3B00H	F1
:	:	:
Voltar ao elemento de origem	011BH	ESC

Figura 4.7 - Exemplo de uma possivel técnica para permitir a redefinição das teclas de edição/navegação.

CONCLUSÕES

Detalhes de Implementação

A linguagem escolhida para implementar o Gerenciador de Apresentação foi a Linguagem C, mas especificamente foi empregado o compilador Turbo C. O código fonte contém aproximadamente 5000 linhas.

Foram utilizadas também, algumas funções da biblioteca gráfica do Turbo C. Em caso de migração para outro ambiente, diferente do MS-DOS, estas funções devem ser repostas por funções equivalentes.

Dificuldades Encontradas

Um dos problemas encontrados durante o desenvolvimento foi a inexistência de um Módulo de Gerenciamento de Dados. Este módulo deveria ter sido implementado de em paralelo ao desenvolvimento do Gerenciador de Apresentação por um outro grupo de trabalho. Como tal fato não ocorreu, houve a necessidade de serem implementados dentro do Gerenciador de Apresentação algumas rotinas de manipulação de dados, as quais, satisfizeram às necessidades mais imediatas. No entanto, o modelo da interface que é hoje totalmente carregado para a memória, poderia ser carregado de acordo com a necessidade liberando uma maior quantidade de memória para a aplicação.

Outro ponto que trouxe atrasos significativos foi a ausência por algum tempo de infra-estrutura de hardware que permitisse os testes das várias especificidades do Gerenciador de Apresentação.

O fato de estar desenvolvendo um trabalho que é parte de um todo, isto é, implementando um Gerenciador de Apresentação que é parte de um grupo que está implementado um Sistema de Prototipagem Rápida implicou em alguns cuidados e atenções com o uso e definição de variáveis, principalmente as globais.

Contudo o maior problema ocorreu quando começaram os testes com o sistema rodando em placas VGA em seus modos de maior resolução e número de cores. Nestes módulos, a quantidade de memória necessária para algumas operações inviabilizava o uso do Gerenciador de Apresentação em máquinas menores, tipo XT, com este tipo de padrão gráfico.

Sugestões para Trabalhos Futuros

Uma das direções previstas para o Sistema de Prototipagem Rápida - AGILE para o qual este Gerenciador de Apresentação foi desenvolvido, é sua transformação em um Sistema de Gerenciamento de Interfaces. Neste caso, as aplicações não estarão sendo apenas simuladas mas de fato *rodando*. Quando isto ocorrer, o nosso Gerenciador de Apresentação ainda será capaz de prover as funções necessárias para gerenciar a entrada e saída.

Com base no cronograma não foi possível implementar algumas funções previstas inicialmente que, se de um lado não são imprescindíveis para se alcançar os principais objetivos do trabalho, devem ser implementadas para que possam ser construídas interfaces com melhores recursos.

Dentre estas funções destacamos as que se seguem:

- A necessidade de dotar os menus e formulários com a possibilidade de *scroll*.
- Menu horizontal.
- Otimizar algoritmos em função de velocidade.

- Introduzir o vídeo na Matriz de Periféricos de Saída.
- Editor Gráfico (os *links* para o mesmo já se encontram no Módulo Gerenciador de Telas).

Uma outra direção a ser seguida é o aproveitamento do *Windows 3.0* como substituto do atual Gerenciador de Telas.

Considerações Finais

Dentre os objetivos propostos para o trabalho, podemos afirmar que a transparência dos dispositivos, bem como a facilidade de acréscimos de novos dispositivos, foram atingidos. Por outro lado, a portabilidade almejada ainda é dependente de algumas funções presentes na biblioteca do compilador C da Borland. Para atingir a portabilidade "total" é necessário reescreve-las.

No que diz respeito a modularidade, o estágio atingido permite que os módulos possam ser repostos por novos módulos, com algorítmos e idéias mais novas, recentes e otimizadas. Por exemplo, seria possível manter a atual API do Gerenciador de Apresentação, e substituir o Módulo Gerenciador de Telas e uma parte do Módulo Gerenciador de Periféricos pelo Microsoft Windows 3.0. Isto é conseguido, a partir de funções de interfaceamento que estão claramente definidas para cada um dos módulos.

Finalmente, uma das deficiências da implementação é a atual pobreza na apresentação gráfica. Nesse sentido, foram deixados espaços para inclusão de um editor gráfico, que permite a criação e posterior uso de primitivas gráficas, oferecendo mais recursos para as interfaces.

Portanto, existe a consciência das limitações deste trabalho com relação ao seu uso pelo Sistema AGILE. No entanto, em outras aplicações, sua contribuição é bastante satisfatória, na medida em que permite a programadores não familiarizados com ambientes gráficos (ou sem tempo para estudarem e serem capazes de utilizar ambientes do tipo *Windows*) uma forma simples de se iniciarem em interfaces gráficas. Além do que, passam a dispor de um número maior de recursos para manipular dispositivos de entrada e de saída, de uma forma transparente e uniforme.

BIBLIOGRAFIA

[ANSON 79] Anson, Ed, *The Semantics of Graphical Input*, ACM SIGGRAPH '79, Agosto, 1979, Vol.13, Num.2, pp. 113-120.

[BRANCO 89] Branco, Simone de O., *Especificação de um Sistema de Prototipagem Rápida de Interfaces Usuário-Computador*, Dissertação de Mestrado, DSC/UFPB, 1990.

[BUXTON 85] Buxton, W., Hill, R. & Rowley, P., *Issues and Techniques in Touch-Sensitive Table Input*, ACM Computers Graphics, San Francisco, Julho, 1985, Vol.19, Num.3, pp. 215-224.

[CARDELLI 85] Cardelli, Luca & Pike, Rob, *Squeak: A Language for Communicating with Mice*, ACM Computers Graphics, San Francisco, Julho, 1985, Vol.19, Num.3, pp. 199-204.

[DeSOI 89] DeSoi, J. F., Lively, W. M. & Sheppard, S. U., *Graphical Specification of User Interfaces with Behavior Abstraction*, CHI'89 Proceedings, Texas A&M University, TX, Maio, 1989, pp. 139-144.

[FLORENCE 90] Florence, R., *UNIX 'termcap' Facility Improves Portability By Hiding Terminal Dependencies*, The C Users Journal, Janeiro, 1990, Vol.8, Num.1, pp. 93-103.

[GAINES 87] Gaines, B. R. & Shaw, M. L. G., *A Interação Computador-Usuário*, LITEC, 1987.

[GREEN 85] Green, M., *The University of Alberta User Interface Management System Development and Application*, ACM SIGGRAPH '85, Julho, 1985, Vol.19, Num.3, pp. 205-213.

[GUARNA 90] Guarna, V. & Krause, J., *User Interface Language Eases Prototyping*, The C Users Journal, Fevereiro, 1990, Vol.8, Num.2, pp. 17-32.

[HAYES 89] Hayes, F. & Baran, N., *A Guide to GUIs*, BYTE, Julho, 1989, Vol.14, Num.7, pp. 250-257.

[JAMSA 88] Jamsa, K., *Windows - Guia do Usuário*, McGraw Hill, 1988.

[JOHNSON 91] Johnson, Eric F. & Reichard, Kevin, *X Window Programming - Part 1: The X Window System*, The C Users Journal, Março, 1991, Vol.9, Num.3, pp. 87-91.

[JOHNSON 91] Johnson, Eric F. & Reichard, Kevin, *X Window Programming - Part 2: The X library*, The C Users Journal, Maio, 1991, Vol.9, Num.5, pp. 30-46.

[JOHNSON 91] Johnson, Eric F. & Reichard, Kevin, *X Window Programming - Part 6: motif programming*, The C Users Journal, Dezembro, 1991, Vol.9, Num.12, pp. 91-99.

[KNOBLAUGH 90] Knoblaugh, R., *Using 'Screen Machine'*, The C Users Journal, Fevereiro, 1990, Vol.8, Num.2, pp. 81-89.

[LEWIS 89] Lewis, T. G., Handloser III F., Bose S., and Yang S., *Prototipes from Standard User Interface Management Systems*, IEEE Computer, Maio, 1989, Vol.32, Num.5, pp. 51-56.

[LÖWGREN 88] Löwgren, J., *History, State and Future of User Interface Management Systems*, ACM SIGCHI Bulletin, Julho, 1988, Vol.20, Num.1, pp. 32-44.

[LUQI 89] Luqi, *Software Evolution Through Rapid Prototyping*, IEEE Computer, Maio, 1989, Vol.22, Num.5, pp. 13-25.

[MARTENSEN 90] Martensen, B., *Prototyping Experiences*, The C Users Journal, Fevereiro, 1990, Vol.8, Num.2, pp. 91-95.

[MYERS 88] Myers, Brad A., *Windows Interfaces - A Taxonomy of Window Manager User Interfaces*, IEEE Computer Graphics & Applications, Setembro, 1988, pp. 65-84.

[MYERS 89] Myers, Brad A., *Encapsulating Interactive Behaviors*, CHI'89 Proceedings, Pittsburgh, PA, Maio, 1989, pp. 319-324.

[NEWMAN 79] Newman, W. M. & Sproull, R. F., *Principles of Interactive Computer Graphics*, McGraw Hill, 1979.

[OSORIO 88] Osório, Fernando S., Pereira, Carlos E. & Barone, Dante A., *Um Gerenciador de Interfaces com Usuário para Editores Gráficos Genéricos*, 1988.

[PERRY 89] Perry, T. S., *Of mice and menus: designing the user-friendly interface*, IEEE Spectrum, September, 1989, Vol. 26, Num.9, pp. 46-51.

[PIKE 83] Pike, Rob, *Graphics in Overlapping Bitmap Layers*, ACM Transactions on Graphics, Abril, 1983, Vol.2, Num.2, pp. 135-160.

[PROCÓPIO 92] Procópio, Claúdia Dantas, *Desenvolvimento do Gerenciador de Diálogo e de Ferramentas de Prototipagem do Sistema AGILE*, Dissertação de Mestrado, DSC/UFPB, 1992.

[RASKIN 89] Raskin, R., *The Packages Behind The Presentation*, PC Magazine, Outubro, 1989, Vol.8, Num.17, pp. 95-142.

[ROGERS 85] Rogers, David F., *Procedural Elements for Computer Graphics*, McGraw Hill, 1985.

[SCHNEIDERMAN 87] Schneiderman, Ben, *Designing User Interface: Strategies for Effective Huma-Computer Interaction*, Addison-Wesley, 1987.

[THOMAS 83] Thomas, James J. & Hamlin, Griffith (Chairman), *Graphical Input Interaction Technique (GIIT): WorkShop Summary*, Computer Graphics, Janeiro, 1983, Vol.17, Num.1, pp. 5-30.

[TORI 87] Tori, R., Arakaki, R., Massola, A. M. A. & Filgueiras, L. V. L., *Fundamentos de Computação Gráfica*, LITEC, 1987.

[USL 92] UNIX System Laboratories, *OPEN LOOK GUI Programmer's Guide*, 1992.

[VAN DEN BOS 78] Van den Bos, J., *Definition and use of higher-level graphics input tools*, Proc. SIGGRAPH '78 (Computer Graphics), Agosto, 1978, Vol.12, Num.3, pp. 38-42.

[WILLIAMS 92] Williams, Al, *A Console Stream Class For Borland C++*, The C Users Journal, Janeiro, 1992, Vol.10, Num.1, pp. 125-132.

[ZANDEN 89] Zanden, Bradley T. V., *Constraint Grammars - A New Model for Specifying Graphical Applications*, CHI'89 Proceedings, Pittsburgh, PA, Maio, 1989, pp. 325-330.